CS703ADVANCE OPERATING SYSTEM

MIDTERM FALL 2015

PAPER=1 DATED:27.12.2015

MCQs Marks=10 **Time= 1hour** **Total Marks=40**

1.	_____ multiplexing involves different programs taking turns in using the resource.
a.	Space
b.	Time
c.	Resource
d.	Turn
2.	Asynchronous exception are caused by events external to the _____.
a.	Processor
b.	Memory
c.	Vector Table
d.	Data Range
3.	A variable 'x' is shared if multiple threads reference at least _____ instance(s) of 'x'.
a.	Zero
b.	One
c.	Two
d.	Three
4.	A variable that is manipulated atomically through two operations, signal and wait, is called _____.
a.	Local variable
b.	Dutch
c.	Dutch
d.	Atomic variable
5	Uses of threads are to exploit CPU parallelism that is to run two _____ at once in the same program.
a.	Threads
b.	Fork()
c.	Kernels
d.	CPUs
6.	In _____ architecture of process assignment, Key kernel functions always run on a particular processor.
a.	Kernel level
b.	Global Queue
c.	Master/slave
d.	Peer Review
7.	In _____memory allocator application allocates and frees space.
a.	Dynamic
b.	Explicit
c.	Implicit
d.	Static
8	_____ fragmentation occurs when there is enough aggregate heap memory, but no single free block is large enough.
a.	External
b.	Internal
c.	Memory
d.	Heap
9	Acquiring two locks always in the same order prevents_____.
a.	Race conditions
b.	Data corruption
c.	Data corruption
d.	Deadlock
10	A resource allocation graph that has a cycle in it _____ means that there is a deadlock

a.	Always
b.	Never
c.	Sometimes
d.	None of above

Long Questions Marks=30

1.	Considering the resource sharing feature of threads, what do you think that 'resource sharing' is an advantage of a thread or disadvantage of a thread. State your answer briefly with strong argument.	5
2.	Why is it difficult to implement mutual exclusion and condition variables in an environment where CPUs do not share any memory	5
3.	Describe a reference pattern that results in severe external fragmentation in an allocator based on simple segregated storage.	5
4.	The Linux scheduler implements soft real-time scheduling. What features are missing that are necessary for some real-time programming tasks? How might they be added to the kernel?	10

MIDTERM FALL 2015

PAPER=2 DATED:27.12.2015

MCQs Marks=10 **Time= 1hour** **Total Marks=40**

1.	Asynchronous exception are caused by events external to the _____.
a.	Processor
b.	Memory
c.	Vector Table
d.	Data Range
2.	To make threads cheap and fast, they need to be implemented at the _____ level.
a.	Kernel
b.	System
c.	CPU
d.	CPU
3.	If two threads are waiting for the lock, and both see it's_____, only one grabs it.
a.	Under process
b.	Free
c.	In Addressspace
d.	In Namespace
4.	A queue of threads waiting for something inside a critical section is called _____.
a.	Local Variable
b.	Global Variable
c.	Condition Variable
d.	Semaphore
5	A function is called _____ if and only if it accesses no shared variables when called from multiple threads.
a.	Atomic
b.	Safe
c.	Reentrant
d.	Unsafe
6.	_____ Algorithm allows the sum of maximum resource needs of all current threads to be greater than the total resources, as long as there is some way for all the threads to finish without getting into deadlock.

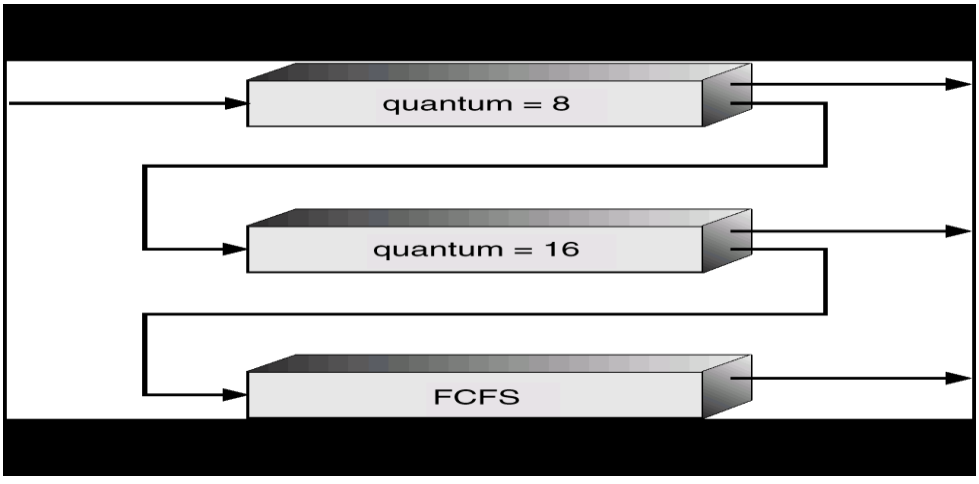
a.	Dijkstra
b.	FCFS
c.	Bankers
d.	Round Robin
7.	In _____memory allocator application allocates and frees space.
a.	Dynamic
b.	Explicit
c.	Implicit
d.	Static
8	To show free block in implicit list method we use _____ digit.
a.	0
b.	00
c.	11
d.	1
9	In reference counting, each _____has "ref count" of pointers to it.
a.	Object
b.	Block
c.	Heap Address
d.	Stack
10	A task in Linux that is present in the run queue (ready queue) at any given time has its state set to_____.
a.	TASK_INTERRUPTIBLE
b.	TASK_STOPPED
c.	TASK_RUNNING
d.	TASK_ZOMBIE

Long Questions Marks=30

1.	Considering the resource sharing feature of threads, what do you think that 'resource sharing' is an advantage of a thread or disadvantage of a thread. State your answer briefly with strong argument.	5

2.	The ctime_ts function is thread-safe, but not reentrant. Explain your answer with valid reasons.
	
	char *ctime_ts(const time_t *timep, char *privatep)
	
	{
	
	char *sharedp;
	
	P(&mutex);
	
	sharedp = ctime(timep);
	
	strcpy(privatep, sharedp); /* Copy string from shared to private */
	
	V(&mutex);
	
	return privatep;
	
	}
		5

3.	Out of MVT (Multiprogramming with Variable Tasks) and MFT (Multiprogramming with Fixed Tasks), which one do you think is best suited to cause internal fragmentation and which one is best suited to cause external fragmentation? Justify your answer.	5
4.	An example diagram for multilevel feedback queue scheduling is given below.	10



Let us consider that there are two processes. Process P1 time period is 18 and process P2 time period is 27. Answer the following questions

a) Which process will complete its execution in 2nd queue?
b) How each process will move across different queues i.e. how each process will enter into the system and how each process will move up and down between different queues.

MIDTERM FALL 2015
PAPER=3 DATED:27.12.2015

MCQs Marks=10 **Time= 1hour** **Total Marks=40**

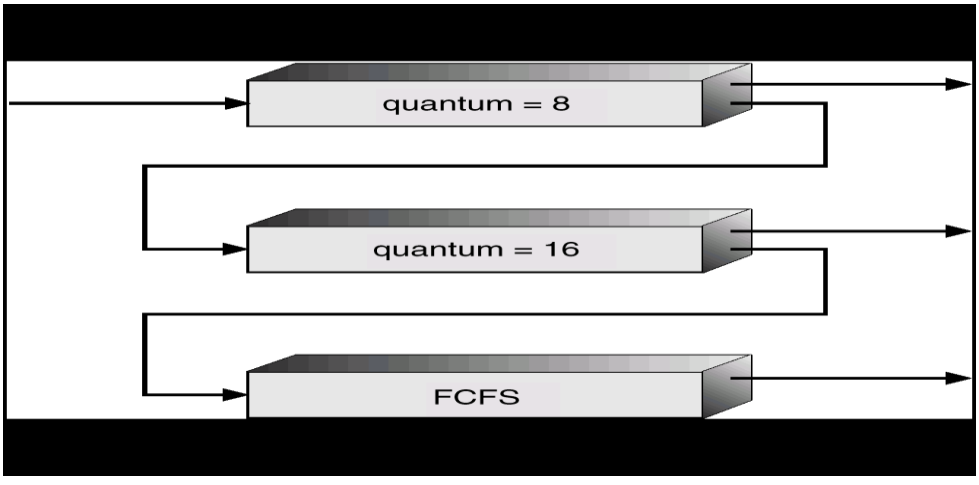
1.	Asynchronous exception are caused by events external to the _____.
a.	Processor
b.	Memory
c.	Vector Table
d.	Data Range
2.	To make threads cheap and fast, they need to be implemented at the _____ level.
a.	Kernel
b.	System
c.	CPU
d.	User
3.	If two threads are waiting for the lock, and both see it's_____, only one grabs it.
a.	Under process
b.	Free
c.	In Addressspace
d.	In Namespace
4.	A queue of threads waiting for something inside a critical section is called _____.
a.	Local Variable
b.	Global Variable
c.	Condition Variable
d.	Semaphore
5	A function is called _____ if and only if it accesses no shared variables when called from multiple threads.
a.	Atomic
b.	Safe
c.	Reentrant

d.	Unsafe
6.	_____ Algorithm allows the sum of maximum resource needs of all current threads to be greater than the total resources, as long as there is some way for all the threads to finish without getting into deadlock.
a.	Dijkstra
b.	FCFS
c.	Bankers
d.	Round Robin
7.	In _____memory allocator application allocates and frees space.
a.	Dynamic
b.	Explicit
c.	Implicit
d.	Static
8	To show free block in implicit list method we use _____ digit.
a.	00
b.	11
c.	1
d.	0
9	In reference counting, each _____has "ref count" of pointers to it.
a.	Object
b.	Block
c.	Heap Address
d.	Stack
10	A task in Linux that is present in the run queue (ready queue) at any given time has its state set to_____.
a.	TASK_INTERRUPTIBLE
b.	TASK_STOPPED
c.	TASK_RUNNING
d.	TASK_ZOMBIE

Long Questions Marks=30

1.	Considering the resource sharing feature of threads, what do you think that 'resource sharing' is an advantage of a thread or disadvantage of a thread. State your answer briefly with strong argument.	5

2.	The ctime_ts function is thread-safe, but not reentrant. Explain your answer with valid reasons.
	
	char *ctime_ts(const time_t *timep, char *privatep)
	
	{
	
	char *sharedp;
	
	P(&mutex);
	
	sharedp = ctime(timep);
	
	strcpy(privatep, sharedp); /* Copy string from shared to private */
	
	V(&mutex);
	
	return privatep;
	
	}
		5

3.	Out of MVT (Multiprogramming with Variable Tasks) and MFT (Multiprogramming with Fixed Tasks), which one do you think is best suited to cause internal fragmentation and which one is best suited to cause external fragmentation? Justify your answer.	5
4.	An example diagram for multilevel feedback queue scheduling is given below.  Let us consider that there are two processes. Process P1 time period is 18 and process P2 time period is 27. Answer the following questions a) Which process will complete its execution in 2nd queue? b) How each process will move across different queues i.e. how each process will enter into the system and how each process will move up and down between different queues.	10

MIDTERM FALL 2015

PAPER=4 DATED:27.12.2015

MCQs Marks=10 **Time= 1hour** **Total Marks=40**

1.	A/An _____ is a transfer of control to the operating system in response to some event.
a.	Thread
b.	Event Handler
c.	Exception
d.	Context Switching
2.	_____ creates a new process with its own identity which is allowed to share the data structures of its parent.
a.	Init
b.	Pthread
c.	Clone
d.	Fork
3.	On a multiprocessor system, interrupt disable doesn't provide_____.
a.	Redundancy
b.	Usability
c.	Atomicity
d.	Concurrency
4.	A queue of threads waiting for something inside a critical section is called _____.
a.	Local Variable
b.	Global Variable

c.	Condition Variable
d.	Semaphore
5	In bottom up view, an operating system acts as a/an _____manager of a complex system.
a.	Data
b.	Event
c.	Resource
d.	Application
6.	A process _____ defines the readiness of the process to be scheduled for execution.
a.	Scheduler
b.	Counter
c.	State
d.	Executer
7.	Explicit reaping is only required for _____ running processes.
a.	Short
b.	Medium
c.	All
d.	Long
8	Condition variables are not commutative. That's why they must be in a _____.
a.	Semaphore
b.	Critical Section
c.	Monitor
d.	Mutex
9	Functions that are called from a thread must be thread _____.
a.	Safe
b.	Oriented
c.	Dependent
d.	Independent
10	A set of processes is _____if each process in the set is waiting for an event that only another process in the set can cause.
a.	Randomized
b.	Mutually Exclusive
c.	Muti Threaded
d.	Deadlocked
11	Linux scheduling builds on traditional UNIX multi-level feedback queue scheduler by adding _____ new scheduling classes.
a.	One
b.	Two
c.	Three
d.	Five
12	_____Scheduling assigns priorities to tasks on the basis of their periods.
a.	Period
b.	Rate Monotonic
c.	Priority
d.	Time Monotonic
13	_____list is linear time in number of free blocks instead of total blocks and much faster allocates when most of the memory is full.
a.	Short
b.	Linked
c.	Implicit
d.	Explicit

Long Questions Marks=30

1.	While creating threads, there exists a possibility that either thread is created or it does not. You are required to specify that how a thread is created and what are the conditions that may be the cause in failure of thread creation.	5
2.	There are number of scheduling algorithm which belongs to different scheduling schemes: may belong to preemptive and non preemptive schemes. What is the main difference between preemptive and non preemptive scheduling? Write down the name of one preemptive and one non preemptive scheduling algorithms.	5
3.	Describe the working of explicit and implicit memory allocators.	5
4.	Determine the minimum block size for each of the following combinations of alignment requirements and block formats. Assumptions: Implicit free list, zero sized payloads are not allowed, and headers and footers are stored in 4-byte words. Alignment — Allocated block — Free block — Minimum block size (bytes) Single word — Header and footer — Header and footer — _____ Single word — Header, but no footer — Header and footer — _____ Double word — Header and footer — Header and footer — _____ Double word — Header, but no footer — Header and footer — _____	10

MIDTERM FALL 2015

PAPER=5 DATED:27.12.2015

MCQs Marks=10 **Time= 1hour** **Total Marks=40**

1.	An Operating System typically has a built-in communication infra-structure that implements a _____ service to map a given name to a destination machine.
a.	Mapping
b.	Finding
c.	Name lookup
d.	Looping
2.	Explicit reaping is only required for _____ running processes.
a.	Short
b.	Medium
c.	All
d.	Long
3.	Linux has a small number of kernel threads that run continuously in the kernel called _____ kernel threads.
a.	External
b.	Link
c.	Batch
d.	Internal
4.	When semaphores are used for mutual exclusion, the semaphore has an initial value of 1 and _____is called before the critical section.
a.	O( )
b.	P( )
c.	Q( )

d.	V( )
5	Uses of threads are to exploit CPU parallelism that is to run two _____ at once in the same program.
a.	Threads
b.	Fork()
c.	Kernels
d.	CPUs
6.	Maximize _____is to keep CPU and I/O devices busy and recurring them with OS scheduling.
a.	Throughput
b.	Latency
c.	Performance
d.	Utilization
7.	The system will not interrupt a _____thread except another FIFO thread of higher priority becomes ready.
a.	SCHED_OTHER
b.	SCHED_FIFO
c.	SCHED_LIFO
d.	SCHED_RR
8	_____Scheduling assigns priorities to tasks on the basis of their periods.
a.	Period
b.	Rate Monotonic
c.	Priority
d.	Time Monotonic
9	In reference counting, each _____has "ref count" of pointers to it.
a.	Object
b.	Block
c.	Heap Address
d.	Stack
10	Identify the strong program symbols from the given code? **1.int foo=5;** **2.p1() {** **}**
a.	Statement 1 is Strong
b.	Statement 2 is Strong
c.	Both statements are Strong symbols.
d.	Both statements are Weak symbols.

Long Questions Marks=30

1.	The synopsis of the fork system call is as follows: You are required to analyze/explain briefly the logic of the given code.
	
	#include<sys/types.h>
	#include <unistd.h>
	pid_t fork(void);
	main()
	if (pid = fork())
	{
	/* parent */
	……
	pid = wait(&status);
	}
		5

2.	A system has two processes and three identical resources. Each process needs a maximum of two resources. Is deadlock possible? Explain your answer.	5
3.	Describe a reference pattern that results in severe external fragmentation in an allocator based on simple segregated storage.	5
4.	What is the algorithmic order of the worst case allocation time and worst case freeing times of the following dynamic memory allocation techniques. a. Implicit free lists b. Explicit free lists (with coalescing while freeing) c. Simple Segregated free lists (without coalescing while freeing)	10

Question 1

A total of how many processes would be created in the function fork test () given below, excluding the main process that called fork test()? How many times will “Whatsup” be printed. Briefly explain your answer.

```
void fork_test( ) {
if (fork() != 0) {
if (fork() != 0)
fork();
else
printf(“Hello 1 \n”);
} else {
if (fork() != 0) {
printf(“Hello \n”);
} else {
fork();
}
}
printf("Whatsup \n");
}
```

Answer:

Five process create in this process excluding the fork test (). Whatsup line prints six times in this coding because each process print this line. fork test () method contain (fork()!=0) condition that is false in case of child process because child process fork() return 0 value. This condition is true for parent process because it return non zero value. If it return -1 then it contain error. First we consider parent process. It enter to this method and then check again condition that is false for child process and true for parent process. Parent process again call fork() method and make child process. Second (fork()!=0) condition that is false for child process run else portion of if condition and print hello line. Now we consider child process of 1st condition (fork()!=0) that is false for child process so we come to the else portion. In the else portion there is another (fork()!=0) condition. That is true for parent so parent process print hello line and this condition false for child process so it come to else portion and call fork() method and make child process. At the end each process print Whatsup line. As we know that is 6 process so it print 6 times Whatsup line

Question 2

Describe what problems would happen when multiple threads will execute the following C statement; also justify your answer with proper reasons.

```
{
static int I;
I++;
}
```

Answer:

1. **Incorrect ordering:** When two thread increment the counter but the result is 1 instead 2.
2. **Concurrence issues:** when executing static methods concurrently are static fields, which only exist one time. So, if the method reads and writes static fields, concurrence issues can occur In concurrent environment two or many threads can run concurrently, if all threads are trying to update some shared

resource concurrently, there might be chance that threads trying to read shared resource get stale data and produce wrong outputs.

3. **Deadlock problem:** A deadlock occurs when each of two threads tries to lock a resource the other has already locked. Neither thread can make any further progress. Deadlock occur when two of many thread are waiting for two or more resources, where each thread need to get access on all resources to progress and those resources are acquired by different threads and waiting for other resources to be release, which will not be possible ever. For example, two threads A and B need both resources X and Y to perform their tasks. If thread A acquires resource X and thread B acquires Y and now both threads are waiting for resources to be release by other thread. Which is not possible and this is called deadlock. You need to take care of the way of your synchronization if this is going to be a cause of deadlock.
4. **Race condition problem:** A race condition is a bug that occurs when the outcome of a program depends on which of two or more threads reaches a particular block of code first. Running the program many times produces different results, and the result of any given run cannot be predicted. In a multithreaded application, a thread that has loaded and incremented the value might be preempted by another thread which performs all three steps; when the first thread resumes execution and stores its value, it overwrites objCt without taking into account the fact that the value has changed in the interim. When writing multi-threaded applications, one of the most common problems experienced are race conditions. A race condition occurs when two or more threads can access shared data and they try to change it at the same time. Because the thread scheduling algorithm can swap between threads at any time, you don't know the order in which the threads will attempt to access the shared data. Therefore, the result of the change in data is dependent on the thread scheduling algorithm, i.e. both threads are "racing" to access/change the data A "race condition" exists when multithreaded (or otherwise parallel) code that would access a shared resource could do so in such as way as to cause unexpected results.
5. **Visibility problem:** Visibility of shared object could be different for different threads. This is one of the problems of multi-threading environment. Suppose there are two threads, one is writing any shared variable and other is reading from shared variable. When reading and writing will be occur in different threads, there is no guarantee that reader thread will see the value written by writer thread. This problem of multi-threading is known as visibility issue of shared resource. This problem can be resolved with proper synchronization of shared resource.

Question 3 Reader/writer locks are specialized locks used to solve the readers/writers problem. Consider the following pseudo-code implementation of reader-writer locks which is a variant of the readers/writers solution discussed in lectures but implemented via semaphores. Note that readers must call the function AcquireReadLock before reading the data while writers must call AcquireWriteLock before modifying or updating the data. Once data access has been completed, the locks must be released by calling the ReleaseReadLock() and ReleaseWriteLock() functions respectively :

Class ReaderWriterLock
{
Semaphore mutex = 1; // declaration of a semaphore
OkToRead = 0; // a flag to check it is OK to read the database
OkToWrite = 0; // a flag to check it it is OK to write the database int
ACTIVEREADERS=0; // number of readers holding the read lock and accessing the DB
WAITINGREADERS=0; // number of readers waiting to acquire read lock
ACTIVEWRITERS=0; // number of writers that have acquired write lock
// (practically this will always be 1)
WAITINGWRITERS=0; // number of writers that are waiting for write lock 3 CS-703 Advance Operating Systems Virtual University of Pakistan

```
void AcquireReadLock() {
P(mutex); //remember that P operation on a semaphore means decrement value
if ((ACTIVEWRITERS == 0) {
V(OkToRead); // remember that V operation on a semaphore
means incrementing its value
ACTIVEREADERS++;
} else {
WAITINGREADERS++;
}
V(mutex);
P(OkToRead);
}
void ReleaseReadLock() {
P(mutex);
ACTIVEREADERS--;
if ((ACTIVEREADERS == 0) && (WAITINGWRITERS > 0)) {
V(OkToWrite);
ACTIVEWRITERS++;
WAITINGWRITERS--;
}
V(mutex);
}
void AcquireWriteLock() {
P(mutex);
If (ACTIVEWRITERS + ACTIVEREADERS == 0) {
V(OkToWrite);
ACTIVEWRITERS++;
} else {
WAITINGWRITERS++;
}
V(mutex);
P(OkToWrite);
}
void ReleaseWriteLock() {
P(mutex);
ACTIVEWRITERS--;
if (WAITINGWRITERS > 0) {
V(OkToWrite);
ACTIVEWRITERS++;
WAITINGWRITERS--;
} else {
while (WAITINGREADERS > 0) {
V(OkToRead);
ACTIVEREADERS++;
WAITINGREADERS--;
}
}
V(mutex);
} } // end of class 4 CS-703 Advance Operating Systems Virtual University of Pakistan
```

(a) Briefly explain why AcquireReadLock() and AcquireWriteLock() functions perform P and V operations on the mutex semaphore in the above code?

Answer:

Both AcquireReadLock() and AcquireWriteLock() functions perform P and V operations for mutual exclusion on mutex semaphore. As we know P operation on a semaphore means decrementing its value and V operation on a semaphore means incrementing its value . P() make mutex value zero so that no one can enter in critical section. And when reader or writer out from the critical section then they perform v() on mutex and makes its value 1 so that other reader or writer can access.

(b) Briefly explain whether readers or writers could be starved due to this implementation?

Answer:

Yes, reader and writer both can face starvation.

Reader starvation In ReleaseWriteLock() function When writer out it give priority to waiting writers as:

```
if(WAITINGWRITERS>0) {
V(OkToWrite);
ACTIVEWRITERS++;
WAITINGWRITERS--;
}
```

So reader starvation occur.

Writer starvation AcquireReadLock() function when if condition will be false then else part execute and priority will be given to waiting reader. In other word when active writer not equal to zero because there are some writer then it is necessary to allow writer to access but in the given code priority is given to waiting reader.

```
if((ACTIVEWRITERS==0) {
V(OkToRead);
ACTIVEREADERS++;
} else {
WAITINGREADERS++;
}
```

So writer starvation occur.

(c) Suggest a mechanism through which a no starvation policy could be implemented. In other words, suggest in words, how would you modify the code such that a starvation-free implementation results.

Answer:

To remove writer starvation we need to modify if condition in the AcquireReadLock() function as:

if(ACTIVEWRITERS+WATINGWRITERS==0).

To remove reader starvation we need to modify the code in a way when fix number of writer access the database after this reader can access. Or when waiting readers increase from their fix limit then reader can access. 5 CS-703 Advance Operating Systems Virtual University of Pakistan

Question 4: Review the Readers/Writers problem discussed in lecture 12, write the code for Reader() and Writer() functions, when readers are given priority over writers, keeping the problem constraints in mind.

Answer:

```
Reader () {
lock.Acquire();
while (AW > 0) {
WR++;
okToRead.wait(&lock);
WR--;
} // while
AR++;
lock.Release();
Access DB
lock.Acquire();
AR--;
If (WR > 0) {
okToRead.Broadcast(&lock);
} else if (AR == 0 & WW > 0) {
okToWrite.Signal(&lock);
}
lock.Release();
}
Writer () {
lock.Acquire();
while ((AR + WR + AW) > 0) {
WW++;
okToWrite.Wait(&lock);
WW--;
}
AW++;
lock.Release();
Access DB
lock.Acquire();
AW--;
If (WR > 0)
okToRead.Broadcast(&lock);
else if (WW > 0)
okToWrite.Signal(&lock)
lock.Release(); }
}
```

Question 5: Which are the necessary conditions to hold the deadlocks?

Answer:

There are four necessary conditions and all four of them must hold simultaneously for a deadlock to occur. There conditions are Mutual Exclusion, Hold & Wait, No preemption and Circular Wait.

I. Mutual Exclusion means that there must be at least one resource which can only be used by a single process at a time i.e. non-shareable mode. If any other process request

for this resource, the requesting process must be delayed until the requested resource is released by the process already using it.
II. Hold and Wait means that a process is holding at least one resource and waiting for other resource(s) which is/are held by other process(es).
III. No preemption means that none of the processes can be forced to release the resources which are being held by them i.e. the processes will released the resources voluntarily.
IV. Circular Wait. There must be a set of processes which are waiting for resources which are being held by the other process. E.g. P0 is waiting for a resource held by P1, P1 is waiting for a resource held by P2, and so on upto Pn, and Pn is waiting for a resource held by P1.

Question 6:
Draw a resource graph to show the deadlock condition.
Following resource graph shows the deadlock condition. Here are three process P1, P2, P3 and four different Resources R1, R2, R3 and R4 each having 1, 1, 2 and 3 instances respectively (shown by filled circles inside resource rectangles).
Process requesting for an instance of resource is represented by directed edge from Process Pi to Resource Rj. Similarly, directed edge from resource instance towards any Process Pi represents that that particular instance of Resource Rj is allocated to Process Pi.
The graph shows that there are two cycles in this graph i.e.
P1 → R1, R1 → P2, P2 → R2, R2 →P3, P3 →R3, R3 →P1
and P2 →R2, R2 →P3, P3 →R3, R3 →P2
None of the processes will release the resources that it has already acquired and all of these three will remain in deadlock state.
Question 7:
How is it possible to prevent the deadlocks?
Answer:
There are four necessary conditions which must hold simultaneously for a deadlock to occur. So, deadlock can be prevented by implementing some methodology that guarantees that at least one of the four necessary conditions must not hold. By implementing this kind of strategy, it is possible to prevent a deadlock from occurring.
Relaxing first three conditions i.e. Mutual exclusion, Hold & wait, and no pre-emption are not always useful. There may be low utilization of resources & starvation.
Circular Wait is the ultimate choice. This is the most suitable condition i.e. ensuring that circular wait never holds. Simple methodology used for this purpose is order all the resources and each process always requests resources in increasing order. Hence, there may never be a process waiting for a low numbered resource holding a higher numbered resource, ensuring that occurrence of cycle is impossible. So, implementing that there will be no circular wait, deadlocks can be prevented.
Question 8:
Why it is considered that threads are harmful?
Threads are a difficult thing to handle for most programmers. Bugs can be introduced easily by the programmers while using threads and the most annoying this is that these are hard to find and even harder to fix. The things are painful for experts, so inexperienced programmers consider threads harmful. 7 CS-703 Advance Operating Systems Virtual University of Pakistan

While synchronization, it is required that threads must coordinate access to shared data with proper locks and missing a lock results in corruption of data. Threads also cause deadlocks if used inefficiently, carelessly. Main purpose of usage of threads is concurrency, but this is hard.
Hence threads are considered harmful and are avoided by normal programmers.

Question 9:

Could you give some reasons to use the threads in a beneficial way?

Threads can be used for concurrency. Responsiveness of applications is increased.
Threads should be used with care to increase responsiveness. Performance and utilization of multi-core, multi CPUs is increased with multiple threads running on multiple CPUs, each thread running in parallel on a different processor. Efficiency of high-end servers is increased.
Threads share memory and resources of the process to which they belong. Code sharing in allows an application to have several different threads of activity, all within the same address space.

Question 10:

Which are the classical issues related to the threads?

Synchronization & Deadlocks are issues to the threads and the classical problems are:
1. Bounded Buffer Problem
2. Readers and Writers Problem
3. Dining Philosophers Problem

Question 11:

Consider the following processes and their related information:

Process Arrival Time Burst Time Priority

P1 0 5 4
P2 2 8 2
P3 4 9 5
P4 6 7 1
P5 8 6 3

Use the appropriate information from the above given. Calculate response time, Waiting Time, and turnaround time and draw the Gant charts for following scheduling algorithms:

1. First come P2 P3 P4 P5
first Serve P1
0 5 13 22 29 35

Question 29:

Differentiate between Locks and Semaphores?

Answer: Locks: locks are synchronization primitives used to protect shared data structures among threads. Locks can be implemented both ways i.e. busy-waiting and blocking (sleep). Semaphores: semaphores are synchronization primitives used to protect shared data structures among threads as well as can provide coordination among different threads. Semaphores are enhancement to locks and can be used as locks (i.e. mutexes).

Question 30:

Differentiate between Semaphores and Condition Variables?

Answer: Semaphores: semaphores are synchronization primitives used to protect shared data structures among threads as well as can provide coordination among different threads. Semaphores are enhancement to locks and can be used as locks (i.e. mutexes). Condition Variables: condition variables are used to provide inter-thread coordination i.e. to coordinate events occurring in different threads. If one or more threads are waiting (blocked) for an event/signal to occur in another thread, we use condition variables to provide coordination among these threads.

Question 31:

How can we use Semaphores for thread reaping?

Answer: Semaphores can be used for thread reaping. Here semaphore is used as mutex i.e. initialized with zero (count = 0). When the we want to wait for the new thread to complete (i.e. pthead_join()) we call P() operation, which causes the calling thread to block as count = -1. The thread who is being joined, would call V() operation when going to exit. Which would signal to the blocked thread and it would resume its execution.

Question 32:

Briefly discuss "Resource Ordering" technique for preventing deadlocks?

Answer: Resource ordering is a practical and commonly used technique to prevent deadlocks. We assign a number to each resource (i.e. resource itself or the lock/semaphore associated with it) or we assign levels (i.e. these particular resources are categorized at level 1, these resources are at level 2 and so on). At each level there can be resources of multiple types. Rule: All threads will acquire the resources in same order (i.e. first level 1 resources, then level 2 resources and so on), that is, holding the resources taken from previous level, we take resources from next level.

If every thread follows that rule then there would be no chance of a deadlock, because there would be no circular wait situation.

Question 33:

Briefly discuss the problem of "Priority Inversion"?

Answer: Priority Inversion problem occurs when:

- A high priority job waits for a low priority job while a medium priority job runs on the CPU.
- Or A high priority job keeps spinning in a loop waiting for a lower priority job to release a lock, but the lower priority job never gets a chance to run on the CPU.

Question 34:
Briefly discuss the disadvantages of "Threads" paradigm?
Answer: Disadvantages of Threads paradigm are as follows:
1. It makes the overall code of the program, non-deterministic. It introduces Heisenbugs in the program. A Heisenbug is a type of bug that disappears or alters its behavior when you attempt to debug it.
2. In large, complex real life programs/problems, the programmer usually does not visualize properly that which synchronization mechanism should be deployed here in a particular situation. Hence, introducing wrong synchronization primitives in different scenarios, resulting in lots of undetectable bugs in the code.
3. As threads grow in numbers, for each thread kernel has to allocate memory internally for its stack. Hence the kernel's memory space is used for the purpose, it grows and soon reaches a limit beyond which it cannot expand. Secondly, when there are too many threads created, the benefit of parallelism achieved by overlapping I/O and CPU activities is destroyed / nullified.
4. As the number of threads grow, the overall thread management, done by a main controller thread, becomes a cumbersome job.
5. Context Switching overhead increases to a great extent, with the increase of no. of threads. Similarly thread scheduling and synchronization overhead increases with the increase in number of threads.

Question 35:
In lectures, we studied that there are three methods possible to achieve concurrency. What is the difference between second (Single Process, multiple events) and third (threads) method?
Answer: Single process, multiple events approach:
In this approach, we have a single process, when we have to block, we do not block in fact we schedule an event for future firing, and in the call back of that event a registered function is used. The process in fact tries to save itself from getting blocked. Drawbacks of this approach:
- The process has to maintain the state of each request/session.
- All requests/sessions can be in different states. So a FSM have to be maintained for each session/client.
- All these FSMs are maintained in a global table of a single process.

Threads approach:
Multiple execution states within a single address space. Instead of managing a global table for the states of every session, we offload all the session processing task to one execution stream (execution state) and another session processing to another execution stream and so on. Each execution stream maintains its data by itself.

Question 36:
What are the things included in a thread's context?
Answer: Thread's context usually consists of:
- CPU registers
- Program Counter
- Stack Pointer

Question 37:
Who generates/creates the user-level thread? And do we require a system call to generate a user-level thread?
Answer: User-level threads are created by a library (run time system). Whenever the program wants to create a thread, it calls the library, no system call is triggered, the library creates the thread itself.
Question 38:
What is the concept of "Scheduler Activation" as discussed in the lectures?
Answer: Scheduler Activation:
Suppose kernel-level thread is going to block, if kernel somehow, notify it to the application / process [that this particular kernel thread is going to block]. The run time system (user-level thread library) checks that the user-level threads which are mapped to this blocking kernel thread, do not have some lock which other kernel level threads may need. If there are such lock(s), it temporarily gets these lock(s) back. This is known as scheduler activation. This scheduler activation is used on top of hybrid threads model.
Question 39:
What is the relationship between different threads in a single process? How one thread can wait for the termination of another thread?
Answer: Different threads in a single process are peers of each other (i.e. there is no parent child relationship). A thread can wait for the termination of another thread by calling the pthread_join() function, passing the joining thread's thread ID as argument in the function call. **Question 40:**
In multithreaded programs, what are the two cases in which we need synchronization mechanisms?
Answer: The two cases are:
1. to protect shared resources/data structures
2. to provide coordination among different threads of a process

Question 41:

In threads execution, what we mean by the term "arbitrary interleaving of executions"? Answer:

Arbitrary interleaving of executions means we cannot predict the order of execution of threads, and execution can switch from one thread to another at arbitrary point of time. Secondly, we also cannot predict the rate of execution of instructions in different threads. So the goal is to write the multi-threaded program in such a way that any arbitrary interleaving of executions of threads does not make our program unpredictable. **Question 42:**

In the "Too Much Milk" problem discussed in the lecture, what is the flaw in solution#2? Answer:

Consider the situation, thread A runs, leaves note A then thread B runs, leaves note B, then thread A runs, checks condition (no Note from B), since note from B is present, hence fails then thread B runs, checks condition (no Note from A), since note from A is present hence fails, then thread A runs, removes note A, then thread B runs, removes note B. Hence, no one buys milk.

Question 43:

Consider the following ways of handling deadlock:

(1) Banker's algorithm
(2) Detect deadlock and kill thread, releasing all resources
(3) Reserve all resources in advance
(4) Restart thread and release all resources if thread needs to wait
(5) Resource ordering
(6) Detect deadlock and roll back thread's actions.

(a) One criterion to use in evaluating different approaches to deadlock is which approach permits the greatest concurrency. In other words, which approach allows the most threads to make progress without waiting when there is no deadlock? Give a rank order from 1 to 6 for each of the ways of handling deadlock just listed, where 1 allows the greatest degree of concurrency. Comment on your ordering.

(b) Another criterion is efficiency; in other words, which requires the least processor overhead. Rank order the approaches from 1 to 6, with 1 being the most efficient, assuming that deadlock is a very rare event. Comment on your ordering. Does your ordering change if deadlocks occur frequently? 22 CS-703 Advance Operating Systems Virtual University of Pakistan

Answer to Question (a)

Ser	Most to Least Degree	Comments
1	Restart thread and release all resources if thread needs to wait	If threads require waiting then resources should be released at once for use of others
2	Detect deadlock and roll back thread's actions	They cause waiting so concurrency is effected
3	Detect deadlock and kill thread, releasing all resources	"
4	Banker's algorithm	Create unwanted delay
5	Resource ordering	"
6	Reserve all resources in advance	Threads wait longer

Question 10:

Which are the classical issues related to the threads?

Synchronization & Deadlocks are issues to the threads and the classical problems are:

1. Bounded Buffer Problem
2. Readers and Writers Problem
3. Dining Philosophers Problem

Question 11:

Consider the following processes and their related information:

Process Arrival Time Burst Time Priority

P1 0 5 4
P2 2 8 2
P3 4 9 5
P4 6 7 1
P5 8 6 3

Use the appropriate information from the above given. Calculate response time, Waiting Time, and turnaround time and draw the Gant charts for following scheduling algorithms:

1. First come P2 P3 P4 P5
first Serve P1
0 5 13 22 29 35

Question 12:

A total of how many processes would be created in the function fork test () given below, excluding the main process that called fork test()? How many times will “Whatsup” be printed. Briefly explain your answer.

```
void fork_test( ) {
if (fork() != 0){
if (fork() != 0)
fork();
else
printf(“Hello 1\n”);
} else{
if (fork() != 0) {
printf(“Hello\n”);
}else{
fork();
}
}
printf("Whatsup\n");
}
```

Solution:-

Using the above code the putput of Whatsup will print total 6 times

As fork() is used to create new processes which known as child process. The child process is normally

copy of the parent, run a different branch of the program. Even this can run a completely different program.

After fork() child process and parent processes execute in similar. In the above program code the parent fort() process created when if(fork()!=0) executes, it prints the “whatsup” and after this every time when fork()called “whatsup” will print on the screen.

I write the code in unbunto window terminal and after given header file and the replace the fork() test function with main function I write the code give in the question and then compile it the execution and file creation process are display in the display screen short as given below;

Question 13:

There are p processes competing for r identical resource units. Each process needs a maximum of m resource units (where m ≤ r).

(A) Under what condition can no deadlock occur?

Answer:

Under following condition or rule, no deadlock occurs:

 Don’t allocate any more resource to a process if the process is already using some resources. Let that process consume and release those resources first.

 If all the resources are consumed and further request comes, do not put the requesting process go into wait state. Let them consume and release pre-acquired resources first.

(B) Give an example of a request sequence leading to deadlock.
Answer:
Suppose that all the resources have been allocated to P processes or a subset of P processes i.e. P1, P2… Pn.
P1 needs one more resource, but no more resource is available, P1 goes into waiting.
P2 needs one more resource, but no more resource is available, P2 goes into waiting.
P3 needs one more resource, but no more resource is available, P3 goes into waiting.
.
Pn needs one more resource, but no more resource is available, Pn goes into waiting.
At this point all the processes are collectively holding all the resources, all the processes are in wait state, no process will release the acquired resources. So the deadlock occurs here.
Question 14:
Write a resource allocation algorithm which will prevent deadlock.
Answer:
Following algorithm will prevent the deadlock:
P processes are requesting for r resources. Resource allocation algorithm does the following:

```
If (process P has ever acquired any resources before)
{
Do not serve the request;
}
Else if (no resource is remaining)
{
Do not serve the request;
}
Else
{
If (no resource is free)
{
Wait until all resources r are free;
}
Grant process P individual access to r resources;
}
```

Question 15:
How should threads coordinate? And how should they wait for an event or state change?
Answer:
A thread, while holding the lock, may change the variables constituting the condition expression. It should then call signals to notify a single waiting thread to resume execution or broadcast to wake all waiting threads. Signal and broadcast move waiting threads off the condition variable's queue and queue them on the associated lock. Only after the signaling/broadcasting thread releases the lock may these threads resume execution. However, it is possible that other threads were previously waiting on the lock, so the waiting threads are not guaranteed that the condition expression is true. Rather, returning from wait is considered a hint that expression may have become true. After resuming, the thread must test the expression again before proceeding. Hence, threads commonly wait within a while loop that tests the condition expression

Question 16:

How does thread synchronization problem change with transactional memory?

Answer:

Transactional memory changes the semantics of both waiting and signaling. For example, most condition variable implementations warn against “naked notifies”, or signaling without holding the condition mutex lock. However, signaling within a transaction raises the possibility that the signaling transaction may abort, so the condition it signaled never occurs. Furthermore, we observe that under some conflict resolution policies a recently woken transaction may cause a signaling thread to abort.

Question 17:

What are conditional critical regions? What are their limitations? And how can we overcome these limitations.

Answer:

Conditional critical regions are the regions in a program that must be executed distinctly by one process and no other process can execute that region at the same time. When we use condition variables for locking purposes then the critical regions are referred to as conditional critical regions.

The limitations of conditional critical region are:

- With conditional critical regions, waiting occurs without any changes to shared state.
- The resource concept is unreliable
- The context switching is inefficient
- The scheduling mechanism is too restrictive
- These limitations can be removed using conditional variables.

Question 18:

What are condition variable? What are their limitations? And how can we overcome these limitations?

Answer:

Condition variable are queues of waiting threads. Condition variables support three operations: wait, signal and broadcast. Logically, every condition variable is associated with one (or more) Boolean condition expressions and a mutex lock. Condition variables are used to complement locks by allowing a program to specify the order of execution.

There are two limitations for condition variables: (1) the signal operation is explicit, so a programmer may forget to call signal when changing shared state, (2) condition variables do not nest within multiple levels of mutex locks because only the inner-most lock is released when waiting. As a result, deadlock may result if the blocked thread can only be woken by code that acquires a lock still held by the waiting thread.

These limits are removed when we use conditional variables with transactional memory.

Question 19:

What is lost wakeup problem and how can it be resolved?

Answer:

Signaling or broadcasting transaction may execute concurrently with a transaction that it woke up. If the waking transaction completes before the signaling transaction, it is called wakeup problem. The waking transaction may view shared state from before the signaling transaction, and hence before the logical condition it awaits is visibly changed.

The wakeup problem may be resolved by deferred signal approach, or by speculative signal approach

Question 20:

What are the requirements and implications for the discussed two implementations of condition variables for transactional memory?

Answer:

The requirements and implications for deferred signal approach are Commit Actions
The requirements and implications for Speculative signal approach are Escape Actions, and Robust Conflicts Detections

Question 21:

Explain, in your own words, how the proposed approach manages to avoid deadlocks when unstructured locking is used.

Answer:

The proposed approach says that the deadlock cannot occur when we have the requested lock and its future lock set. The future lockset is the set of locks when the lock is requested and all locks in between till a matching unlock is found. The proposed approach starts by tracking the simple effects. Simple effects may be the empty sequences of locks and unlocks events. Proposed approach uses a continuation effects i.e. the effect of code to its following expression. The proposed approach has the feature to add notes to the lock operation according to their continuation effects.

The continuation effects are calculated statically, but the future lock sets are computed at run time. When a function is called the continuation effects is pushed on to the stack up to the time the call is not fully executed. During the continuation effect, the locks are identified and added to the lock set. At runtime the lock operation unblocks whenever we find a lockset. This assists to avoid deadlocks. The proposed approach gives only a single lock to every lock operation, this increase the degree of parallelism.

Now let us see constitutes of the proposed approach and its implementation in various programming language constructs.

Effects: Simple effects may be the empty sequences of locks and unlocks events. Continuation effect i.e. the effect of code to its following expression. Notes are added to the lock operation according to their continuation effects. The continuation effects are calculated statically, but the future lock sets are computed at run time. When a function is called the continuation effects is pushed on to the stack up to the time the call is not fully executed. During the continuation effect, the locks are identified and added to the lock set. At runtime the lock operation unblocks whenever we find a lockset. This assists to avoid deadlocks. The proposed approach gives only a single lock to every lock operation, this increase the degree of parallelism. The runtime system utilizes the dynamically computed future locksets so that each lock operation can only proceed when its future lockset is available to the requesting thread.

Function Calls: As we know that the continuation approach is intra-procedural so the unlock operation for a lock operation may not reside in the same function. This problem is resolved by adding the notes to function call according to the continuation effect. As static analysis says ensures that there is always an unlock operation for a lock, so the algorithm traverse the lockset and find the matching unlock terminates.

Conditionals: A demerit defining effects as ordered events is that, in case of conditional expressions/statements, it becomes less likely that the branches will have the same effect.

Proposed approach can deal with it. Proposed approach traversed each branch and keep tracks of each branch, and algorithms calculates the lockset for each branch individually.

Loops and recursions: loops and recursions introduce additional problems. In the case of recursion and loops, function and its body name must express the same name. But this is no possible, due to the reason that the two effects cannot be structure-wise equivalent: the effect of a function name is contained in the effect of its body, due to the recursive call. To handle this problem, a summary is tagged to the function names showing the effect of their bodies.

Question 22:

How static analysis is performed and what are its limitations?

Answer:

Static analysis is performed in the following four phases:

Pointer Analysis: First, the heap and stack state is formulated at each point of the program by doing a pointer analysis that is based on symbolic expressions. The analysis has been modified to treat the heap allocation in a context-sensitive manner. The out of first stage is a mapping for each expression to a set of abstract locations. Each abstract location may have any forms e.g. a formal parameter, a global variable or a heap-allocated location.

Effect Interference: we have a functional control flow graph in the analysis. Here a forward dataflow algorithm is run on the graph to compute the effect for each function. Nodes of the graph show input, current and output effects. The input effects are formulated from its front edges. The output effect is computed by appending the current effect to the input effect and is propagated to a node's successors until a fixed point is reached. Output effects that have no successors are joined to computer a function's effect.

Loops: at this phase, lock counts effects flowing from back edges must be equivalent to the input effect of the same node. Due to this bound, we can efficiently encode loop effects: from the entrance to exit the counts of each lock should match. We take effect of the entire loop. The empty effect on all branches compensates for the case where a loop is not executed.

Effect Optimization: function effect repetitions should be avoids. For this optimization techniques are applied. One technique is to find the common prefixes and suffixes so that the number of branches may be reduced. Another technique for removing the nested loop is to take the nested loops out. We also trigger the data flow algorithm, which is CPU-intensive, only for functions that are known to contain lock operations, to avoid additional overheads.

Limitations are as follows:

Non C code: C language can be tightly dealt with our static analysis. However, library code cannot be handled by our analysis. In our assumptions, we take the library functions' effect as empty effects. However, we can add user-defined notes for library functions. There is another limitation with our analysis that it cannot handle non-local jumps (including signals) and inline assembly.

Pointer analysis: our analysis is not capable of handling pointers arithmetic that involves locks in it. Our analysis also fails to track heap allocation at recursive functions and loops. Another limitation is that we need that lock pointers are changed only before they are shared between threads, and that locks pointed by with at least two levels of indirection are not aliased at function calls.

Conditional Execution: if the locks and their matching unlock operation happens to be executed in separate conditional statements, and they have similar guards, in this case, our analysis will not accept the program.

Question 23:

What is are message-oriented models?

Mechanism of this model is passing messages/events among processes with an easy an efficient way. Such model also provides the facility of putting the messages into the queues unless they are served. Messages have the states of wait and ready etc. pre-emption for high priority messages is facilitated. Message-oriented models bear these features:

- Special types of processes communicate among each other with special paths like sockets, ports, or channels etc.
- Process communication paths are static. Creating, deleting or changing connection among processes is difficult.
- Processes work in their separate spaces and rarely access the shared data
- Message-oriented models bear the following characteristics:
- Process synchronization is made possible through queues that are attached with processes
- When more than one processes operate on data, the data structure are passed by reference
- Peripheral devices are also work like messages. Signal is sent to devices and an interrupt is generated by devices in response
- Priorities for process are set statically at the time of system design
- Mostly one process is served at a time and served completely before looking into the queue for next process

Messages-oriented models provide the following facilities:

- It carries a handle for identifying a message, and message data portion
- Messages communication is established through channels and ports.
- Messages are sent and received through operations like SendMessage, AwaitReply, WaitForMessage, and SendReply etc.

Question 24:

What is Procedure-Oriented Model?

Mechanism of this model is quickly switching the context of processes by giving the protection and addressing mechanism. Processes use sharing of data extensively. Inter process communication is secured by semaphores, locks, monitors etc. Pre-emption of resources is only given to high priority processes after the resources are released by the using process. Procedure-oriented models bear these features:

- Processes access global data in a synchronized and controlled way.
- As a process and no communication channel associated with it so process creation is easy. Process deletion is also easy when a process holds no lock.
- Process is assigned a single task to complete and the process wanders at many location and states while changing its context.

Procedure-oriented models bear the following characteristics:

- Process synchronization in case of many resources is made possible through locks
- Process are made to share the data, and data access is given for shortest possible time and smallest possible part of resource
- Control of devices and interrupts from devices are also dealt with the help of locks

 Process are given priority dynamically, this priority is on the basis of short time of completion
 To save the context switching from conflicts, global naming is important

Procedure-oriented models provide the following facilities:
 They provide procedures. Procedures have algorithms, local data, parameters and results
 They provide synchronous and asynchronous procedure calls
 FORK, JOIN, WAIT, SIGNAL calls are provided for processes and protection of processes

Question 25:
What factor must be considered while choosing message-oriented or procedure-oriented system model?
We have seen that in the paper that both message-oriented and procedure-oriented models are counter parts of each other. System designed under any one model and be transformed into the other more with confidence. Same functionality can be achieved through the implementation of any of these two models. So why to choose and prefer over the other? For the answer of this question we will have to think two steps down. Process and synchronization facilities are things which make one or the other style more attractive or more tedious.
For example there is a system which is more tending towards utilizing the virtual memory mechanism to perform its tasks; here it is easier to distribute the memory into messages blocks and queue messages; but it is very difficult to protect the shared memory; so in such system it is a better decision to choose a message-oriented model.
There is another system which is more tending towards the usage of stack and block structured allocation. In such case we can choose the procedure-oriented model.
Following factors may also be considered while choosing one these models:
 Organization of real and virtual memory
 Space and size of the word that is used to save the state of each process context switch
 The ease with which process scheduling and dispatching can be accomplished
 The control and arrangement of peripheral devices and interrupts generated by these devices
 Architecture of the register that can be programmed
 Architecture of the instructions

Although the above factors can be considered while choosing one of the models, but it is to be noted that it is very rare that a programmer has a clear influence by one of them over the other. And the other thing even rarer is the fact that a programmer has a full knowledge of the two models and factors influencing them.
Question 26:
How are address translations and physical memory managed in the Exokernel?
For address translation, the virtual address space is divided into two portions. First portion keeps the application data and code. Virtual addresses in this portion are mapped and typically hold exception handling code and page-tables.
Following actions will take place On a TLB miss:
 Exokernel checks where the virtual address is located. In case of the standard user segment, the exception is dispatched directly to the application. In case of the second segment and guaranteed mapping, exokernal installs the TLB entry and continues; and in case of second segment and not guaranteed mapping exokernal forwards it to the application.
 The application searches the virtual address in its page-table structure and, if the access is not valid, an appropriate exception is generated. If the virtual address mapping is allowed,

the application constructs the appropriate TLB entry and its associated capability and invokes the appropriate exokernal routine.

 Exokernel then checks whether the access is allowed are not. If access is allowed then mapping is installed in the TLB and control is returned to the application. And if the access is not allowed an error is returned.

 At the last application does some cleanup work and then resumes its execution.

For efficient virtual memory management, TLB must be refilled fast. To do this , Exokernal use the caching for TLB entries in the kernel. Exokernal uses overlaying the hardware TLB with big software TLB (STLB) to absorb capacity misses. On a TLB miss, Exokernal first checks that the required mapping is in the STLB. If it is located there, Exokernal installs it and resumes execution; in the opposite case, the miss is forwarded to the application.
The STLB have 8 bytes entry and it has total 4096 entries. It has direct mapping and located in unmapped physical memory. When an STLB has a “hit”, it takes 18 instructions. In the opposite case, performing an up call to application level on a TLB miss, a system call to install a new mapping is at least 3 to 6 microseconds more expensive. As we know that exokernel has a principle of exposing kernel bookkeeping structures, the STLB can be mapped to a capability that can search the entries efficiently.

Question 26:

How does Exokernel use hardware support to improve the performance?

Exokernel improve its performance by securely exposing the hardware. The main principal of the exokernel architecture is that the kernel should provide secure low-level primitives so that these primitives permit all hardware resources to be accessed as directly as possible. So the the major struggle of an exokernel designer is therefore to safely export all privileged instructions, hardware DMA capabilities, and machine resources. The exported resources are hardware i.e. physical memory, the CPU, disk memory, translation look-aside buffer (TLB), and addressing context identifiers. Due to this securely exposing hardware principle, an improvement occurs in the less tangible machine resources such as interrupts, exceptions, and cross-domain calls. Exokernel should not apply higher-level abstractions on these events. Most of the physical resources should be partitioned and sub-divided in a fine manner. The number, format, and current set of TLB mappings should be visible to library systems, and these should be replaceable by library operating systems. These should also be visible to and replaceable any “privileged” co-processor state. An exokernel must export privileged instructions to library operating systems so that they may be able to implement traditional operating system abstractions such as processes and address spaces. A system call can have an exported operation that checks the ownership of any resources involved. In negative wording, we can say that this principle states that an exokernel should avoid resource management. The resources should be managed to up to the required by protection (i.e., management of allocation, revocation, and ownership). We believe that using the hardware under this principle brings improvement in the performance because distributed, application-specific, and resource management is the best way to build efficient flexible systems.

Question 27:

Can we say that Exokernel is a micro-kernel? What abstractions does it support?

No, we cannot say that exokernel is a micro-kernel. In practice, exokernel system provides applications with greater flexibility and better performance than microkernel systems. The low level interface of Exokernel permits application-level software to manipulate resources very efficiently. Protected control transfer in exokernal is recoded 7 times faster than that of the current implementations of microkernel. Similarly, the exception dispatch is 5 time better than

that of the microkernel. Exokernel gives flexibility to the application level software, while microkernel does not provide this flexibility. For example, virtual memory is implemented at application level, where it can be tightly integrated with distributed shared memory systems and garbage collectors. Efficient protected control transfer in exokernel permits applications to build a large array of efficient IPC primitives by trading performance for additional functionality. As opposite to exokernel, microkernel systems do not give permission to un-trusted application software to build specialized IPC primitives because virtual memory and message passing services are implemented by the kernel and trusted servers. In the same way, microkernel does not provide the capability of modifying many other abstractions, such as page-table structures and process abstractions. Moreover, many of the hardware resources in microkernel systems, such as the network, screen, and disk, are encapsulated in heavyweight servers that cannot be bypassed or made suitable to application-specific needs.
Another aspect to be note is that, like microkernel systems, an exokernel can provide backward compatibility in three ways:

- Binary emulation of the operating system and its programs
- By implementing its hardware abstraction layer on top of an exokernel
- Re-implementing the operating system's abstractions on top of an exokernel

Just similar to microkernels, exokernels are designed to increase extensibility. And just opposite to traditional microkernels, an exokernel pushes the kernel interface much closer to the hardware, which allows for greater flexibility.

Question 28:

Abstractions are slow and are not tuned to all applications, how does Exokernel addresses this problem?

Most of the fixed high-level abstractions are slow. They put a resistance in the performance of application. This slow performance is due to the reason that there is no single way to abstract physical resources or to implement an abstraction that is best for all applications. When an operating system needs an abstraction to be implemented, the operating system has some restrictions to make trade-offs between read-intensive or write-intensive workloads, support for sparse or dense address spaces, etc. These tradeoffs put some penalties on performance on classes of applications. For example, relational databases and garbage collectors sometimes have very predictable data access patterns, and their performance suffers when a general purpose page replacement strategy such as LRU is imposed by the operating system. But we know that the performance improvement is very important factor in application-specific policies. To address this problem of efficiency, Exokernal uses the techniques of application-controlled file caching. And it has been measured that this technique can reduce application running time by as much as 45%.

Question 29:

Differentiate between Locks and Semaphores?

Answer: Locks: locks are synchronization primitives used to protect shared data structures among threads. Locks can be implemented both ways i.e. busy-waiting and blocking (sleep).
Semaphores: semaphores are synchronization primitives used to protect shared data structures among threads as well as can provide coordination among different threads. Semaphores are enhancement to locks and can be used as locks (i.e. mutexes

CS703 – Advanced Operating Systems - MidTerm Papers – Spring 2016

By: Asif Mansoor

My Todsy's Paper (05-06-2016):

- 10 MCQs
- IPC problem in client-server architecture
- Classes of thread-unsafe

- Uses of threads with examples
- Long-term schedulers handle OS Jobs

Q1. The maximum no. of pages in process address space is one million & total size (p+ d) of process address space is 32 bits with page size of 4096 bytes.

Answer:

p = 12 bits

off set =5 ==========

Let the number of bits required = x so, 2^x = 4096 so, x=12

offset = 5 ==========

2^p = 4096

2^p = 2^12

p =12 2^d = 32 2^d =2^5

d =5

Q2. An operating system is running three concurrent processes, P1, P2, and P3 for three different users. Each process requires six units of computing time. Process P1 consists of a single thread. Process P2 consists of two threads, each requiring three of P2's six total units of computing time. Process P3 consists of three threads, each requiring two of P3's six total units of time. Assume that the threads never block and that the processes are entirely resident in primary memory. a. Suppose that the threads are kernel-supported, and that the kernel implements preemptive, round-robin scheduling of threads, with a quantum on one time unit. Of the first six units of time, how many will be devoted to each of the three processes?

Answer:

a. Process 1 will receive one time unit, process 2 will receive 2 units, and process 3 will receive three.

b. Each process will receive two time units.

c. Scheduler (b) is a fair share scheduler. Scheduler (a) is not. Scheduler (a) could be modified to become a two-level scheduler. The top level would select a process to run, using a fair scheduling discipline like round-robin. The bottom level would select one thread from the process that was selected by the top level scheduler.

Q3. What is the way that two processes cannot enter in critical section simultaneously?

Answer:

A solution to the critical section problem must show that • mutual exclusion is preserved

• progress requirement is satisfied • bounded-waiting requirement is met.

Q4. Choose the best algorithm for the scenario, and scenario was given. and this q is related to FIFO, SJF, and RR.

Answer:

RR was the best answer of this question. Solve it according to the scenario give.

Q. Consider a variant of the RR scheduling algorithm where the entries in the ready queue are pointers to the PCBs.

a. What would be the effect of putting two pointers to the same process in the ready queue?

b. What would be the major advantages and disadvantages of this scheme?

c. How would you modify the basic RR algorithm to achieve the same effect without the duplicate pointers?

Answer:

a. In effect, that process will have increased its priority since by getting time more often it is receiving preferential treatment.

b. The advantage is that more important jobs could be given more time, in other words, higher priority in treatment. The consequence, of course, is that shorter jobs will suffer.

c. Allot a longer amount of time to processes deserving higher priority. In other words, have two or more quantums possible in the Round-Robin scheme.

Q. Writ down the main difference between preemitive and non-premitive scheduling schemes and write down at least one from each.

Sol:

Tasks are usually assigned with priorities. At times it is necessary to run a certain task that has a higher priority before another task although it is running. Therefore, the running task is interrupted for some time and resumed later when the priority task has finished its execution. This is called preemptive scheduling. Eg: Round robin

In non-preemptive scheduling, a running task is executed till completion. It cannot be interrupted. Eg First In First Out.

Q. The ctime_ts function is thread safe but not reentrant. Explain your answer with valid reasons?

```
char *ctime_ts(const time_t *timep, char *privatep)
{
char *sharedp;
P(&mutex);
sharedp = ctime(timep);
strcpy(privatep, sharedp); /* Copy string from shared to private */
V(&mutex);
return privatep;
}
```

Answer:

This wraps the call to ctime in a lock that means only one thread can access a call to ctime.

The locked critical section will copy the string pointed to by the static pointer of ctime and copy it into a local non-shared pointer.

Thus, one thread's call to ctime_ts cannot be affected by another thread's call to ctime_ts.

“The ctime_ts function is not reentrant because each invocation shares the same static variable returned by the ctime function.

Q. Describe a mechanism by which one segment could belong to the address space of two different processes.

Answer:

Since segment tables are a collection of base–limit registers, segments can be shared when entries in the segment table of two different jobs point to the same physical location. The two segment tables must have identical base and limit values.

Q. Describe the working of explicit and implicit memory allocators?

Explicit: application allocates and frees space e.g., malloc and free in C Implicit: application allocates, but does not free space e.g. garbage collection in Java, ML or Lisp.

Q. Describe the differences among short-term, medium-term, and long-term scheduling.

 Short-term (CPU scheduler): selects a process from those that are in memory and ready to execute, and and allocates the CPU to it.

 Medium-term (memory manager): selects processes from the ready or blocked queue and removes them from memory, then reinstates them later to continue running.

 Long-term (job scheduler): determines which jobs are brought into the system for processing.

Q. Process Burst Time Priority
*P*1 10 3
*P*2 1 1
*P*3 2 3
*P*4 1 4
*P*5 5 2
The processes are assumed to have arrived in the order *P*1, *P*2, *P*3, *P*4, *P*5,
all at time 0.
a. What is the turnaround time of each process for each of the scheduling algorithms in part a?
b. What is the waiting time of each process for each of the scheduling algorithms in part a?
c. Which of the schedules in part a results in the minimal average waiting time (over all processes)?
a. Turnaround time
FCFS RR SJF Priority
*P*1 10 19 19 16
*P*2 11 2 1 1
*P*3 13 7 4 18
*P*4 14 4 2 19
*P*5 19 14 9 6
b. Waiting time (turnaround time minus burst time)
FCFS RR SJF Priority
*P*1 0 9 9 6

*P*2 10 1 0 0
*P*3 11 5 2 16
*P*4 13 3 1 18
*P*5 14 9 4 1
c. Shortest Job First (3.2 ms).

Q. How is it possible to prevent the deadlocks?

Answer:

There are four necessary conditions which must hold simultaneously for a deadlock to occur. So, deadlock can be prevented by implementing some methodology that guarantees that at least one of the four necessary conditions must not hold. By implementing this kind of strategy, it is possible to prevent a deadlock from occurring. Relaxing first three conditions i.e. Mutual exclusion, Hold & wait, and no pre-emption are not always useful. There may be low utilization of resources & starvation. Circular Wait is the ultimate choice. This is the most suitable condition i.e. ensuring that circular wait never holds. Simple methodology used for this purpose is order all the resources and each process always requests resources in increasing order. Hence, there may never be a process waiting for a low numbered resource holding a higher numbered resource, ensuring that occurrence of cycle is impossible. So, implementing that there will be no circular wait, deadlocks can be prevented.

Chapter 3: MEMORY MANAGEMENT

Question 1: A swapping system eliminates holes by compaction. Assuming a random distribution of many holes and many data segments and a time to read or write a 32-bit memory word of 10 nsec, about how long does it take to compact 128MB? For simplicity, assume that 0 word is part of hole and that the highest word in memory contains valid data.

Answer: Almost the entire memory has to be copied, which requires each word to be read and then rewritten at a different location. Reading 4 bytes takes 10 nsec, so reading 1 byte takes 2.5 nsec and writing it takes another 2.5 nsec, for a total of 5 nsec per byte compacted. This is a rate of 200,000,000 bytes/sec. To copy 128 MB (227 bytes, which is about 1.34 × 108 bytes), the computer needs 227 /200,000,000 sec, which is about 671 msec. This number is slightly pessimistic because if the initial hole at the bottom of memory is k bytes, those k bytes do not need to be copied. However, if there are many holes and many data segments, the holes will be small so k will be small and the error in the calculation will also be small.

Question 2: Consider a swapping system in which memory consists of the following hole sizes in memory order: 10 KB, 4 KB, 20 KB, 18 KB, 7 KB, 9 KB, 12 KB, and 15 KB. Which hole is taken for successive segment requests of (a) 12 KB (b) 10 KB (c) 9 KB.

For first fit? Now repeat the question for best fit, worst fit, and next fit.

Answer: First fit takes 20 KB, 10 KB, 18 KB. Best fit takes 12 KB, 10 KB, and 9 KB. Worst fit takes 20 KB, 18 KB, and 15 KB. Next fit takes 20 KB, 18 KB, and 9 KB.

Question 3: What is the difference between a physical address and a virtual address?

Answer: Real memory uses physical addresses. These are the numbers that the memory chips react to on the bus. Virtual addresses are the logical addresses that refer to a process' address space. Thus a machine with a 16-bit word can generate virtual addresses up to 64K, regardless of whether the machine has more or less memory than 64 KB.

Question 4: The amount of disk space that must be available for page storage is related to the maximum number of processes, n, the number of bytes in the virtual address space, v, and the number of bytes of RAM, r. Give an expression for the worst-case disk space requirements. How realistic is this amount?

Answer: The total virtual address space for all the processes combined is nv so this much storage is needed for pages. However an amount r can be in RAM, so the amount of disk storage required is only nv − r. This amount is far more than is ever needed in practice because rarely will there be n processes actually running and even more rarely will all of them need the maximum allowed virtual memory.

Question 5: If an instruction takes 10 nsec and a page fault takes an additional n nsec, give a formula for the effective instruction time if page faults occur every k instructions.

Answer: A page fault every k instructions adds an extra overhead of n /k μsec to the average, so the average instruction takes 10 + n /k nsec.

Question 6: Using the page table of Fig. 3-9, give the physical address corresponding to each of the following virtual addresses: (a) 20 (b) 4100 (c) 8300

Figure 3-9. The relation between virtual addresses and physical memory addresses is given by the page table. Every page begins on a multiple of 4096 and ends 4095 addresses higher, so 4K-8K really means 4096-8191 and 8K to 12K means 8192-12287.

Answer: (a) 8212 (b) 4100 (c) 24684

Question 7: A machine has a 32-bit address space and an 8-KB page. The page table is entirely in hardware, with one 32-bit word per entry. When a process starts, the page table is copied to the hardware from memory, at one word every 100 nsec. If each process runs for 100 msec (including the time to load the page table), what fraction of the CPU time is devoted to loading the page tables?

Answer: The page table contains $2_{32} / 21_3$ entries, which is 524,288. Loading the page table takes 52 msec. If a process gets 100 msec, this consists of 52 msec for loading the page table and 48 msec for running. Thus 52 percent of the time is spent loading page tables.

Question 8: Suppose that a 32-bit virtual address is broken up into four fields, a, b, c, and d. The first three are used for a three-level page table system. The fourth field, d, is the offset. Does the number of pages depend on the sizes of all four fields? If not, which ones matter and which ones do not?

Answer: The number of pages depends on the total number of bits in a, b, and c combined. How they are split among the fields does not matter.

Question 9: A computer has 32-bit virtual addresses and 4-KB pages. The program and data together fit in the lowest page (0-4095). The stack fits in the highest page. How many en tries are needed in the page table if traditional (one-level) paging is used? How many page table entries are needed for two-level paging, with 10 bits in each part?

Answer: For a one-level page table, there are $2_{32} / 2_{12}$ or 1M pages needed. Thus the page table must have 1M entries. For two-level paging, the main page table has 1K entries, each of which points to a second page table. Only two of these are used. Thus in total only three page table entries are needed, one in the top-level table and one in each of the lower-level tables.

Question 10: The TLB on the VAX does not contain an R bit. Why?

Answer: The R bit is never needed in the TLB. The mere presence of a page there means the page has been referenced; otherwise it would not be there. Thus the bit is completely redundant. When the entry is written back to memory, however, the R bit in the memory page table is set.

Question 11: A computer with an 8-KB page, a 256-KB main memory, and a 64-GB virtual address space uses an inverted page table to implement its virtual memory. How big should the hash table be to ensure a mean hash chain length of less than 1? Assume that the hash table size is a power of two.
Answer: The main memory has $2_{28} / 2_{13} = 32{,}768$ pages. A 32K hash table will have a mean chain length of 1. To get under 1, we have to go to the next size, 65,536 entries. Spreading 32,768 entries over a 65,536 table slots will give a mean chain length of 0.5, which ensures fast lookup.
Question 12: A student in a compiler design course proposes to the professor a project of writing a compiler that will produce a list of page references that can be used to implement the optimal page replacement algorithm. Is this possible? Why or why not? Is there anything that could be done to improve paging efficiency at run time?
Answer: This is probably not possible except for the unusual and not very useful case of a program whose course of execution is completely predictable at compilation time. If a compiler collects information about the locations in the code of calls to procedures, this information might be used at link time to rearrange the object code so procedures were located close to the code that calls them. This would make it more likely that a procedure would be on the same page as the calling code. Of course this would not help much for procedures called from many places in the program.
Question 13: If FIFO page replacement is used with four page frames and eight pages, how many page faults will occur with the reference string 0172327103 if the four frames are initially empty? Now repeat this problem for LRU.
Answer:
The page frames for FIFO are as follows:
x0172333300 xx017222233 xxx01777722 xxxx0111177
The page frames for LRU are as follows:
x0172327103 xx017232710 xxx01773271 xxxx0111327
Question 14: A small computer has four page frames. At the first clock tick, the **R** bits are 0111 (page 0 is 0, the rest are 1). At subsequent clock ticks, the values are 1011, 1010, 1101, 0010, 1010, 1100, and 0001. If the aging algorithm is used with an 8-bit counter, give the values of the four counters after the last tick.
Answer: The counters are: Page 0: 0110110, Page 1: 01001001, Page 2: 00110111, Page 3: 10001011
Question 15: How long does it take to load a 64-KB program from a disk whose average seek time is 10 msec, whose rotation time is 10 msec, and whose tracks hold 32 KB
(a) for a 2-KB page size?
(b) for a 4-KB page size?
The pages are spread randomly around the disk and the number of cylinders is so large that the chance of two pages being on the same cylinder is negligible.
Answer: The seek plus rotational latency is 20 msec. For 2-KB pages, the transfer time is 1.25 msec, for a total of 21.25 msec. Loading 32 of these pages will take 680 msec. For 4-KB pages, the transfer time is doubled to 2.5 msec, so the total time per page is 22.50 msec. Loading 16 of these pages takes 360 msec.

Question 16: Suppose that τ = 400 in Fig. 3-20. Which page will be removed?
Answer: The age of the page is 2204 − 1213 = 991. If τ = 400, it is definitely out of the working set and it was not recently referenced so it will be evicted. The τ = 1000 the situation is different. Now the page falls within the working set (barely), so it is not removed.
Question 17: A computer has four page frames. The time of loading, time of last access, and the R and M bits for each page are as shown below (the times are in clock ticks)
a) Which page will NRU replace?
b) Which page will FIFO replace?
c) Which page will LRU replace?
d) Which page will second chance replace?

Answer:
a) NRU removes page
b) FIFO removes page
c) LRU removes page
d) Second chance removes page 2.

Question 18: One of the first timesharing machines, the PDP-1, had a memory of 4K 18-bit words. It held one process at a time in memory. When the scheduler decided to run another process, the process in memory was written to a paging drum, with 4K 18-bit words around the circumference of the drum. The drum could start writing (or reading) at any word, rather than only at word 0. Why do you suppose this drum was chosen?
Answer: The PDP-1 paging drum had the advantage of no rotational latency. This saved half a rotation each time memory was written to the drum.

Question 19: A computer provides each process with 65,536 bytes of address space divided into pages of 4096 bytes. A particular program has a text size of 32,768 bytes, a data size of 16,386 bytes, and a stack size of 15,870 bytes. Will this program fit in the address space? If the page size were 512 bytes, would it fit? Remember that a page may not contain parts of two different segments.
Answer: The text is eight pages, the data are five pages, and the stack is four pages. The program does not fit because it needs 17 4096-byte pages. With a 512-byte page, the situation is different. Here the text is 64 pages, the data are 33 pages, and the stack is 31 pages, for a total of 128 512-byte pages, which fits. With the small page size it is ok, but not with the large one.
Question 20: If a page is shared between two processes, is it possible that the page is read-only for one process and read-write for the other? Why or why not?
Answer: If pages can be shared, yes. For example, if two users of a timesharing system are running the same editor at the same time and the program text is shared rather than copied, some of those pages may be in each user's working set at the same time.
Question 21: It has been observed that the number of instructions executed between page faults is directly proportional to the number of page frames allocated to a program. If the available memory is doubled, the mean interval between page faults is also doubled. Suppose that a normal instruction takes 1 microsec, but if a page fault occurs, it takes 2001 usee (i.e., 2 msec to handle the fault). If a program takes 60 sec to run, during which time it gets 15,000 page faults, how long would it take to run if twice as much memory were available?
Answer: The program is getting 15,000 page faults, each of which uses 2 msec of extra processing time. Together, the page fault overhead is 30 sec. This means that of the 60 sec used, half was spent on page fault overhead, and half on running the program. If we run the program with twice as much memory, we get half as memory page faults, and only 15 sec of page fault overhead, so the total run time will be 45 sec.
Question 22: A group of operating system designers for the Frugal Computer Company are thinking about ways to reduce the amount of backing store needed in their new operating system. The head guru has just suggested not bothering to save the program text in the swap area at all, but just page it in directly from the binary file whenever it is needed. Under what conditions, if any, does this idea work for the program text? Under what conditions, if any, does it work for the data?
Answer: It works for the program if the program cannot be modified. It works for the data if the data cannot be modified. However, it is common that the program cannot be modified and extremely rare that the data cannot be modified. If the data area on the binary file were overwritten with updated pages, the next time the program was started, it would not have the original data.
Question 23: A machine language instruction to load a 32-bit word into a register contains the 32- bit address of the word to be loaded. What is the maximum number of page faults this instruction can cause?
Answer: The instruction could lie astride a page boundary, causing two page faults just to fetch the instruction. The word fetched could also span a page boundary, generating two more faults, for a total of four. If words must be aligned in memory, the data word can cause only one fault, but an instruction to load a 32-bit word at address 4094 on a machine with a 4-KB page is legal on some machines (including the Pentium).

Question 24: Explain the difference between internal fragmentation and external fragmentation. Which one occurs in paging systems? Which one occurs in systems using pure segmentation?
Answer: Internal fragmentation occurs when the last allocation unit is not full. External fragmentation occurs when space is wasted between two allocation units. In a paging system, the wasted space in the last page is lost to internal fragmentation. In a pure segmentation system, some space is invariably lost between the segments. This is due to external fragmentation.
Question 25: When segmentation and paging are both being used, as in MULTICS, first the segment descriptor must be looked up, then the page descriptor. Does the TLB also work this way, with two levels of lookup?
Answer: No. The search key uses both the segment number and the virtual page number, so the exact page can be found in a single match.

Chapter 4: FILE SYSTEMS

Question 1: Give five different path names for the file /etc/posswd. Hint. Think about the directory entries "." and

Answer: You can go up and down the tree as often as you want using ‘‘..’’. Some of the many paths are
/etc/passwd
/./etc/passwd
/./././etc/passwd
/./././etc/passwd
/etc/../etc/passwd
/etc/../etc/../etc/passwd
/etc/../etc/../etc/../etc/passwd
/etc/../etc/../etc/../etc/../etc/passwd

Question 2: In Windows, when a user double clicks on a file listed by Windows Explorer, a program is run and given that file as a parameter. List two different ways the operating system could know which program to run.

Answer: The Windows way is to use the file extension. Each extension corresponds to a file type and to some program that handles that type. Another way is to remember which program created the file and run that program. The Macintosh works this way.

Question 3: In early UNIX systems, executable files (a.out files) began with a very specific magic number, not one chosen at random. These files began with a header, followed by the text and data segments. Why do you think a very specific number was chosen for executable files, whereas other file types had a more-or-less random magic number as the first word?

Answer: These systems loaded the program directly in memory and began executing at word 0, which was the magic number. To avoid trying to execute the header as code, the magic number was a BRANCH instruction with a target address just above the header. In this way it was possible to read the binary file directly into the new process’ address space and run it at 0, without even knowing how big the header was.

Question 4: In Fig. 4-4, one of the attributes is the record length. Why does the operating system ever care about this?

Answer: The operating system cares about record length when files can be structured as records with keys at a specific position within each record and it is possible to ask for a record with a given key. In that case, the system has to know how big the records are so it can search each one for the key.

Question 5: Systems that support sequential files always have an operation to rewind files. Do systems that support random access files need this too?

Answer: No. If you want to read the file again, just randomly access byte 0.

Question 6: In some systems it is possible to map part of a file into memory. What restrictions must such systems impose? How is this partial mapping implemented?

Answer: The mapped portion of the file must start at a page boundary and be an integral number of pages in length. Each mapped page uses the file itself as backing store. Unmapped memory uses a scratch file or partition as backing store.

Question 7: A simple operating system only supports a single directory but allows that directory to have arbitrarily many files with arbitrarily long file names. Can something approximating a hierarchical file system be simulated? How?

Answer: Use file names such as /usr/ast/file. While it looks like a hierarchical path name, it is really just a single name containing embedded slashes.

Question 8: In UNIX and Windows, random access is done by having a special system call that moves the "current position" pointer associated with a file to a given byte in the file. Propose an alternative way to do random access without having this system call.

Answer:

One way is to add an extra parameter to the read system call that tells what address to read from. In effect, every read then has a potential for doing a seek within the file. The disadvantages of this scheme are (1) an extra parameter in every read call, and (2) requiring the user to keep track of where the file pointer is.

Question 9: Consider the directory tree of Fig. 4-8. If /usr/jim is the working directory, what is the absolute path name for the file whose relative path name is Jast/xl

Answer: The dotdot component moves the search to /usr, so ../ast puts it in /usr/ast. Thus ../ast/x is the same as /usr/ast/x.

Question 10: Contiguous allocation of files leads to disk fragmentation, as mentioned in the text, because some space in the last disk block will be wasted in files whose length is not an integral number of blocks. Is this internal fragmentation or external fragmentation? Make an analogy with something discussed in the pervious chapter.

Answer: Since the wasted storage is between the allocation units (files), not inside them, this is external fragmentation. It is precisely analogous to the external fragmentation of main memory that occurs with a swapping system or a system using pure segmentation.

Question 11: In light of the answer to the pervious question, does compacting the disk ever make any sense?
Answer: If done right, yes. While compacting, each file should be organized so that all of its blocks are consecutive, for fast access. Windows has a program that defragments and reorganizes the disk. Users are encouraged to run it periodically to improve system performance. But given how long it takes, running once a month might be a good frequency.
Question 12: Some digital consumer devices need to store data, for example as files. Name a modern device that requires file storage and for which contiguous allocation would be a fine idea.
Answer: A digital still camera records some number of photographs in sequence on a nonvolatile storage medium (e.g., flash memory). When the camera is reset, the medium is emptied. Thereafter, pictures are recorded one at a time in sequence until the medium is full, at which time they are uploaded to a hard disk. For this application, a contiguous file system inside the camera (e.g., on the picture storage medium) is ideal.
Question 13: How does MS-DOS implement random access to files?
Answer: It finds the address of the first block in the directory entry. It then follows the chain of block pointers in the FAT until it has located the block it needs. It then remembers this block number for the next read system call.
Question 14: Consider the i-node shown in Fig. 4-13. If it contains 10 direct addresses of 4 bytes each and all disk blocks are 1024 KB, what is the largest possible file?
Answer: The indirect block can hold 256 disk addresses. Together with the 10 direct disk addresses, the maximum file has 266 blocks. Since each block is 1 KB, the largest file is 266 KB.
Question 15: It has been suggested that efficiency could be improved and disk space saved by storing the data of a short file within the i-node. For the i-node of Fig. 4-13, how many bytes of data could be stored inside the i-node?
Answer: There must be a way to signal that the address block pointers hold data, rather than pointers. If there is a bit left over somewhere among the attributes, it can be used. This leaves all nine pointers for data. If the pointers are k bytes each, the stored file could be up to 9k bytes long. If no bit is left over among the attributes, the first disk address can hold an invalid address to mark the following bytes as data rather than pointers. In that case, the maximum file is 8k bytes.

Question 16: Name one advantage of hard links over symbolic links and one advantage of symbolic links over hard links.
Answer: Hard links do not require any extra disk space, just a counter in the i-node to keep track of how many there are. Symbolic links need space to store the name of the file pointed to. Symbolic links can point to files on other machines, even over the Internet. Hard links are restricted to pointing to files within their own partition.
Question 17: Free disk space can be kept track of using a free list or a bitmap. Disk addresses require D bits. For a disk with B blocks, F of which are free, state the condition under which the free list uses less space than the bitmap. For D having the value 16 bits, express your answer as a percentage of the disk space that must be free.
Answer: The bitmap requires B bits. The free list requires DF bits. The free list requires fewer bits if DF < B. Alternatively, the free list is shorter if F/B < 1/D, where F/B is the fraction of blocks free. For 16-bit disk addresses, the free list is shorter if 6 percent or less of the disk is free.
Question I8: What would happen if the bitmap or free list containing the information about free disk blocks was completely lost due to a crash? Is there any way to recover from this disaster, or is it bye-bye disk? Discuss your answers for UNIX and the FAT-16 file system separately.
Answer: It is not a serious problem at all. Repair is straightforward; it just takes time. The recovery algorithm is to make a list of all the blocks in all the files and take the complement as the new free list. In UNIX this can be done by scanning all the i-nodes. In the FAT file system, the problem cannot occur because there is no free list. But even if there were, all that would have to be done to recover it is to scan the FAT looking for free entries.
Question 19: Oliver Owl's night job at the university computing center is to change the tapes used for overnight data backups. While waiting for each tape to complete, he works on writing his thesis that proves Shakespeare's plays were written by extraterrestrial visitors. His text processor runs on the system being backed up since that is the only one they have. Is there a problem with this arrangement?
Answer: Ollie’s thesis may not be backed up as reliably as he might wish. A backup program may pass over a file that is currently open for writing, as the state of the data in such a file may be indeterminate.
Question 20: We discussed making incremental dumps in some detail in the text. In Windows it is easy to tell when to dump a file because every file has an archive bit. This bit is missing in UNIX. How do UNIX backup programs know which files to dump?
Answer: They must keep track of the time of the last dump in a file on disk. At every dump, an entry is appended to this file. At dump time, the file is read and the time of the last entry noted. Any file changed since that time is dumped.
Question 21: It has been suggested that the first part of each UNIX file be kept in the same disk block as its i-node. What good would this do?
Answer: Many UNIX files are short. If the entire file fits in the same block as the inode, only one disk access would be needed to read the file, instead of two, as is presently the case. Even for longer files there would be a gain, since one fewer disk accesses would be needed.
Question 22: Suppose that file 21 in Fig. 4-25 was not modified since the last dump. In what way would the four bitmaps of Fig. 4-26 be different?

Answer: In (a) and (b), 21 would not be marked. In (c), there would be no change. In (d), 21 would not be marked.

Question 23: Consider Fig. 4-27. Is it possible that for some particular block number the counters in both lists have the value 2? How should this problem be corrected?

Answer: It should not happen, but due to a bug somewhere it could happen. It means that some block occurs in two files and also twice in the free list. The first step in repairing the error is to remove both copies from the free list. Next a free block has to be acquired and the contents of the sick block copied there. Finally, the occurrence of the block in one of the files should be changed to refer to the newly acquired copy of the block. At this point the system is once again consistent.

Question 24: The performance of a file system depends upon the cache hit rate (fraction of blocks found in the cache). If it takes 1 msec to satisfy a request from the cache, but 40 msec to satisfy a request if a disk read is needed, give a formula for the mean time required to satisfy a request if the hit rate is h. Plot this function for values of h varying from 0 to 1.0.

Answer: The time needed is $h + 40 \times (1 - h)$. The plot is just a straight line.

Question 25: Consider the idea behind Fig. 4-21, but now for a disk with a mean seek time of 8 msec, a rotational rafe "of-15,000 rpm, and 262,144 bytes per track. What are the data rates for block sizes of 1 KB, 2 KB, and 4 KB, respectively?

Answer: At 15,000 rpm, the disk takes 4 msec to go around once. The average access time (in msec) to read k bytes is then 8 + 2 + (k /262144) × 4. For blocks of 1 KB, 2 KB, and 4 KB, the access times are 10.015625 msec, 10.03125 msec, and 10.0625 msec, respectively (hardly any different). These give rates of about 102,240 KB/sec, 204,162 KB/sec, and 407,056 KB/sec, respectively.

Question 26: A certain file system uses 2-KB disk blocks. The median file size is 1 KB. If all files were exactly 1 KB, what fraction of the disk space would be wasted? Do you think the wastage for a real file system will be higher than this number or lower than it? Explain your answer.

Answer: If all files were 1 KB, then each 2-KB block would contain one file and 1 KB of wasted space. Trying to put two files in a block is not allowed because the unit used to keep track of data is the block, not the semiblock. This leads to 50 percent wasted space. In practice, every file system has large files as well as many small ones, and these files use the disk much more efficiently. For example, a 32,769-byte file would use 17 disk blocks for storage, given a space efficiency of 32768/34816, which is about 94 percent.

Question 27: The MS-DOS FAT-16 table contains 64K entries. Suppose that one of the bits had been needed for some other purpose and that the table contained exactly 32,768 entries instead. With no other changes, what would the largest MS-DOS file have been under this condition?

Answer: The largest block is 32,768. With 32,768 of these blocks, the biggest file would be 1 GB.

Question 28: Files in MS-DOS have to compete for space in the FAT-16 table in memory. If one file uses k entries, that is k entries that are not available to any other file, what constraint does this place on the total length of all files combined?

Answer: It constrains the sum of all the file lengths to being no larger than the disk. This is not a very serious constraint. If the files were collectively larger than the disk, there would be no place to store all of them on the disk.

Question 29: A UNIX file system has I-KB blocks and 4-byte disk addresses. What is the maximum file size if i-nodes contain 10 direct entries, and one single, double, and triple indirect entry each?

Answer: The i-node holds 10 pointers. The single indirect block holds 256 pointers. The double indirect block is good for 2562 pointers. The triple indirect block is good for 2563 pointers. Adding these up, we get a maximum file size of 16,843,018 blocks, which is about 16.06 GB.

Question 30: How many disk operations are needed to fetch the i-node for the file /usr/ast/courses/os/handout.t? Assume that the i-node for the root directory is in memory, but nothing else along the path is in memory. Also assume that all directories fit in one disk block.
Answer: The following disk reads are needed:
directory for /
i-node for /usr
directory for /usr
i-node for /usr/ast
directory for /usr/ast
i-node for /usr/ast/courses
directory for /usr/ast/courses
i-node for /usr/ast/courses/os
directory for /usr/ast/courses/os
i-node for /usr/ast/courses/os/handout.t
In total, 10 disk reads are required.
Question 31: In many UNIX systems, the i-nodes are kept at the start of the disk. An alternative design is to allocate an i-node when a file is created and put the i-node at the start of the first block of the file. Discuss the pros and cons of this alternative.
Answer: Some pros are as follows. First, no disk space is wasted on unused i-nodes. Second, it is not possible to run out of i-nodes. Third, less disk movement is needed since the i-node and the initial data can be read in one operation. Some cons are as follows. First, directory entries will now need a 32-bit disk address instead of a 16-bit i-node number. Second, an entire disk will be used even for files which contain no data (empty files, device files). Third, file system integrity checks will be slower because of the need to read an entire block for each i-node and because i-nodes will be scattered all over the disk. Fourth, files whose size has been carefully designed to fit the block size will no longer fit the block size due to the i-node, messing up performance.

Chapter 5: INPUT/OUTPUT

Question 1: Given the speeds listed in Fig. 5-1, is it possible to scan documents from a scanner and transmit them over an 802.1 Ig network at full speed? Defend your answer.

Answer: Easy. The scanner puts out 400 KB/sec maximum. The bus and disk both run at 16.7 MB/sec, so neither the disk nor the bus comes anywhere near saturation.

Question 2: Figure 5.3 shows one way of having memory-mapped I/O even in the presence of separate buses for memory and I/O devices, namely, to first try the memory bus and if that fails try the I/O bus. A clever computer science student has thought of an improvement on this idea: try both in parallel, to speed up the process of accessing I/O devices. What do you think of this idea?

Answer: It is not a good idea. The memory bus is surely faster than the I/O bus, otherwise why bother with it? Consider what happens with a normal memory request. The memory bus finishes first, but the I/O bus is still busy. If the CPU waits until the I/O bus finishes, it has reduced memory performance to that of the I/O bus. If it just tries the memory bus for the second reference, it will fail if this one is an I/O device reference. If there were some way to instantaneously abort the previous I/O bus reference to try the second one, the improvement might work, but there is never such an option. All in all, it is a bad idea.

Question 3: A DMA controller has four channels. The controller is capable of requesting a 32-bit word every 100 nsec. A response takes equally long. How fast does the bus have to be to avoid being a bottleneck?

Fig: 5.1

Fig: 5.3

Answer: Each bus transaction has a request and a response, each taking 100 nsec, or 200 nsec per bus transaction. This gives 5 million bus transactions/sec. If each one is good for 4 bytes, the bus has to handle 20 MB/sec. The fact that these transactions may be sprayed over four I/O devices in round-robin fashion is irrelevant. A bus transaction takes 200 nsec, regardless of whether consecutive requests are to the same device or different devices, so the number of channels the DMA controller has does not matter. The bus does not know or care.

Question 4: Suppose that a computer can read or write a memory word in 10 nsec. Also suppose that when an interrupt occurs, all 32 CPU registers, plus the program counter and PSW are pushed onto the stack. What is the maximum number of interrupts per second this machine can process?

Answer: An interrupt requires pushing 34 words onto the stack. Returning from the interrupt requires fetching 34 words from the stack. This overhead alone is 680 nsec. Thus the maximum number of interrupts per second is no more than about 1.47 million, assuming no work for each interrupt.
Question 5: In Fig. 5-9(b), the interrupt is not acknowledged until after the next character has been output to the printer. Could it have equally well been acknowledged right at the start of the interrupt service procedure? If so, give one reason for doing it at the end, as in the text. If not, why not?
Answer: It could have been done at the start. A reason for doing it at the end is that the code of the interrupt service procedure is very short. By first outputting another character and then acknowledging the interrupt, if another interrupt happens immediately, the printer will be working during the interrupt, making it print slightly faster. A disadvantage of this approach is slightly longer dead time when other interrupts may be disabled.
Question 6: A computer has a three-stage pipeline as shown in Fig. l-6(a). On each clock cycle, one new instruction is fetched from memory at the address pointed to by the PC and put into the pipeline and the PC advanced. Each instruction occupies exactly one memory word. The instructions already in the pipeline are each advanced one stage. When an interrupt occurs, the current PC is pushed onto the stack, and the PC is set to the address of the interrupt handler. Then the pipeline is shifted right one stage and the first instruction of the interrupt handler is fetched into the pipeline. Does this machine have precise interrupts? Defend your answer.
Answer: Yes. The stacked PC points to the first instruction not fetched. All instructions before that have been executed and the instruction pointed to and its successors have not been executed. This is the condition for precise interrupts. Precise interrupts are not hard to achieve on machine with a single pipeline. The trouble comes in when instructions are executed out of order, which is not the case here.
Question 7: A typical printed page of text contains 50 lines of 80 characters each. Imagine that a certain printer can print 6 pages per minute and that the time to write a character to the printer's output register is so short it can be ignored. Does it make sense to run this printer using interrupt-driven I/O if each character printed requires an interrupt that takes 50 usee all-in to service?
Answer: The printer prints 50 × 80 × 6 = 24,000 characters/min, which is 400 characters/sec. Each character uses 50 μsec of CPU time for the interrupt, so collectively in each second the interrupt

overhead is 20 msec. Using interrupt-driven I/O, the remaining 980 msec of time is available for other work. In other words, the interrupt overhead costs only 2% of the CPU, which will hardly affect the running program at all.

Question 8: What is "device independence"?

Answer: Device independence means that files and devices are accessed the same way, independent of their physical nature. Systems that have one set of calls for writing on a file, but a different set of calls for writing on the console (terminal) do not exhibit device independence.

Question 9: In which of the four I/O software layers is each of the following done.

a) Computing the track, sector, and head for a disk read.

b) Writing commands to the device registers.

c) Checking to see if the user is permitted to use the device.

d) Converting binary integers to ASCII for printing.

Answer:

a) Device driver.

b) Device driver.

c) Device-independent software.

d) User-level software.

Question 10: A local area network is used as follows. The user issues a system call to write data packets to the network. The operating system then copies the data to a kernel buffer. Then it copies the data to the network controller board. When all the bytes are safely inside the controller, they are sent over the network at a rate of 10 megabits/sec. The receiving network controller stores each bit a microsecond after it is sent. When the last bit arrives, the destination CPU is interrupted, and the kernel copies the newly arrived packet to a kernel buffer to inspect it. Once it has figured out which user the packet is for, the kernel copies the data to the user space. If we assume that each interrupt and its associated processing takes 1 msec, that packets are 1024 bytes (ignore the headers), and that copying a byte takes 1 usee, what is the maximum rate at which one process can pump data to another? Assume that the sender is blocked until the work is finished at the receiving side and an acknowledgement comes back. For simplicity, assume that the time to get the acknowledgement back is so small it can be ignored.

Answer: A packet must be copied four times during this process, which takes 4.1 msec. There are also two interrupts, which account for 2 msec. Finally, the transmission time is 0.83 msec, for a total of 6.93 msec per 1024 bytes. The maximum data rate is thus 147,763 bytes/sec, or about 12 percent of the nominal 10 megabit/sec network capacity. (If we include protocol overhead, the figures get even worse.)

Question 11: Why are output files for the printer normally spooled on disk before being printed?

Answer: If the printer were assigned as soon as the output appeared, a process could tie up the printer by printing a few characters and then going to sleep for a week.

Question 12: How much cylinder skew is needed for a 7200-RPM disk with a track-to-track seek time of 1 msec? The disk has 200 sectors of 512 bytes each on each track.

Answer: The disk rotates at 120 rpm, so 1 rotation takes 1000/120 msec. With 200 sectors per rotation, the sector time is 1/200 of this number or 5/120 = 1/24 msec. During the 1-msec seek, 24 sectors pass under the head. Thus the cylinder skew should be 24.

Question 13: Calculate the maximum data rate in MB/sec for the disk described in the previous problem.

Answer: As we saw in the previous problem, the sector time is 1/24 msec. This means that the disk can read 24,000 sectors/sec. Since each sector contains 512 bytes, the data rate is 12,288,000 bytes/sec. This rate is 11.7 MB/sec.

Question 14: RAID level 3 is able to correct single-bit errors using only one parity drive. What is the point of RAID level 2? After all, it also can only correct one error and takes more drives to do so.
Answer: A magnetic field is generated between two poles. Not only is it difficult to make the source of a magnetic field small, but also the field spreads rapidly, which leads to mechanical problems trying to keep the surface of a magnetic medium close to a magnetic source or sensor. A semiconductor laser generates light in a very small place, and the light can be optically manipulated to illuminate a very small spot at a relatively great distance from the source.
Question 15: What are the advantages and disadvantages of optical disks versus magnetic disks? RAID level 2 can not only recover from crashed drives, but also from undetected transient errors. If one drive delivers a single bad bit, RAID level 2 will correct this, but RAID level 3 will not.
Answer: A magnetic field is generated between two poles. Not only is it difficult to make the source of a magnetic field small, but also the field spreads rapidly, which leads to mechanical problems trying to keep the surface of a magnetic medium close to a magnetic source or sensor. A semiconductor laser generates light in a very small place, and the light can be optically manipulated to illuminate a very small spot at a relatively great distance from the source.
Question 16: If a disk controller writes the bytes it receives from the disk to memory as fast as it receives them, with no internal buffering, is interleaving conceivably useful? Discuss.
Answer: Possibly. If most files are stored in logically consecutive sectors, it might be worthwhile interleaving the sectors to give programs time to process the data just received, so that when the next request is issued, the disk would be in the right place. Whether this is worth the trouble depends strongly on the kind of programs run and how uniform their behavior is.
Question 17: If a disk has double interleaving, does it also need cylinder skew in order to avoid missing data when making a track-to-track seek? Discuss your answer.
Answer: Maybe yes and maybe no. Double interleaving is effectively a cylinder skew of two sectors. If the head can make a track-to-track seek in fewer than two sector times, than no additional cylinder skew is needed. If it cannot, then additional cylinder skew is needed to avoid missing a sector after a seek.
Question 18: A disk manufacturer has two 5.25-inch disks that each have 10,000 cylinders. The newer one has double the linear recording density of the older one. Which disk properties are better on the newer drive and which are the same?
Answer: The drive capacity and transfer rates are doubled. The seek time and average rotational delay are the same.
Question 19: A computer manufacturer decides to redesign the partition table of a Pentium hard disk to provide more than four partitions. What are some consequences of this change?
Answer: One fairly obvious consequence is that no existing operating system will work because they all look there to see where the disk partitions are. Changing the format of the partition table will cause all the operating systems to fail. The only way to change the partition table is to simultaneously change all the operating systems to use the new format.
Question 20: Disk requests come in to the disk driver for cylinders 10, 22, 20, 2, 40, 6, and 38, in that order. A seek takes 6 msec per cylinder moved. How much seek time is needed for
a) First-come, first served.
b) Closest cylinder next.
c) Elevator algorithm (initially moving upward).

In all cases, the arm is initially at cylinder 20.
Answer:
a) 10 + 12 + 2 + 18 + 38 + 34 + 32 = 146 cylinders= 876 msec.
b) 0 + 2 + 12 + 4 + 4 + 36 +2 = 60 cylinders = 360 msec.
c) 0 + 2 + 16 + 2 + 30 + 4 + 4 = 58 cylinders = 348 msec.

Question 21: A personal computer salesman visiting a university in South-West Amsterdam remarked during his sales pitch that his company had devoted substantial effort to making their version of UNIX very fast. As an example, he noted that their disk driver used the elevator algorithm and also queued multiple requests within a cylinder in sector order. A student, Harry Hacker, was impressed and bought one. He took it home and wrote a program to randomly read 10,000 blocks spread across the disk. To his amazement, the performance that he measured was identical to what would be expected from first-come, first-served. Was the salesman lying?
Answer: Not necessarily. A UNIX program that reads 10,000 blocks issues the requests one at a time, blocking after each one is issued until after it is completed. Thus the disk driver sees only one request at a time; it has no opportunity to do anything but process them in the order of arrival. Harry should have started up many processes at the same time to see if the elevator algorithm worked.
Question 22: In the discussion of stable storage using nonvolatile RAM, the following point was glossed over. What happens if the stable write completes but a crash occurs before the operating system can write an invalid block number in the nonvolatile RAM? Does this race condition ruin the abstraction of stable storage? Explain your answer.
Answer: There is a race but it does not matter. Since the stable write itself has already completed, the fact that the nonvolatile RAM has not been updated just means that the recovery program will know which block was being written. It will read both copies. Finding them identical, it will change neither, which is the correct action. The effect of the crash just before the nonvolatile RAM was updated just means the recovery program will have to make two disk reads more than it should.
Question 23: The clock interrupt handler on a certain computer requires 2 msec (including process switching overhead) per clock tick. The clock runs at 60 Hz. What fraction of the CPU is devoted to the clock?
Answer: Two msec 60 times a second is 120 msec/sec, or 12 percent of the CPU
Question 24: Consider the performance of a 56-Kbps modem. The driver outputs one character and then blocks. When the character has been printed, an interrupt occurs and a message is sent to the blocked driver, which outputs the next character and then blocks again. If the time to pass a message, output a character, and block is 100 usee, what fraction of the CPU is eaten by the modem handling? Assume that each character has one start bit and one stop bit, for 10 bits in all.
Answer: At 56 Kbps, we have 5600 interrupts/sec, which is 560 msec. This is 56% of the CPU.
Question 25: A bitmap terminal contains 1280 by 960 pixels. To scroll a window, the CPU (or controller) must move all the lines of text upward by copying their bits from one part of the video RAM to another. If a particular window is 60 lines high by 80 characters wide (5280 characters, total), and a character's box is 8 pixels wide by 16 pixels high, how long does it take to scroll the whole window at a copying rate of 50 nsec per byte? If all lines are 80 characters long, what is the equivalent baud rate of the terminal? Putting a character on the screen takes 5 psec. How many lines per second can be displayed?
Answer: Scrolling the window requires copying 59 lines of 80 characters or 4720 characters. Copying 1 character (16 bytes) takes 800 nsec, so the whole window takes 3.776 msec. Writing 80 characters to the screen takes 400 nsec, so scrolling and displaying a new line take 4.176 msec. This gives about 239.5 lines/sec.

Question 26: After receiving a DEL (SIGINT) character, the display driver discards all output currently queued for that display. Why?
Answer: Suppose that the user inadvertently asked the editor to print thousands of lines. Then he hits DEL to stop it. If the driver did not discard output, output might continue for several seconds after the DEL, which would make the user hit DEL again and again and get frustrated when nothing happened.
Question 27: A user at a terminal issues a command to an editor to delete the word on line 5 occupying character positions 7 through and including 12. Assuming the cursor is not on line 5 when the command is given, what ANSI escape sequence should the editor emit to delete the word?
Answer: It should move the cursor to line 5 position 7 and then delete 6 characters. The sequence is ESC [5 ; 7 H ESC [6 P
Question 28: On the original IBM PC's color display, writing to the video RAM at any time other than during the CRT beam's vertical retrace caused ugly spots to appear all over the screen. A screen image is 25 by 80 characters, each of which fits in a box 8 pixels by 8 pixels. Each row of 640 pixels is drawn on a single horizontal scan of the beam, which takes 63.6 usee, including the horizontal retrace. The screen is redrawn 60 times a second, each of which requires a vertical retrace period to get the beam back to the top. What fraction of the time is the video RAM available for writing in?
Answer: The 25 lines of characters, each 8 pixels high, requires 200 scans to draw. There are 60 screens a second, or 12,000 scans/sec. At 63.6 μsec/scan, the beam is moving horizontally 763 msec per second, leaving 237 msec for writing in the video RAM. Thus the video RAM is available 23.7% of the time.
Question 29: The designers of a computer system expected that the mouse could be moved at a maximum rate of 20 cm/sec. If a mickey is 0.1 mm and each mouse message is 3 bytes, what is the maximum data rate of the mouse assuming that each mickey is reported separately?
Answer: The maximum rate the mouse can move is 200 mm/sec, which is 2000 mickeys/sec. If each report is 3 byte, the output rate is 6000 bytes/sec.
Question 30: One way to place a character on a bitmapped screen is to use bitblt from a font table. Assume that a particular font uses characters that are 16 **x** 24 pixels in true RGB color.
a) How much font table space does each character take?
b) If copying a byte takes 100 nsec, including overhead, what is the output rate to the screen in characters/sec?

Answer:
a) Each pixel takes 3 bytes in RGB, so the table space is 16 × 24 × 3 bytes, which is 1152 bytes.

b) At 100 nsec per byte, each character takes 115.2 μsec. This gives an output rate of about 8681 chars/sec.

Question 31: Assuming that it takes 10 nsec to copy a byte, how much time does it take to completely rewrite the screen of an 80 character **x** 25 line text mode memory-mapped screen? What about a 1024 **x** 768 pixel graphics screen with 24-bit color?

Answer: Rewriting the text screen requires copying 2000 bytes, which can be done in 20 μseconds. Rewriting the graphics screen requires copying 1024 × 768 × 3 = 2,359,296 bytes, or about 23.6 msec.

Question 32: In Fig. 5-40 there is a class to RegisterClass. In the corresponding X Window code, in Fig. 5-38, there is no such call or anything like it. Why not?

Answer: In Windows, the OS calls the handler procedures itself. In X Windows, nothing like this happens. X just gets a message and processes it internally.

Question 33: In the text we gave an example of how to draw a rectangle on the screen using the Windows GDI: Rectangle(hdc, xleft, ytop, xright, ybottom);
Is there any real need for the first parameter (hdc), and if so, what? After all, die coordinates of the rectangle are explicitly specified as parameters.
Answer: The first parameter is essential. First of all, the coordinates are relative to some window, so hdc is needed to specify the window and thus the origin. Second, the rectangle will be clipped if it falls outside the window, so the window coordinates are needed. Third, the color and other properties of the rectangle are taken from the context specified by hdc. It is quite essential.
Question 34: It has been observed that the THINC system works well with a 1-Mbps network in a test. Are any problems likely in a multiuser situation? Hint: Consider a large number of users watching a scheduled TV show and the same number of users browsing the World Wide Web.
Answer: The bandwidth on a network segment is shared, so 100 users requesting different data simultaneously on a 1-Mbps network will each see a 10-Kbps effective speed. With a shared network, a TV program can be multicast, so the video packets are only broadcast once, no matter how many users there are and it should work well. With 100 users browsing the Web, each user will get 1/100 of the bandwidth, so performance may degrade very quickly.
Question 35: If a CPU's maximum voltage, V, is cut to V7«, its power consumption drops to 1/n2 of its original value and its clock speed drops to 1/n of its original value. Suppose that a user is typing at 1 char/sec, but the CPU time required to process each character is 100 msec. What is the optimal value of **n** and what is the corresponding energy saving in percent compared to not cutting the voltage? Assume that an idle CPU consumes no energy at all.
Answer: If n = 10, the CPU can still get its work done on time, but the energy used drops appreciably. If the energy consumed in 1 sec at full speed is E, then running at full speed for 100 msec then going idle for 900 msec uses E/10. Running at 1/10 speed for a whole second uses E/100, a saving of 9E/100. The percent savings by cutting the voltage is 90%.
Question 36: A notebook computer is set up to take maximum advantage of power saving features including shutting down the display and the hard disk after periods of inactivity. A user sometimes runs UNIX programs in text mode, and at other times uses the X Window System. She is surprised to find that battery life is significantly better when she uses text-only programs. Why?
Answer: The windowing system uses much more memory for its display and uses virtual memory more than the text mode. This makes it less likely that the hard disk will be inactive for a period long enough to cause it to be automatically powered down.

Objectives of OS: – convenience, – efficiency, – extensibility.

5.6. What advantage is there in having different time-quantum sizes on different levels of a multilevel queueing system?

Processes which need more frequent servicing, such as interactive processes, can be in a queue with a small q. Processes that are computationally intensive can be in a queue with a larger quantum, requiring fewer context switches to complete the processing, making more efficient use of the CPU.

4.2. Describe the differences among short-term, medium-term, and long-term scheduling.

- Short-term (CPU scheduler): selects a process from those that are in memory and ready to execute, and and allocates the CPU to it.
- Medium-term (memory manager): selects processes from the ready or blocked queue and removes them from memory, then reinstates them later to continue running.
- Long-term (job scheduler): determines which jobs are brought into the system for processing.

4.6. Describe the actions taken by a kernel to switch context

(a) Among threads

(b) Among processes

- The thread context (registers, PC, and stack, plus accounting info if appropriate) must be saved and another thread's context must be loaded.
- The same as (a), except that the memory and resource info must also be stored and that for the next process must be loaded.

4.7. What are the differences between user-level threads and kernel-supported threads? Under what circumstances is one type "better" than the other?

- User-level threads have no kernel support, so they are very inexpensive to create, destroy, and switch among. Kernel-supported threads are more expensive because system calls are needed to create and destroy them and the kernel must schedule them.
- In most current systems with threads, when a user-level thread blocks, the whole process blocks. Since kernel-supported threads are scheduled independently, they can block individually; that is, the fact that one blocks does not hold up the others.

Q. int foo=5;

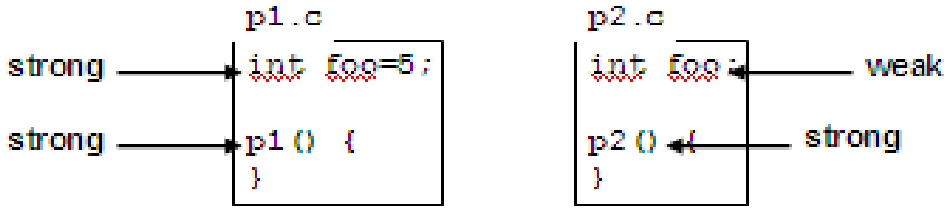
p1()l

which is strong symbol and which is weak symbol.

Sol:

Strong and Weak Symbols

- Program symbols are either strong or weak
  - *strong*: procedures and initialized globals
  - *weak*: uninitialized globals



Q, using SJF(short job first) calculate the waiting time.

example 3:

Process	Burst Time	Arrival	Start	Wait	Finish	TA
1	10	0	10	10	20	20
2	29	0	32	32	61	61
3	3	0	0	0	3	3
4	7	0	3	3	10	10
5	12	0	20	20	32	32

Gantt chart:

SOL:

average waiting time: (10+32+0+3+20)/5 = 13

average turnaround time: (10+39+42+49+61)/5 = 25.2

5.4. Suppose that the following processes arrive for execution at the times indicated. Each process will run the listed amount of time. In answering the questions, use nonpreemptive scheduling and base all decisions on the information you have at the time the decision must be made.

Process	Arrival Time	Burst Time
P1	0.0	8
P2	0.4	4
P3	1.0	1

(a) What is the average turnaround time for these processes with the FCFS scheduling algorithm?

(b) What is the average turnaround time for these processes with the SJF scheduling algorithm?

(c) The SJF algorithm is supposed to improve performance, but notice that we chose to run process P1 at time 0 because we did not know that two shorter processes would arrive soon. Compute what the average turnaround time will be if the CPU is left idle for the first 1 unit and then SJF scheduling is used. Remember that processes P1 and P2 are waiting during this idle time, so their waiting time may increase. This algorithm could be known as future-knowledge scheduling.

- (a) 10.53
- (b) 9.53
- (c) 6.86

Q4/what is main trouble programmers face while writing real time operating system 5marks

Sol:

The main difficulty is keeping the operating system within the fixed time constraints of a real-time system. If the system does not complete a task in a certain time frame, it may cause a breakdown of the entire system it is running. Therefore when writing an operating system for a real-time system, the writer

must be sure that his scheduling schemes don't allow response time to exceed the time constraint.

Q. Explicit vs. Implicit Memory Allocator

SOL:

- **Explicit: application allocates and frees space**
  - **E.g., malloc and free in C**
- **Implicit: application allocates, but does not free space**
  - **E.g. garbage collection in Java, ML or Lisp**

Question:

12. An operating system is running three concurrent processes, P_1, P_2, and P_3 for three different users. Each process requires six units of computing time. Process P_1 consists of a single thread. Process P_2 consists of two threads, each requiring three of P_2's six total units of computing time. Process P_3 consists of three threads, each requiring two of P_3's six total units of time. Assume that the threads never block and that the processes are entirely resident in primary memory.

    a. Suppose that the threads are kernel-supported, and that the kernel implements preemptive, round-robin scheduling of threads, with a quantum on one time unit. Of the first six units of time, how many will be devoted to each of the three processes?

    b. Suppose that threads are implemented entirely at the user level, and that the kernel implements preemptive round-robin scheduling of processes with a quantum of one time unit. Of the first six units of time, how many will be devoted to each of the three processes?

    c. Suppose that we define a *fair share* scheduler to be one that devotes $1/n$ of the processor's time to each process when there are n processes in the system. Are the schedulers described in parts (a) and (b) fair share schedulers? If not, suggest how they might be modified to become fair share schedulers. Your suggestion(s) should require no more than a sentence or two.

Solution:

12. **a.** Process 1 will receive one time unit, process 2 will receive 2 units, and process 3 will receive three.

b. Each process will receive two time units.

c. Scheduler (b) is a fair share scheduler. Scheduler (a) is not. Scheduler (a) could be modified to become a two-level scheduler. The top level would select a process to run, using a fair scheduling discipline like round-robin. The bottom level would select one thread from the process that was selected by the top level scheduler.

Sol 1.

The purpose of memory management is to ensure fair, secure, orderly, and efficient use

of memory.

#15 Writ down the main difference between preemitive and non-premitive scheduling schemes and write down at least one from each.

Sol:

Tasks are usually assigned with priorities. At times it is necessary to run a certain task that has a higher priority before another task although it is running. Therefore, the running task is interrupted for some time and resumed later when the priority task has finished its execution. This is called preemptive scheduling.

Eg: Round robin

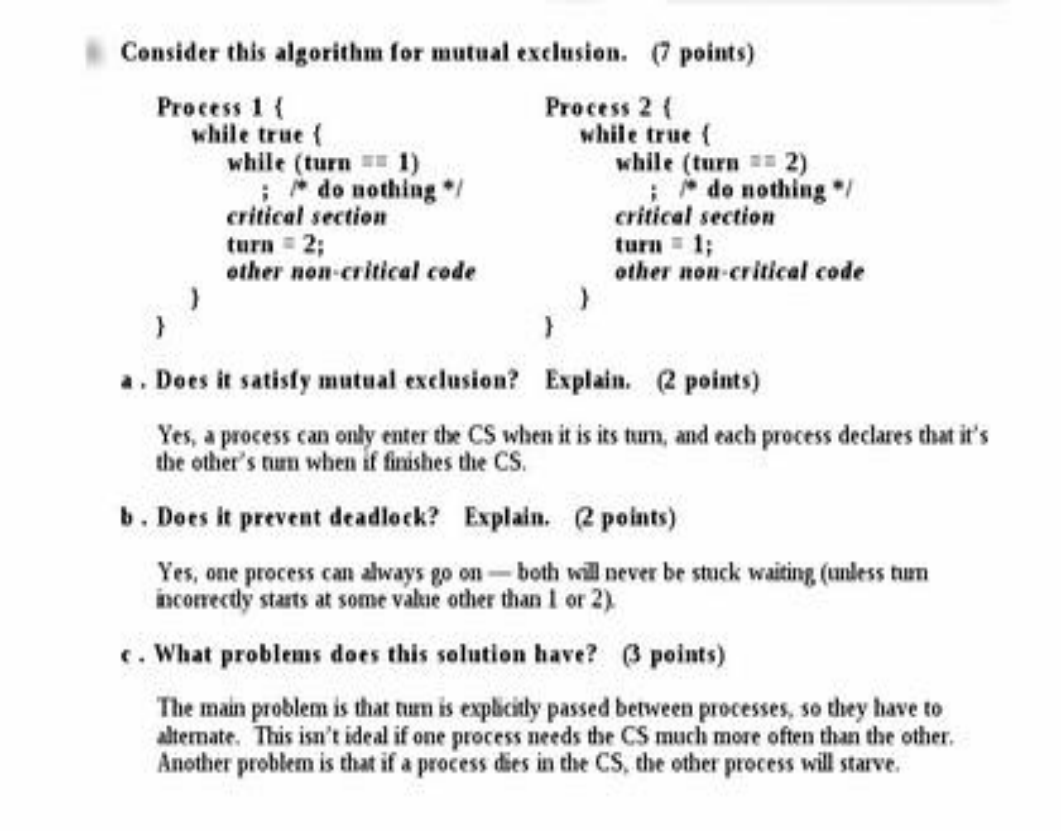
In non-preemptive scheduling, a running task is executed till completion. It cannot be interrupted.

Eg First In First Out.

Q#17 this was about two processes like, will they produce deadlock

```
process1(){
while true{
while(turn ==1); /* do nothing*/
turn = 2;
critical section
}
}
 Process2(){
while true{
```

while(turn == 2); /* do nothing */
turn = 1;
critical section
}
}
Sol: no they donot produce deadlock:



Consider this algorithm for mutual exclusion. (7 points)

```
Process 1 {                          Process 2 {
   while true {                         while true {
      while (turn == 1)                    while (turn == 2)
         ; /* do nothing */                   ; /* do nothing */
      critical section                     critical section
      turn = 2;                            turn = 1;
      other non-critical code              other non-critical code
   }                                    }
}                                    }
```

a. Does it satisfy mutual exclusion? Explain. (2 points)

Yes, a process can only enter the CS when it is its turn, and each process declares that it's the other's turn when if finishes the CS.

b. Does it prevent deadlock? Explain. (2 points)

Yes, one process can always go on — both will never be stuck waiting (unless turn incorrectly starts at some value other than 1 or 2).

c. What problems does this solution have? (3 points)

The main problem is that turn is explicitly passed between processes, so they have to alternate. This isn't ideal if one process needs the CS much more often than the other. Another problem is that if a process dies in the CS, the other process will starve.

Q#13 Do you think there are some events in computer systems that causes CPU scheduling?

- CPU scheduling decisions take place under one of four conditions:
  1. When a process switches from the running state to the waiting state, such as for an I/O request or invocation of the wait( ) system call.
  2. When a process switches from the running state to the ready state, for example in response to an interrupt.
  3. When a process switches from the waiting state to the ready state, say at completion of I/O or a return from wait( ).
  4. When a process terminates.

Q2. Describe any three main objective of an operating system.

Answer:

•To provide an environment for a computer user to execute programs on computer

hardware in a convenient and efficient manner.
•To allocate the separate resources of the computer as needed to solve the problem given. The allocation process should be as fair and efficient as possible.
•As a control program it serves two major functions: (1) supervision of the execution of user programs to prevent errors and improper use of the computer, and (2) management of the operation and control of I/O devices.

Q3. Features of real time operating system.5 marks

Answer RTOS is a real time operating system. The **important features** are :-

1. The necessary signaling functions between interrupt routines and task codes are handled by RTOS.

2. It works as an independent system with no internal or external interdependencies.
3. There are no loop decisions in RTOS
4. The RTOS can suspend one task code subroutine in the middle order to run another
5. The time lag is very less compared to other systems
6. There are no random time variables; this is good for a direct relationship between instruction and process.
7. Tasks are simpler to write.
8. Under most RTOS tasks are simply subroutines

Q4. linux implements soft real-time scheduling what features are missing to make it real time programming task? How might they be added to the kernel?

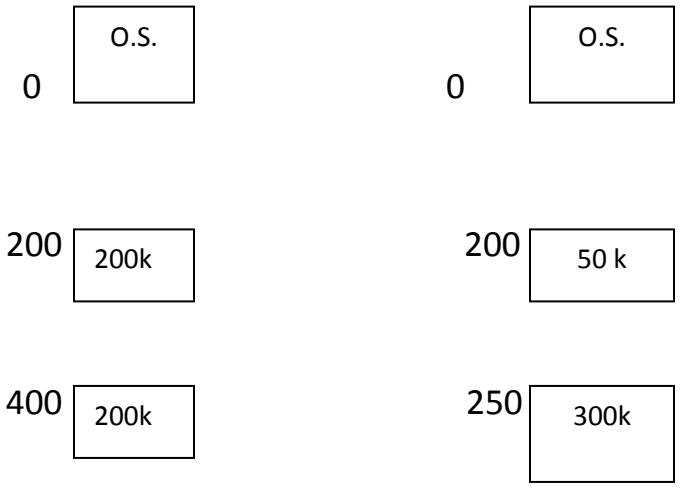
Sol: Answer: Linux's "soft" real-time scheduling provides ordering guarantees concerning the priorities of runnable processes: real-time processes will always be given a higher priority by the scheduler than normal time-sharing processes, and a real-time process will never be interrupted by another process with a lower real-time priority.
However, the Linux kernel does not support "hard" real-time functionality. That is, when a process is executing a kernel service routine, that routine will always execute to completion unless it yields control back to the scheduler either explicitly or implicitly (bywaiting for some asynchronous event). There is no support for preemptive scheduling of kernel-mode processes. As a result, any kernel system call that runs for a significant amount of time without rescheduling will block execution of any real-time processes.
Many real-time applications require such hard real-time scheduling. In particular, they often require guaranteed worst-case response times to external events. To achieve these guarantees, and to give user-mode real time processes a true higher priority than kernel-mode lower-priority

processes, it is necessary to find a way to avoid having to wait for lowpriority kernel calls to complete before scheduling a real-time process.
For example, if a device driver generates an interrupt that wakes up a high-priority real-time process, then the kernel needs to be able to schedule that process as soon as possible, even if some other process is already executing in kernel mode.
Such preemptive rescheduling of kernel-mode routines comes at a cost. If the kernel cannot rely on non-preemption to ensure atomic updates of shared data structures, then reads of or updates to those structures must be protected by some other, finer-granularity locking mechanism. This fine-grained locking of kernel resources is the main requirement for provision of tight scheduling guarantees.
Many other kernel features could be added to support real-time programming. Deadline-based scheduling could be achieved by making modifications to the scheduler. Prioritization of IO operations could be implemented in the block-device IO request layer.

Q5. Is MVT (multiprograming with variable task) and MFT (multiprograming with fixed task). which one is best suited for internal fregmentation and which one is suited for external fregmentation. justify your answer?

SOL:

- **MULTIPROGRAMMING WITH FIX PARTITION(Contagious Memory Allocation):**
  - To support the multiprogramming the main memory is divide into fix partitions.
  - These partitions may be equal size or unequal size.



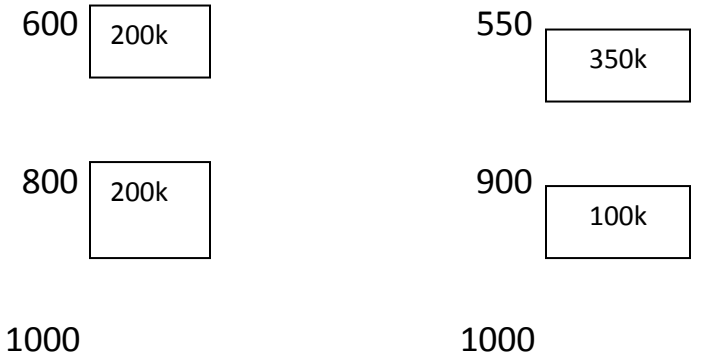


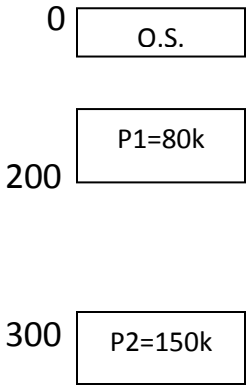
Fig.1[equal-size partition] Fig.2[unequal-size partitions]

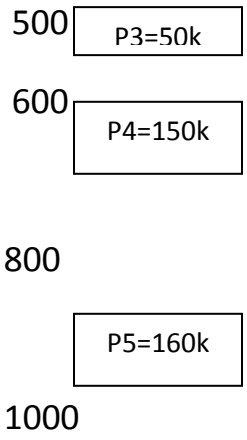
- In fig.1 memory is divided into 4 equal size partitions. Each partitions of 200k.
- In fig.2 memory is divide into 4 unequal size partitions. partition size is not equal.
- Operating system will load the processes into the partitions.
- Suppose main memory is divide into 8 partitions. Than 8 process can be loaded into memory. Only one process can enter into one partitions.

- There are two types of fragmentation:
  1. Internal Fregmentation

  2. External Fragmentation

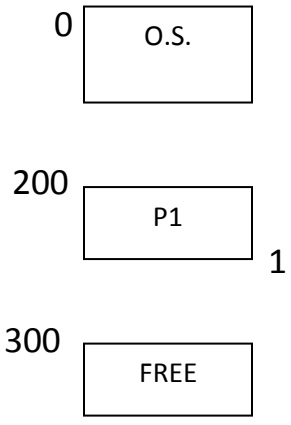
- **Internal Fregmentation(**MFT (multiprograming with fixed task))
- **Defination:** Difference between total space of partition and space occupied by process is called internal fregmentation.
- In fixed partitions internal fregmentation will be created.

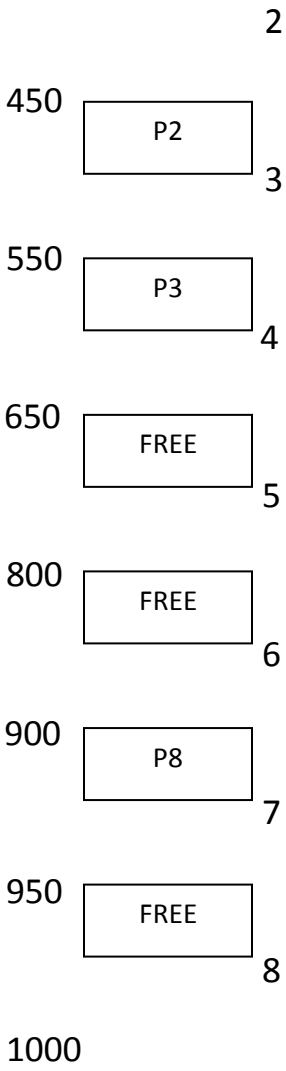




- Here p1,p2,p3,p4,p5 is loaded into memory.
- Process P1 is loaded into first partition whose size is 100k and size of process P1 is 80k ,so in partition 20k space is waste. This is called Internal Fregmentation.
- For partition-2 size of partition is 200k and size of P2 is 150k , so here Internal fregmentation of 50k.
- When process will execute it will remove from memory. And new process will enter into memory in the free partition.
- Operating system keep information about the free and allocated partitions.
- There are three strategies are used to allocate memory into free partitions.

    1. **First fit**
    2. **Best fit**
    3. **worst fit**





- Here total 8 partitions. Partition-2nd ,5th,6th and 8th are free and other partitions are allocated processes.
- Now to allocate new processes into these free partitions 3 algorithms are used.

First fit

Allocate first free partition which is enough for process. searching will be start from first partitions.

Best fit

Allocate the small free partition which is small and enough for process. Here all partition will be search from top to bottom and allocated the partition in which less amount of waste of space.

Worst Fit

Allocate the largest partition in which high amount of waste of space, here all partitions will be search from top to bottom.

- Best fit algorithms is most better compare to first fit and worst fit.
- First Fit Algorithm is faster compare to other.

Advantage & Disadvantage of Fixed Partition

Advantage : - simple to implement

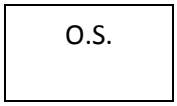
Disadvantage.: - Problem of internal fregmentation.

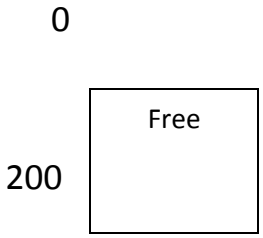
- Not efficient use of memory.

- **MULTIPROGRAMMING WITH DYNAMIC PARTITION** External Fragmentation **(variable size):-(MVT)**
  - To avoid the difficulties of fixed partitions, partitions are created dynamically.

  - In Dynamic partitions, no partition is fixed into memory but the partition are created only when process is load into memory, the partitions size is exactly size of process. If process size is 125k, when process will enter into memory than 125k size partition will be created.

  - If we have 5 process p1, p2,p3,p4,p5 and size of user process area is 800k which is currently free and no any partition.





- Size of each process is :

  p1 – 80 k

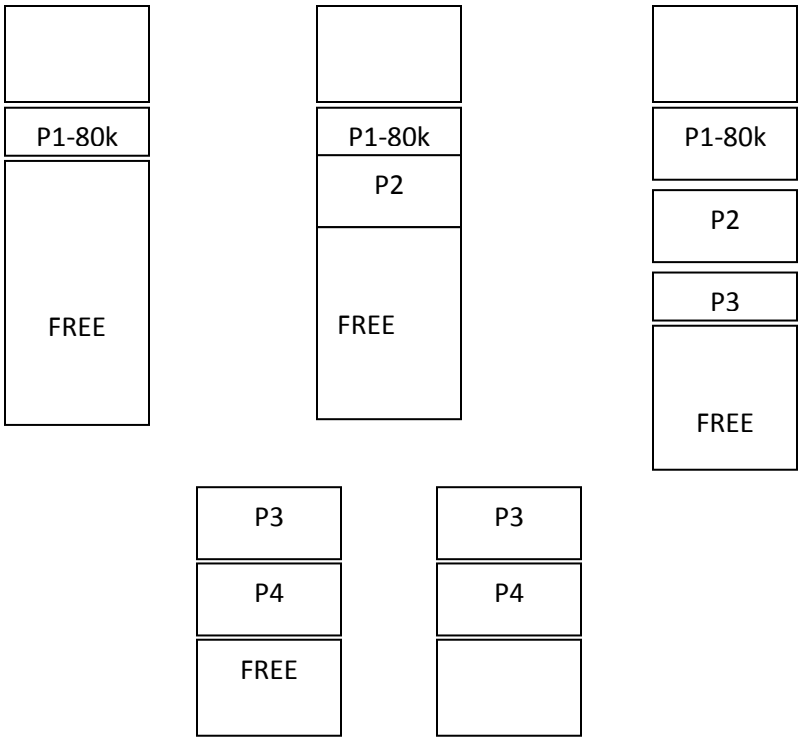
  p2 – 150k

  p3 – 170k

  p4 – 300k

  p5 – 40k

- when process will loaded than partition will be created.



External

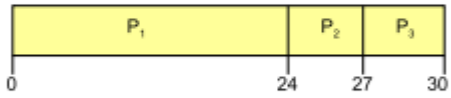
Fregmentation

Here the not problem of internal fragmentation. But here the problem of external fregmentation.

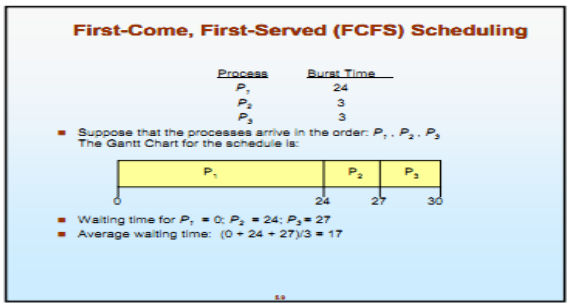
- **External Fregmentation:**
- **Definition** : Difference between total space of memory and total memory allocated to processes is called **external fregmentation**.
- when all process will be loaded than waste of space at end of memory is called external fregmentaion.

Example: FCFS Scheduling

- **Run jobs in order that they arrive**
  - Called "*First-come first-served*" (FCFS)
  - E.g., Say P_1 needs 24 sec, while P_2 and P_3 need 3.
  - Say P_2, P_3 arrived immediately after P_1, get:



- **Dirt simple to implement—how good is it?**
- **Throughput: 3 jobs / 30 sec = 0.1 jobs/sec**
- **Turnaround Time:** P_1 : 24, P_2 : 27, P_3 : 30
  - Average TT: $(24 + 27 + 30)/3 = 27$



***Q7. Solution to the Critical Section Problem* must meet three conditions...**

1. **mutual exclusion**: if process p_i is executing in its critical section, no other process is executing in its critical section
2. **progress**: if no process is executing in its critical section and there exists some processes that wish to enter their critical sections, then only those processes that are not executing in their remainder section can participate in the decision of which will enter its critical section next, and this decision cannot be postponed indefinitely
    - if no process is in critical section, can decide quickly who enters
    - only one process can enter the critical section so in practice, others are put on the queue
3. **bounded waiting**: there must exist 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
    - The wait is the time from when a process makes a request to enter its critical section until that request is granted
    - in practice, once a process enters its critical section, it does not get another turn until a waiting process gets a turn (managed as a queue)

Q. Three processes are sharing 4 resources of same type in system. Every process need 2 resources show system is deadlock free.

solution

If the system is deadlocked, it implies that each process is holding one resource and is waiting for one more. Since there are 3 processes and 4 resources, one process must be able to obtain two resources. This process requires no more resources and therefore it will return its resources when done.

Q. 2 process-three resources are there? is there deadlock possible?

Sol:

Same above.

Q. block size of single word and double word

SOL

word is a collection of characters which is 16 bits in size.

Double words = 2 words or 32bits.

1. Q:

In your opinion what could be the ways that make it possible that the processes should not enter in critical section simultaneously and fulfill critical section requirements in such a way that no need to code for it explicitly.

2. Q:

Why is it difficult to implement mutual exclusion and condition variables in an environment where CPUs do not share any memory?

3. Q:

What advantage is there in having different time-quantum sizes on different levels of a multilevel queueing system?

4. Q:

Determine the minimum block size for each of the following combinations of alignment requirements and block formats. Assumptions: Implicit free list, zero sized payloads are not allowed, and headers and footers are stored in 4-byte words.

Alignment	Allocated block	Free block	Minimum block size (bytes)
Single word	Header and footer	Header and footer	_____
Single word	Header, but no footer	Header and footer	_____
Double word	Header and footer	Header and footer	_____
Double word	Header, but no footer	Header and footer	_____

5. Q:

Out of two solutions given below. Which solution do you think is better than other for solving critical section problem? Give reasons to support your answer.

Solution 1:	Solution 2:

boolean flag[2];
int turn;
do
{
flag[i]=true;
turn=j;
while(flag[j] && turn==j);
critical section
flag[i]=false;
	boolean flag[2]
	do
	{
	flag[i]=true;
	while(flag[j]);
	critical section
	flag[i]=false;
	remainder section
	} while(1)

remainder section } while(1)	

6. Q:

   Describe four (non-disjoint) classes of thread-unsafe functions.

7. Q:

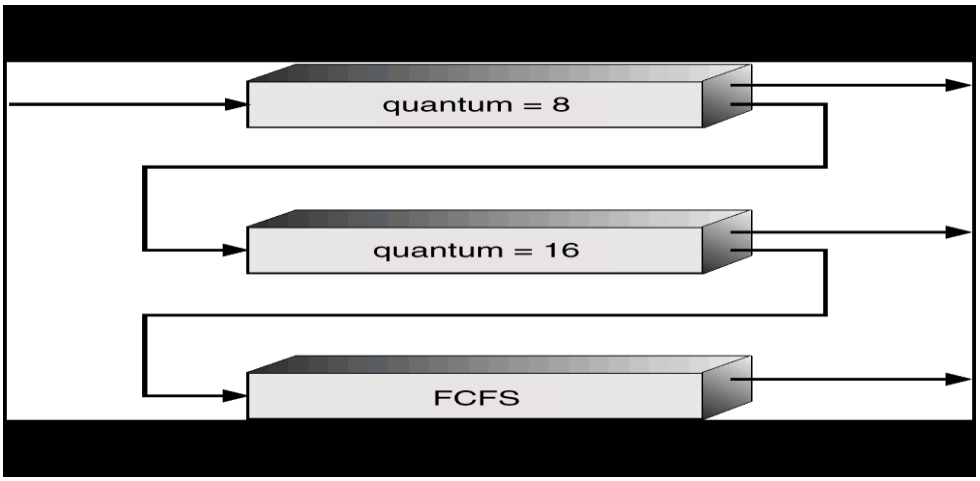
There are five processes given below with their run time units. Assume that all processes arrived in order at time $P_1 = 0$ $P_2 = 4$ $P_3 = 2$ $P_4 = 5$ $P_5 = 0$ respectively, then answer the questions given below:

Process ID	CPU Requirements
P_1	6
P_2	4
P_3	9
P_4	1
P_5	10

Show the scheduling order for these processes under first-come-first-served, shortest-remaining-job-first, and round-robin scheduling (quantum = 4).

8. Q:

An example diagram for multilevel feedback queue scheduling is given below.



Let us consider that there are two processes. Process P1 time period is 18 and process P2 time period is 27. Answer the following questions

a) Which process will complete its execution in 2nd queue?
b) How each process will move across different queues i.e. how each process will enter into the system and how each process will move up and down between different queues.

Q1.

What are the extra costs incurred by the creation and scheduling of a process, as compared to the cost of a cloned thread?

Sol:

Creation of a thread in Linux involves only the creation of some very simple data structures to describe the new thread. Space must be reserved for the new thread's execution context its saved registers, its kernel stack page and dynamic information such as its security profile and signal state but no new virtual address space is created.

Creating this new virtual address space is the most expensive part of the creation of a new process. The entire page table of the parent process must be copied, with each page being examined so that copy on- write semantics can be achieved and so that reference counts to physical pages can be updated. The parent process's virtual memory is also affected by the process creation: any private read/write pages owned by the parent must be marked read-only so that copy-on-write can happen (copy-on-write relies on a page fault being generated when a write to the page occurs).

Scheduling of threads and processes also differs in this respect. The decision algorithm performed when deciding what process to run next is the same regardless of whether the process is a fully independent process or just a thread, but the action of context-switching to a separate process is much more costly than switching to a thread. A process requires that the CPU's virtual memory control registers be updated to point to the new virtual address space's page tables.

In both cases: creation of a process or context switching between processes the extra virtual memory operations have a significant cost. On many CPUs, changing page tables or swapping between page tables is not cheap: all or part of the virtual address translation look aside buffers in the CPU must be purged when the page tables are changed. These costs are not incurred when creating or scheduling between threads.

Q. Describe the main features of Blocking and Non-blocking I/O.

Sol:

The main features of Blocking and Non-blocking I/O devices are as under:

Blocking:

I. process suspended until I/O completed
II. Easy to use and understand
III. Insufficient for some needs

Non-blocking:

I. I/O call returns as much as available
II. User interface, data copy (buffered I/O)
III. Implemented via multi-threading inside the kernel
IV. Returns quickly with count of bytes read or written

Q.How authentication is achieved through Biometrics?

Sol:

Authentication of a person based on a physiological or behavioral characteristic is called biometric authentication . A number of biometric methods have been introduced over the years, but few have gained wide acceptance.

Eye scans: This favorite of spy movies and novels presents its ownproblems. The hardware is expensive and specialized, and using it is slow and inconvenient and may make users uneasy. In fact, two parts of the eye can be scanned, using different technologies: the retina and theiris.

Fingerprint recognition: Everyone knows fingerprints are unique. They are also readily accessible and require little physical space either for the reading hardware or the stored data.

Hand or palm geometry: We're used to fingerprints but seldom think of an entire handas an individual identifier. This method relies on devices that measure the length and angles of individual fingers. Although more user-friendly than retinal scans, it's still cumber some.

Voice recognition: This is different from speech recognition. The idea is toverify the individual speaker against a stored voice pattern, not to understand what isbeing said.

Facial recognition: Uses distinctive facial features, including upper outlines of eye sockets, areas around cheekbones, the sides of the mouth and the location of the nose and eyes. Most technologies avoid areas of the face near the hairline so that hairstyle changes won't affectrecognition.

Signature dynamics: Based on an individual's signature, but considered unforgeable because what is recorded isn't the final image but how it is produced -- i.e., differences in pressure and writing speed at various points in thesignature.

Typing patterns: Similar to signature dynamics but extended to the keyboard, recognizingnotjustapasswordthatistypedinbuttheintervalsbetweencharacters andtheoverallspeedsandpattern.ThisisakintothewayWorldWarIIintelligence analystscouldrecognizeaspecificcovertagent'sradiotransmissionsbyhis"hand" the way he used the telegraphkey.

Q. Page Table Entries control mapping is given below, you are required to describe the meaning of each bit.

V	R	M	Prot	Page frame number

Solution:

V (Valid Bit): The valid bit consist of 1 Bit says whether or not the PTE can be used. It also says that whether or not a virttual address is valid. It is checked each of the time a virtual address is used.

R(Referenced Bit): The Referenced Bit says whether the page has been accessed. It is set when page has been read or written to.

M(Modified Bit): The Modified Bit says whether or not the page is dirty. It is set when a write to the page has occurred.

Prot(Protection Bit): The Protection Bit control which operations are allowed to read, write and execute.

Page Frame Number: The Page Frame Number determines the physical page. The start address of physical page is PFN.

Q. In the following chunk of code, describe the functionality of each line.

```
/*Parent's Code */

ssize_t read (int fd, void *buf, size_t count);

{

    1. close(pipefd[1]);
    2. n = strlen(testString);
    3. nr = read(pipefd[0], buf, nA);
    4. rc = write(1, buf, nr);
    5. wait(&status);
    6. printf("Good work child!\n");
    7. return(0);
}
```

Sol:

Code	Functionality
ssize_t read(int fd, void buf, size_t count)	The line attempts to read up to *size_tcount* from file descriptor *fd* into the buffer starting at *buf*. On files that support seeking, the read operation commences at the current file offset, and the file offset is incremented by the number of bytes read. If the current file offset is at or past the end of file, no bytes are read, and **read**() returns zero.
close(pipe[1]);	Close the write side of Pipe and Reader will see EOF
n=strlen(testingString);	Returns the length of the string *testing String* into variable n
nr= read(pipe[0], buf, nA);	The function shall attempt to read *nA* bytes from the file associated with the open file descriptor,*fildes*, into the buffer pointed to by *buf*. The behavior of multiple concurrent reads on the same pipe, FIFO, or terminal device is unspecified. The function shall return a non-negative integer into nr indicating the number of bytes actually read.
rc = write(1, buf, nr);	The function will write up to *nr* bytes to the standard output from the buffer starting at *buf.* on success, the number of bytes written is returned by the function and stored in rc.
wait(&status);	suspends the execution of the calling process until one of its children terminates.
printf("Good work child");	print Good Work child on out put screen

Summary of decription of the Program:

In the given program, the parent process first creates a pipe and then forks a child process. On successful execution, the pipe() system call creates a pipe, with its read end descriptor stored in pipefd[0] and write end descriptor stored in pipefd[1]. We call fork() to create a child process, and then use the fact that the memory image of the child process is identical to the memory image of the parent process, so the

pipefd[] array is still defined the same way in both of them, and thus they both have the file descriptors of the pipe. Further more, since the file descriptor table is also copied during the fork, the file descriptors are still valid inside the child process. Thus, the parent and child processes can use the pipe for one-way communication as outlined above

Q. In a multiprogramming and time-sharing environment, several users share the system simultaneously. This situation can result in various security problems.

a. What are two such problems?

b. Can we ensure the same degree of security in a time-shared machine as we have in a dedicated machine? Explain your answer.

Sol:

a. Stealing or copying one's programs or data; using system resources (CPU, memory, disk space, peripherals) without proper accounting.

b. Probably not, since any protection scheme devised by humans can inevitably be broken by a human, and the more complex the scheme, the more difficult it is to feel confident of its correct implementation.

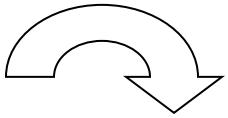
Q. Following are the prototypes of two C functions:

```
char *strcpy(char *dest, const char *src);

char *strncpy(char *dest, const char *src, size_t n);
```

Briefly describe why strcpy is a dangerous function that may invite buffer overflow attacks whereas strncpy is safer in this respect.

Sol



Top of box

Program P: exec(“”/bin/sh”)

The strcpy() function copies the string pointed to be src (including the terminating `\0' character) to the array pointed to by dest. The strings may not overlap, and the destination string dest must be large enough to receive the copy. The strncpy() function is similar, except that not more than n bytes of src are copied. Thus, if there is no null byte among the first n bytes of src, the result wil not be null-terminated. In the case where the length of src is less than that of n, the remainder of dest will be padded with nulls.

RETURN Value

The strcpy() and strncpy() functions return a pointer to the destination string dest.

Bugs

If the destination string of a strcpy() is not large enough (that is, if the programmer was stupid/lazy, and failed to check the size before copying) then anything might happen. Overflowing fixed length strings is a favourite cracker technique.

So we can say that the function strcpy is dangerous function because there is no range checking, it is with a typical buffer overflow coding error.

MIDTERM FALL 2015
PAPER=3 DATED:27.12.2015
IMRAN

MCQs Marks=10 **Time= 1hour** **Total Marks=40**

1.	A virtual address has two parts which are virtual page _____ & offset.
a.	Title
b.	**Number**
c.	Frame
d.	Value
2.	In _____ Access method FS contains an index to a particular field of each record in a file and applications can find a file based on value in that record.
a.	Direct
b.	Sequential

c.	Record
d.	**Indexed**
3.	In Fast File system multiple file systems with different _____ sizes can co-reside.
a.	**Block**
b.	Cell
c.	Sector
d.	Track
4.	In Virtual File System _____ object represents a specific file.
a.	Inode
b.	**Dentry**
c.	Superblock
d.	File
5	Network Devices varying enough from block and character to have own _____.
a.	Address
b.	Driver
c.	Buffer
d.	**Interface**
6.	_____timers may be dynamically created and destroyed.
a.	I/O
b.	Static
c.	**Dynamic**
d.	Memory
7.	_____ are free program made available to unsuspecting user which actually contains code to do harm.
a.	Malwares
b.	Suspicions programs
c.	**Trojan Horses**
d.	Logic bombs
8	Security reference monitor uses _____to identify the security context of a process or thread.
a.	**Tokens**
b.	Frames
c.	Pages
d.	Reference
9	
a.	
b.	
c.	
d.	
10	
a.	
b.	
c.	
d.	

Long Questions Marks=30

1.	For an arbitrary sized atomic disk write down the functionality of the following two functions. put(blk, address) : get(address) -> blk :	5
2.	Describe the main roles of a conflict resolution module?	5
3.	How does a hacker find a buffer overflow? • Run web server on local machine. • Issue requests with long tags. All long tags end with "$$$$$". • If web server crashes, search core dump for "$$$$$" to find overflow location. • Some automated tools exist. (eEye Retina, ISIC).	5
4.	Consider a scenario when multiple processes are running in a system and CPU is switching among processes. In multiprogramming, the main issue is CPU protection. If a process doesn't relinquish CPU for a long or infinite period of time then how will you prevent the user process or program from getting stuck in an infinite loop or not calling system services and never returning control to the CPU? Write down all necessary steps to accomplish CPU protection in this scenario.	10
5.	Provide details of upper-level data structures for file system support. Also write a detailed note on In-Memory Data Structures.	
6.	Why do some systems keep track of the type of a file, while others leave it to the user or simply do not implement multiple file types. Which system is better and why.	10

7.	Following are the prototypes of two C functions:
	
	char *strcpy(char *dest, const char *src);
	
	char *strncpy(char *dest, const char *src, size_t n);
	
	Briefly describe why strcpy is a dangerous function that may invite buffer overflow attacks whereas strncpy is safer in this respect.
		10

Today Paper CS703

Dated: 29-05-2016 Time: 7:30am

Q. 1 (10)

10 MCQs related Subject

Q.2 **(5)**

What is the main difference between preemptive and non-preemptive scheduling? Write the name of one Preemptive and one Non-preemptive algorithm?

Q.3 **(5)**

Describe a mechanism by which one segment could belong to the address space of two different processes?

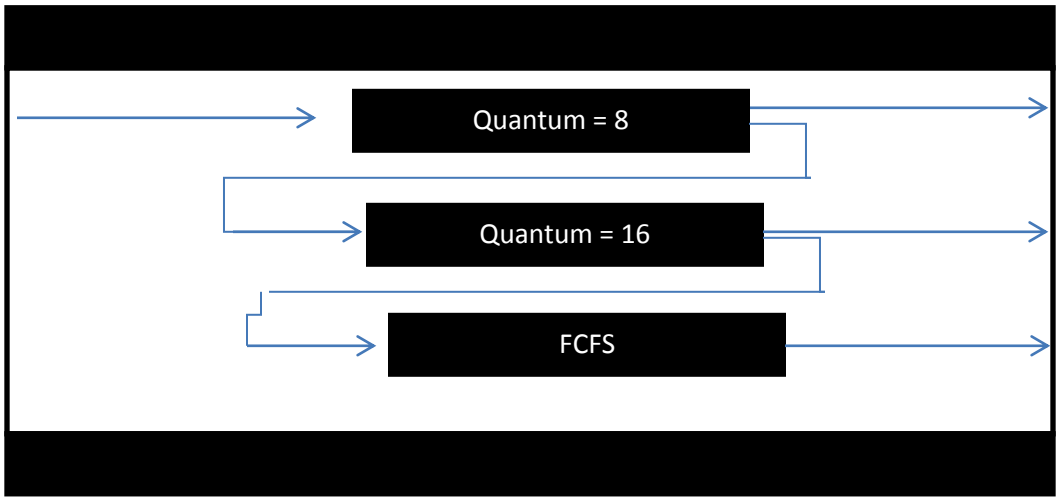
Q. 4 **(5)**

The ctime_ts for is thread-safe but not reentrant. Explain your answer with valid reason?

```
Char * ctime_ts (const time_ts *timep, char * private)
{
Char *sharedp;
P (&mutex);
Sharedp = ctim(timep);
Strcpy(private, sharedp);     /*copy string from shared to private */
V (&mutex);
Return private;
```

Q. 5 **(10)**

The mention example diagram for multilevel feedback queue scheduling is given below:



Let us consider: Process P1 time period is 18 and Process P2 time period is 27. Answer the following questions:

1. Which process will complete its execution in 2nd queues?
2. How each process will move across different queues i.e. how each process will enter into the system and how each process will move up and down between different queues?

Objectives of OS: – convenience, – efficiency, – extensibility.

Q. int foo=5;

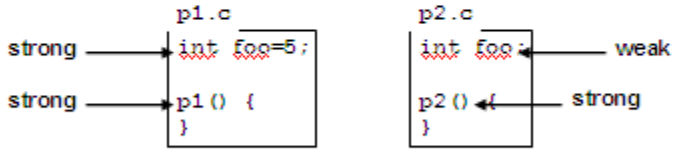
p1()l

which is strong symbol and which is weak symbol.

Sol:

Strong and Weak Symbols

- Program symbols are either strong or weak
  - *strong*: procedures and initialized globals
  - *weak*: uninitialized globals



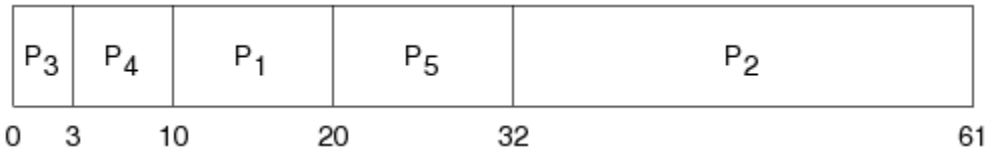
Q, using SJF(short job first) calculate the waiting time.

example 3:

Process	Burst Time	Arrival	Start	Wait	Finish	TA
1	10	0	10	10	20	20
2	29	0	32	32	61	61
3	3	0	0	0	3	3
4	7	0	3	3	10	10

5	12	0	20	20	32	32

Gantt chart:



SOL:

average waiting time: (10+32+0+3+20)/5 = 13

average turnaround time: (10+39+42+49+61)/5 = 25.2

Q4/what is main trouble programmers face while writing real time operating system 5marks

Sol:

The main difficulty is keeping the operating system within the fixed time constraints of a real-time system. If the system does not complete a task in a certain time frame, it may cause a breakdown of the entire system it is running. Therefore when writing an operating system for a real-time system, the writer must be sure that his scheduling schemes don't allow response time to exceed the time constraint.

Q. Explicit vs. Implicit Memory Allocator

SOL:

- **Explicit: application allocates and frees space**
  - **E.g., malloc and free in C**
- **Implicit: application allocates, but does not free space**
  - **E.g. garbage collection in Java, ML or Lisp**

Question:

12. An operating system is running three concurrent processes, P_1, P_2, and P_3 for three different users. Each process requires six units of computing time. Process P_1 consists of a single thread. Process P_2 consists of two threads, each requiring three of P_2's six total units of computing time. Process P_3 consists of three threads, each requiring two of P_3's six total units of time. Assume that the threads never block and that the processes are entirely resident in primary memory.

    a. Suppose that the threads are kernel-supported, and that the kernel implements preemptive, round-robin scheduling of threads, with a quantum on one time unit. Of the first six units of time, how many will be devoted to each of the three processes?

    b. Suppose that threads are implemented entirely at the user level, and that the kernel implements preemptive round-robin scheduling of processes with a quantum of one time unit. Of the first six units of time, how many will be devoted to each of the three processes?

    c. Suppose that we define a *fair share* scheduler to be one that devotes $1/n$ of the processor's time to each process when there are n processes in the system. Are the schedulers described in parts (a) and (b) fair share schedulers? If not, suggest how they might be modified to become fair share schedulers. Your suggestion(s) should require no more than a sentence or two.

Solution:

12. **a.** Process 1 will receive one time unit, process 2 will receive 2 units, and process 3 will receive three.

    b. Each process will receive two time units.

    c. Scheduler (b) is a fair share scheduler. Scheduler (a) is not. Scheduler (a) could be modified to become a two-level scheduler. The top level would select a process to run, using a fair scheduling discipline like round-robin. The bottom level would select one thread from the process that was selected by the top level scheduler.

Sol 1.

The purpose of memory management is to ensure fair, secure, orderly, and efficient use

of memory.

Q#15 Writ down the main difference between preemitive and non-premitive scheduling schemes and write down at least one from each.

Sol:

Tasks are usually assigned with priorities. At times it is necessary to run a certain task that has a higher priority before another task although it is running. Therefore, the running task is interrupted for some time and resumed later when the priority task has finished its execution. This is called preemptive scheduling.

Eg: Round robin

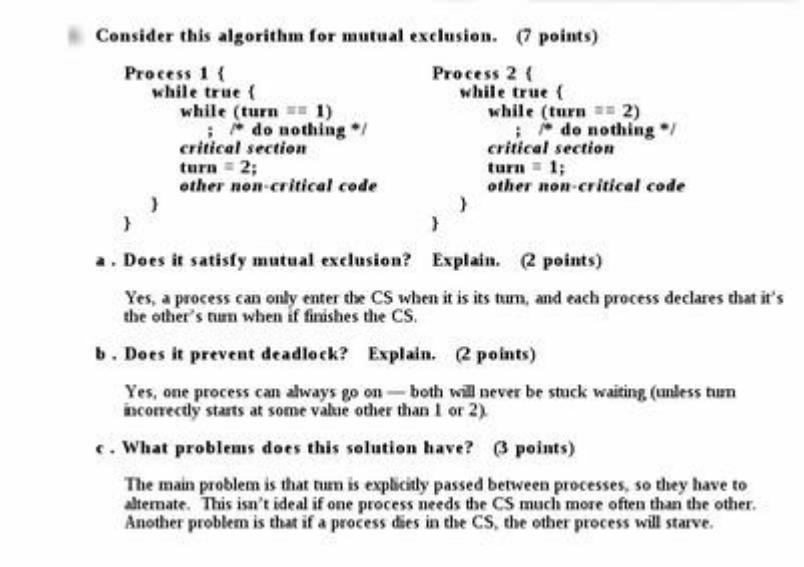
In non-preemptive scheduling, a running task is executed till completion. It cannot be interrupted.
Eg First In First Out.

Q#17 this was about two processes like, will they produce deadlock

```
process1(){
while true{
while(turn ==1); /* do nothing*/

turn = 2;
critical section
}
}
 Process2(){
while true{
while(turn == 2); /* do nothing */
 turn = 1;
critical section
}
}
```

Sol: no they donot produce deadlock:



Consider this algorithm for mutual exclusion. (7 points)

```
Process 1 {                          Process 2 {
   while true {                         while true {
      while (turn == 1)                    while (turn == 2)
         ; /* do nothing */                   ; /* do nothing */
      critical section                     critical section
      turn = 2;                            turn = 1;
      other non-critical code              other non-critical code
   }                                    }
}                                    }
```

a. Does it satisfy mutual exclusion? Explain. (2 points)

Yes, a process can only enter the CS when it is its turn, and each process declares that it's the other's turn when if finishes the CS.

b. Does it prevent deadlock? Explain. (2 points)

Yes, one process can always go on — both will never be stuck waiting (unless turn incorrectly starts at some value other than 1 or 2).

c. What problems does this solution have? (3 points)

The main problem is that turn is explicitly passed between processes, so they have to alternate. This isn't ideal if one process needs the CS much more often than the other. Another problem is that if a process dies in the CS, the other process will starve.

Q#13 Do you think there are some events in computer systems that causes CPU scheduling?

- CPU scheduling decisions take place under one of four conditions:
  1. When a process switches from the running state to the waiting state, such as for an I/O request or invocation of the wait( ) system call.
  2. When a process switches from the running state to the ready state, for example in response to an interrupt.
  3. When a process switches from the waiting state to the ready state, say at completion of I/O or a return from wait( ).
  4. When a process terminates.

Q2. Describe any three main objective of an operating system.

Answer:

•To provide an environment for a computer user to execute programs on computer hardware in a convenient and efficient manner.

•To allocate the separate resources of the computer as needed to solve the problem given. The allocation process should be as fair and efficient as possible.

•As a control program it serves two major functions: (1) supervision of the execution of user programs to prevent errors and improper use of the computer, and (2) management of the operation and control of I/O devices.

Q3. Features of real time operating system.5 marks

Answer RTOS is a real time operating system. The **important features** are :-

1. The necessary signaling functions between interrupt routines and task codes are handled by RTOS.

9. It works as an independent system with no internal or external interdependencies.
10. There are no loop decisions in RTOS
11. The RTOS can suspend one task code subroutine in the middle order to run another
12. The time lag is very less compared to other systems
13. There are no random time variables; this is good for a direct relationship between instruction and process.
14. Tasks are simpler to write.
15. Under most RTOS tasks are simply subroutines

Q4. linux implements sof real-time scheduling what features are missing to make it real time programming task? How might they be added to the kernel?

Sol:

Answer: Linux's "soft" real-time scheduling provides ordering guarantees concerning the priorities of runnable processes: real-time processes will always be given a higher priority by the scheduler than normal time-sharing processes, and a real-time process will never be interrupted by another process with a lower real-time priority.
However, the Linux kernel does not support "hard" real-time functionality. That is, when a process is executing a kernel service routine, that routine will always execute to completion unless it yields control back to the scheduler either explicitly or implicitly (by waiting for some asynchronous event). There is no support for preemptive scheduling of kernel-mode processes. As a result, any kernel system call that runs for a significant amount of time without rescheduling will block execution of any real-time processes.
Many real-time applications require such hard real-time scheduling. In particular, they often require guaranteed worst-case response times to external events. To achieve these guarantees, and to give user-mode real time processes a true higher priority than kernel-mode lower-priority processes, it is necessary to find a way to avoid having to wait for low-priority kernel calls to complete before scheduling a real-time process. For example, if a device driver generates an interrupt that wakes up a high-priority real-time process, then the kernel needs to be able to schedule that process as soon as possible, even if some other process is already executing in kernel mode.

Such preemptive rescheduling of kernel-mode routines comes at a cost. If the kernel cannot rely on non-preemption to ensure atomic updates of shared data structures, then reads of or updates to those structures must be protected by some other, finer-granularity locking mechanism. This fine-grained locking of kernel resources is the main requirement for provision of tight scheduling guarantees.
Many other kernel features could be added to support real-time programming. Deadline-based scheduling could be achieved by making modifications to the scheduler. Prioritization of IO operations could be implemented in the block-device IO request layer.

Q5. Is MVT (multiprograming with variable task) and MFT (multiprograming with fixed task). which one is best suited for internal fregmentation and which one is suited for external fregmentation. justify your answer?

SOL:

❖ **MULTIPROGRAMMING WITH FIX PARTITION(Contagious Memory Allocation):**

- To support the multiprogramming the main memory is divide into fix partitions.
- These partitions may be equal size or unequal size.

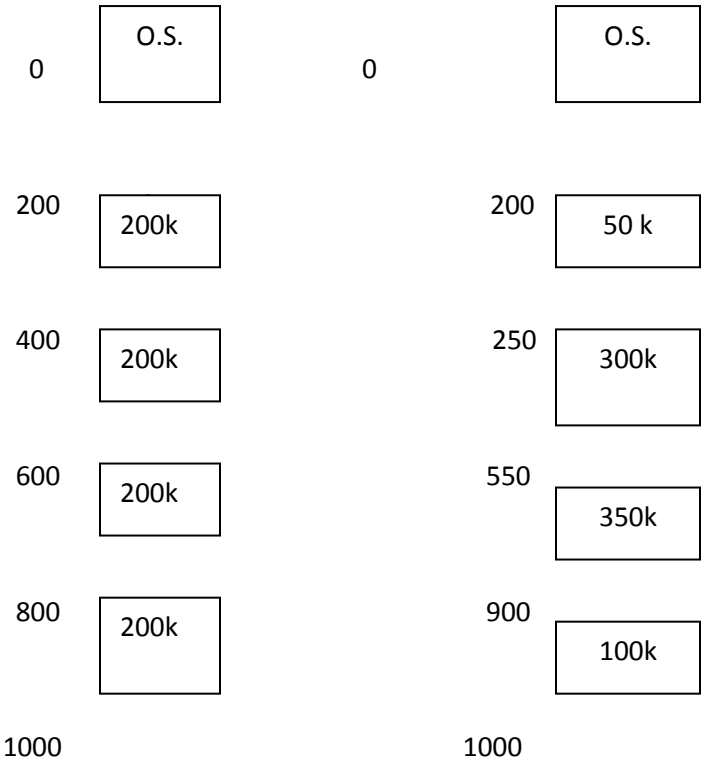


Fig.1[equal-size partition] Fig.2[unequal-size partitions]

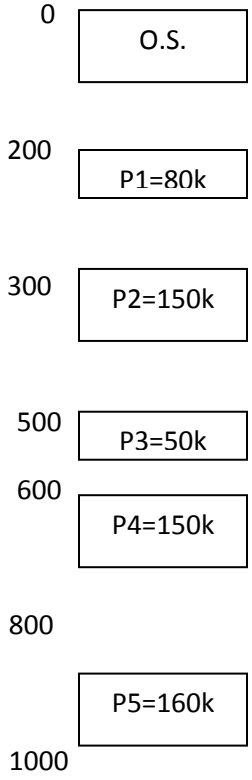
- In fig.1 memory is divided into 4 equal size partitions. Each partitions of 200k.

- In fig.2 memory is divide into 4 unequal size partitions. partition size is not equal.
- Operating system will load the processes into the partitions.
- Suppose main memory is divide into 8 partitions. Than 8 process can be loaded into memory. Only one process can enter into one partitions.

- There are two types of fragmentation:
  1. Internal Fregmentation

  2. External Fragmentation

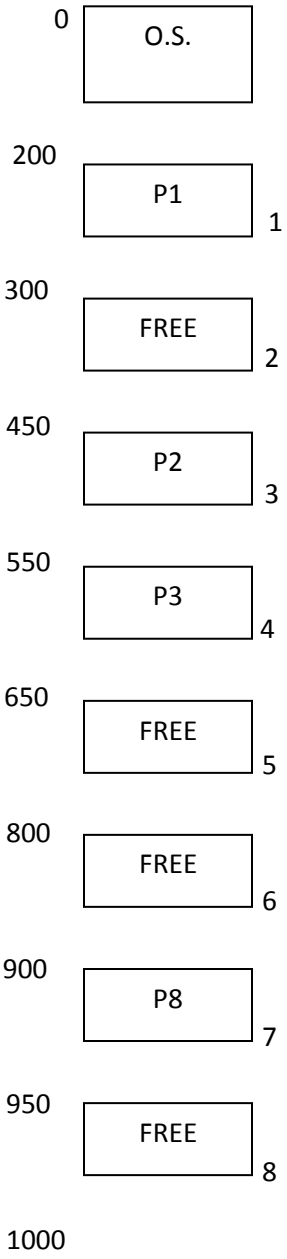
- **Internal Fregmentation(**MFT (multiprograming with fixed task))
- **Defination:** Difference between total space of partition and space occupied by process is called internal fregmentation.
- In fixed partitions internal fregmentation will be created.



- Here p1,p2,p3,p4,p5 is loaded into memory.
- Process P1 is loaded into first partition whose size is 100k and size of process P1 is 80k ,so in partition 20k space is waste. This is called Internal Fregmentation.
- For partition-2 size of partition is 200k and size of P2 is 150k , so here Internal fregmentation of 50k.
- When process will execute it will remove from memory. And new process will enter into memory in the free partition.
- Operating system keep information about the free and allocated partitions.

- There are three strategies are used to allocate memory into free partitions.

4. **First fit**
5. **Best fit**
6. **worst fit**



- Here total 8 partitions. Partition-2nd ,5th,6th and 8th are free and other partitions are allocated processes.
- Now to allocate new processes into these free partitions 3 algorithms are used.

First fit

Allocate first free partition which is enough for process. searching will be start from first partitions.

Best fit

Allocate the small free partition which is small and enough for process. Here all partition will be search from top to bottom and allocated the partition in which less amount of waste of space.

Worst Fit

Allocate the largest partition in which high amount of waste of space, here all partitions will be search from top to bottom.

- Best fit algorithms is most better compare to first fit and worst fit.
- First Fit Algorithm is faster compare to other.

Advantage & Disadvantage of Fixed Partition

Advantage : - simple to implement

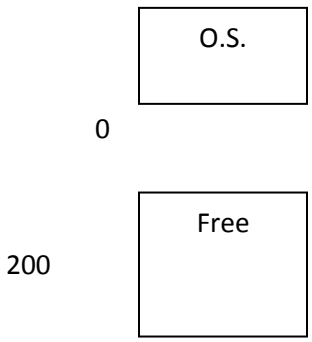
Disadvantage.: - Problem of internal fregmentation.

- Not efficient use of memory.

- **MULTIPROGRAMMING WITH DYNAMIC PARTITION** External Fragmentation **(variable size):-(MVT)**
  - To avoid the difficulties of fixed partitions, partitions are created dynamically.

  - In Dynamic partitions, no partition is fixed into memory but the partition are created only when process is load into memory, the partitions size is exactly size of process. If process size is 125k, when process will enter into memory than 125k size partition will be created.

  - If we have 5 process p1, p2,p3,p4,p5 and size of user process area is 800k which is currently free and no any partition.



1000

- Size of each process is :

  p1 – 80 k

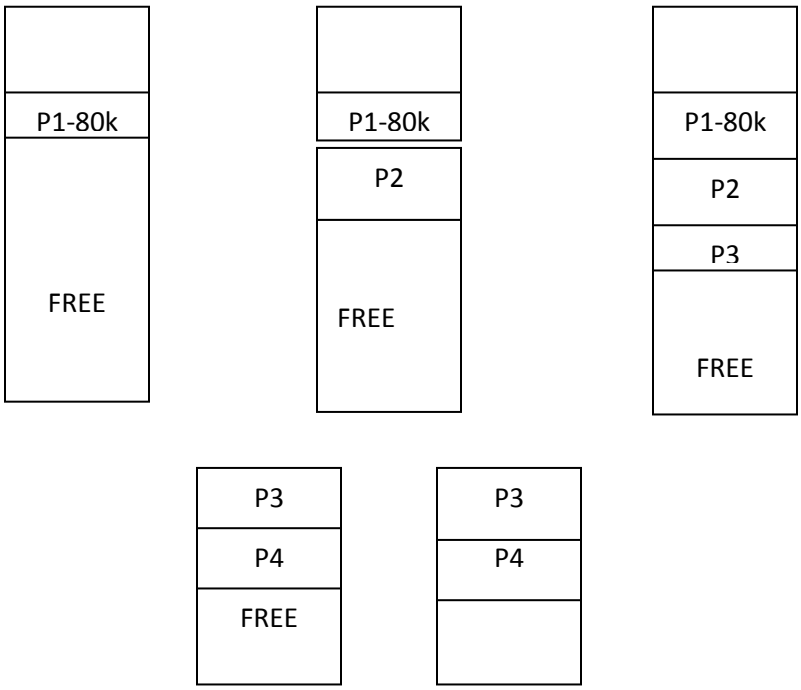
  p2 – 150k

  p3 – 170k

  p4 – 300k

  p5 – 40k

- when process will loaded than partition will be created.



External

Fregmentation

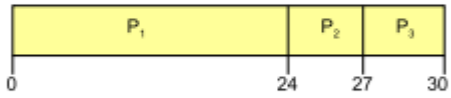
Here the not problem of internal fragmentation. But here the problem of external fregmentation.

- **External Fregmentation:**
- **Definition** : Difference between total space of memory and total memory allocated to processes is called **external fregmentation**.

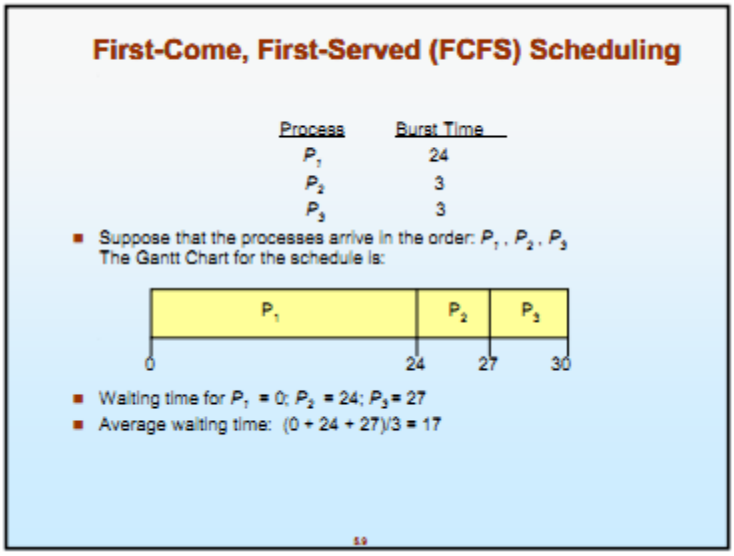
- when all process will be loaded than waste of space at end of memory is called external fregmentaion.

Example: FCFS Scheduling

- **Run jobs in order that they arrive**
  - Called *"First-come first-served"* (FCFS)
  - E.g., Say P_1 needs 24 sec, while P_2 and P_3 need 3.
  - Say P_2, P_3 arrived immediately after P_1, get:



- **Dirt simple to implement—how good is it?**
- **Throughput: 3 jobs / 30 sec = 0.1 jobs/sec**
- **Turnaround Time:** P_1 : 24, P_2 : 27, P_3 : 30
  - Average TT: $(24 + 27 + 30)/3 = 27$



Q7. Solution to the Critical Section Problem must meet three conditions...

4. **mutual exclusion**: if process p_i is executing in its critical section, no other process is executing in its critical section
5. **progress**: if no process is executing in its critical section and there exists some processes that wish to enter their critical sections, then only those processes that are not executing in their remainder section can participate in the decision of which will enter its critical section next, and this decision cannot be postponed indefinitely
    - if no process is in critical section, can decide quickly who enters
    - only one process can enter the critical section so in practice, others are put on the queue
6. **bounded waiting**: there must exist 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
    - The wait is the time from when a process makes a request to enter its critical section until that request is granted
    - in practice, once a process enters its critical section, it does not get another turn until a waiting process gets a turn (managed as a queue)

Q. Three processes are sharing 4 resources of same type in system. Every process need 2 resources show system is deadlock free.

solution

If the system is deadlocked, it implies that each process is holding one resource and is waiting for one more. Since there are 3 processes and 4 resources, one process must be able to obtain two resources. This process requires no more resources and therefore it will return its resources when done.

Q. 2 process-three resources are there? is there deaklock possible?

Sol:

Same above.

Q. block size of single word and double word

SOL

word is a collection of characters which is 16 bits in size.

Double words = 2 words or 32bits.