

VU

Virtual University

CS604 Operating System Update MCQS For Quiz-3 File Solve By Vu Topper RM



80 To 100% Marks



وَتَعَزُّ مِنْ تَشَاءٍ وَتُدَلُّ مِنْ تَشَاءٍ

PROFESSIONAL ONLINE ACADEMY



WE Offers

LMS Handling

Important Notes

Online Classes

Assignments

Quiz & GDB's

Projects

**NOTHING
IS
IMPOSSIBLE**

**Join Us
Now**

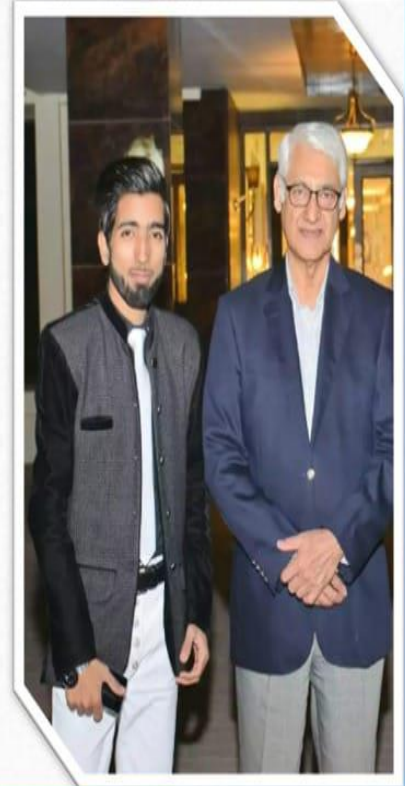
For More Info

Contact us at:

Rizwan Manzoor



0322-4021365



بري صحبت سے تہائی بہتر ہے اور تہائی سے نیک صحبت بہتر ہے

For More Help Contact What's app 03224021365

Possible side effects of _____deadlocks are low device utilization and reduced system throughput.

- A. Ignoring
- B. Avoiding
- C. Recovering
- D. Preventing**

Which of the following is correct definition for wait operation?

- A. wait(S) { while(S<=0);// no o S--; Page 108**
- B. wait(S) { S++; }
- C. wait(S) { while(S>=0) ;// no op S--; }
- D. wait(S) { S--; }

The -----scheme is not applicable to a resource allocation system with multiple instances of each resource type.

- A. Wait for graph Page 145**
- B. None of the given options
- C. Resource allocation graph
- D. Both Resource-allocation and wait-for graph

Semaphores are used to synchronize _____ processes.

- A. Tough
- B. Parallel
- C. Running
- D. Concurrent**

Following is not the classical problem of synchronization.

- A. Bounded buffer problem
- B. Reader writer problem
- C. Dining philosophers problem
- D. Counting Semaphore problem Page 118**

بري صحبت سے تھائی بہتر ہے اور تھائی سے نیک صحبت بہتر ہے

For More Help Contact What's app 03224021365

In Resource Allocation Graph, a _____ $P_i \rightarrow R_j$ indicates that process P_i may request resource R_j at some time in the future.

A. Claim edge **Page 138**

- B. Request edge
- C. Allocation edge
- D. Assignment edge

_____ is used in the detection and recovery mechanism to handle deadlocks.

A. Circular Graph

B. Wait-for Graph **Page 144**

- C. Claim Edge Graph
- D. Resource allocation Graph

Typically monitor, a high level synchronization tool is characterized by _____ and _____.

A. Signal, wait

B. Local variables, semaphores

C. Global variable, local variable

D. Local data, programmer defined operators **Page 125**

In order to remove the problem like busy waiting, some high level synchronization constructs are defined. What are they?

Critical regions and Monitors **Page 124**

Critical regions and Semaphores

The integer value of _____ semaphores can not be greater than 1.

A. Mutex

B. Binary **Page 117**

- C. Counting
- D. Bounded buffer

بري صحبت سے تھائی بہتر ہے اور تھائی سے نیک صحبت بہتر ہے

For More Help Contact What's app 03224021365

Banker's algorithm is used for _____

- A. Deadlock removal
- B. Deadlock detection
- C. Deadlock prevention

D. Deadlock avoidance **Page 140**

_____ is an integer variable accessible through wait and signal which are atomic operations.

- A. Signal
- B. Mutex

C. Semaphore **Page 111**

D. Busy waiting

The _____ requires that once a writer is ready, that writer performs its write as soon as possible. In other words, if a writer is waiting to access the object, no new readers may start reading.

- A. first readers-writers problem
- B. third readers-writers problem
- C. fourth readers-writers problem

D. second readers-writers problem **Page 116**

A state is _____ if the system can allocate resources for each process in some order and still avoid a deadlock.

A. Safe **Page 134**

- B. Unsafe
- C. Mutual
- D. Starvation

_____ algorithm is used in Deadlock avoidance.

A. Bakery

B. Banker's **Page 139**

- C. Safe Sequence
- D. Mutual exclusion

بري صحبت سے تھائی بہتر ہے اور تھائی سے نیک صحبت بہتر ہے

For More Help Contact What's app 03224021365

Binary semaphore whose integer value cannot be can be _____ simpler to implement.

- A. >1** **Page 114**
- B. <1
- C. <=1
- D. >=1

Deadlock _____ provides a set of methods for ensuring that at least one of the necessary conditions cannot hold.

- A. Handling
- B. Avoidance
- C. Recovery
- D. Prevention** **Page 132**

The condition in which a set $\{P_0, P_1 \dots P_n\}$ of waiting processes must exist such that P_0 is waiting for a resource that is held by P_1 , P_1 is waiting for a resource that is held by P_2 , and so on, P_{n-1} is waiting for a resource held by P_n , and P_n is waiting for a resource held by P_0 . This condition is known as _____.

- A. Circular wait** **Page 131**
- B. Hold and wait
- C. No preemption
- D. Mutual exclusion

The problem of Deadlocks can be solved by _____ method(s).

- A. All of the given** **Page 132**
- B. Deadlock avoidance
- C. Deadlock prevention
- D. Allowing deadlock to occur, then detect and recover

بري صحبت سے تھائی بہتر ہے اور تھائی سے نیک صحبت بہتر ہے

For More Help Contact What's app 03224021365

The integer value of _____ semaphores can range over an unrestricted integer domain.

- A. Mutex
- B. Binary
- C. Counting**
- D. Bounded buffer

Page 117

If the system can allocate resources to each process in some order and still avoid a deadlock then it is said to be in _____ state.

- A. Safe**
- B. Mutual
- C. Un-Safe
- D. Starvation

Page 134

Wrong use of wait and signal operations (in context with semaphores) can cause _____ problem(s).

- A. Deadlock
- B. Mutual Exclusion
- C. Bounded Waiting
- D. All of the given options are correct**

Deadlock can be deal with _____ ways.

- A. 1
- B. 2
- C. 3**
- D. 4

Page 132

The condition where a set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set, is termed as _____.

Deadlock
Starvation

Page 130

بري صحبت سے تھائی بہتر ہے اور تھائی سے نیک صحبت بہتر ہے

For More Help Contact What's app 03224021365

If a system is not in a safe state, there can be NO deadlocks.

False Page 137

True

Deadlock detection and recovery technique is exactly similar to deadlock avoidance technique to handle deadlock in the system.

True

False

A dashed line is used to represent a _____ in Resource Allocation Graph.

A. Claim edge Page 136

B. Request edge

C. Allocation edge

D. Assignment edge

Removing the possibility of deadlock in dining philosopher problem does not ensure the _____ problem will not occur.

A. Starvation Page 123

B. Critical Section

C. Bounded Buffer

D. Mutual Exclusion

In deadlock detection and recovery algorithm, a deadlock exists in the system if and only if the wait for graph contains a _____.

A. Node

B. Edge

C. Cycle Page 147

D. Graph

The process of holding at least one resource and waiting to acquire additional resources that are currently being held by other processes is known as _____.

بري صحبت سے تھائی بہتر ہے اور تھائی سے نيك صحبت بہتر ہے

For More Help Contact What's app 03224021365

A. Circular wait

B. Hold and wait **Page 131**

C. No preemption

D. Mutual exclusion

Memory protection in paging is achieved by associating _____ with each page. This bit indicates whether the page is in the process address space or not.

Protection bits **Page 169**

_____ consists of a large array of words or bytes, each with its own address.

Memory **Page 150**

Logical memory is divided into blocks of the same size, called.....

Pages **Page 162**

Variable names are ----- addresses.

Relocatable **Google**

_____ is constrained by factors like quantum for RR scheduler and pending I/O for swapped out process.

Swapping **Page 157**

The system maintains a _____ of all processes whose memory images are on the backing store or in memory and are ready to run.

Ready queue **Page 157**

بري صحبت سے تتهائي بهتر ہے اور تتهائي سے نيك صحبت بهتر ہے

For More Help Contact What's app 03224021365



CS604-Operating System

(Solved MCS's)

LECTURE FROM
(23 to 45)



Junaidfazal08@gmail.com
Bc190202640@vu.edu.pk

FOR MORE VISIT
VULMSHELP.COME

JUNAID MALIK
0304-1659294

AL-JUNAID TECH INSTITUTE



www.vulmshelp.com



Language Courses Training Available

I'm providing paid courses in different languages within 3 Months, Certificate will be awarded after completion.

- HTML
- CSS
- JAVASCRIPT
- BOOTSTRAPS
- IQUERY
- PHP MYSQL
- NODES.JS
- REACT JS

LMS Handling Services

LMS Activities Paid Task

Assignments 95% Results

Quizes 95% Results

GDB 95% Results

For CS619 Project Feel Free To Contact With Me

Ph# 0304-1659294
Email: junaidfazal08@gmail.com

AL-JUNAI D INSTITUTE GROUP

- 1.....is the separation of user logical memory from physical memory.
 - a. physical memory
 - b. ROM
 - c. **virtual memory pg#185**
 - d. one of the given

2. When the address used in the program gets converted to an actual physical RAM address is called.....
 - a. Execution
 - b. loading
 - c. compiling
 - d. **Address binding pg#151**

3. The set of all Logical address generated by a program fromof the process.
 - a. process address space
 - a. **Physical address space pg#155**
 - b. logical address space
 - c. none of the given

4. 1 KB is equivalent to.....
 - a. 10 bytes
 - b. 100 bytes
 - c. **1000bytes from Google**
 - d. 1024 bytes

- 5Page replacement algorithm suffers from Belay's anomaly.
 - a. LIFO
 - b. **FIFO pg#199**
 - c. MRU
 - d. LRU

- 6points to the smallest memory address of a process.

AL-JUNAID INSTITUTE GROUP

- a. Limit register
 - b. **Base register pg#10 Base register – it holds the smallest legal physical memory address for a process**
 - c. Stack register
 - d. None of the given
7. In system, the user space contain one process at a time because only one process is executing at given time
- a. Multi programmed
 - b. **Batch pg#5**
 - c. Time-sharing system
 - d. None of the given
- 8.....wastes CPU cycles and hence a problem in real multiprogramming system.
- a. **Busy Waiting pg#111**
 - b. Semaphore
 - c. Critical Section
 - d. Mutex
9. A new process is called by the.....system call
- a. exit
 - b. **fork pg#36**
 - c. wait
 - d. exec
- 10.The address generated by the CPU after any indexing or other addressing-mode arithmetic is called address and the address it gets translated by MMU is called address.
- a. **Virtual, physical pg#153**
 - b. Hexadecimal, Binary
 - c. Valid, Virtual
 - d. Physical, Virtual
- 11.....is single program that produces an object file.
- a. Linker
 - b. **Compiler pg#86**
 - c. loader
 - d. text editor

AL-JUNAID INSTITUTE GROUP

12. In frame allocation scheme free frames are equally divided among processes.

- a. **Fixed pg#205 or Fixed Allocation**
- b. Proportional
- c. Priority
- d. All of the given

13. Named pipes giveflow of data by default.

- a. **half-duplex pg#46**
- b. full-duplex
- c. both half and full duplex
- d. none of the above

14.....Is the variation of fork system call in several Unix operating system used for virtual memory.

- a. **vfork() pg#194**
- b. wfork()
- c. afork()
- d. bfork()

15. In scan algorithm the disk..... starts at one end of the disk, and moves towards other end, servicing requests as it reaches each cylinder, until it gets to the other end of the disk.

- a. **arm pg#245**
- b. cylinder
- c. head
- d. none

16.The problem of Deadlock can be solved bymethod(s).

- a. Deadlock prevention
- b. Deadlock avoidance
- c. Allowing Deadlock
- d. **All of the given Google**

17. In, each file is a linked lists blocks may be scattered anywhere on the disk.

- a. Indexed Allocation
- b. Contiguous Allocation
- c. **Linked Allocation pg#235**
- d. none of the above

AL-JUNAI D INSTITUTE GROUP

- 18.....enables process to communicate with each other.
- Directory
 - FIFO pg#218**
 - Linked File
 - Ordinary File
19. In allocation scheme number of frames allocated to a process is proportional to its size.
- None of the given
 - Fixed
 - Proportional pg#205**
 - Priority
20. If validation bit 0, it indicates a/anstate of segment.
- protected
 - shared
 - legal
 - illegal pg#178**
21. In technology, memory is divided into several fixed-size parties.
- swapping
 - overlays
 - Multiprogramming with Fixed Task(MKT) pg#160**
 - Multiprogramming with Variable Task(MVT)
- 22.....Automatically holds for printer and other non-sharables.
- Hold and Wait
 - Circular Wait
 - Mutual exclusion pg#132**
 - no preemption
23. In page replacement algorithm oldest frame is replaced with another.
- LIFO
 - FIFO pg#119**
 - LRU
 - Optimal
24. When a link is created, a directory entry for the existing file is created.

AL-JUNAID INSTITUTE GROUP

- a. **Hard pg#225**
- b. Soft
- c. soft and hard
- d. hard or soft

25. In case of thrashing if CPU utilization is too low the operating systemthe degree of multiprogramming.

- a. **Increase pg#208**
- b. Decrease
- c. Increase or Decease
- d. None

26. Which one of the following is correct syntax to copy file 1 to file 2?

- a. cp fie2, fie1
- b. **cp file1/file2 Pg#27**
- c. cp file2 file1
- d. cp file1 file2

27 is/are a memory management scheme that supports programmer's view of memory.

- a. paging
- b. **segmentation pg#176**
- c. All of the above
- d. demand paging

28 keeps in memory only those instructions and data that are needed at any given time.

- a. paging
- b. swapping
- c. **overlays pg#156**
- d. fragmentation

29. The bottom layer in the layer in the layered approach of Operating System is.

- a. user interface
- b. **hardware pg#20**
- c. kernel
- d. none

AL-JUNAI D INSTITUTE GROUP

30. The process of switching from one process to another is called -----

- a. **context switching pg#34**
- b. Scheduling
- c. quantum periods
- d. latency

31. Assume the logical address space of 16 pages of 1024 word, each mapped into physical memory of 32 frames. Each word consists of 2 bytes. what will be the total number of bits to require for the P (page number)

- a. **4 bits pg#166**
- b. 8 bits
- c. 16 bits
- d. 32 bits

32. is based on the locality of reference concept at least frequency used page is not on the current locality

Least Frequently Used algorithm pg#204

- b. Page Buffering Algorithm
- c. None
- d. Most Frequently Used algorithm

33. Every generated by the CPU is divided into two parts a page number (p) and a page offset(d)

- a. page
- b. process address
- c. physical
- d. **Logical address pg#164**

34. High-level synchronization constructor that allows the safe sharing of an abstract data type among concurrent process is called.....

- a. Read/Write Lock
- b. Swap
- c. Semaphore
- d. **Monitor pg#122**

35. Preemptive scheduling is sometimes called Shortest-remaining-time-first scheduling

- a. FCFS(First-Come-First-Served)
- b. Round-Rubin
- c. **Shorted Job First Pg#83**

AL-JUNAID INSTITUTE GROUP

- d. Priority
36. Which of the Following is not an Operating System?
- Linux
 - Unix
 - Windows Xp
 - Database pg#7**
37. Operating system enables the user to use the Hardware Resources.
- True pg#4**
 - False
38. Which of the following is NOT a Hardware Resource?
- CPU
 - OS pg#4**
 - I/O Devices
 - Memory
39. Managing Secondary Storage Involves all of the Following except
- Allocating storage space
 - Deal locating Storage
 - Prevent Overwriting (Page 5)**
 - Insure integrity of shared data
40. In Layered approach of OS, Lowest Layer is known as__.
- Software Layer
 - Hardware Layer pg#21**
 - Lower Level Layer
 - None of The Given
41. Mach, Mac OS X Server, QNX, OS/2 and Windows NT are examples of OS Based on
- Layered
 - Micro Kernel pg#22**
 - Virtual Machine
 - None of The Given
42. _____ determines How to do something.
- Mechanism pg#24**
 - Policy
 - Mechanism and Policy:
 - None of the given

AL-JUNAID INSTITUTE GROUP

43. User Goal of OS is that It easy to use, reliable, safe and fast.
- True pg#24**
 - False
44. A pathname is the list of directories separated by_.
- #
 - \$
 - &
 - / pg#25**
45. The Home Directory for super user in Linux and Unix is
- /home
 - /root pg#27**
 - None of the given
46. _____ is a virtual directory in Linux and Unix.
- /proc pg#27**
 - /temp
 - /ver
 - /boot
47. The procedure “The time at which the process finished working MINUS the arrival time of the process MINUS CPUburst for that process” will help calculate the.
- On-preemptive Shortest Job first scheduling.
 - Preemptive Shortest Job First scheduling. Pg# 85**
 - FCFS
 - RR Scheduling
48. The number of processes completed per unit time is called_____.
- Turnaround time
 - Throughput pg#83**
 - Response time
 - Dispatch latency
49. If your processor does not have two slots empty in Per Process File Descriptor Table, then your_ system call will fail.
- Pipe pg#55**
 - read
 - write

AL-JUNAID INSTITUTE GROUP

- d. open
50. Linux uses ___ directory to store system configuration files.
- /bin
 - /dev
 - /boot
 - /etc** pg#26
51. command displays the contents of current working directory.
- ls** pg#28
 - Cs
 - Mv
52. A process consists of _
- One or more threads
 - Code
 - Data
 - All of the given** google
53. _____ is a solution to the problem of indefinite blockage of low-priority processes.
- Starvation
 - Deadlock
 - Aging** pg#87
 - None of the these
54. _____ is the basis of queuing theory which is branch of mathematics used to analyze systems involving queues and servers.
- Little's Formula** pg#96
 - Deterministic modeling
 - Queuing Theory
 - Queuing Analysis
55. POSIX is a standard developed by ANSI
- IEEE
 - ISO**
 - ACM
56. In Unix/ Linux, by default the standard output file is attached to the ___
- File

AL-JUNAID INSTITUTE GROUP

- b. **Screen pg#59**
 - c. Printer
 - d. Scanner
57. The scheduling of are done by the operating system.
- a. **Kernel threads pg#73**
 - b. User level threads
 - c. Both kernel and user level thread
 - d. None of the give option
58. _____ displays information about the top processes.
- a. s
 - b. Cs
 - c. **Top pg#67**
 - d. Cd
59. All threads within a process share the _____ address space.
- a. Same
 - b. **Different pg#71**
60. A major problem with priority scheduling algorithms is_.
- a. Deadlock
 - b. Aging
 - c. **Starvation pg#86**
 - d. None of the these
61. When process opens its first file explicitly it will get descriptor number _____
- a. 1
 - b. 2
 - c. **3 google**
 - d. 4
62. Round Robin algorithm is similar to scheduling but preemption is added to switch between processes.
- a. Shortest job first
 - b. Shortest Remaining Time First
 - c. **First Come First Server pg#88**
 - d. None of these
63. DOS is single user operating system.
- a. **True pg#7**
 - b. False

AL-JUNAI D INSTITUTE GROUP

64. _____ is a piece of code in a cooperating process in which the process may update shared data (variable, file, database, etc.).
- Critical analysis
 - Critical section Pg#100**
 - Critical path
 - Critical code
65. n-process critical section problem can be solved by using
- The bakery algorithm pg#105**
 - Determinizing modeling
 - Analytic evaluation
 - None of above
66. A parent process calling system call will be suspended until children process terminates.
- wait google**
 - fork
 - exit
 - exec
67. OS helps manages the following except
- Application software
 - Bus speed of the system google**
 - Memory
 - Virtual memory
68. The Shortest-Job-First Scheduling algorithm can be
- Preemptive only
 - non-preemptive only
 - Preemptive or non-preemptive. Pg#85**
 - None of the given options
69. First-Come, First-Served (FCFS) is a scheduling algorithm.
- preemptive
 - Non-preemptive pg#83**
 - both preemptive and non-preemptive
 - none of the given options
70. The time it takes for the dispatcher to stop one process and start another running is known as the -----.
- Dispatch latency pg#82**
 - Scheduling

AL-JUNAID INSTITUTE GROUP

- c. Context switching
 - d. None of the given options
71. You can display all of the signals supported by your system, along with their numbers, by using the-----command
- a. <Ctrl-A>
 - b. fg
 - c. jobs
 - d. **kill -l** pg#69
72. A is an abstract key for accessing a file.
- a. **File descriptor** google
 - b. Input Redirection
 - c. Output Redirection
 - d. FIFO
73. The _____ are used for communication between related or unrelated processes on the same system or unrelated processes on different systems.
- a. Pipes
 - b. **BSD Sockets** pg#53
 - c. Named pipe (FIFO)
 - d. None of the given options
74. The creating process is called a ----- process while the new processes are called theof that Process
- a. None of the given options
 - b. Children, parent
 - c. **Parent, children** pg#38
 - d. Zombie, single
75. Files that start with a in UNIX/Linux directory structure are known as hidden files.
- a. **.(dot)** pg#28
 - b. # (hash)
 - c. / (slash)
 - d. ~ (tilt)
76. You can use the command in UNIX to create a directory.
- a. rmdir
 - b. **mkdir** pg#29

AL-JUNAID INSTITUTE GROUP

- c. cp
- d. gcc

77 -----has a hierarchical file system structure.

- a. DOS
- b. Windows
- c. **UNIX** pg#25
- d. None of the given options

78. 1 MB or 1 megabyte is equivalent to----

- a. 1024 bytes
- b. **1024² bytes** Google
- c. 1024³ bytes
- d. 1000000 bytes

79. If the fork system call fails, it returns

- a. 1
- b. **-1** pg#40
- c. 2
- d. 0

80. In ___ communication the process which wants to communicate with the other process must explicitly name the recipient and the sender.

- a. **Direct** pg#46
- b. Indirect
- c. Automatic
- d. Self

81. Bounded Buffer is a buffer of ___ size

- a. variable
- b. **fixed** pg#44

82. A Process that has finished working, as well as its parent process has also finished its execution. In this state the process A will be called as _____ process.

- a. **Child**
- b. Thread
- c. Zombie
- d. Fork

83. Taking the CPU from one process and giving the CPU to another process is termed as

AL-JUNAID INSTITUTE GROUP

a. **Context switching** Google

b. Dispatching

c. Swapping

d. Tracking

84. rm and [r]mkdir commands are used to ___ directory.

a. Create

b. Move

c. **Remove** pg#30

d. Modify

85. Utilities used for system administration (halt, ifconfig, fdisk, etc.) are stored in directory.

a. /dev

b. /boot

c. /lib

d. **/sbin** pg#27

86. In Linux directory structure, there is _____ root directory.

a. **1** pg#26

b. 2

c. 3

d. 4

87. I/O instructions are Privileged Instructions.

a. **True** pg#12

b. False

88. Command-line interpreter is also called _ in some operating systems.

a. Kernel

b. **Shell** Pg#16

c. Signal

d. API

89. The major advantage of multi-programming system is

a. More than one jobs can be processed at a given time

b. **CPU utilization can be increased** Pg#8

c. Jobs can be completed quickly

d. All of the options are correct

90. The Operating system is a layer of software between _____ and _____

a. **Hardware, software application** pg#21

AL-JUNAID INSTITUTE GROUP

- b. Kernel, hardware
 - c. Dos, Windows
 - d. Windows, Kernel
91. Linux OS can support multiple users at a time
- a. **True pg#9**
 - b. False
92. _____ is also called Swapper.
- a. Swap space
 - b. **Medium term scheduler pg#37**
 - c. Short term scheduler
 - d. Long term scheduler
93. The performance of Round Robin algorithm does NOT depends heavily on the size of the time quantum.
- a. **True pg#89**
 - b. False
94. _____ Scheduler selects the process from the job pool and put them in main memory.
- a. **Long term pg#36**
 - b. Short term
 - c. Medium term
 - d. Swapper
95. The main characteristic of a Real time system is
- a. Efficiency
 - b. Large Virtual Memory
 - c. Large secondary storage device
 - d. **Usability google**
96. A time sharing system is
- a. Multitasking
 - b. Interactive
 - c. Multi user
 - d. **All of these pg#9**
97. You can terminate a foreground process by pressing -----
- a. <Ctrl-A>
 - b. **<Ctrl-C> pg#69**
 - c. <Ctrl-Z>
 - d. None of the given options

AL-JUNAID INSTITUTE GROUP

98. You can use the--- command to display the status of suspended and background processes

- a. fg
- b. bg
- c. **jobs pg#68**
- d. kill

99. The manual pages can be read in Lenux Using -----command.

- a. **man pg#27**
- b. wan
- c. desc
- d. help

100. A program in execution is called a_____.

- a. Command
- b. **Process pg#31**
- c. Software
- d. Compiler

101. Current working directory can be accessed using Command.

- a. . (dot)
- b. # (hash)
- c. / (slash)
- d. **~ (tilt) pg#25**

102. Mkfifo() is a_____.

- a. **Library Call pg#58**
- b. Command
- c. Directory
- d. None of Above

103. _____command gives a snapshot of the current processes.

- a. **ps pg#66**
- b. top
- c. who
- d. ls

104. Time interval when the I/O Devices are accessed is known as_____.

- a. CPU Burst

AL-JUNAID INSTITUTE GROUP

- b. **IO Burst google**
c. Time Slice
d. None of Above
105. _____ Directory includes essential system boot files including the kernel image.
a. /bin
b. **/boot pg#26**
c. /dev
d. /etc
106. IPC provides a mechanism to allow processes to communicate and to _____ synchronize their actions without sharing the same
a. **Address space pg#46**
b. Address Name
c. Address ID
d. None of all these
107. Each process must first request permission to enter its critical section. The section of code implementing this request is called the _____
a. **entry section pg#100**
b. Critical Section
c. remainder section
d. None of all these
108. The interval from the time of submission to the time of completion is the _____
a. **Turnaround time pg#83**
b. Waiting time
c. Response time
d. None of all these
109. You can create a threads by using the pthread_create() call.
a. **True pg#64**
b. False
110. Typically the execvp system call is used after a fork system call.

AL-JUNAID INSTITUTE GROUP

- a. **True pg#39**
b. False
111. A solution to the critical section problem must satisfy the following requirements
a. Progress
b. Mutual exclusion
c. Bounded Waiting
d. **All of these pg#101**
112. In _____ addressing, the recipient is not required to name the sender.
a. Symmetric
b. **Asymmetric Pg#47**
c. Both symmetric and asymmetric
d. None of the given options
113. The process id returned to the child process after successful fork system call execution is.
a. **0 Pg#40**
b. 1
c. 2
d. 3
114. _____ command to resume the execution of a suspended job in the foreground
a. **fg pg#68**
b. bg
c. jobs
d. kill
115. A _____ is an integer variable that, apart from initialization is accessible only through two standard atomic operations: wait and signal.
a. **Semaphore pg#111**
b. Monitor
c. Critical region
d. Critical section
116. In Resource Allocation Graph, A _____ $P_i \rightarrow R_j$ indicates that process

AL-JUNAID INSTITUTE GROUP

Pi may request resource Rj at sometime in the future.

- a. **Claim edge pg#138**
 - b. Request edge
 - c. Assignment edge
 - d. Allocation edge
117. In pages segmentation, the logical address is legal if d is _____ segment length.
- a. **< (less than) Pg#180**
 - b. > (greater than)
 - c. = (equal to)
118. The ----- scheme is not applicable to a resource allocation system with multiple instances of each resource type.
- a. **Wait for graph Pg#148**
 - b. Resource allocation graph
 - c. Both Resource-allocation and wait-for graph
 - d. None of the given options
119. A semaphore that cause Busy-Waiting is termed as _____.
- a. **Spinlock Pg#113**
 - b. Monitor
 - c. Critical region
 - d. Critical section
120. The condition in which a set $\{P_0, P_1 \dots P_n\}$ of waiting processes must exist such that P0 is waiting for a resource that is held by P1, P1 is waiting for a resource that is held by P2, and so on, Pn-1 is waiting for a resource held by Pn, and Pn is waiting for a resource held by P0. This condition is known as.
- a. Mutual exclusion
 - b. Hold and wait
 - c. No preemption
 - d. **Circular wait Pg#131**
121. Preventing a condition of _____ to happen, deadlocks can be prevented to happen.

AL-JUNAI D INSTITUTE GROUP

- a. Critical region
 - b. **Circular wait Pg#136**
 - c. Monitors
 - d. Critical section
122. What do we name to an address that is loaded into the memory----- address register of the memory?
- a. Logical address
 - b. **Physical address Pg#155**
 - c. Binary addresses
 - d. None of the given options
123. ----- register contains the size of the process
- a. Base register
 - b. Index register
 - c. **Limit register Pg#13**
 - d. Stack pointers register
124. If the system can allocate resources to each process in some order and still avoid a deadlock then it is said to be in _____ state.
- a. **Safe Pg#137**
 - b. Un-Safe
 - c. Mutual
 - d. Starvation
125. _____ Algorithm is used in Deadlock avoidance.
- a. Bakery
 - b. **Banker's Pg#139**
 - c. Mutual exclusion
 - d. Safe Sequence
126. refers to the situation when free memory space exists to load a process in the memory but the space is not contiguous.
- a. Segmentation
 - b. Internal fragmentation
 - c. Swapping
 - d. **External Fragmentation Pg#165**

AL-JUNAID INSTITUTE GROUP

127. The main criteria of page replacement in optimal page replacement algorithm is to

a. **Replacement that page will not be use for the longest period of time**

Pg#199

b. Replacement that page will be required most frequently in the execution of process

c. Replace the page which is biggest in size

128. When the process tries to access locations that are not in memory, the hard traps the operating system. This is called as _____.

a. **Page fault Pg#188**

b. Page replacement

c. Paging

d. Segmentation

129. The pager is used in connection with _____.

a. **Demand paging Pg#186**

b. Paging

c. Segmentation

d. Page segmentation

130. Segmentation is a memory management scheme that support ____?

a. **Programmer's view of memory Pg#175**

b. System's view of memory

c. Hardware's view of memory

d. None of the given

131. The segment table maps the _____ the physical addresses.

a. Page addresses

b. Shared page addresses

c. One-dimensional logical addresses

d. **Two-dimensional logical addresses Pg#175**

132. _____ Point to the page table.

a. Translation look-aside buffers

AL-JUNAID INSTITUTE GROUP

- b. Page offset
- c. Page-table length registers (PRLR)
- d. **Page-table base registers (PTBR) Pg#166**

133. _____ is used in the detection and recovery mechanism to handle deadlocks.

- a. **Wait-for graph Pg#144**
- b. Resource allocation graph
- c. Circular graph
- d. Claim edge graph

134. Address Binding will be at _____ in multiprogramming with fixed tasks (MFT)

- a. Rub time
- b. **Load time Pg#160**
- c. Dynamic time
- d. None of the

135. What do we name to an address that is generated by CPU?

- a. **Logical address Pg#152**
- b. Physical address
- c. Binary address
- d. None of the above

136. Secondary storage memory devices have _____ memory.

- a. Volatile
- b. **Permanent and nonvolatile google**
- c. Temporary
- d. None of the given

137. The collection of process that is waiting on the disk to be brought into the memory for execution forms the...

- a. **Input queue Pg#154**
- b. Output queue
- c. Both of the
- d. None of the above

AL-JUNAID INSTITUTE GROUP

138. is a segment of code that accesses a shared resource like data structure or device that must not be concurrently accessed by more than one thread of execution.
- Multithreading
 - Context switching
 - Critical section Pg#105**
 - Pipelining
139. Use of semaphore create a problem of busy waiting, this wastes CPU cycles that some other process may be able to use productively. This type of semaphore is also called
- Semaphore S
 - Spinlock Pg#112**
 - Locking
 - Semaphore Mutex
140. Using hardware solution to synchronization for complex problems, introduce a new synchronization tool known as _____.
- Test And Set
 - Semaphore Pg#111**
 - Swap
 - Trap
141. Batch programs are usually _____ programs.
- Interactive
 - Non-interactive Google**
 - Foreground
 - Preemptive
142. _____ Algorithm is used for solving n-process critical section problem.
- Bankers
 - Bakery Pg#105**
 - Babbles
 - None of the given

AL-JUNAID INSTITUTE GROUP

143. The integer value of _____ semaphores can range over an unrestricted integer domain.
- Counting Pg#117**
 - Binary
 - Mutex
 - Bounded buffer
144. _____ is a preemptive scheduling algorithm.
- First Come First Serve
 - Shortest Job First
 - Round Robin Pg#89**
 - None of these
145. The priority of a process can be changed using _____ command.
- nice Pg#94**
 - cmd
 - Cat
 - Grep
146. Removing the possibility of deadlock in dining philosopher problem does not ensure the problem will not occur.
- Mutual Exclusion
 - Starvation Pg#123**
 - Critical Section
 - Bounded Buffer
147. In producer-Consumer problem synchronization is required. On which shared area this synchronization actually effect.
- Counter
 - Buffer google**
 - Entry section
 - Exit section
148. Typically monitor, a high-level synchronization tool is characterized by _____ and...
- Global variable, local

AL-JUNAID INSTITUTE GROUP

- b. variableSignal, wait
 - c. **Local data, programmer defined operators Pg#125**
 - d. Local variables, semaphores
149. Following is not the classical problem of synchronization.
- a. Bounded buffer problem
 - b. Reader writer problem
 - c. Dining philosophers' problem
 - d. **Counting Semaphore problem Pg#118**
150. Consider a scenario in which one process P1 enters in its critical section, no other process is allowed to execute in its critical section. This is called _____
- a. **Mutua exclusion Google**
 - b. Context switching
 - c. Multithreading
 - d. Progress
151. The Swap instruction which is the hardware solution to synchronization problem does not satisfy the condition, hence not considered to be a good solution.
- a. Progress
 - b. **Bounded waiting Pg#109**
 - c. Mutual exclusion
 - d. None of the given
152. The logical address of Intel 80386 is _____
- a. 36 bits
 - b. **48 bits Pg#185**
 - c. 64 bits
 - d. 128 bits
153. Intel 80386 used paged segmentation with ___ level paging.
- a. One
 - b. **Two Pg#185**
 - c. Three
 - d. Four

AL-JUNAID INSTITUTE GROUP

154. In paged segmentation, we divide every segment in a process into _____pages.

- a. **Fixed size Pg#182**
- b. Variable size

155. Which command, Display permissions and some other attributes for prog1.c in your current directory?

- a. **ls -l prog1.c Pg#234**
- b. ls -d prog1.c
- c. ls file prog1.c
- d. ls -l prog1.c /Directory

156. The_____requires that once a writer is ready, that writer performs its write as soon as possible , if a writerwaiting to access the object, no new readers may start reading.

- a. first readers-writers problem
- b. **second readers-writers problem Pg#119**
- c. third readers-writers problem
- d. fourth readers-writers problem

157. Banker's algorithm is used for-----

- a. **Deadlock avoidance Pg#140**
- b. Deadlock detection
- c. Deadlock prevention
- d. Deadlock removal

158. The basic purpose of____is to help the users to utilize the hardware resources for completingdifferent tasks in a simplified manner

- a. **Operating system Pg#6**
- b. Application software
- c. All Software
- d. All of the given

AL-JUNAI D INSTITUTE GROUP

159. _____ is used to store data on secondary storage device, e.g., a source program(in C), an executable program.
- Block Special File
 - Link File
 - Ordinary File Pg#220**
 - Directory
160. indicates size of the page table
- translation look-aside buffers
 - Page-table length register (PTLR) Pg#169**
 - Page-table base register (PTBR)
 - Page offset
161. The set of all physical addresses corresponding to the logical addresses is a of the process
- Physical address space Pg#155**
 - Process address space
 - None of the given options
 - Logical address space
162. Deadlocks can be described more precisely in terms of a directed graph called a system -----
- Directed graph
 - Critical path
 - Resource allocation graph Google**
 - Mixed graph
163. With----- you use condition variables.
- Semaphores
 - Read/Write Locks
 - Swaps
 - Monitor Pg#126**
164. buffer places no practical limit on the size of the buffer
- Bounded

AL-JUNAID INSTITUTE GROUP

- b. **Unbounded Pg#44)**
- c. Both Unbounded & bounded
- d. None of the given options

165. spend more time doing IO than computations

- a. short CPU bursts
- b. CPU bound processes
- c. **IO bound processes Pg#32**
- d. None of the given options

166. You can display the contents (names of files and directories) of a directory in UNIX/Linux directory structure with the _____ command.

- a. l
- b. s
- c. **ls Pg#28)**
- d. none of the given options

167. The critical section problem can be solved by the following except

- a. Software based solution
- b. **Firmware based solution Pg#101**
- c. Operating system based solution
- d. Hardware based solution

168. A modification of free-list approach in free space management is to store the addresses of n free blocks in the first free block. Known as _____.

- a. counting
- b. linked list
- c. bit vector
- d. **grouping Pg#243**

169. In deadlock detection and recovery algorithm, a deadlock exists in the system if and only if the wait for graph contains a _____.

- a. **Cycle Pg#147**
- b. Graph

AL-JUNAID INSTITUTE GROUP

- c. Edge
- d. Node

170. In _____ page replace algorithm we will replace the page that has not been used for the longest period of time.

- a. counter based
- b. Least Frequently Used
- c. FIFO

d. LRU Pg#202

171. **Overlays are implemented by the _____**

- a. Operating system

b. Programmer pg#159

- c. Kernel
- d. Shell

172. If a process continues to fault, replacing pages, for which it then faults and brings back in right away. This high paging activity is called.

- a. paging
- b. thrashing Pg#210**
- c. page fault
- d. CPU utilization

173. The _____ method requires each file to occupy a set of contiguous blocks on the disk.

- a. Contiguous Allocation pg#236**
- b. Linked Allocation
- c. Indexed Allocation
- d. None of the given options

174. a way to establish a connection between the file to be shared and the directory entries of the users who want to have access to this file.

a. Link pg#231

- b. Directory
- c. Common Group

AL-JUNAID INSTITUTE GROUP

d. Access Permission

175. _____ is used to request the OS by the process to take an I/O or initiating child process.
- System call
 - Interrupt
 - Trap
 - Signal
176. You can use the mv file1 file2 command to move _____
- file1 to file2.
 - file 2 to file 1
 - this command will not work for moving files
 - None of the option is correct.
177. _____ commands in Linux is used to copy file
- Is
 - cp
 - mv
 - mkdir
178. User mode can run the Privileged instructions.
- True
 - False
179. The manual pages can be read in Linux using _____ command.
- man
 - wan
 - desc
 - help
180. Swapper is also termed as Short term scheduler.
- 1
 - 0
181. First _____ entries in Per Process File Descriptor Table are used as soon as the process is created.
- 1
 - 2
 - 3
 - 4

AL-JUNAID INSTITUTE GROUP

182. Shared libraries and kernel modules are stored in _____ directory.

- a. /bin
- b. /dev
- c. /boot
- d. **/lib**

183. The _____ system call suspends the calling process.

- a. fork
- b. **wait pg#42**
- c. exec
- d. exit

184. The hardware mechanism that enables a device to notify CPU is called an _____

- a. **Interrupt google**
- b. Signal
- c. Trap
- d. Process

185. A _____ (or an *exception*) is a software-generated interrupt caused either by an error (division by zero or invalid memory access) or by a user request for an operating system service

- a. Interrupt
- b. **Trap (Page 7)**
- c. Signal
- d. Process

186. Which register holds the smallest legal physical memory address for a process?

- a. **Base register (Page 13)**
- b. Limit register
- c. Status register
- d. None of the given options

187. The -----semaphore provides mutual exclusion for accesses to the buffer pool and is initialized to the value 1.

- a. **mutex (Page 118)**
- b. binary
- c. counting
- d. none of the given options

AL-JUNAID INSTITUTE GROUP

188. Binary semaphores are those that have only two values-----
- 0 and n
 - 0 and 0
 - 0 and 1 (Page 117)**
 - None of the given options
189. Addresses generated *relative* to part of program, not to start of physical memory are
- Virtual
 - Physical
 - Relocatable**
 - Symbolic
190. Object files and libraries are combined by a----- program to produce the executable binary
- Compiler
 - Linker
 - Text editor
 - Loader**
191. Physical memory is broken down into fixed-sized blocks, called----- and Logical memory is divided into blocks of the same size, called -----
- Frames, pages (Page 165)**
 - Pages, Frames
 - Frames, holes
 - Holes, segments
192. A page table needed for keeping track of pages of the page table is called _____
- 2-level paging
 - Page directory (Page 173)**
 - Page size
 - Page table size
193. Each page is a power of ----- bytes long in paging scheme.
- 2
 - 3
 - 4 (Page 167)**
 - 5

AL-JUNAI D INSTITUTE GROUP

194. Which part of the computer system helps in managing the file and memory management system?

- a. **Operating System (Page 5)**
- b. Device Drivers
- c. Application Software
- d. Hardware

195. Which of the following is correct definition for wait operation?

a. **wait(S) { (Page 111)**
while(S<=0)
;// no op
S--;
}

b. wait(S) {
S++;
}

c. wait(S) {
while(S>=0)
;// no op
S--;
}

d. wait(S) {
S--;
}

196. Wrong use of wait and signal operations (in context with semaphores) can cause _____ problem(s).

- a. Mutual Exclusion
- b. Deadlock
- c. Bounded Waiting
- d. **All of the given options are correct**

197. If a system is not in a safe state, there can be no deadlocks.

- a. True
- b. **False (Page 137)**

198. An acyclic graph does not allow directories to have shared

AL-JUNAID INSTITUTE GROUP

subdirectories and files.

- a. True
- b. **False (Page 225)**

199. The size of pages and frames are same in logical memory and physical memory respectively.

- a. **True (Page 165)**
- b. False

200. Intel is basically designed for following Operating Systems except

- a. **MULTICS (Page 182)**
- b. OS/2
- c. Windows
- d. Linux

201. Following is NOT true about Virtual memory.

- a. **Virtual memory help in executing bigger programs even greater in size that of main memory.**
- b. Virtual memory makes the processes to stuck when the collective size of all the processes becomes greater than the size of main memory.
- c. Virtual memory also allows files and memory to be shared by several different processes through page sharing.
- d. Virtual memory makes the task of programming easier because the programmer need not worry about the amount of physical memory,

202. The execution of critical sections must NOT be mutually exclusive

- a. True
- b. **False (Page 100)**

203. The integer value of _____semaphores can not be greater than 1.

- a. Counting
- b. **Binary (Page 117)**
- c. Mutex
- d. Bounded buffer

204. Starvation is infinite blocking caused due to unavailability of resources.

- a. **True (Page 115)**
- b. False

AL-JUNAI D INSTITUTE GROUP

205. Progress and Bounded Waiting are some of the characteristics to solve the critical section problems.
- True (Page 101)**
 - False
206. The _____ requires that no reader will be kept waiting unless a writer has already obtained permission to use the shared object.
- first readers-writers problem (Page 119)**
 - second readers-writers problem
 - third readers-writers problem
 - fourth readers-writers problem
207. The process of holding at least one resource and waiting to acquire additional resources that are currently being held by other processes is known as _____.
- Mutual exclusion
 - Hold and wait (Page 131)**
 - No preemption
 - Circular wait
208. The condition where a set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set, is termed as _____.
- Deadlock (Page 130)**
 - Starvation
209. A program cannot execute unless whole or necessary part of it resides in the main memory.
- True**
 - False
210. In the C-Scan and C-Look algorithms, when the disk head reverses its direction, it moves all the way to the other end, without serving any requests, and then reverses again and starts serving requests.
- True (Page 249)**
 - False
211. The following requirement for solving critical section problem is known as _____.

AL-JUNAID INSTITUTE GROUP

“There exists a bound on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.”

- a. Progress
 - b. **Bounded Waiting (Page 101)**
 - c. Mutual Exclusion
 - d. Critical Region
212. The section of code after the critical section is called _____.
- a. Crystal section
 - b. **Entry section**
 - c. Remainder section
 - d. Exit section
213. A process is said to be in critical section if it executes code that manipulates shared data.
- a. **True (Page 100)**
 - b. False
214. _____ an integer variable accessible through wait and signal which are atomic operations.
- a. **Semaphore (Page 111)**
 - b. Mutex
 - c. Busy waiting
 - d. Signal
215. Software solution to critical section problem can run only in _____ environment.
- a. **Multiprocessor**
 - b. **Multithreading**
 - c. **Uniprocessor**
 - d. **Separate address spacing**
216. _____ integer shows the highest priority of a process in CPU scheduling
- a. **Small (Page 86)**
 - b. Large
217. Cache is non-volatile memory.

AL-JUNAI D INSTITUTE GROUP

- a. True
b. **False (Page 153)**
218. _____ is used due to un-used space in fixed size blocks/ pages.
a. **Internal fragmentation**
b. External fragmentation
c. Paging
d. MVT
219. Fragmentation when using ICMP for path MTU should be avoided.
a. **True**
b. False
220. Variable name are _____ address.
a. Physical
b. Reloadable
c. Relative
d. **Symbolic**
221. _____ is caused due to un-used in physical memory.
a. **Internal fragmentation**
b. External fragmentation
c. Paging
d. MVT
222. The run-time mapping from virtual to physical address is done by a piece of hardware in the CPU, called the _____
a. **Memory management unit (MMU) (Page 155)**
b. CPU scheduler
c. Registers
d. None of the above
223. Main memory is _____ memory.
a. **Volatile memory**
b. Non-volatile
c. Permanent
d. Virtual
224. An optimal page-replacement algorithm has the lowest page fault rate of all algorithms.

AL-JUNAID INSTITUTE GROUP

- a. **True (Page 199)**
- b. False

225. In _____, the library files are linked at load time.

- a. **Static linking**
- b. Dynamic linking

226. In swapping technique of Memory Management, the total amount transfer is directly proportional to the _____

- a. **amount of the memory swapped**
- b. amount of space on backing store
- c. space on main memory
- d. all the given options are correct

227. We want a page replacement algorithm with the _____ page-fault rate.

- a. **Lowest (Page 198)**
- b. Highest

228. In a UNIX system, _____ system call can be used to request the operating system to memory map an opened file.

- a. **mmap() (Page 195)**

229. The main memory is usually divided into two partitions, one for _____ and other for _____.

- a. **resident operating System, User processes (Page 158)**

230. A section of code or collection of operations in which only one process may be executing at a given time, is called critical section. Consider a system containing n processes {P₀, P₁, 2, ..., P_n }. Each process has a segment of code called a _____

- a. **N-Process Critical Section Problem**

231. Semaphore S is a/an _____ type of variable to use as synchronization tool.

- a. **Integer (Page 111)**

232. In order to remove the problem like busy waiting, some high level synchronization constructs are defined. What are they?

AL-JUNAID INSTITUTE GROUP

a. Critical regions and Monitors (Page 124)

233. In instruction TestAndSet mutual exclusion implementation is done by declaring a Boolean variable lock_____.

a. Initialized as false (Page 109)

234. We can use semaphores to deal with the number of _____ process critical section problem.

a. n-process critical section problem

235. Linux is a version of _____ operating system.

a. OS/2

b. Windows

c. Unix

d. None of the above

236. _____ scheduling algorithm is sometimes called shortest remaining time first scheduling algorithm.

a. Non-preemptive SJF

b. Priority Scheduling

c. Preemptive Shortest Job First (Page 85)

d. FCFS

237. Preemptive -----scheduling is sometimes called shortest remaining-time-first scheduling.

a. First-Come-First-Served (FCFS)

b. Round-Robin

c. Sorted Job First (SJF) (Page 85)

d. Priority

238. OS helps manages the following except

a. Application software

b. Bus speed of the system

c. Memory

d. Virtual memory

239. /usr/X11R6 is used by the X Window System.

a. True (Page 27)

b. False

AL-JUNAID INSTITUTE GROUP

240. /opt is used for storage of large applications.
a. **True (Page 27)**
b. False
241. The Home Directory for superuser in Linux and Unix is
a. /home
b. **/root (Page 27)**
c. None of the given
242. Linux Treats Devices as Files.
a. **True (Page 26)**
b. False
243. An absolute pathname starts with the root directory (/) and a relative pathname starts with your home directory.
a. **True (Page 25)**
b. False
244. We can install and run multiple OS by using VMWare.
a. **True**
b. False
245. Mach, MacOS X Server, QNX, OS/2 and Windows NT are examples of OS Based on_____.
a. Layered
b. **Micro Kernal (Page 22)**
c. Virtual Machine
d. None of The Given
246. In Layered Approach of OS, the Layer highest Layer is User Interface layer.
a. **True (Page 21)**
b. False
247. Operating System is the Manager of Hardware Resources.
a. **True (Page 6)**
b. False

AL-JUNAID INSTITUTE GROUP

248. An operating system is a control program that manages the execution of user programs to prevent errors and improper use of a computer.

a. **True (Page 6)**

b. **False**

249. The bottom-up view is that operating system is a resource manager who manages the hardware and software resources in the computer system.

a. **True (Page 6)**

b. **False**

250. _____ determines What will be done.

a. Mechanism

b. **Policy (Page 24)**

c. Mechanism and Policy

d. None of the given

251. copy file1 file2 is an example of _____ OS view.

a. **Top down (Page 6)**

b. **Bottum Up**

252. The Top-down view is that it is a program that acts as an intermediary between a user of a computer and the computer hardware, and makes the computer system convenient to use.

a. **True (Page 6)**

b. **False**

253. The Purpose of Operating System is to generate Executable Programs and to _____ them.

a. Regenerate

b. **Execute (Page 5)**

c. Store

d. Remove

254. Users are the People, machines or a computer that uses the Hardware resources.

a. **True (Page 4)**

b. **False**

AL-JUNAID INSTITUTE GROUP

255. Database, Compiler, Video games are examples of _____.
- Hardware
 - Application (Page 4)**
 - Operating System
 - Users
256. Hardware provide basic computing resource.
- True (Page 4)**
 - False
257. The priorities of processes in the _____ group remain fixed.
- Kernel (Page 93)**
 - User
258. The process of switching from one process to another is called latency.
- True
 - False (Page 34)**
259. In Unix/ Linux, by default the standard input file is attached to the _____.
- Mouse
 - Keyboard (Page 55)**
 - Light pen
 - Joystick
260. The nice value helps in assigning _____ to a process.
- Priority (Page 94)**
 - Weight
 - Time
 - Scheduling
261. You can use the rm file1 command to _____ file1
- Retrieve
 - Remove (Page 30)**
 - Make
 - modify
262. The correct command for compiling C program named program.c in Linux environment is
- gcc program.c -o FirstPrgram (Page 31)**

AL-JUNAI D INSTITUTE GROUP

- b. gcc -o FirstProgram program.c
- c. gcc -z FirstProgram program.c
- d. gcc program.c -m FirstPrgram

263. Using _____ system, we can create a new process in Linux.

a. **Fork (Page 39)**

- b. Exec
- c. Wait
- d. Exit

264. Cooperating processes never share any data, code, memory or state.

a. True

b. **False (Page 5)**

265. _____ command display the status of a process.

a. ls

b. **ps (Page 66)**

- c. gcc
- d. Cat

266. Swapper is also termed as Short term scheduler.

a. True

b. **False (Page 36)**

267. _____ system call is used to write to a file or FIFO or any other IPC channel.

a. read

b. **write (Page 48)**

- c. open
- d. Fork

268. A Process 'A' that has finished working but its parent process has also finished its execution. In this state the process 'A' will be called as _____ process.

a. Child

b. Thread

c. **Zombie (Page 42)**

d. Fork

AL-JUNAID INSTITUTE GROUP

269. _____ scheduling allows a process to move between queues.

- a. Round Robin
- b. First Come First Serve
- c. **Multilevel Feedback Queue (Page 92)**
- d. Shortest Remaining Time First

270. Round Robin algorithm is most suitable for _____.

- a. **Time sharing system (Page 88)**
- b. Real time systems and batch systems
- c. Running Batch programs
- d. Expert system

271. Kernel is responsible for scheduling the user level threads.

- a. True
- b. **False (Page 73)**

272. A system call _____

- a. **Is an entry point into the kernel code (Page 18)**
- b. Allows a program to request a kernel service
- c. Is a technique to protect I/O devices and other system resources
- d. All of the these

273. Operating System provides services such as Managing Primary and Secondary Storage, Processes and Allowing user to manage his/her files and directories.

- a. **True (Page 5)**
- b. False

274. _____ is used in real time operating systems.

- a. **Non-preemptive scheduling**
- b. Preemptive scheduling
- c. Dispatching scheduling
- d. FCFS scheduling

275. Deadlock detection and recovery technique is exactly similar to deadlock avoidance technique to handle deadlock in the system.

- a. **True**
- b. False

AL-JUNAID INSTITUTE GROUP

276. In Overlay technique, we can overload any part of the program with the part of the program required needed recently.

- a. True
- b. False**

277. _____ is the process of mapping a name to an address.

- a. Addressing
- b. Binding
- c. Routing
- d. Memory

278. _____ scheme works well if members of the team are to work on these shared files sequentially.

- a. Common Group
- b. Duplicating Files PG # 288**
- c. Common Login
- d. Link

279. _____ UNIX recognizes _____ modes of access

- a. One
- b. Two
- c. Three PG # 230**
- d. Four

280. The _____ algorithm selects the request with the minimum seek time from the current head position

- a. Scan
- b. SSTF PG # 244**
- c. Look
- d. C-Look

281. Which part of the computer system helps in managing the file and memory management system?

- a. Operating System**
- b. Device Drivers

AL-JUNAID INSTITUTE GROUP

- c. Application Software
d. Hardware
282. _____ is used to store data on secondary storage device.
- a. Block Special File
b. Link File
c. Ordinary File PG # 218
d. Directory
283. _____ is a way to establish a connection between the file to be shared and the directory entries of the users who want to have access to this file.
- a. Link PG # 229**
b. Directory
c. Common Group
d. Access Permission
284. Which one of the following indicates end of the file in a read () system call?
- a. -1
b. 0
c. 1
d. 2
285. In one of the deadlock prevention methods, impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration. This violates the _____ condition of deadlock.
- a. Mutual exclusion
b. Hold and Wait
c. Circular Wait PG # 134
d. No Preemption Right

AL-JUNAID INSTITUTE GROUP

286. The bottom layer in the layered approach of Operating System is ____.

a. User interface

b. Hardware **PG # 18**

c. Kernel

d. API

287. The non-critical section code in any critical section problem is termed as

a. Critical Section

b. Entry Section

c. Leave/Exit Section

d. Remainder Section

288. The process that satisfies all three requirements of critical section problem is called

—

a. Algorithm 1

b. Algorithm 2

c. Algorithm 3

d. Algorithm N

289. The main disadvantage of semaphore is _____.

a. Context switching

b. Busy waiting **PG # 111**

c. Synchronization

d. Deadlock

290. In Unix/Linux, pipes provide a method of one-way communication and for this reason it may be called

a. Soft pipes

b. FIFO named pipes

c. Full-duplex pipes

d. Half-duplex pipes **PG # 46**

291. Consider process A is trying to access a page that does not belong to its address space. Which of the following protection scheme provide protection against such actions under paging?

AL-JUNAID INSTITUTE GROUP

a.Valid/invalid (v) bit **PG # 170**

- b.Null bit/parity (n) bit
- c.Execute/parity (e) bit
- d.Write/null (w) bit

292. ----- is the separation of user logical memory from physical memory.

o Virtual Memory **PG # 186**

- o RAM
- o Physical memory
- o ROM

293. For page replacement algorithms, the page fault rate may increase as the number of allocated frames_____.

a. Keep Constant

b. Increases **PG # 200**

- c. Decreases
- d. Available

294. In the Scan algorithm, disk_____starts at one end of the disk, and moves toward the other end, servicing requests as it reaches each cylinder, until it gets to the other end of the disk.

a.Arm **PG # 245**

- b.Cylinder
- c.Head
- d.Vector

295. In case of thrashing, if CPU utilization is too low, the operating system_____the degree of multiprogramming

a.Increases **PG # 208**

- b.Decreases
- c.Increases or Decreases
- d.Keeps constant

296. When there is no external fragmentation, free memory blocks are scattered on hard disk then_____is used in order to utilize these blocks for space management.

AL-JUNAID INSTITUTE GROUP

- a. Indexed Allocation
 - b. Contiguous Allocation
 - c. Linked Allocation PG # 235**
 - d. Variable Allocation
297. In a Multilevel Queue, the foreground queue has ----- scheduling algorithm and background queue has ----- scheduling algorithm.
- a. First Come First Serve, Round-Robin
 - b. Round-Robin, First Come First Serve**
 - c. Round Robin, Round Robin
 - d. First Come First Serve, First Come First Serve
298. Main memory is a large array of -- called memory locations ranging in size from hundreds of thousands to billions.
- a. Interrupts
 - b. Registers
 - c. Digits
 - d. Words PG # 12**
299. The syntax for input redirection is
- a. command < input-file PG # 55**
 - b. command > input-file
 - c. command >= input-file
 - d. command =< input-file
300. The semaphore empty is initialized to the value----- ; the semaphore full is initialized to the value.
- a. 0,n
 - o n,0 PG # 116**
 - b. n,n
 - o 0,0
301. High-level synchronization construct that allows the safe sharing of an abstract data type among concurrent processes is called a -----

AL-JUNAID INSTITUTE GROUP

- a. Read/Write Lock
- b. Swap
- c. Semaphore
- d. **Monitor**

302. When drawing the resource allocation graph, processes are represented by _____ and resources by _____

- a. Squares, Squares
- b. Circles, Circles
- c. Circles, squares

d. Squares, Circles

303. Preventing the condition of _____ to happen, deadlocks can be prevented to happen.

- a. Critical region
- b. Monitors

c. Circular wait PG # 135

d. Critical section

304. If validation bit is 0, it indicates a/an --state of segment.

- a. Protected
- b. Shared
- c. Legal

d. Illegal PG # 178

305. Intel is basically designed for following Operating Systems except _.

- a. **MULTICS PG # 180**
- b. OS/2
- c. Windows
- d. Linux

306. The term **pager** is used in connection with _____.

- a. **Demand Paging PG # 187**
- b. Paging
- c. Segmentation
- d. Paged Segmentation

307. In deadlock detection and recovery algorithm, a deadlock exists in the

AL-JUNAID INSTITUTE GROUP

system if and only if the wait for graph contains a _

a.Cycle PG # 145

b.Graph

c.Edge

d.Node

308. Object files and libraries are combined by a ---- program to produce the executable binary

Compiler

b.Linker

c. Text editor

d.Loader

309. ----- holds the smallest legal physical memory address for a process

o Stack pointers register

o **Base register PG # 10**

o Limit register

o Index register

310. External Fragmentation takes place in _____

o **Multiprogramming with Variable Tasks (MVT) PG # 163**

o Multiprogramming with Fixed Tasks (MFT)

o Multi-tasking

o Paging

311. A process is _____ if it is spending more time on paging

o **Thrashing PG # 206**

o Demand paging

o Paging

o Fixed Allocation

312. Every _____ generated by the CPU is divided into two parts: a page number (p) and a page offset (d).

o Page

o Process address space

o Physical address

o **Logical address PG # 164**

AL-JUNAID INSTITUTE GROUP

313. The most important property of the working set is its _.

- o Delay
- o Thrashing
- o Time

o Size PG # 211

313. ___ algorithm is the optimal scheduling algorithm among all the non-preemptive scheduling algorithms.

• **Shortest Job First scheduling**

- First Come First Serve
- Priority Scheduling
- Round Robin Scheduling

315. The Swap instruction which is the hardware solution to synchronization problem does not satisfy the _ condition, hence not considered to be a good solution.

a. Bounded waiting

- b. Mutual exclusion
- c. No preemption
- d. Progress

316. The statement “A process may hold allocated resources while waiting for of other resources” best describes which one of the following conditions?

- a. Mutual Exclusion
- b. No preemption

c. Hold and wait

d. Circular wait

317. A state is ___ if the system can allocate resources for each process in some order and still avoid a deadlock.

- None
- Unsafe
- Starvation

Safe pg#134

318. ___ algorithm is used in Deadlock avoidance.

AL-JUNAID INSTITUTE GROUP

- Bakery
- Mutual exclusion

Banker's Google

- Safe sequence

319. _____ Typical
ly monitor, a high level synchronization tool is characterized by _____ and _____.

- Single, wait
- Global variable, local variable

Local data, programmer defined operator pg#122

- Local variables, semaphores

320. _____ The integer value of `_semaphores` can not be
greater than 1.

- Bounded buffer

Binary pg#114

- Counting
- Mutex

321. _____ Banker's algorithm is used for _____.

- Deadlock removal

Deadlock avoidance Google

- Deadlock detection
- Deadlock prevention

322. _____ Possible side effects of
_____ deadlocks are low device
utilization and reduced system throughput.

- Ignoring

Preventing Google

- Recovering
- Avoiding

323. _____ Preventing the condition of _____ to happen,
deadlocks can be prevented to happen.

- Critical section
- Monitors
- Critical region

AL-JUNAID INSTITUTE GROUP

. Circular wait pg#134

324. The process of holding at least one resource and waiting to acquire additional resources that are currently being held by other processes is known as _____.

. Hold and wait pg#128

- . Circular wait
- . No preemption
- . Mutual exclusion

325. The _____ requires that once a writer is ready, that writer performs its write as soon as possible. In other words, if a writer is waiting to access the object, no new readers may start reading.

- . third readers-writers problem
- . fourth readers-writers problem
- . first readers-writers problem

. second readers-writers problem pg#116

326. Binary semaphore whose integer value cannot be less than 1 is simpler to implement.

- . ≤ 1
- . ≥ 1
- . < 1

. > 1 pg#114

327. The condition where a set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set, is termed as _____.

. Deadlock pg#127

- . Starvation

328. Following is not the classical problem of synchronization.

- . Reader writer problem
- . Dining philosophers problem

. Counting semaphore problem pg#115

- . Bounded buffer problem

329. The integer value of _____ semaphores can range over an unrestricted

AL-JUNAID INSTITUTE GROUP

integer domain.

. Mutex

. **Counting pg#114**

. Bounded buffer

. Binary

330. The _____ scheme is not applicable to a resource allocation system with multiple instances of each resource type.

. None of the given options

. **Wait for graph pg#145**

. Resource allocation graph

. Both resource-allocation and wait-for graph

331. A dashed line is used to represent a _____ in Resource Allocation Graph.

. Assignment edge

. **Claim edge pg#135**

. Allocation edge

. Request edge

332. _____ is an integer variable accessible through wait and signal which are atomic operations.

. Signal

. **Semaphore pg#108**

. Busy waiting

. Mutex

333. _____ is an integer variable accessible through wait and signal which are atomic operations.

. Signal

. **Semaphore page 108**

. Busy waiting

. Mutex

334. Removing the possibility of deadlock in dining philosopher problem does not ensure the

_____ problem will not occur.

. **Starvation page 120**

AL-JUNAID INSTITUTE GROUP

- Mutual Exclusion
- Critical Section
- Bounded Buffer

335. In deadlock detection and recovery algorithm, a deadlock exists in the system if and only if the waitfor graph contains a ___.

- Edge
- Cycle page 144

- Graph
- Node

336. Deadlock ___ provides a set of methods for setting that at least one of the necessary conditions cannot hold.

- prevention page 132

- Avoidance
- Handling
- .Recovery

337. In Resource Allocation Graph, a ___ $P_i \rightarrow R_j$ indicates that process P_i may request resource R_j at some time in the future

- claim edge page 135

- assignment edge
- allocation edge
- request edge

338. Deadlock can be deal with ___ ways.

- 4
- 6

- 3 page 132

- 2

339. The problem of Deadlocks can be solved by ___ method(s).

- Deadlock avoidance
- Allowing deadlock to occur, then detect and recover

- All of the given page 132

- Deadlock prevention

AL-JUNAID INSTITUTE GROUP

340. _____ is used in the detection and recovery mechanism to handle deadlocks.

- Claim Edge Graph
- Resource allocation Graph
- Wait-for Graph page 144
- Circular Graph

341. Wrong use of wait and signal operations (in context with semaphores) can cause _____ problem(s).

- All of the given options are correct Page 112
- Bounded Waiting
- Mutual Exclusion
- Deadlock

342. If system is not in a safe state, there can be NO deadlocks.

- True
- False page 134

343. How does a logical-address space is represented?

Through Critical section

- Through segments

344. The main memory is usually divided into partitions, one for _____ and other for _____

- Operating System, User processes
- Operating system, CPU
- Processes, Virtual Memory
- Base Register, Limit Register

350. A program cannot execute unless whole or necessary part of it resides in the main memory.

- TRUE
- FALSE

351. The _____ protection scheme guards against a process trying to access a page that does not belong to its address space.

- Primary
- Hybrid
- Secondary
- Tertiary

AL-JUNAID INSTITUTE GROUP

352. The system maintains a _____ of all processes whose memory images are on the backing store or in memory and are ready to run.

- Que
- **Ready que**
- List
- Environment

353. If there exists a logical address space of 16 pages of 1024 words, each mapped into a physical memory of _____ frames.

- 64
- **32**
- 16
- 48

354. The size of pages and frames are same in logical memory and physical memory respectively.

- **TRUE**
- FALSE

355. Address Binding will be _____ in Multiprogramming with Variable Tasks (MVT)

- Run Time
- Load Time
- Dynamic Time
- **None of Given Right answer FIX**

356. Every _____ generated by the CPU is divided into two parts: a page number (p) and a page offset (d).

- Page
- **Logical Error**
- Physical Error
- Process Address Space

355. _____ points to the page table.

- Translation look-aside buffers
- Page offset
- Page-table length register (PRLR)
- **Page-table base register (PTBR)**

AL-JUNAID INSTITUTE GROUP

356. To eliminate external fragmentation in segmentation the scheme used is

_____.

Fixed size partition scheme

Variable size partition scheme

Fixed size paging scheme

Variable size paging scheme





CS604- Operating Systems
Solved MCQS
From Final term Papers

July 10,2013

MC100401285

Moaaz.pk@gmail.com

Mc100401285@gmail.com

PSMD01

FINALTERM EXAMINATION
Spring 2010
CS604- Operating Systems (Session - 4)

Question No: 1 (Marks: 1) - Please choose one

A _____ (or an *exception*) is a software-generated interrupt caused either by an error (division by zero or invalid memory access) or by a user request for an operating system service.

- ▶ Interrupt
- ▶ **Trap (Page 7)**
- ▶ Signal
- ▶ Process

Question No: 2 (Marks: 1) - Please choose one

Which register holds the smallest legal physical memory address for a process?

- ▶ **Base register (Page 13)**
- ▶ Limit register
- ▶ Status register
- ▶ None of the given options

Question No: 3 (Marks: 1) - Please choose one

The process of switching from one process to another is called -----

- ▶ **context switching (Page 34)**
- ▶ scheduling
- ▶ quantum period
- ▶ latency

دنیا میں سب سے مشکل کام اپنی اصلاح اور سب سے آسان کام دوسروں پر نکتہ چینی کرنا ہے

Question No: 4 (Marks: 1) - Please choose one

The -----semaphore provides mutual exclusion for accesses to the buffer pool and is initialized to the value 1.

- ▶ **mutex (Page 118)**
- ▶ binary
- ▶ counting
- ▶ none of the given options

Question No: 5 (Marks: 1) - Please choose one

Binary semaphores are those that have only two values-----

- ▶ 0 and n
- ▶ 0 and 0
- ▶ **0 and 1 (Page 117)**
- ▶ None of the given options

Question No: 6 (Marks: 1) - Please choose one

Addresses generated *relative* to part of program, not to start of physical memory are

- ▶ Virtual
- ▶ Physical
- ▶ **Relocatable** [Click here for detail](#)
- ▶ Symbolic

Question No: 7 (Marks: 1) - Please choose one

Object files and libraries are combined by a ----- program to produce the executable binary

- ▶ Compiler
- ▶ Linker
- ▶ Text editor
- ▶ **Loader** [Click here for detail](#)

Question No: 8 (Marks: 1) - Please choose one

Physical memory is broken down into fixed-sized blocks, called----- and Logical memory is divided into blocks of the same size, called -----

- ▶ **Frames, pages (Page 165)**
- ▶ Pages, Frames
- ▶ Frames, holes
- ▶ Holes, segments

اللہ کا خوف سب سے بڑی دانائی ہے

Muhammad Moaaz Siddiq – MCS(4th)

Moaaz.pk@gmail.com

**Campus:- Institute of E-Learning & Modern Studies
(IEMS) Samundari**

Question No: 9 (Marks: 1) - Please choose one

A page table needed for keeping track of pages of the page table is called -----

- ▶ 2-level paging
- ▶ **Page directory (Page 173)**
- ▶ Page size
- ▶ Page table size

Question No: 10 (Marks: 1) - Please choose one

The address generated by the CPU, after any indexing or other addressing-mode arithmetic, is called a ----- address, and the address it gets translated to by the MMU is called a ----- address.

- ▶ **Virtual, physical [click here for detail](#)**
- ▶ Hexadecimal, Binary,
- ▶ Valid, invalid
- ▶ Physical, Virtual

Question No: 11 (Marks: 1) - Please choose one

Each page is a power of ----- bytes long in paging scheme.

- ▶ 2
- ▶ 3
- ▶ **4 (Page 167)**
- ▶ 5

Question No: 12 (Marks: 1) - Please choose one

_____ is a way to establish a connection between the file to be shared and the directory entries of the users who want to have access to this file.

- ▶ **Link (Page 231)**
- ▶ Directory
- ▶ Common Group
- ▶ Access Permission

Question No: 13 (Marks: 1) - Please choose one

When a _____ link is created, a directory entry for the existing file is created

- ▶ Soft
- ▶ **Hard (Page 227)**
- ▶ Soft or Hard
- ▶ None of the given options

Question No: 14 (Marks: 1) - Please choose one

The _____ method requires each file to occupy a set of contiguous blocks on the disk.

- ▶ **Contiguous Allocation (Page 236)**
- ▶ Linked Allocation
- ▶ Indexed Allocation
- ▶ None of the given options

Question No: 15 (Marks: 1) - Please choose one

Which part of the computer system helps in managing the file and memory management system?

- ▶ **Operating System (Page 5)**
- ▶ Device Drivers
- ▶ Application Software
- ▶ Hardware

Question No: 16 (Marks: 1) . - Please choose one

Which of the following is correct definition for wait operation?

▶ **wait(S) { (Page 111)**
while(S<=0)
;// no op
S--;
}

▶ wait(S) {
S++;
}

▶ wait(S) {
while(S>=0)
;// no op
S--;
}

▶ wait(S) {
S--;
}

بري صحبت سے تھائی بہتر ہے اور تھائی سے نيك صحبت بہتر ہے

Muhammad Moaaz Siddiq – MCS(4th)

Moaaz.pk@gmail.com

**Campus:- Institute of E-Learning & Moderen Studies
(IEMS) Samundari**

Question No: 17 (Marks: 1) - Please choose one

Wrong use of wait and signal operations (in context with semaphores) can cause _____ problem(s).

- ▶ Mutual Exclusion
- ▶ Deadlock
- ▶ Bounded Waiting
- ▶ **All of the given options are correct**

Question No: 18 (Marks: 1) - Please choose one

If a system is not in a safe state, there can be no deadlocks.

- ▶ True
- ▶ **False (Page 137)**

Question No: 19 (Marks: 1) - Please choose one

If a process continues to fault, replacing pages, for which it then faults and brings back in right away. This high paging activity is called _____.

- ▶ paging
- ▶ **thrashing (Page 210)**
- ▶ page fault
- ▶ CPU utilization

Question No: 20 (Marks: 1) - Please choose one

In _____ page replace algorithm we will replace the page that has not been used for the longest period of time.

- ▶ counter based
- ▶ Least Frequently Used
- ▶ FIFO
- ▶ **LRU (Page 202)**

Question No: 21 (Marks: 1) . - Please choose one

Overlays are implemented by the _____

- ▶ Operating system
- ▶ **Programmer (Page 159)**
- ▶ Kernel
- ▶ Shell

Question No: 22 (Marks: 1) - Please choose one

An acyclic graph does not allow directories to have shared subdirectories and files.

- ▶ True
- ▶ **False (Page 225)**

Question No: 23 (Marks: 1) - Please choose one

The size of pages and frames are same in logical memory and physical memory respectively.

- ▶ **True (Page 165)**
- ▶ False

Question No: 24 (Marks: 1) - Please choose one

A modification of free-list approach in free space management is to store the addresses of n free blocks in the first free block. Known as _____.

- ▶ counting
- ▶ linked list
- ▶ bit vector
- ▶ **grouping (Page 243)**

Question No: 25 (Marks: 1) - Please choose one

In deadlock detection and recovery algorithm, a deadlock exists in the system if and only if the wait for graph contains a _____.

- ▶ **Cycle (Page 147)**
- ▶ Graph
- ▶ Edge
- ▶ Node

Question No: 26 (Marks: 1) - Please choose one

Intel is basically designed for following Operating Systems except _____.

- ▶ **MULTICS (Page 182)**
- ▶ OS/2
- ▶ Windows
- ▶ Linux

Question No: 27 (Marks: 1) - Please choose one

Following is NOT true about Virtual memory.

- ▶ **Virtual memory help in executing bigger programs even greater in size that of main memory.**
- ▶ Virtual memory makes the processes to stuck when the collective size of all the processes becomes greater than the size of main memory.
- ▶ Virtual memory also allows files and memory to be shared by several different processes through page sharing.
- ▶ Virtual memory makes the task of programming easier because the programmer need not worry about the amount of physical memory,

Question No: 28 (Marks: 1) - Please choose one

The execution of critical sections must NOT be mutually exclusive

- ▶ True
- ▶ **False (Page 100)**

Question No: 29 (Marks: 1) - Please choose one

The critical section problem can be solved by the following except

- ▶ Software based solution
- ▶ **Firmware based solution (Page 101)**
- ▶ Operating system based solution
- ▶ Hardware based solution

Question No: 30 (Marks: 1) - Please choose one

The bottom layer in the layered approach of Operating System is-----

- ▶ User interface
- ▶ **Hardware (Page 21)**
- ▶ Kernel
- ▶ None of the given options

FINAL TERM EXAMINATION
Spring 2010
CS604- Operating Systems (Session - 4)

Question No: 1 (Marks: 1) - Please choose one

You can display the contents (names of files and directories) of a directory in UNIX/Linux directory structure with the ----- command.

- ▶ l
- ▶ s
- ▶ **ls (Page 28)**
- ▶ none of the given options

ایماندار کو غصہ دیر سے آتا ہے اور جلدی دور ہو جاتا ہے

Muhammad Moaaz Siddiq – MCS(4th)
Moaaz.pk@gmail.com
Campus:- Institute of E-Learning & Modern Studies
(IEMS) Samundari

Question No: 2 (Marks: 1) - Please choose one

----- spend more time doing IO than computations

- ▶ short CPU bursts
- ▶ CPU bound processes
- ▶ **IO bound processes (Page 32)**
- ▶ None of the given options

Question No: 3 (Marks: 1) - Please choose one

-----*buffer* places no practical limit on the size of the buffer

- ▶ Bounded
- ▶ **Unbounded (Page 44)**
- ▶ Both Unbounded & bounded
- ▶ None of the given options

Question No: 4 (Marks: 1) - Please choose one

With -----you use condition variables.

- ▶ Semaphores
- ▶ Read/Write Locks
- ▶ Swaps
- ▶ **Monitor (Page 126)**

Question No: 5 (Marks: 1) - Please choose one

Deadlocks can be described more precisely in terms of a directed graph called a system -----

- ▶ Directed graph
- ▶ Critical path
- ▶ **Resource allocation graph [Click here for detail](#)**
- ▶ Mixed graph

Question No: 6 (Marks: 1) - Please choose one

The integer value of _____ semaphores can not be greater than 1.

- ▶ Counting
- ▶ **Binary (Page 117)**
- ▶ Mutex
- ▶ Bounded buffer

زندگی میں کامیابی کا یہی راز ہے کہ پریشانیوں سے پریشان مت بنو

Muhammad Moaaz Siddiq – MCS(4th)

Moaaz.pk@gmail.com

**Campus:- Institute of E-Learning & Modern Studies
(IEMS) Samundari**

Question No: 7 (Marks: 1) - Please choose one

Starvation is infinite blocking caused due to unavailability of resources.

- ▶ **True (Page 115)**
- ▶ False

Question No: 8 (Marks: 1) - Please choose one

The set of all physical addresses corresponding to the logical addresses is a ----- of the process

- ▶ **Physical address space (Page 155)**
- ▶ Process address space
- ▶ None of the given options
- ▶ Logical address space

Question No: 9 (Marks: 1) - Please choose one

----- indicates size of the page table

- ▶ translation look-aside buffers
- ▶ **Page-table length register (PTLR) (Page 169)**
- ▶ Page-table base register (PTBR)
- ▶ Page offset

Question No: 10 (Marks: 1) - Please choose one

If validation bit is 0, it indicates a/an ----- state of segment.

- ▶ protected
- ▶ shared
- ▶ legal
- ▶ **illegal (Page 180)**

Question No: 11 (Marks: 1) - Please choose one

In _____ allocation scheme free frames are equally divided among processes

- ▶ **Fixed Allocation (Page 207)**
- ▶ Proportional Allocation
- ▶ Priority Allocation
- ▶ None of the given options

Question No: 12 (Marks: 1) - Please choose one

Progress and Bounded Waiting are some of the characteristics to solve the critical section problems.

- ▶ **True (Page 101)**
- ▶ False

Question No: 13 (Marks: 1) - Please choose one

_____ is used to store data on secondary storage device, e.g., a source program(in C), an executable program.

- ▶ Block Special File
- ▶ Link File
- ▶ **Ordinary File (Page 220)**
- ▶ Directory

Question No: 14 (Marks: 1) - Please choose one

The basic purpose of _____ is to help the users to utilize the hardware resources for completing different tasks in a simplified manner

- ▶ **Operating system (Page 6)**
- ▶ Application software
- ▶ All Software
- ▶ All of the given

Question No: 15 (Marks: 1) - Please choose one

User mode can run the Privileged instructions

- ▶ **True (Page 11)**
- ▶ False

Question No: 16 (Marks: 1) - Please choose one

_____ wastes CPU cycles and hence is a problem in real multiprogramming system.

- ▶ **Busy waiting (Page 13)**
- ▶ Spinlock
- ▶ Critical section
- ▶ Mutex

Question No: 17 (Marks: 1) - Please choose one

The _____ requires that no reader will be kept waiting unless a writer has already obtained permission to use the shared object.

- ▶ **first readers-writers problem (Page 119)**
- ▶ second readers-writers problem
- ▶ third readers-writers problem
- ▶ fourth readers-writers problem

دنیا کی سب سے بڑی فتح نفس پر قابو رکھنا ہے

Muhammad Moaaz Siddiq – MCS(4th)

Moaaz.pk@gmail.com

**Campus:- Institute of E-Learning & Modern Studies
(IEMS) Samundari**

Question No: 18 (Marks: 1) - Please choose one

The process of holding at least one resource and waiting to acquire additional resources that are currently being held by other processes is known as_____.

- ▶ Mutual exclusion
- ▶ **Hold and wait (Page 131)**
- ▶ No preemption
- ▶ Circular wait

Question No: 19 (Marks: 1) - Please choose one

The condition where a set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set, is termed as _____.

- ▶ **Deadlock (Page 130)**
- ▶ Starvation

Question No: 20 (Marks: 1) - Please choose one

Banker's algorithm is used for _____

- ▶ **Deadlock avoidance (Page 140)**
- ▶ Deadlock detection
- ▶ Deadlock prevention
- ▶ Deadlock removal

Question No: 21 (Marks: 1) - Please choose one

A program can not execute unless whole or necessary part of it resides in the main memory.

- ▶ **True**
- ▶ False

Question No: 22 (Marks: 1) - Please choose one

The _____ requires that once a writer is ready, that writer performs its write as soon as possible , if a writer waiting to access the object, no new readers may start reading.

- ▶ first readers-writers problem
- ▶ **second readers-writers problem (Page 119)**
- ▶ third readers-writers problem
- ▶ fourth readers-writers problem

Question No: 23 (Marks: 1) - Please choose one

Which command, Display permissions and some other attributes for prog1.c in your current directory?

- ▶ **ls -l prog1.c (Page 234)**
- ▶ ls -d prog1.c
- ▶ ls file prog1.c
- ▶ ls -l prog1.c /Directory

Question No: 24 (Marks: 1) - Please choose one

In the C-Scan and C-Look algorithms, when the disk head reverses its direction, it moves all the way to the other end, without serving any requests, and then reverses again and starts serving requests.

- ▶ True (Page 249)
- ▶ False

Question No: 25 (Marks: 1) - Please choose one

In paged segmentation, we divide every segment in a process into _____ pages.

- ▶ Fixed size (Page 182)
- ▶ Variable size

Question No: 26 (Marks: 1) - Please choose one

Intel 80386 used paged segmentation with _____ level paging.

- ▶ One
- ▶ Two (Page 185)
- ▶ Three
- ▶ Four

Question No: 27 (Marks: 1) - Please choose one

The logical address of Intel 80386 is _____

- ▶ 36 bits
- ▶ 48 bits (Page 185)
- ▶ 64 bits
- ▶ 128 bits

Question No: 28 (Marks: 1) - Please choose one

The Swap instruction which is the hardware solution to synchronization problem does not satisfy the _____ condition, hence not considered to be a good solution.

- ▶ Progress
- ▶ Bounded waiting (Page 109)
- ▶ Mutual exclusion
- ▶ None of the given

جھوٹ انسان اور ایمان دونوں کا دشمن ہے

Muhammad Moaaz Siddiq – MCS(4th)

Moaaz.pk@gmail.com

Campus:- Institute of E-Learning & Modern Studies
(IEMS) Samundari

Question No: 29 (Marks: 1) - Please choose one

The -----scheme is not applicable to a resource allocation system with multiple instances of each resource type.

- ▶ **Wait for graph (Page 148)**
- ▶ Resource allocation graph
- ▶ Both Resource-allocation and wait-for graph
- ▶ None of the given options

Question No: 30 (Marks: 1) - Please choose one

The following requirement for solving critical section problem is known as_____.

“There exists a bound on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.”

- ▶ Progress
- ▶ **Bounded Waiting (Page 101)**
- ▶ Mutual Exclusion
- ▶ Critical Region

CS604 Solved Quizzes (Final term)

Quiz No.2(19-June-2013)

Question No: 1 of 10 (Marks: 1) - Please choose one

Consider a scenario in which one process P1 enters in its critical section, no other process is allowed to execute in its critical section. This is called -----

Mutual exclusion [Click here for detail](#)

Context switching
Multithreading
Progress

Question No: 1 of 10 (Marks: 1) - Please choose one

Following is not the classical problem of synchronization.

Bounded buffer problem
Reader writer problem
Dining philosophers problem
Counting Semaphore problem (Page 118)

Question No: 1 of 10 (Marks: 1) - Please choose one

Typically monitor, a high level synchronization tool is characterized by _____ and _____.

Global variable,local variable
Signal, wait
Local data, programmer defined operators (Page 125)
Local variables, semaphores

Question No: 1 of 10 (Marks: 1) - Please choose one

The section of code after the critical section is called _____.

Crystal section
Entry section
Remainder section
Exit section

Question No: 1 of 10 (Marks: 1) - Please choose one

A process is said to be in critical section if it executes code that manipulates shared data.

True (Page 100)

False

Question No: 1 of 10 (Marks: 1) - Please choose one

In producer-Consumer problem synchronization is required. On which shared area this synchronization actually affect?

Counter

Buffer

Entry section

Exit section

Question No: 1 of 10 (Marks: 1) - Please choose one

Critical section problem can be solved by using how many ways?

4

3 (Page 101)

1

2

Question No: 1 of 10 (Marks: 1) - Please choose one

_____ is an integer variable accessible through wait and signal which are atomic operations.

Semaphore (Page 111)

Mutex

Busy waiting

Signal

Question No: 1 of 10 (Marks: 1) - Please choose one

Software solution to critical section problem can run only in environment _____.

Multiprocessor

Multithreading

Uniprocessor

Separate address spacing

Question No: 10 of 10 (Marks: 1) - Please choose one

_____ integer shows the highest priority of a process in CPU scheduling

▶ **Small (Page 86)**

▶ Large

Muhammad Moaaz Siddiq – MCS(4th)

Moaaz.pk@gmail.com

**Campus:- Institute of E-Learning & Moderen Studies
(IEMS) Samundari**

Question No: 1 of 10 (Marks: 1) - Please choose one

Removing the possibility of deadlock in dining philosopher problem does not ensure the _____ problem will not occur.

Mutual Exclusion

Starvation (Page 123)

Critical Section

Bounded Buffer

Question No: 1 of 10 (Marks: 1) - Please choose one

The priority of a process can be changed using _____ command.

▶ **nice (Page 94)**

▶ cmd

▶ Cat

▶ grep

Question No: 1 of 10 (Marks: 1) - Please choose one

The integer value of _____ semaphores can range over an unrestricted integer domain.

▶ **Counting (Page 117)**

▶ Binary

▶ Mutex

▶ Bounded buffer

Question No: 1 of 10 (Marks: 1) - Please choose one

_____ is a preemptive scheduling algorithm.

▶ First Come First Serve

▶ Shortest Job First

▶ **Round Robin (Page 89)**

▶ None of these

Question No: 1 of 10 (Marks: 1) - Please choose one

_____ algorithm is used for solving n-process critical section problem.

▶ Bankers

▶ **Bakery (Page 105)**

▶ Babbles

▶ None of the given

Question No: 5 of 10(Marks: 1) - Please choose one

Batch programs are usually _____ programs.

- ▶ Interactive
- ▶ **Non-interactive** [click here for detail](#)
- ▶ Foreground
- ▶ Preemptive

Question No: 1 of 10(Marks: 1) - Please choose one

Using hardware solution to synchronization for complex problems, introduce a new synchronization tool know as _____.

TestAndSet

Semaphore (Page 111)

Swap

Trap

Question No: 1 of 10 (Marks: 1) - Please choose one

Use of semaphore create a problem of busy waiting, this wastes CPU cycles that some other process may be able to use productively. This type of semaphore is also called _____

Semaphore S

Spinlock (Page 112)

Locking Semaphore

Mutex

Question No: 1 of 10 (Marks: 1) - Please choose one

----- is a segment of code that accesses a shared resource like data structure or device that must not be concurrently accessed by more than one thread of execution.

Multithreading

Context switching

Critical section (Page 105)

Pipelining

Question 1 of 10 (Marks: 1) - Please choose one

Cache is non-volatile memory.

- ▶ True
- ▶ **False** (Page 153)

عقل مند کہتا ہے میں کچھ نہیں جانتا جبکہ بے وقوف کہتا ہے کہ میں سب کچھ جانتا ہوں

Muhammad Moaaz Siddiq – MCS(4th)

Moaaz.pk@gmail.com

**Campus:- Institute of E-Learning & Moderen Studies
(IEMS) Samundari**

Question No: 1 of 10 (Marks: 1) - Please choose one

While executing the statement c++/c-- in Producer-Consumer problem, at back end certain number of instructions are executed, if interleaving of statements happen, it create race condition. Tell number of instructions that require “no interleaving” while executing c++/c--?

- 3
- 1
- 2
- 0

Question 1 of 10 (Marks: 1) - Please choose one

The collection of process that is waiting on the disk to be brought into the memory for execution forms the _____

▶ **Input queue** (Page 154)

- ▶ Output queue
- ▶ Both of the
- ▶ None of the above

Question 1 of 10 (Marks: 1) - Please choose one

_____ is used due to un-used space in fixed size blocks/ pages.

▶ **Internal fragmentation** [Click here for detail](#)

- ▶ External fragmentation
- ▶ Paging
- ▶ MVT

Question 1 of 10 (Marks: 1) - Please choose one

Fragmentation when using ICMP for path MTU should be avoided.

- ▶ **True**
- ▶ False

Question 1 of 10 (Marks: 1) - Please choose one

Variable name are _____ address.

- ▶ Physical
- ▶ Reloadable
- ▶ Relative
- ▶ **Symbolic** [Click here for detail](#)

خود کو تمہیں سے بڑھ کر کوئی اچھا مشورہ نہیں دے سکتا

Question 1 of 10 (Marks: 1) - Please choose one
Secondary storage memory devices have ____ memory.

- ▶ Volatile
- ▶ **Permanent and non volatile** [Click here for detail](#)
- ▶ Temporary
- ▶ None of the

Question 1 of 10 (Marks: 1) - Please choose one
_____ is caused due to un-used in physical memory.

- ▶ **Internal fragmentation** [Click here for detail](#)
- ▶ External fragmentation
- ▶ Paging
- ▶ MVT

Question 1 of 10 (Marks: 1) - Please choose one
The run-time mapping from virtual to physical address is done by a piece of hardware in the CPU, called the _____

- ▶ **Memory management unit (MMU)** (Page 155)
- ▶ CPU scheduler
- ▶ Registers
- ▶ None of the above

Question 1 of 10 (Marks: 1) - Please choose one
Main memory is _____ memory.

- ▶ **Volatile memory** [Click here for detail](#)
- ▶ Non-volatile
- ▶ Permanent
- ▶ Virtual

Question 1 of 10 (Marks: 1) - Please choose one
What do we name to an address that is generated by CPU?

- ▶ **Logical address** (Page 152)
- ▶ Physical address
- ▶ Binary address
- ▶ None of the above

جو شخص ناکامیوں سے ٹر کر بھاگتا ہے کامیابی اُس سے ٹر کر بھاگتی ہے

Question 1 of 10 (Marks: 1) - Please choose one

Address Binding will be at _____ in multiprogramming with fixed tasks (MFT)

- ▶ Rub time
- ▶ **Load time (Page 160)**
- ▶ Dynamic time
- ▶ None of the

Question 1 of 10 (Marks: 1) - Please choose one

In _____ technique, memory is divided into several fixed-size partitions.

- ▶ Swapping
- ▶ Overlays
- ▶ **Multiprogramming with fixed tasks (MFT) (Page 159)**
- ▶ Multiprogramming with fixed tasks (MFT)

Question 1 of 10 (Marks: 1) - Please choose one

_____ is used in the detection and recovery mechanism to handle deadlocks.

- ▶ **Wait-for graph (Page 144)**
- ▶ Resource allocation graph
- ▶ Circular graph
- ▶ Claim edge graph

Question 1 of 10 (Marks: 1) - Please choose one

An optimal page-replacement algorithm has the lowest page fault rate of all algorithms.

- ▶ **True (Page 199)**
- ▶ False

Question 1 of 10 (Marks: 1) - Please choose one

_____ Point to the page table.

- ▶ Translation look-aside buffers
- ▶ Page offset
- ▶ Page-table length registers (PRLR)
- ▶ **Page-table base registers (PTBR) (Page 166)**

Question 1 of 10 (Marks: 1) - Please choose one

The segment table maps the _____ the physical addresses.

- ▶ Page addresses

- ▶ Shared page addresses
- ▶ One-dimensional logical addresses
- ▶ **Two-dimensional logical addresses (Page 175)**

Question 1 of 10 (Marks: 1) - Please choose one

Segmentation is a memory management scheme that support_____?

- ▶ **Programmer's view of memory (Page 175)**
- ▶ System's view of memory
- ▶ Hardware's view of memory
- ▶ None of the given

Question 1 of 10 (Marks: 1) - Please choose one

The pager is used in connection with _____.

- ▶ **Demand paging (Page 186)**
- ▶ Paging
- ▶ Segmentation
- ▶ Page segmentation

Question 1 of 10 (Marks: 1) - Please choose one

When the process tries to access locations that are not in memory, the hard traps the operating system. This is called as_____.

- ▶ **Page fault (Page 188)**
- ▶ Page replacement
- ▶ Paging
- ▶ Segmentation

Question 1 of 10 (Marks: 1) - Please choose one

The main criteria of page replacement in optimal page replacement algorithm is to_____

- ▶ **Replacement that page will not be use for the longest period of time (Page 199)**
- ▶ Replacement that page will be required most frequently in the execution of process
- ▶ Replace the page which is biggest in size.

Question 1 of 10 (Marks: 1) - Please choose one

-----refers to the situation when free memory space exists to load a process in the memory but the space is not contiguous.

- ▶ Segmentation
- ▶ Internal fragmentation
- ▶ Swapping
- ▶ **External Fragmentation (Page 165)**

جو لوگوں کے سامنے فخر کرتا ہے وہ لوگوں کی نظروں سے گر جاتا ہے

Question 1 of 10 (Marks: 1) - Please choose one
_____ algorithm is used in Deadlock avoidance.

- ▶ Bakery
- ▶ **Banker's (Page 139)**
- ▶ Mutual exclusion
- ▶ Safe Sequence

Question 1 of 10 (Marks: 1) - Please choose one
-----keep in memory only those instructions and data that are needed at any given time.

- ▶ Fragmentation
- ▶ Paging
- ▶ Swapping
- ▶ **Overlays (Page 156)**

Question 1 of 10 (Marks: 1) - Please choose one
In _____, the library files are linked at load time.

- ▶ **Static linking** [Click here for detail](#)
- ▶ **Dynamic linking**

Question 1 of 10 (Marks: 1) - Please choose one
In swapping technique of Memory Management, the total amount transfer is directly proportional to the _____

- ▶ **amount of the memory swapped** [Click here for detail](#)
- ▶ amount of space on backing store
- ▶ space on main memory
- ▶ all the given options are correct

Question 1 of 10 (Marks: 1) - Please choose one
When the address used in a program gets converted to an actual physical RAM address, it is called --

- ▶ Execution
- ▶ Loading
- ▶ **Address Binding** [Click here for detail](#)
- ▶ Compiling

عقل مند اپنے عیب خود دیکھتا ہے اور بیوقوفوں کے عیب دنیا دیکھتی ہے

Question 1 of 10 (Marks: 1) - Please choose one

If the system can allocate resources to each process in some order and still avoid a deadlock then it is said to be in _____ state.

▶ Safe (Page 137)

- ▶ Un-Safe
- ▶ Mutual
- ▶ Starvation

Question 1 of 10 (Marks: 1) - Please choose one

_____ register contains the size of the process

- ▶ Base register
- ▶ Index register
- ▶ Limit register (Page 13)
- ▶ Stack pointers register

Question 1 of 10 (Marks: 1) - Please choose one

In Resource Allocation Graph, a _____ $P_i \rightarrow R_j$ indicates that process P_i may request resource R_j at some time in the future.

▶ Claim edge (Page 138)

- ▶ Request edge
- ▶ Assignment edge
- ▶ Allocation edge

Question 1 of 10 (Marks: 1) - Please choose one

What do we name to an address that is loaded into the memory-address register of the memory?

- ▶ Logical address
- ▶ Physical address (Page 155)
- ▶ Binary addresses
- ▶ None of the given options

Question 1 of 10 (Marks: 1) - Please choose one

The ----- is a single program that produces an object file

- ▶ Linker

▶ **Compiler** [Click here for detail](#)

▶ Loader

▶ Text editor

Question No: 1 of 10 (Marks: 1) - Please choose one

Preventing a condition of _____ to happen, deadlocks can be prevented to happen.

▶ Critical region

▶ **Circular wait** (Page 136)

▶ Monitors

▶ Critical section

Question No: 1 of 10 (Marks: 1) - Please choose one

A condition where a set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set is termed as _____.

▶ **Deadlock** (Page 130)

▶ Starvation

Question No: 1 of 10 (Marks: 1) - Please choose one

The following is NOT a classical problem of synchronization

▶ Bounded buffer problem

▶ Reader writer problem

▶ Dining philosopher's problem

▶ **Counting semaphore problem** (Page 118)

Question 1 of 10 (Marks: 1) - Please choose one

The condition in which a set $\{P_0, P_1 \dots P_n\}$ of waiting processes must exist such that P_0 is waiting for a resource that is held by P_1 , P_1 is waiting for a resource that is held by P_2 , and so on, P_{n-1} is waiting for a resource held by P_n , and P_n is waiting for a resource held by P_0 . This condition is known as _____.

▶ Mutual exclusion

▶ Hold and wait

▶ No preemption

▶ **Circular wait** (Page 131)

Question No: 9 (Marks: 1) - Please choose one

A semaphore that cause Busy-Waiting is termed as _____.

▶ **Spinlock** (Page 113)

▶ Monitor

Muhammad Moaaz Siddiq – MCS(4th)

Moaaz.pk@gmail.com

**Campus:- Institute of E-Learning & Moderen Studies
(IEMS) Samundari**

- ▶ Critical region
- ▶ Critical section

Question No: 5 of 10 (Marks: 1) - Please choose one

The -----scheme is not applicable to a resource allocation system with multiple instances of each resource type.

- ▶ **Wait for graph (Page 148)**
- ▶ Resource allocation graph
- ▶ Both Resource-allocation and wait-for graph
- ▶ None of the given options

Question No: 2 of 10 (Marks: 1) - Please choose one

The _____ requires that once a writer is ready, that writer performs its write as soon as possible , if a writer waiting to access the object, no new readers may start reading.

- ▶ first readers-writers problem
- ▶ **second readers-writers problem (Page 119)**
- ▶ third readers-writers problem
- ▶ fourth readers-writers problem

Question No: 5 of 10 (Marks: 1) - Please choose one

Starvation is infinite blocking caused due to unavailability of resources.

- ▶ **True (Page 115)**
- ▶ False

Question 1 of 10 (Marks: 1) - Please choose one

In pages segmentation, the logical address is legal if d is _____segment length.

- ▶ **< (less than) (Page 180)**
- ▶ >(greater than)
- ▶ =(equal to)

Question 1 of 10 (Marks: 1) - Please choose one

In _____ allocation scheme number of frames allocated to a process is proportional to its size .

- ▶ **Proportional Allocation (Page 207)**
- ▶ Fixed allocation
- ▶ Priority allocation
- ▶ None of these

بصورت چہرہ بصورت لماغ سے بہتر ہے

Question No: 1 of 10 (Marks: 1) - Please choose one

In Resource Allocation Graph, A _____ $P_i \rightarrow R_j$ indicates that process P_i may request resource R_j at some time in the future.

▶ **Claim edge (Page 138)**

▶ Request edge

▶ Assignment edge

▶ Allocation edge

ub

Question No: 14 (Marks: 1) - Please choose one

A _____ is an integer variable that, apart from initialization is accessible only through two standard atomic operations: wait and signal.

▶ **Semaphore (Page 111)**

▶ Monitor

▶ Critical region

▶ Critical section

Question 1 of 10 (Marks: 1) - Please choose one

In case of thrashing if CPU utilization is too low the operating system _____ the degree of multiprogramming.

▶ **Increases (Page 207)**

▶ Decrease

Question 1 of 10 (Marks: 1) - Please choose one

We want a page replacement algorithm with the _____ page-fault rate.

▶ **Lowest (Page 198)**

▶ Highest

Question 1 of 10 (Marks: 1) - Please choose one

In a UNIX system, _____ system call can be used to request the operating system to memory map an opened file.

▶ **mmap() (Page 195)**

Muhammad Moaaz Siddiq – MCS(4th)

Moaaz.pk@gmail.com

**Campus:- Institute of E-Learning & Modern Studies
(IEMS) Samundari**

Question 1 of 10 (Marks: 1) - Please choose one

The high paging activity is called _____

▶ [Thrashing \(Page 207\)](#)

Question 1 of 10 (Marks: 1) - Please choose one

The main memory is usually divided into two partitions, one for _____ and other for _____.

▶ [resident operating System, User processes \(Page 158\)](#)

Question No: 1 of 10 (Marks: 1) - Please choose one

A section of code or collection of operations in which only one process may be executing at a given time, is called critical section. Consider a system containing n processes {P0, P1, 2, ..., Pn }. Each process has a segment of code called a _____

▶ [N-Process Critical Section Problem Click here for detail](#)

Question No: 1 of 10 (Marks: 1) - Please choose one

Semaphore S is a/an _____ type of variable to use as synchronization tool.

▶ [Integer \(Page 111\)](#)

Question No: 1 of 10 (Marks: 1) - Please choose one

In order to remove the problem like busy waiting, some high level synchronization constructs are defined. What are they?

▶ [Critical regions and Monitors \(Page 124\)](#)

Question No: 1 of 10 (Marks: 1) - Please choose one

In instruction TestAndSet mutual exclusion implementation is done by declaring a Boolean variable lock _____.

▶ [Initialized as false \(Page 109\)](#)

Question No: 1 of 10 (Marks: 1) - Please choose one

We can use semaphores to deal with the number of _____ process critical section problem.

▶ [n-process critical section problem](#)

عقل مند آدمی اس وقت تک نہیں بولتا جب تک خاموشی نہیں ہو جاتی

CS604 – Some More Quizzes

Question No: 1 (Marks: 1) - Please choose one

_____ command to resume the execution of a suspended job in the foreground

▶ **fg (Page 68)**

- ▶ bg
- ▶ jobs
- ▶ kill

Question No: 2 (Marks: 1) - Please choose one

_____ commands in Linux is used to copy file

▶ is

▶ **cp (Page 30)**

- ▶ mv
- ▶ mkdir

Question No: 3 (Marks: 1) - Please choose one

The process id returned to the child process after successful fork system call execution is _____.

▶ **0 (Page 40)**

- ▶ 1
- ▶ 2
- ▶ 3

Question No: 4 (Marks: 1) - Please choose one

In _____ addressing, the recipient is not required to name the sender.

▶ Symmetric

▶ **Asymmetric (Page 47)**

- ▶ Both symmetric and asymmetric
- ▶ None of the given options

Question No: 5 (Marks: 1) - Please choose one

A solution to the critical section problem must satisfy the following requirements

- ▶ Progress
- ▶ Mutual exclusion
- ▶ Bounded Waiting
- ▶ **All of these (Page 101)**

Question No: 6 (Marks: 1) - Please choose one

Typically the execp system call is used after a fork system call.

- ▶ **True (Page 39)**
- ▶ False

Question No: 7 (Marks: 1) - Please choose one

You can create a threads by using the pthread_create() call.

- ▶ **True (Page 76)**
- ▶ False

Question No: 8 (Marks: 1) - Please choose one

The interval from the time of submission to the time of completion is the _____

- ▶ **Turnaround time (Page 83)**
- ▶ Waiting time
- ▶ Response time
- ▶ None of all these

Question No: 9 (Marks: 1) - Please choose one

Each process must first request permission to enter its critical section. The section of code implementing this request is called the _____

- ▶ **entry section (Page 100)**
- ▶ Critical Section
- ▶ remainder section
- ▶ None of all these

Question No: 10 (Marks: 1) - Please choose one

IPC provides a mechanism to allow processes to communicate and to synchronize their actions without sharing the same_____

- ▶ **Address space (Page 46)**
- ▶ Address Name
- ▶ Address ID
- ▶ None of all these

Question No: 11 (Marks: 1) - Please choose one

Linux is a version of _____ operating system.

- ▶ OS/2
- ▶ Windows
- ▶ **Unix** [click here for detail](#)
- ▶ None of the above

Question No: 12 (Marks: 1) - Please choose one

Current working directory can be accessed using ----- Command.

- ▶ . (dot)
- ▶ # (hash)
- ▶ / (slash)
- ▶ **~ (tilt)** (Page 25)

Question No: 13 (Marks: 1) - Please choose one

Mkfifo() is a _____.

- ▶ **Library Call** (Page 58)
- ▶ Command
- ▶ Directory
- ▶ None of Above

Question No: 14 (Marks: 1) - Please choose one

_____ command gives a snapshot of the current processes.

- ▶ **ps** (Page 66)
- ▶ top
- ▶ who
- ▶ ls

Question No: 15 (Marks: 1) - Please choose one

Time interval when the I/O Devices are accessed is known as -----.

- ▶ CPU Burst
- ▶ **IO Burst** [Click here for detail](#)
- ▶ Time Slice
- ▶ None of Above

Question No: 16 (Marks: 1) - Please choose one

_____ directory includes essential system boot files including the kernel image.

- ▶ /bin
- ▶ **/boot** (Page 26)
- ▶ /dev
- ▶ /etc

Question No: 17 (Marks: 1) - Please choose one

_____ scheduling algorithm is sometimes called shortest remaining time first scheduling algorithm.

- ▶ Non-preemptive SJF
- ▶ Priority Scheduling
- ▶ **Preemptive Shortest Job First (Page 85)**
- ▶ FCFS

Question No: 18 (Marks: 1) - Please choose one

A program in execution is called a _____.

- ▶ Command
- ▶ **Process (Page 31)**
- ▶ Software
- ▶ Compiler

Question No: 19 (Marks: 1) - Please choose one

The manual pages can be read in Linux using _____ command.

- ▶ **man (Page 27)**
- ▶ wan
- ▶ desc
- ▶ help

Question No: 20 (Marks: 1) - Please choose one

The hardware mechanism that enables a device to notify CPU is called an -----

- ▶ **Interrupt [click here for detail](#)**
- ▶ Signal
- ▶ Trap
- ▶ Process

Question No: 21 (Marks: 1) - Please choose one

The ----- system call suspends the calling process.

- ▶ fork
- ▶ **wait (Page 42)**
- ▶ exec
- ▶ exit

Question No: 22 (Marks: 1) - Please choose one

You can use the ----- command to display the status of suspended and background processes

- ▶ fg

- ▶ bg
- ▶ **jobs** (Page 68)
- ▶ kill

انسان دکھ نہیں دیتے بلکہ انسانوں سے وابستہ امیدیں دکھ دیتی ہیں

Question No: 23 (Marks: 1) - Please choose one

You can terminate a foreground process by pressing -----

- ▶ <Ctrl-A>
- ▶ **<Ctrl-C>** (Page 69)
- ▶ <Ctrl-Z>
- ▶ None of the given options

Question No: 24 (Marks: 1) - Please choose one

A time sharing system is

- ▶ Multi tasking
- ▶ Interactive
- ▶ Multi user
- ▶ **All of these** (Page 9)

Question No: 25 (Marks: 1) - Please choose one

The main characteristic of a Real time system is

- ▶ Efficiency
- ▶ Large Virtual Memory
- ▶ Large secondary storage device
- ▶ **Usability** [click here for detail](#)

Question No: 26 (Marks: 1) - Please choose one

Shared libraries and kernel modules are stored in directory

- ▶ /bin
- ▶ /dev
- ▶ /boot
- ▶ **/lib** (Page 26)

Question No: 27 (Marks: 1) - Please choose one

_____scheduler selects the process from the job pool and put them in main memory.

- ▶ **Long term** (Page 36)
- ▶ Short term
- ▶ Medium term

▶ Swapper

Question No: 28 (Marks: 1) - Please choose one

In indirect inter process communication, a sender_____mention the name of the recipient.

▶ do

▶ **do not (Page 47)**

Question No: 29 (Marks: 1) - Please choose one

The performance of Round Robin algorithm does NOT depends heavily on the size of the time quantum.

▶ **True (Page 89)**

▶ False

Question No: 30 (Marks: 1) - Please choose one

_____ is also called Swapper.

▶ Swap space

▶ **Medium term scheduler (Page 37)**

▶ Short term scheduler

▶ Long term scheduler

Question No: 31 (Marks: 1) - Please choose one

Linux OS can support multiple users at a time

▶ **True (Page 9)**

▶ False

Question No: 32 (Marks: 1) - Please choose one

The Operating system is a layer of software between _____ and _____.

▶ **hardware, software application (Page 21)**

▶ Kernel, hardware

▶ Dos, Windows

▶ Windows, Kernel

Question No: 33 (Marks: 1) - Please choose one

The major advantage of multi-programming system is

▶ More than one jobs can be processed at a given time

▶ **CPU utilization can be increased (Page 8)**

▶ Jobs can be completed quickly

▶ All of the options are correct

Question No: 34 (Marks: 1) - Please choose one

Command-line interpreter is also called _____ in some operating systems.

- ▶ Kernel
- ▶ **Shell (Page 16)**
- ▶ Signal
- ▶ API

Question No: 35 (Marks: 1) - Please choose one

I/O instructions are Privileged Instructions.

- ▶ **True (Page 12)**
- ▶ False

Question No: 36 (Marks: 1) - Please choose one

In Linux directory structure, there is _____ root directory.

- ▶ **1 (Page 26)**
- ▶ 2
- ▶ 3
- ▶ 4

Question No: 37 (Marks: 1) - Please choose one

Utilities used for system administration (halt, ifconfig, fdisk, etc.) are stored in _____ directory.

- ▶ /dev
- ▶ /boot
- ▶ /lib
- ▶ **/sbin (Page 27)**

Question No: 38 (Marks: 1) - Please choose one

rm and [r]mkdir commands are used to _____ directory.

- ▶ Create
- ▶ Move
- ▶ **Remove (Page 30)**
- ▶ Modify

Question No: 39 (Marks: 1) - Please choose one

You can use the mv file1 file2 command to move _____

- ▶ **file1 to file2. (Page 30)**
- ▶ file 2 to file 1
- ▶ this command will not work for moving files
- ▶ None of the option is correct.
- ▶ Both option a and b are correct

Question No: 40 (Marks: 1) - Please choose one

Taking the CPU from one process and giving the CPU to another process is termed as

▶ **Context Switching** [click here for detail](#)

- ▶ Dispatching
- ▶ Swapping
- ▶ Tracking

Question No: 41 (Marks: 1) - Please choose one

A Process that has finished working, as well as its parent process has also finished its execution. In this state the process A will be called as _____ process.

▶ **Child**

- ▶ Thread
- ▶ Zombie
- ▶ Fork

Question No: 42 (Marks: 1) - Please choose one

Bounded Buffer is a buffer of _____ size

- ▶ variable
- ▶ **fixed** (Page 44)

Question No: 43 (Marks: 1) - Please choose one

In _____ communication the process which wants to communicate with the other process must explicitly name the recipient and the sender.

▶ **Direct** (Page 46)

- ▶ Indirect
- ▶ Automatic
- ▶ Self

Question No: 44 (Marks: 1) - Please choose one

If the fork system call fails, it returns

- ▶ 1
- ▶ **-1** (Page 40)
- ▶ 2
- ▶ 0

Question No: 45 (Marks: 1) - Please choose one

When a process opens its first file explicitly it will get descriptor number _____

- ▶ 1
- ▶ 2
- ▶ **3** [click here for detail](#)

▶ 4

Question No: 46(Marks: 1) - Please choose one

1 MB or 1 megabyte is equivalent to----

- ▶ 1024 bytes
- ▶ **1024² bytes** [click here for detail](#)
- ▶ 1024³ bytes
- ▶ 1000000 bytes

Question No: 47 (Marks: 1) - Please choose one

-----has a hierarchical file system structure.

- ▶ DOS
- ▶ Windows
- ▶ **UNIX (Page 25)**
- ▶ None of the given options

Question No: 48 (Marks: 1) - Please choose one

You can use the -----command in UNIX to create a directory.

- ▶ rmdir
- ▶ **mkdir (Page 29)**
- ▶ cp
- ▶ gcc

Question No: 49 (Marks: 1) - Please choose one

Files that start with a ----- in UNIX/Linux directory structure are known as hidden files .

- ▶ **.(dot) (Page 28)**
- ▶ # (hash)
- ▶ / (slash)
- ▶ ~ (tilt)

Question No: 50 (Marks: 1) - Please choose one

The creating process is called a----- process while the new processes are called the ----- of that process

- ▶ None of the given options
- ▶ Children, parent
- ▶ **Parent, children (Page 38)**
- ▶ Zombie, single

Question No: 51 (Marks: 1) - Please choose one

The _____ are used for communication between related or unrelated processes on the same system or unrelated

Muhammad Moaaz Siddiq – MCS(4th)

Moaaz.pk@gmail.com

**Campus:- Institute of E-Learning & Moderen Studies
(IEMS) Samundari**

processes on different systems.

- ▶ Pipes
- ▶ **BSD Sockets (Page 53)**
- ▶ Named pipe (FIFO)
- ▶ None of the given options

Question No: 52 (Marks: 1) - Please choose one

A_____ is an abstract key for accessing a file.

- ▶ **File descriptor [click here for detail](#)**
- ▶ Input Redirection
- ▶ Output Redirection
- ▶ FIFO

Question No: 53 (Marks: 1) - Please choose one

You can display all of the signals supported by your system, along with their numbers, by using the ----- command

- ▶ <Ctrl-A>
- ▶ fg
- ▶ jobs
- ▶ **kill -l (Page 69)**

Question No: 54 (Marks: 1) - Please choose one

The time it takes for the dispatcher to stop one process and start another running is known as the-----.

- ▶ **Dispatch latency (Page 82)**
- ▶ Scheduling
- ▶ Context switching
- ▶ None of the given options

Question No: 55 (Marks: 1) - Please choose one

First-Come, First-Served (FCFS) is a -----scheduling algorithm.

- ▶ preemptive
- ▶ **non-preemptive (Page 83)**
- ▶ both preemptive and non- preemptive
- ▶ none of the given options

Question No: 56 (Marks: 1) - Please choose one

The Shortest-Job-First Scheduling algorithm can be

- ▶ Preemptive only
- ▶ non-preemptive only
- ▶ **preemptive or non-preemptive.** (Page 85)
- ▶ None of the given options

بہترین تجربہ وہ ہے جس سے نصیحت حاصل ہو

Question No: 57 (Marks: 1) - Please choose one
Preemptive -----scheduling is sometimes called shortestremaining-time-first scheduling.

- ▶ First-Come-First-Served (FCFS)
- ▶ Round-Robin
- ▶ **Sorted Job First (SJF)** (Page 85)
- ▶ Priority

Question No: 58 (Marks: 1) - Please choose one
OS helps manages the following except

- ▶ Application software
- ▶ **Bus speed of the system** [Click here for detail](#)
- ▶ Memory
- ▶ Virtual memory

Question No: 59 (Marks: 1) - Please choose one
A parent process calling _____ system call will be suspended until children process terminates.

- ▶ **wait** [click here for detail](#)
- ▶ fork
- ▶ exit
- ▶ exec

Question No: 60 (Marks: 1) - Please choose one
n-process critical section problem can be solved by using

- ▶ **The bakery algorithm** (Page 105)
- ▶ Deterministic modeling
- ▶ Analytic evaluation
- ▶ None of above

Question No: 61 (Marks: 1) - Please choose one
_____ is a piece of code in a cooperating process in which the process may updates shared data (variable, file, database, etc.).

- ▶ Critical analysis
- ▶ **Critical section (Page 100)**
- ▶ Critical path
- ▶ Critical code

خوبصورتی علم و الہ سے ہوتی ہے لباس و حسن سے نہیں

Question No: 62 (Marks: 1) - Please choose one

Round Robin algorithm is similar to _____ scheduling but preemption is added to switch between processes.

- ▶ Shortest job first
- ▶ Shortest Remaining Time First
- ▶ **First Come First Server (Page 88)**
- ▶ None of these

Question No: 63 (Marks: 1) - Please choose one

DOS is single user operating system.

- ▶ **True (Page 7)**
- ▶ False

Question No: 64 (Marks: 1) - Please choose one

When process opens its first file explicitly it will get descriptor number _____

- ▶ 1
- ▶ 2
- ▶ 3 [Click here for detail](#)
- ▶ 4

Question No: 65 (Marks: 1) - Please choose one

A major problem with priority scheduling algorithms is _____.

- ▶ Deadlock
- ▶ Aging
- ▶ **Starvation (Page 86)**
- ▶ None of the these

Question No: 66 (Marks: 1) - Please choose one

All threads within a process share the _____ address space.

- ▶ Same
- ▶ **Different (Page 71)**

Question No: 67 (Marks: 1) - Please choose one

_____ displays information about the top processes.

- ▶ Is
- ▶ Cs
- ▶ **Top (Page 67)**
- ▶ Cd

Question No: 68 (Marks: 1) - Please choose one

The scheduling of _____ are done by the operating system.

- ▶ **Kernel threads (Page 73)**
- ▶ User level threads
- ▶ Both kernel and user level thread
- ▶ None of the give option

Question No: 69 (Marks: 1) - Please choose one

In Unix/ Linux, by default the standard output file is attached to the _____

- ▶ File
- ▶ **Screen (Page 59)**
- ▶ Printer
- ▶ Scanner

Question No: 70 (Marks: 1) - Please choose one

POSIX is a standard developed by ANSI

- ▶ IEEE (not sure)
- ▶ **ISO**
- ▶ ACM

Question No: 71 (Marks: 1) - Please choose one

_____ is the basis of queuing theory which is branch of mathematics used to analyze systems involving queues and servers.

- ▶ **Little's Formula (Page 96)**
- ▶ Deterministic modeling
- ▶ Queuing Theory
- ▶ Queuing Analysis

Question No: 72 (Marks: 1) - Please choose one

_____ is a solution to the problem of indefinite blockage of low-priority processes.

- ▶ Starvation
- ▶ Deadlock
- ▶ **Aging (Page 87)**

▶ None of the these

Question No: 73 (Marks: 1) - Please choose one

A process consists of _____

▶ One or more threads

▶ Code

▶ Data

▶ **All of the given** [click here for detail](#)

Question No: 74 (Marks: 1) - Please choose one

/usr/X11R6 is used by the X Window System.

▶ **True (Page 27)**

▶ False

Question No: 75 (Marks: 1) - Please choose one

command displays the contents of current working directory.

▶ **Is (Page 28)**

▶ Cs

▶ Mv

Question No: 76 (Marks: 1) - Please choose one

Linux uses _____ directory to store system configuration files.

▶ /bin

▶ /dev

▶ /boot

▶ **/etc (Page 26)**

Question No: 77 (Marks: 1) - Please choose one

If your processor does not have two slots empty in Per Process File Descriptor Table, then your _____ system call will fail.

▶ **Pipe (Page 55)**

▶ read

▶ write

▶ open

Question No: 78 (Marks: 1) - Please choose one

First _____ entries in Per Process File Descriptor Table are used as soon as the process is created.

▶ 1

▶ 2

▶ 3

▶ **4 (Page 54)**

Question No: 79 (Marks: 1) - Please choose one

The number of processes completed per unit time is called _____.

- ▶ Turn around time
- ▶ **Throughput (Page 83)**
- ▶ Response time
- ▶ Dispatch latency

Question No: 80 (Marks: 1) - Please choose one

The procedure “The time at which the process finished working MINUS the arrival time of the process MINUS CPU burst for that process” will help calculate the _____.

- ▶ on-preemptive Shortest Job First scheduling.
- ▶ **Preemptive Shortest Job First scheduling. (Page 85)**
- ▶ FCFS
- ▶ RR Scheduling

Question No: 81 (Marks: 1) - Please choose one

/opt is used for storage of large applications.

- ▶ **True (Page 27)**
- ▶ False

Question No: 82 (Marks: 1) - Please choose one

_____ is a virtual directory in Linux and Unix.

- ▶ **/proc (Page 27)**
- ▶ /temp
- ▶ /ver
- ▶ /boot

Question No: 83 (Marks: 1) - Please choose one

The Home Directory for superuser in Linux and Unix is

- ▶ /home
- ▶ **/root (Page 27)**
- ▶ None of the given

Question No: 84 (Marks: 1) - Please choose one

Linux Treats Devices as Files.

- ▶ **True (Page 26)**
- ▶ False

Question No: 85 (Marks: 1) - Please choose one

An absolute pathname starts with the root directory (/) and a relative pathname starts with your home directory.

- ▶ **True (Page 25)**

Muhammad Moaaz Siddiq – MCS(4th)

Moaaz.pk@gmail.com

**Campus:- Institute of E-Learning & Modern Studies
(IEMS) Samundari**

▶ False

Question No: 86 (Marks: 1) - Please choose one

A pathname is the list of directories separated by _____.

▶ #

▶ \$

▶ &

▶ / (Page 25)

Question No: 87 (Marks: 1) - Please choose one

_____ determines How to do something.

▶ **Mechanism (Page 24)**

▶ Policy

▶ Mechanism and Policy:

▶ None of the given

Question No: 88 (Marks: 1) - Please choose one

User Goal of OS is that It easy to use, reliable, safe and fast.

▶ **True (Page 24)**

▶ False

Question No: 89 (Marks: 1) - Please choose one

We can install and run multiple OS by using VMWare.

▶ **True [click here for detail](#)**

▶ False

Question No: 90 (Marks: 1) - Please choose one

Mach, MacOS X Server, QNX, OS/2 and Windows NT are examples of OS Based on _____.

▶ Layered

▶ **Micro Kernal (Page 22)**

▶ Virtual Machine

▶ None of The Given

Question No: 91 (Marks: 1) - Please choose one

In Layered Approach of OS, the Layer highest Layer is User Interface layer.

▶ **True (Page 21)**

▶ False

Question No: 92 (Marks: 1) - Please choose one

In Layered approach of OS, Lowest Layer is known as _____.

▶ Software Layer

▶ **Hardware Layer (Page 21)**

- ▶ Lower Level Layer
- ▶ None of The Given

Question No: 93 (Marks: 1) - Please choose one

Operating System is the Manager of Hardware Resources.

- ▶ **True (Page 6)**
- ▶ False

Question No: 94 (Marks: 1) - Please choose one

An operating system is a control program that manages the execution of user programs to prevent errors and improper use of a computer.

- ▶ **True (Page 6)**
- ▶ False

Question No: 95 (Marks: 1) - Please choose one

The bottom-up view is that operating system is a resource manager who manages the hardware and software resources in the computer system.

- ▶ **True (Page 6)**
- ▶ False

Question No: 96 (Marks: 1) - Please choose one

_____ determines What will be done.

- ▶ Mechanism
- ▶ **Policy (Page 24)**
- ▶ Mechanism and Policy
- ▶ None of the given

Question No: 97 (Marks: 1) - Please choose one

copy file1 file2 is an example of _____ OS view.

- ▶ **Top down (Page 6)**
- ▶ Bottum Up

Question No: 98 (Marks: 1) - Please choose one

The Top-down view is that it is a program that acts as an intermediary between a user of a computer and the computer hardware, and makes the computer system convenient to use.

- ▶ **True (Page 6)**
- ▶ False

Question No: 99 (Marks: 1) - Please choose one

Managing Secondary Storage Involves all of the Following except

- ▶ Allocating storage space
- ▶ Deallocating Storage
- ▶ **Prevent Overwriting (Page 5)**
- ▶ Insure integrity of shared data

تم اچھا کرو زمانہ تم کو برا سمجھے یہ اس سے بہتر ہے کہ تم برا کرو اور زمانہ تم کو اچھا سمجھے

Question No: 100 (Marks: 1) - Please choose one

The Purpose of Operating System is to generate Executable Programs and to _____ them.

- ▶ Regenstrate
- ▶ **Execute (Page 5)**
- ▶ Store
- ▶ Remove

Question No: 101 (Marks: 1) - Please choose one

Users are the People, machines or computers that uses the Hardware resources.

- ▶ **True (Page 4)**
- ▶ False

Question No: 102 (Marks: 1) - Please choose one

Database, Compiler, Video games are examples of _____.

- ▶ Hardware
- ▶ **Application (Page 4)**
- ▶ Operating System
- ▶ Users

Question No: 103 (Marks: 1) - Please choose one

Which of the Following is not an Operating System.

- ▶ Linux
- ▶ Unix
- ▶ Windows Xp
- ▶ **Datebase (Page 7)**

Question No: 104 (Marks: 1) - Please choose one

Operating system enables the user to use the Hardware Resources.

- ▶ **True (Page 4)**
- ▶ False

Question No: 105 (Marks: 1) - Please choose one

Which of the following is NOT a Hardware Resource.

- ▶ CPU
- ▶ **OS (Page 4)**
- ▶ I/O Devices
- ▶ Memory

Question No: 106 (Marks: 1) - Please choose one

Hardware provide basic computing resource.

- ▶ **True (Page 4)**
- ▶ False

Question No: 107 (Marks: 1) - Please choose one

The priorities of processes in the _____ group remain fixed.

- ▶ **Kernel (Page 93)**
- ▶ User

Question No: 108 (Marks: 1) - Please choose one

The process of switching from one process to another is called latency.

- ▶ True
- ▶ **False (Page 34)**

Question No: 109 (Marks: 1) - Please choose one

In Unix/ Linux, by default the standard input file is attached to the _____

- ▶ Mouse
- ▶ **Keyboard (Page 55)**
- ▶ Light pen
- ▶ Joystick

Question No: 110 (Marks: 1) - Please choose one

The nice value helps in assigning _____ to a process.

- ▶ **Priority (Page 94)**
- ▶ Weight
- ▶ Time
- ▶ Scheduling

Question No: 111 (Marks: 1) - Please choose one

You can use the rm file1 command to _____ file1

- ▶ Retrieve
- ▶ **Remove (Page 30)**
- ▶ Make

Muhammad Moaaz Siddiq – MCS(4th)

Moaaz.pk@gmail.com

**Campus:- Institute of E-Learning & Moderen Studies
(IEMS) Samundari**

▶ modify

Question No: 112 (Marks: 1) - Please choose one

The correct command for compiling C program named program.c in Linux environment is

▶ **gcc program.c -o FirstPrgram (Page 31)**

▶ gcc -o FirstProgram program.c

▶ gcc -z FirstProgram program.c

▶ gcc program.c -m FirstPrgram

Question No: 113 (Marks: 1) - Please choose one

Using _____ system, we can create a new process in Linux.

▶ **Fork (Page 39)**

▶ exec

▶ wait

▶ exit

Question No: 114 (Marks: 1) - Please choose one

Cooperating processes never share any data, code, memory or state.

▶ True

▶ **False (Page 5)**

Question No: 115 (Marks: 1) - Please choose one

_____ command display the status of a process.

▶ ls

▶ **ps (Page 66)**

▶ gcc

▶ cat

Question No: 116 (Marks: 1) - Please choose one

Swapper is also termed as Short term scheduler.

▶ True

▶ **False (Page 36)**

Question No: 117 (Marks: 1) - Please choose one

_____ system call is used to write to a file or FIFO or any other IPC channel.

▶ read

▶ **write (Page 48)**

▶ open

▶ fork

Question No: 118 (Marks: 1) - Please choose one

A Process 'A' that has finished working but its parent process has also finished its execution. In this state the process 'A' will be called as _____ process.

- ▶ Child
- ▶ Thread
- ▶ **Zombie (Page 42)**
- ▶ Fork

Question No: 119 (Marks: 1) - Please choose one

_____ scheduling allows a process to move between queues.

- ▶ Round Robin
- ▶ First Come First Serve
- ▶ **Multilevel Feedback Queue (Page 92)**
- ▶ Shortest Remaining Time First

Question No: 120 (Marks: 1) - Please choose one

Round Robin algorithm is most suitable for _____.

Time sharing system (Page 88)

- Real time systems and batch systems
- Running Batch programs
- Expert system

Question No: 121 (Marks: 1) - Please choose one

Kernel is responsible for scheduling the user level threads.

- ▶ True
- ▶ **False (Page 73)**

Question No: 122 (Marks: 1) - Please choose one

A system call_____

- ▶ **Is an entry point into the kernel code (Page 18)**
- ▶ Allows a program to request a kernel service
- ▶ Is a technique to protect I/O devices and other system resources
- ▶ All of the these

Question No: 123 (Marks: 1) - Please choose one

Operating System provides services such as Managing Primary and Secondary Storage, Processes and Allowing user to manage his/her files and directories.

- ▶ **True (Page 5)**
- ▶ False

Question No: 124 (Marks: 1) - Please choose one
_____ is used in real time operating systems.

▶ **Non-preemptive scheduling** [Click here for detail](#)

- ▶ Preemptive scheduling
- ▶ Dispatching scheduling
- ▶ FCFS scheduling

Question 125 (Marks: 1) - Please choose one

Deadlock detection and recovery technique is exactly similar to deadlock avoidance technique to handle deadlock in the system.

▶ **True**

- ▶ False

Question 126 (Marks: 1) - Please choose one

In Overlay technique, we can overload any part of the program with the part of the program required needed recently.

▶ True

▶ **False**

Question 127 (Marks: 1) - Please choose one

_____ is the process of mapping a name to an address.

- ▶ Addressing
- ▶ **Binding**
- ▶ Routing
- ▶ Memory

جھوٹ رزق کو کہا جاتا ہے

اس سے پہلے کہ تمہیں شہوت فتنے میں ڈالے نکاح کرلو

انسان کے لئے بری صحبت سے بڑھ کر بری کوئی چیز نہیں



Operating Systems

Lecture No. 18 and 19

Reading Material

- Chapter 7 of the textbook
- Lectures 18 and 19 on Virtual TV

Summary

- Process Synchronization: the basic concept
- The Critical Section Problem
- Solutions for the Critical Section Problem
- 2-Process Critical Section Problem solutions

Process Synchronization

Concurrent processes or threads often need access to shared data and shared resources. If there is no controlled access to shared data, it is often possible to obtain an inconsistent state of this data. Maintaining data consistency requires mechanisms to ensure the orderly execution of cooperating processes, and hence various process synchronization methods are used. In the producer-consumer problem that was discussed earlier, the version only allows one item less than the buffer size to be stored, to provide a solution for the buffer to use its entire capacity of N items is not simple. The producer and consumer share data structure 'buffer' and use other variables shown below:

```
#define BUFFER_SIZE 10
typedef struct
{
    ...
} item;
item buffer[BUFFER_SIZE];
int in=0;
int out=0;
```

The code for the producer process is:

```
while(1)
{
    /*Produce an item in nextProduced*/
    while(counter == BUFFER_SIZE); /*do nothing*/
    buffer[in]=nextProduced;
    in=(in+1)%BUFFER_SIZE;
    counter++;
}
```

The code for the consumer process is:

```
while(1)
{
    while(counter==0); //do nothing
    nextConsumed=buffer[out];
    out=(out+1)%BUFFER_SIZE;
    counter--;
    /*Consume the item in nextConsumed*/
}
```

Both producer and consumer routines may not execute properly if executed concurrently. Suppose that the value of the counter is 5, and that both the producer and the consumer execute the statement `counter++` and `counter--` concurrently. Following the execution of these statements the value of the counter may be 4,5, or 6! The only correct result of these statements should be `counter=5`, which is generated if the consumer and the producer execute separately. Suppose `counter++` is implemented in machine code as the following instructions:

```
MOV R1, counter
INC R1
MOV counter, R1
```

whereas `counter--` maybe implemented as:

```
MOV R2, counter
DEC R2
MOV counter, R2
```

If both the producer and consumer attempt to update the buffer concurrently, the machine language statements may get interleaved. Interleaving depends upon how the producer and consumer processes are scheduled. Assume counter is initially 5. One interleaving of statements is:

```
producer: MOV R1, counter    (R1 = 5)
           INC R1            (R1 = 6)
consumer: MOV R2, counter    (R2 = 5)
           DEC R2            (R2 = 4)
producer: MOV counter, R1    (counter = 6)
consumer: MOV counter, R2    (counter = 4)
```

The value of `count` will be 4, where the correct result should be 5. The value of `count` could also be 6 if producer executes `MOV counter, R1` at the end. The reason for this state is that we allowed both processes to manipulate the variable `counter` concurrently. A situation like this, where several processes access and manipulate the same data concurrently and the outcome of the manipulation depends on the particular order in which the access takes place, is called a **race condition**. To guard against such race conditions, we require synchronization of processes.

Concurrent transactions in a bank or in an airline reservation (or travel agent) office are a couple of other examples that illustrates the critical section problem. We show

interleaving of two bank transactions, a deposit and a withdrawal. Here are the details of the transactions:

- Current balance = Rs. 50,000
- Check deposited = Rs. 10,000
- ATM withdrawn = Rs. 5,000

The codes for deposit and withdrawal are shown in Figure 18.1.

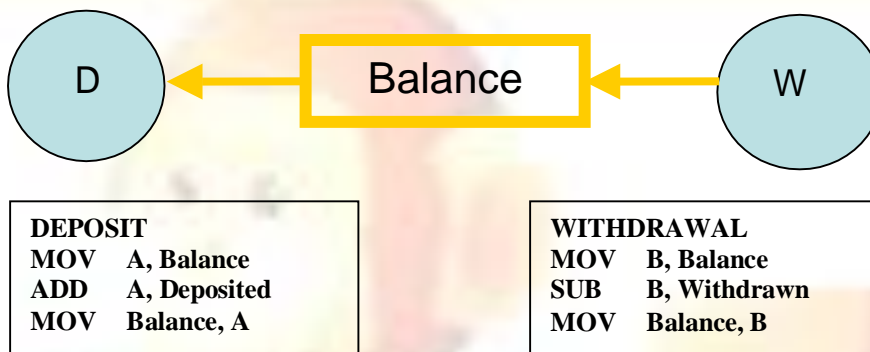


Figure 18.1 Bank transactions—deposit and withdrawal

Here is what may happen if the two transactions are allowed to execute concurrently, i.e., the transactions are allowed to interleave. Note that in this case the final balance will be Rs. 45,000, i.e., a loss of Rs. 5,000. If MOV Balance, A executes at the end, the result will be a gain of Rs. 5,000. In both cases, the final result is wrong.

Check Deposit:

```

    MOV A, Balance      // A = 50,000
    ADD A, Deposited    // A = 60,000
  
```

ATM Withdrawal:

```

    MOV B, Balance      // B = 50,000
    SUB B, Withdrawn    // B = 45,000
  
```

Check Deposit:

```

    MOV Balance, A      // Balance = 60,000
  
```

ATM Withdrawal:

```

    MOV Balance, B      // Balance = 45,000
  
```

The Critical Section Problem

Critical Section: A piece of code in a cooperating process in which the process may updates shared data (variable, file, database, etc.).

Critical Section Problem: Serialize executions of critical sections in cooperating processes.

When a process executes code that manipulates shared data (or resource), we say that the process is in its critical section (for that shared data). The execution of critical sections must be mutually exclusive: at any time, only one process is allowed to execute in its critical section (even with multiple processors). So each process must first request permission to enter its critical section. The section of code implementing this request is

called the **entry section**. The remaining code is the **remainder section**. The critical section problem is to design a protocol that the processes can use so that their action will not depend on the order in which their execution is interleaved (possibly on many processors).

There can be three kinds of solution to the critical section problem:

- Software based solutions
- Hardware based solutions
- Operating system based solution

We discuss the software solutions first. Regardless of the type of solution, the structure of the solution should be as follows. The Entry and Exit sections comprise solution for the problem.

```
do
{
    Entry section
    critical section
    Exit section
    remainder section
} while(1)
```

Solution to the Critical Section Problem

A solution to the critical section problem must satisfy the following three requirements:

1. Mutual Exclusion

If process P_i is executing in its critical section, then no other process can be executing in their critical section.

2. Progress

If no process is executing in its critical section and some processes wish to enter their critical sections, then only those processes that are not executing in their remainder section can participate in the decision on which will enter its critical section next, and this selection cannot be postponed indefinitely.

3. Bounded Waiting

There exists a bound on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

Assumptions

While formulating a solution, we must keep the following assumptions in mind:

- Assume that each process executes at a nonzero speed
- No assumption can be made regarding the relative speeds of the N processes.

2-Process Solutions to the Critical Section Problem

In this section algorithms that are applicable to two processes will be discussed. The processes are P_0 and P_1 . When presenting P_1 , we use P_j to denote the other process. An assumption is that the basic machine language instructions such as load and store are executed atomically, that is an operation that completes in its entirety without interruption.

Algorithm 1

The first approach is to let the processes share a common integer variable **turn** initialized to 0 or 1. If $\text{turn} = i$, then process P_i is allowed to execute in its critical section. The structure of the process P_i is as follows:

```
do
{
    while(turn!=j);

    critical section

    turn=j;

    remainder section
} while(1)
```

This solution ensures mutual exclusion, that is only one process at a time can be in its critical section. However it does not satisfy the progress requirement, since it requires strict alternation of processes in the execution of the critical section. For example, if $\text{turn} = 0$ and P_1 is ready to enter its critical section, P_1 cannot do so even though P_0 may be in its remainder section. The bounded wait condition is satisfied though, because there is an alternation between the turns of the two processes.

Algorithm 2

In algorithm two, the variable **turn** is replaced with an array `boolean flag[2]` whose elements are initialized to false. If `flag` is true for a process that indicates that the process is ready to enter its critical section. The structure of process P_i is shown:

```
do
{
    flag[i]=true;
    while(flag[j]);

    critical section

    flag[i]=false;

    remainder section
} while(1)
```

In this algorithm P_i sets $flag[i] = true$ signaling that it is ready to enter its critical section. Then P_i checks to verify that process P_j is not also ready to enter its critical section. If P_j were ready, then P_i would wait until P_j had indicated that it no longer needed to be in the critical section (that is until $flag[j] = false$). At this point P_i would enter the critical section. On exiting the critical section, P_i would set $flag[i] = false$ allowing the other process to enter its critical section. In this solution, the mutual exclusion requirement is satisfied. Unfortunately the progress condition is not met; consider the following execution sequence:

T_0 : P_0 sets $flag[0] = true$

T_1 : P_1 sets $flag[1] = true$

Now both the processes are looping forever in their respective while statements.

Operating Systems

Lecture No. 20

Reading Material

- Chapter 7 of the textbook
- Lecture 20 on Virtual TV

Summary

- 2-Process Critical Section Problem (continued)
- n-Process Critical Section Problem
- The Bakery Algorithm

2-Process Critical Section Problem (continued)

We discussed two solutions for the 2-process critical section problem in lecture 19 but both were not acceptable because they did not satisfy the progress condition. Here is a good solution for the critical section problem that satisfies all three requirements of a good solution.

Algorithm 3

The processes share two variables:

```
boolean flag[2];  
int turn;
```

The boolean array of 'flag' is initialized to false, whereas 'turn' maybe 0 or 1. The structure of the process is as follows:

```
do  
{  
    flag[i]=true;  
    turn=j;  
    while(flag[j] && turn==j);  
    critical section  
    flag[i]=false;  
    remainder section  
} while(1)
```

To enter its critical section, P_i sets $flag[i]$ to true, and sets 'turn' to j , asserting that if the other process wishes to enter its critical section, it may do so. If both try to enter at the

same time, they will attempt to set 'turn' to i and j . However, only one of these assignments will last, the other will occur but be overwritten instantly. Hence, the eventual value of 'turn' will decide which process gets to enter its critical section.

To prove mutual exclusion, note that P_i enters its critical section only if either $\text{flag}[j]=\text{false}$ or $\text{turn}=i$. Also, if both processes were executing in their critical sections at the same time, then $\text{flag}[0]=\text{flag}[1]=\text{true}$. These two observations suggest that P_0 and P_1 could not have found both conditions in the while statement true at the same time, since the value of 'turn' can either be 0 or 1. Hence only one process say P_0 must have successfully exited the while statement. Hence mutual exclusion is preserved.

To prove bounded wait and progress requirements, we note that a process P_i can be prevented the critical section only if it is stuck in the while loop with the condition $\text{flag}[j]=\text{true}$ and $\text{turn}=j$. If P_j is not ready to enter the critical section, then $\text{flag}[j]=\text{false}$ and P_i can enter its critical section. If P_j has set $\text{flag}[j]=\text{true}$ and is also executing its while statement then either $\text{turn}=i$ or $\text{turn}=j$. If $\text{turn}=i$ then P_i enters its critical section, otherwise P_j . However, whenever a process finishes executing in its critical section, let's assume P_j , it resets $\text{flag}[j]$ to false allowing P_i to enter its critical section. If P_j resets $\text{flag}[j]=\text{true}$, then it must also set 'turn' to i , and since P_i does not change the value of 'turn' while executing in its while statement, P_i will enter its critical section (progress) after at most one entry by P_j (bounded waiting).

N-Process Critical Section Problem

In this section we extend the critical section problem of two processes to include n processes. Consider a system of n processes (P_0, P_1, \dots, P_{n-1}). Each process has a segment of code called a critical section in which the process may be changing common variables, updating a table, writing a file and so on. The important feature of the system is that, when one process enters its critical section, no other process is allowed to execute in its critical section. Thus the execution of critical sections by the processes is mutually exclusive in time. The critical section problem is to design a protocol to serialize executions of critical sections. Each process must request permission to enter its critical section. Many solutions are available in the literature to solve the N-process critical section problem. We will discuss a simple and elegant solution, known as the Bakery algorithm.

The Bakery Algorithm

The bakery algorithm is due to Leslie Lamport and is based on a scheduling algorithm commonly used in bakeries, ice-cream stores, and other locations where order must be made out of chaos. On entering the store, each customer receives a number. The customer with the lowest number is served next. Before entering its critical section, process receives a ticket number. Holder of the smallest ticket number enters its critical section. Unfortunately, the bakery algorithm cannot guarantee that two processes (customers) will not receive the same number. In the case of a tie, the process with the lowest ID is served first. If processes P_i and P_j receive the same number, if $i < j$, then P_i is served first; else P_j is served first. The ticket numbering scheme always generates numbers in the increasing order of enumeration; i.e., 1, 2, 3, 4, 5 ...

Since process names are unique and totally ordered, our algorithm is completely deterministic. The common data structures are:

```
boolean choosing [n];  
int number[n];
```

Initially these data structures are initialized to false and 0, respectively. The following notation is defined for convenience:

- (ticket #, process id #)
- $(a,b) < (c,d)$ if $a < c$ or if $a = c$ and $b < d$.
- $\max(a_0, \dots, a_{n-1})$ is a number, k , such that $k \geq a_i$ for $i=0, \dots, n-1$

The structure of process P_i used in the bakery algorithm is as follows:

```
do  
{  
  choosing[i] = true;  
  number[i] = max(number[0], number[1], .. number[n-1])+1;  
  choosing[i] = false;  
  
  for(j=0; j<n; j++) {  
    while(choosing[j]);  
    while((number[j]!=0) && ((number[j],j) < (number[i],i)));  
  }  
  
  Critical section  
  
  number[i]=0;  
  
  Remainder section  
  
} while(1);
```

To prove that the bakery algorithm is correct, we need to first show that if P_i is in its critical section and P_k has already chosen its number $k \neq 0$, then $((number [i],i) < (number[k],k))$. Consider P_i in its critical section and P_k trying to enter its critical section. When process P_k executes the second while statement for $j = i$ it finds that,

- $number[i] \neq 0$
- $(number[i],i) < (number[k],k)$

Thus it keeps looping in the while statement until P_i leaves the P_i critical section. Hence mutual exclusion is preserved. For progress and bounded wait we observe that the processes enter their critical section on a first come first serve basis.

Following is an example of how the Bakery algorithm works. In the first table, we show that there are five processes, P_0 through P_4 . P_1 's number is 0 because it is not interested in getting into its critical section at this time. All other processes are interested in entering their critical sections and have chosen non-zero numbers by using the $\max()$ function in their entry sections.

| Process | Number |
|---------|--------|
| P0 | 3 |
| P1 | 0 |
| P2 | 7 |
| P3 | 4 |
| P4 | 8 |

The following table shows the status of all the processes as they execute the ‘for’ loops in their entry sections. The gray cells show processes waiting in the second while loops in their entry sections. The table shows that P0 never waits for any process and is, therefore, the first process to enter its critical section, while all other processes wait in their second while loops for $j = 0$, indicating that they are waiting for P0 to get out of its critical section and then they would make progress (i.e., they will get out the while loop, increment j by one, and continue their execution).

You can make the following observations by following the Bakery algorithm closely with the help of this table:

- P1 not interested to get into its critical section \Rightarrow number[1] is 0
- P2, P3, and P4 wait for P0
- P0 gets into its CS, get out, and sets its number to 0
- P3 get into its CS and P2 and P4 wait for it to get out of its CS
- P2 gets into its CS and P4 waits for it to get out
- P4 gets into its CS
- Sequence of execution of processes: <P0, P3, P2, P4>

| j | P0 | P2 | P3 | P4 |
|---|---------------|---------------|---------------|---------------|
| 0 | (3,0) < (3,0) | (3,0) < (7,2) | (3,0) < (4,3) | (3,0) < (8,4) |
| 1 | Number[1] = 0 | Number[1] = 0 | Number[1] = 0 | Number[1] = 0 |
| 2 | (7,2) < (3,0) | (7,2) < (7,2) | (7,2) < (4,3) | (7,2) < (8,4) |
| 3 | (4,3) < (3,0) | (4,3) < (7,2) | (4,3) < (4,3) | (4,3) < (8,4) |
| 4 | (8,4) < (3,0) | (8,4) < (7,2) | (8,4) < (4,3) | (8,4) < (8,4) |

Operating Systems

Lecture No. 21

Reading Material

- Chapter 7 of the textbook
- Lecture 21 on Virtual TV

Summary

- Hardware solutions

Hardware Solutions for the Critical Section Problem

In this section, we discuss some simple hardware (CPU) instructions that can be used to provide synchronization between processes and are available on many systems.

The critical section problem can be solved simply in a uniprocessor environment if we could forbid interrupts to occur while a shared variable is being modified. In this manner, we could be sure that the current sequence of instructions would be run, so no unexpected modifications could be made to the shared variable.

Unfortunately this solution is not feasible in a multiprocessing environment, as disabling interrupts can be time consuming as the message is passed to all processors. This message passing delays entry into each critical section, and system efficiency decreases.

Normally, access to a memory location excludes other accesses to that same location. Designers have proposed machine instructions that perform two operations atomically (indivisibly) on the same memory location (e.g., reading and writing). The execution of such an instruction is also mutually exclusive (even on Multiprocessors). They can be used to provide mutual exclusion but other mechanisms are needed to satisfy the other two requirements of a good solution to the critical section problem.

We can use these special instructions to solve the critical section problem. These instructions are TestAndSet (also known as TestAndSetLock; TSL) and Swap. The semantics of the TestAndSet instruction are as follows:

```
boolean TestAndSet (Boolean &target)
{
    boolean rv=target;
    target=true;
    return rv;
}
```

The semantics simply say that the instruction saves the current value of 'target', set it to true, and returns the saved value.

The important characteristic is that this instruction is executed atomically. Thus if two TestAndSet instructions are executed simultaneously, they will be executed sequentially in some arbitrary order.

If the machine supports TestAndSet instruction, then we can implement mutual exclusion by declaring a Boolean variable lock, initialized to false. The structure of process P_i becomes:

```
do
{
    while (TestAndSet(lock)) ;
        Critical section
    lock=false;
        Remainder section
} while(1);
```

The above TSL-based solution is no good because even though mutual exclusion and progress are satisfied, bounded waiting is not.

The semantics of the Swap instruction, another atomic instruction, are, as expected, as follows:

```
boolean Swap(boolean &a, boolean &b)
{
    boolean temp=a;
    a=b;
    b=temp;
}
```

If the machine supports the Swap instruction, mutual exclusion can be implemented as follows. A global Boolean variable lock is declared and is initialized to false. In addition each process also has a local Boolean variable key. The structure of process P_i is:

```
do
{
    key=true;
    while(key == true)
        Swap(lock, key);
        Critical section
    lock=false;
        Remainder section
} while(1);
```

Just like the TSL-based solution shown in this section, the above Swap-based solution is not good because even though mutual exclusion and progress are satisfied, bounded waiting is not. In the next lecture, we will discuss a good solution for the critical section problem by using the hardware instructions.

Operating Systems

Lecture No. 22

Reading Material

- Chapter 7 of the textbook
- Lecture 22 on Virtual TV

Summary

- Hardware based solutions
- Semaphores
- Semaphore based solutions for the critical section problem

Hardware Solutions

In lecture 21 we started discussing the hardware solutions for the critical section problem. We discussed two possible solutions but realized that whereas both solutions satisfied the mutual exclusion and bounded waiting conditions, neither satisfied the progress condition. We now describe a solution that satisfies all three requirements of a solution to the critical section problem.

Algorithm 3

In this algorithm, we combine the ideas of the first two algorithms. The common data structures used by a cooperating process are:

```
boolean waiting[n];
boolean lock;
```

The structure of process P_i is:

```
do
{
    waiting[i] = true;
    key = true;
    while (waiting[i] && key)
        key = TestAndSet(lock);
    waiting[i] = false;

    Critical section

    j = (i+1) % n;
    while ((j!=i) && !waiting[j])
        j = (j+1) % n;
    if (j == i)
        lock = false;
    else
        waiting[j] = false;

    Remainder section
} while(1);
```

These data structures are initialized to false. To prove that the mutual exclusion requirement is met, we note that process P_i can enter its critical section only if either $\text{waiting}[i] = \text{false}$ or $\text{key} = \text{false}$. The value of key can become false only if TestAndSet is executed. The first process to execute the TestAndSet instruction will find $\text{key} = \text{false}$; all others must wait. The variable $\text{waiting}[i]$ can only become false if another process leaves its critical section; only one $\text{waiting}[i]$ is set to false, maintaining the mutual exclusion requirement.

To prove the progress requirement is met, we note that the arguments presented for mutual exclusion also apply here, since a process exiting the critical section either sets lock to false or sets $\text{waiting}[j]$ to false. Both allow a process that is waiting to enter its critical section to proceed.

To prove that the bounded waiting requirement is met, we note that, when a process leaves its critical section, it scans the array waiting in the cyclic ordering $(i+1, i+2, \dots, n-1, 0, 1, \dots, i-1)$. It designates the first process it sees that is in its entry section with $\text{waiting}[j] = \text{true}$ as the next one to enter its critical section. Any process waiting to do so will enter its critical section within $n-1$ turns.

Semaphores

Hardware solutions to synchronization problems are not easy to generalize to more complex problems. To overcome this difficulty we can use a synchronization tool called a semaphore. A **semaphore** S is an integer variable that, apart from initialization is accessible only through two standard atomic operations: wait and signal. These operations were originally termed P (for wait) and V (for signal). The classical definitions of wait and signal are:

```
wait(S) {  
    while (S <= 0)  
        ; // no op  
    S--;  
}
```

```
signal(S) {  
    S++;  
}
```

Modifications to the integer value of the semaphore in the wait and signal operations must be executed indivisibly. That is, when one process is updating the value of a semaphore, other processes cannot simultaneously modify that same semaphore value. In addition, in the case of the $\text{wait}(S)$, the testing of the integer value of S ($S \leq 0$) and its possible modification ($S--$) must also be executed without interruption.

We can use semaphores to deal with the n -process critical section problem. The n processes share a semaphore, **mutex** (standing for mutual exclusion) initialized to 1. Each process P_i is organized as follows:

```
do
{
    wait(mutex);
        Critical section
    signal(mutex);
        Remainder section
} while(1);
```

As was the case with the hardware-based solutions, this is not a good solution because even though it satisfies mutual exclusion and progress, it does not satisfy bounded wait.

In a uni-processor environment, to ensure atomic execution, while executing wait and signal, interrupts can be disabled. In case of a multi-processor environment, to ensure atomic execution is one can lock the data bus, or use a soft solution such as the Bakery algorithm.

The main disadvantage of the semaphore discussed in the previous section is that it requires **busy waiting**. While a process is in its critical section, any other process that tries to enter its critical section must loop continuously in the entry code. This continual looping is clearly a problem in a real multiprogramming system, where a single CPU is shared among many processes. Busy waiting wastes CPU cycles that some other process may be able to use productively. This type of semaphore is also called a **spinlock** (because the process spins while waiting for the lock). Spinlocks are useful in multiprocessor systems. The advantage of a spinlock is that no context switch is required when a process must wait on a lock, and a context switch may take considerable time. This is, spinlocks are useful when they are expected to be held for short times. The definition of semaphore should be modified to eliminate busy waiting. We will discuss the modified definition of semaphore in the next lecture.

Operating Systems

Lecture No. 23

Reading Material

- Chapter 7 of the textbook
- Lecture 23 on Virtual TV

Summary

- Busy waiting
- New definition of semaphore
- Process synchronization
- Problems with the use of semaphore: deadlock, starvation, and violation of mutual exclusion

Semaphores

The main disadvantage of the semaphore discussed in the previous section is that they all require **busy waiting**. While a process is in its critical section, any other process that tries to enter its critical section must loop continuously in the entry code. This continual looping is clearly a problem in a real multiprogramming system, where a single CPU is shared among many processes. Busy waiting wastes CPU cycles that some other process may be able to use productively. This type of semaphore is also called a **spinlock** (because the process spins while waiting for the lock). Spinlocks are useful in multiprocessor systems. The advantage of a spinlock is that no context switch is required when a process must wait on a lock, and a context switch may take considerable time. This, when locks are expected to be held for short times, spinlocks are useful.

To overcome the need for busy waiting, we can modify the definition of semaphore and the wait and signal operations on it. When a process executes the wait operation and finds that the semaphore value is not positive, it must wait. However, rather than busy waiting, the process can block itself. The block operation places a process into a waiting queue associated with the semaphore, and the state of the process is switched to the waiting state. Then, control is transferred to the CPU scheduler, which selects another process to execute.

A process that is blocked, waiting on a semaphore *S*, should be restarted when some other process executes a signal operation. The process is restarted by a wakeup operation, which changes the process from the waiting state to the ready state. The process is then placed in the ready queue. (The CPU may or may not be switched from the running process to the newly ready process, depending on the CPU scheduling algorithm.)

Such an implementation of a semaphore is as follows:

```
typedef struct {
    int value;
    struct process *L;
} semaphore;
```

Each semaphore has an integer value and a list of processes. When a process must wait on a semaphore; it is added to the list of processes. A signal operation removes one process from the list of the waiting processes and awakens that process. The wait operation can be defined as:

```
void wait(semaphore S) {
    S.value--;
    if(S.value < 0) {
        add this process to S.L;
        block();
    }
}
```

The signal semaphore operation can be defined as

```
void signal wait(semaphore S) {
    S.value++;
    if(S.value <= 0) {
        remove a process P from S.L;
        wakeup(P);
    }
}
```

The block operation suspends the process that invokes it. The wakeup(P) operation resumes the execution of a blocked process P. These two operations are provided by the operating system as basic system calls. The negative value of S.value indicates the number of processes waiting for the semaphore. A pointer in the PCB needed to maintain a queue of processes waiting for a semaphore. As mentioned before, the busy-waiting version is better when critical sections are small and queue-waiting version is better for long critical sections (when waiting is for longer periods of time).

Process Synchronization

You can use semaphores to synchronize cooperating processes. Consider, for example, that you want to execute statement B in P_j only after statement A has been executed in P_i. You can solve this problem by using a semaphore S initialized to 0 and structuring the codes for P_i and P_j as follows:

| | |
|----------------------|----------------------|
| P_i | P_j |
| ... | ... |
| A; | wait(S); |
| signal(S); | B; |
| ... | ... |

P_j will not be able to execute statement B until P_i has executed its statements A and signal(S).

Here is another synchronization problem that can be solved easily using semaphores. We want to ensure that statement S₁ in P₁ executes only after statement S₂ in P₂ has

executed, and statement S2 in P2 should execute only after statement S3 in P3 has executed. One possible semaphore-based solution uses two semaphores, A and B. Here is the solution.

```

semaphore A=0, B=0;

P1          P2          P3
...          ...          ...
wait(A);    wait(B);    S3;
S1;         S2;         signal(B);
           signal(A);
...          ...          ...

```

Problems with Semaphores

Here are some key points about the use of semaphores:

- Semaphores provide a powerful tool for enforcing mutual exclusion and coordinating processes.
- The wait(S) and signal(S) operations are scattered among several processes. Hence, it is difficult to understand their effects.
- Usage of semaphores must be correct in all the processes.
- One bad (or malicious) process can fail the entire system of cooperating processes.

Incorrect use of semaphores can cause serious problems. We now discuss a few of these problems.

Deadlocks and Starvation

A set of processes are said to be in a deadlock state if every process is waiting for an event that can be caused only by another process in the set. Here are a couple of examples of deadlocks in our daily lives.

- Traffic deadlocks
- One-way bridge-crossing

Starvation is infinite blocking caused due to unavailability of resources. Here is an example of a deadlock.

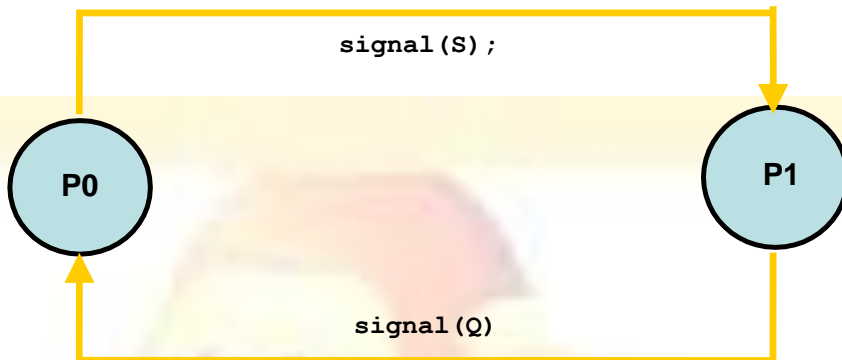
```

P0          P1
wait(S);    wait(Q);
wait(Q);    wait(S);
...         ...
signal(S);  signal(Q);
signal(Q);  signal(S);
...         ...

```

P0 and P1 need to get two semaphores, S and Q, before executing their critical sections. The following code structures can cause a deadlock involving P0 and P1. In this example, P0 grabs semaphore S and P1 obtains semaphore Q. Then, P0 waits for Q and P1 waits for S. P0 waits for P1 to execute signal(Q) and P1 waits for P0 to execute signal(S).

Neither process will execute the respective instruction—a typical deadlock situation. The following diagram shows the situation pictorially.



Here is an example of starvation. The code structures are self-explanatory.

| | |
|-----------|------------|
| P0 | P1 |
| wait(S); | wait(S); |
| ... | ... |
| wait(S); | signal(S); |
| ... | ... |

Violation of Mutual Exclusion

In the following example, the principle of mutual exclusion is violated. Again, the code structures are self-explanatory. If you have any questions about them, please see the lecture video.

| | |
|------------|------------|
| P0 | P1 |
| signal(S); | wait(S); |
| ... | ... |
| wait(S); | signal(S); |
| ... | ... |

These problems are due to programming errors because of the tandem use of the wait and signal operations. The solution to these problems is higher-level language constructs such as critical region (region statement) and monitor. We discuss these constructs and their use to solve the critical section and synchronization problems in the next lecture.

Operating Systems

Lecture No. 24

Reading Material

- Chapter 7 of the textbook
- Lecture 24 on Virtual TV

Summary

- Counting semaphores
- Classical synchronization problems
- Bounded buffer problem
- Readers and writers problem
- Dining philosophers problem

Semaphores

There are two kinds of semaphores:

- **Counting semaphore** whose integer value can range over an unrestricted integer domain.
- **Binary semaphore** whose integer value cannot be > 1 ; can be simpler to implement.

Let S be a counting semaphore. To implement it in terms of binary semaphores we need the following data structures:

```
binary-semaphore S1, S2;  
int C;
```

Initially $S1=1$, $S2=0$, and the value of integer C is set to the initial value of the counting semaphore S . The wait operation on the counting semaphore S can be implemented as follows:

```
wait(S1);  
C--;  
if(C < 0) {  
    signal(S1);  
    wait(S2);  
}  
signal(S1);
```

The signal operation on the counting semaphore S can be implemented as follows:

```
wait(S1);  
C++;  
if(C <= 0)  
    signal(S2);  
else  
    signal(S1);
```

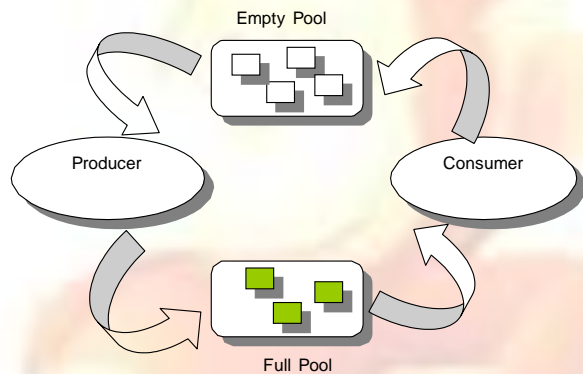
Classic Problems of Synchronization

The three classic problems of synchronization are:

- Bounded-Buffer Problem
- Readers and Writers Problem
- Dining Philosophers Problem

Bounded Buffer Problem

The bounded-buffer problem, which was introduced in a previous lecture, is commonly used to illustrate the power of synchronization primitives. The solution presented in this section assumes that the pool consists of n buffers, each capable of holding one item.



The mutex semaphore provides mutual exclusion for accesses to the buffer pool and is initialized to the value 1. The empty and full semaphores count the number of empty and full buffers, respectively. The semaphore empty is initialized to the value n ; the semaphore full is initialized to the value 0.

The code for the producer is as follows:

```
do {  
    ...  
    produce an item in nextp  
    ...  
    wait(empty);  
    wait(mutex);  
    ...  
    add nextp to buffer  
    ...  
    signal(mutex);  
    signal(full);  
} while(1);
```

And that for the consumer is as follows:

```

do {
    wait(full);
    wait(mutex);
    ...
    remove an item from
    buffer to nextc
    ...
    signal(mutex);
    signal(empty);
    ...
    consume the item in nextc
    ...
} while(1);

```

Note the symmetry between the producer and the consumer process. This code can be interpreted as the producer producing full buffers for the consumer, or as the consumer producing empty buffers for the producer.

Readers Writers Problem



A data object (such as a file or a record) is to be shared among several concurrent processes. Some of these processes, called **readers**, may want only to read the content of the shared object whereas others, called **writers**, may want to update (that is to read and write) the shared object. Obviously, if two readers access the data simultaneously, no adverse effects will result. However, if a writer and some other process (whether a writer or some readers) access the shared object simultaneously, chaos may ensue.

To ensure these difficulties do not arise, we require that the writers have exclusive access to the shared object. This synchronization problem is referred to the readers-writers problem. Since it was originally stated, it has been used to test nearly every new synchronization primitive. The readers-writers problem has several variations, all involving priorities. The simplest one, referred to as the **first readers-writers problem**, requires that no reader will be kept waiting unless a writer has already obtained permission to use the shared object. In other words, no reader should wait for other readers to finish simply because a writer is waiting. The **second readers-writers problem** requires that once a writer is ready, that writer performs its write as soon as

possible. In other words, if a writer is waiting to access the object, no new readers may start reading.

A solution to either problem may result in starvation. In the first case, writers may starve; in the second case, readers may starve. For this reason, other variants of the problem have been proposed. In this section, we discuss a solution to the first readers-writers problem. In the solution to the first readers-writers problem, processes share the following data structures.

```
semaphore mutex, wrt;  
int readcount;
```

The semaphores `mutex` and `wrt` are initialized to 1; `readcount` is initialized to 0. The semaphore `wrt` is common to both the reader and writer processes. The mutex semaphore is used to ensure mutual exclusion when the reader processes update the `readcount` variable. The `readcount` variable keeps track of how many processes are currently reading the object. The `wrt` semaphore is used to ensure mutual exclusion for writers or a writer and readers. This semaphore is also used by the first and last readers to block entry of a writer into its critical section and to allow open access to the `wrt` semaphore, respectively. It is not used by readers who enter or exit, while at least one reader is in its critical sections.

The codes for reader and writer processes are shown below:

```
wait(mutex);  
  readcount++;  
  if(readcount == 1)  
    wait(wrt);  
  signal(mutex);  
  ...  
  reading is performed  
  ...  
wait(mutex);  
  readcount--;  
  if(readcount == 0)  
    signal(wrt);  
  signal(mutex);
```

```
wait(wrt);  
  ...  
  writing is performed  
  ...  
  signal(wrt);
```

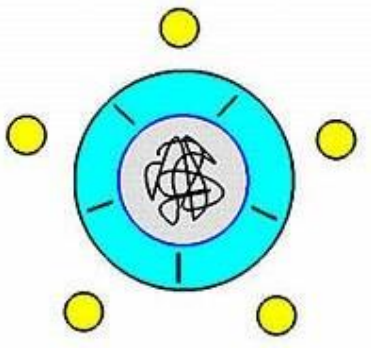
Note that, if a writer is in the critical section and n readers are waiting, then one reader is queued on `wrt`, and $n-1$ readers are queued on `mutex`. Also observe that when a writer executes `signal(wrt)` we may resume the execution of either the waiting readers or a single waiting writer; the selection is made by the CPU scheduler.

Dining Philosophers Problem

Consider five philosophers who spend their lives thinking and eating, as shown in the following diagram.



The philosophers share a common circular table surrounded by five chairs, each belonging to one philosopher. In the center of the table is a bowl of rice, and the table is laid with five single chopsticks.



When a philosopher thinks, she does not interact with her colleagues. From time to time, a philosopher gets hungry and tries to pick up the two chopsticks that are closest to her (the chopsticks that are between her and her left and right neighbors). A philosopher may pick up only one chopstick at a time. Obviously, she cannot pick up a chopstick that is already in the hand of her neighbor. When a hungry philosopher has both her chopsticks at the same time, she eats without releasing her chopsticks. When she is finished eating, she puts down both of her chopsticks and starts thinking again.

The dining philosophers problem is considered to be a classic synchronization problem because it is an example of a large class of concurrency control problems. It is a simple representation of the need to allocate several resources among several processes in a deadlock and starvation free manner.

One simple solution is to represent each chopstick by a semaphore. A philosopher tries to grab the chopstick by executing a wait operation on that semaphore; she releases her chopsticks by executing the signal operation on the appropriate semaphores. Thus the shared data are:

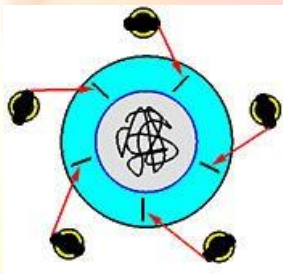
```
semaphore chopstick[5];
```

All the chopsticks are initialized to 1. The structure of philosopher i is as follows:

```
do {
    wait(chopstick[i];
    wait(chopstick[(i+1)%5]);
    ...
    eat
    ...
    signal(chopstick[i]);
    signal(chopstick[(i+1)%5]);
    ...
    think
    ...
}
```

Although this solution guarantees that no two neighbors are eating simultaneously, it nevertheless must be rejected because it has the possibility of creating a deadlock.

Suppose that all five gets hungry at the same time and pick up their left chopsticks as shown in the following figure. In this case, all chopsticks are locked and none of the philosophers can successfully lock her right chopstick. As a result, we have a circular waiting (i.e., every philosopher waits for his right chopstick that is currently being locked by his right neighbor), and hence a deadlock occurs.



There are several possible good solutions of the problem. We will discuss these in the next lecture.

Operating Systems

Lecture No. 25

Reading Material

- Chapter 7 of the textbook
- Lecture 25 on Virtual TV

Summary

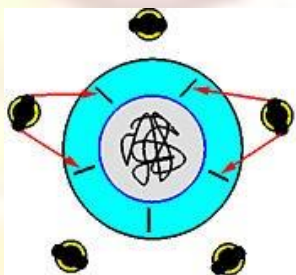
- Dining philosophers problem
- High-level synchronization constructs
- Critical region
- Monitor

Dining Philosophers Problem

Several possibilities that remedy the deadlock situation discussed in the last lecture are listed. Each results in a good solution for the problem.

- Allow at most four philosophers to be sitting simultaneously at the table.
- Allow a philosopher to pick up her chopsticks only if both chopsticks are available (to do this she must pick them up in a critical section)
- Use an asymmetric solution; that is, an odd philosopher picks up first her left chopstick, whereas an even philosopher picks up her right chopstick and then her left chopstick.

Removing the possibility of deadlock does not ensure that starvation does not occur. Imagine that two philosophers are fast thinkers and fast eaters. They think fast and get hungry fast. Then, they sit down in opposite chairs as shown below. Because they are so fast, it is possible that they can lock their chopsticks and eat. After finish eating and before their neighbors can lock the chopsticks and eat, they come back again and lock the chopsticks and eat. In this case, the other three philosophers, even though they have been sitting for a long time, they have no chance to eat. This is a starvation. Note that it is not a deadlock because there is no circular waiting, and everyone has a chance to eat!



High-level Synchronization Constructs

We discussed the problems of deadlock, starvation, and violation of mutual exclusion caused by the poor use of semaphores in lecture 23. We now discuss some high-level synchronization constructs that help solve some of these problems.

Critical regions

Although semaphores provide a convenient and effective mechanism for process synchronization, their incorrect usage can still result in timing errors that are difficult to detect, since these errors occur only if some particular execution takes place, and these sequences do not always happen.

To illustrate how, let us review the solution to the critical section problem using semaphores. All processes share a semaphore variable `mutex`, which is initialized to 1. Each process must execute `wait(mutex)` before entering the critical section and `signal(mutex)` afterward. If this sequence is not observed, two processes may be in their critical sections simultaneously.

To deal with the type of errors we outlined above and in lecture 23, a number of high-level constructs have been introduced. In this section we describe one fundamental high-level synchronization construct—the **critical region**. We assume that a process consists of some local data, and a sequential program that can operate on the data. Only the sequential program code that is encapsulated within the same process can access the local data. That is, one process cannot directly access the local data of another process. Processes can however share global data.

The critical region high-level synchronization construct requires that a variable `v` of type `T`, which is to be shared among many processes, be declared as:

```
v:shared T;
```

The variable `v` can be accessed only inside a region statement of the following form:

```
region v when B do S;
```

This construct means that, while statement `S` is being executed, no other process can access the variable `v`. The expression `B` is a Boolean expression that governs the access to the critical region. When a process tries to enter the critical-section region, the Boolean expression `B` is evaluated. If the expression is true, statement `S` is executed. If it is false, the process relinquishes the mutual exclusion and is delayed until `B` becomes true and no other process is in the region associated with `v`. Thus if the two statements,

```
region v when(true) S1;  
region v when(true) S2;
```

are executed concurrently in distinct sequential processes, the result will be equivalent to the sequential execution “`S1` followed by `S2`” or “`S2` followed by `S1`”.

The critical region construct can be effectively used to solve several certain general synchronization problems. We now show use of the critical region construct to solve the bounded buffer problem. Here is the declaration of buffer:

```
struct buffer {
    item pool[n];
    int count, in, out;
};
```

The producer process inserts a new item (stored in nextp) into the shared buffer by executing

```
region buffer when(count < n) {
    pool[in] = nextp;
    in = (in+1)%n;
    count++;
}
```

The consumer process removes an item from the shared buffer and puts it in nextc by executing

```
region buffer when(count > 0) {
    nextc = pool[out];
    out = (out+1)%n;
    count--;
}
```

Monitors

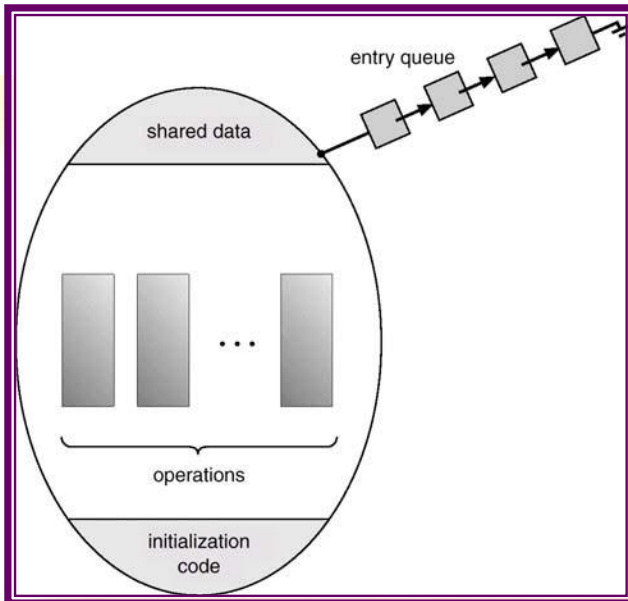
Another high-level synchronization construct is the monitor type. A **monitor** is characterized by local data and a set of programmer-defined operators that can be used to access this data; local data be accessed only through these operators. The representation of a monitor type consists of declarations of variables whose values define the state of an instance of the type, as well as the bodies of procedures or functions that implement operations on the type. Normal scoping rules apply to parameters of a function and to its local variables. The syntax of the monitor is as follows:

```
monitor monitor_name
{
    shared variable declarations

    procedure body P1(..) { ...}
    procedure body P1(..) { ...}
    ...
    procedure body P1(..) { ...}
    {
        initialization code
    }
}
```

The monitor construct ensures that only one process at a time can be active within the monitor. Consequently, the programmer does not need to code this synchronization

construct explicitly. While one process is active within a monitor, other processes trying to access a monitor wait outside the monitor. The following diagram shows the big picture of a monitor.



However, the monitor construct as defined so far is not powerful enough to model some synchronization schemes. For this purpose we need to define additional synchronization mechanisms. These mechanisms are provided by the **condition construct** (also called **condition variable**). A programmer who needs to write her own tailor made synchronization scheme can define one or more variables of type condition.

```
condition x, y;
```

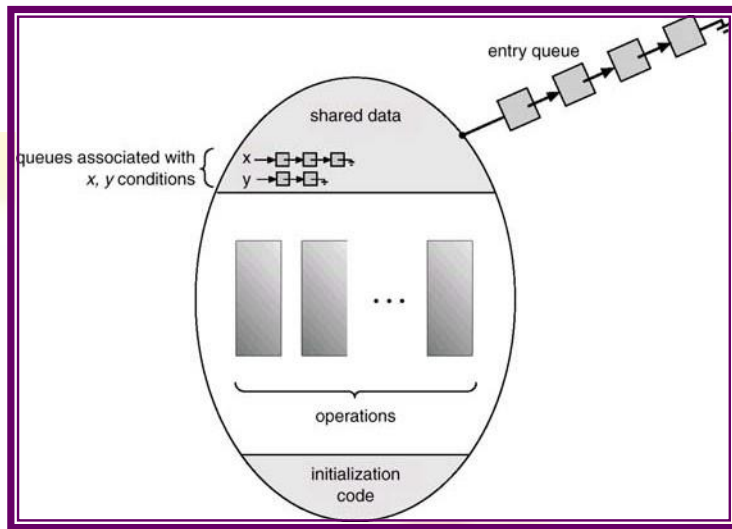
The only operations that can be invoked on a condition variable are wait and signal. The operation

```
x.wait();
```

means that the process invoking this operation is suspended until another process invokes.

```
x.signal();
```

The `x.signal()` operation resumes exactly one suspended process. If no process is suspended, then the signal operation has no effect; that is, the state of `x` is as though the operation were never executed. This is unlike the signal operation on a semaphore, where a signal operation always increments value of the semaphore by one. Monitors with condition variables can solve more synchronization problems than monitors alone. Still only one process can be active within a monitor but many processes may be waiting for a condition variable within a monitor, as shown in the following diagram.



In the next lecture we will discuss a monitor-based solution for the dining philosophers problem.

Operating Systems

Lecture No. 26

Reading Material

- Chapters 7 and 8 of the textbook
- Lecture 26 on Virtual TV

Summary

- Monitor-based solution of the dining philosophers problem
- The deadlock problem
- Deadlock characterization
- Deadlock handling
- Deadlock prevention

Monitor-based Solution for the Dining Philosophers Problem

Let us illustrate these concepts by presenting a deadlock free solution to the dining philosophers problem. Recall that a philosopher is allowed to pick up her chopsticks only if both of them are available. To code this solution we need to distinguish among three states in which a philosopher may be. For this purpose we introduce the following data structure:

```
enum {thinking, hungry, eating} state[5];
```

Philosopher i can set the variable $state[i]=eating$ only if her two neighbors are not eating: $(state[(i+4)\%5] \neq eating)$ and $(state[(i+1)\%5] \neq eating)$.

We also need to declare five condition variables, one for each philosopher as follows. A philosopher uses her condition variable to delay herself when she is hungry, but is unable to obtain the chopsticks she needs.

```
condition self[5];
```

We are now in a position to describe our monitor-based solution to the dining-philosophers problem. The distribution of the chopsticks is controlled by the monitor dp ; whose definition is as follows:

```

monitor dp
{
    enum {thinking,hungry,eating} state[5];
    condition self[5];

    void pickup(int i)
    {
        state[i]=hungry;
        test(i);
        if (state[i] != eating)
            self[i].wait();
    }
    void putdown(int i)
    {
        state[i]=thinking;
        test((i+4)%5);
        test((i+1)%5);
    }
    void test(int i)
    {
        if ((state[(i+4)%5]!=eating) &&
            (state[i]==hungry)&& state[(i+1)%5]!=eating) {
            state[i]=eating;
            self[i].signal();
        }
    }
    void init()
    {
        for(int i=0;i<5;i++)
            state[i]=thinking;
    }
}

```

Each philosopher before starting to eat must invoke the pickup operation. This operation ensures that the philosopher gets to eat if none of its neighbors are eating. This may result in the suspension of the philosopher process. After the successful completion of the operation, the philosopher may eat. Following this, the philosopher invokes the putdown operation and may start to think. The putdown operation checks if a neighbor (right or left—in this order) of the leaving philosopher wants to eat. If a neighboring philosopher is hungry and neither of that philosopher's neighbors is eating, then the leaving philosopher signals it so that she could eat. In order to use this solution, a philosopher *i* must invoke the operations pickup and putdown in the following sequence:

```

dp.pickup(i);
...
eat
...
dp.putdown(i);

```

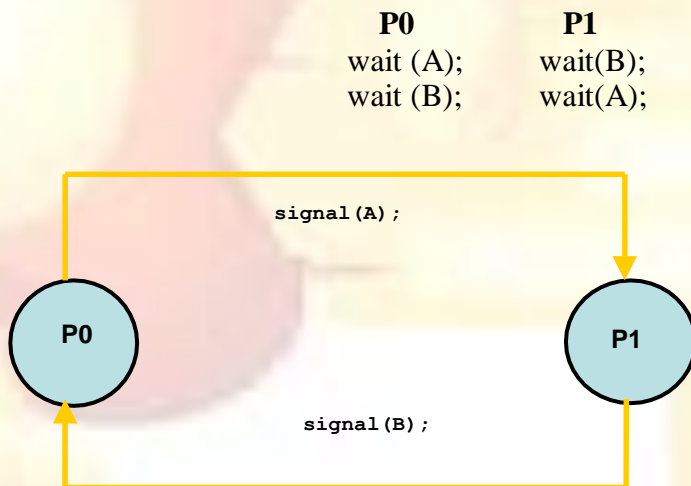
It is easy to show that this solution ensures that no two neighbors are eating simultaneously and that no deadlocks will occur. We note, however, that it is possible for a philosopher to starve to death. You should think about this problem and satisfy yourself.

The Deadlock Problem

A set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set. Here's an example:

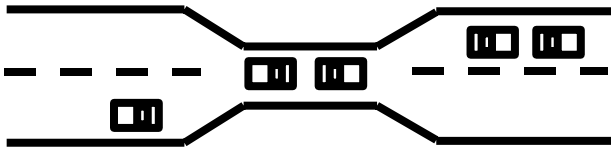
- System has 2 tape drives.
- P1 and P2 each hold one tape drive and each needs another one.

Another deadlock situation can occur when the poor use of semaphores, as discussed in lecture 23. We reproduce that situation here. Assume that two processes, P0 and P1, need to access two semaphores, A and B, before executing their critical sections. Semaphores are initialized to 1 each. The following code snippets show how a situation can arise where P0 holds semaphore A, P1 holds semaphore B, and both wait for the other semaphore—a typical deadlock situation as shown in the figure that follows the code.



In the first solution for the dining philosophers problem, if all philosophers become hungry at the same time, they will pick up the chopsticks on their right and wait for getting the chopsticks on their left. This causes a deadlock.

Yet another example of a deadlock situation can occur on a one-way bridge, as shown below. Traffic flows only in one direction, and each section of a bridge can be viewed as a resource. If a deadlock occurs, it can be resolved if one car backs up (preempt resources and rollback). Several cars may have to be backed up if a deadlock occurs. Starvation is possible.



In the next three to four lectures, we will discuss the issue of deadlocks in computer systems in detail.

System Model

A system consists of a finite number of resources to be distributed among a finite number of cooperating processes. The resources are partitioned into several types, each of which consists of some number of identical instances. Memory space, CPU cycles, disk drive, file are examples of resource types. A system with two identical tape drives is said to have two instances of the resource type disk drive.

If a process requests an instance of a resource type, the allocation of any instance of that type will satisfy the request. If it will not, then the instances are not identical and the resource type classes have not been defined properly.

A process must request a resource before using it, and must release the resource after using it. A process may request as many resources as it requires in order to carryout its designated task. Obviously, the number of resources requested may not exceed the total number of resources available in the system. Under the normal mode of operation, a process may utilize a resource in only the following sequence:

1. **Request:** The process requests a needed resource. If the request cannot be granted immediately, then the requesting process must wait until it can acquire the resource.
2. **Use:** The process can use the resource.
3. **Release:** The process releases the resource.


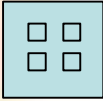
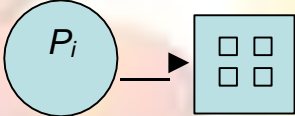
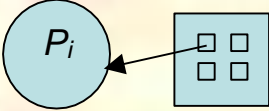
Deadlock Characterization

The following four conditions must hold simultaneously for a deadlock to occur:

1. **Mutual exclusion:** At least one resource must be held in a non-sharable mode; that is only one process at a time can use the resource. If another process requests that resource, the requesting process must be delayed until the resource has been released.
2. **Hold and wait:** A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes.
3. **No preemption:** Resources cannot be preempted. That is, after using it a process releases a resource only voluntarily.
4. **Circular wait:** A set $\{P_0, P_1 \dots P_n\}$ of waiting processes must exist such that P_0 is waiting for a resource that is held by P_1 , P_1 is waiting for a resource that is held by P_2 , and so on, P_{n-1} is waiting for a resource held by P_n , and P_n is waiting for a resource held by P_0 .

Resource Allocation Graphs

Deadlocks can be described more precisely in terms of a directed graph called a system **resource allocation graph**. This graph consists of a set of vertices V and a set of edges E . The set of vertices is partitioned into two different types of nodes $P = \{P_0, P_1, \dots, P_n\}$, the set of the active processes in the system, and $R = \{R_0, R_1, \dots, R_n\}$, the set consisting of all resource types in the system. A directed edge from a process P_i to resource type R_j signifies that process P_i requested an instance of R_j and is waiting for that resource. A directed edge from R_j to P_i signifies that an instance of R_j has been allocated to P_i . We will use the following symbols in a resource allocation graph.

- Process 
- Resource Type with 4 instances 
- P_i requests instance of R_j 
- P_i is holding an instance of R_j 

The resource allocation graph shown below depicts the following situation:

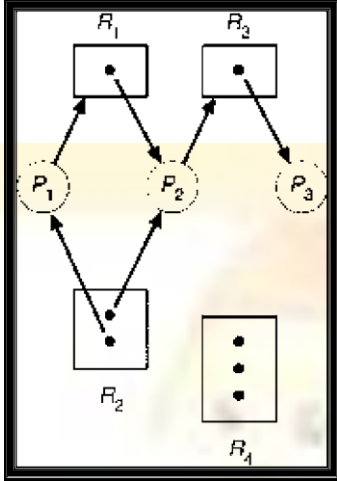
- $P = \{P_1, P_2, P_3\}$
- $R = \{R_1, R_2, R_3\}$
- $E = \{P_1 \rightarrow R_1, P_2 \rightarrow R_3, R_1 \rightarrow P_2, R_2 \rightarrow P_2, R_2 \rightarrow P_1, P_3 \rightarrow R_3\}$

Resource Instances

- One instance of resource type R_1
- Two instances of resource type R_2
- One instance of resource type R_3
- Three instances of resource type R_4

Process States

- Process P_1 is holding an instance of resource R_2 , and is waiting for an instance of resource R_1 .
- Process P_2 is holding an instance of resource R_1 and R_2 , and is waiting for an instance of resource R_3 .
- Process P_3 is holding an instance of resource R_3 .



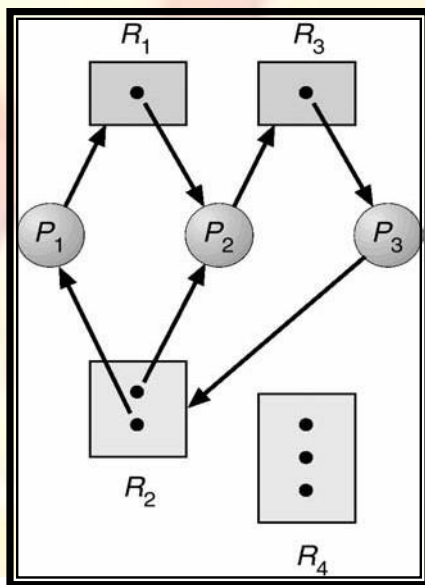
Given the definition of a resource allocation graph, it can be shown that if the graph contains no cycles, then no process is deadlocked. If the graph contains cycles then:

- If only one instance per resource type, then a deadlock exists.
- If several instances per resource type, possibility of deadlock exists.

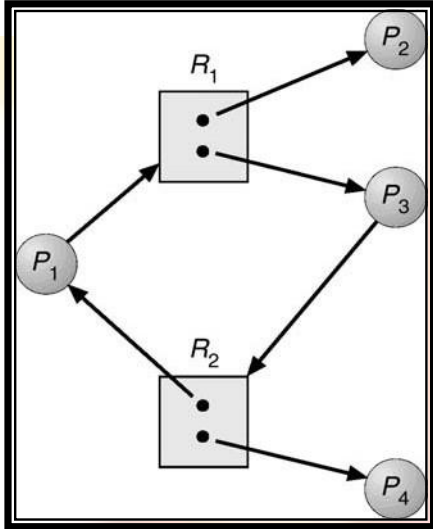
Here is a resource allocation graph with a deadlock. There are two cycles in this graph:

$\{P_1 \rightarrow R_1, R_1 \rightarrow P_2, P_2 \rightarrow R_3, R_3 \rightarrow P_3, P_3 \rightarrow R_2, R_2 \rightarrow P_1\}$ and
 $\{P_2 \rightarrow R_3, R_3 \rightarrow P_3, P_3 \rightarrow R_2, R_2 \rightarrow P_2\}$

No process will release an already acquired resource and the three processes will remain in the deadlock state.



The graph shown below has a cycle but there is no deadlock because processes P2 and P4 do not require further resources to complete their execution and will release the resources they are currently hold in finite time. These resources can then be allocated to P1 and P3 for them to resume their execution.



In the next lecture, we will characterize deadlocks. In other words, we will discuss the condition that must hold for a deadlock to occur. Following this we will discuss the various techniques to handle deadlocks.

Operating Systems

Lecture No. 27

Reading Material

- Chapter 8 of the textbook
- Lecture 27 on Virtual TV

Summary

- Deadlock handling
- Deadlock prevention
- Deadlock avoidance

Deadlock Handling

We can deal with deadlocks in a number of ways:

- Ensure that the system will never enter a deadlock state.
- Allow the system to enter a deadlock state and then recover from deadlock.
- Ignore the problem and pretend that deadlocks never occur in the system.

These three ways result in the following general methods of handling deadlocks:

1. **Deadlock prevention:** is a set of methods for ensuring that at least one of the necessary conditions cannot hold. These methods prevent deadlocks by constraining how processes can request for resources.
2. **Deadlock Avoidance:** This method of handling deadlocks requires that processes give advance additional information concerning which resources they will request and use during their lifetimes. With this information, it may be decided whether a process should wait or not.
3. **Allowing Deadlocks and Recovering:** One method is to allow the system to enter a deadlocked state, detect it, and recover.

Deadlock Prevention

By ensuring that one of the four necessary conditions for a deadlock does not occur, we may prevent a deadlock.

Mutual exclusion

The mutual exclusion condition must hold for non-sharable resources, e.g., printer. Sharable resources do not require mutually exclusive access and thus cannot be involved in a deadlock, e.g., read-only files. Also, resources whose states can be saved and restored can be shared, such as a CPU. In general, we cannot prevent deadlocks by denying the mutual exclusion condition, as some resources are intrinsically non-sharable.

Hold and Wait

To ensure that the hold and wait condition does not occur in a system, we must guarantee that whenever a process requests a resource, it does not hold any other resources. One protocol that can be used requires each process to request and be allocated all its

resources before it begins execution. We can implement this provision by requiring that system calls requesting resources for a process precede all other system calls.

An alternative protocol requires a process to request resources only when it has none. A process may request some resources and use them. But it must release these before requesting more resources.

The two main disadvantages of these protocols are: firstly, resource utilization may be low, since many resources may be allocated but unused for a long time. Secondly, starvation is possible. A process that needs several popular resources may have to wait indefinitely, because at least one of the resources that it needs is always allocated to some other process.

No preemption

To ensure that this condition does not hold we may use the protocol: if a process is holding some resources and requests another that cannot be allocated immediately to it, then all resources currently being held by the process are preempted. These resources are implicitly released, and added to the list of resources for which the process is waiting. The process will be restarted when it gets all its old, as well as the newly requested resources.

Circular Wait

One way to ensure that this condition never holds is to impose a total ordering of all resource types, and to require that each process requests resources in an increasing ordering of enumeration.

Let $R = \{ R_1, R_2, R_3 \}$ be resource types. We assign to each a unique integer, which allows us to compare two resources and to determine whether one precedes another in our ordering. For example, if the set of resource types R includes tape drives, disk drives, and printers then the function $F: R \rightarrow \mathbb{N}$ might be used to assign positive integers to these resources as follows:

$F(\text{tape drive}) = 1$

$F(\text{disk drive}) = 5$

$F(\text{printer}) = 12$

Each process can request resources in an increasing order of enumeration. For example, a process wanting to use the tape and the disk drive must first request the tape drive and then the disk drive.

We can prove that if processes use this protocol then circular wait can never occur. We will prove this by contradiction. Let's assume that there is a cycle involving process P_0 through P_k and that P_i is holding an instance of R_i , as shown below. The proof follows.

$P_0 \rightarrow P_1 \rightarrow P_2 \rightarrow \dots \rightarrow P_k \rightarrow P_0$
 $R_0 \quad R_1 \quad R_2 \quad \quad \quad R_k \quad R_0$

$\Rightarrow F(R_0) < F(R_1) < \dots < F(R_k) < F(R_0)$

$\Rightarrow F(R_0) < F(R_0)$, which is impossible

\Rightarrow There can be no circular wait.

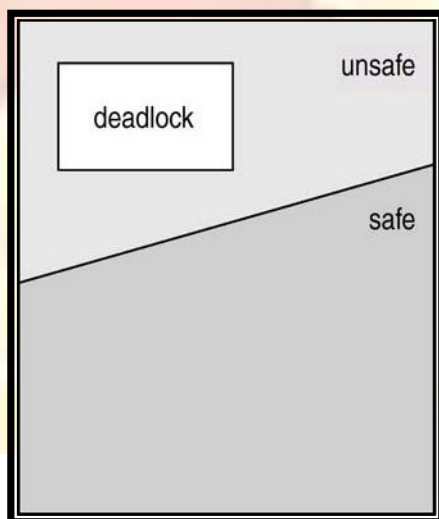
Deadlock Avoidance

One method for avoiding deadlocks is to require additional information about how resources may be requested. Each request for resources by a process requires that the system consider the resources currently available, the resources currently allocated to the process, and the future requests and releases of each process, to decide whether the current request can be satisfied or must wait to avoid a possible future deadlock. The simplest and most useful model requires that each process declare the maximum number of resources of each type that it may need. Given a priori information about the maximum number of resources of each type that may be requested by each process, it is possible to construct an algorithm that ensures that the system will never enter a deadlocked state. A deadlock avoidance algorithm dynamically examines the resource-allocation state to ensure that a circular wait condition can never exist.

Safe State

A state is safe if the system can allocate resources to each process in some order and still avoid a deadlock. More formally a system is in a safe state only if there exists a safe sequence. A sequence of processes $\langle P_1, P_2, \dots, P_n \rangle$ is a safe sequence for the current allocation state if, for each P_i , the resources that P_i can still request can be satisfied by the currently available resources plus all the resources held by all the P_j with $j < i$. In this situation, if the resources that P_i needs are not immediately available, then P_i can wait until all P_j have finished. When they have finished, P_i can obtain all of its needed resources, complete its designated task, return its allocated resources and terminate. When P_i terminates, P_{i+1} can obtain its needed resources and terminate. If no such sequence exists, then the system is said to be unsafe.

If a system is in a safe state, there can be no deadlocks. An unsafe state is not a deadlocked state; a deadlocked state is conversely an unsafe state. Not all unsafe states are deadlocks, however an unsafe state may lead to a deadlock state. Deadlock avoidance makes sure that a system never enters an unsafe state. The following diagram shows the relationship between safe, unsafe, and deadlock states.



Let's consider the following example to explain how a deadlock avoidance algorithm works. There is a system with 12 tape drives and three processes. The current system state is as shown in the following table. The available column shows that initially there are three tape drives available and when process P1 finishes, the two tape drives allocated to it are returned, making the total number of tape drives 5. With 5 available tape drives, the maximum remaining future needs of P0 (of 5 tape drives) can be met. Once this happens, the 5 tape drives that P0 currently holds will go back to the available pool of drives, making the grand total of available tape drives 10. With 10 available drives, the maximum future need of P2 of 7 drives can be met. Therefore, system is currently in a safe state, with the safe sequence $\langle P1, P0, P2 \rangle$.

| Process | Max Need | Allocated | Available |
|---------|----------|-----------|-----------|
| P0 | 10 | 5 | 3 |
| P1 | 4 | 2 | 5 |
| P2 | 9 | 2 | 10 |

Now, consider that P2 requests and is allocated one more tape drive. Assuming that the tape drive is allocated to P2, the new system state will be:

| Process | Max Need | Allocated | Available |
|---------|----------|-----------|-----------|
| P0 | 10 | 5 | 2 |
| P1 | 4 | 2 | 4 |
| P2 | 9 | 3 | |

This new system is not safe. With two tape drives available, P1's maximum remaining future need can be satisfied which would increase the number of available tapes to 4. With 4 tapes available, neither P0's nor P2's maximum future needs can be satisfied. This means that if P2 request for an additional tape drive is satisfied, it would the system in an unsafe state. Thus, P2's request should be denied at this time.

Resource Allocation Graph Algorithm

In addition to the request and assignment edges explained in the previous lectures, we introduce a new type of edge called a **claim edge** to resource allocation graphs. A claim edge $P_i \rightarrow R_j$ indicates that process P_i may request resource R_j at some time in the future. A dashed line is used to represent a claim edge. When P_i requests resource R_j the claim edge is converted to a request edge.

Suppose that P_i requests resource R_j . The request can be granted only if converting the request edge $P_i \rightarrow R_j$ into an assignment edge $R_j \rightarrow P_i$ does not result in the formation of a cycle. If no cycle exists, then the allocation of the resource will leave the system in a safe state. If a cycle is found, then the allocation will put the system in an unsafe state.

Operating Systems

Lecture No. 28

Reading Material

- Chapter 8 of the textbook
- Lecture 28 on Virtual TV

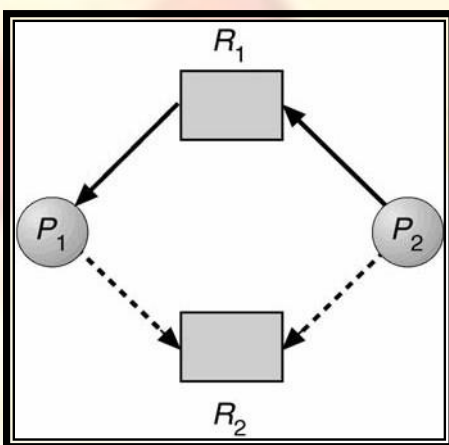
Summary

- Deadlock avoidance
- Banker's algorithms
- Safety algorithm
- Safe Sequence

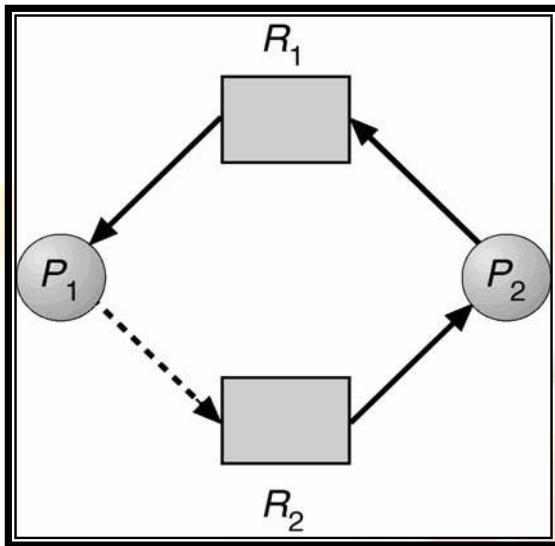
Deadlock Avoidance

Resource Allocation Graph Algorithm

In addition to the request and assignment edges explained in the previous lectures, we introduce a new type of edge called a **claim edge** to resource allocation graphs. A claim edge $P_i \rightarrow R_j$ indicates that process P_i may request resource R_j at some time in the future. A dashed line is used to represent a claim edge. When P_i requests resource R_j the claim edge is converted to a request edge. In the following resource allocation graph, the edge $P_2 \rightarrow R_2$ is a claim edge.



Suppose that P_1 requests resource R_j . The request can be granted only if converting the request edge $P_i \rightarrow R_j$ into an assignment edge $R_j \rightarrow P_i$ does not result in the formation of a cycle. If no cycle exists, then the allocation of the resource will leave the system in a safe state. If a cycle is found, then the allocation will put the system in an unsafe state. The following resource allocation graph shows that the system is in an unsafe state:



Banker's Algorithm

In this algorithm, when a new process enters the system, it must declare the maximum number of instances of each resource type that it may need, i.e., each process must a priori claim maximum use of various system resources. This number may not exceed the total number of instances of resources in the system, and there can be multiple instances of resources. When a process requests a set of resources, the system must determine whether the allocation of these resources will leave the system in a safe state. If it will, the resources are allocated; otherwise the process must wait until some other process releases enough resources. We say that a system is in a safe state if all of the processes in the system can be executed to termination in some order; the order of process termination is called safe sequence. When a process gets all its resources, it must use them and return them in a finite amount of time.

Let n be the number of processes in the system and m be the number of resource types. We need the following data structures in the Banker's algorithm:

- **Available:** A vector of length m indicates the number of available instances of resources of each type. $\text{Available}[j] = k$ means that there are k available instances of resource R_j .
- **Max:** An $n \times m$ matrix defines the maximum demand of resources of each process. $\text{Max}[i,j] = k$ means that process P_i may request at most k instances of resource R_j .
- **Allocation:** An $n \times m$ matrix defines the number of instances of resources of each type currently allocated to each process. $\text{Allocation}[i,j] = k$ means that P_i is currently allocated k instances of resource type R_j .
- **Need:** An $n \times m$ matrix indicates the remaining resource need of each process. $\text{Need}[i,j] = k$ means that P_i may need k more instances of resource type R_j to complete its task. Note that $\text{Need}[i,j] = \text{Max}[i,j] - \text{Allocation}[i,j]$.

Safety Algorithm

The algorithm for finding out whether or not a system is in a safe state can be described as follows:

1. Let Work and Finish be vectors of length m and n, respectively. Initialize Work = Available and Finish[i] = false for $i = 1, 2, \dots, n$.
2. Find an i such that both
 - a) Finish[i] == false
 - b) Need_i <= WorkIf no such i exists go to step 4.
3. Work = Work + Allocation_i
Finish[i] = true
Go to step 2
4. If Finish[i] == true for all i, then the system is in a safe mode.

This algorithm may require an order of $m \times n^2$ operations to decide whether a state is safe.

Resource Request Algorithm

Let Request_i be the request vector for process P_i. if Request_i [j]=k, then process P_i wants k instances of resource R_j. When a request for resources is made by process P_i the following actions are taken:

1. If Request_i <= Need_i go to step 2. Otherwise, raise an error condition since the process has exceeded its maximum claim.
2. If Request_i <= Available, go to step 3. Otherwise P_i must wait, since the resources are not available.
3. Have the system pretend to have allocated the requested resources to process P_i by modifying the state as follows:
Available = Available - Request_i ;
Allocation_i = Allocation_i + Request_i ;
Need_i = Need_i - Request_i ;

Invoke the Safety algorithm. If the resulting resource allocation graph is safe, the transaction is completed. Else, the old resource allocation state is restored and process P_i must wait for Request_i.

An illustrative example

We now show a few examples to illustrate how Banker's algorithm works. Consider a system with five processes P₀ through P₄ and three resource types: A, B, C. Resource type A has 10 instances, resource type B has 5 instances and resource type C has 7 instances. Suppose that at a time T₀ the following snapshot of the system has been taken:

| | Allocation | | | Max | | | Available | | |
|----------------|------------|---|---|-----|---|---|-----------|---|---|
| | A | B | C | A | B | C | A | B | C |
| P ₀ | 0 | 1 | 0 | 7 | 5 | 3 | 3 | 3 | 2 |
| P ₁ | 2 | 0 | 0 | 3 | 2 | 2 | | | |
| P ₂ | 3 | 0 | 2 | 9 | 0 | 2 | | | |
| P ₃ | 2 | 1 | 1 | 2 | 2 | 2 | | | |

| | | | | | | | | | |
|----------------|---|---|---|---|---|---|--|--|--|
| P ₄ | 0 | 0 | 2 | 4 | 3 | 3 | | | |
|----------------|---|---|---|---|---|---|--|--|--|

The content of the matrix Need is defined to be Max- Allocation and is:

| | Need | | |
|----------------|------|---|---|
| | A | B | C |
| P ₀ | 7 | 4 | 3 |
| P ₁ | 1 | 2 | 2 |
| P ₂ | 6 | 0 | 0 |
| P ₃ | 0 | 1 | 1 |
| P ₄ | 4 | 3 | 1 |

In the following sequence of snapshots, we show execution of the Safety algorithm for the given system state to determine if the system is in a safe state. We progressively construct a safe sequence.

| | Allocation | | | Need | | | Available | | |
|----------------|------------|---|---|------|---|---|-----------|---|---|
| | A | B | C | A | B | C | A | B | C |
| P ₀ | 0 | 1 | 0 | 7 | 4 | 3 | 3 | 3 | 2 |
| P ₁ | 2 | 0 | 0 | 1 | 2 | 2 | 5 | 3 | 2 |
| P ₂ | 3 | 0 | 2 | 6 | 0 | 0 | | | |
| P ₃ | 2 | 1 | 1 | 0 | 1 | 1 | | | |
| P ₄ | 0 | 0 | 2 | 4 | 3 | 1 | | | |

- Safe Sequence: < P₁ >

| | Allocation | | | Need | | | Available | | |
|----------------|------------|---|---|------|---|---|-----------|---|---|
| | A | B | C | A | B | C | A | B | C |
| P ₀ | 0 | 1 | 0 | 7 | 5 | 3 | 3 | 3 | 2 |
| P ₁ | 2 | 0 | 0 | 1 | 2 | 2 | 5 | 3 | 2 |
| P ₂ | 3 | 0 | 2 | 6 | 0 | 0 | 7 | 4 | 3 |
| P ₃ | 2 | 1 | 1 | 0 | 1 | 1 | | | |
| P ₄ | 0 | 0 | 2 | 4 | 3 | 1 | | | |

- Safe Sequence: < P₁, P₃ >

| | Allocation | | | Need | | | Available | | |
|----------------|------------|---|---|------|---|---|-----------|---|---|
| | A | B | C | A | B | C | A | B | C |
| P ₀ | 0 | 1 | 0 | 7 | 5 | 3 | 3 | 3 | 2 |
| P ₁ | 2 | 0 | 0 | 1 | 2 | 2 | 5 | 3 | 2 |
| P ₂ | 3 | 0 | 2 | 6 | 0 | 0 | 7 | 4 | 3 |
| P ₃ | 2 | 1 | 1 | 0 | 1 | 1 | 7 | 4 | 5 |
| P ₄ | 0 | 0 | 2 | 4 | 3 | 1 | | | |

- Safe Sequence: < P₁, P₃, P₄ >

| | Allocation | | | Need | | | Available | | |
|----------------|------------|---|---|------|---|---|-----------|---|---|
| | A | B | C | A | B | C | A | B | C |
| P ₀ | 0 | 1 | 0 | 7 | 5 | 3 | 3 | 3 | 2 |

| | | | | | | | | | |
|----------------|---|---|---|---|---|---|---|---|---|
| P ₁ | 2 | 0 | 0 | 6 | 0 | 0 | 5 | 3 | 2 |
| P ₂ | 3 | 0 | 2 | 0 | 1 | 1 | 7 | 4 | 3 |
| P ₃ | 2 | 1 | 1 | 4 | 3 | 1 | 7 | 4 | 5 |
| P ₄ | 0 | 0 | 2 | 7 | 5 | 3 | 7 | 5 | 5 |

- Safe Sequence: $\langle P_1, P_3, P_4, P_0 \rangle$

The Safety algorithm concludes that the system is in a safe state, with $\langle P_0, P_1, P_2, P_3, P_4 \rangle$ being a safe sequence.

Suppose now that process P₁ requests one additional instance of resource type A and two instances of resource type C so Request₁ = (1, 0, 2). To decide whether this request can be fulfilled immediately, we invoke Banker's algorithm, which first check that Request₁ ≤ Available, which is true because (1,0,2) ≤ (3,3,2). It then pretends that this request has been fulfilled, and arrives at the following state:

| | Allocation | | | Need | | | Available | | |
|----------------|------------|---|---|------|---|---|-----------|---|---|
| | A | B | C | A | B | C | A | B | C |
| P ₀ | 0 | 1 | 0 | 7 | 4 | 3 | 2 | 3 | 0 |
| P ₁ | 3 | 0 | 2 | 0 | 2 | 0 | | | |
| P ₂ | 3 | 0 | 2 | 6 | 0 | 0 | | | |
| P ₃ | 2 | 1 | 1 | 0 | 1 | 1 | | | |
| P ₄ | 0 | 0 | 2 | 4 | 3 | 1 | | | |

Banker's algorithm then executes the Safety algorithm to determine if the resultant system will be in a safe state. Here is the complete working of Banker's algorithm. If P₁ requests (1,0,2), let's evaluate if this request may be granted immediately. Banker's algorithm takes the following steps.

1. Is Request₁ ≤ Need₁?
(1,0,2) ≤ (1,2,2) ⇒ true
2. Is Request₁ ≤ Available?
(1,0,2) ≤ (3,3,2) ⇒ true

It then pretends that request is granted and updates the various data structures accordingly. It then invokes the Safety algorithm to determine if the resultant state is safe. Here is sequence of steps taken by the Safety algorithm. The algorithm progressively constructs a safe sequence.

| | Need | | | Allocation | | | Available | | |
|----------------|------|---|---|------------|---|---|-----------|---|---|
| | A | B | C | A | B | C | A | B | C |
| P ₀ | 7 | 4 | 3 | 0 | 1 | 0 | 3 | 3 | 2 |
| P ₁ | 0 | 2 | 0 | 3 | 0 | 2 | | | |
| P ₂ | 6 | 0 | 0 | 3 | 0 | 2 | | | |
| P ₃ | 0 | 1 | 1 | 2 | 1 | 1 | | | |
| P ₄ | 4 | 3 | 1 | 0 | 0 | 2 | | | |

| | Need | | | Allocation | | | Work | | |
|----------------|------|---|---|------------|---|---|------|---|---|
| | A | B | C | A | B | C | A | B | C |
| P ₀ | 7 | 4 | 3 | 0 | 1 | 0 | 2 | 3 | 0 |
| P ₁ | 0 | 2 | 0 | 3 | 0 | 2 | | | |
| P ₂ | 6 | 0 | 0 | 3 | 0 | 2 | | | |
| P ₃ | 0 | 1 | 1 | 2 | 1 | 1 | | | |
| P ₄ | 4 | 3 | 1 | 0 | 0 | 2 | | | |

| | Need | | | Allocation | | | Work | | |
|----------------|------|---|---|------------|---|---|------|---|---|
| | A | B | C | A | B | C | A | B | C |
| P ₀ | 7 | 4 | 3 | 0 | 1 | 0 | 2 | 3 | 0 |
| P ₁ | 0 | 2 | 0 | 3 | 0 | 2 | 5 | 3 | 2 |
| P ₂ | 6 | 0 | 0 | 3 | 0 | 2 | | | |
| P ₃ | 0 | 1 | 1 | 2 | 1 | 1 | | | |
| P ₄ | 4 | 3 | 1 | 0 | 0 | 2 | | | |

- Safe Sequence: < P₁ >

| | Need | | | Allocation | | | Work | | |
|----------------|------|---|---|------------|---|---|------|---|---|
| | A | B | C | A | B | C | A | B | C |
| P ₀ | 7 | 4 | 3 | 0 | 1 | 0 | 2 | 3 | 0 |
| P ₁ | 0 | 2 | 0 | 3 | 0 | 2 | 5 | 3 | 2 |
| P ₂ | 6 | 0 | 0 | 3 | 0 | 2 | 7 | 4 | 3 |
| P ₃ | 0 | 1 | 1 | 2 | 1 | 1 | | | |
| P ₄ | 4 | 3 | 1 | 0 | 0 | 2 | | | |

- Safe Sequence: < P₁, P₃ >

| | Need | | | Allocation | | | Work | | |
|----------------|------|---|---|------------|---|---|------|---|---|
| | A | B | C | A | B | C | A | B | C |
| P ₀ | 7 | 4 | 3 | 0 | 1 | 0 | 2 | 3 | 0 |
| P ₁ | 0 | 2 | 0 | 3 | 0 | 2 | 5 | 3 | 2 |
| P ₂ | 6 | 0 | 0 | 3 | 0 | 2 | 7 | 4 | 3 |
| P ₃ | 0 | 1 | 1 | 2 | 1 | 1 | 7 | 4 | 5 |
| P ₄ | 4 | 3 | 1 | 0 | 0 | 2 | | | |

- Safe Sequence: < P₁, P₃, P₄ >

| | Need | | | Allocation | | | Work | | |
|----------------|------|---|---|------------|---|---|------|---|---|
| | A | B | C | A | B | C | A | B | C |
| P ₀ | 7 | 4 | 3 | 0 | 1 | 0 | 2 | 3 | 0 |
| P ₁ | 0 | 2 | 0 | 3 | 0 | 2 | 5 | 3 | 2 |
| P ₂ | 6 | 0 | 0 | 3 | 0 | 2 | 7 | 4 | 3 |
| P ₃ | 0 | 1 | 1 | 2 | 1 | 1 | 7 | 4 | 5 |
| P ₄ | 4 | 3 | 1 | 0 | 0 | 2 | 7 | 4 | 5 |

- Safe Sequence: < P₁, P₃, P₄, P₀ >

Hence executing Safety algorithm shows that sequence $\langle P_1, P_3, P_4, P_0, P_2 \rangle$ satisfies the safety requirement and so P_1 's request may be granted immediately. Note that safe sequence is not necessarily a unique sequence. There are several safe sequences for the above example. See lecture slides for more details.

Here is another example. P_0 requests $(0,2,0)$. Should this request be granted? In order to answer this question, we again follow Banker's algorithm as shown in the following sequence of steps.

1. Is $\text{Request}_0 \leq \text{Need}_0$?
 $(0,2,0) \leq (7,4,3) \Rightarrow \text{true}$
2. Is $\text{Request}_1 \leq \text{Available}$?
 $(0,2,0) \leq (3,3,2) \Rightarrow \text{true}$

| | Need | | | Allocation | | | Available | | |
|-------|------|---|---|------------|---|---|-----------|---|---|
| | A | B | C | A | B | C | A | B | C |
| P_0 | 7 | 4 | 3 | 0 | 1 | 0 | 3 | 3 | 2 |
| P_1 | 1 | 2 | 2 | 2 | 0 | 0 | | | |
| P_2 | 6 | 0 | 0 | 3 | 0 | 2 | | | |
| P_3 | 0 | 1 | 1 | 2 | 1 | 1 | | | |
| P_4 | 4 | 3 | 1 | 0 | 0 | 2 | | | |

The following is the updated system state. We run the Safety algorithm on this state now and show the steps of executing the algorithm.

| | Need | | | Allocation | | | Work | | |
|-------|------|---|---|------------|---|---|------|---|---|
| | A | B | C | A | B | C | A | B | C |
| P_0 | 7 | 4 | 3 | 0 | 1 | 0 | 3 | 1 | 2 |
| P_1 | 1 | 2 | 2 | 2 | 0 | 0 | | | |
| P_2 | 6 | 0 | 0 | 3 | 0 | 2 | | | |
| P_3 | 0 | 1 | 1 | 2 | 1 | 1 | | | |
| P_4 | 4 | 3 | 1 | 0 | 0 | 2 | | | |

| | Need | | | Allocation | | | Work | | |
|-------|------|---|---|------------|---|---|------|---|---|
| | A | B | C | A | B | C | A | B | C |
| P_0 | 7 | 2 | 3 | 0 | 3 | 0 | 3 | 1 | 2 |
| P_1 | 1 | 2 | 2 | 2 | 0 | 0 | 5 | 2 | 3 |
| P_2 | 6 | 0 | 0 | 3 | 0 | 2 | | | |
| P_3 | 0 | 1 | 1 | 2 | 1 | 1 | | | |
| P_4 | 4 | 3 | 1 | 0 | 0 | 2 | | | |

- Safe Sequence: $\langle P_3 \rangle$

| | Need | | | Allocation | | | Work | | |
|-------|------|---|---|------------|---|---|------|---|---|
| | A | B | C | A | B | C | A | B | C |
| P_0 | 7 | 2 | 3 | 0 | 3 | 0 | 3 | 1 | 2 |
| P_1 | 1 | 2 | 2 | 2 | 0 | 0 | 5 | 2 | 3 |
| P_2 | 6 | 0 | 0 | 3 | 0 | 2 | 7 | 2 | 3 |
| P_3 | 0 | 1 | 1 | 2 | 1 | 1 | | | |

| | | | | | | | | | |
|----------------|---|---|---|---|---|---|--|--|--|
| P ₄ | 4 | 3 | 1 | 0 | 0 | 2 | | | |
|----------------|---|---|---|---|---|---|--|--|--|

- Safe Sequence: $\langle P_3, P_1 \rangle$

| | Need | | | Allocation | | | Work | | |
|----------------|------|---|---|------------|---|---|------|---|---|
| | A | B | C | A | B | C | A | B | C |
| P ₀ | 7 | 2 | 3 | 0 | 3 | 0 | 3 | 1 | 2 |
| P ₁ | 1 | 2 | 2 | 2 | 0 | 0 | 5 | 2 | 3 |
| P ₂ | 6 | 0 | 0 | 3 | 0 | 2 | 7 | 2 | 3 |
| P ₃ | 0 | 1 | 1 | 2 | 1 | 1 | 10 | 2 | 5 |
| P ₄ | 4 | 3 | 1 | 0 | 0 | 2 | | | |

- Safe Sequence: $\langle P_3, P_1, P_2 \rangle$

| | Need | | | Allocation | | | Work | | |
|----------------|------|---|---|------------|---|---|------|---|---|
| | A | B | C | A | B | C | A | B | C |
| P ₀ | 7 | 2 | 3 | 0 | 3 | 0 | 3 | 1 | 2 |
| P ₁ | 1 | 2 | 2 | 2 | 0 | 0 | 5 | 2 | 3 |
| P ₂ | 6 | 0 | 0 | 3 | 0 | 2 | 7 | 2 | 3 |
| P ₃ | 0 | 1 | 1 | 2 | 1 | 1 | 10 | 2 | 5 |
| P ₄ | 4 | 3 | 1 | 0 | 0 | 2 | 10 | 5 | 5 |

- Safe Sequence: $\langle P_3, P_1, P_2, P_0, P_4 \rangle$

Hence executing the safety algorithm shows that sequence $\langle P_3, P_1, P_2, P_0, P_4 \rangle$ satisfies safety requirement. And so P₀'s request may be granted immediately.

Suppose P₀ requests (0,2,0). Can this request be granted after granting P₁'s request of (1,0,2)?

Operating Systems

Lecture No. 29

Reading Material

- Chapter 8 of the textbook
- Lecture 29 on Virtual TV

Summary

- Deadlock detection: resources with single and multiple instances
- Recovery from deadlocks
- Process termination
- Resource preemption

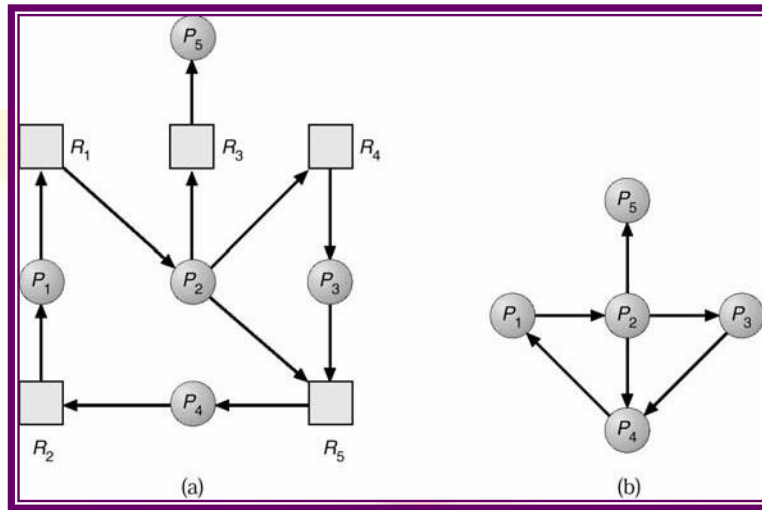
Deadlock Detection

If a system does not employ either a deadlock prevention or a deadlock avoidance algorithm then a deadlock may occur. In this environment, the system must provide:

- An algorithm that examines (perhaps periodically or after certain events) the state of the system to determine whether a deadlock has occurred
- A scheme to recover from deadlocks

Single Instance of Each Resource Type

If all resources have only a single instance, then we can define a deadlock detection algorithm that uses a variant of the resource allocation graph, called a **wait-for graph**. We obtain this graph from the resource allocation graph by removing the nodes of type resource and collapsing the appropriate edges. More precisely, an edge from P_i to P_j in a wait-for graph implies that process P_i is waiting for process P_j to release a resource that P_i needs. An edge $P_i \rightarrow P_j$ exists in a wait-for graph exists if and only if the corresponding resource allocation graph contains two edges for $P_i \rightarrow R_q$ and $R_q \rightarrow P_j$ some resource R_q . As before, a deadlock exists in the system if and only if the wait for graph contains a cycle. To detect deadlocks the system needs to maintain the wait-for graph and periodically to invoke an algorithm that searches for a cycle in the graph. The following diagram shows a resource allocation graph and the corresponding wait-for graph. The system represented by the given wait-for graph has a deadlock because the graph contains a cycle.



Resource-Allocation Graph

Corresponding wait-for graph

Several Instances of a Resource Type

The wait for graph scheme is not applicable to a resource allocation system with multiple instances of each resource type. The deadlock detection algorithm described next is applicable to such a system. It uses the following data structures:

- **Available:** A vector of length m indicates the number of available resources of each type.
- **Allocation:** An $n \times m$ matrix defines the number of resources of each type currently allocated to each process.
- **Request:** An $n \times m$ matrix indicates the current request of each process. If $\text{Request}[i,j] = k$, then process P_i is requesting k more instances of resource type R_j .

The algorithm is:

- 1) Let $Work$ and $Finish$ be vectors of length m and n respectively. Initialize $Work = Available$. For $i = 1, 2, \dots, n$ if $Allocation[i] \neq 0$ the $Finish[i] = false$; otherwise $Finish[i] = true$
- 2) Find an index i such that both
 - a. $Finish[i] = false$
 - b. $Request_i \leq Work$
 - c. If no such i exists go to step 4.
- 3) $Work = Work + Allocation_i$
 - a. $Finish[i] = true$
 - b. Go to step 2.
- 4) If $Finish[i] = false$, for some $i, 1 \leq i \leq n$, then the system is in a deadlock state. Moreover, if $Finish[i] = false$, then P_i is deadlocked.

We show the working of this algorithm with an example. Consider the following system:

$P = \{ P_0, P_1, P_2, P_3, P_4 \}$

$R = \{ A, B, C \}$

A: 7 instances

B: 2 instances

C: 6 instances

The system is currently in the following state. We want to know if the system has a deadlock. We find this out by running the above algorithm with the following state and construct a sequence in which requests for the processes may be granted.

| | Allocation | | | Request | | | Work | | |
|----------------|------------|---|---|---------|---|---|------|---|---|
| | A | B | C | A | B | C | A | B | C |
| P ₀ | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| P ₁ | 2 | 0 | 0 | 2 | 0 | 2 | | | |
| P ₂ | 3 | 0 | 2 | 0 | 0 | 0 | | | |
| P ₃ | 2 | 1 | 1 | 1 | 0 | 0 | | | |
| P ₄ | 0 | 0 | 2 | 0 | 0 | 2 | | | |

| | Allocation | | | Request | | | Work | | |
|----------------|------------|---|---|---------|---|---|------|---|---|
| | A | B | C | A | B | C | A | B | C |
| P ₀ | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| P ₁ | 2 | 0 | 0 | 2 | 0 | 2 | 0 | 1 | 0 |
| P ₂ | 3 | 0 | 2 | 0 | 0 | 0 | | | |
| P ₃ | 2 | 1 | 1 | 1 | 0 | 0 | | | |
| P ₄ | 0 | 0 | 2 | 0 | 0 | 2 | | | |

- Finish Sequence: $\langle P_0 \rangle$

| | Allocation | | | Request | | | Work | | |
|----------------|------------|---|---|---------|---|---|------|---|---|
| | A | B | C | A | B | C | A | B | C |
| P ₀ | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| P ₁ | 2 | 0 | 0 | 2 | 0 | 2 | 0 | 1 | 0 |
| P ₂ | 3 | 0 | 2 | 0 | 0 | 0 | 3 | 1 | 2 |
| P ₃ | 2 | 1 | 1 | 1 | 0 | 0 | | | |
| P ₄ | 0 | 0 | 2 | 0 | 0 | 2 | | | |

- Finish Sequence: $\langle P_0, P_2 \rangle$

| | Allocation | | | Request | | | Work | | |
|----------------|------------|---|---|---------|---|---|------|---|---|
| | A | B | C | A | B | C | A | B | C |
| P ₀ | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| P ₁ | 2 | 0 | 0 | 2 | 0 | 2 | 0 | 1 | 0 |
| P ₂ | 3 | 0 | 2 | 0 | 0 | 0 | 3 | 1 | 2 |
| P ₃ | 2 | 1 | 1 | 1 | 0 | 0 | 5 | 2 | 3 |
| P ₄ | 0 | 0 | 2 | 0 | 0 | 2 | | | |

- Finish Sequence: $\langle P_0, P_2, P_3 \rangle$

| | Allocation | | | Request | | | Work | | |
|----------------|------------|---|---|---------|---|---|------|---|---|
| | A | B | C | A | B | C | A | B | C |
| P ₀ | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| P ₁ | 2 | 0 | 0 | 2 | 0 | 2 | 0 | 1 | 0 |
| P ₂ | 3 | 0 | 2 | 0 | 0 | 0 | 3 | 1 | 2 |
| P ₃ | 2 | 1 | 1 | 1 | 0 | 0 | 5 | 2 | 3 |
| P ₄ | 0 | 0 | 2 | 0 | 0 | 2 | 5 | 2 | 5 |

Here is the sequence in which requests of processes P₀ through P₄ may be satisfied: < P₀, P₂, P₃, P₄, P₁>. This is not a unique sequence. A few other possible sequences are the following.

- < P₀, P₂, P₃, P₁, P₄>
- < P₀, P₂, P₄, P₁, P₃>
- < P₀, P₂, P₄, P₃, P₁>

Now let us assume that P₂ requests an additional instance of C. Do we have a finish sequence? The work below shows that if this request is granted, the system will enter a deadlock. P₀'s request can be satisfied with currently available resources, but request for no other process can be satisfied after that. Thus, a deadlock exists, consisting of processes P₁, P₂, P₃, and P₄.

| Process | Request | | |
|----------------|---------|---|---|
| | A | B | C |
| P ₀ | 0 | 0 | 0 |
| P ₁ | 2 | 0 | 2 |
| P ₂ | 0 | 0 | 1 |
| P ₃ | 1 | 0 | 0 |
| P ₄ | 0 | 0 | 2 |

| | Allocation | | | Request | | | Work | | |
|----------------|------------|---|---|---------|---|---|------|---|---|
| | A | B | C | A | B | C | A | B | C |
| P ₀ | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| P ₁ | 2 | 0 | 0 | 2 | 0 | 2 | | | |
| P ₂ | 3 | 0 | 2 | 0 | 0 | 1 | | | |
| P ₃ | 2 | 1 | 1 | 1 | 0 | 0 | | | |
| P ₄ | 0 | 0 | 2 | 0 | 0 | 2 | | | |

| | Allocation | | | Request | | | Work | | |
|----------------|------------|---|---|---------|---|---|------|---|---|
| | A | B | C | A | B | C | A | B | C |
| P ₀ | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| P ₁ | 2 | 0 | 0 | 2 | 0 | 2 | 0 | 1 | 0 |
| P ₂ | 3 | 0 | 2 | 0 | 0 | 1 | | | |
| P ₃ | 2 | 1 | 1 | 1 | 0 | 0 | | | |
| P ₄ | 0 | 0 | 2 | 0 | 0 | 2 | | | |

Detection Algorithm Usage

When should we invoke the deadlock detection algorithm? The answer depends on two factors:

1. How often is a deadlock likely to occur?
2. How many processes will be affected by deadlock when it happens?

Hence the options are:

- Every time a request for allocation cannot be granted immediately—expensive but process causing the deadlock is identified, along with processes involved in deadlock
- Periodically, or based on CPU utilization
- Arbitrarily—there may be many cycles in the resource graph and we would not be able to tell which of the many deadlocked processes “caused” the deadlock.

Recovery from Deadlock

When a deadlock detection algorithm determines that a deadlock exists, several alternatives exist. One possibility is to inform the operator that a deadlock has occurred, and to let the operator deal with the deadlock manually. The other possibility is to let the system recover from the deadlock automatically. There are two options for breaking a deadlock. One solution is simply to abort one or more processes to break the circular wait. The second option is to preempt some resources from one or more of the deadlocked processes.

Process Termination

To eliminate deadlocks by aborting a process, we use one of two methods. In both methods the system reclaims all resources allocated to the terminated process.

- Abort all deadlocked processes: This method clearly will break the deadlock cycle, but at a great expense; these processes may have computed for a long time, and the results of these partial computations must be discarded and probably recomputed later.
- Abort one process at a time until the deadlock cycle is eliminated: This method incurs considerable overhead since after each process is aborted, a deadlock detection algorithm must be invoked to determine whether any processes are still deadlocked.

Aborting a process may not be so easy. If a process was in the midst of updating a file, terminating it will leave the system in an inconsistent state. If the partial termination method is used, then given a set of deadlocked processes, we must determine which process should be terminated in an attempt to break the deadlock. This determination is a policy decision similar to CPU scheduling problems. The question is basically an economic one, we should abort those processes the termination of which will incur the minimum cost.

Unfortunately, the term minimum cost is not a precise one. Many factors determine which process is chosen, including:

1. What the priority of the process is
2. How long the process has computed, and how much longer the process will compute before completing its designated task.
3. How many and what type of resources the process has used

4. How many resources the process needs in order to complete
5. How many processes will need to be terminated
6. Whether the process is interactive or batch

Resource Preemption

To eliminate deadlocks using resource preemption, we successively preempt some resources from processes and give these to other processes until the deadlock cycle is broken. If preemption is required to deal with deadlocks, then three issues need to be addressed:

1. **Selecting a victim:** Which resources and which processes are to be preempted? As in process termination, we must determine the order of preemption to minimize cost. Cost factors may include such parameters as the number of resources a deadlock process is holding, and the amount of time a deadlocked process has thus far consumed during its execution.
2. **Rollback:** If we preempt a resource from a process, what should be done with that process? Clearly, it cannot continue with its normal execution; it is missing some needed resource. We must roll back the process to some safe state and restart it from that state. Since, in general it is difficult to determine what a safe state is, the simplest solution is a total rollback: Abort the process and then restart it. However it is more effective to roll back the process only as far as necessary to break the deadlock. On the other hand, this method requires the system to keep more information about the state of all the running processes.
3. **Starvation:** In a system where victim selection is based primarily on cost factors, it may happen that the same process is always picked as the victim. As a result this process never completes its designated task, a starvation situation that needs to be dealt with in any practical system. Clearly, we must ensure that a process is picked as a victim only a finite number of times. The most common solution is to include the number of rollbacks in the cost factor.

Operating Systems

Lecture No. 30

Reading Material

- Chapter 9 of the textbook
- Lecture 30 on Virtual TV

Summary

- Basic concepts
- Logical to physical address translation
- Various techniques for memory management

Basic Concepts

Selection of memory-management method for a specific system depends on many factors especially on the *hardware* design of the system. Recent designs have integrated the hardware and operating system.

Memory consists of a large array of words or bytes, each with its own address. The CPU fetches instructions from memory according to the value of its program counter and other memory management registers such as segment registers in Intel CPUs. These instructions may cause additional loading from and storing to specific memory addresses.

A typical instruction-execution cycle, e.g., first fetches an instruction from memory, which is then decoded and executed. Operands may have to be fetched from memory. After the instruction has been executed, the results are stored back in memory. The memory unit sees only a stream of memory addresses; it does not know how they are generated or what they are for (instructions or data).

Memory Hierarchy

The memory hierarchy includes:

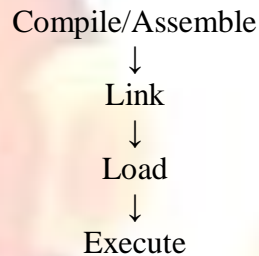
- Very small, extremely fast, extremely expensive, and volatile CPU registers
- Small, very fast, expensive, and volatile cache
- Hundreds of megabytes of medium-speed, medium-price, volatile main memory
- Hundreds of gigabytes of slow, cheap, and non-volatile secondary storage
- Hundreds and thousands of terabytes of very slow, almost free, and non-volatile Internet storage (Web pages, Ftp repositories, etc.)

Memory Management

The purpose of memory management is to ensure fair, secure, orderly, and efficient use of memory. The task of memory management includes keeping track of used and free memory space, as well as when, where, and how much memory to allocate and deallocate. It is also responsible for swapping processes in and out of main memory

Source to Execution

Translation of a source program in a high-level or assembly language involves compilation and linking of the program. This process generates the machine language executable code (also known as a binary image) for the give source program. To execute the binary code, it is loaded into the main memory and the CPU state is set appropriately. The whole process is shown in the following diagram.



Address Binding

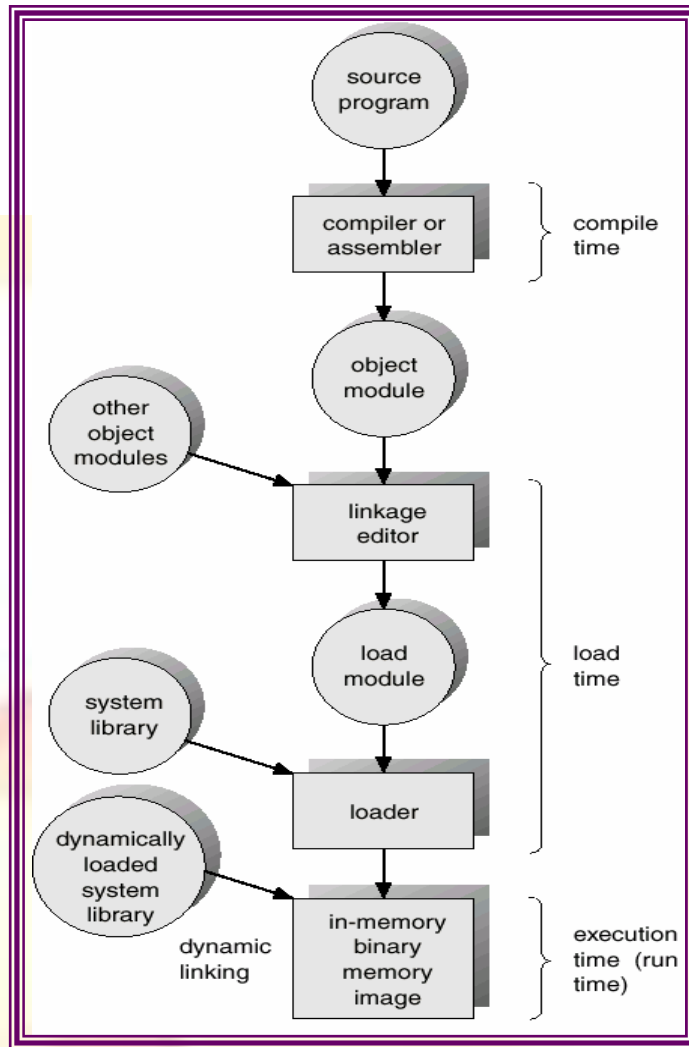
Usually a program resides on a disk as a binary executable or script file. The program must be brought into the memory it to be executed. The collection of processes that is waiting on the disk to be brought into the memory for execution forms the **input queue**.

The normal procedure is to select one of the processes in the input queue and to load that process into the memory. As the process is executed, it accesses instructions and data from memory. Eventually the process terminates and its memory space is become available for reuse.

In most cases, a user program will go through several steps—some of which may be optional—before being executed. These steps are shown in the following diagram. Addresses may be bound in different ways during these steps. Addresses in the source program are generally symbolic (such as an integer variable *count*). Address can be bound to instructions and data at different times, as discussed below briefly.

- **Compile time:** if you know at compile where the process will reside in memory, the **absolute addresses** can be assigned to instructions and data by the compiler.
- **Load time:** if it is not known at compile time where the process will reside in memory, then the compiler must generate **re-locatable code**. In this case the final binding is delayed until load time.
- **Execution time:** if the process can be moved during its execution from one memory segment to another, then binding must be delayed until run time. Special hardware must be available for this to work.

In case of compile and load time binding, a program may not be moved around in memory at run time.



Logical- Versus Physical-Address Space

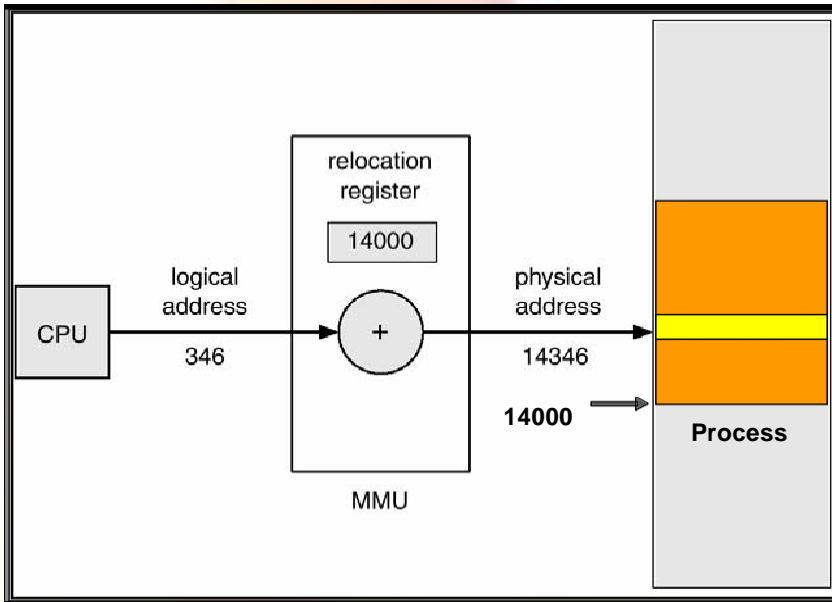
An address generated by the CPU is commonly referred to as a **logical address**, where as an address seen by the memory unit—that is, the one loaded into the **memory-address register** of the memory—is commonly referred to as the **physical address**. In essence, logical data refers to an instruction or data in the process address space where as the physical address refers to a main memory location where instruction or data resides.

The compile time and load time binding methods generate identical logical and physical addresses, where as the execution time binding method results in different physical and logical addresses. In this case we refer to the logical address as the **virtual address**. The set of all logical addresses generated by a program form the **logical address space** of a process; the set of all physical addresses corresponding to these logical addresses is a **physical address space** of the process. The total size of physical address space in a system is equal to the size of its main memory.

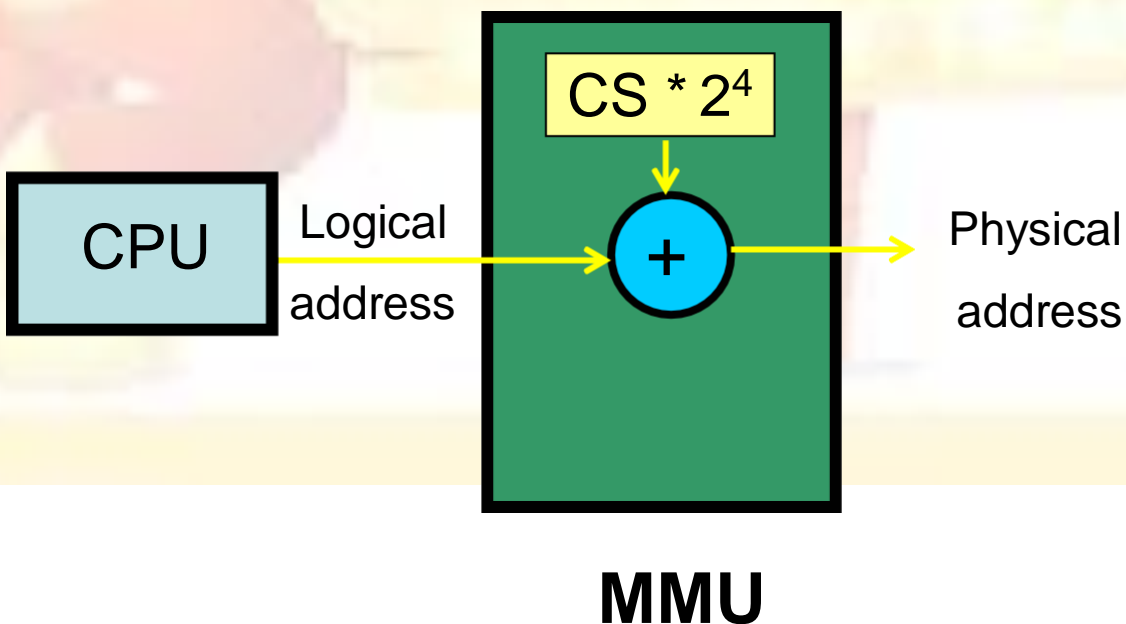
The run-time mapping from virtual to physical addresses is done by a piece of hardware in the CPU, called the **memory management unit (MMU)**.

Translation Examples

In the following two diagrams, we show two simple ways of translating logical addresses into physical address. In both case, there is a “base” register which is loaded with the address of the first byte in the program (instruction or data—in case of the second example, separate registers are used to point to the beginning of code, data, and stack portions of a program). In the first case, the base register is called the **relocation register**. The logical address is translated into the corresponding physical address by adding the logical address to the value of the relocation register, as shown below.



In i8086, the logical address of the next instruction is specified by the value of **instruction pointer (IP)**. The physical address for the instruction is computed by shifting the **code segment register (CS)** left by four bits and adding IP to it, as shown below.



In the following example, we show the logical address for a program instruction and computation of physical address for the given logical address.

- Logical address (16-bit)
IP = 0B10h
CS = D000h
- Physical address (20-bit)
CS * 24 + IP = D0B10h

Various techniques for memory management

Here are some techniques of memory management, which are used in addition to the main techniques of memory management such as paging and segmentation discussed later in the course.

Dynamic Loading

The size of a process is limited to the size of physical memory. To obtain better memory-space utilization, we can use **dynamic loading**. With dynamic loading, a routine is not loaded until it is called. All routines are kept on a disk in a re-locatable format. The main program is loaded into memory and is executed. When a routine needs to call another routine, the calling routine first checks to see whether the other routine has been loaded or not. If not, the re-locatable linking loader is called to load the desired routine into the memory and to update the program's address tables to reflect this change. The control is then passed to the newly loaded routine.

The advantage of dynamic loading is that an unused routine is never loaded. This means that potentially less time is needed to load a program and less memory space is required. However the run time activity involved in dynamic loading is a disadvantage. Dynamic programming does not require special support from the operating system.

Dynamic Linking and Shared Libraries

Some operating systems support only **static linking** in which system language libraries are treated like any other object module and are combined by the loader into the binary proper image. The concept of dynamic linking is similar to that of dynamic loading. Rather than the loading being postponed until execution time, linking is postponed until run-time. This feature is usually used with system libraries. Without this facility, all programs on a system need to have a copy of their language library included in the executable image. This requirement wastes both disk space and main memory. With dynamic linking, a stub is included in the image for each library-routine reference. This *stub* is a small piece of code that indicates how to locate the appropriate memory-resident library routine or how to load the library if the routine is not already present. During execution of a process, stub is replaced by the address of the relevant library code and the code is executed. If library code is not in memory, it is loaded at this time

This feature can be extended to update libraries. A library may be replaced by a new version and all programs that reference the library will automatically use the new version without any need to be re-linked. More than one version of a library may be loaded into the memory and each program uses its version information to decide which copy of the library to use. Only major changes increment the version number. Only programs that are

compiled with the new library version are affected by the incompatible changes incorporated in it. Programs linked before the new library was installed will continue using the older library. This system is also known as **shared libraries**.

Dynamic linking requires potentially less time to load a program. Less disk space is needed to store binaries. However it is a time-consuming run-time activity, resulting in slower program execution. Dynamic linking requires help from the operating system. The `gcc` compiler invokes dynamic linking by default. The `-static` option allows static linking.



Operating Systems

Lecture No. 31

Reading Material

- Chapter 9 of the textbook
- Lecture 31 on Virtual TV

Summary

- Overlays
- Swapping
- Contiguous memory allocation
- MFT

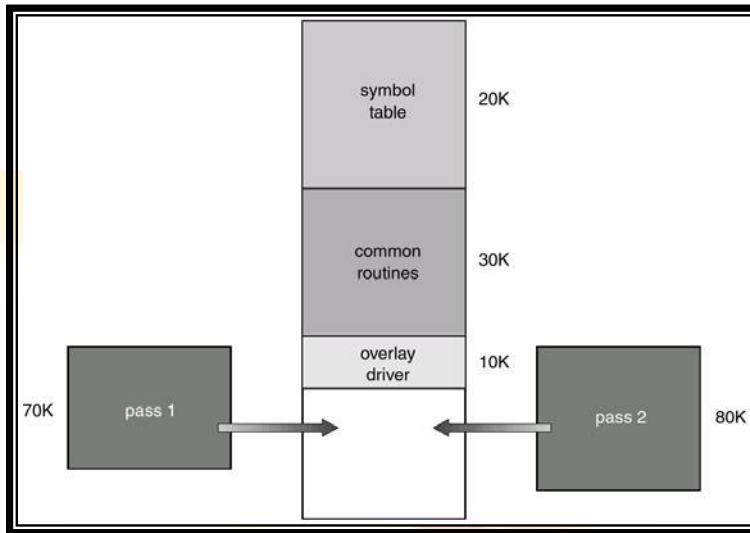
Overlays

To enable a process to be larger than the amount of memory allocated to it, we can use **overlays**. The idea of overlays is to keep in memory only those instructions and data that are needed at any given time. When other instructions are needed, they are loaded into space occupied previously by instructions that are no longer needed. We illustrate the concept of overlays with the example of a two-pass compiler. Here are the various specifications:

- 2-Pass assembler/compiler
- Available main memory: 150k
- Code size: 200k
 - Pass 1..... 70k
 - Pass 2..... 80k
 - Common routines..... 30k
 - Symbol table 20k

Common routines, symbol table, overlay driver, and Pass 1 code are loaded into the main memory for the program execution to start. When Pass 1 has finished its work, Pass 2 code is loaded on top of the Pass 1 code (because this code is not needed anymore). This way, we can execute a 200K process in a 150K memory. The diagram below shows this pictorially.

The problems with overlays are that a) you may not be able to partition all problems into overlays, and b) programmer is responsible of writing the overlays driver.



Overlays Example

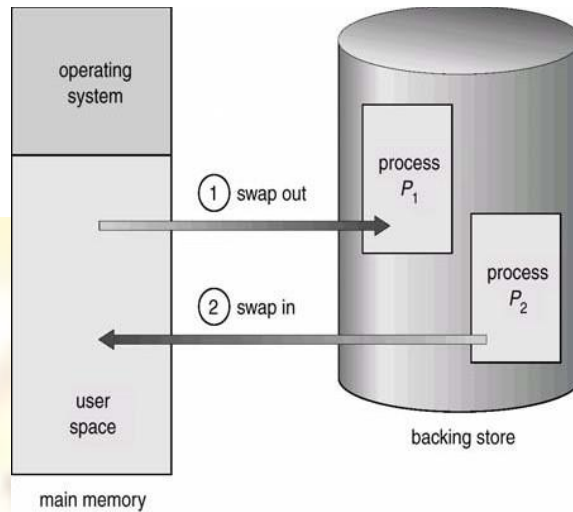
Swapping

A process needs to be in the memory to be executed. A process, however, can be *swapped* temporarily out of memory to a *backing store*, and then brought back into memory for continued execution. Backing store is a fast disk large enough to accommodate copies of all memory images for all users; it must provide direct access to these memory images. The system maintains a *ready queue* of all processes whose memory images are on the backing store or in memory and are ready to run.

For example, assume a multiprogramming environment with a round robin CPU scheduling algorithm. When a quantum expires, the memory manager will start to swap out the process that just finished, and to swap in another process to the memory space that has been freed. A variant of this swapping policy can be used for priority-based scheduling algorithms. If a higher-priority process arrives and wants service, the memory manager can swap out the lower-priority process so that it can load and execute the higher-priority process. When the higher-priority process finishes, the lower-priority process can be swapped back in and continued. This technique is called **roll out, roll in**.

The major part of swap time is transfer time; the total transfer time is directly proportional to the *amount* of memory swapped.

Swapping is constrained by factors like quantum for RR scheduler and pending I/O for swapped out process. Assume that I/O operation was queued because the device was busy. Then if we were to swap out P1, and swap in process P2, the I/O operation might attempt to access memory that now belongs to P2. The solution to this problem are never to swap out processes with pending I/O or to execute I/O in kernel space



Schematic View of Swapping

Cost of Swapping

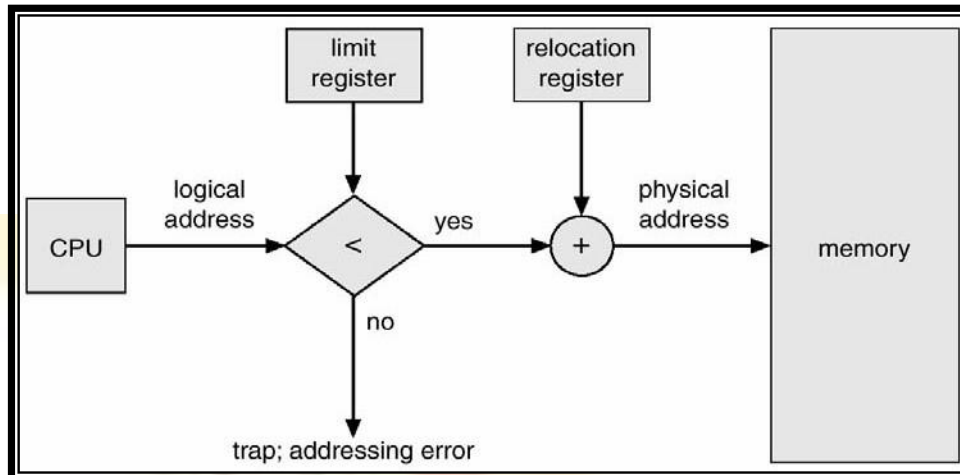
| | |
|--------------------|------------|
| Process size | = 1 MB |
| Transfer rate | = 5 MB/sec |
| Swap out time | = 1/5 sec |
| | = 200 ms |
| Average latency | = 8 ms |
| Net swap out time | = 208 ms |
| Swap out + swap in | = 416 ms |

Contiguous memory allocation

The main memory must accommodate both operating system and the various user spaces. Thus memory allocation should be done efficiently.

The memory is usually divided into two partitions: one for the resident operating system and one for the user processes. The operating system may be placed in the high memory or the low memory. The position of the interrupt vector usually affects this decision. Since the interrupt vector is often in the low memory, programmers place the OS in low memory too.

- It is desirable to have several user processes residing in the memory at the same time. In contiguous memory allocation, each process is contained in a single contiguous section of memory. The **base** (re-location) and **limit** registers are used to point to the smallest memory address of a process and its size, respectively.

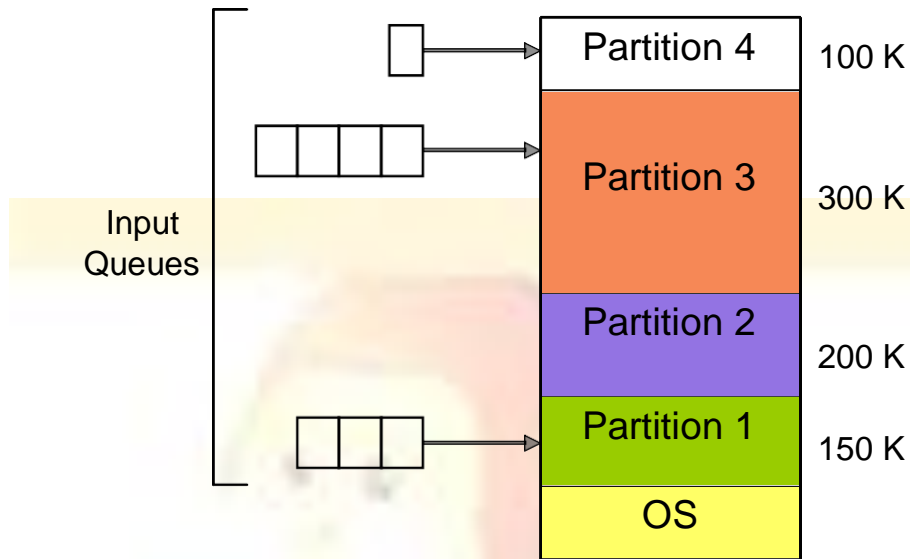


Contiguous Allocation

Multiprogramming with Fixed Tasks (MFT)

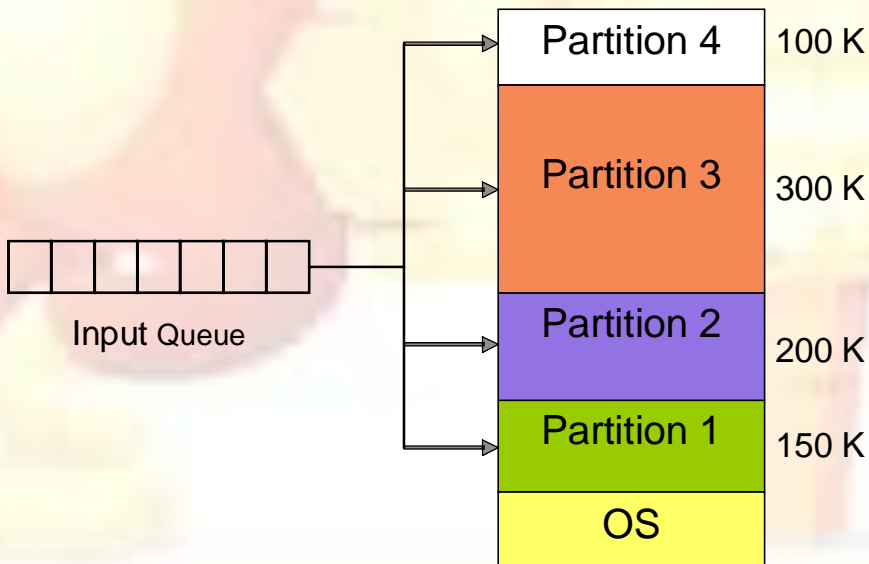
In this technique, memory is divided into several fixed-size partitions. Each partition may contain exactly one process. Thus the degree of multiprogramming is bound by the number of partitions. In this multiple partition method, when a partition is free, a process is selected from the input queue and is loaded in the free partition. When the process terminates, the partition becomes available for another process.

- This was used by IBM for system 360 OS/MFT (multiprogramming with a fixed number of tasks).
- Can have a single input queue instead of one for each partition.
 - So that if there are no big jobs can use big partition for little jobs.
 - Can think of the input queue(s) as the ready list(s) with a scheduling policy of FCFS in each partition.
- The partition boundaries are *not* movable and are set at boot time (must reboot to move a job).
 - MFT can have large **internal fragmentation**, i.e., wasted space *inside* a region
- Each process has a single ``segment'' (we will discuss segments later)
 - No sharing between processes.
 - No dynamic address translation.
 - At load time must ``establish addressability''.
 - Must set a base register to the location at which the process was loaded (the bottom of the partition).
 - The base register is part of the programmer visible register set.
 - This is an example of address translation during load time.
 - Also called **relocation**.
- Storage keys are adequate for protection (IBM method).
- Alternative protection method is base/limit registers.
- An advantage of base/limit is that it is easier to move a job.
- But MFT didn't move jobs so this disadvantage of storage keys is moot.



Multiprogramming with Fixed Tasks (MFT) with a queue per partition

MFT with multiple queues involves load-time address binding. In this technique, there is a potential for wasted memory space i.e. an empty partition but no process in the associated queue. However in MFT with single queue there is a single queue for each partition. The queue is searched for a process when a partition becomes empty. *First-fit*, *best-fit*, *worst-fit* space allocation algorithms can be applied here. The following diagram shows MFT with single input queue.



Multiprogramming with Fixed Tasks (MFT) with one input queue

Operating Systems

Lecture No. 32

Reading Material

- Chapter 9 of the textbook
- Lecture 32 on Virtual TV

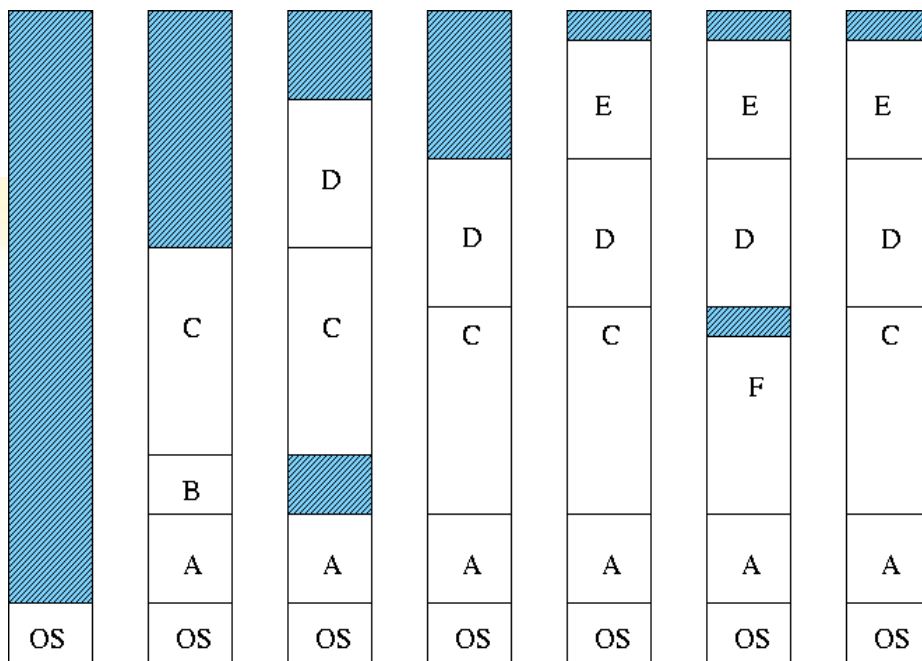
Summary

- MVT
- Paging
- Logical to physical address translation

Multiprogramming with Variable Tasks (MVT)

This is the generalization of the fixed partition scheme. It is used primarily in a batch environment. This scheme of memory management was first introduced in IBM OS/MVT (multiprogramming with a varying number of tasks). Here are the main characteristics of MVT.

- Both the number and size of the partitions change with time.
- Job still has only one segment (as with MFT) but now can be of any size up to the size of the machine and can change with time.
- A single ready list.
- Job can move (might be swapped back in a different place).
- This is dynamic address translation (during run time).
- Must perform an addition on every memory reference (i.e. on every address translation) to add the start address of the partition.
- **Eliminates internal fragmentation.**
 - Find a region the exact right size (leave a hole for the remainder).
 - Not quite true, can't get a piece with 10A755 bytes. Would get say 10A760. But internal fragmentation is *much* reduced compared to MFT. Indeed, we say that internal fragmentation has been eliminated.
- **Introduces external fragmentation**, i.e., holes *outside* any region.
- What do you do *if no hole is big enough* for the request?
 - Can compact memory
 - Transition from bar 3 to bar 4 in diagram below.
 - This is expensive.
 - Not suitable for real time systems.
 - Can swap out one process to bring in another
 - Bars 5-6 and 6-7 in the following diagram



Multiprogramming with Variable Tasks (MVT), external fragmentation, and compaction

External fragmentation

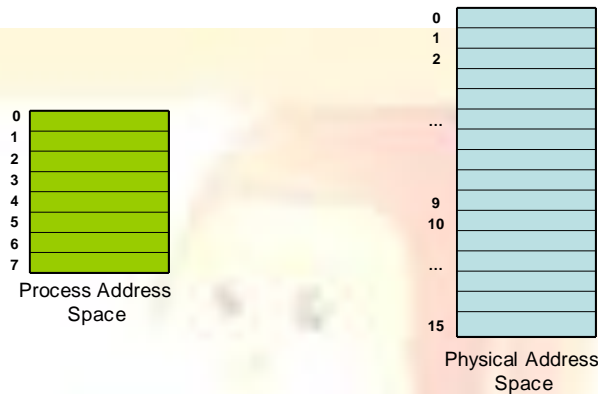
As processes come and go, *holes* of free space are created in the main memory. External Fragmentation refers to the situation when free memory space exists to load a process in the memory but the space is not contiguous. Compaction eliminates external fragmentation by shuffling memory contents (processes) to place all free memory into one large block. The cost of compaction is slower execution of processes as compaction takes place.

Paging

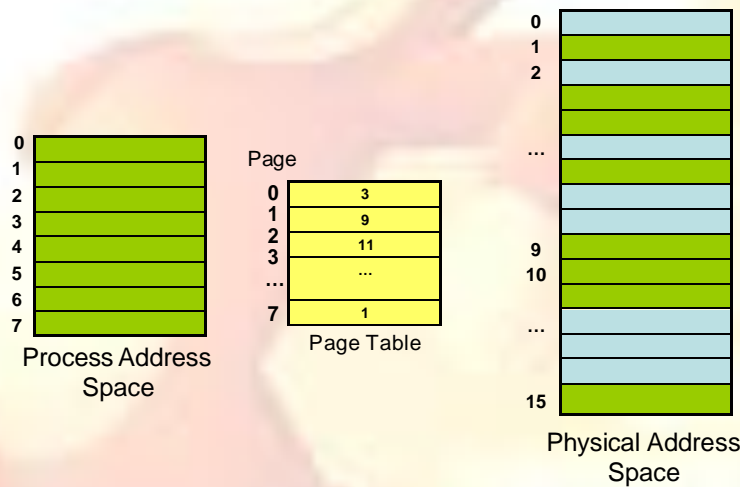
In the memory management techniques discussed so far, two Paging is a memory management scheme that permits the physical address space of a process to be non-contiguous. It avoids the considerable problem of fitting the various sized memory chunks onto the backing store, from which most of the previous memory-management schemes suffered. When some code fragments or data residing in main memory need to be swapped out, space must be found on the backing store. The fragmentation problems discussed in connection with main memory are also prevalent with backing store, except that access is much slower so compaction is impossible.

Physical memory is broken down into fixed-sized blocks, called **frames**, and logical memory is divided into blocks of the same size, called **pages**. The size of a page is a power of 2, the typical page table size lying between 1K and 16K. It is important to keep track of all free frames. In order to run a program of size n pages, we find n free frames and load program pages into these frames. In order to keep track of a program's pages in the main memory a **page table** is used.

Thus when a process is to be executed, its pages are loaded into any available memory frames from the backing store. The following snapshots show process address space with pages (i.e., logical address space), physical address space with frames, loading of paging into frames, and storing mapping of pages into frames in a page table.

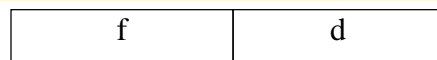


a) Logical and physical address spaces

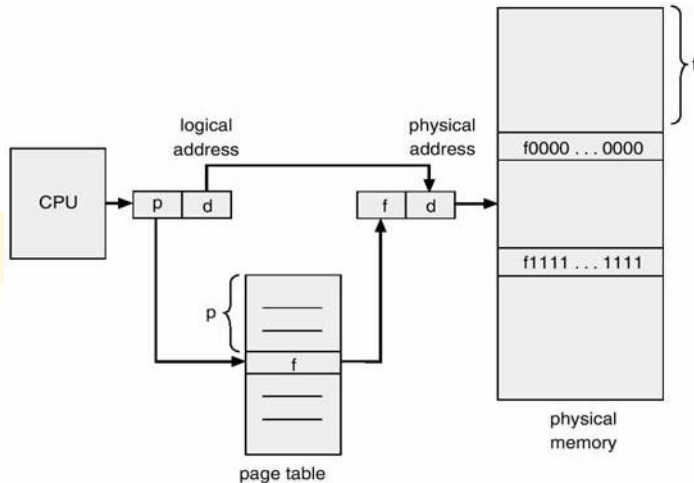


b) Mapping paging in the logical into the frames in the physical address space and keeping this mapping in the page table

Every **logical address** generated by the CPU is divided into two parts: a page number (p) and a page offset (d). The page table contains the base address (frame number) of each page in physical memory. The frame number is combined with the page offset to obtain the physical memory address of the memory location that contains the object addressed by the corresponding logical address. Here p is used to index the process page table; page table entry contains the frame number, f, where page p is loaded. The **physical address** of the location referenced by (p,d) is computed by appending d at the end of f, as shown below:



The hardware support needed for this address translation is shown below.

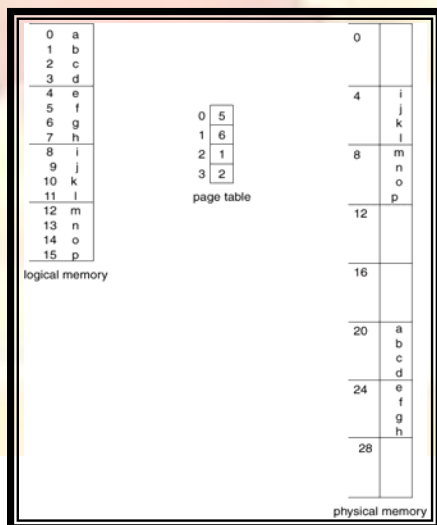


Hardware support for paging

Paging itself is a form of dynamic relocation. When we use a paging scheme, we have no external fragmentation; however we may have **internal fragmentation**. An important aspect of paging is the clear separation between the user's view of memory and the actual physical memory. The user views that memory as one single contiguous space, containing only this program. In fact, the user program is scattered throughout the physical memory, which also holds other programs.

Paging Example

- Page size = 4 bytes
- Process address space = 4 pages
- Physical address space = 8 frames
- Logical address: (1,3) = 0111
- Physical address: (6,3) = 1011



Operating Systems

Lecture No. 33

Reading Material

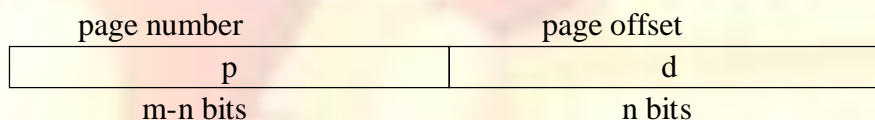
- Chapter 9 of the textbook
- Lecture 33 on Virtual TV

Summary

- Addressing and logical to physical address translation
- Examples: Intel P4 and PDP-11
- Page table implementation
- Performance of paging

Addressing in Paging

The page size is defined by the CPU hardware. If the size of logical address space is 2^m and a page size is 2^n addressing units (bytes or words), then the high-order $m-n$ bits of a logical address designate the page number and the n low order bits designate offset within the page. Thus, the logical address is as follows:



Example:

Assume a **logical address space** of 16 pages of 1024 words, each mapped into a physical memory of 32 frames. Here is how you calculate the various parameters related to paging.

No. of bits needed for **p** = ceiling $[\log_2 16]$ bits = 4 bits

No. of bits needed for **f** = ceiling $[\log_2 32]$ bits = 5 bits

No. of bits needed for **d** = ceiling $[\log_2 2048]$ bits = 11 bits

Logical address size = $|p| + |d| = 4+11$ bits = 15 bits

Physical address size = $|f| + |d| = 5+11$ bits = 16 bits

Page Table Size

Page table size = $NP * PTES$, where NP is the number of pages in the process address space and PTES is the page table entry size (equal to $|f|$ based on our discussion so far).

Page table size = $16 * 5$ bits (for the above example; assuming a byte size page table entry)

Paging in Intel P4

32-bit *linear* address

4K page size

Maximum pages in a process address space = $2^{32} / 4K$

Number of bits needed for **d** = $\log_2 4K$ bits = 12 bits

Number of bits needed for **p** = $32 - 12$ bits = 20

Paging in PDP-11

16-bit logical address

8K page size

Maximum pages in a process address space = $2^{16} / 8K = 8$

$|d| = \log_2 8K = 13$ bits

$|p| = 16 - 13 = 3$ bits

Another Example

Logical address = 32-bit

Process address space = 2^{32} B = 4 GB

Main memory = RAM = 512 MB

Page size = 4K

Maximum pages in a process address space = $2^{32} / 4K = 1M$

$|d| = \log_2 4K = 12$ bits

$|p| = 32 - 12 = 20$ bits

No. of frames = $512 M / 4 K = 128 K$

$|f| = \text{ceiling} [\log_2 128 K] \text{ bits} = 17 \text{ bits} \approx 4 \text{ bytes (rounding to next even-numbered byte)}$

Physical address = 17+12 bits

Implementation of Page table

▪ In the CPU registers

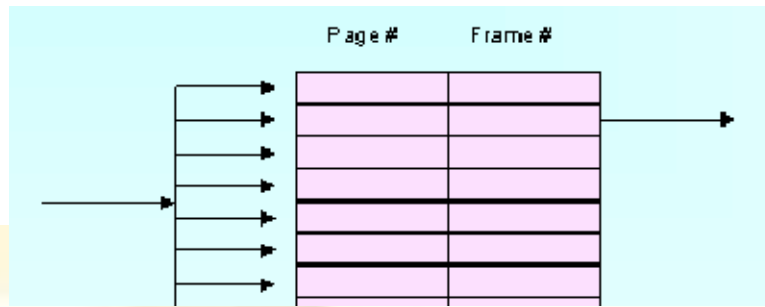
This is OK for small process address spaces and large page sizes. It has the advantage of having *effective memory access time* ($T_{\text{effective}}$) about the same as memory access time (T_{mem}). An example of this implementation is in PDP-11.

▪ In the main memory

A page table base register (PTBR) is needed to point to the page table. With page table in main memory, the effective memory access time, $T_{\text{effective}}$, is $2T_{\text{mem}}$, which is not acceptable because it would slow down program execution by a factor of two.

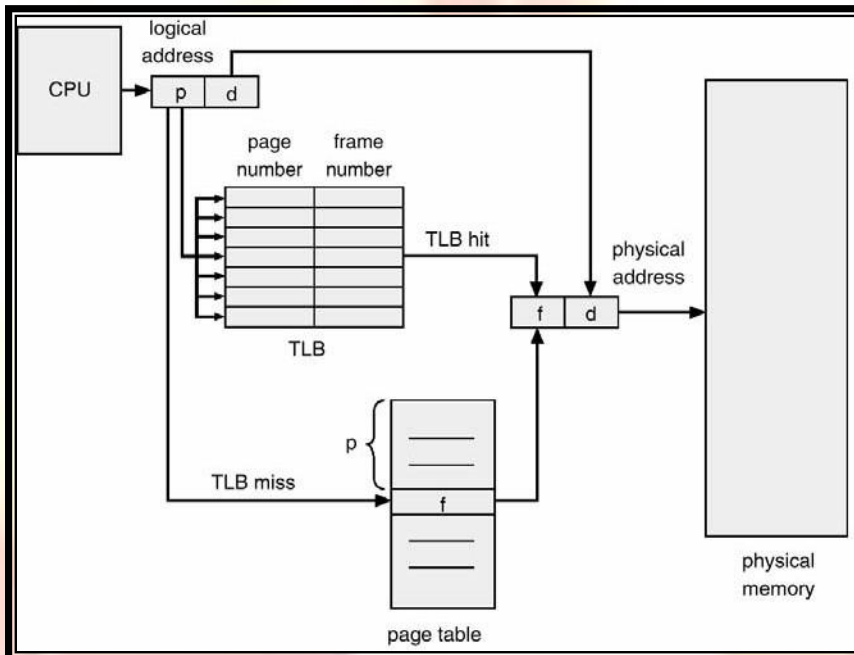
▪ In the translation look-aside buffer (TLB)

A solution to this problem is to use special, small, fast lookup hardware, called translation look-aside buffer (TLB), which typically has 64–1024 entries. Each entry is (key, value). The key is searched for in parallel; on a hit, value is returned. The (key,value) pair is (p,f) for paging. For a logical address, (p,d), TLB is searched for p. If an entry with a key p is found, we have a hit and f is used to form the physical address. Else, page table in the main memory is searched.



TLB –Logical address: (p,d)

The TLB is loaded with the (p,f) pair so that future references to p are found in the TLB, resulting in improved hit ratio. On a context switch, the TLB is flushed and is loaded with values for the scheduled process. Here is the hardware support needed for paging with part of the page table stored in TLB.



Paging Hardware with TLB

Performance of Paging

We discuss performance of paging in this section. The performance measure is the effective memory access time. With part of the page table in the TLB and the rest in the main memory, the effective memory access time on a hit is $T_{mem} + T_{TLB}$ and on a miss is $2T_{mem} + T_{TLB}$.

If HR is hit ratio and MR is miss ratio, the effective access time is given by the following equation

$$T_{effective} = HR (T_{TLB} + T_{mem}) + MR (T_{TLB} + 2T_{mem})$$

We give a few examples to help you better understand this equation.

Example 1

$T_{mem} = 100 \text{ nsec}$

$T_{TLB} = 20 \text{ nsec}$

Hit ratio is 80%

$T_{effective} = 0.8 (20 + 100) + 0.2 (20 + 2*100) \text{ nanoseconds} = 140 \text{ nanoseconds}$

This means that with 80% chances of finding a page table entry in the TLB, the effective access time becomes 40% worse than memory access time without paging.

Example 2

$T_{mem} = 100 \text{ nsec}$

$T_{TLB} = 20 \text{ nsec}$

Hit ratio is 98%

$T_{effective} = 0.98 (20 + 100) + 0.02 (20 + 2*100) \text{ nanoseconds} = 122 \text{ nanoseconds}$

This means that with 98% chances of finding a page table entry in the TLB, the effective access time becomes 22% worse than memory access time without paging. This means that with a small cache and good hit ratio, we can maintain most of the page table in the main memory and get much better performance than keeping the page table in the main memory and not using any cache.

Operating Systems

Lecture No. 34

Reading Material

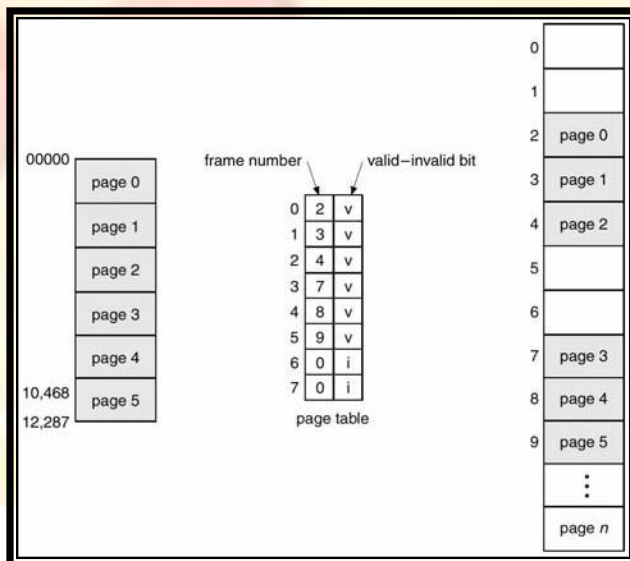
- Chapter 9 of the textbook
- Lecture 34 on Virtual TV

Summary

- Protection under paging
- Structure of the page table
 - Multi-level paging
 - Hashed page tables
 - Inverted page table

Protection under Paging

Memory protection in paging is achieved by associating protection bits with each page. These bits are associated with each page table entry and specify protection on the corresponding page. The primary protection scheme guards against a process trying to access a page that does not belong to its address space. This is achieved by using a valid/invalid (v) bit. This bit indicates whether the page is in the process address space or not. If the bit is set to invalid, it indicates that the page is not in the process's logical address space. Illegal addresses are trapped by using the valid-invalid bit and control is passed to the operating system for appropriate action. The following diagram shows the use of v bit in the page table. In this case, logical address space is six page and any access to pages 6 and 7 will be trapped because the v bits for these pages is set to invalid.



Use of valid/invalid (v) bit for protection under paging

One bit can define the page table to be read and write or read only. Every reference to memory goes through the page table to find the correct frame number. At the same time that the physical address is being computed, the protection bits can be checked to verify that no writes are being made to a read only page. An attempt to write to a read-only page causes a hardware trap to the operating system (memory-protection violation).

This approach can be expanded to provide a finer level of protection. Read, write, and execute bits (r, w, x) can be used to allow a combination of these accesses, similar to the file protection scheme used in the UNIX operating system. Illegal attempts will be trapped to the operating system.

Structure of the Page Table

As logical address spaces become large (32-bit or 64-bit), depending on the page size, page table sizes can become larger than a page and it becomes necessary to page the page table. Additionally, large amount of memory space is used for page table. The following schemes allow efficient implementations of page tables.

- Hierarchical / Multilevel Paging
- Hashed Page Table
- Inverted Page Table

Hierarchical/Multilevel Paging

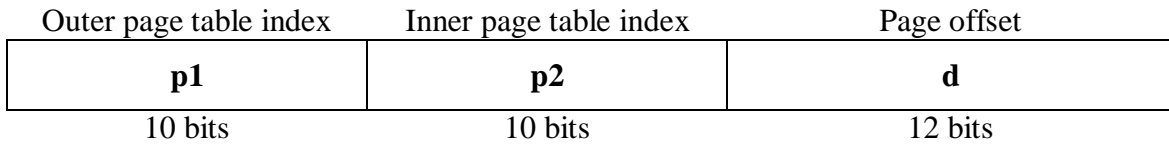
Most modern computers support a large logical address space: (2^{32} to 2^{64}). In such an environment, the page table itself becomes excessively large. Consider the following example:

- Logical address = 32-bit
- Page size = 4K bytes (212 bytes)
- Page table entry = 4 bytes
- Maximum pages in a process address space = $2^{32} / 4K = 1M$
- Maximum pages in a process address space = $2^{32} / 4K = 1M$
- Page table size = 4M bytes

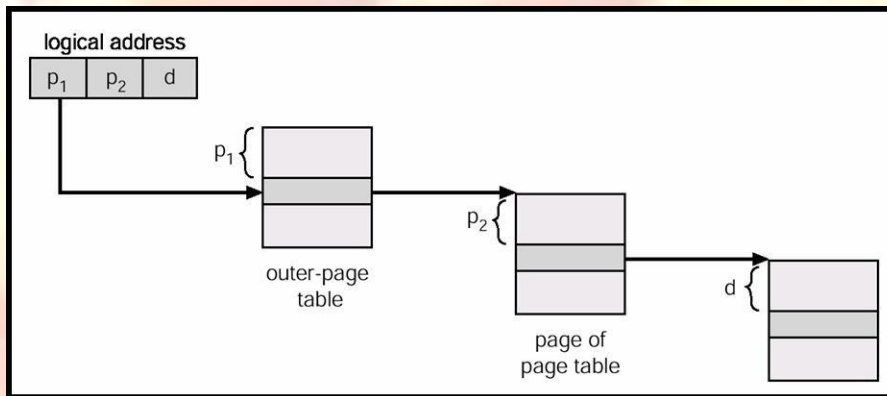
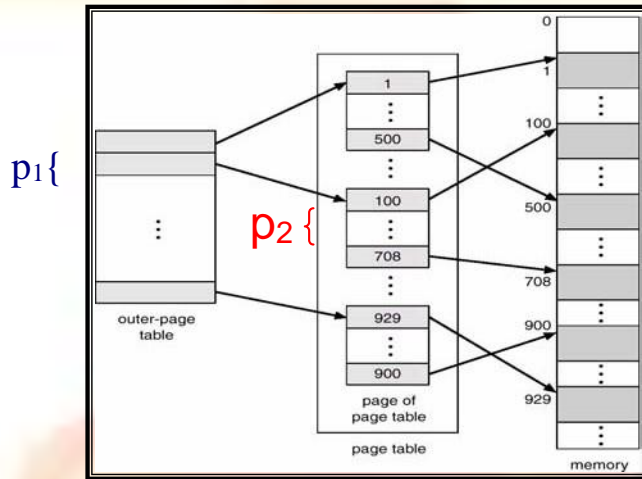
This page table cannot fit in one page. One solution is to page the page table, resulting in a 2-level paging. A page table needed for keeping track of pages of the page table—called the outer page table or page directory. In the above example:

- No. of pages in the page table is $4M / 4K = 1K$
- Size of the outer page table is $1K * 4 \text{ bytes} = 4K \text{ bytes} \Rightarrow$ outer page will fit in one page

In the 32-bit machine described above, we need to partition p into two parts, p1 and p2. p1 is used to index the outer page table and p2 to index the inner page table. Thus the logical address is divided into a page number consisting of 20 bits and a page offset of 12 bits. Since we page the page table, the page number is further divided into a 10-bit page number, and a 10-bit page offset. This is known as **two-level paging**. The following diagram shows division of the logical address in 2-level paging and hierarchical views of the page table.



a) Logical address



b) Two views of address translation for a two-level paging architecture

Another Example: DEC VAX

- Logical address = 32 bits
- Page size = 512 bytes = 2^9 bytes
- Process address space divided into four equal sections
- Pages per section = $2^{30} / 2^9 = 2^{21} = 2\text{M}$
- Size of a page table entry = 4 bytes
- Bits needed for page offset = $\log_2 512 = 9$ bits
- Bits to specify a section = $\log_2 4 = 2$ bits
- Bits needed to index page table for a section = $\log_2 2^{21} = 21$ bits
- Size of a page table = $2^{21} * 4 = 8\text{ MB}$
- 8 MB page table is paged into $8\text{MB} / 512 = 2\text{ K}$ pages

- Size of the outer page table ($2K * 4 = 8 \text{ KB}$) is further paged, resulting in 3-level paging per section

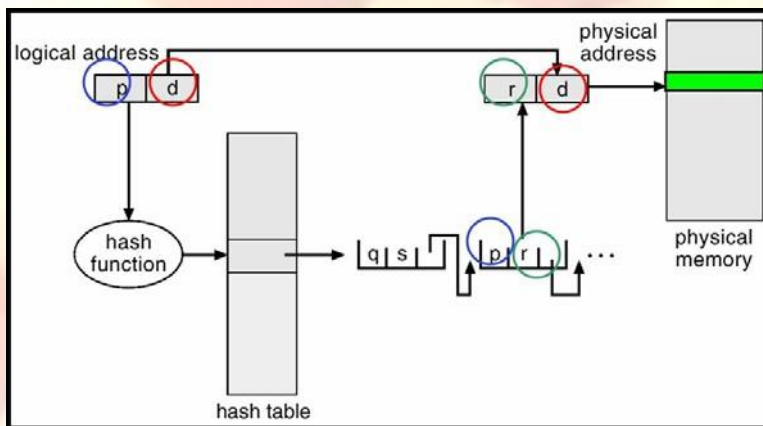
| Section | Page number | Page offset |
|----------|-------------|-------------|
| s | p | d |
| 2 | 21 | 9 |

More Examples

- 32-bit Sun SPARC supports 3-level paging
- 32-bit Motorola 68030 supports 4-level paging
- 64-bit Sun UltraSPARC supports 7-level paging – too many memory references needed for address translation

Hashed Page Table

This is a common approach to handle address spaces larger than 32 bits. Usually open hashing is used. Each entry in the linked list has three fields: page number, frame number for the page, and pointer to the next element—(p, f, next). The page number in the logical address (specified by p) is hashed to get index of an entry in the hash table. This index is used to search the linked list associated with this entry to locate the frame number corresponding to the given page number. The advantage of hashed page tables is smaller page tables.

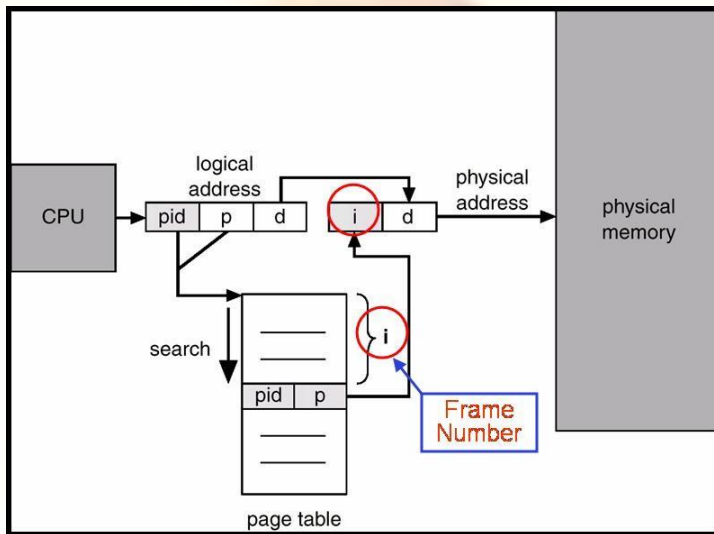


Inverted Page Table

Usually each process has a page table associated with it. The page table has one entry for each page in the address space of the process. For large address spaces (32-bit and above), each page table may consist of millions of entries. These tables may consume large amounts of physical memory, which is required just to keep track of how the mapping of logical address spaces of processes onto the physical memory.

A solution is to use an inverted page table. An **inverted page table** has one entry for each real page (frame) of memory. Each entry consists of the virtual address of the page stored in the in that real memory location, with information about the process that own the page.

Page table size is limited by the number of frames (i.e., the physical memory) and not process address space. Each entry in the page table contains (pid, p). If a page 'p' for a process is loaded in frame 'f', its entry is stored at index 'f' in the page table. We effectively index the page table with frame number; hence the name inverted page table. Examples of CPUs that support inverted pages tables are 64-bit UltraSPARC and PowerPC. The following diagram shows how logical addresses are translated into physical addresses.



Address translation with inverted page table

Operating Systems

Lecture No. 35

Reading Material

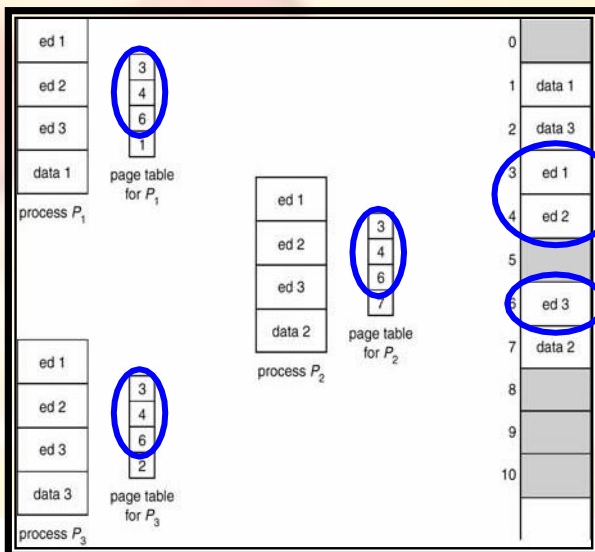
- Chapter 9 of the textbook
- Lecture 35 on Virtual TV

Summary

- Sharing in paging
- Segmentation
- Logical to physical address translation
- Hardware support needed
- Protection and sharing

Sharing in Paging

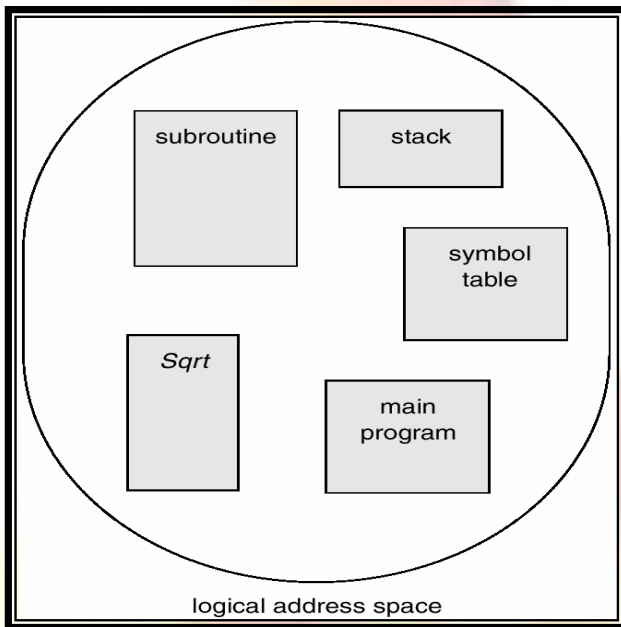
Another advantage of paging is the possibility of *sharing* common code. Reentrant (read-only) code pages of a process address can be shared. If the code is reentrant, it never changes during execution. Thus two or more processes can execute the same code at the same time. Each process has its own copy of registers and data storage to hold the data for the process' execution. The data for two different processes will, of course, vary for each process. Consider the case when multiple instances of a text editor are invoked. Only one copy of the editor needs to be kept in the physical memory. Each user's page table maps on to the same physical copy of the editor, but data pages are mapped onto different frames. Thus to support 40 users, we need only one copy of the editor, which results in saving total space.



Sharing in paging

Segmentation

Segmentation is a memory management scheme that supports programmer's view of memory. A logical-address space is a collection of segments. A segment is a logical unit such as: main program, procedure, function, method, object, global variables, stack, and symbol table. Each segment has a name and length. The addresses specify both the segment name and the offset within the segment. An example of the logical address space of a process with segmentation is shown below.



Logical address space with segmentation

For simplicity of implementation, segments are numbered and are referred to by a segment number, rather than by a segment name. Thus a logical address consists of a two tuple:

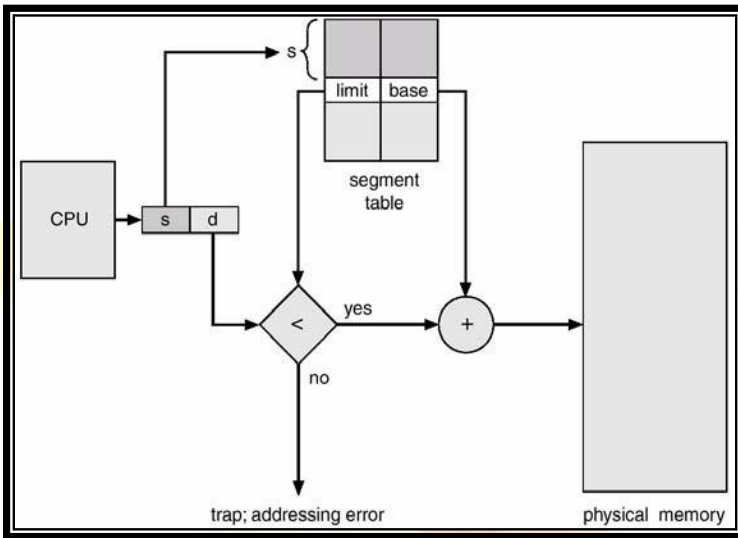
$$\langle \text{segment-number}, \text{offset} \rangle \text{ or } \langle s, d \rangle$$

The segment table maps the two-dimensional logical addresses to physical addresses. Each entry of a segment table has a *base* and a *segment limit*. The segment base contains the starting physical address where the segment resides in memory, whereas the segment limit specifies the length of the segment.

There are two more registers, relevant to the concept of segmentation:

- **Segment-table base register** (STBR) points to the segment table's location in memory.
- **Segment-table length register** (STLR) indicates number of segments used by a program

Segment number s is legal if $s < \text{STLR}$, and offset, d , is legal if $d < \text{limit}$. The following diagram shows the hardware support needed for translating a logical address into the physical address when segmentation is used. This hardware is part of the MMU in a CPU.

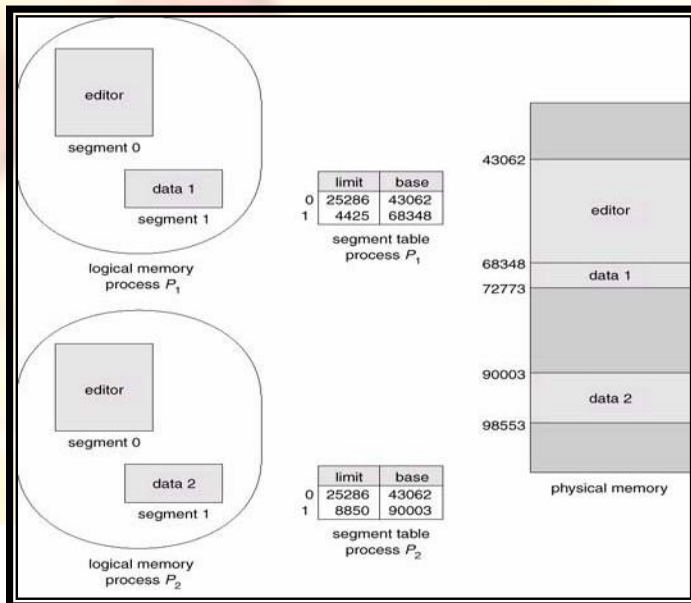


Hardware support for segmentation

For logical to physical address conversion, segment number, s , is used to index the segment table for the process. If $d < \text{limit}$, it is added to the base value to compute the physical address for the given logical address. The segment base and limit values are used to relocate and bound check the reference at runtime.

Sharing of Segments

Another advantage of segmentation is sharing of code or data. Each process has a segment table associated with it, which the dispatcher uses to define the hardware segment table when this process is given the CPU. Segments are shared when entries in the segment tables of two different processes point to the same physical location. The sharing occurs at segment level, thus, any information defined as a segment can be shared.



Sharing in segmentation

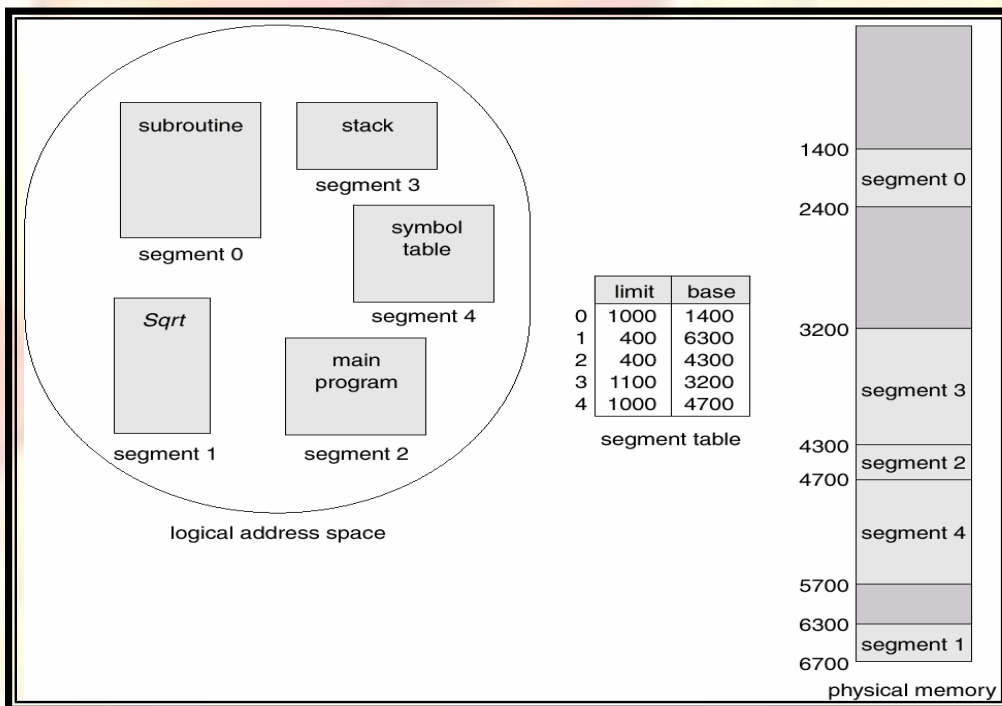
The long-term scheduler must find and allocate memory for all the segments of a user program. This situation is similar to paging except that the segments are of *variable* length; pages are all the same size. Thus memory allocation is a dynamic storage allocation problem, usually solved with a best fit or worst fit algorithm.

Protection

A particular advantage of segmentation is the association of protection with segments. Because the segments represent a semantically defined portion of the program, it is likely that all the entries will be used the same way. Hence, some segments are instructions, whereas other segments are data. In a modern architecture, instructions are non-self-modifying so they can be defined as read only. Or execute only. The memory mapping hardware will check the protection bits associated with each segment-table entry to prevent illegal access to memory, such as attempts to write into a read only segment. By placing an array in its own segment, the memory management hardware will automatically check that array indexes are legal and do not stray outside array boundaries.

The bits associated with each entry in the segment table, for the purpose of protection are:

- Validation bit : if the validation bit is 0, it indicates an illegal segment
- Read, write, execute bits



Issues with Segmentation

Segmentation may then cause external fragmentation (i.e. total memory space exists to satisfy a space allocation request for a segment, but memory space is not contiguous), when all blocks of memory are too small to accommodate a segment. In this case, the process may simply have to wait until more memory (or at least a larger hole) becomes

available or until compaction creates a larger hole. Since segmentation is by nature a dynamic relocation algorithm, we can compact memory whenever we want.

If we define each process to be one segment, this approach reduces to the variable sized partition scheme. T the other extreme, every byte could be put in its own segment and relocated separately. This eliminates external fragmentation altogether, however every byte would need a base register for its relocation, doubling memory use. The next logical step- fixed sized, small segments, is paging i.e. paged segmentation.

Also it might latch a job in memory while it is involved in I/O. To prevent this I/O should be done only into OS buffers.



Operating Systems

Lecture No. 36

Reading Material

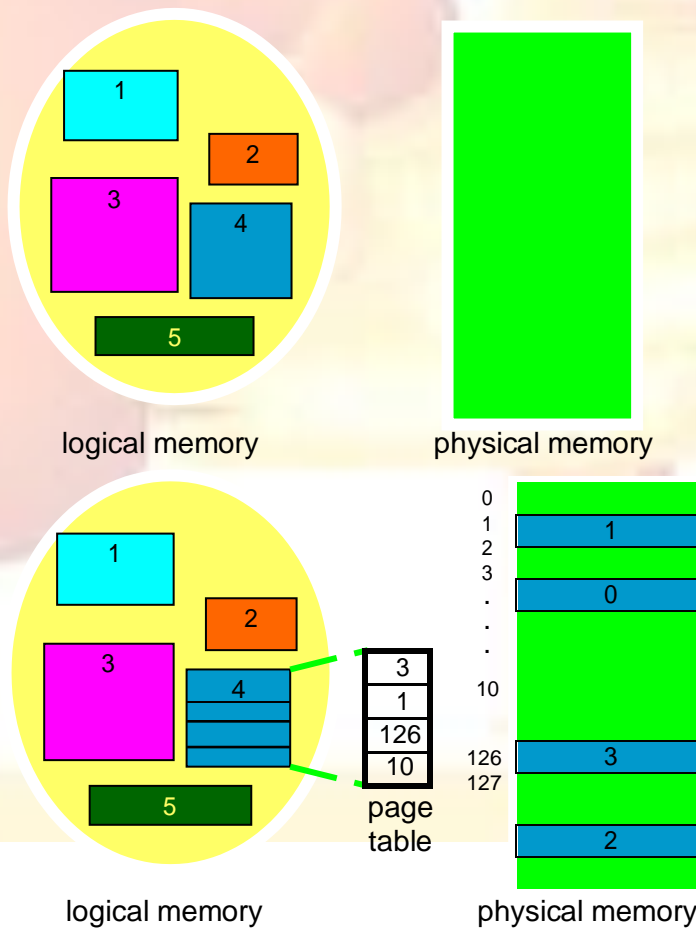
- Chapter 9 of the textbook
- Lecture 36 on Virtual TV

Summary

- Paged segmentation
- Examples of paged segmentation: MULTICS under GE 345 and OS/2, Windows, and Linux under Intel CPUs

Paged Segmentation

In paged segmentation, we divide every segment in a process into fixed size pages. We need to maintain a page table per segment CPU's memory management unit must support both segmentation and paging. The following snapshots illustrate these points.

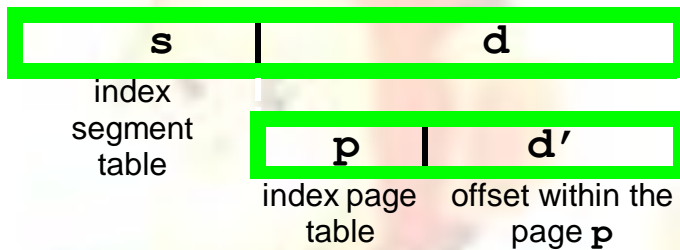


Paged Segmentation

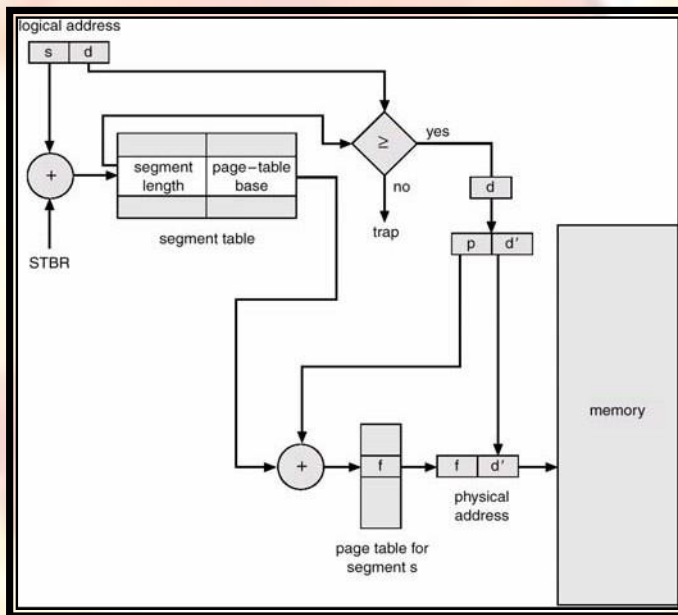
The logical address is still $\langle s, d \rangle$, with s used to index the segment table. Each segment table entry consists of the tuple

$\langle \text{segment-length, page-table-base} \rangle$

The logical address is legal if $d < \text{segment-length}$. The segment offset, d , is partitioned into two parts: p and d' , where p is used to index the page table associated with the segment s and d' is used as offset within the page p . p indexes the page table to retrieve frame, f , and physical address (f, d') is formed. The following diagrams show the format of logical address and its division, and the hardware support needed for logical to physical address translation.



a) Logical address and its partition



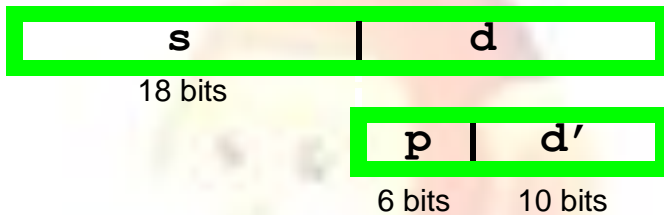
b) Hardware support needed for logical to physical address translation

MULTICS Example

We now take the example of one of the finest operating systems of late 1960s and early 1970s, known as the MULTICS operating system. Here are the specifications of the CPU supported by MULTICS and calculation of its various parameters such as the largest segment size supported by MULTICS.

- GE 345 processor
- Logical address = 34 bits

- Page size = 1 KB
- s is 18 bits and d is 16 bits
- Size of p and d' , largest segment size, and max. number of segments per process?
- Largest segment = 2^d bytes = 2^{16} bytes
- Maximum number of pages per segment = $2^{16} / 1 \text{ K} = 64$
- $|p| = \log_2 64 \text{ bits} = 6 \text{ bits}$
- $|d'| = \log_2 1 \text{ K} = 10 \text{ bits}$
- Maximum number of segments per process = $2^s = 2^{18}$



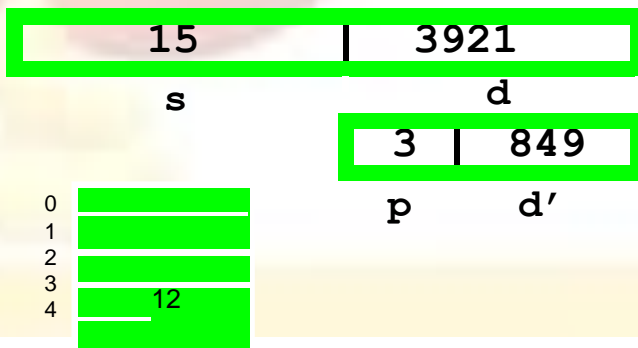
Logical address and its partition for GE645 on which MULTICS ran

Consider a process with its segment 15 having 5096 bytes. The process generates a logical address (15,3921). Is it a legal address? How many pages does the segment have? What page does the logical address refer to? Is it a legal address? Yes

- How many pages does the segment have?
ceiling[5096/1024]= 5
- What page does the logical address refers to?
ceiling[3921/1024]= 4 (i.e., page number 3)
- What are the value of d' and the physical address if page number 3 (i.e., the fourth page) is in frame 12? Here is how we compute these parameters, along with the graphical representation of the various parameters. Logical to physical address translation is shown in the figure on the next page.

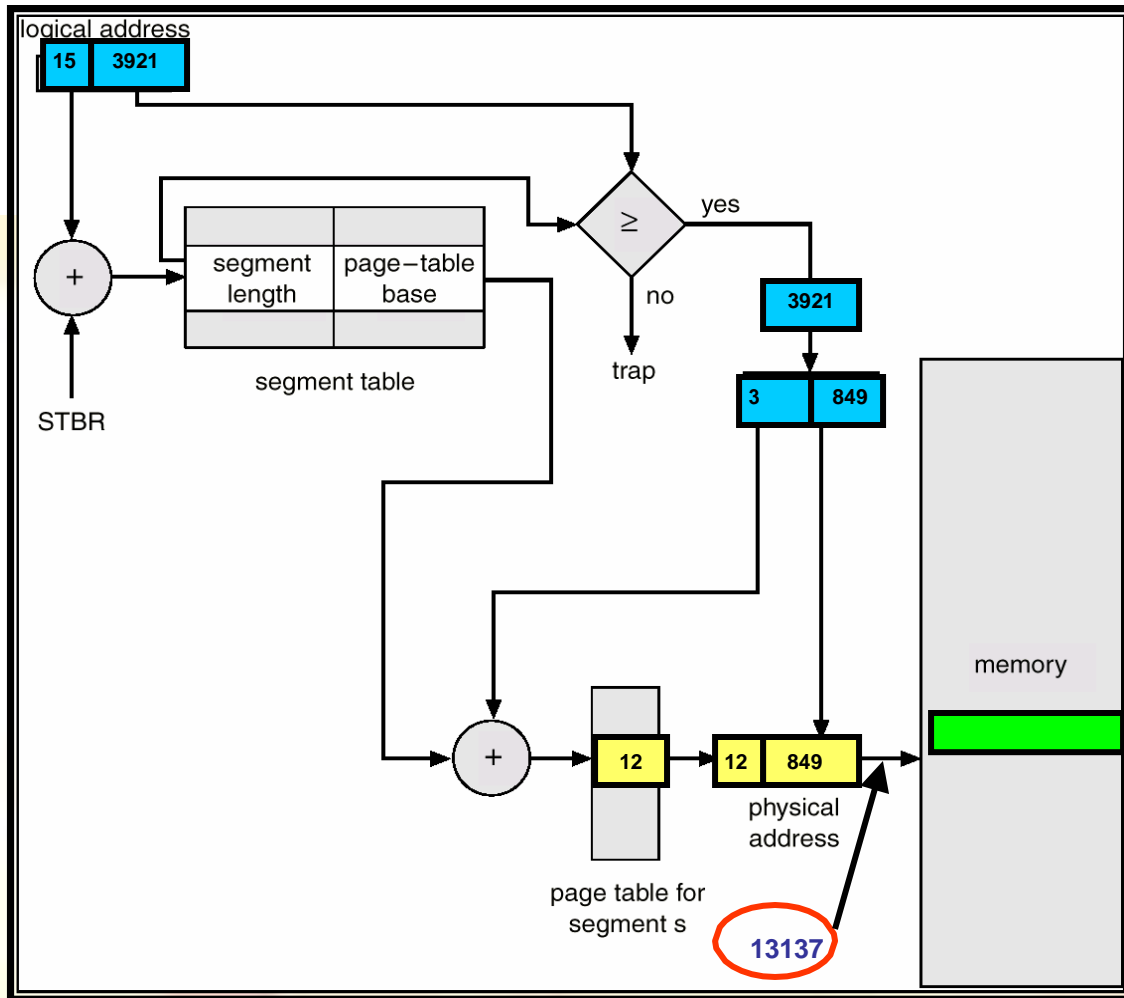
$$d' = 3921 - 3 * 1K = 849$$

$$\text{Physical address} = 12 * 1K + 849 = 13137$$



page table for segment 15

An example related to MULTICS



Intel 80386 Example

- IBM OS/2, Microsoft Windows, and Linux run on it
- Paged segmentation with two-level paging
- Logical address = 48 bits
- 16-bit selector and 32-bit offset
- Page size = 4 KB
- 4-byte page table entry
- 32-entry TLB, covering 32*4K (128 KB) memory ... TLB Reach

16-bit
Selector

32-bit Offset



13-bit Segment #

s

g

p

1-bit field to
specify GDT or
LDT

2-bit field for
specifying the
privilege level

Logical/virtual address and its division for Intel 80386 and higher

Real Mode

20-bit physical address is obtained by shifting left the Selector value by four bits and adding to it the 16-bit effective address.

Operating Systems

Lecture No. 37

Reading Material

- Chapters 9 and 10 of the textbook
- Lecture 37 on Virtual TV

Summary

- Intel 80386 Virtual Memory Support
- Virtual Memory Basic Concept
- Demand Paging
- Page Fault
- Performance of Demand Paging

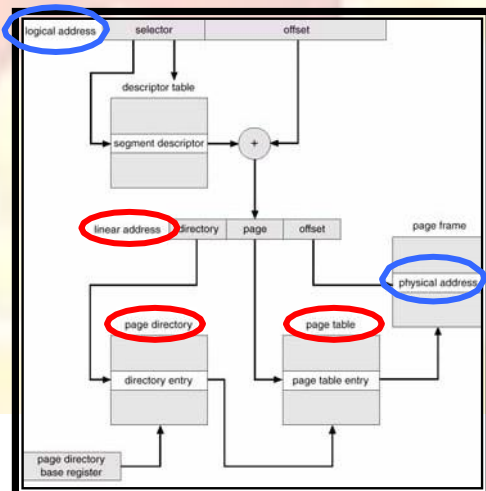
Intel 80386 Virtual Memory Support

We discussed logical to physical address translation in the real mode operation of the Intel 80386 processor in the last lecture. Here we discuss address translation in the protected mode.

Protected Mode

- 248 bytes virtual address space
- 232 bytes linear address space
- Max segment size = 4 GB
- Max segments / process = 16K
- Six CPU registers allow access to six segments at a time
- Selector is used to index a segment descriptor table to obtain an 8-byte segment descriptor entry. Base address and offset are added to get a 32-bit linear address, which is partitioned into p1, p2, and d for supporting 2-level paging.

The following figure shows the hardware support needed for this translation.



Intel 80386 address translation in protected mode

Virtual Memory Basic Concept

An examination of real programs shows that in many cases the existence of the entire program in memory is not necessary:

- Programs often have code to handle unusual error conditions. Since these errors seldom occur in practice, this code is almost never executed.
- Arrays, lists and tables are often allocated more memory than they actually need. An array may be declared 100 by 100 elements even though it is seldom larger than 10 by 10 elements.
- Certain options of a program may be used rarely.

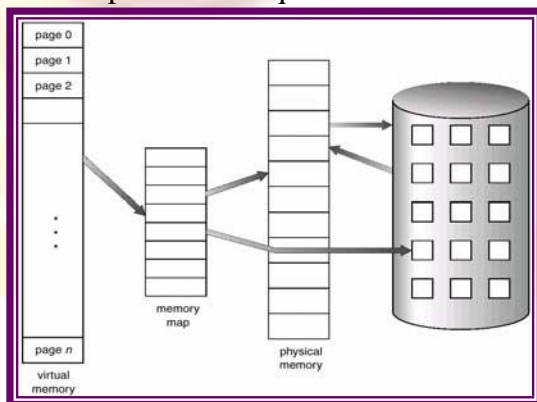
Even in cases where the entire program is needed, it may not be all needed at the same time. The ability to execute a program that is only partially in memory confers many benefits.

- A program would no longer be constrained by the amount of physical memory that is available. Users would be able to write programs for an extremely large virtual address space simplifying the programming task.
- Because each user program could take less physical memory, more programs could be run at the same time, with a corresponding increase in CPU utilization and throughput with no increase in response time or turnaround time.
- Less I/O would be needed to load or swap each user program into memory, so each user program would run faster.

Thus running a program that is not entirely in memory would benefit both the system and the user.

Virtual Memory is the separation of user logical memory from physical memory. This separation allows an extremely large virtual memory to be provided for programmers when only a smaller physical memory is available. Virtual memory makes the task of programming easier because the programmer need not worry about the amount of physical memory, or about what code can be placed in overlays; she can concentrate instead on the problem to be programmed.

In addition to separating logical memory from physical memory, virtual memory also allows files and memory to be shared by several different processes through page sharing. The sharing of pages further allows performance improvements during process creation. Virtual memory is commonly implemented as demand paging. It can also be implemented in a segmentation system. One benefit of virtual memory is efficient process creation. Yet another is the concept of memory mapped files. We will discuss these topics in subsequent lectures.



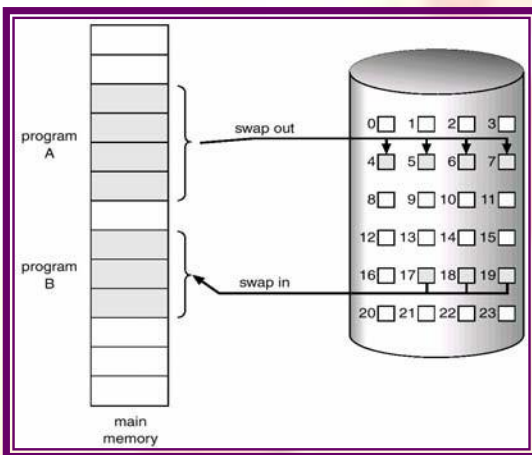
Mapping of logical memory onto physical memory under paging

Demand Paging

A demand paging system is similar to a paging system with swapping. Processes reside on secondary memory (which is usually a disk). When we want to execute a process, we swap it into memory. Rather than swapping the entire process into memory, however we use a lazy swapper. A lazy swapper never swaps a page into memory unless that page will be needed. Since we are now viewing a process as a sequence of pages rather than as one large contiguous address space, use of swap is technically incorrect. A swapper manipulates entire processes, whereas a pager is concerned with the individual pages of a process. Thus the term pager is used in connection with demand paging.

Basic Concepts

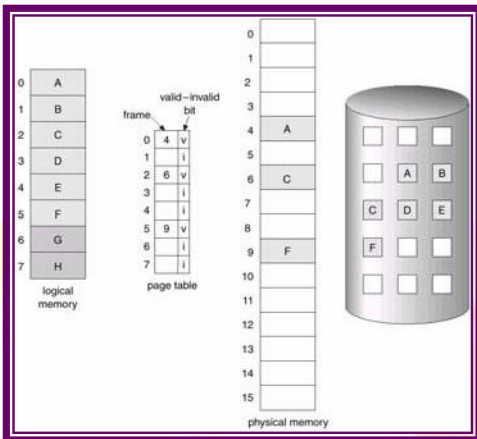
When a process is to be swapped in, the paging software guesses which pages would be used before the process is swapped out again. Instead of swapping in a whole process, the pager brings only those necessary pages into memory. Thus it avoids reading into memory pages that will not be used anyway, decreasing the swap time and the amount of physical memory needed.



Swapping in and out of pages

With this scheme, we need some form of hardware support to distinguish which pages are in memory and which are on disk. The valid-invalid bit scheme described in previous lectures can be used. This time however when the bit is set to valid, this value indicates that the associated page is both legal and in memory. If the bit is set to invalid this value indicates that the page either is invalid or valid but currently on the disk. The page table entry for a page that is brought into memory is set as usual but the page table entry for a page that is currently not in memory is simply marked invalid or contains the address of the page on disk.

Notice that marking a page invalid will have no effect if the process never attempts to access that page. Hence if we guess right and page in all and only those pages that are actually needed, the process will run exactly as though we had brought in all pages. While the process executes and accesses pages that are memory resident, execution proceeds normally.

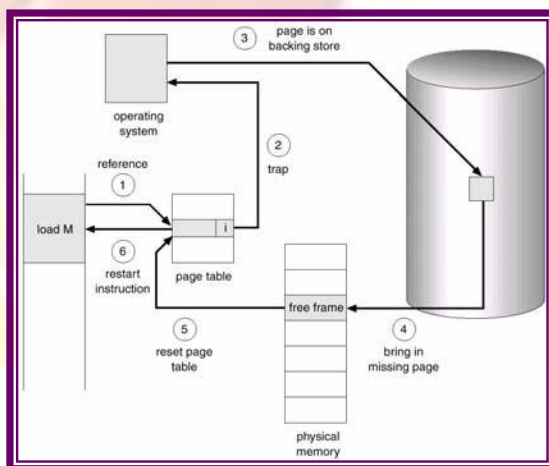


Protection under paging

Page Fault

But what happens if the process tries to access a page that was not brought into memory? Access to a page marked invalid causes a page fault trap. The paging hardware in translating the address through the page table will notice that the invalid bit is set, causing a trap to the operating system. This trap is the result of the operating system's failure to bring the desired page into memory (in an attempt to minimize disk transfer overhead and memory requirements) rather than an invalid address error as a result of an attempt to use an illegal memory address. The procedure for handling a page fault is straightforward:

1. We check an internal table (usually kept with the process control block) for this process to determine whether the reference was valid or invalid memory access.
2. If the reference was invalid we terminate the process. If it was valid, but we have not yet brought in that page, we now page it in.
3. We find a free frame (by taking one from the free-frame list, for example)
4. We schedule a disk operation to read the desired page into the newly allocated frame.
5. When the disk read is complete, we modify the internal table kept with the process and the page table to indicate that the page is now in memory.
6. We restart the instruction that was interrupted by the illegal address trap. The process can now access the page as though it had always been in memory.



Steps needed for servicing a page fault

Since we save the state (registers, condition code, instruction counter) of the interrupted process when the page fault occurs, we can restart the process in exactly the same place and state except that the desired page is now in memory and is accessible. In this way we are able to execute a process even though portions of it are not yet in memory. When the process tries to access locations that are not in memory, the hardware traps the operating system (page fault). The operating system reads the desired into memory and restarts the process as though the page had always been in memory.

In the extreme case, we could start executing a process with no pages in memory. When the operating system sets the instruction pointer to the first instruction of the process, which is on a non memory resident page, the process immediately faults for the page. After this page is brought into memory, the process continues to execute faulting as necessary until every page that it needs is in memory. At that point, it can execute with no more faults. This scheme is called pure demand paging: never bring a page into memory until it is required.

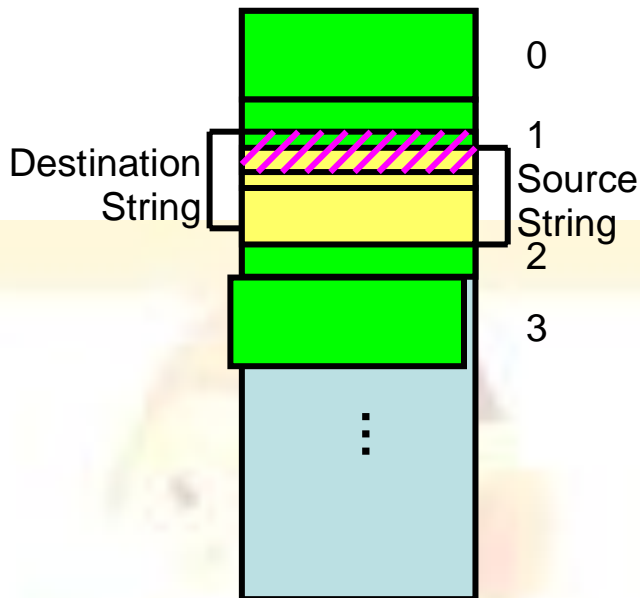
The hardware needed to support demand paging is the same as the hardware for paging and swapping:

- Page table: This table has the ability to mark an entry invalid through a valid-invalid bit or special value of protection bits.
- Secondary memory: This memory holds those pages that are not present in main memory. The secondary memory is usually a high speed disk. It is known as the swap device, and the section of disk used for this purpose is called the swap space.

In addition to this hardware, additional architectural constraints must be imposed. A crucial one is the need to be able to restart any instruction after a page fault. In most cases this is easy to meet, a page fault occurs while we are fetching an operand, we must fetch and decode the instruction again, and then fetch the operand. A similar problem occurs in machines that use special addressing modes, including auto increment and auto decrement modes. These addressing modes use a register as a pointer and automatically increment or decrement the register. Auto decrement automatically decrements the register before using its contents as the operand address; auto increment increments the register after using its contents. Thus the instruction

MOV (R2) +, -(R3)

Copies the contents of the location pointed to by register2 into that pointed to by register3. Now consider what will happen if we get a fault when trying to store into the location pointed to by register3. To restart the instruction we must reset the two registers to the values they had before we started the execution of the instruction.



Execution of a block (string) move instruction causing part of the source to be overwritten before a page fault occurs

Another problem occurs during the execution of a block (string) move instruction. If either source or destination straddles a page boundary a page fault might occur after the move is partially done. In addition if the source and destination blocks overlap the source block may have been modified in which case we cannot simply restart the instruction, as shown in the diagram on the previous page.

Performance of demand paging

Demand paging can have a significant effect on the performance of a computer system. To see why, let us compute the effective access time for a demand paged memory. For most computer systems, the memory access time, denoted m_a now ranges from 10 to 200 nanoseconds. As long as we have no page faults, the effective access time is equal to the memory access time. If, however a page fault occurs, we must first read the relevant page from disk, and then access the desired word.

Let p be the probability of a page fault ($0 \leq p \leq 1$). We would expect p to be close to zero, that is, there will be only a few page faults. The effective access time is then:

$$\text{Effective access time} = (1-p) * m_a + p * \text{page fault time}$$

To compute the effective access time, we must know how much time is needed to service a page fault. A page fault causes the following sequence to occur:

1. Trap to the operating system
2. Save the user registers and process states
3. Determine that the interrupt was a page fault
4. Check that the page reference was legal and determine the location of the page on disk
5. Issue a read from the disk to a free frame:
 - a. Wait in a queue for this device until the read request is serviced
 - b. Wait for the device seek and/or latency time
 - c. Begin the transfer of the page to a free frame
6. While waiting, allocate the CPU to some other user (CU scheduling; optimal)

7. Interrupt from the disk (I/O completed)
8. Save the registers and process state for the other user (if step 6 is executed)
9. Determine that the interrupt was from the disk
10. Correct the page table and other tables to show that the desired page is now in memory
11. Wait for the CPU to be allocated to this process again
12. Restore the user registers, process state and new page table

Not all these steps are necessary in every case. For example we are assuming that in step 6, the CPU is allocated to another process while the I/O occurs. This arrangement allows multiprogramming to maintain CPU utilization, but requires additional time to resume the page fault service routine when the I/O transfer is complete.

In any case we are faced with three major components of the page fault service time:

1. Service the page fault interrupt
2. Read in the page
3. Restart the process

The first and third tasks may be reduced, with careful coding, to several hundred instructions. These tasks may take from 1 to 100 microseconds each. The page switch time, on the other hand, will probably be close to 24 milliseconds. A typical hard disk has an average latency of 8 milliseconds, a seek of 15 milliseconds, and a transfer time of 1 millisecond. Thus, the total paging time would be close to 25 milliseconds, including hardware and software time. Remember that we are looking at only the device service time. If a queue of processes is waiting for the device we have to add device queuing time as we wait for the paging device to be free to service our request, increasing even more the time to swap.

If we take an average page fault service time of 25 milliseconds and a memory access time of 100 nanoseconds, then the effective access time in nanoseconds is

$$\begin{aligned} \text{Effective access time} &= (1-p) * (100) + p (25 \text{ milliseconds}) \\ &= (1-p) * 100 + p * 25,000,000 \\ &= 100 + 24,999,900 * p \end{aligned}$$

We see then that the effective access time is directly proportional to the page fault rate. If one access out of 1,000 causes a page fault, the effective access time is 25 microseconds. The computer would be slowed down by a factor of 250 because of demand paging! If we want less than 10 percent degradation, we need:


$$\begin{aligned} 110 &> 100 + 25,000,000 * p \\ 10 &> 25,000,000 * p \\ p &< 0.0000004 \end{aligned}$$

That is, to keep the slowdown due to paging to a reasonable level, we can allow only less than one memory access out of 2,500,000 to page fault.

It is important to keep the slowdown due to paging to a reasonable level, we can allow only less than one memory access out of 2,500,000 to page fault.

It is important to keep the page fault rate low in a demand-paging system. Otherwise the effective access time increases, slowing process execution dramatically.

One additional aspect of demand paging is the handling and overall use of swap space. Disk I/O to swap space is generally faster than that to the file system. It is faster because swap space is allocated in much larger blocks, and file lookups and indirect allocation methods are not used. It is therefore possible for the system to gain better paging throughput by copying an entire file image into the swap space at process startup,



and then performing demand paging from the swap space. Another option is to demand pages from the file system initially, but to write the pages to swap space as they are replaced. This approach will ensure that only needed pages are ever read from the file system, but all subsequent paging is done from swap space.

Some systems attempt to limit the amount of swap space when binary files are used. Demand pages for such files are brought directly from the file system. However, when page replacement is called for, these pages can simply be overwritten and read in from the file system again if ever needed. Using this approach, the file system itself serves as the backing store. However swap space must still be used for pages not associated with a file; these pages include the stack and heap for a process. This technique is used in several systems including Solaris.

Operating Systems

Lecture No. 38

Reading Material

- Chapter 10 of the textbook
- Lecture 38 on Virtual TV

Summary

- Performance of Demand Paging
- Process Creation
- Memory Mapped Files

Performance of Demand Paging with Page Replacement

When there is no free frame available, page replacement is required, and we must select the pages to be replaced. This can be done via several replacement algorithms, and the major criterion in the selection of a particular algorithm is that we want to minimize the number of page faults. The victim page that is selected depends on the algorithm used, it might be the least recently used page, or the most frequently used etc depending on the algorithm.

Another Example

- Effective memory access is 100 ns
- Page fault overhead is 100 microseconds = 105 ns
- Page swap time is 10 milliseconds = 107 ns
- 50% of the time the page to be replaced is “dirty”
- Restart overhead is 20 microseconds = 2 x 104 ns

$$\begin{aligned} \text{Effective access time} &= 100 * (1-p) + (105 + 2 * 104 + 0.5 * 107 + 0.5 * 2 * 107) * p \\ &= 100 * (1-p) + 15,120,000 * p \end{aligned}$$

What is a Good Page Fault Rate?

For the previous example suppose p is 1%, then EAT is

$$\begin{aligned} &= 100 * (1-p) + 15,120,000 * p \\ &= 151299 \text{ ns} \end{aligned}$$

Thus a slowdown of $151299 / 100 = 1513$ occurs.

For the luxury of virtual memory to cost only 20% overhead, we need

$$\begin{aligned} 120 &> 100 * (1-p) + 15,120,000 * p \\ 120 &> 100 - 100p + 15,120,000p \\ p &< 0.00000132 \end{aligned}$$

⇒ Less than one page fault for every 755995 memory accesses!

Process Creation and Virtual Memory

Paging and virtual memory provide other benefits during process creation, such as copy on write and memory mapped files.

Copy on Write `fork()`

Demand paging is used when reading a file from disk into memory and such files may include binary executables. However, process creation using `fork()` may bypass initially the need for demand paging by using a technique similar to page sharing. This technique provides for rapid process creation and minimizes the number of new pages that must be allocated to newly created processes.

Recall the `fork()` system call creates a child process as a duplicate of its parent. Traditionally `fork()` worked by creating a copy of the parent's address space for the child, duplicating the pages belonging to the parent. However, considering that many child processes invoke the `exec()` system call immediately after creation, the copying of the parent's address space may be unnecessary. Alternatively we can use a technique known as copy on write. This works by allowing the parent and child processes to initially share the same pages. These shared pages are marked as copy-on-write pages, meaning that if either process writes to a shared page, a copy of the shared page is created. For example assume a child process attempts to modify a page containing portions of the stack; the operating system recognizes this as a copy-on-write page. The operating system will then create a copy of this page mapping it to the address space of the child process. Therefore the child page will modify its copied page, and not the page belonging to the parent process. Using the copy-on-write technique it is obvious that only the pages that are modified by either process are copied; all non modified pages may be shared by the parent and the child processes. Note that only pages that may be modified are marked as copy-on-write. Pages that cannot be modified (i.e. pages containing executable code) may be shared by the parent and the child. Copy-on-write is a common technique used by several operating systems such as Linux, Solaris 2 and Windows 2000.

When it is determined a page is going to be duplicated using copy-on-write it is important to note where the free page will be allocated from. Many operating systems provide a pool of free pages for such requests. These free pages are typically allocated when the stack or heap for a process must expand or for managing copy-on-write pages. Operating systems typically allocate these pages using a technique known as zero-fill-on-demand. Zero-fill-on-demand pages have been zeroed out before allocating, thus deleting the previous contents on the page. With copy-on-write the page being copied will be copied to a zero-filled page. Pages allocated for the stack or heap are similarly assigned zero-filled pages.

`vfork()`

Several versions of UNIX provide a variation of the `fork()` system call—`vfork()` (for virtual memory fork). `vfork()` operates differently than `fork()` with copy on write. With `vfork()` the parent process is suspended and the child process uses the address space of the parent. Because `vfork()` does not use copy-on-write, if the child process changes any pages of the parent's address space, the altered pages will be visible to the parent once it resumes. Therefore, `vfork()` must be used with caution, ensuring that the child process does not modify the address space of the parent. `vfork()` is intended to be used when the

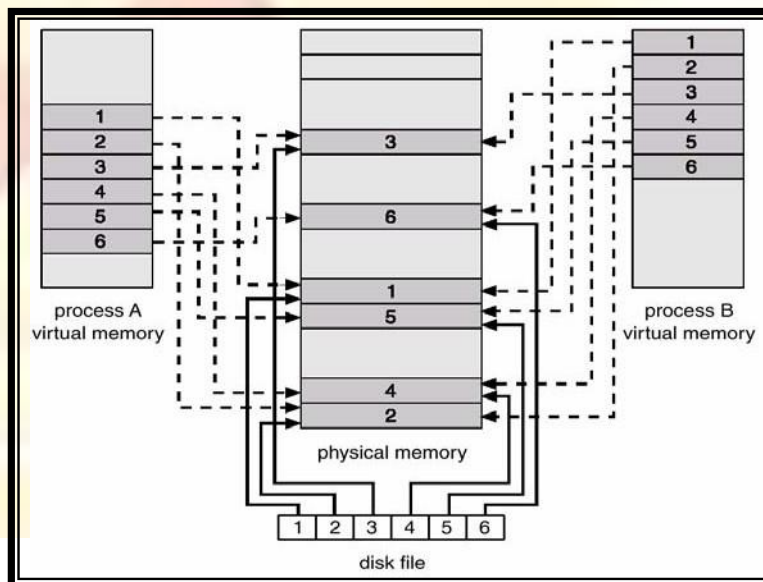
child process calls `exec()` immediately after creation. Because no copying of pages takes place, `vfork()` is an extremely efficient method of process creation and is sometimes used to implement UNIX command-line shell interfaces.

Linux Implementation

In Linux, shared pages are marked read-only after `fork()`. If either process tries to modify a shared page, a page fault occurs and the page is copied. The other process (who later faults on write) discovers it is the only owner; so no copying takes place. In other words, Linux implementation of `fork()` is based on the “copy-on-write” semantics.

Memory Mapped files

Consider a sequential read of a file on disk using the standard system calls `open()`, `read()`, `write()`. Every time the file is accessed requires a system call and disk access. Alternatively we can use the virtual memory techniques discussed so far to treat file I/O as routine memory accesses. This approach is known as memory mapping a file, allowing a part of the virtual address space to be logically associated with a file. Memory mapping a file is possible by mapping a disk block to a page (or pages) in memory. Initial access to the file proceeds using ordinary demand paging resulting in a page fault. However, a page sized portion of the file is read from the file system into a physical page. Subsequent reads and writes to the file are handled as routine memory accesses, thereby simplifying file access and usage by allowing file manipulation through memory rather than the overhead of using the `read()` and `write()` system calls. Note that writes to the file mapped in memory may not be immediate writes to the file on disk. Some systems may choose to update the physical file when the operating system periodically checks if the page in memory mapping the file has been modified. Closing the file results in all the memory mapped data being written back to disk and removed from the virtual memory of the process. The concept of memory mapped files is shown pictorially in the following diagram.



Memory mapped files

Memory-Mapped Files in Solaris 2

Some operating systems provide memory mapping only through a specific system call and treat all other file I/O using the standard system calls. However, some systems choose to memory map a file regardless of whether a file was specified as a memory map or not. For example: Solaris 2 treats all file I/O as memory mapped, allowing file access to take place in memory, whether a file has been specified as memory mapped using `mmap()` system call or not.

Multiple processes may be allowed to map the same file into the virtual memory of each to allow sharing of data. Writes by any of the processes modify the data in virtual memory and can be seen by all others that map the same section of the file. Given our knowledge of virtual memory it should be clear how the sharing of memory mapped sections of memory is implemented. The virtual memory map of each sharing process points to the same page of physical memory – the page that holds a copy of the disk block. This memory mapping is illustrated as:

The memory mapping system calls can only support copy-on-write functionality allowing processes to share a file in read-only mode, but to have their own copies of any data they modify. So that access to the shared data is coordinated, the processes involved might use one of the mechanisms for achieving mutual exclusion.

`mmap()` System Call

In a UNIX system, `mmap()` system call can be used to request the operating system to memory map an opened file. The following code snippets show “normal” way of doing file I/O and file I/O with memory mapped files.

“Normal” File I/O

```
fildes = open(...);  
lseek(...);  
read(fildes, buf, len);  
/* use data in buf */
```

File I/O with `mmap()`

```
fildes = open(...)  
address = mmap((caddr_t) 0, len, (PROT_READ | PROT_WRITE), MAP_PRIVATE, fildes, offset);  
/* use data at address */
```

Operating Systems

Lecture No. 39

Reading Material

- Chapter 10 of the textbook
- Lecture 39 on Virtual TV

Summary

- Page replacement (basic concept and replacement algorithms)

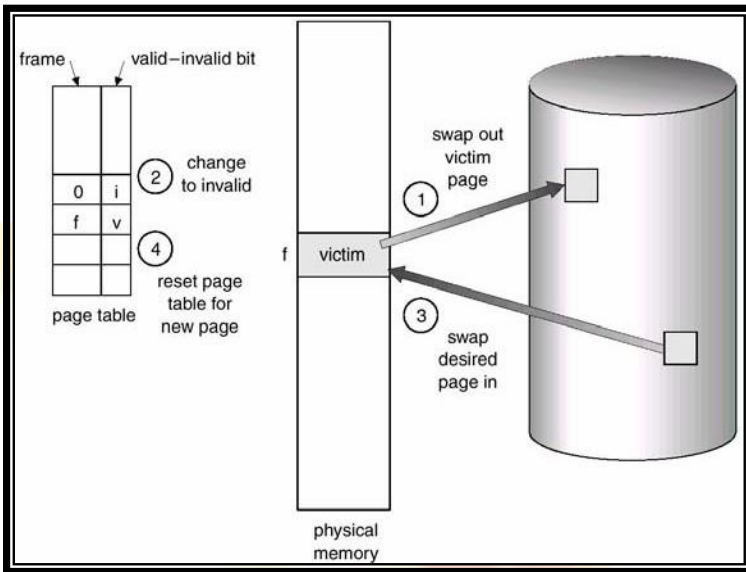
Page replacement

While a user process is executing, a page fault occurs. The hardware traps to the operating system, which checks its internal tables to see that this page is a genuine one rather than an illegal memory access. The operating system determines where the desired page is residing on the disk, but then finds that there are no free frames on the free frame list: All memory is in use.

The operating system has several options at this point. It could terminate the user process. However, demand paging is the operating system's attempt to improve the computer system's utilization and throughput. Users should not be aware that their processes are running on a paged system – paging should be logically transparent to the user. So this option is not the best choice. The operating system could swap out a process, but that would reduce the level of multiprogramming. So we explore page replacement. This means that if no free frame is available on a page fault, we replace a page in memory to load the desired page. The page-fault service routine is modified to include page replacement. We can free a frame by writing its contents to swap space, and changing the page table to indicate that the page is no longer in memory. The modified page fault service routine is:

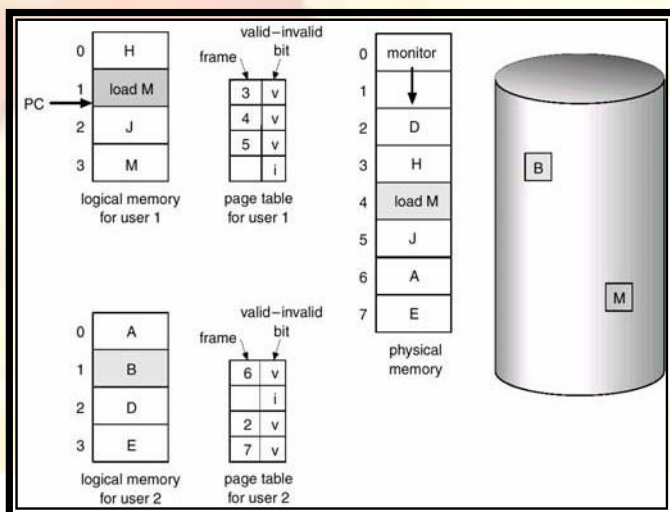
1. Find the location of the desired page on the disk
2. Find a free frame
 - a) If there is a free frame use it.
 - b) If there is no free frame, use a page replacement algorithm to select a victim frame.
3. Read the desired page into the newly freed frame; change the page and frame tables.
4. Restart the user process.

The following diagram shows these steps pictorially.



Steps needed for page replacement

We can reduce overhead by using a *modify* bit (or *dirty* bit). Each page or frame may have a modify bit associated with it in hardware. The modify bit is set by the hardware whenever any word or byte in the page is written into, indicating that the page has been modified. When we select a page for replacement we examine its modify bit. If the bit is set, we know that the page has been modified since it was read in from the disk. In this case we must write that page to the disk. If the modify bit is not set however, the page has not been modified since it was read into memory, and hence we can avoid writing that page to disk. In the following figure we show two processes with four pages each, main memory having eight frames, with two used for resident part of operating system (leaving six frames for user processes). Both processes have three of their pages in memory and therefore there is no free frame. When the upper process (user 1) tries to access its fourth page (page number 3), a page fault is caused and page replacement is needed.

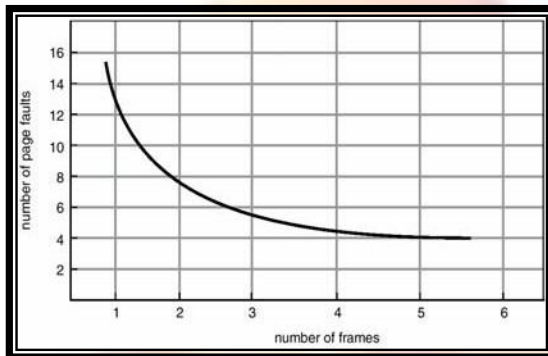


Page fault and page replacement

Page Replacement Algorithms

In general we want a page replacement algorithm with the lowest page-fault rate. We evaluate an algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string.

To determine the number of page faults for a particular reference string and page replacement algorithm, we also need to know the number of page frames available. Obviously as the number of frames available increases, the number of page faults decreases.



Expected relationship between number of free frames allocated to a process and the number of page faults caused by it

FIFO Page Replacement

The simplest page-replacement algorithm is a FIFO algorithm. A FIFO replacement algorithm associates with each page the time when that page was brought into memory. When a page must be replaced, the oldest page is chosen. Notice that it is not strictly necessary to record the time when a page is brought in. We can create a FIFO queue to hold all pages in memory. We replace the page at the head of the queue. When a page is brought into memory we insert it at the tail of the queue.

Consider the following example, in which the number of frames allocated is 4, and the reference string is 1, 2, 3, 4, 5, 1, 6, 7, 8, 7, 8, 9, 5, 4, 5, 4, 4. The number of page faults caused by the process is nine, as shown below.

1, 2, 3, 4, 5, 3, 4, 1, 6, 7, 8, 7, 8, 9, 5, 4, 5, 4, 4

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | 5 | 5 | 5 | 5 | 8 | 8 | 8 | 8 |
| 2 | 2 | 1 | 1 | 1 | 1 | 9 | 9 | 9 |
| 3 | 3 | 3 | 6 | 6 | 6 | 6 | 5 | 5 |
| 4 | 4 | 4 | 4 | 7 | 7 | 7 | 7 | 4 |

Example for the FIFO page replacement algorithm

The problem with this algorithm is that it suffers from Belady's anomaly: For some page replacement algorithms the page fault rate may increase as the number of allocated

frames increases, whereas we would expect that giving more memory to a process would improve its performance.

Optimal Algorithm

An optimal page-replacement algorithm has the lowest page fault rate of all algorithms, and will never suffer from the Belay's algorithm. This algorithm is simply to replace the page that will not be used for the longest period of time. Use of this algorithm guarantees the lowest possible page-fault rate for a fixed number of frames. In case of the following example (which uses the same replacement string as the example for the FIFO algorithm), the number of page faults caused by the process is seven.

1, 2, 3, 4, 5, 3, 4, 1, 6, 7, 8, 7, 8, 9, 5, 4, 5, 4, 4

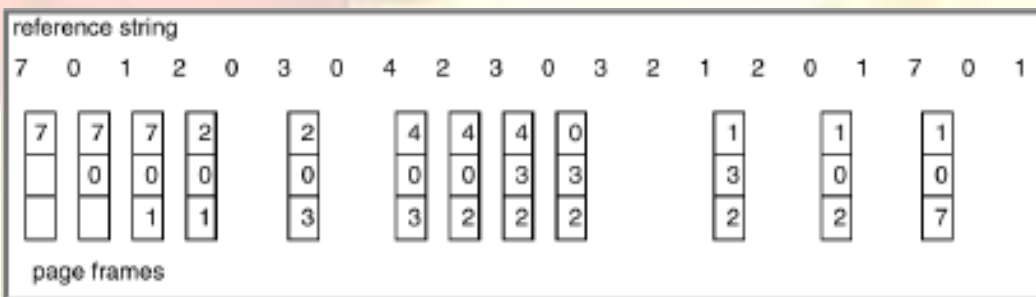
| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 1 | 6 | 6 | 8 | 9 | 9 |
| 2 | 5 | 5 | 5 | 5 | 5 | 5 |
| 3 | 3 | 3 | 7 | 7 | 7 | 7 |
| 4 | 4 | 4 | 4 | 4 | 4 | 4 |

Example for the optimal page replacement algorithm

Unfortunately this algorithm is difficult to implement because it requires future knowledge of the reference string. As a result this algorithm is used mainly for comparison.

LRU Page Replacement

If we use the recent past as an approximation of the near future, then we will replace the page that has not been used for the longest period of time. This approach is the least recently used algorithms. The following example illustrates the working of LRU algorithm.



Example for the LRU page replacement algorithm

Here is another example, which uses the same reference string as used in the examples for the FIFO and optimal replacement algorithms. The number of page faults in this case is nine.

1, 2, 3, 4, 5, 3, 4, 1, 6, 7, 8, 7, 8, 9, 5, 4, 5, 4, 4

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | 5 | 5 | 6 | 6 | 6 | 6 | 5 | 5 |
| 2 | 2 | 1 | 1 | 1 | 1 | 9 | 9 | 9 |
| 3 | 3 | 3 | 3 | 7 | 7 | 7 | 7 | 4 |
| 4 | 4 | 4 | 4 | 4 | 8 | 8 | 8 | 8 |

Another example for the LRU algorithm

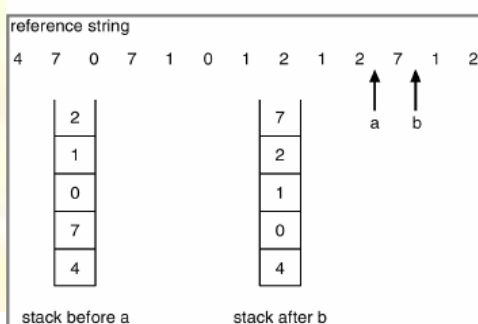
An LRU page replacement may require substantial hardware assistance. The problem is to determine an order for the frames defined by the time of last use. Two implementations are feasible:

Counter-based Implementation of LRU

In the simplest case we associate with each page table entry a time-of-use field and add to the CPU a logical clock or counter. The clock is incremented for every memory reference. Whenever a reference to a page is made, the contents of the clock register are copied to the time-of-use field in the page entry for that page. In that way we always have the time of the last reference to each page. We replace the page that has the smallest time value. This scheme requires a search of the page table to find the LRU page and a write to memory for each memory access. The times must also be maintained when page tables are changed. Overflow of the clock must be considered.

Stack-based Implementation of LRU

Another approach to implementing the LRU algorithm is to keep a stack of page numbers. Whenever a page is referenced, it is removed from the stack and put on top. In this way, the top of the stack is always the most recently used page and the bottom is the LRU page. Because entities must be removed from the middle of the stack, it is best implementing by a doubly linked list with a head and tail pointer. Removing a page and putting it on the top of the stack then requires changing six pointers at worst. Each update is a little more expensive, but there is no search for a replacement the tail pointer points to the bottom of the stack which is the LRU page. The following diagram shows the working of stack-based implementation of the LRU algorithm.



Stack based implementation of the LRU page replacement algorithm

Operating Systems

Lecture No. 40

Reading Material

- Chapter 10 of the textbook
- Lecture 40 on Virtual TV

Summary

- Belady's Anomaly
- Page Replacement Algorithms
 - Least Frequently Used (LFU)
 - Most Frequently Used (MFU)
 - Page Buffering Algorithm
- Allocation of Frames
- Minimum Number of Frames
- Thrashing

Belady's Anomaly

Consider the following example of the FIFO algorithm.

- Number of frames allocated = 3
- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- Number of page faults = 9

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 4 | 4 | 4 | 5 | 5 | 5 |
| | 2 | 2 | 2 | 1 | 1 | 1 | 3 | 3 |
| | | 3 | 3 | 3 | 2 | 2 | 2 | 4 |

Now an intuitive idea is that if we increase the number of frames allocated to 4 from 3, the page faults should decrease, but the following example demonstrates otherwise.

- Number of frames allocated = 4
- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- Number of page faults = 10

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 5 | 5 | 5 | 5 | 4 | 4 |
| | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 5 |
| | | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 |
| | | | 4 | 4 | 4 | 4 | 3 | 3 | 3 |

This is due to the **Belady's Anomaly** which states that "For some page replacement algorithms, the page fault rate may increase as the number of allocated frames increases."



Belady's anomaly

Stack Replacement Algorithms

These are a class of page replacement algorithms with the following property:

Set of pages in the main memory with n frames is a subset of the set of pages in memory with $n+1$ frames.

These algorithms do not suffer from Belady's Anomaly. An example is the LRU algorithm.

Consider the following example which shows that LRU does not suffer from Belady's anomaly for the given reference string.

- Number of frames allocated = 3
- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- Number of page faults = 10

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 4 | 4 | 4 | 5 | 3 | 3 | 3 |
| | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 4 | 4 |
| | | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 5 |

- Number of frames allocated = 4
- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- Number of page faults = 8

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 5 |
| | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| | | 3 | 3 | 5 | 5 | 4 | 4 |
| | | | 4 | 4 | 3 | 3 | 3 |

LRU Approximation Algorithm

Few computer systems provide sufficient hardware support for true LRU page replacement. Some systems provide no hardware support and other page replacement algorithms must be used. Many systems provide some help however, in the form of a reference bit. The reference bit for a page is set by the hardware whenever that page is referenced. Reference bits are associated with each entry in the page table.

Initially all bits are cleared by the operating system. As a user process executes the bit associated with each page referenced is set to 1 by the hardware. After some time we can determine which pages have been used and which have not been used by examining the reference bits. We do not know the order of use however, but we know which pages were used and which were not used.

Least frequently used algorithm

This algorithm is based on the locality of reference concept—the least frequently used page is not in the current locality. LFU requires that the page with the smallest count be replaced. The reason for this selection is that an actively used page should have a large reference count. This algorithm suffers from the situation in which a page is used heavily during the initial phase of a process, but then is never used again. Since it was used heavily it has a large count and remains in memory even though it is no longer needed. One solution is to shift the counts right by 1 bit at regular intervals, forming an exponentially decaying average user count.

Most Frequently Used

The MFU page replacement algorithm is based on the argument that the page with the smallest count was probably just brought in and has yet to be used; it will be in the **locality** that has just started.

Page Buffering Algorithm

The OS may keep a pool of free frames. When a page fault occurs a victim page is chosen as before. However the desired page is read into a free frame from the pool before the victim is written out. This allows the process to restart as soon as possible, without waiting for the victim to be written out. When the victim is later written out, its frame is added to the free frame pool. Thus a process in need can be given a frame quickly and while victims are selected, free frames are added to the pool in the background

An expansion of this idea is to maintain a list of modified pages. Whenever the paging device is idle, a modified page is selected and is written to disk. Its modify bit is then reset. This scheme increases the probability that a page will be clean when it is selected for replacement and will not need to be written out.

Another modification is to keep a pool of free frames, but to remember which page was in which frame. Since the frame contents are not modified when a frame is written to disk, the old page can be reused directly from the free-frame pool if it is needed before that frame is reused. No I/O is needed in this case. When a page fault occurs we check whether the desired page is in the free-frame pool. If it is not we must select a free frame and read into it. This method is used together with FIFO replacement in the VAX/VMS operating system.

Local vs Global Replacement

If process P generates a page fault, page can be selected in two ways:

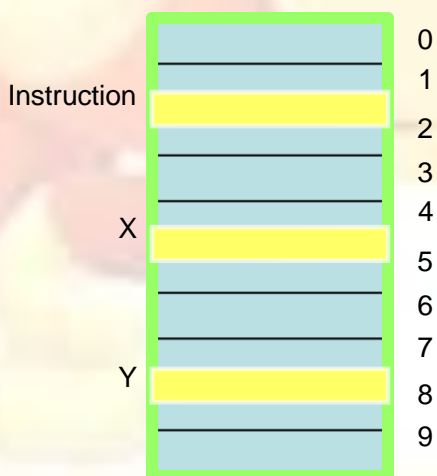
- Select for replacement one of its frames.
- Select for replacement a frame from a process with lower priority number.

Global replacement allows a process to select a replacement frame from the set of all frames, even if that frame belongs to some other process; one process can take a frame from another. Local replacement requires that each process select from only its allocated frames.

Consider an allocation scheme where we allow high priority processes to select frames from low priority processes for replacement. A process can select a replacement from among its own frames or the frames of any lower priority process. This approach allows a high priority process to increase its frame allocation at the expense of the low priority process.

Allocation of frames

Each process needs a minimum number of frames so that its execution may be guaranteed on a given machine. Let's consider the MOV X,Y instruction. The instruction is 6 bytes long (16-bit offsets) and might span 2 pages. Also, two pages to handle source and two pages are required to handle destination (assuming 16-bit source and destination).



Minimum frames required to guarantee execution of the MOV X,Y instruction

There are three major allocation schemes:

- **Fixed allocation**
In this scheme free frames are equally divided among processes
- **Proportional Allocation**
Number of frames allocated to a process is proportional to its size in this scheme.

- **Priority allocation**

Priority-based proportional allocation

Here is an example of frame allocation:

Number of free frames = 64

Number of processes = 3

Process sizes: P1 = 10 pages; P2 = 40 pages; P3 = 127 pages

- **Fixed allocation**

$64/3 = 21$ frames per process and one put in the free frames list

- **Proportional Allocation**

- s_i = Size of process P_i

- $S = \sum s_i$

- m = Number of free frames

- a_i = Allocation for $P_i = (s_i / S) * m$

- $a_1 = (10 / 177) * 64 = 3$ frames

- $a_2 = (40 / 177) * 64 = 14$ frames

- $a_3 = (127 / 177) * 64 = 45$ frames

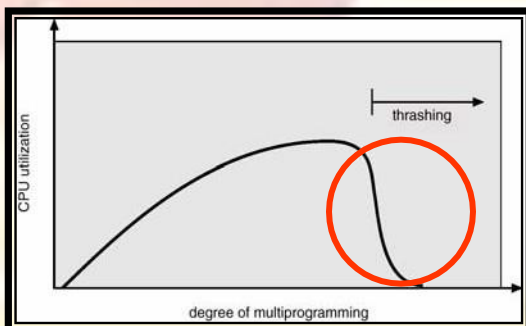
- Two free frames are put in the list of free frames

Thrashing

If a process does not have “enough” pages, the page-fault rate is very high. This leads to low CPU utilization. The operating system thinks that it needs to increase the degree of multiprogramming, because it monitors CPU utilization and find it to be decreasing due to page faults. Thus another process is added to the system and hence thrashing occurs and causes throughput to plunge.

A process is **thrashing** if it is spending more time paging (i.e., swapping pages in and out) than executing. Thrashing results in severe performance problems:

- Low CPU utilization
- High disk utilization
- Low utilization of other I/O devices



Thrashing

The figure shows that as the degree of multiprogramming increases CPU utilization also increases, although more slowly, until a maximum is reached. If the degree of multiprogramming is increased further, thrashing sets in and CPU utilization drops sharply. At this point we must decrease the degree of multiprogramming. We can limit

the effects of thrashing by using a local replacement scheme. With local replacement if one process starts thrashing it cannot steal frames from another process and cause the latter to thrash also. Pages are replaced with regard to the process of which they are a part. Hence local page replacement prevents thrashing to spread among several processes. However if processes are thrashing, they will be in the queue for the paging device most of the time. The average service time for a page fault will increase and effective access time will increase even for a process that is not thrashing.



Operating Systems

Lecture No. 41

Reading Material

- Chapter 10 of the textbook
- Lecture 41 on Virtual TV

Summary

- Thrashing
- The Working Set Model
- Page Fault Frequency Model
- Other Considerations
 - Prepaging
 - Page size
 - Program structure
- Examples of Virtual Memory Systems

Thrashing

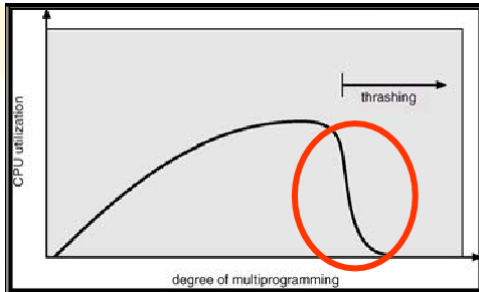
If a process does not have enough frames, it will quickly page fault. At this point, if a free frame is not available, one of its pages must be replaced so that the desired page can be loaded into the newly vacated frame. However since all its pages are in active use, the replaced page will be needed right away. Consequently it quickly faults again and again. The process continues to fault, replacing pages for which it then faults and brings back in right away. This high paging activity is called **thrashing**. In this case, *only one process is thrashing*. A process is thrashing if it is spending more time paging than executing.

Thrashing results on severe performance problems. The operating system monitors CPU utilization and, if CPU utilization is too low, the operating system increases the degree of multiprogramming by introducing one or more new processes to the system. This decreases the number of frames allocated to each process currently in the system, causing more page faults and further decreasing the CPU utilization. This causes the operating system to introduce more processes into the system. As a result CPU utilization drops even further and the CPU scheduler tries to increase the degree of multiprogramming even more. Thrashing has occurred and system throughput plunges. The page fault rate increases tremendously. As a result the effective memory access time increases. Along with low CPU utilization, there is high disk utilization. There is low utilization of other I/O devices. No work is getting done, because the processes are spending all their time paging and the system spend most of its time servicing page fault. Now *the whole system is thrashing*—the CPU utilization plunges to almost zero, the paging disk utilization becomes very high, and utilization of other I/O devices becomes very low.

If a global page replacement algorithm is used, it replaces pages with no regard to the process to which they belong. Now suppose that a process enters a phase in its execution and needs more frames. It starts faulting and taking frames away from other processes. These processes need those pages however and so they also fault taking frames away

from other processes. These faulting processes must use the paging device to swap pages in and out. As they queue up for the paging device, the ready queue empties. As processes wait for the paging device, CPU utilization decreases.

The following graph shows the relationship between the degree of multiprogramming and CPU utilization.

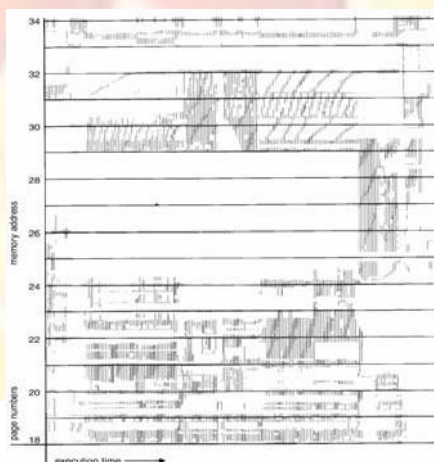


Relationship between the degree of multiprogramming and CPU utilization

Thus in order to stop thrashing, the degree of multiprogramming needs to be reduced. The effects of thrashing can be reduced by using a local page replacement. With local replacement if one process starts thrashing it cannot steal frames from another process and cause the latter to thrash also. Pages are replaced with regard to the process if which they are a part. However, if processes are thrashing they will be in the queue for the paging device most of the time. The average service time for a page fault will increase due to the longer average queue for the paging device. Thus the effective access time will increase even for a process that is not thrashing, since a thrashing process is consuming more resources

Locality of Reference

The locality model states that as a process executes it moves from locality to locality. A locality is a set of pages that are actively used together. A program is generally composed of several different localities, which may overlap. The following diagram shows execution trace of a process, showing localities of references during the execution of the process.

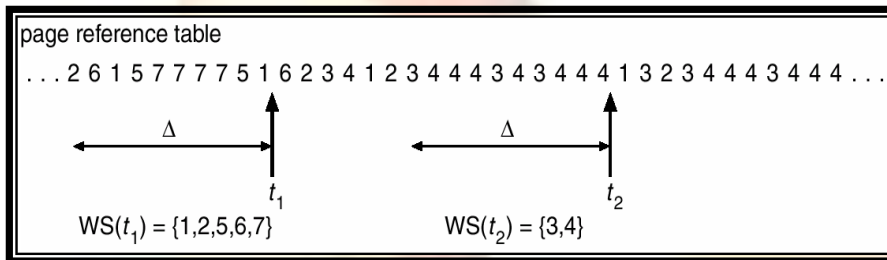


Process execution and localities of reference

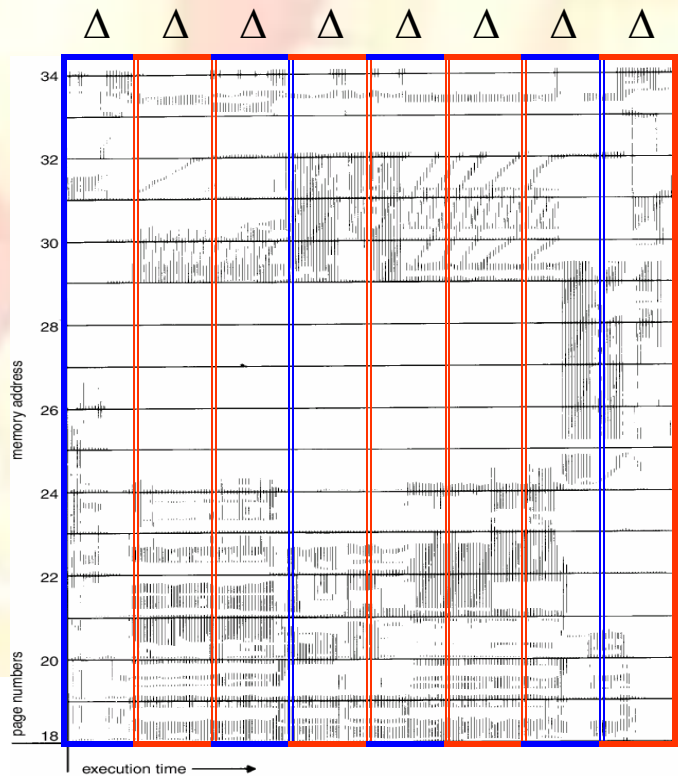
Working Set Model

The working set model is based on the assumption of locality. This model uses a parameter Δ to define the working set window. The idea is to examine the most recent Δ page references. The set of pages in the most recent Δ page references is called the working set. If a page is in active use it will be in the working set. If it no longer being used it will drop from the working set Δ time units after its last reference. Thus the working set is an approximation of the program's locality.

In the following example, we use a value of Δ to be 10 and identify two localities of reference, one having five pages and the other having two pages.



We now identify various localities in the process execution trace given in the previous section. Here are the first two and last localities are: $L1 = \{18-26, 31-34\}$, $L2 = \{18-23, 29-31, 34\}$, and Last = $\{18-20, 24-34\}$. Note that in the last locality, pages 18-20 are referenced right in the beginning only and are effectively out of the locality.



Process execution trace and localities of reference

The accuracy of the working set model depends on the selection of Δ . If Δ is too small, it will not encompass the entire locality; if Δ is too large, it may overlap several localities. In the extreme if Δ is infinite, the working set is the set of pages touched during the process execution. The most important property of the working set is its size. If we compute the working set size, WSS_i for each process in the system we can consider

$$D = \sum WSS_i$$

where, D is the total demand for frames. Each process is actively using the pages in its working set. Thus, process i needs WSS_i frames. If the total demand is greater than the total number of frames ($D > m$), thrashing will occur, because some processes will not have enough frames.

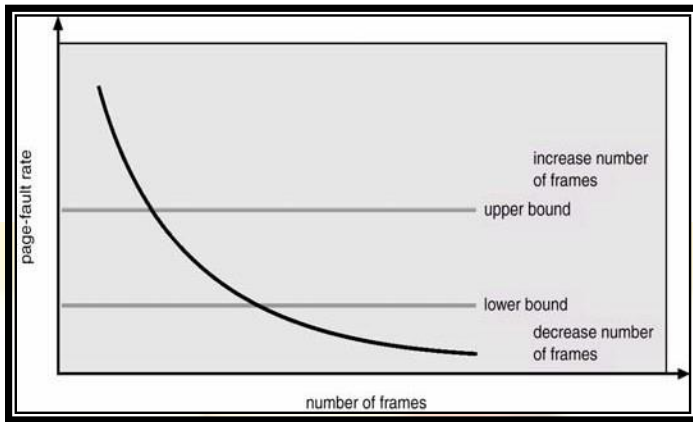
Use of the working set model is then simple, the operating system monitors the working set of each process and allocates to that working set enough frames to provide it with its working set size. If there are enough extra frames another process can be initiated. If the sum of the working set sizes increases, exceeding the total number of available frames, the operating system selects a process to suspend. The process' pages are written out and its frames are reallocated to other processes. The suspended process can be restarted later.

The difficulty with the working set model is to keep track of the working set. The working set window is a moving size window. At each memory reference a new reference appears at one end and the oldest reference drops off the other end. We can approximate the working set model with a fixed interval timer interrupt and a reference bit.

For example, assume $\Delta = 10,000$ references and the timer interrupts every 5000 references. When we get a timer interrupt we copy and clear the reference bit values for each page. Thus if a page fault occurs we can examine the current reference bit and 2 in memory bits to determine whether a page was used within the last 10,000 to 15,000 references. If it was used at least one of these bits will be on, otherwise they will be off. Thus after Δ references, if one of the bits in memory = 1 then the page is in the working set. Note that this arrangement is not completely accurate because we cannot tell where within an interval of 5,000 a reference occurred. We can reduce the uncertainty by increasing the number of our history bits and the frequency of interrupts. However the cost to service these more frequent interrupts will be correspondingly higher.

Page Fault Frequency

Page fault frequency is another method to control thrashing. Since thrashing has a high page fault rate, we want to control the page fault frequency. When it is too high we know that the process needs more frames. Similarly if the page-fault rate is too low, then the process may have too many frames. The operating system keeps track of the upper and lower bounds on the page-fault rates of processes. If the page-fault rate falls below the lower limit, the process loses frames. If page-fault rate goes above the upper limit, process gains frames. Thus we directly measure and control the page fault rate to prevent thrashing. The following diagram shows the working of this scheme.



Controlling thrashing with page fault frequency

Other considerations

Many other things can be done to help control thrashing. We discuss some of the important ones in this section.

Pre-paging

An obvious property of a pure demand paging system is the large number of page faults that occur when a process is started. This situation is the result of trying to get the initial locality into memory. Pre-paging is an attempt to prevent this high level of initial paging. The strategy is to bring into memory at one time all the pages that will be needed.

Pre-paging may be an advantage in some cases. The question is simply whether the cost of using pre-paging is less than the cost of servicing the corresponding page faults.

Page Size

How do we select a page size? One concern is the size of the page table. For a given virtual memory space, decreasing the page size increases the number of pages and hence the size of the page table. Because each active process must have its own copy of the page table, a large page size is desirable.

On the other hand, memory is better utilized with smaller pages. If a process is allocated memory starting at location 00000, and continuing till it has as much as it needs, it probably will not end exactly on a page boundary. Thus, a part of the final page must be allocated. This causes internal fragmentation and to minimize this, we need a small page size.

Another problem is the time required to read or write a page. I/O time is composed of seek, latency and transfer times. Transfer time is proportional to the amount transferred, and this argues for a small page size. However, latency and seek times usually dwarf transfer times, thus a desire to minimize I/O times argues for a larger page size. I/O overhead is also reduced with small page size because locality improves. This is because a smaller page size allows each page to match program locality more accurately.

Some factors (internal fragmentation, locality) argue for a small page size, whereas others (table size, I/O time) argue for a large page size. There is no best answer. However the historical trend is towards larger pages.

Program Structure

Demand paging is designed to be transparent to the user program. However, in some cases system performance can be improved if the programmer has an awareness of the underlying demand paging and execution environment of the language used in the program. We illustrate this with an example, in which we initialize a two dimensional array (i.e., a matrix).

Consider the following program structure in the C programming language. Also note that arrays are stored in row-major order in C (i.e., matrix is stored in the main memory row by row), and page size is such that each row is stored on one page.

Program 1

```
int A[1024][1024];

for (j = 0; j < 1024; j++)
    for (i = 0; i < 1024; i++)
        A[i,j] = 0;
```

Since this code snippet initializes the matrix column by column, it causes 1024 page faults while initializing one column. This means that execution of the code causes **1024 x 1024** page faults.

Now consider the following program structure.

Program 1

```
int A[1024][1024];

for (i = 0; i < 1024; i++)
    for (j = 0; j < 1024; j++)
        A[i,j] = 0;
```

In this case, matrix is accessed row by row, causing 1 page fault per row. This means that execution of the code causes **1024** page faults.

Example Systems

1. A demand paging system with the following utilizations:

| | |
|-------------------|---------|
| CPU | = 20% |
| Paging disk | = 97.7% |
| Other I/O devices | = 5% |

Which of the following will improve CPU utilization?

- Install a faster CPU
- Increase degree of multiprogramming
- Decrease degree of multiprogramming
- Install more main memory

Clearly, the system is thrashing, so the first two are not going to help and the last two will help. Think about the reasons of this answer.

2. Which of the following programming techniques and structures are “good” for a demand paged environment? Which are bad? Explain your answer.

- Stack
- Hash table
- Sequential search
- Binary search
- Indirection
- Vector operations

You should try to answer this question on your own. Focus on how the given data structures and techniques access data. Sequential access means “good” for demand paging (because it causes less page faults) and non-sequential access means “bad” for demand paging environment.

Operating Systems

Lecture No. 42

Reading Material

- Chapter 11 of the textbook
- Lecture 42 on Virtual TV

Summary

- File Concept
- File Types
- File Operations
- Access Methods
- Directories
- Directory Operations
- Directory Structure

The File Concept

Computers can store information on several different storage media, such as magnetic disks, magnetic tapes and optical disks. The operating system abstracts from the physical properties of its storage devices to define a logical storage unit (the file). Files are mapped by the OS onto physical devices. These storage devices are usually non-volatile, so the contents are persistent through power failures, etc. A file is a named collection of related information that is recorded on secondary storage. Data cannot be written to secondary storage unless they are within a file. Commonly, files represent programs (source and object forms) and data. Data files may be numeric, alphabetic, alphanumeric or binary. In essence it is a contiguous logical address space.

File Structure

A file has certain defined structure characteristics according to its type. A few common types of file structures are:

None – file is a sequence of words, bytes

Simple record structure

- Lines
- Fixed length
- Variable length

Complex Structures

- Formatted document
- Relocatable load file

UNIX considers each file to be a sequence of bytes; no interpretation of these bytes is made by the OS. This scheme provides maximum flexibility but little support. Each application program must include its own code to interpret an input file into the

appropriate structure. However all operating systems must support at least one structure- that of an executable file- so that the system is able to load and run programs.

File Attributes

Every file has certain attributes, which vary from one OS to another, but typically consist of these:

Name: The symbolic file name is the only information kept in human-readable form

Type: This information is needed for those systems that support different types.

Location: This location is a pointer to a device and to the location of the file on that device.

Size: The current size of the file (in bytes, words or blocks) and possibly the maximum allowed size are included in this attribute.

Protection: Access control information determines who can do reading , writing, etc.

Owner

Time and date created: useful for security, protection and usage monitoring.

Time and date last updated: useful for security, protection and usage monitoring.

Read/write pointer value

Where are Attributes Stored?

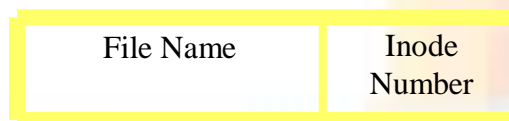
File attributes are stored in the directory structure, as part of the **directory entry** for a file, e.g., in DOS, Windows, or in a separate data structure; in UNIX/Linux this structure is known as the **inode** for the file.

Directory Entry

A file is represented in a directory by its directory entry. Contents of a directory entry vary from system to system. For example, in DOS/Windows a directory entry consists of file name and its attributes. In UNIX/Linux, a directory entry consists of file name and inode number. Name can be up to 255 characters in BSD UNIX compliant systems. Inode number is used to access file's inode. The following diagrams show directory entries for DOS/Windows and UNIX/Linux systems.



DOS/Windows



UNIX/Linux

File Operations

Various operations can be performed on files. Here are some of the commonly supported operations. In parentheses are written UNIX/Linux system calls for the corresponding operations.

- Create (`creat`) —two steps are necessary to create a file. First, space must be found for the file in the file system. Second, an entry for the new file must be made in the directory.
- Open (`open`) — The open operation takes a file name and searches the directory, copying the directory entry into the open-file table. The open system call can also accept access-mode information-read-only, read-write, etc. It typically returns a pointer to the entry in open-file table.
- Write (`write`) —To write to a file, we make a system call, specifying both the name of the file and the information to be written to the file. Given the name of the file, the system searches the directory to find the location of the file. The system must keep a *write* pointer to the location in the file where the next write is to take place. The write pointer must be updated whenever a write occurs.
- Read (`read`) — To read from a file we use a system call that specifies the name of the file, and where (in memory) the next block of the file should be put. The system needs to keep a read pointer to the location in the file where the next read is to take place. Once the read has taken place, the read pointer needs to be updated. A given process is usually only reading or writing to a file. The current pointer location is kept as a process **current-file-position pointer**. Both read and write use the same pointer
- Reposition within file (`lseek`) — A directory is searched for the appropriate entry and the current-file-position is set to a given value. This is often known as a file seek.
- Delete (`unlink`) — Search the directory for the named file, and then release the file space and erase the directory entry. File can be deleted using the `unlink` system call.
- Truncate (`creat`) — A user may want to erase the contents of the file but keep its attributes. This function allows all attributes to be unchanged except for file length., which is set to zero and file space is released. This can be achieved using `creat` with a special flag
- Close (`close`) — When a file is closed, the OS removes its entry in the open-file table.

File Types: Extensions

A common technique for implementing files is to include the type of the file as part of the file name. The name is split into two parts, a name and an extension, usually separated by a period character. In this way, the user and the OS can tell from the name alone, what the type of a file is.

The operating system uses the extension to indicate the type of the file and the type of operations that can be done on that file. In DOS/Windows only a file with `.exe`, `.com`, `.bat` extension can be executed.

The UNIX system uses a crude magic number stored at the beginning of some files to indicate roughly the type of the file-executable program, batch file/shell script, etc. Not all files have magic numbers, so system features cannot be based solely on this type of information. UNIX does allow file name extension hints, but these extensions are not enforced or depended on by the OS; they are mostly to aid users in determining the type of contents of the file. Extension can be used or ignored by a given application.

The following tables shows some of the commonly supported file extensions on different operating systems.

| file type | usual extension | function |
|----------------|--------------------------------|---|
| executable | exe, com, bin or none | read to run machine-language program |
| object | obj, o | compiled, machine language, not linked |
| source code | c, cc, java, pas, asm, a | source code in various languages |
| batch | bat, sh | commands to the command interpreter |
| text | txt, doc | textual data, documents |
| word processor | wp, tex, rrf, doc | various word-processor formats |
| library | lib, a, so, dll, mpeg, mov, rm | libraries of routines for programmers |
| print or view | arc, zip, tar | ASCII or binary file in a format for printing or viewing |
| archive | arc, zip, tar | related files grouped into one file, sometimes compressed, for archiving or storage |
| multimedia | mpeg, mov, rm | binary file containing audio or A/V information |

Common file types

File Types in UNIX

UNIX does not support supports seven types of file:

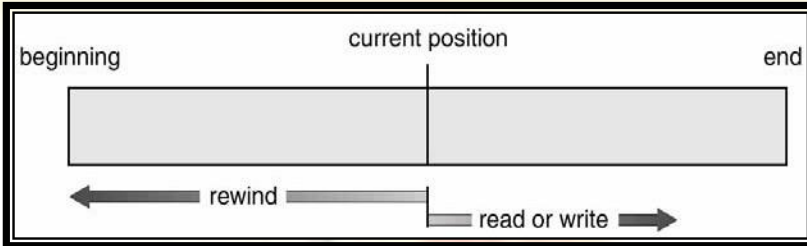
- **Ordinary file:** used to store data on secondary storage device, e.g., a source program(in C), an executable program. Every file is a sequence of bytes.
- **Directory:** contains the names of other files and/or directories.
- **Block-special file:** correspond to block oriented devices such as a disk. They are used to access such hardware devices.
- **Character-special file:** correspond to character oriented devices, such as keyboard
- **Link file** (created with the `ln -s` command): is created by the system when a symbolic link is created to an existing file, allowing you to rename the existing file and share it without duplicating its contents without
- **FIFO** (created with the `mkfifo` or `mknod` commands or system calls): enable processes to communicate with each other. A FIFO(name pipe) is an area in the kernel that allows two processes to communicate with each other provided they are running on the same system , but the processes do not have to be related to each other.
- **Socket** (in BSD-compliant systems—socket): can be used by the process on the same computer or on different computers to communicate with each other.

File Access

Files store information that can be accessed in several ways:

Sequential Access

Information in the file is processed in order, one record after the other. A read operation reads the next portion of the file and automatically advances a file pointer which tracks the I/O location. Similarly, a write operation appends to the end of the file and advances to the end of the newly written material. Such a file can be reset to the beginning and on some systems; a program may be able to skip forward or backward, n records.



Sequential Access File

Direct Access

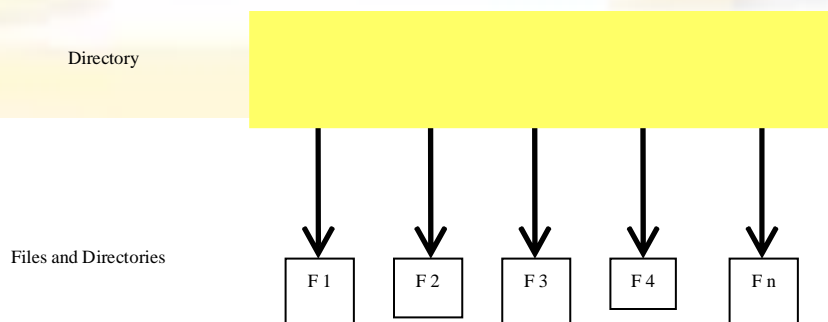
A file is made up of fixed length logical record that allow program to read and write records in no particular order. For the direct-access method, the file operations must be modified to include the block number as a parameter (read n (n = relative block number), write n for instance). An alternate approach is to retain read next and write next and to add an operation, *position file to n* , where n is the block number. The block number provided by the user to the OS is normally a *relative block number*, an index relative to the beginning of the file.

| sequential access | implementation for direct access |
|-------------------|---------------------------------------|
| <i>reset</i> | <i>cp = 0;</i> |
| <i>read next</i> | <i>read cp;</i> <i>cp = cp+1;</i> |
| <i>write next</i> | <i>write cp;</i> <i>cp = cp+1;</i> |

Sequential Access on a Direct Access File

Directory Structure

It is a collection of directory entries. To manage all the data, first disks are split into one or more partitions. Each partition contains information about files within it. This information is kept within device directory or volume table of contents.



Directory Operations

The following directory operations are commonly supported in contemporary operating systems. Next to each operation are UNIX system calls or commands for the corresponding operation.

- Create — `mkdir`
- Open — `opendir`
- Read — `readdir`
- Rewind — `rewinddir`
- Close — `closedir`
- Delete — `rmdir`
- Change Directory — `cd`
- List — `ls`
- Search

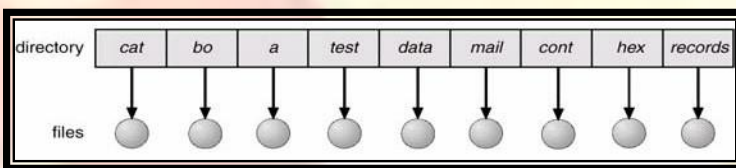
Directory Structure

When considering a particular directory structure we need to consider the following issues:

1. **Efficient Searching**
2. **Naming** – should be convenient to users
 - Two users can have same name for different files
 - The same file can have several different names
3. **Grouping** – logical grouping of files by properties, (e.g., all Java programs, all games, ..)

Single-Level Directory

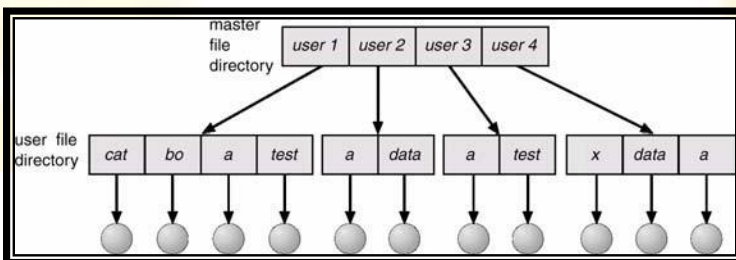
All files are contained in the same directory, which is easy to support and understand. However when the number of files increases or the system has more than one user, it has limitations. Since all the files are in the same directory, they must have unique names.



Single-level directory structure

Two-Level Directory

There is a separate directory for each user.

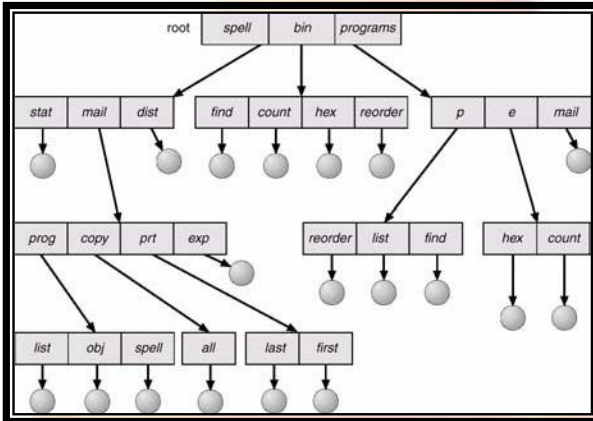


Two-level directory structure

When a user refers to a particular file, only his own user file directory (UFD) is searched. Thus different users can have the same file name as long as the file names within each UFD are unique. This directory structure allows efficient searching. However, this structure effectively isolates one user from another, hence provides no grouping capability.

Tree Directory

Here is the tree directory structure. Each user has his/her own directory (known as user's home directory) under which he/she can create a complete directory tree of his/her own.



Tree directory structure

The tree has a root directory. Every file in the system has a unique pathname. A path name is the path from the root, through all the subdirectories to a specified file. A directory/subdirectory contains a set of files or subdirectories. In normal use, each user has a current directory. The current directory should contain most of the files that are of current interest to the user. When a reference to a file is made, the current directory is searched. If a file is needed that is not in the current directory, then the user must either specify a path name or change the directory to the directory holding the file(using the cd system call).This structure hence supports efficient searching. Allowing the user to define his own subdirectories permits him to impose a structure on his files. Also users can access files of other users.

UNIX / Linux Notations and Concepts

- Root directory (/)
- Home directory
 - ~, \$HOME, \$home
 - cd ~
 - cd
- Current/working directory (.)
 - pwd
- Parent of Current Directory (..)
- Absolute Pathname
 - Starts with the root directory
 - For example, /etc, /bin, /usr/bin, /etc/passwd, /home/students/ibraheem
- Relative Pathname

- Starts with the current directory or a user's home directory
- For example, ~/courses/cs604, ./a.out



Operating Systems

Lecture No. 42

Reading Material

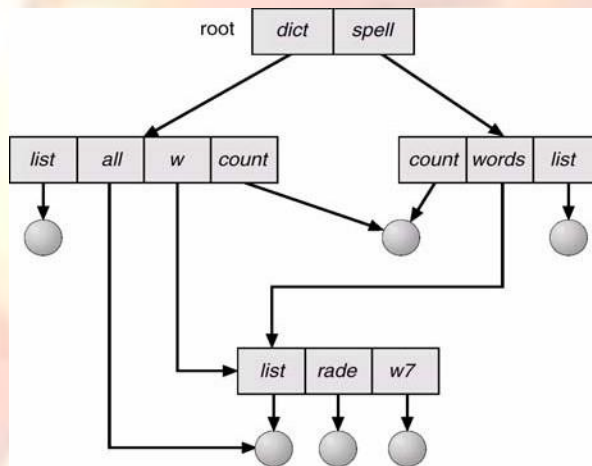
- Chapter 11 of the textbook
- Lecture 43 on Virtual TV

Summary

- Directory Structures
- Links in UNIX/Linux
- File System Mounting
- File Sharing
- File Protection

Acyclic-Graph Directories

A tree structure prohibits sharing of files. An acyclic graph allows directories to have shared subdirectories and files. The same file may be in two different directories.



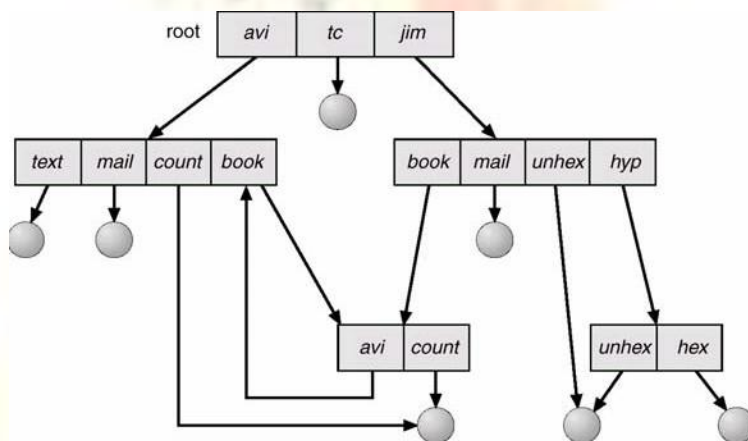
Acyclic-graph directory structure

A shared file is not the same as two copies of the file. Only one actual copy exists, so any changes made by one user are immediately visible to the other. A common way of implementing shared files and directories is to create a new directory entry called a link, which is effectively a pointer to another file or subdirectory. A link can be implemented as an absolute or relative path name. A file may now have multiple absolute path names. This problem is similar to the aliasing problem in programming languages. Consequently distinct file name may refer to the same files. If we are traversing the entire file system-to find a file, to accumulate statistics, etc, this problem becomes significant since we do not want to traverse the shared structures more than once. Another problem involves deletion.

If the file is removed when anyone deletes it, we may end up with dangling pointers to the now-nonexistent file.

Solutions: Another approach is to preserve the file until all references to it are deleted. When a link or a copy of the directory entry is established, a new entry is added to the file-reference list. When a link is deleted, we remove its entry on the list. The file is deleted when its file-reference list is empty. Since the reference list can be very large we can keep a count of the number of references. A new link or directory increments the **reference count**, deleting a link or entry decrements the count. When the count is 0, the file can be deleted. UNIX uses this solution for hard links. **Backpointers** can also be maintained so we can delete all pointers.

General Graph Directory



General graph directory

One serious problem with using an acyclic-graph structure is ensuring that there are no cycles. A solution is to allow only links to files not subdirectories. Also every time a new link is added use a **cycle detection algorithm** to determine whether it is OK. If cycles are allowed, we want to avoid searching any component twice. A similar problem exists when we are trying to determine when a file can be deleted. A value of 0 in the reference count means no more references to the file/directory can be deleted. However, cycles can exist, e.g, due to self-referencing. In this case we need to use a garbage collection scheme, which involves traversing the entire file system, marking everything that can be accessed. Then a second pass collects everything that is not marked onto a list of free space. However this is extremely time consuming and is seldom used. However it is necessary because of possible cycles in a graph.

Links in UNIX

UNIX supports two types of links:

- **Hard links**
- **Soft (symbolic) links**

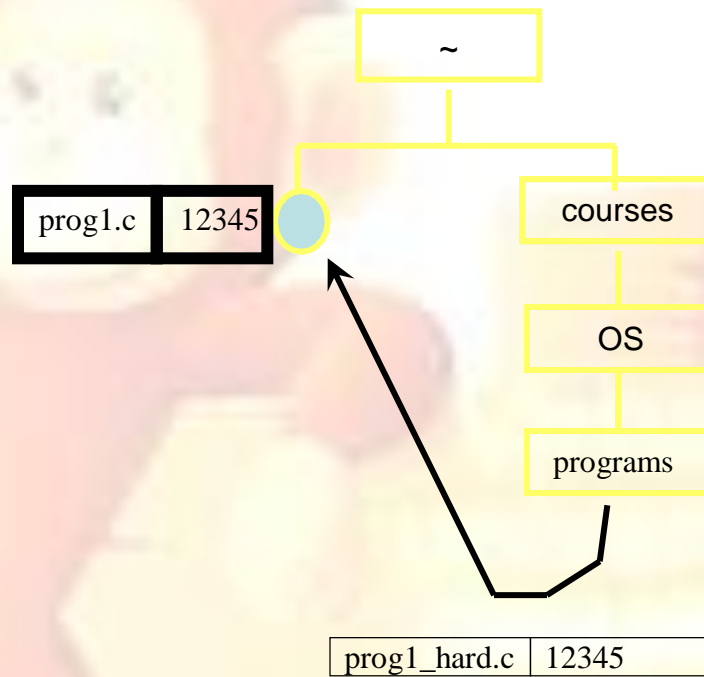
The `ln` command is used to create both links, `ln -s` is used to create a soft link

- `ln [options] existing-file new-file`
- `ln [options] existing-file-list directory`

Examples: The first command creates a hard link `~/courses/OS/programs/prog1_hard.c` to an existing file `~/prog1.c`. The second command creates a soft link `~/prog2_soft.c` to an existing file `~/courses/OS/programs/prog2.c`. The diagrams below show the directory structures after these links have been created. Note that directory entries for hard links to the same file have the same inode number.

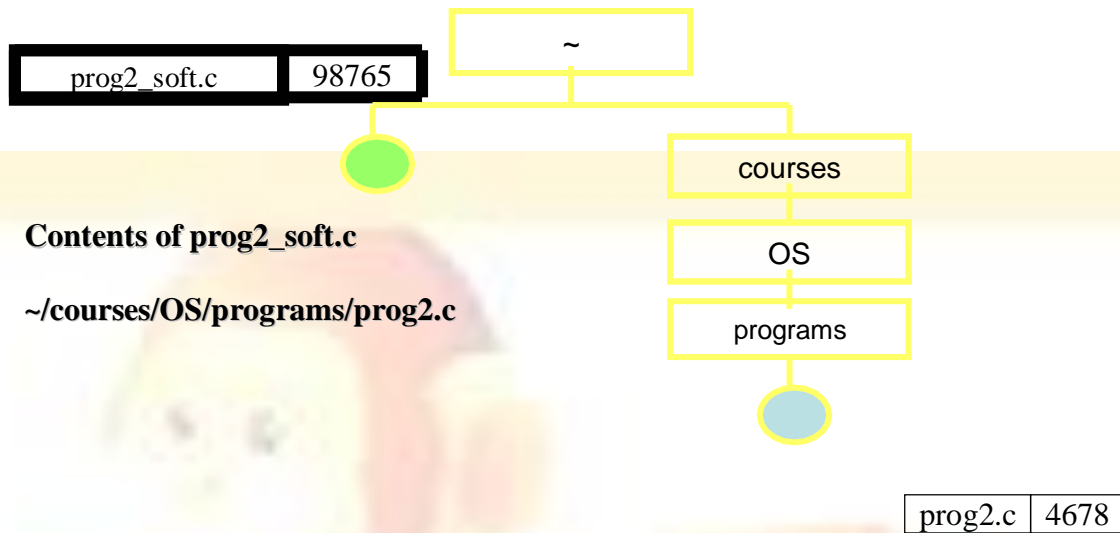
```
ln ~/prog1.c ~/courses/OS/programs/prog1_hard.c
ln -s ~/courses/OS/programs/prog2.c ~/prog2_soft.c
```

Hard Links



When a hard link is created, a directory entry for the existing file is created—there is still only one file. Both entries have the same inode number. The link count is incremented by one in the inode for the file. No hard links can be created for directories. Also hard links cannot be established between files that are on different file systems. In UNIX, a file is removed from the file system only if its hard link count is 0.

Soft Links



Contents of prog2_soft.c

~/courses/OS/programs/prog2.c

A file of type 'link' is created, which contains the pathname for the existing file as specified in the ln command. The existing file and the new (link) files have different inode numbers. When you make a reference to the link file, the UNIX system sees that the type of file is link and reads the link file to find the pathname for the actual file to which you are referring. When the existing file is removed, you have a 'dangling pointer' to it in the link file. Soft links take care of all the problems inherent in hard links. They are flexible. You may have soft links to directories and across file systems. However, UNIX has to support an additional file type, the link type, and a new file is created for every link, slowing down file operations.

File System Mounting

A file system is best visualized as a tree, rooted at /. /dev, /etc, /usr, and other directories in the root directory are branches, which may have their own branches, such as /etc/passwd, /usr/local, and /usr/bin. Filling up the root file system is not a good idea, so splitting /var from / is a good idea. Another common reason to contain certain directory trees on other file systems is if they are to be housed on separate physical disks, or are separate virtual disks, or CDROM drives.

Mounting makes file systems, files, directories, devices, and special files available for use at a particular location. **Mount point** is the actual location from which the file system is mounted and accessed. You can mount a file or directory if you have access to the file or directory being mounted and write permission for the mount point

There are types of mounts:

- Remote mount
- Local mount

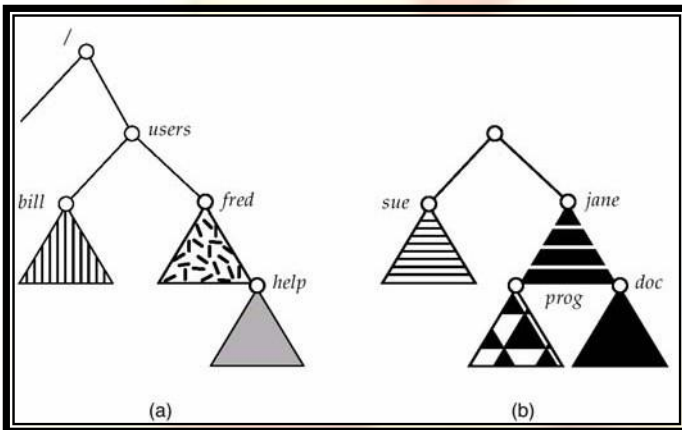
Remote mounts are done on a remote system on which data is transmitted over a telecommunication line. **Local mounts** are mounts done on your local system.

Mounting in UNIX

All files accessible in a Unix system are arranged in one big tree, the file hierarchy, rooted at `/`. These files can be spread out over several devices. The `mount` command serves to attach the file system found on some device to the big file tree. Conversely, the `umount` command will detach it again. Here is the syntax of the `mount` command

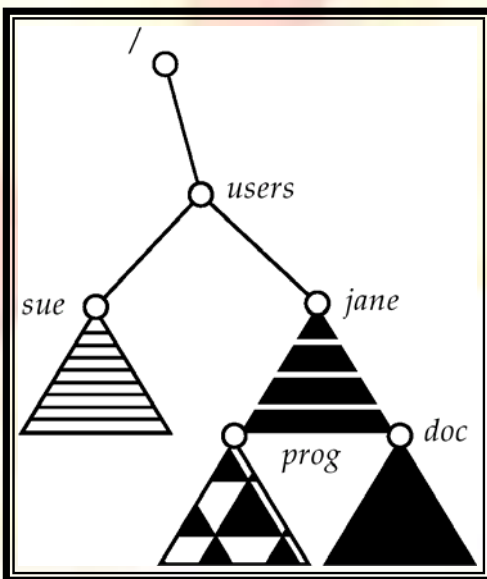
```
mount -t type device dir
```

This command tells the kernel to attach the file system found on *device* (which is of type *type*) at the directory *dir*. The previous contents (if any) and owner and mode of *dir* become invisible. As long as this file system remains mounted, the pathname *dir* refers to the root of the file system on *device*.



Existing Tree

Unmounted filesystem



New Tree after mounting Filesystem

File System Space Usage

On SuSE Linux

```
$ df
```

```
Filesystem      1K-blocks      Used Available  Use% Mounted on
/dev/hda3        74837584  11127072  59908892   16% /
tmpfs            257948         12     257936    1% /dev/shm
/dev/hda1        19976          6960     11985    37% /boot
inbox:/var/spool/mail
                  66602516    3319996  59899232    6% /var/spool/mail
upfile1a:/usr1.a 230044816    70533680 147825456   33% /usr1.a
upfile2a:/usr2.a 230044816   118228296 100130840   55% /usr2.a
upibma:/usr3.a   16713676     804252   15039103    6% /usr3.a
upfile4a:/usr4.a 230044816   14594384 203764752    7% /usr4.a
$
```

On Solaris 2

```
$ df -v
```

```
Mount Dir      Filesystem      blocks      used      free      %used
/         /dev/dsk/c0t12d 7557677     2484225   4997876   34%
/proc    /proc           0           0         0         0%
/etc/mntta mnttab         0           0         0         0%
/dev/fd   fd             0           0         0         0%
/var/run  swap           510103      22        510081    1%
/tmp     swap           683241     173160    510081    26%
/oldexport /dev/dsk/c0t8d0 4668856    4229110   393058    92%
/export/ho /dev/dsk/c0t12d 23684712   21714309  1733556   93%
$
```

File Sharing

Sharing of files on multi-user systems is desirable. People working on the same project need to share information. For instance: software engineers working on the same project need to share files or directories related to the project

Sharing may be done through

- **Duplicating files:** Make copies of the file and give them to all team members. This scheme works well if members of the team are to work on these shared files sequentially. If they work on the files simultaneously, the copies become inconsistent and no single copy reflects the works done by all members. However it is simple to implement.
- **Common login** for members of a team: The system admin creates a new user group and gives the member access to the new account. All files and directories created by any team member under this account and are owned by the team. This works well if number of teams is small and teams are stable. However a separate account is needed for the current project and the system administrator has to create a new account for every team
- Setting appropriate **access permissions**. Team members put all shared files under one member's account and the access permissions are set so all the members can access it. This scheme works well if *only* this team's members form the user group. File access permissions can be changed using the `chmod` system call:

```
chmod [options] octal-mode file list
chmod [options] symbolic -mode file-list
```

A few examples:

–To let people in your UNIX group add, delete, and rename files in a directory of yours - and read or edit other people's files if the file permissions let them - use

chmod 775 *dirname*.

–To make a private file that only you can edit, use `chmod 600 filename`. To protect it from accidental editing, use `chmod 400 filename`.

- **Common groups** for members of a team. : System admin creates a new user group consisting of the members of team only. All team members get individual logins and set access permissions for their files so that they are accessible to other group members
- **Links.** A link is a way to establish a connection between the file to be shared and the directory entries of the users who want to have access to this file. The two types of links supported by UNIX:
 - Hard link
 - Soft/symbolic link

Operating Systems

Lecture No. 44

Reading Material

- Chapters 11 and 12 of the textbook
- Lecture 44 on Virtual TV

Summary

- File Protection
- In-Memory Data Structures
- Space Allocation Techniques
- Contiguous, Linked, Index

Protection

The need to protect files is a direct result of the ability to access files. Systems that do not permit access to the files of other users do not need protection. Thus we could provide complete protection by prohibiting access. Alternatively we could provide free access with no protection. Both approaches are too extreme for general use. What is needed is controlled access. File owner/creator should be able to control

- What can be done
- By whom

Several types of operations may be controlled:

- Read: read from the file
- Write: write or rewrite to the file
- Execute: Load the file into memory and execute it
- Append: Write new information at the end of the file
- Delete: Delete the file and free its space for possible reuse
- List: List the name and attributes of the file

UNIX Protection

UNIX recognizes three modes of access: **read**, **write**, and **execute** (r, w, x). The execute permission on a directory specifies permission to **search** the directory.

The three classes of users are:

- **Owner:** user is the owner of the file
- **Group:** someone who belongs to the same group as the owner
- **Others:** everyone else who has an account on the system

A user's access to a file can be specified by an octal digit. The first bit of the octal digit specifies the read permission, the second bit specifies the write permission, and the third bit specifies the execute permission. A bit value 1 indicates permission for access and 0 indicates no permission. Here is an example:

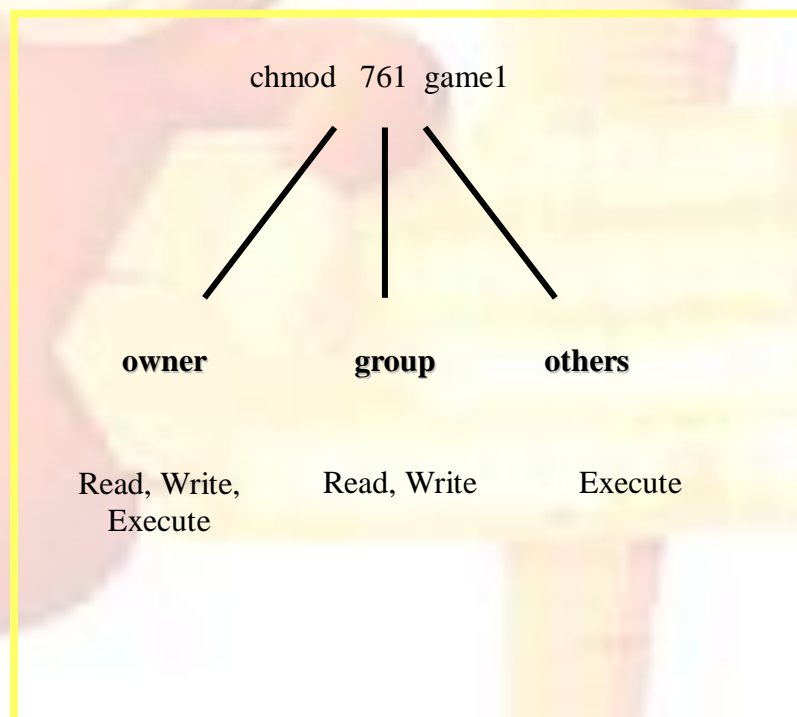
| | <u>r</u> | <u>w</u> | <u>x</u> |
|----------------------------|----------|----------|----------|
| a) Owner access: 7 | 1 | 1 | 1 |
| b) Group access: 6 | 1 | 1 | 0 |
| c) Public access: 1 | 0 | 0 | 1 |

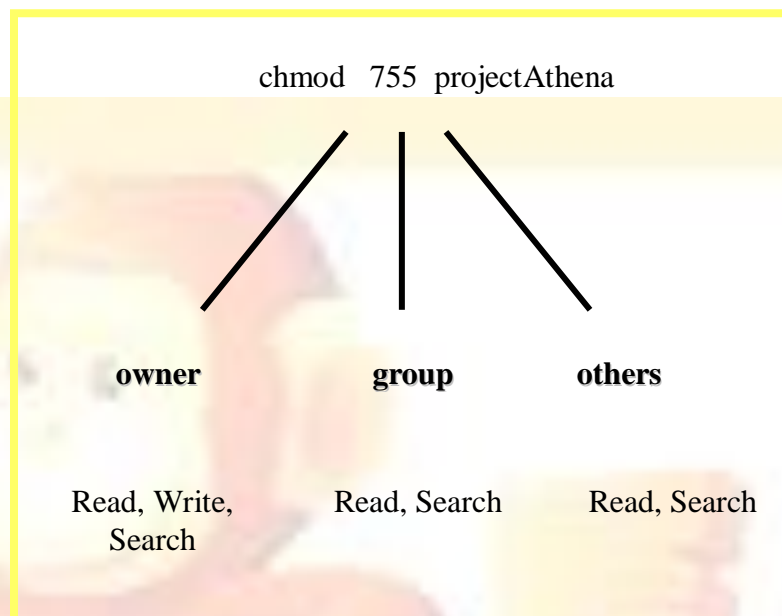
Each user in a UNIX system belongs to a group of users as assigned by the system administrator when a user is allocated an account on the system. A user can belong to multiple groups, but a typical UNIX user belongs to a single group.

For a particular file or subdirectory, we need to set appropriate **access permissions** for proper protection.

Default Permissions

The default permissions on a UNIX/Linux system are 777 for executable files and directories and 666 for text files. You can use the `umask` command to set permission bits on newly created files and directories to 1, except for those bits that are set to 1 in the 'mask'. You can use the `chmod` command to set permissions on existing files and directories. We give some examples of the `chmod` and `umask` commands below.





Sample commands

- chmod 700 ~.....Set permissions on home directory to 700
- chmod 744 ~/file.....Set permissions on ~/file to 744
- chmod 755 ~/directory... Set permissions on ~/directory 755
- ls -l ~ Display permissions and some other attributes for all files and directories in your home directory
- ls -ld ~..... Display permissions and some other attributes for your home directory
- ls -l prog1.c..... Display permissions and some other attributes for prog1.c in your current directory
- ls -ld ~/courses.....Display permissions and some other attributes for your home directory

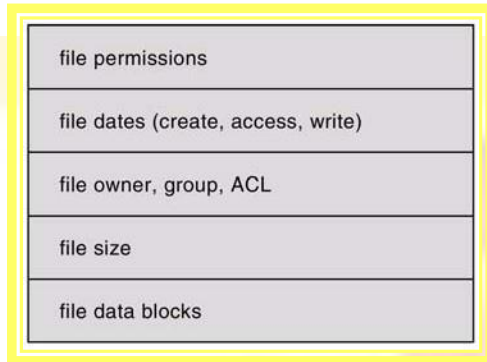
The **umask** command sets default permissions on newly created files and directories as
(default permissions – mask value)

Here are some sample commands

- umask Display current mask value (in octal)
- umask 022 Set mask value to octal 022 (turn off write permission for 'group' and 'others')
- touch temp1 .. Create an empty file called temp1
- ls -l temp1 Display default permissions and some other attributes for the temp1 file

File Control Block

A file control block is a memory data structure that contains most of the attributes of a file. In UNIX, this data structure is called inode (for index node). Here are possible values in this data structure.



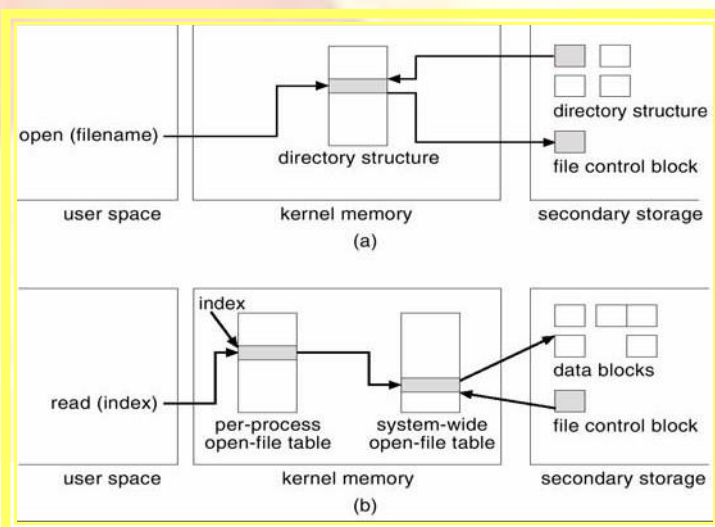
UNIX inode

In-Memory Data Structures

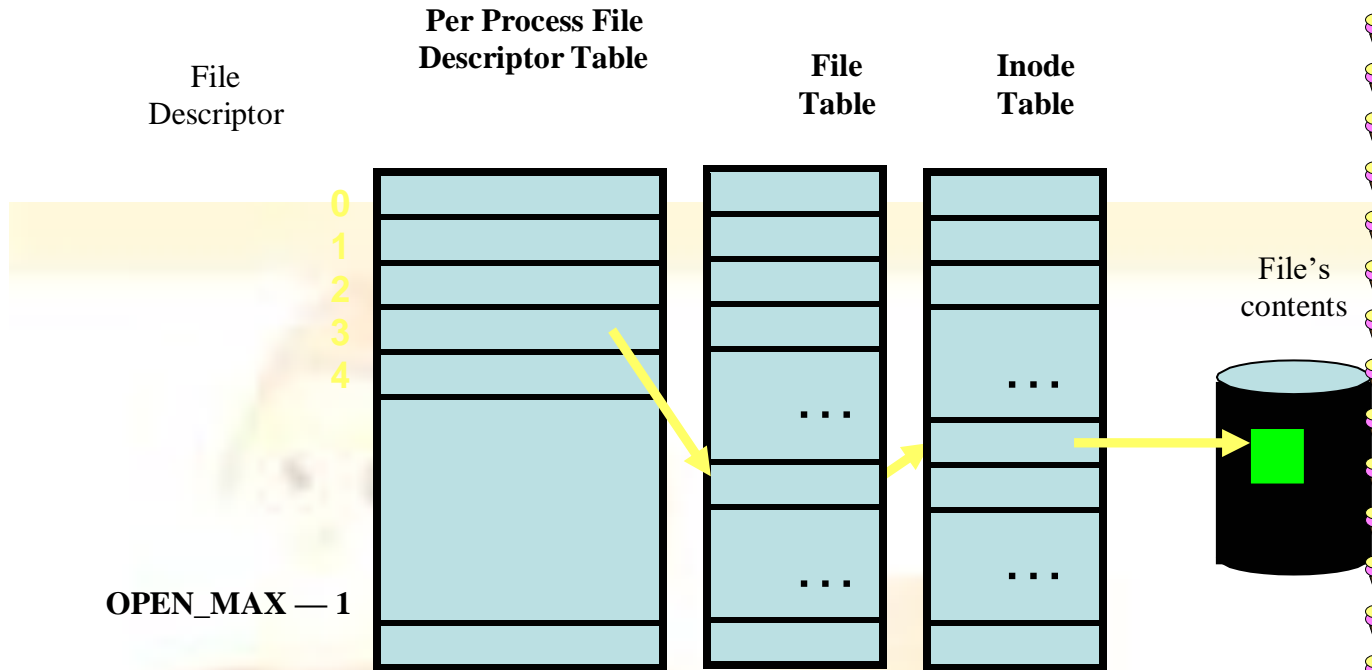
The following upper-level data structures needed for file system support.

- An in-memory partition table containing information about each mounted partition
- An in-memory directory structure that holds the directory information of recently accessed directories
- The system-wide open file table contains pointer to the FCB (UNIX inode) of each open file as well as read/write pointer
- The FCB for each open file
- The per process file descriptor table contains a pointer to the appropriate entry in the system wide open file table as well as other information

Here are the connections between various in-memory data structures. UNIX specific mappings follow this diagram.



Connections between various in-memory data structures



From File Descriptor to File Contents—The UNIX/Linux In-Memory Data Structures

The `open` call passes a file name to the file system. When a file is opened, the directory structure is searched for the given file name and file's inode. An entry is made in the per process open-file table (aka the file descriptor table), with a pointer to the entry in the system wide open file table. The system wide open file table contains the pointer to the current location in the file and a pointer to file's inode. The open call returns an index for the appropriate entry in the per-process file system table. All file operations are performed via this index, which is called the file descriptor in UNIX/Linux jargon.

Space Allocation Methods

We now turn to some file system implementation issues, in particular space allocation techniques and free space management methods. Here are the three commonly used methods for file space allocation.

- Contiguous allocation
- Linked allocation
- Indexed allocation

Contiguous Allocation

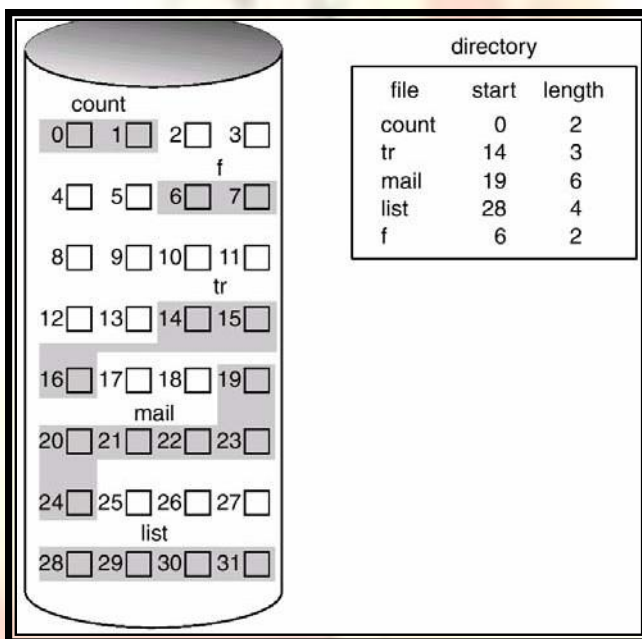
The contiguous allocation method requires each file to occupy a set of contiguous blocks on the disk. The directory entry for each file contains starting block number and file size (in blocks). Disk addresses define a linear ordering on the disk. With this ordering, assuming only one job is accessing the disk, accessing $b+1$ block after block b normally requires no head movement. When head movement is needed it is only one track. Both sequential and direct access can be supported by contiguous allocation. For direct access to block I of a file that starts at block b we can immediately access block $b+i$.

Best-fit, first-fit, or worst-fit algorithms are the strategies used to select a hole from the set of available holes. Neither first fit, nor best fit is clearly best in terms of both time and storage utilization, but first fit is generally faster.

These algorithms suffer from the problem of external fragmentation. As files are allocated or deleted, the free disk is broken into little pieces. This situation results in **external fragmentation** of disk (similar to external fragmentation of main memory due to segmentation). Disk defragmenter utility needs to be used for removing external fragmentation.

Determining how much space is needed for a file is another problem. User needs to declare file size, and estimating file size may be difficult. Also file growth is expensive in contiguous allocation. Worst-fit space allocation algorithm can be used to allow growth in a file's size.

The following diagram shows an example of the contiguous allocation scheme.

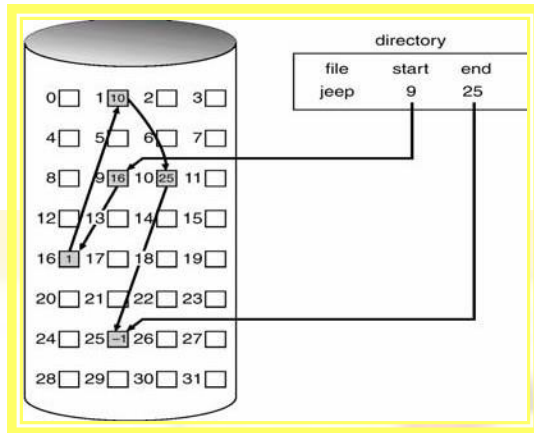


Contiguous allocation

Linked Allocation

Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk. The directory contains a pointer to the first and last blocks of the file. There is no external fragmentation with linked allocation, and any free block on the free-space list can be used to satisfy a request. There is no wastage of space. However, a major disadvantage with linked allocation is that it can be used only for sequential access files. To find the i th block of a file, we must start at the beginning of that file and follow the pointers until we get back to the i th block. Consequently it is inefficient to support a direct access capability for linked allocation files.

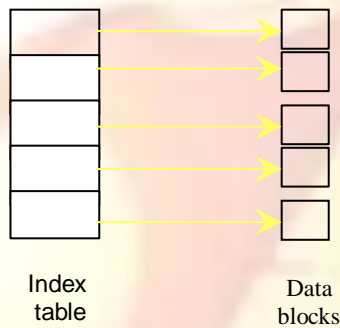
Here is an example of linked allocation.



Linked allocation

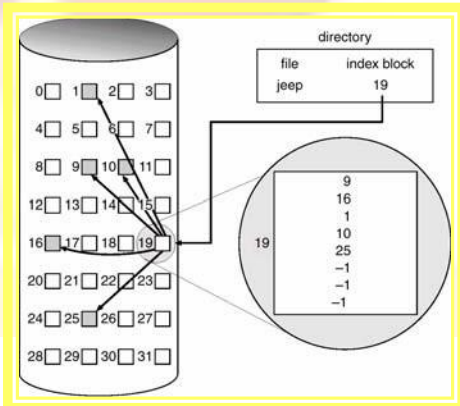
Index Allocation

Indexed allocation brings all the pointers to the block together into a disk block, known as the **index block**. Here is the logical view of the relationship between an index block and a file's data blocks.



Logical view of index allocation

Each file has its own index block, which is an array of disk block addresses. The i th entry in the index block points to the i th block of the file. The directory contains the address of the index block. To read the i th block, we use the pointer in the i th index-block entry to find and read the desired block. Here is an example of index allocation.



Index allocation

Operating Systems

Lecture No. 45

Reading Material

- Chapters 12 and 14 of the textbook
- Lecture 45 on Virtual TV

Summary

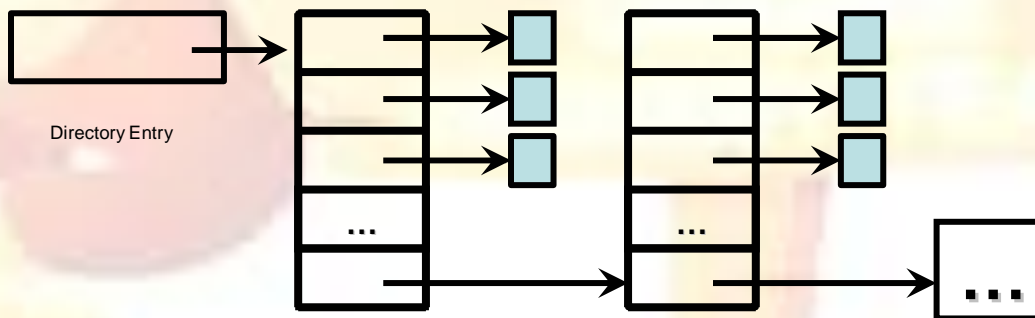
- Space Allocation Techniques (continued)
- Free Space Management
- Disk Structure and Scheduling

Index Allocation (continued from previous lecture)

Indexed allocation supports direct access without suffering from external fragmentation because any free block on the disk may satisfy a request for more space. Depending on the disk block size and file system size, a file may need more than one index block. In this case there are two ways of organizing index blocks:

Linked scheme (linked list of index blocks)

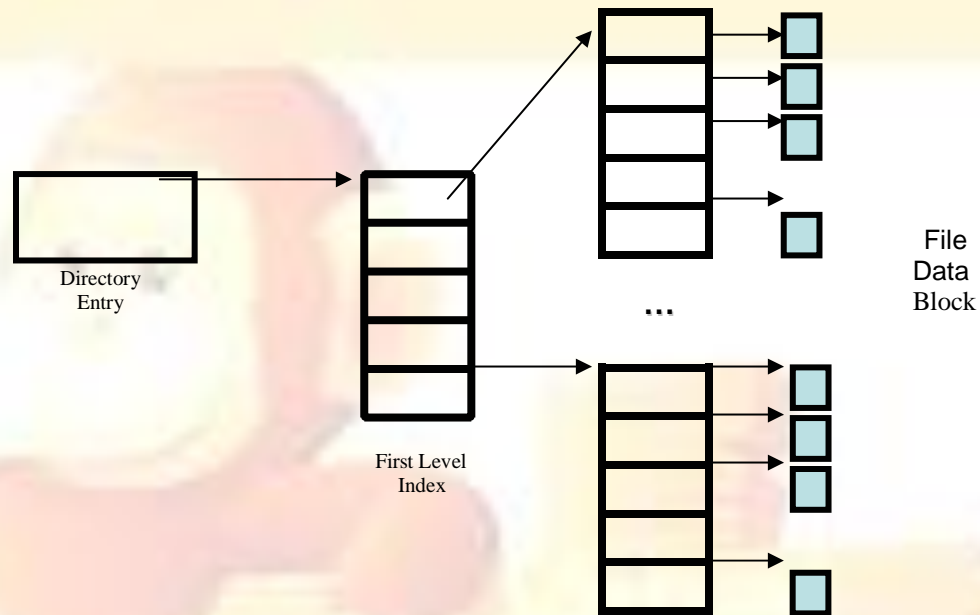
An index block is normally one disk block. Thus, it can be read and written directly by itself. To allow for large files, we may link together several index blocks. For example, an index block might contain a small header giving the name of the file and a set of first 100 disk-blocks addresses. The next address (the last word in the index block) is nil (for a small file) or a pointer to another index block (for a large file), as shown below.



Linked scheme for interconnecting index blocks

Multi-level index scheme

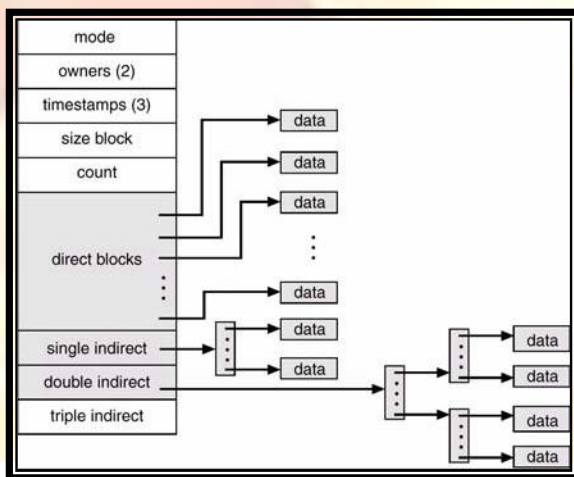
The second method of handling multiple index blocks is to maintain multi-level indexing. In the following diagram, we show two-level index table.



Two level Index Table

UNIX Space Allocation

The UNIX file manager uses a combination of indexed allocation and linked lists for the index table. It maintains 10-15 direct pointers to file blocks, and three indirect pointers (one-level indirect, two-level indirect, and three-level indirect), all maintained in file's inode, as shown below.



UNIX inode

Let's consider a UNIX system with following attributes:

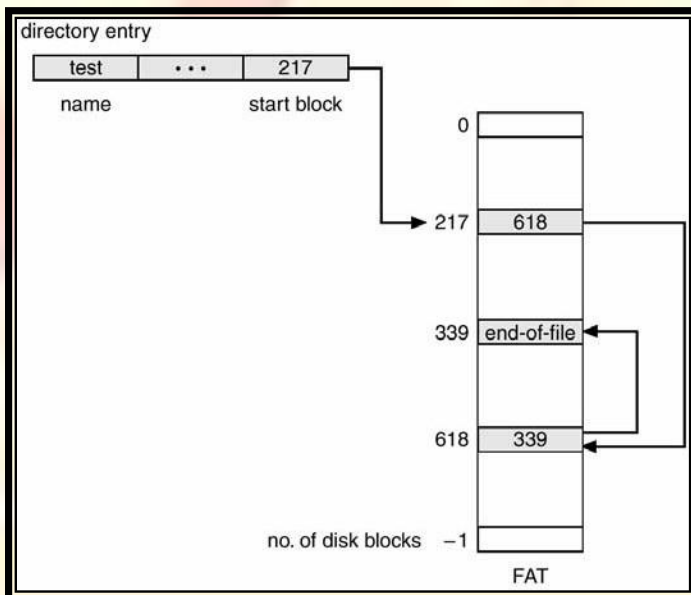
- The block size is 4 KB block
- 4-byte disk pointers (which means, 1024 points per disk block)
- 10 direct pointers to file blocks

Maximum file size and pointer overhead?

10*4=40 KB of data may be accessed directly (in the above case). The maximum file size depends on the size of the blocks and the size of the disk addresses used in the system. The next pointers point to indirect blocks. The single indirect block is an index block containing not the data but rather the addresses of blocks that do contain data. Then there is a double indirect block pointer, which contains the address of a block that contains the addresses of blocks that contain pointers to the actual data blocks. Finally, the triple indirect block pointer points to first-level index block, which points to second-level index blocks, which point to third-level index blocks, which point to data blocks. With the given parameters, the maximum file size will be $[10 + 1024 + 1024^2 + 1024^3]$ blocks—multiply this by the block size to get size in bytes. Similarly, you can calculate the pointer overhead for the largest file.

File Allocation Table (FAT)

The file system on an MS-DOS floppy disk is based on **file allocation table** (FAT) file system in which the disk is divided into a reserved area (containing the boot program) and the actual file allocation tables, a root directory and file space. Space allocated for files is represented by values in the allocation table, which effectively provide a linked list of all the blocks in the file. Each entry is indexed by a block number and value in a table location contains block number for the next file block. First block number for a file is contained in file's directory entry. Special values designate end of file, unallocated and bad blocks. The following diagram summarizes the overall picture of FAT.



File Allocation Table (FAT)

Free-Space Management

Since disk space is limited, we need to reuse the space from deleted files for new files if possible. To keep track of free disk space, the system maintains a **free-space list**. The free space list records all *free* disk blocks—those not allocated to some file or directory. To create a file we search the free-space list for the required amount of space and allocate the space to the new file. This space is then removed from the free-space list. When a file is deleted, its disk space is added to the free space list.

Bit vector

Frequently, the free space list is implemented as a bit map or bit vector. Each block is represented by 1 bit. If the block is free, the bit is 1; if it is allocated, the bit is 0. This approach is relatively simple and efficient in finding the first free block or n consecutive free blocks on the disk.



$$\text{bit}[i] \begin{cases} 0 \Rightarrow \text{block}[i] \text{ is free} \\ 1 \Rightarrow \text{block}[i] \text{ is occupied} \end{cases}$$

The calculation of block number is:

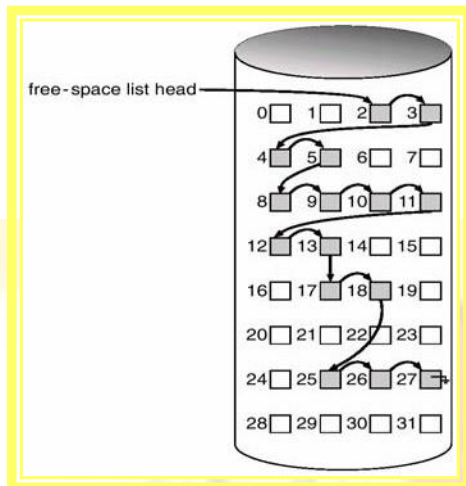
(number of bits per word) * (number of 0-value words) + offset of first 1 bit

Example for overhead of bit map

$$\begin{aligned} \text{Block size} &= 4 \text{ KB} = 2^{12} \text{ bytes} \\ \text{Disk size} &= 40 \text{ GB} = 40 * 2^{30} \text{ bytes} \\ \text{Overhead} &= 40 * 2^{30} / 2^{12} = 40 * 2^{18} \text{ bits} \\ &= 40 * 32 \text{ KB} = 1280 \text{ KB} \end{aligned}$$

Linked list (free list)

Another approach to free space management is to link together all the free disk blocks, keeping a pointer to the first free block in a special location on the disk and caching it in memory. The first block contains a pointer to the next free disk block and so on. However this scheme is not efficient. To traverse the list, we must read each block, which requires substantial I/O time. It cannot get contiguous space easily. The following diagram shows an example of free space management by using the linked list approach.



Linked free space list on disk

Similar to the example given for the bit map above, you can calculate the overhead for maintaining free space with linked list. We leave it as an exercise for you.

Grouping

A modification of free-list approach is to store the addresses of n free blocks in the first free block. The first $n-1$ blocks of these blocks are actually free. The last block contains addresses of the next n free blocks, and so on. The importance of this implementation is that the addresses of a large number of free blocks can be found quickly.

Counting

We keep the address of the first free block and the number n of free contiguous blocks that follow the first block in each entry of a block. This scheme is good for contiguous allocation. Although each entry requires more space, the overall list will be shorter.

I/O Operations

A number of I/O operations (inserting, deleting, and reading a file block) needed for the various allocation schemes indicate the goodness of these schemes. The following example illustrates this.

Assumptions

- Directory, Bit-map, and index blocks are in the main memory
- Worst-case and best-case scenarios
- File size of 100 blocks

Determine the number of I/O operations needed to

1. Insert a block after the 50th block
2. Read 50th block
3. Insert 101st block
4. Delete 50th block

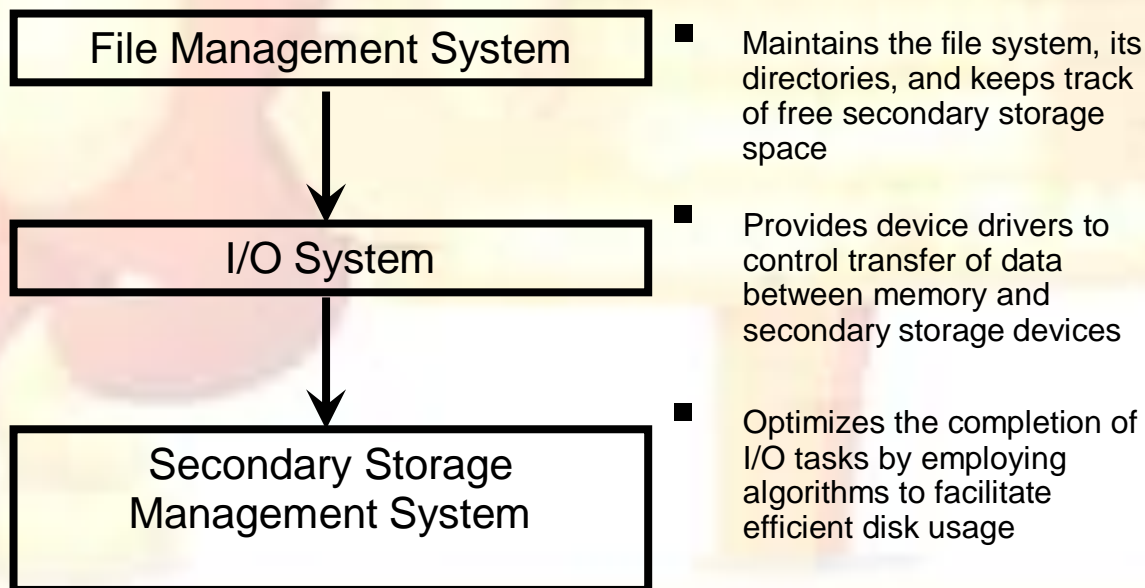
We discussed this in the lecture. Please review the lecture. Here is how we approach the first part for the worst case scenario. In the worst-case, you don't have free block before or after the file. This means that you need to identify 101 contiguous free blocks on the disk, move the first 50 blocks to the new location (read into memory and write them to the new disk location, requiring 100 I/O operations), write the new block (one I/O operation), and move the last 50 blocks to the new location (another 100 I/O operations). Since the directory entry and bit-map blocks will be modified, we need to write them to disk (two I/O operations). This results in a total of $100+1+100+2 = 203$ I/O operations.

In the best-case, we do have at least one free block available before or after the file, resulting in a total of $100+1+2 = 103$ I/O operations. 100 operations are needed for shifting (i.e., moving) the first or last 50 blocks to left or right.

You can answer the remaining questions for contiguous allocation following the same approach and reasoning. Similarly, you can answer these questions for linked and index approach. When you are done, you will realize that index allocation approach is the best because it requires the smallest number of I/O operations for various file operations.

Secondary Storage Management

The following diagram shows the hierarchy of three kernel modules used for mapping user view of directory structure, free space management, file I/O, and secondary storage management. We have discussed some details of the top-most layer. We will not discuss details of the I/O system. Here is the discussion of one of the primary functions of the lowest layer in the diagram, i.e., disk scheduling.



Three layers of file OS kernel used for managing user view of files, file operations, and file storage to disk

Disk Structure

Disks provide the bulk of secondary storage for modern computer systems. Magnetic tape was used as an early secondary storage medium but the access is much slower than for disks. Thus tapes are currently used mainly for backup, for storage of infrequently used information etc.

Modern disk drives are addressed as large one dimensional array of logical blocks, where the logical block is the smallest unit of transfer. The size of a logical block is usually 512 bytes, although some disks can be low-level formatted to choose a different logical block size, such as 1024 bytes.

The one dimensional array of logical blocks is mapped onto the sectors of the disk sequentially. Block 0 is the first sector of the first track on the outermost sector. The mapping proceeds in order through that track, then through the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to the innermost.

By using this mapping, we can – at least in theory – convert a logical block number into an old style disk address that consists of a cylinder number, a track number within the cylinder and a sector number within that rack. In practice it is difficult to perform this translation for two reasons. First, most disks have some defective sectors but the mapping hides this by substituting spare sectors from elsewhere on the disk. Second, the number of sectors per track is not a constant on some drives. On media that use a constant linear velocity (CLV) the density of bits per track is uniform. The farther a track is from the center of the disk, the greater its length so the more sectors it can hold. As we move from the outer zones to the inner zones, the number of sectors per track decreases. Tracks in the outermost tracks typically hold 40% more sectors than do tracks in the innermost zone. The drive increases its rotation speed as the head moves from the outer to the inner tracks to keep the same rate of data, moving under the head. Alternatively the disk rotation speed can stay constant and the density of bits decreases from inner tracks to outer tracks to keep the data rate constant. This method is used in hard disks and is known as constant angular velocity (CAV).

Disk Scheduling

One of the responsibilities of the operating system is to use the computer system hardware efficiently. For the disk drives, meeting this responsibility entails having a fast access time and disk bandwidth. The access time has two major components. The seek time is the time for the disk arm to move the heads to the cylinder containing the desired sector. The rotational latency is the additional time waiting for the disk to rotate the desired sector to the disk head. The disk bandwidth is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer. We can improve both the access time and the bandwidth by scheduling the servicing of disk I/O requests in a good order. Some of the popular disk-scheduling algorithms are:

- First-come-first-serve (FCFS)
- Shortest seek time first (SSTF)
- Scan
- Look
- Circular scan (C-Scan)
- Circular look (C-Look)

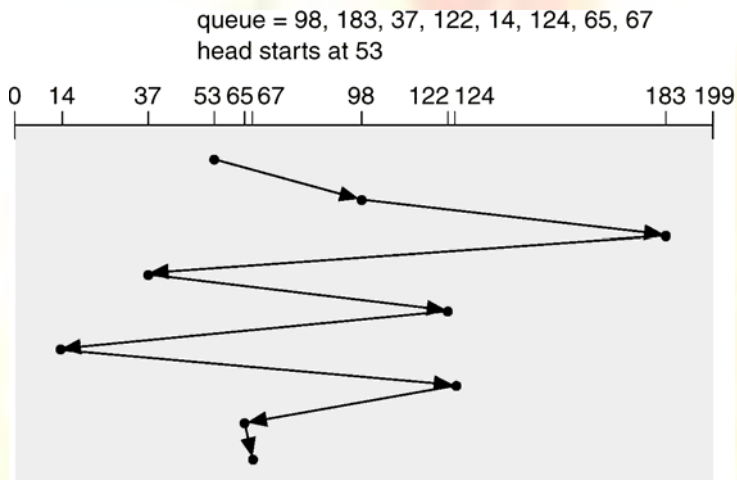
We now discuss the first four of these algorithms with an example each. We assume a disk with 200 cylinders.

First Come First Served Scheduling

The simplest form of disk scheduling is FCFS. This algorithm is intrinsically fair, but it generally does not provide the fastest service. Consider for example a disk queue with requests for I/O to blocks on cylinders

98,183,37,122,14,124,65,67

in that order. If the disk head is initially at cylinder 53 and the direction of movement is from left to right (i.e., from cylinder 0 to cylinder 199), it will first move from 53 to 98, then to 183, 37, 122, 14, 124, 65 and finally to 67, for a total head movement to of 640 cylinders.

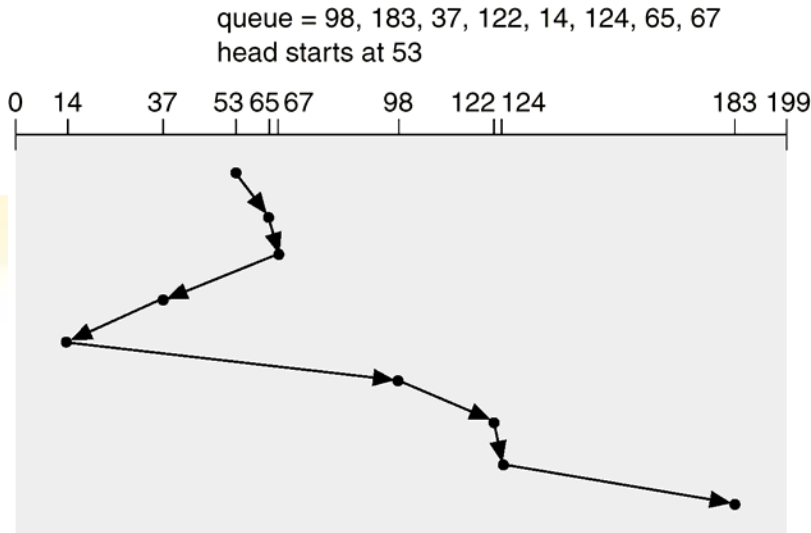


First-come-first-serve disk scheduling example

The wild swing from 122 to 14 and then back to 124 illustrates the problem with this schedule. If the requests for cylinders 37 and 14 could be serviced together before or after the requests at 122 and 124, the total head movement could be decreased substantially and performance could be thereby improved.

SSTF Scheduling

It seems reasonable to service all the requests close to the current head position, before moving the head far away to service other requests. This assumption is the basis for the shortest seek time first (SSTF) algorithm. The SSTF algorithm selects the request with the minimum seek time from the current head position. Since seek time increases with the number of cylinders traversed by the head, SSTF chooses the pending request closest to the current head position.



Shortest-seek-time-first (SSTF) disk scheduling algorithm

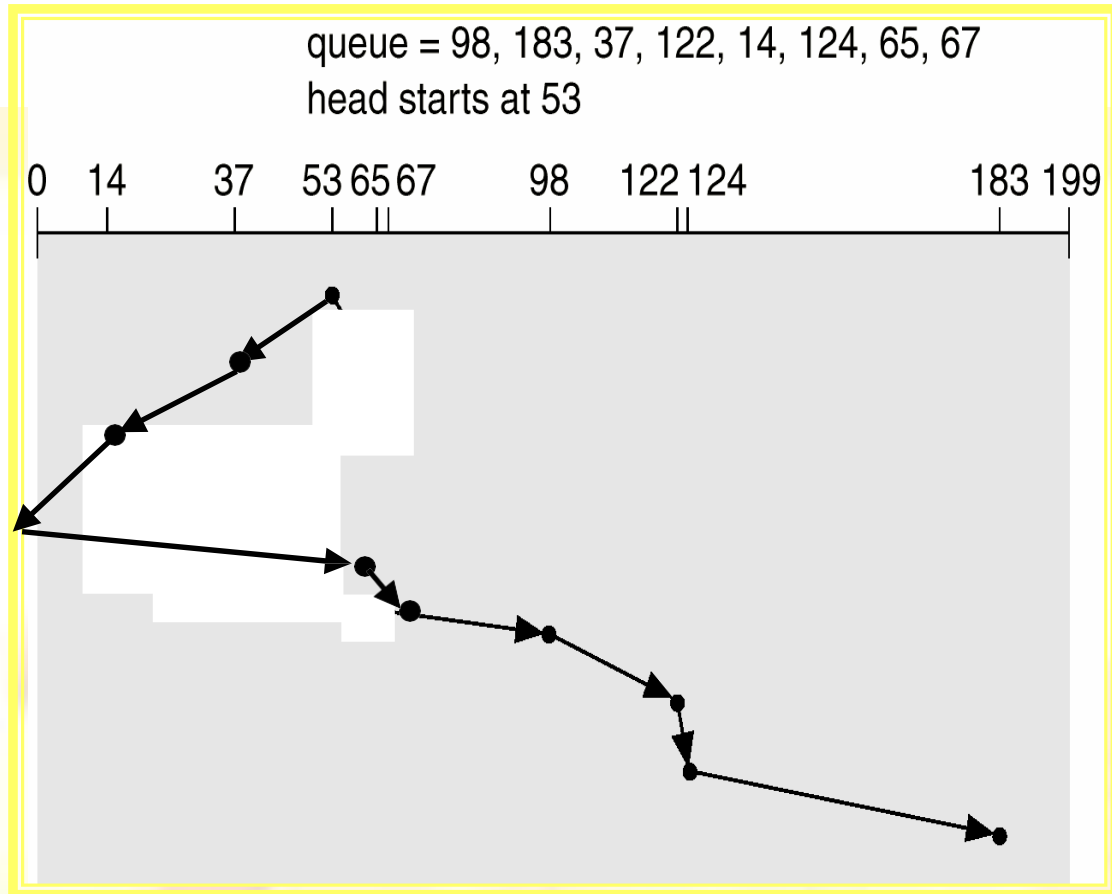
For our example request queue, the closest request to the initial head position 53 is at cylinder 67. From there, the request at cylinder 37 is closer than 98, so 37 is served next. Continuing we service the request at cylinder 14, then 98, 122, 124 and finally 183. This scheduling method results in a total head movement of only 236 cylinders—a little more than one third of the distance needed for FCFS scheduling of this request queue. This algorithm gives a substantial improvement in performance. However, it is not optimal; for the given example, the total head movement will be 208 cylinders if requests at cylinders 37 and 14 are served first.

Scan

In the Scan algorithm the disk arm starts at one end of the disk, and moves toward the other end, servicing requests as it reaches each cylinder, until it gets to the other end of the disk. At the other end, the direction of head movement is reversed and servicing continues. The head continuously scans back and forth across the disk. We again use our example.

Before applying Scan to schedule requests, we need to know the direction of head movement in addition to the head's current position. If the disk arm is moving towards 0, the head will service 37 and then 14. At cylinder 0, the arm will reverse and will move toward the other end of the disk servicing the requests at 65, 67, 98, 122, 124 and 183. The total head movement (or seek distance) is 236 cylinders. If a request arrives in queue just in front of the head, it will be serviced almost immediately; a request arriving behind the head will have to wait until the arm moves to the end of the disk, reverses direction and comes back.

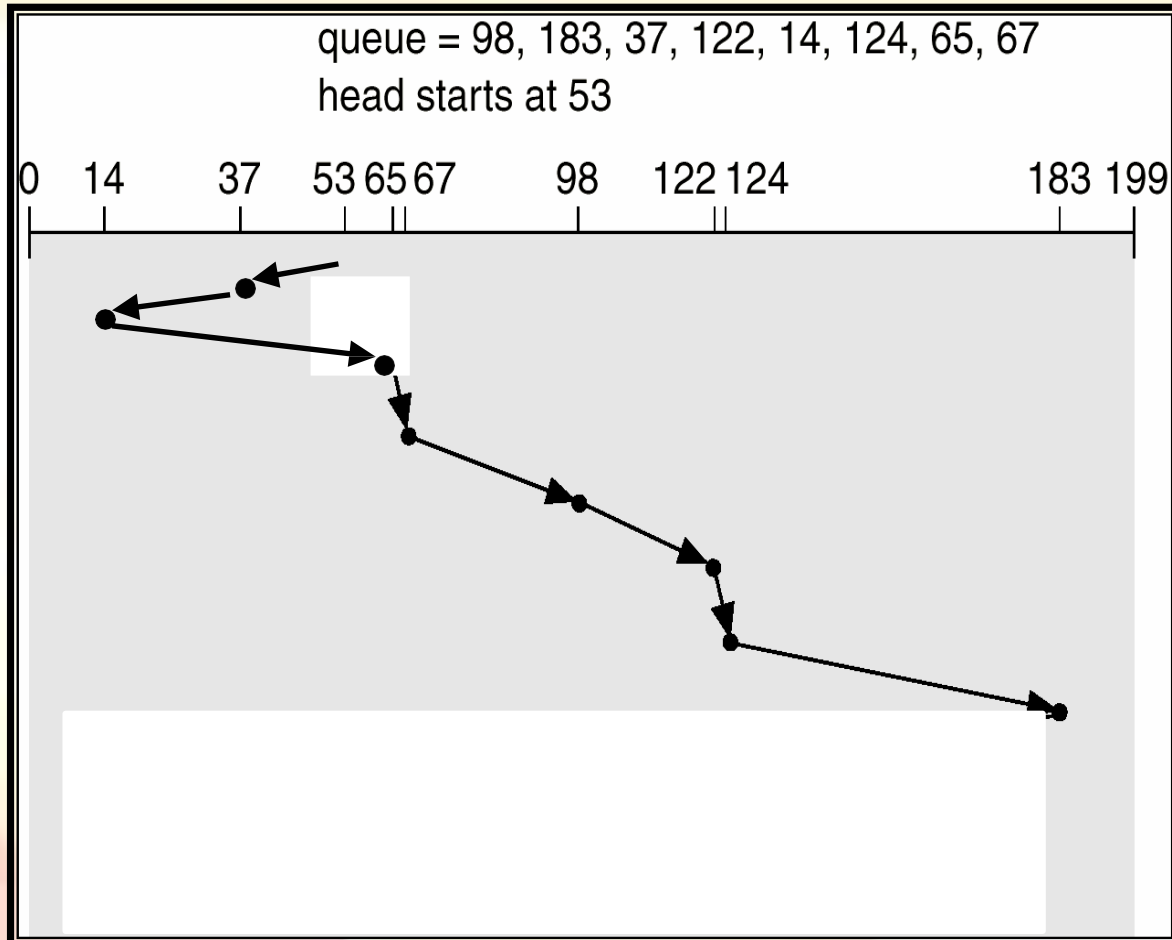
The Scan algorithm is sometimes called the **elevator algorithm**, since the disk arm behaves like an elevator in a building servicing all the requests (people at floors), going up and then reversing to service the requests going down. The figure in the following diagram shows movement of the disk head for the request queue used for the previous examples.



Scan disk scheduling algorithm with disk head moving from right to left

Look algorithm

This algorithm is a version of SCAN. In this algorithm the arm only goes as far as the last request in each direction, then reverses direction immediately, serving requests while going in the other direction. That is, it looks for a request before continuing to move in a given direction. For the given request queue, the total head movement (seek distance) for the Look algorithm is 208.



Look disk scheduling algorithm with the disk head moving from right to left

C-Scan and C-Look algorithms

In the C-Scan and C-Look algorithms, when the disk head reverses its direction, it moves all the way to the other end, without serving any requests, and then reverses again and starts serving requests. In other words, these algorithms serve requests in only one direction.