GumNaaM_Helpers

Virtual University

Mega File for MiD TerM

All Files in ONE FILE

(Handouts_waqar_moaz_junaid_vu toper_orange monkey)

Alhamdulillah GumNaaM_Helpers Bs..IT/CS ky Tamam Semesters ki tamam Books ki help krty hn.

Group Join krty hi description Read krna na bholyn

Semester Name	Group Link
Bs..IT/CS...smstr:1	https://chat.whatsapp.com/Ciya2X0DI0l8iQ14m0VNNj
Bs..IT/CS...smstr:2	https://chat.whatsapp.com/Kwh85sRK02z1Qk5B7aMR19
Bs..IT/CS...smstr:3	https://chat.whatsapp.com/IjIpdzi8pIA4lQVNUMmNYE
Bs..IT/CS...smstr:4	https://chat.whatsapp.com/CiGpHsSftir0oyhM47WfKh
Bs..IT/CS...smstr:5	https://chat.whatsapp.com/Hf7NWhdW06uCW7Mbt1n4Db
Bs..IT/CS...smstr:6	https://chat.whatsapp.com/DyAoT7VFkwy8DuvryDqm2L
Bs..IT/CS...smstr:7	https://chat.whatsapp.com/ENiyfIDRWatJFPbXCUGYNN
Bs..IT/CS...smstr:8	https://chat.whatsapp.com/Bnkk4n7utpdEuXazHLxRcA

Operating System
CS604

Delivered by

Dr. Syed Mansoor Sarwar

Virtual University of Pakistan

Knowledge beyond the boundaries

TABLE OF CONTENTS

Operating Systems
Lecture No. 1

Reading Material

- Operating Systems Concepts, Chapter 1
- PowerPoint Slides for Lecture 1

Summary

- Introduction and purpose of the course
- Organization of a computer system
- Purpose of a computer system
- Requirements for achieving the purpose – Setting the stage for OS concepts and principles
- Outline of topics to be discussed
- What is an Operating System?

Organization of a Computer System

As shown in Figure 1.1, the major high-level components of a computer system are:

1. **Hardware**, which provides basic computing resources (CPU, memory, I/O devices).
2. **Operating system**, which manages the use of the hardware among the various application programs for the various users and provides the user a relatively simple machine to use.
3. **Applications programs** that define the ways in which system resources are used to solve the computing problems of the users (compilers, database systems, video games, business programs).
4. **Users,** which include people, machines, other computers.

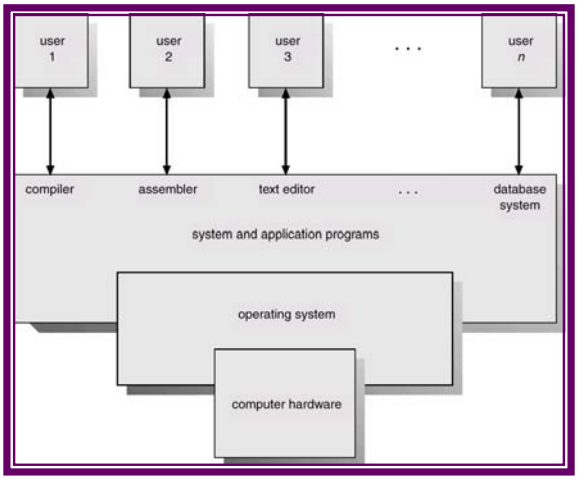


Figure 1.1. High-level components of a computer system

Purpose of a Computer—Setting the Stage for OS Concepts and Principles

Computer systems consist of software and hardware that are combined to provide a tool to implement solutions for specific problems in an efficient manner and to execute programs. Figure 1.2 shows the general organization of a contemporary computer system and how various system components are interconnected.

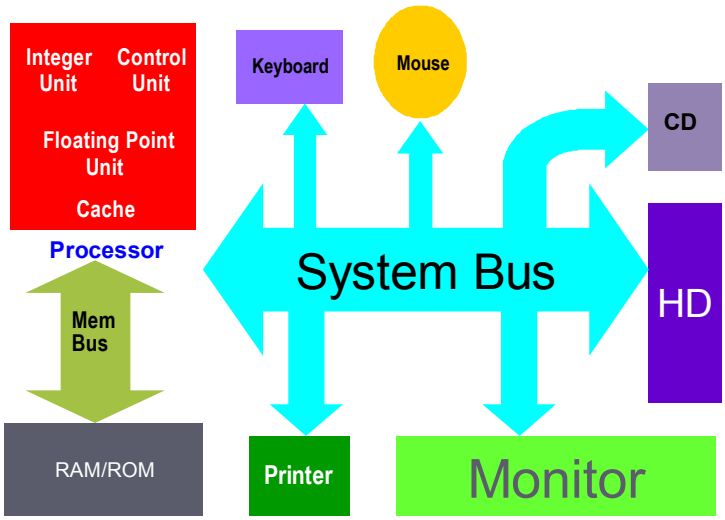


Figure 1.2. Organization of a Computer System

Viewing things closely will reveal that the primary purpose of a computer system is to generate executable programs and execute them. The following are some of the main issues involved in performing these tasks.

1. Storing an executable on a secondary storage device such as hard disk
2. Loading executable from disk into the main memory
3. Setting the CPU state appropriately so that program execution could begin
4. Creating multiple cooperating processes, synchronizing their access to shared data, and allowing them to communicate with each other

The above issues require the operating system to provide the following services and much more:

- Manage secondary storage devices
  - Allocate appropriate amount of disk space when files are created
  - Deallocate space when files are removing
  - Insure that a new file does not overwrite an existing file
  - Schedule disk requests
- Manage primary storage
  - Allocate appropriate amount of memory space when programs are to be loaded into the memory for executing
  - Deallocate space when processes terminate
  - Insure that a new process is not loaded on top of an existing process
  - Insure that a process does not access memory space that does not belong to it
  - Minimize the amount of unused memory space
  - Allow execution of programs larger in size than the available main memory
- Manage processes

- Allow simultaneous execution of processes by scheduling the CPU(s)
- Prevent deadlocks between processes
- Insure integrity of shared data
- Synchronize executions of cooperating processes

- Allow a user to manage his/her files and directories properly
  - User view of directory structure
  - Provide a mechanism that allows users to protect their files and directories

In this course, we will discuss in detail these operating system services (and more), with a particular emphasis on the UNIX and Linux operating systems. See the course outline for details of topics and lecture schedule.

What is an Operating System?

There are two views about this. The top-down view is that it is a program that acts as an intermediary between a user of a computer and the computer hardware, and makes the computer system convenient to use. It is because of the operating system that users of a computer system don't have to deal with computer's hardware to get their work done. Users can use simple commands to perform various tasks and let the operating system do the difficult work of interacting with computer hardware. Thus, you can use a command like `copy file1 file2` to copy 'file1' to 'file2' and let the operating system communicate with the controller(s) of the disk that contain(s) the two files.

A computer system has many hardware and software resources that may be required to solve a problem: CPU time, memory space, file storage space, I/O devices etc. The operating system acts as the manager of these resources, facing numerous and possibly conflicting requests for resources, the operating system must decide how (and when) to allocate (and deallocate) them to specific programs and users so that it can operate the computer system efficiently, fairly, and securely. So, the bottom-up view is that operating system is a resource manager who manages the hardware and software resources in the computer system.

A slightly different view of an operating system emphasizes the need to control the various I/O devices and programs. An operating system is a control program that manages the execution of user programs to prevent errors and improper use of a computer.

Operating Systems
Lecture No. 2

Reading Material

- Operating Systems Concepts, Chapter 1
- PowerPoint Slides for Lecture 2

Summary

- Single-user systems
- Batch systems
- Multi programmed systems
- Time-sharing systems
- Real time systems
- Interrupts, traps and software interrupts (UNIX signals)
- Hardware protection

Single-user systems

A computer system that allows only one user to use the computer at a given time is known as a **single-user system**. The goals of such systems are maximizing user convenience and responsiveness, instead of maximizing the utilization of the CPU and peripheral devices. Single-user systems use I/O devices such as keyboards, mice, display screens, scanners, and small printers. They can adopt technology developed for larger operating systems. Often individuals have sole use of computer and do not need advanced CPU utilization and hardware protection features. They may run different types of operating systems, including DOS, Windows, and MacOS. Linux and UNIX operating systems can also be run in single-user mode.

Batch Systems

Early computers were large machines run from a console with card readers and tape drives as input devices and line printers, tape drives, and card punches as output devices. The user did not interact directly with the system; instead the user prepared a job, (which consisted of the program, data, and some control information about the nature of the job in the form of control cards) and submitted this to the computer operator. The job was in the form of punch cards, and at some later time the output was generated by the system—user didn't get to interact with his/her job. The output consisted of the result of the program, as well as a dump of the final memory and register contents for debugging.

To speed up processing, operators batched together jobs with similar needs, and ran them through the computer as a group. For example, all FORTRAN programs were complied one after the other. The major task of such an operating system was to transfer control automatically from one job to the next. In this execution environment, the CPU is often idle because the speeds of the mechanical I/O devices such as a tape drive are slower than that of electronic devices. Such systems in which the user does not get to

interact with his/her jobs and jobs with similar needs are executed in a "batch", one after the other, are known as **batch systems**. Digital Equipment Corporation's VMS is an example of a batch operating system.

Figure 2.1 shows the memory layout of a typical computer system, with the **system space** containing operating system code and data currently in use and the **user space** containing user programs (processes). In case of a batch system, the user space contains one process at a time because only one process is executing at a given time.

Figure 2.1 Memory partitioned into user and system spaces

Multi-programmed Systems

Multi-programming increases CPU utilization by organizing jobs so that the CPU always has one to execute. The operating system keeps several jobs in memory simultaneously, as shown in Figure 2.2. This set of jobs is a subset of the jobs on the disk which are ready to run but cannot be loaded into memory due to lack of space. Since the number of jobs that can be kept simultaneously in memory is usually much smaller than the number of jobs that can be in the job pool; the operating system picks and executes one of the jobs in the memory. Eventually the job has to wait for some task such as an I/O operation to complete. In a non multi-programmed system, the CPU would sit idle. In a multi-programmed system, the operating system simply switches to, and executes another job. When that job needs to wait, the CPU simply switches to another job and so on.

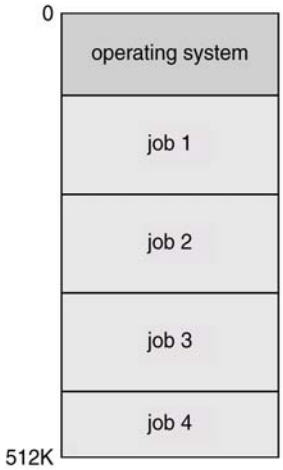


Figure 2.2 Memory layout for a multi-programmed batch system

Figure 2.3 illustrates the concept of multiprogramming by using an example system with two processes, P1 and P2. The CPU is switched from P1 to P2 when P1 finishes its CPU burst and needs to wait for an event, and vice versa when P2 finishes it CPU burst and has to wait for an event. This means that when one process is using the CPU, the other is waiting for an event (such as I/O to complete). This increases the utilization of the CPU and I/O devices as well as throughput of the system. In our example below, P1 and P2 would finish their execution in 10 time units if no multiprogramming is used and in six time units if multiprogramming is used.

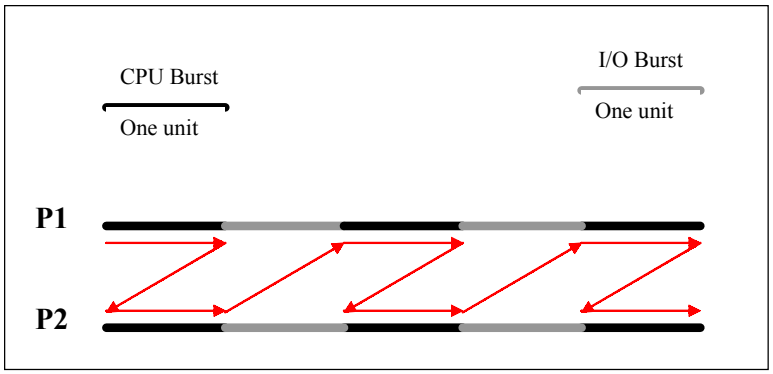


Figure 2.3 Illustration of the multiprogramming concept

All jobs that enter the system are kept in the **job pool**. This pool consists of all processes residing on disk awaiting allocation of main memory. If several jobs are ready to be brought into memory, and there is not enough room for all of them, then the system must choose among them. This decision is called *job scheduling*. In addition if several jobs are ready to run at the same time, the system must choose among them. We will discuss CPU scheduling in Chapter 6.

Time-sharing systems

A **time-sharing system** is multi-user, multi-process, and interactive system. This means that it allows multiple users to use the computer simultaneously. A user can run one or more processes at the same time and interact with his/her processes. A time-shared system uses multiprogramming and CPU scheduling to provide each user with a small portion of a time-shared computer. Each user has at least one separate program in memory. To obtain a reasonable response time, jobs may have to be swapped in and out of main memory. UNIX, Linux, Widows NT server, and Windows 2000 server are time-sharing systems. We will discuss various elements of time-sharing systems throughout the course.

Real time systems

Real time systems are used when rigid time requirements are placed on the operation of a processor or the flow of data; thus it is often used as a control device in a dedicated application. Examples are systems that control scientific experiments, medical imaging systems, industrial control systems and certain display systems.

A **real time system** has well defined, fixed time constraints, and if the system does not produce output for an input within the time constraints, the system will fail. For instance, it would not do for a robot arm to be instructed to halt after it had smashed into the car it was building.

Real time systems come in two flavors: hard and soft. A **hard real time system** guarantees that critical tasks be completed on time. This goal requires that all delays in the system be completed on time. This goal requires that all delays in the system be bounded, from the retrieval of stored data to the time it takes the operating system to finish any request made of it. Secondary storage of any sort is usually limited or missing, with data instead being stored in short-term memory or in read only memory. Most advanced operating system features are absent too, since they tend to separate the user from the hardware, and that separation results in uncertainty about the amount of time an operation will take.

A less restrictive type of real time system is a **soft real time system**, where a critical real-time task gets priority over other tasks, and retains that priority until it completes. As in hard real time systems, the operating system kernel delays need to be bounded. Soft real time is an achievable goal that can be mixed with other types of systems, whereas hard real time systems conflict with the operation of other systems such as time-sharing systems, and the two cannot be mixed.

Interrupts, traps and software interrupts

An **interrupt** is a signal generated by a hardware device (usually an I/O device) to get CPU's attention. Interrupt transfers control to the **interrupt service routine** (ISR), generally through the **interrupt vector table**, which contains the addresses of all the service routines. The interrupt service routine executes; on completion the CPU resumes the interrupted computation. Interrupt architecture must save the address of the interrupted instruction. Incoming interrupts are disabled while another interrupt is being processed to prevent a *lost interrupt*. An operating system is an interrupt driven software.

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

A **signal** is an event generated to get attention of a process. An example of a signal is the event that is generated when you run a program and then press `<Ctrl-C>`. The signal generated in this case is called SIGINT (Interrupt signal). Three actions are possible on a signal:

1. Kernel-defined default action—which usually results in process termination and, in some cases, generation of a 'core' file that can be used the programmer/user to know the state of the process at the time of its termination.
2. Process can intercept the signal and ignore it.
3. Process can intercept the signal and take a programmer-defined action.

We will discuss signals in detail in some of the subsequent lectures.

Hardware Protection

Multi-programming put several programs in memory at the same time; while this increased system utilization it also increased problems. With sharing, many processes

could be adversely affected by a bug in one program. One erroneous program could also modify the program or data of another program or even the resident part of the operating system. A file may overwrite another file or folder on disk. A process may get the CPU and never relinquish it. So the issues of hardware protection are: I/O protection, memory protection, and CPU protection. We will discuss them one by one, but first we talk about the dual-mode operation of a CPU.

a) Dual Mode Operation

To ensure proper operation, we must protect the operating system and all other programs and their data from any malfunctioning program. Protection is needed for any shared resources. Instruction set of a modern CPU has two kinds of instructions, privileged instructions and non-privileged instructions. Privileged instructions can be used to perform hardware operations that a normal user process should not be able to perform, such as communicating with I/O devices. If a user process tries to execute a privileged instruction, a trap should be generated and process should be terminated prematurely. At the same time, a piece of operating system code should be allowed to execute privileged instructions. In order for the CPU to be able to differentiate between a user process and an operating system code, we need two separate modes of operation: user mode and monitor mode (also called supervisor mode, system mode, or privileged mode). A bit, called the **mode bit**, is added to the hardware of the computer to indicate the current mode: **monitor mode** (0) or **user mode** (1). With the mode bit we are able to distinguish between a task that is executed on behalf of the operating system and one that is executed on behalf of the user.

The concept of privileged instructions also provides us with the means for the user to interact with the operating system by asking it to perform some designated tasks that only the operating system should do. A user process can request the operating system to perform such tasks for it by executing a **system call**. Whenever a system call is made or an interrupt, trap, or signal is generated, CPU mode is switched to system mode before the relevant kernel code executes. The CPU mode is switched back to user mode before the control is transferred back to the user process. This is illustrated by the diagram in Figure 2.4.

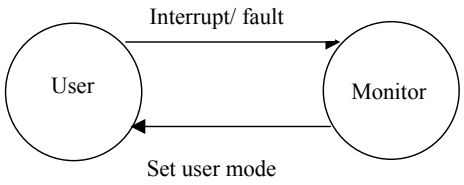


Figure 2.4 The dual-mode operation of the CPU

b) I/O Protection

A user process may disrupt the normal operation of the system by issuing illegal I/O instructions, by accessing memory locations within the operating system itself, or by

refusing to relinquish the CPU. We can use various mechanisms to ensure that such disruptions cannot take place in the system.

To prevent users from performing illegal I/O, we define all I/O instructions to be privileged instructions. Thus users cannot issue I/O instructions directly; they must do it through the operating system. For I/O protection to be complete, we must be sure that a user program can never gain control of the computer in monitor mode. If it could, I/O protection could be compromised.

Consider a computer executing in user mode. It will switch to monitor mode whenever an interrupt or trap occurs, jumping to the address determined from the interrupt from the interrupt vector. If a user program, as part of its execution, stores a new address in the interrupt vector, this new address could overwrite the previous address with an address in the user program. Then, when a corresponding trap or interrupt occurred, the hardware would switch to monitor mode and transfer control through the modified interrupt vector table to a user program, causing it to gain control of the computer in monitor mode. Hence we need all I/O instructions and instructions for changing the contents of the system space in memory to be protected. A user process could request a privileged operation by executing a system call such as read (for reading a file).

Operating Systems
Lecture No. 3

Reading Material

- Computer System Structures, Chapter 2
- Operating Systems Structures, Chapter 3
- PowerPoint Slides for Lecture 3

Summary

- Memory and CPU protection
- Operating system components and services
- System calls
- Operating system structures

Memory Protection

The region in the memory that a process is allowed to access is known as **process address space**. To ensure correct operation of a computer system, we need to ensure that a process cannot access memory outside its address space. If we don't do this then a process may, accidentally or deliberately, overwrite the address space of another process or memory space belonging to the operating system (e.g., for the interrupt vector table).

Using two CPU registers, specifically designed for this purpose, can provide memory protection. These registered are:

- Base register – it holds the smallest legal physical memory address for a process
- Limit register – it contains the size of the process

When a process is loaded into memory, the base register is initialized with the starting address of the process and the limit register is initialized with its size. Memory outside the defined range is protected because the CPU checks that every address generated by the process falls within the memory range defined by the values stored in the base and limit registers, as shown in Figure 3.1.

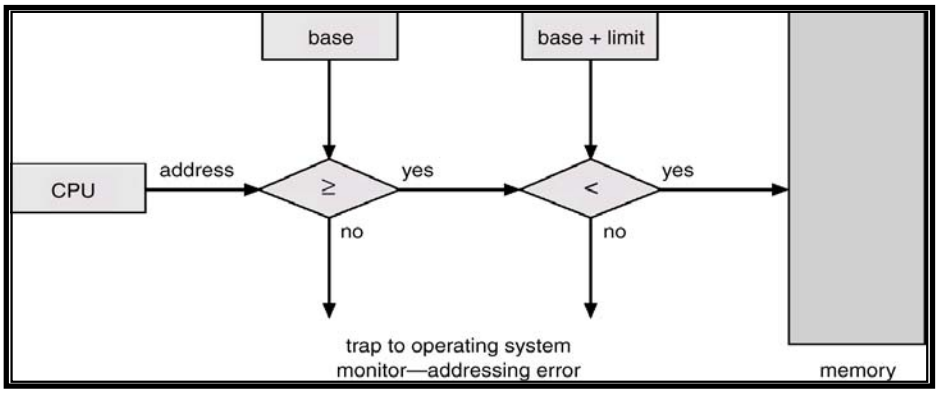


Figure 3.1 Hardware address protection with base and limit registers

In Figure 3.2, we use an example to illustrate how the concept outlined above works. The base and limit registers are initialized to define the address space of a process. The process starts at memory location 300040 and its size is 120900 bytes (assuming that memory is byte addressable). During the execution of this process, the CPU insures (by using the logic outlined in Figure 3.1) that all the addresses generated by this process are greater than or equal to 300040 and less than (300040+120900), thereby preventing this process to access any memory area outside its address space. Loading the base and limit registers are privileged instructions.

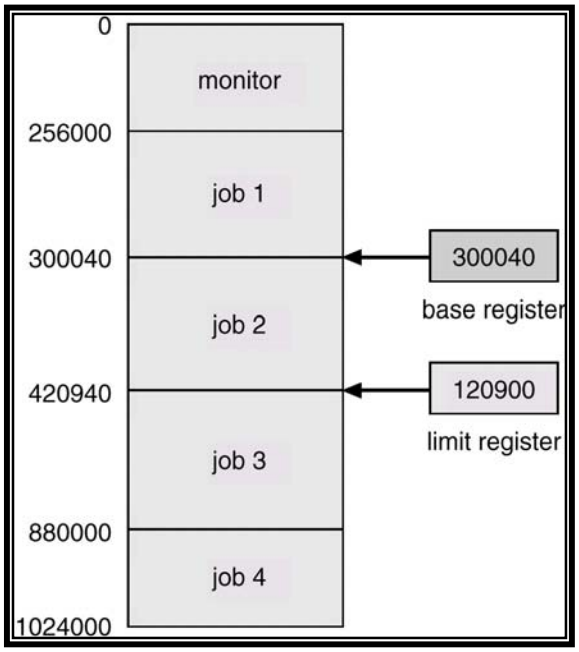


Figure 3.2 Use of Base and Limit Register

CPU Protection

In addition to protecting I/O and memory, we must ensure that the operating system maintains control. We must prevent the user program from getting stuck in an infinite loop or not calling system services and never returning control to the CPU. To accomplish this we can use a **timer**, which interrupts the CPU after specified period to ensure that the operating system maintains control. The timer period may be variable or fixed. A *fixed-rate clock* and a *counter* are used to implement a variable timer. The OS initializes the counter with a positive value. The counter is decremented every clock tick by the clock interrupt service routine. When the counter reaches the value 0, a timer interrupt is generated that transfers control from the current process to the next scheduled process. Thus we can use the timer to prevent a program from running too long. In the most straight forward case, the timer could be set to interrupt every N milliseconds, where N is the **time slice** that each process is allowed to execute before the next process gets control of the CPU. The OS is invoked at the end of each time slice to perform various housekeeping tasks. This issue is discussed in detail under CPU scheduling in Chapter 7.

Another use of the timer is to compute the current time. A timer interrupt signals the passage of some period, allowing the OS to compute the current time in reference to some initial time. Load-timer is a privileged instruction.

OS Components

An operating system has many components that manage all the resources in a computer system, insuring proper execution of programs. We briefly describe these components in this section.

- Process management

A process can be thought of as a program in execution. It needs certain resources, including CPU time, memory, files and I/O devices to accomplish its tasks. The operating system is responsible for:

- Creating and terminating both user and system processes
- Suspending and resuming processes
- Providing mechanisms for process synchronization
- Providing mechanisms for process communication
- Providing mechanisms for deadlock handling

- Main memory management

Main memory is a large array of words or bytes (called memory locations), ranging in size from hundreds of thousands to billions. Every word or byte has its own address. Main memory is a repository of quickly accessible data shared by the CPU and I/O devices. It contains the code, data, stack, and other parts of a process. The central processor reads instructions of a process from main memory during the machine cycle—fetch-decode-execute.

The OS is responsible for the following activities in connection with memory management:

- Keeping track of free memory space
- Keeping track of which parts of memory are currently being used and by whom
- Deciding which processes are to be loaded into memory when memory space becomes available
- Deciding how much memory is to be allocated to a process
- Allocating and deallocating memory space as needed
- Insuring that a process is not overwritten on top of another

- Secondary storage management

The main purpose of a computer system is to execute programs. The programs, along with the data they access, must be in the main memory or **primary storage** during their execution. Since main memory is too small to accommodate all data and programs, and because the data it holds are lost when the power is lost, the computer system must provide **secondary storage** to backup main memory. Most programs are stored on a disk until loaded into the memory and then use disk as both the source and destination of their processing. Like all other resources in a computer system, proper management of disk storage is important.

The operating system is responsible for the following activities in connection with disk management:

- Free-space management

- Storage allocation and deallocation
- Disk scheduling

■ I/O system management

The I/O subsystem consists of:

- A memory management component that includes buffering, caching and spooling
- A general device-driver interface
- Drivers for specific hardware devices

■ File management

Computers can store information on several types of physical media, e.g. magnetic tape, magnetic disk and optical disk. The OS maps files onto physical media and accesses these media via the storage devices.

The OS is responsible for the following activities with respect to file management:

- Creating and deleting files
- Creating and deleting directories
- Supporting primitives (operations) for manipulating files and directories
- Mapping files onto the secondary storage
- Backing up files on stable (nonvolatile) storage media

■ Protection system

If a computer system has multiple users and allows concurrent execution of multiple processes then the various processes must be protected from each other's activities.

Protection is any mechanism for controlling the access of programs, processes or users to the resources defined by a computer system.

■ Networking

A **distributed system** is a collection of processors that do not share memory, peripheral devices or a clock. Instead, each processor has it own local memory and clock, and the processors communicate with each other through various communication lines, such as high- speed buses or networks.

The processors in a communication system are connected through a **communication network**. The communication network design must consider message routing and connection strategies and the problems of contention and security.

A distributed system collects physically separate, possibly heterogeneous, systems into a single coherent system, providing the user with access to the various resources that the system maintains.

■ Command-line interpreter (shells)

One of the most important system programs for an operating system is the **command interpreter**, which is the interface between the user and operating system. Its purpose is to read user commands and try to execute them. Some operating systems include the command interpreter in the kernel. Other operating systems (e.g. UNIX, Linux, and DOS) treat it as a special program that runs when a job is initiated or when a user first logs on (on time sharing systems). This program is sometimes called the **command-line interpreter** and is often known as the **shell**. Its function is simple: to get the next command statement and execute it. Some of the famous shells for UNIX and Linux are

Bourne shell (sh), C shell (csh), Bourne Again shell (bash), TC shell (tcsh), and Korn shell (ksh). You can use any of these shells by running the corresponding command, listed in parentheses for each shell. So, you can run the Bourne Again shell by running the `bash` or `/usr/bin/bash` command.

Operating System Services

An operating system provides the environment within which programs are executed. It provides certain services to programs and users of those programs, which vary from operating system to operating system. Some of the common ones are:

- **Program execution:** The system must be able to load a program into memory and to run that programs. The program must be able to end its execution.
- **I/O Operations:** A running program may require I/O, which may involve a file or an I/O device. For efficiency and protection user usually cannot control I/O devices directly. The OS provides a means to do I/O.
- **File System Manipulation:** Programs need to read, write files. Also they should be able to create and delete files by name.
- **Communications:** There are cases in which one program needs to exchange information with another process. This can occur between processes that are executing on the same computer or between processes that are executing on different computer systems tied together by a computer network. Communication may be implemented via shared **memory** or **message passing**.
- **Error detection:** The OS constantly needs to be aware of possible errors. Error may occur in the CPU and memory hardware, in I/O devices and in the user program. For each type of error, the OS should take appropriate action to ensure correct and consistent computing.

  In order to assist the efficient operation of the system itself, the system provides the following functions:

- **Resource allocation:** When multiple users are logged on the system or multiple jobs are running at the same time, resources must be allocated to each of them. There are various routines to schedule jobs, allocate plotters, modems and other peripheral devices.
- **Accounting:** We want to keep track of which users use how many and which kinds of computer resources. This record keeping may be used for accounting or simply for accumulating usage statistics.
- **Protection:** The owners of information stored in a multi user computer system may want to control use of that information. When several disjointed processes execute concurrently it should not b possible for one process to interfere with the others or with the operating system itself. Protection involves ensuring that all access to system resources is controlled.

Entry Points into Kernel

As shown in Figure 3.3, there are four events that cause execution of a piece of code in the kernel. These events are: interrupt, trap, system call, and signal. In case of all of these events, some kernel code is executed to service the corresponding event. You have

discussed interrupts and traps in the computer organization or computer architecture course. We will discuss system calls execution in this lecture and signals subsequent lectures. We will talk about many UNIX and Linux system calls and signals throughout the course.

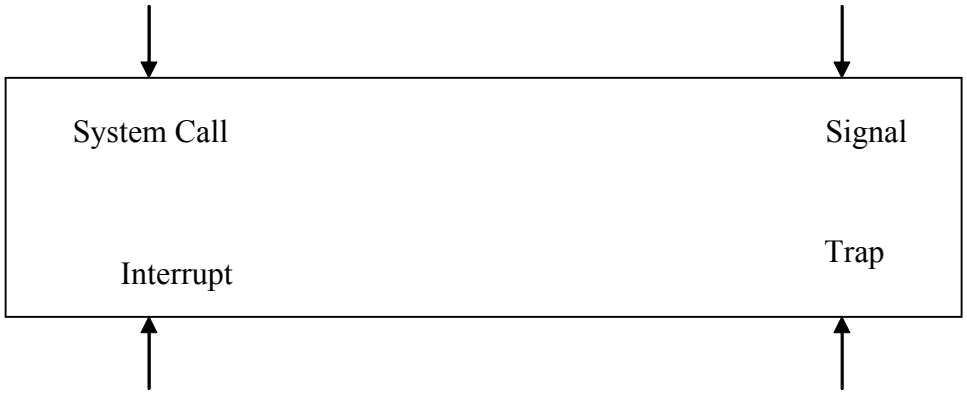


Figure 3.3 Entry points into the operating system kernel

System Calls

System calls provide the interface between a process and the OS. These calls are generally available as assembly language instructions. The system call interface layer contains entry point in the kernel code; because all system resources are managed by the kernel any user or application request that involves access to any system resource must be handled by the kernel code, but user process must not be given open access to the kernel code for security reasons. So that user processes can invoke the execution of kernel code, several openings into the kernel code, also called *system calls*, are provided. System calls allow processes and users to manipulate system resources such as files and processes.

System calls can be categorized into the following groups:

- Process Control
- File Management
- Device Management
- Information maintenance
- Communications

Semantics of System Call Execution

The following sequence of events takes place when a process invokes a system call:

- The user process makes a call to a library function
- The library routine puts appropriate parameters at a well-known place, like a register or on the stack. These parameters include arguments for the system call, return address, and call number. Three general methods are used to pass parameters between a running program and the operating system.
  - Pass parameters in *registers*.
  - Store the parameters in a table in the main memory and the table address is passed as a parameter in a register.
  - *Push* (store) the parameters onto the *stack* by the program, and *pop* off the stack by operating system.

- A `trap` instruction is executed to change mode from user to kernel and give control to operating system.
- The operating system then determines which system call is to be carried out by examining one of the parameters (the call number) passed to it by library routine.
- The kernel uses call number to index a kernel table (the **dispatch table)** which contains pointers to service routines for all system calls.
- The service routine is executed and control given back to user program via return from trap instruction; the instruction also changes mode from system to user.
- The library function executes the instruction following trap; interprets the return values from the kernel and returns to the user process.

Figure 3.4 gives a pictorial view of the above steps.

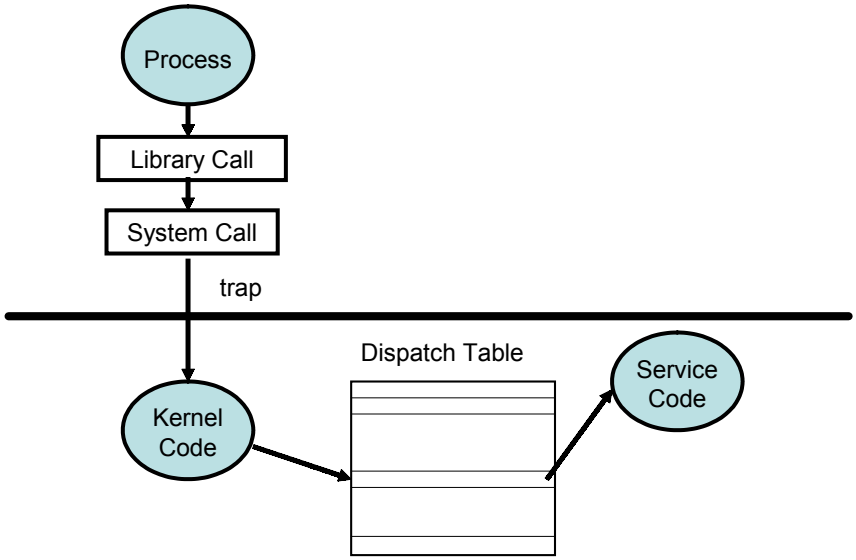


Figure 3.4 Pictorial view of the steps needed for execution of a system call

Operating Systems Structures

Just like any other software, the operating system code can be structured in different ways. The following are some of the commonly used structures.

- Simple/Monolithic Structure

In this case, the operating system code has not structure. It is written for functionality and efficiency (in terms of time and space). DOS and UNIX are examples of such systems, as shown in Figures 3.5 and 3.6. UNIX consists of two separable parts, the kernel and the system programs. The kernel is further separated into a series of interfaces and devices drivers, which were added and expanded over the years. Every thing below the system call interface and above the physical hardware is the kernel, which provides the file system, CPU scheduling, memory management and other OS functions through system calls. Since this is an enormous amount of functionality combined in one level, UNIX is difficult to enhance as changes in one section could adversely affect other areas. We will discuss the various components of the UNIX kernel throughout the course.

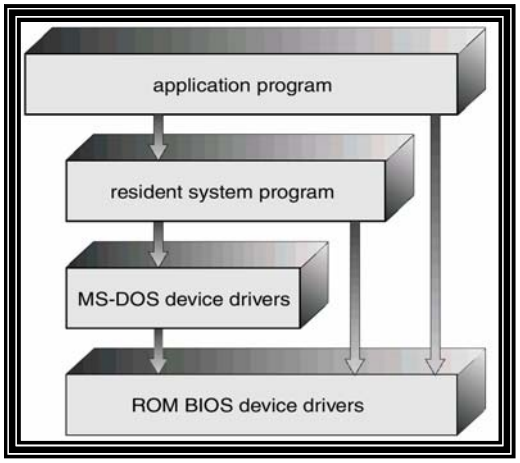


Figure 3.5 Logical structure of DOS

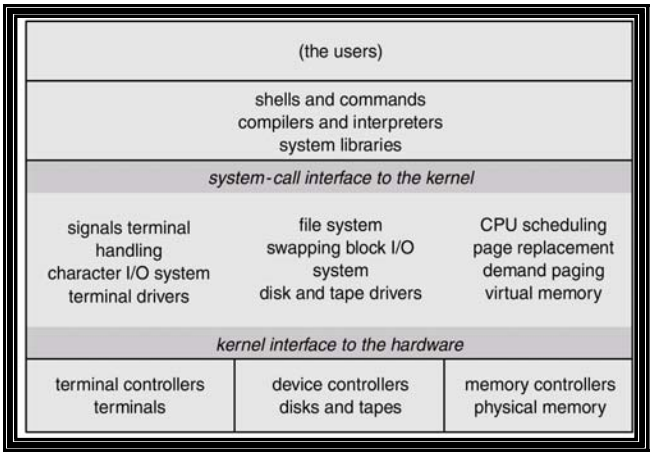


Figure 3.6 Logical structure of UNIX

Operating Systems
Lecture No. 4

Reading Material

- Operating Systems Structures, Chapter 3
- PowerPoint Slides for Lecture 3

Summary

- Operating system structures
- Operating system design and implementation
- UNIX/Linux directory structure
- Browsing UNIX/Linux directory structure

Operating Systems Structures (continued)

■ Layered Approach

The modularization of a system can be done in many ways. As shown in Figure 4.1, in the layered approach the OS is broken up into a number of layers or levels each built on top of lower layer. The bottom layer is the hardware; the highest layer (layer N) is the user interface. A typical OS layer (layer-M) consists of data structures and a set of routines that can be invoked by higher-level layers. Layer M in turn can invoke operations on lower level layers.

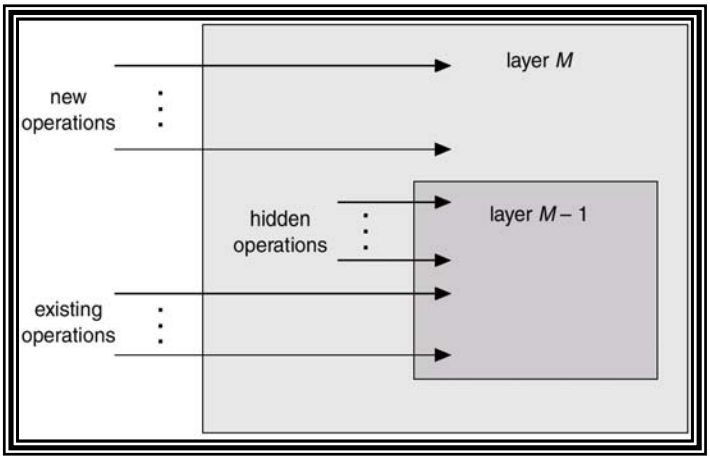


Figure 4.1 The layered structure

The main advantage of the layered approach is modularity. The layers are selected such that each uses functions and services of only lower layers. This approach simplifies debugging and system verification.

The major difficulty with layered approach is careful definition of layers, because a layer can only use the layers below it. Also it tends to be less efficient than other approaches. Each layer adds overhead to a system call (which is trapped when the

program executes a I/O operation, for instance). This results in a system call that takes longer than does one on a non-layered system. THE operating system by Dijkstra and IBM's OS/2 are examples of layered operating systems.

■ Micro kernels

This method structures the operating system by removing all non-essential components from the kernel and implementing as system and user level programs. The result is a smaller kernel. Micro kernels typically provide minimum process and memory management in addition to a communication facility. The main function of the micro kernel is to provide a communication facility between the client program and the various services that are also running in the user space.

The benefits of the micro kernel approach include the ease of extending the OS. All new services are added to user space and consequently do not require modification of the kernel. When the kernel does have to be modified, the changes tend to be fewer because the micro kernel is a smaller kernel. The resulting OS is easier to port from one hard ware design to another. It also provides more security and reliability since most services are running as user rather than kernel processes. Mach, MacOS X Server, QNX, OS/2, and Windows NT are examples of microkernel based operating systems. As shown in Figure 4.2, various types of services can be run on top of the Windows NT microkernel, thereby allowing applications developed for different platforms to run under Windows NT.

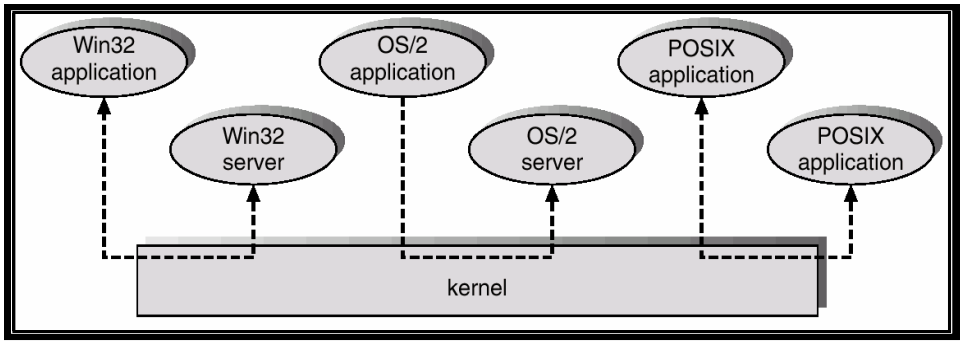


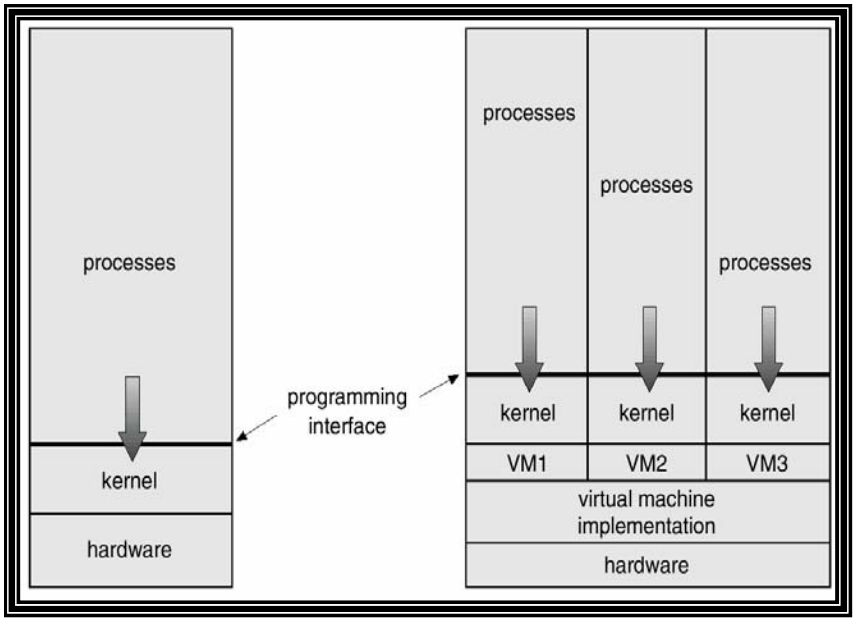
Figure 4.2 Windows NT client-server structure

■ Virtual Machines

Conceptually a computer system is made up of layers. The hardware is the lowest level in all such systems. The kernel running at the next level uses the hardware instructions to create a set of system call for use by outer layers. The system programs above the kernel are therefore able to use either system calls or hardware instructions and in some ways these programs do not differentiate between these two. System programs in turn treat the hardware and the system calls as though they were both at the same level. In some systems the application programs can call the system programs. The application programs view everything under them in the hierarchy as though the latter were part of the machine itself. This layered approach is taken to its logical conclusion in the concept of a **virtual machine** (VM). The VM operating system for IBM systems is the best example of VM concept.

By using CPU scheduling and virtual memory techniques an operating system can create the illusion that a process has its own memory with its own (virtual) memory. The

virtual machine approach on the other hand does not provide any additional functionality but rather provides an interface that is identical to the underlying bare hardware. Each process is provided with a virtual copy of the underlying computer. The physical computer shares resources to create the virtual machines. Figure 4.3 illustrates the concepts of virtual machines by a diagram.



Non Virtual Machine Virtual Machine

Figure 4.3 Illustration of virtual and non-virtual machines

Although the virtual machine concept is useful it is difficult to implement. There are two primary advantages to using virtual machines: first by completely protecting system resources the virtual machine provides a robust level of security. Second the virtual machine allows system development to be done without disrupting normal system operation.

Java Virtual Machine (**JVM**) loads, verifies, and executes programs that have been translated into Java Bytecode, as shown in Figure 4.4. **VMWare** can be run on a Windows platform to create a virtual machine on which you can install an operating of your choice, such as Linux. We have shown a couple of snapshots of VMWare on a Windows platform in the lecture slides. **Virtual PC** software works in a similar fashion.

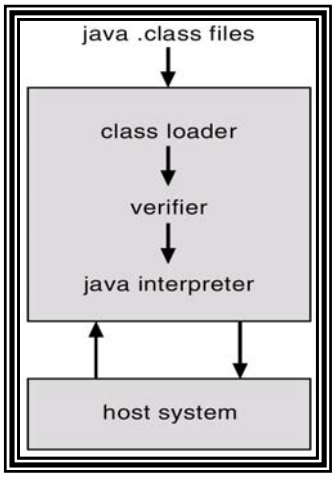


Figure 4.4 Java Virtual Machine

System Design and Implementation

- Design Goals

At the highest level, the deign of the system will be affected by the choice of hardware and type of system: batch , time shared, single user, multi user, distributed , real time or general purpose. Beyond this highest level, the requirements may be much harder to specify. The requirements can be divided into much two basic groups: *user goal* and *system goals*. Users desire a system that is easy to use, reliable, safe and fast. People who design, implement and operate the system, require a system that is easy to design, implement and maintain. An important design goal is separation of mechanisms and policies.

- **Mechanism:** they determine how to do something. A general mechanism is more desirable. Example: CPU protection.
- **Policy:** determine what will be done. Example: Initial value in the counter used for CPU protection.

The separation of policy and mechanism is important for flexibility, as policies are likely to change across places or over time. For example, the system administrator can set the initial value in counter before booting a system.

- Implementation

Once an operating system is designed, it must be implemented. Traditionally operating systems have been written in assembly language. Now however they are written in higher-level languages such as C/ C++ since these allow the code to be written faster, more compact, easier to understand and easier to port.

UNIX/LINUX Directory Structure

Dennis Ritchie and Ken Thomsom wrote UNIX at the Bell Labs in 1969. It was initially written in assembly language and a high-level language called Bit was later converted from B to C language. Linus Torvalds, an undergraduate student at the University of

Helsinki, Finland, wrote Linux in 1991. It is one of the most popular operating systems, certainly for PCs.

UNIX has a **hierarchical file system structure** consisting of a **root directory** (denoted as /) with other directories and files hanging under it. Unix uses a directory hierarchy that is commonly represented as folders. However, instead of using graphical folders typed commands (in a command line user interface) are used to navigate the system. Particular files are then represented by paths and filenames much like they are in html addresses. A pathname is the list of directories separated by slashes (/). If a pathname starts with a /, it refers to the root directory. The last component of a path may be a file or a directory. A pathname may simply be a file or directory name. For example, /usr/include/sys/param.h, ~/courses/cs604, and prog1.c are pathnames.

When you log in, the system places you in a directory called your **home directory** (also called **login directory**). You can refer to your home directory by using the ~ or $PATH in Bash, Bourne shell, and Korn shells and by using $path in the C and TC shells.

Shells also understand both **relative** and **absolute pathnames**. An absolute pathname starts with the root directory (/) and a relative pathname starts with your home directory, your current directory, or the parent of your **current directory** (the directory that you are currently in). For example, /usr/include/sys/param.h is an absolute pathname and ~/courses/cs604 and prog1.c are relative pathnames.

You can refer to your current directory by using `.` (pronounced dot) and the parent of your current directory by using `..` (pronounced dotdot). For example, if nadeem is currently in the courses directory, he can refer to his home directory by using .. and his personal directory by using ../personal. Similarly, he can refer to the directory for this course by using cs604.

Figures 4.5 and 4.6 show sample directory structures in a UNIX/Linux system. The user nadeem has a subdirectory under his home directory, called courses. This directory contains subdirectories for the courses that you have taken, including one for this course.

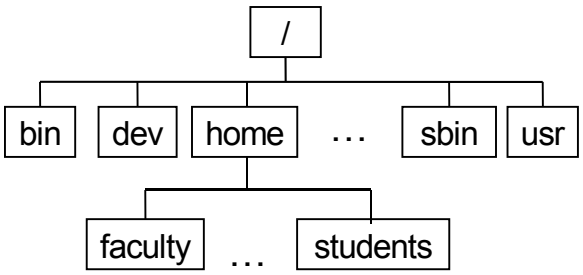


Figure 4.5 UNIX/Linux directory hierarchy

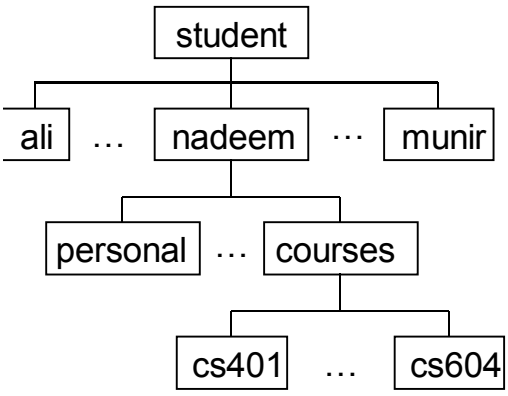


Figure 4.6 Home directories of students

Directory Structure

Some of the more important and commonly used directories in the Linux directory hierarchy are listed in Table 4.1. Many of the directories listed in the table are also found in a UNIX file system.

Table 4.1 Important directories in the Linux operating system and their purpose

/	**The root directory** (not to be concerned with the root account) is similar to a drive letter in Windows (C:\, D:\, etc.) except that in the Linux directory structure there is only one root directory and everything falls under it (including other file systems and partitions). The root directory is the directory that contains all other directories. When a directory structure is displayed as a tree, the root directory is at the top. Typically no files or programs are stored directly under root.
/bin	This directory holds binary executable files that are essential for correct operation of the system (exactly which binaries are in this directory is often dependent upon the distribution). These binaries are usually available for use by all users. **/usr/bin** can also be used for this purpose as well.
/boot	This directory includes essential system boot files including the kernel image .
/dev	This directory contains the devices available to Linux. Remember that Linux treats devices like files and you can read and write to them as if they were. Everything from floppy drives to printers to your mouse is contained in this directory. Included in this directory is the notorious /dev/null, which is most useful for deleting outputs of various, functions and programs.
/etc	Linux uses this directory to store system configuration files. Most files in this directory are text and can be edited with your favorite text editor. This is one of Linux's greatest advantages because there is never a hidden check box and just about all your configurations are in one place. **/etc/inittab** is a text file that details what processes are started at system boot up and during regular operation. **/etc/fstab** identifies file systems and their mount points (like floppy, CD-ROM, and hard disk drives). **/etc/passwd** is where users are defined.
/home	This is where every user on a Linux system will have a personal directory. If your username is "chris" then your home directory will be "/home/chris". A quick way to return to your home directory is by entering the "cd" command. Your current working directory will be changed to your home directory. Usually, the permissions on user directories are set so that only root and the user the directory belongs to can access or store information inside of it. When partitioning a Linux file system this directory will typically need the most space.
/lib	Shared libraries and kernel modules are stored in this directory. The

libraries can be dynamically linked which makes them very similar to DLL files in the Windows environment.

/lost+found This is the directory where Linux keeps files that are restored after a crash or when a partition hasn't been unmounted properly before a shutdown.

/mnt Used for mounting temporary filesystems. Filesystems can be mounted anywhere but the /mnt directory provides a convenient place in the Linux directory structure to mount temporary file systems.

/opt Often used for storage of large applications packages

/proc This is a special, "virtual" directory where system processes are stored. This directory doesn't physically exist but you can often view (or read) the entries in this directory.

/root The home directory for the superuser (root). Not to be confused with the root (/) directory of the Linux file system.

/sbin Utilities used for system administration (halt, ifconfig, fdisk, etc.) are stored in this directory. /usr/sbin, and /usr/local/sbin are other directories that are used for this purpose as well. **/sbin/init.d** are scripts used by /sbin/init to start the system.

/tmp Used for storing temporary files. Similar to C:\Windows\Temp.

/usr Typically a shareable, read-only directory. Contains user applications and supporting files for those applications. **/usr/X11R6** is used by the X Window System. **/usr/bin** contains user accessible commands. **/usr/doc** holds documentation for /usr applications. **/usr/include** this directory contains header files for the C compiler. **/usr/include/g++** contains header files for the C++ compiler. **/usr/lib** libraries, binaries, and object files that aren't usually executed directly by users. **/usr/local** used for installing software locally that needs to be safe from being overwritten when system software updates occur. **/usr/man** is where the manual pages are kept. **/usr/share** is for read-only independent data files. **/usr/src** is used for storing source code of applications installed and kernel sources and headers.

/var This directory contains variable data files such as logs (**/var/log**), mail (**/var/mail**), and spools (**/var/spool**) among other things.

(Source: http://www.chrisshort.net/archives/2005/01/linux-directory-structure.php)

Operating Systems
Lecture No. 5

Reading Material

- Operating Systems Structures, Chapter 4
- PowerPoint Slides for Lecture 3

Summary

- Browsing UNIX/Linux directory structure
- Useful UNIX/Linux commands
- Process concept
- Process scheduling concepts
- Process creation and termination

Browsing UNIX/Linux directory structure

We discussed in detail the UNIX/Linux directory structure in lecture 4. We will continue that discussion and learn how to browse the UNIX/Linux directory structure. In Figure 5.1, we have repeated for our reference the home directory structure for students. In the rest of this section, we discuss commands for creating directories, removing directories, and browsing the UNIX/Linux directory structure.

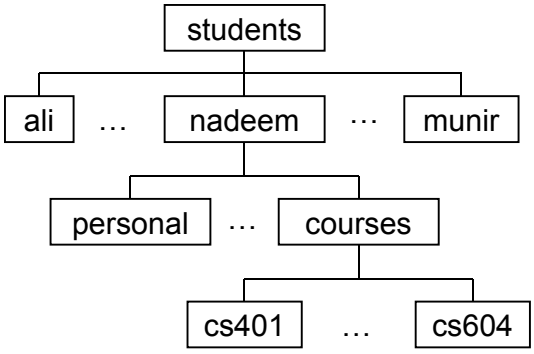


Figure 5.1 Home directories for students

Displaying Directory Contents

You can display the contents (names of files and directories) of a directory with the `ls` command. Without an argument, it assumes your current working directory. So, if you run the `ls` command right after you login, it displays names of files and directories in your home directory. It does not list those files whose names start with a dot (.). Files that start with a dot are known as **hidden files** (also called dot files). You should not modify these files unless you are quite familiar with the

purpose of these files and why you want to modify them. You can display all the files in a directory by using `ls -a` command. Your can display the long listing for the contents of a directory by using the `ls -l` command. The following session shows sample runs of these commands.

```
$ ls
books          courses          LinuxKernel      chatClient.c  chatServer.c
$ ls -a
.              .bash_history    courses          .login        .profile
..             .bash_profile    .cshrc           books
chatClient.c   chatServer.c     LinuxKernel
$ ls -l
drwxr-xr-x   3 msarwar  faculty      512 Oct 28 10:28 books
-rw-r--r--   1 msarwar  faculty     9076 Nov  4 10:14 chatClient.c
-rw-r--r--   1 msarwar  faculty     8440 Nov  4 10:16 chatServer.c
drwxr-xr-x   2 msarwar  faculty      512 Feb 27 17:21 courses
drwxr-xr-x   2 msarwar  faculty      512 Oct 21 14:55 LinuxKernel
$
```

The output of the `ls -l` command gives you the following information about a file:

- 1st character: type of a file
- Rest of letters in the 1st field: access privileges on the file
- 2nd field: number of hard links to the file
- 3rd field: owner of the file
- 4th field: Group of the owner
- 5th field: File size in bytes
- 6th and 7th fields: Date last updated
- 8th field: Time last updated
- 9th field: File name

We will talk about file types and hard links later in the course.

Creating Directories

You can use the `mkdir` command to create a directory. In the following session, the first command creates the courses directory in your current directory. If we assume that your current directory is your home directory, this command creates the courses directory under your home directory. The second command creates the cs604 directory under the ~/courses directory (i.e., the under the courses directory under your home directory). The third command creates the programs directory under your ~/courses/cs604 directory.

```
$ mkdir courses
$ mkdir ~/courses/cs604
$ mkdir ~/courses/cs604/programs
$
```

You could have created all of the above directories with the `mkdir -p ~/courses/cs604/programs` command.

Removing (Deleting) Directories
You can remove (delete) an empty directory with the `mkdir` command. The command in the following session is used to remove the ~/courses/cs604/programs directory if it is empty.

```
$ rmdir courses
$
```

Changing Directory
You can jump from one directory to another (i.e., change your working directory) with the `cd` command. You can use the `cd` ~/courses/cs604/programs command to make ~/courses/cs604/programs directory your working directory. The `cd` or `cd $HOME` command can be used to make your home directory your working directory.

Display Absolute Pathname of Your Working Directory
You can display the absolute pathname of your working directory with the `pwd` command, as shown below.

```
$ pwd
/home/students/nadeem/courses/cs604/programs
$
```

Copying, Moving, and Removing Files

We now discuss the commands to copy, move (or rename), and remove files.

Copying Files
You can use the `cp` command for copying files. You can use the `cp file1 file2` command to copy file1 to file2. The following command can be used to copy file1 in your home directory to the ~/memos directory as file2.

```
$ cp ~/file1 ~/memos/file2
$
```

Moving Files
You can use the `mv` command for moving files. You can use the `mv file1 file2` command to move file1 to file2. The following command can be used to move file1 in your home directory to the ~/memos directory as file2.

```
$ mv ~/file1 ~/memos/file2
$
```

Removing Files
You can use the `rm` command to remove files. You can use the `rm file1` command to remove file1. You can use the first command the following command

to remove the test.c file in the ~/courses/cs604/programs directory and the second command to remove all the files with .o extension (i.e., all object files) in your working directory.

```
$ rm ~/courses/cs604/programs/test.c
$ rm *.o
$
```

Compiling and Running C Programs

You can compile your program with the `gcc` command. The output of the compiler command, i.e., the executable program is stored in the a.out file by default. To compile a source file titled program.c, type:

```
$ gcc program.c
$
```

You can run the executable program generated by this command by typing./a.out and hitting the <Enter> key, as shown in the following session.

```
$ ./a.out
[ ... program output ... ]
$
```

You can store the executable program in a specific file by using the `-o` option. For example, in the following session, the executable program is stored in the assignment file.

```
$ gcc program.c -o assignment
$
```

The `gcc` compiler does not link many libraries automatically. You can link a library explicitly by using the `-l` option. In the following session, we are asking the compiler to link the math library with our object file as it creates the executable file.

```
$ gcc program.c -o assignment -lm
$ assignment
[ ... program output ... ]
$
```

Process Concept

A process can be thought of as a program in execution. A process will need certain resources – such as CPU time, memory, files, and I/O devices – to accompany its task. These resources are allocated to the process either when it is created or while it is executing.

A process is the unit of work in most systems. Such a system consists of a collection of processes: operating system processes execute system code and user processes execute user code. All these processes may execute concurrently.

Although traditionally a process contained only a single thread of control as it ran, most modern operating systems now support processes that have multiple threads.

A batch system executes jobs (background processes), whereas a time-shared system has user programs, or tasks. Even on a single user system, a user may be able to run several programs at one time: a word processor, web browser etc.

A process is more than program code, which is sometimes known as the text section. It also includes the current activity, as represented by the value of the program counter and the contents of the processor's register. In addition, a process generally includes the process stack, which contains temporary data (such as method parameters, the process stack, which contains temporary data), and a data section, which contains global variables.

A program by itself is not a process: a program is a passive entity, such as contents of a file stored on disk, whereas a process is an active entity, with a program counter specifying the next instruction to execute and a set of associated resources. Although two processes may be associated with the same program, they are considered two separate sequences of execution. E.g. several users may be running different instances of the mail program, of which the text sections are equivalent but the data sections vary.

Processes may be of two types:

- **IO bound processes:** spend more time doing IO than computations, have many short CPU bursts. Word processors and text editors are good examples of such processes.
- **CPU bound processes:** spend more time doing computations, few very long CPU bursts.

Process States

As a process executes, it changes states. The state of a process is defined in part by the current activity of that process. Each process may be in either of the following states, as shown in Figure 5.2:

- **New:** The process is being created.
- **Running:** Instructions are being executed.
- **Waiting:** The process is waiting for some event to occur (such as an I/O completion or reception of a signal.
- **Ready:** The process is waiting to be assigned to a processor.
- **Terminated:** The process has finished execution.

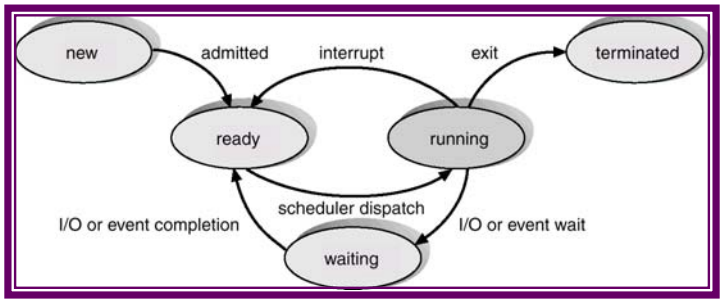


Figure 5.2 Process state diagram

Process Control Block

Each process is represented in the operating system by a **process control block** (PCB) – also called a task control block, as shown in Figure 5.3. A PCB contains many pieces of information associated with a specific process, including these:

- **Process state:** The state may be new, ready, running, waiting, halted and so on.
- **Program counter:** The counter indicates the address of the next instruction to be executed for this process.
- **CPU registers:** The registers vary in number and type, depending on the computer architecture. They include accumulators, index registers, stack pointers and general-purpose registers, plus any condition code information. Along with the program counter, this state information must be saved when an interrupt occurs, to allow the process to be continued correctly afterwards.
- **CPU Scheduling information:** This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.
- **Memory-management information:** This information may include such information such as the value of the base and limit registers, the page tables, or the segment tables, depending on the memory system used by the operating system.
- **Accounting information:** This information includes the amount of CPU and real time used, time limits, account numbers, job or process numbers, and so on.
- **I/O status information:** The information includes the list of I/O devices allocated to the process, a list of open files, and so on.

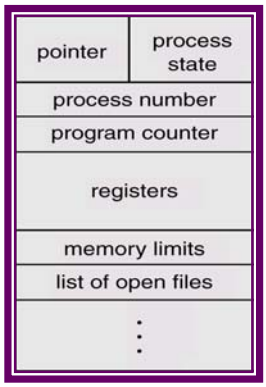


Figure 5.3 Process control block (PCB)

Process Scheduling

The objective of multiprogramming is to have some process running all the time so as to maximize CPU utilization. The objective of time-sharing is to switch the CPU among processors so frequently that users can interact with each program while it is running. A uniprocessor system can have only one running process at a given time. If more processes exist, the rest must wait until the CPU is free and can be rescheduled. Switching the CPU from one process to another requires saving of the context of the current process and loading the state of the new process, as shown in Figure 5.4. This is called **context switching**.

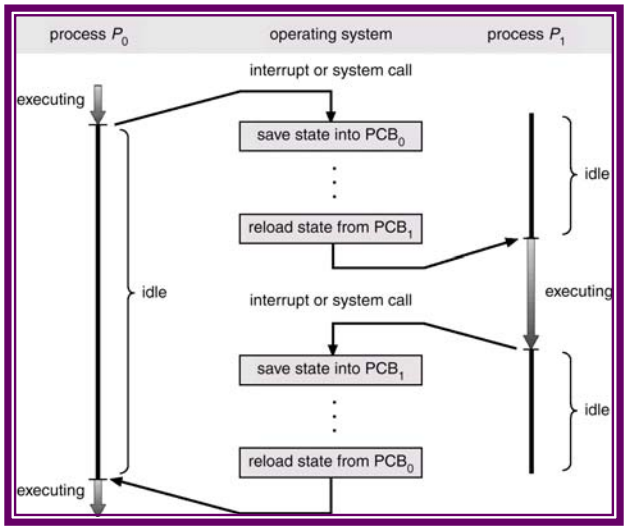


Figure 5.4 Context switching

Scheduling Queues

As shown in Figure 5.5, a contemporary computer system maintains many scheduling queues. Here is a brief description of some of these queues:

- **Job Queue:** As processes enter the system, they are put into a job queue. This queue consists of all processes in the system.
- **Ready Queue:** The processes that are residing in main memory and are ready and waiting to execute are kept on a list called the ready queue. This queue is generally stored as a linked list. A ready-queue header contains pointers to the first and final PCBs in the list. Each PCB is extended to include a pointer field that points to the next PCB in the ready queue.
- **Device Queue:** When a process is allocated the CPU, it executes for a while, and eventually quits, is interrupted or waits for a particular event, such as completion of an I/O request. In the case of an I/O request, the device may be busy with the I/O request of some other process, hence the list of processes waiting for a particular I/O device is called a device queue. Each device has its own device queue.

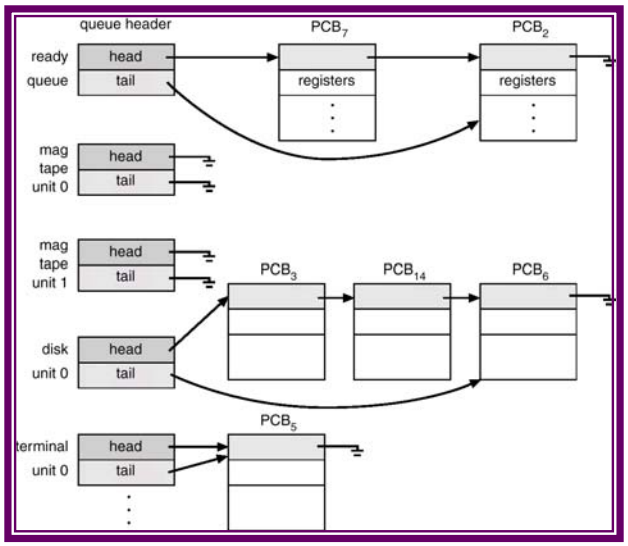


Figure 5.5 Scheduling queue

In the queuing diagram shown in Figure 5.6 below, each rectangle box represents a queue, and two such queues are present, the ready queue and an I/O queue. A new process is initially put in the ready queue, until it is dispatched. Once the process is executing, one of the several events could occur:

- The process could issue an I/O request, and then be placed in an I/O queue.
- The process could create a new sub process and wait for its termination.
- The process could be removed forcibly from the CPU, as a result of an interrupt, and be put back in the ready queue.

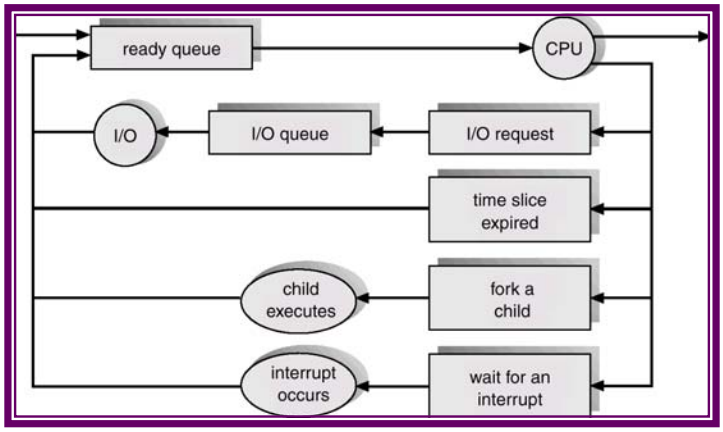


Figure 5.6 Queuing diagram of a computer system

Schedulers

A process migrates between the various scheduling queues throughout its lifetime. The operating system must select, for scheduling purposes, processes from these queues in some fashion. The appropriate scheduler carries out this selection process. The **Long-term scheduler** (or job scheduler) selects which processes should be brought into the ready queue, from the job pool that is the list of all jobs in the system. The **Short-term scheduler** (or CPU scheduler) selects which process should be executed next and allocates CPU.

The primary distinction between the two schedulers is the frequency of execution. The short-term scheduler must select a new process for the CPU frequently. A process may execute for only a few milliseconds before waiting for an I/O request. Often the short-term scheduler executes at least once every 100 milliseconds. Because of the brief time between executions, the short-term scheduler must be fast. If it takes 10 milliseconds to decide to execute a process for 100 milliseconds, then 10/(100+10)=9 % of the CPU is being used for scheduling only. The long-term scheduler, on the other hand executes much less frequently. There may be minutes between the creations of new processes in the system. The long-term scheduler controls the degree of multiprogramming – the number of processes in memory. If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average department rate of processes leaving the system. Because of the longer interval between execution s, the long-term scheduler can afford to take more time to select a process for execution.

The long-term scheduler must select a good mix of I/O bound and CPU bound jobs. The reason why the long-term scheduler must select a good mix of I/O bound and CPU bound jobs is that if the processes are I/O bound, the ready queue will be mostly empty and the short-term scheduler will have little work. On the other hand, if the processes are mostly CPU bound, then the devices will go unused and the system will be unbalanced.

Some operating systems such as time-sharing systems may introduce a **medium-term scheduler**, which removes processes from memory (and from active contention for the CPU) and thus reduces the degree of multiprogramming. At some later time the process can be reintroduced at some later stage, this scheme is called **swapping.** The process is swapped out, and is later swapped in by the medium term scheduler. Swapping may be necessary to improve the job mix, or because a change is memory requirements has over committed available memory, requiring memory to be freed up. As shown in Figure 5.7, the work carried out by the swapper to move a process from the main memory to disk is known as swap out and moving it back into the main memory is called swap in. The area on the disk where swapped out processes are stored is called the **swap space**.

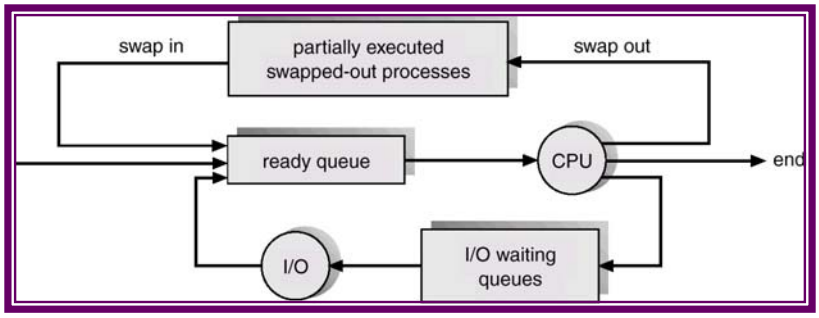


Figure 5.7 Computer system queues, servers, and swapping

Operating Systems
Lecture No. 6

Reading Material

- Operating Systems Concepts, Chapter 4
- UNIX/Linux manual pages for the `fork()` system call

Summary

- Process creation and termination
- Process management in UNIX/Linux— system calls: `fork`, `exec`, `wait`, `exit`
- Sample codes

Operations on Processes

The processes in the system execute concurrently and they must be created and deleted dynamically thus the operating system must provide the mechanism for the creation and deletion of processes.

Process Creation

A process may create several new processes via a create-process system call during the course of its execution. The creating process is called a **parent process** while the new processes are called the **children** of that process. Each of these new processes may in turn create other processes, forming a tree of processes. Figure 6.1 shows partially the process tree in a UNIX/Linux system.

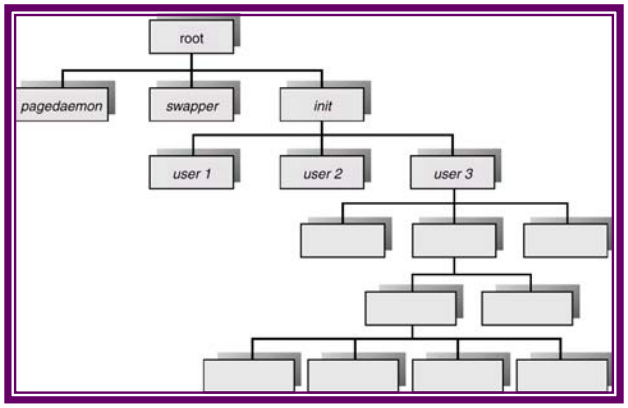


Figure 6.1 Process tree in UNIX/Linux

In general, a process will need certain resources (such as CPU time, memory files, I/O devices) to accomplish its task. When a process creates a sub process, also known as a child, that sub process may be able to obtain its resources directly from the operating system or may be constrained to a subset of the resources of the parent process. The parent may have to partition its resources among several of its children. Restricting a

process to a subset of the parent's resources prevents a process from overloading the system by creating too many sub processes.

When a process is created it obtains in addition to various physical and logical resources, initialization data that may be passed along from the parent process to the child process. When a process creates a new process, two possibilities exist in terms of execution:

1. The parent continues to execute concurrently with its children.
2. The parent waits until some or all of its children have terminated.

There are also two possibilities in terms of the address space of the new process:

1. The child process is a duplicate of the parent process.
2. The child process has a program loaded into it.

In order to consider these different implementations let us consider the UNIX operating system. In UNIX its process identifier identifies a process**,** which is a unique integer. A new process is created by the `fork` system call. The new process consists of a copy of the address space of the parent. This mechanism allows the parent process to communicate easily with the child process. Both processes continue execution at the instruction after the `fork` call, with one difference, the return code for the `fork` system call is zero for the child process, while the process identifier of the child is returned to the parent process.

Typically the `execlp` system call is used after a `fork` system call by one of the two processes to replace the process' memory space with a new program. The `execlp` system call loads a binary file in memory –destroying the memory image of the program containing the `execlp` system call.—and starts its execution. In this manner, the two processes are able to communicate and then go their separate ways. The parent can then create more children, or if it has nothing else to do while the child runs, it can issue a `wait` system call to move itself off the ready queue until the termination of the child. The parent waits for the child process to terminate, and then it resumes from the call to wait where it completes using the `exit` system call.

Process termination

A process terminates when it finishes executing its final statement and asks the operating system to delete it by calling the exit system call. At that point, the process may return data to its parent process (via the wait system call). All the resources of the process including physical and virtual memory, open the files and I/O buffers – are de allocated by the operating system.

Termination occurs under additional circumstances. A process can cause the termination of another via an appropriate system call (such as abort). Usually only the parent of the process that is to be terminated can invoke this system call. Therefore parents need to know the identities of its children, and thus when one process creates another process, the identity of the newly created process is passed to the parent.

A parent may terminate the execution of one of its children for a variety of reasons, such as:

- The child has exceeded its usage of some of the resources that it has been allocated. This requires the parent to have a mechanism to inspect the state of its children.
- The task assigned to the child is no longer required.

- The parent is exiting, and the operating system does not allow a child to continue if its parent terminates. On such a system, if a process terminates either normally or abnormally, then all its children must also be terminated. This phenomenon referred to as cascading termination, is normally initiated by the operating system.

Considering an example from UNIX, we can terminate a process by using the `exit` system call, its parent process may wait for the termination of a child process by using the wait system call. The wait system call returns the process identifier of a terminated child, so that the parent can tell which of its possibly many children has terminated. If the parent terminates however all its children have assigned as their new parent, the `init` process. Thus the children still have a parent to collect their status and execution statistics.

The `fork()` system call

When the fork system call is executed, a new process is created. The original process is called the parent process whereas the process is called the child process. The new process consists of a copy of the address space of the parent. This mechanism allows the parent process to communicate easily with the child process. On success, both processes continue execution at the instruction after the `fork` call, with one difference, the return code for the `fork` system call is zero for the child process, while the process identifier of the child is returned to the parent process. On failure, a -1 will be returned in the parent's context, no child process will be created, and an error number will be set appropriately.

The synopsis of the fork system call is as follows:

```
#include <sys/types.h>
#include <unistd.h>
pid_t fork(void);
```

```
main()
{
     int pid;
     ...
     pid = fork();
     if (pid == 0) {
     /* Code for child */
     ...
     }
     else {
     /* Code for parent */
     ...
     }
     ...
}
```

Figure 6.2 Sample code showing use of the `fork()` system call

Figure 6.2 shows sample code, showing the use of the `fork()` system call and Figure 6.3 shows the semantics of the fork system call. As shown in Figure 6.3, `fork()`

creates an exact memory image of the parent process and returns 0 to the child process and the process ID of the child process to the parent process.

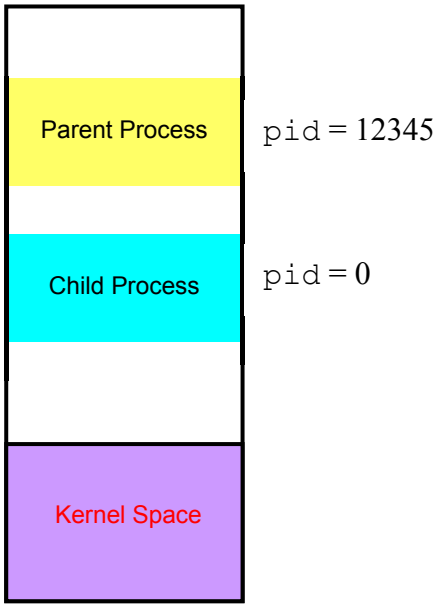


Figure 6.3 Semantics of the fork system call

After the `fork()` system call the parent and the child share the following:

- Environment
- Open file descriptor table
- Signal handling settings
- Nice value
- Current working directory
- Root directory
- File mode creation mask (umask)

The following things are different in the parent and the child:

- Different process ID (PID)
- Different parent process ID (PPID)
- Child has its own copy of parent's file descriptors

The `fork()` system may fail due to a number of reasons. One reason maybe that the maximum number of processes allowed to execute under one user has exceeded, another could be that the maximum number of processes allowed on the system has exceeded. Yet another reason could be that there is not enough swap space.

Operating Systems
Lecture No. 7

Reading Material

- Operating Systems Concepts, Chapter 4
- UNIX/Linux manual pages for `execlp()`, `exit()`, and `wait()` system calls

Summary

- The `execlp()`, `wait()`, and `exec()` system calls and sample code
- Cooperating processes
- Producer-consumer problem
- Interprocess communication (IPC) and process synchronization

The `wait()` system call

The wait system call suspends the calling process until one of the immediate children terminate, or until a child that is being traced stops because it has hit an event of interest. The wait will return prematurely if a signal is received. If all child processes stopped or terminated prior to the call on wait, return is immediate. If the call is successful, the process ID of a child is returned. If the parent terminates however all its children have assigned as their new parent, the init process. Thus the children still have a parent to collect their status and execution statistics. The synopsis of the wait system call is as follows:

```
#include <sys/types.h>
#include <sys/wait.h>
pid_t wait(int *stat_loc);
```

A **zombie process** is a process that has terminated but whose exit status has not yet been received by its parent process or by init. Sample code showing the use of `fork()` and `wait()` system calls is given in Figure 7.1 below.

```
#include <stdio.h>
void main(){
  int pid, status;
  pid = fork();
  if(pid == -1) {
   printf("fork failed\n");
   exit(1);
  }
   if(pid == 0) {  /* Child */
       printf("Child here!\n");
       exit(0);
  }
  else { /* Parent */
    wait(&status);
```

```
    printf("Well done kid!\n");
    exit(0);
  }
}
```

Figure 7.1 Sample code showing use of the `fork()` and `wait()` system calls

The `execlp()` system call

Typically, the `execlp()` system call is used after a `fork()` system call by one of the two processes to replace the process' memory space with a new program. The new process image is constructed from an ordinary, executable file. This file is either an executable object file, or a file of data for an interpreter. There can be no return from a successful *exec* because the calling process image is overlaid by the new process image. In this manner, the two processes are able to communicate and then go their separate ways. The synopsis of the `execlp()` system call is given below:

```
#include <unistd.h>
int execlp (const char *file, const,char *arg0, ...,
            const char *argn,(char *)0);
```

Sample code showing the use of `fork()` and `execlp()` system calls is given in Figure 7.2 below.

```
#include <stdio.h>
void main()
{
  int pid, status;

  pid = fork();
  if(pid == -1) {
    printf("fork failed\n");
    exit(1);
  }
  if(pid == 0) {  /* Child */
      if (execlp("/bin/ls", "ls", NULL)< 0) {
        printf("exec failed\n");
        exit(1);
      }
  }
  else { /* Parent */
    wait(&status);
    printf("Well done kid!\n");
    exit(0);
  }
}
```

Figure 7.2 Sample code showing use of `fork()`, `execlp()`, `wait()`, and `exit()`

The semantics of `fork()`, followed by an execlp() system call are shown In Figure 7.3 below.

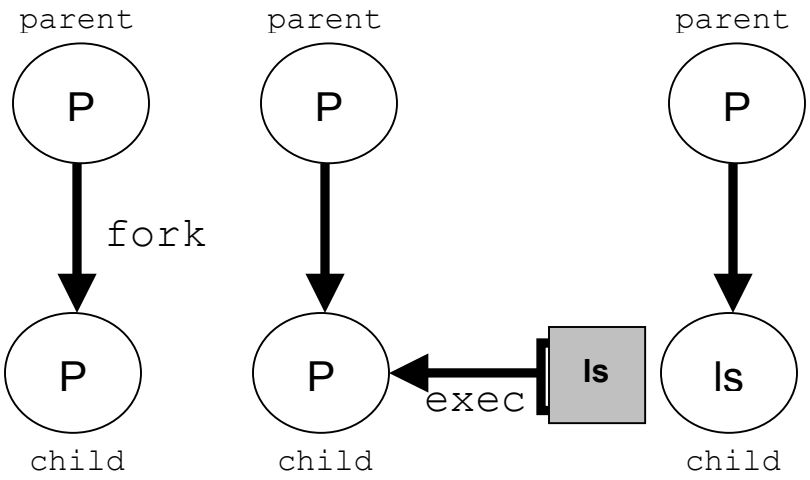


Figure 7.3 Semantics of `fork()` followed by `exec()`

Cooperating Processes

The concurrent processes executing in the operating system may be either independent processes or cooperating processes. A process is independent if it cannot affect or be affected by any other process executing in the system. Clearly any process that shares data with other processes is a cooperating process. The advantages of cooperating processes are:

- **Information sharing:** Since several users may be interested in the same piece of information (for instance, a shared file) we must provide an environment to allow concurrent users to access these types of resources.
- **Computation speedup:** If we want a particular task to run faster, we must break it into subtasks each of which will be running in parallel with the others. Such a speedup can be obtained only if the computer has multiple processing elements (such as CPU's or I/O channels).
- **Modularity:** We may want to construct the system in a modular fashion, dividing the system functions into separate processes or threads.
- **Convenience:** Even an individual user may have many tasks on which to work at one time. For instance, a user may be editing, printing, and compiling in parallel.

To illustrate the concept of communicating processes, let us consider the producer-consumer problem. A **producer** process produces information that is consumed by a **consumer** process. For example, a compiler may produce assembly code that is consumed by an assembler. To allow a producer and consumer to run concurrently, we must have available a buffer of items that can be filled by a producer and emptied by a consumer. The producer and consumer must be synchronized so that the consumer does not try to consume an item that has not yet been produced. The bounded buffer problem assumes a fixed buffer size, and the consumer must wait if the buffer is empty and the producer must wait if the buffer is full, whereas the unbounded buffer places no practical limit on the size of the buffer. Figure 7.4 shows the problem in a diagram. This buffer may be provided by interprocess communication (discussed in the next section) or with the use of shared memory.

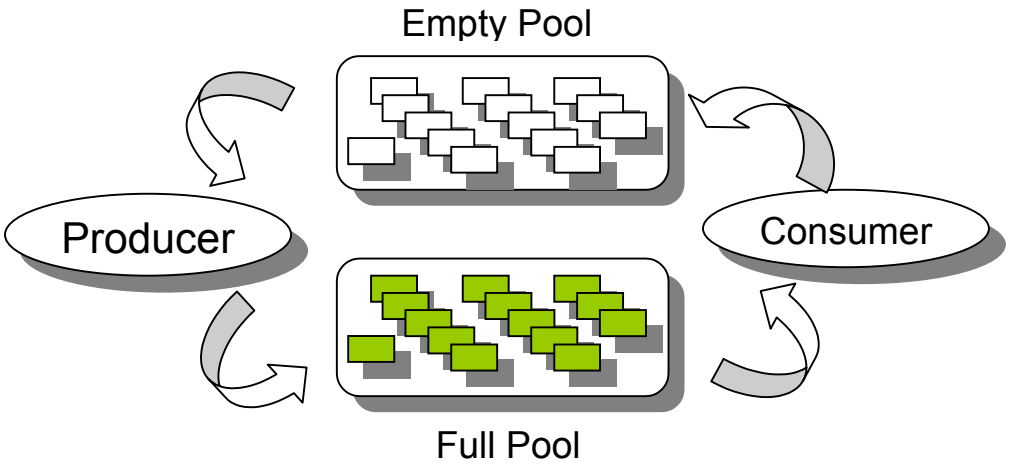


Figure 7.4 The producer-consumer problem

Figure 7.5 shows the shared buffer and other variables used by the producer and consumer processes.

```
#define BUFFER_SIZE 10
typedef struct
{
…
} item;
item buffer[BUFFER_SIZE];
int in=0;
int out=0;
```

Figure 7.5 Shared buffer and variables used by the producer and consumer processes

The shared buffer is implemented as a circular array with two logical pointers: in an out. The 'in' variable points to the next free position in the buffer; 'out' points to the first full position in the buffer. The buffer is empty when `in==out`, the buffer is full when `((in+1)%BUFFER_SIZE)==out`. The code structures for the producer and consumer processes are shown in Figure 7.6.

```
Producer Process
while(1) {
    /*Produce an item in nextProduced*/
    while(((in+1)%BUFFER_SIZE)==out); /*do nothing*/
       buffer[in]=nextProduced;
       in=(in+1)%BUFFER_SIZE;
}
Consumer Process
while(1) {
   while(in == out); //do nothing
   nextConsumed=buffer[out];
   out=(out+1)%BUFFER_SIZE;
   /*Consume the item in nextConsumed*/
}
```

Figure 7.6 Code structures for the producer and consumer processes

Operating Systems
Lecture No. 8

Reading Material

- Operating Systems Concepts, Chapter 4
- UNIX/Linux manual pages for `pipe()`, `fork()`, `read()`, `write()`, `close()`, and `wait()` system calls

Summary

- Interprocess communication (IPC) and process synchronization
- UNIX/Linux IPC tools (pipe, named pipe—FIFO, socket, TLI, message queue, shared memory)
- Use of UNIC/Linux pipe in a sample program

Interprocess Communication (IPC)

IPC provides a mechanism to allow processes to communicate and to synchronize their actions without sharing the same address space. We discuss in this section the various message passing techniques and issues related to them.

Message Passing System

The function of a message system is to allow processes to communicate without the need to resort to the shared data. Messages sent by a process may be of either fixed or variable size. If processes P and Q want to communicate, a communication link must exist between them and they must send messages to and receive messages from each other through this link. Here are several methods for logically implementing a link and the send and receive options:

- Direct or indirect communication
- Symmetric or asymmetric communication
- Automatic or explicit buffering
- Send by copy or send by reference
- Fixed size or variable size messages

We now look at the different types of message systems used for IPC.

Direct Communication

With direct communication, each process that wants to communicate must explicitly name the recipient or sender of the communication. The send and receive primitives are defined as:

- Send(P, message) – send a message to process P
- Receive(Q, message) – receive a message from process Q.

A communication link in this scheme has the following properties:

- A link is established automatically between every pair of processes that want to communicate. The processes need to know only each other's identity to communicate
- A link is associated with exactly two processes.
- Exactly one link exists between each pair of processes.

Unlike this symmetric addressing scheme, a variant of this scheme employs asymmetric addressing, in which the recipient is not required to name the sender.

- Send(P, message) – send a message to process P
- Receive(id, message) – receive a message from any process; the variable id is set to the name of the process with which communication has taken place.

Indirect Communication

With indirect communication, messages can be sent to and received from mailboxes. Here, two processes can communicate only if they share a mailbox. The send and receive primitives are defined as:

- Send(A, message) – send a message to mailbox A.
- Receive(A, message) – receive a message from mailbox A.

A communication link in this scheme has the following properties:

- A link is established between a pair of processes only if both members have a shared mailbox.
- A link is associated with more than two processes.
- A number of different links may exist between each pair of communicating processes, with each link corresponding to one mailbox.

Synchronization

Communication between processes takes place by calls to send and receive primitives (i.e., functions). Message passing may be either blocking or non-blocking also called as synchronous and asynchronous.

- **Blocking send:** The sending process is blocked until the receiving process or the mailbox receives the message.
- **Non-blocking send:** The sending process sends the message and resumes operation.
- **Blocking receive:** The receiver blocks until a message is available.
- **Non-blocking receiver:** The receiver receives either a valid message or a null.

Buffering

Whether the communication is direct or indirect, messages exchanged by the processes reside in a temporary queue. This queue can be implemented in three ways:

- **Zero Capacity:** The queue has maximum length zero, thus the link cannot have any messages waiting in it. In this case the sender must block until the message has been received.
- **Bounded Capacity:** This queue has finite length n; thus at most n messages can reside in it. If the queue is not full when a new message is sent, the later is placed in the queue and the sender resumes operation. If the queue is full, the sender blocks until space is available.

- **Unbounded Capacity:** The queue has infinite length; thus the sender never blocks.

UNIX/Linux IPC Tools

UNIX and Linux operating systems provide many tools for interprocess communication, mostly in the form of APIs but some also for use at the command line. Here are some of the commonly supported IPC tools in the two operating systems.

- Pipe
- Named pipe (FIFO)
- BSD Socket
- TLI
- Message queue
- Shared memory
- Etc.

Overview of `read()`, `write()`, and `close()` System Calls

We need to understand the purpose and syntax of the read, write and close system calls so that we may move on to understand how communication works between various Linux processes. The read system call is used to read data from a file descriptor. The synopsis of this system call is:

```
#include <unistd.h>
ssize_t read(int fd, void *buf, size_t count);
```

`read()` attempts to read up to count bytes from file descriptor fd into the buffer starting at buf. If count is zero, `read()` returns zero and has no other results. If count is greater than SSIZE_MAX, the result is unspecified. On success, `read()` returns the number of bytes read (zero indicates end of file) and advances the file position pointer by this number.

The `write()` system call is used to write to a file. Its synopsis is as follows:

```
#include <unistd.h>
ssize_t write(int fd, const void *buf, size_t count);
```

`write()` attempts to write up to count bytes to the file referenced by the file descriptor fd from the buffer starting at buf. On success, `write()` returns the number of bytes written are returned (zero indicates nothing was written) and advances the file position pointer by this number. On error, `read()` returns -1, and *`errno`* is set appropriately. If count is zero and the file descriptor refers to a regular file, 0 will be returned without causing any other effect.

The `close()` system call is used to close a file descriptor. Its synopsis is:

```
#include <unistd.h>
int close(int fd);
```

`close()` closes a file descriptor, so that it no longer refers to any file and may be reused. If fd is the last copy of a particular file descriptor the resources associated with it are freed; if the descriptor was the last reference to a file which has been removed using

`unlink(2)` the file is deleted. `close()` returns zero on success, or -1 if an error occurred.

Pipes

A UNIX/Linux pipe can be used for IPC between related processes on a system. Communicating processes typically have sibling or parent-child relationship. At the command line, a pipe can be used to connect the standard output of one process to the standard input of another. Pipes provide a method of one-way communication and for this reason may be called half-duplex pipes.

The `pipe()` system call creates a pipe and returns two file descriptors, one for reading and second for writing, as shown in Figure 8.1. The files associated with these file descriptors are streams and are both opened for reading and writing. Naturally, to use such a channel properly, one needs to form some kind of protocol in which data is sent over the pipe. Also, if we want a two-way communication, we'll need two pipes.

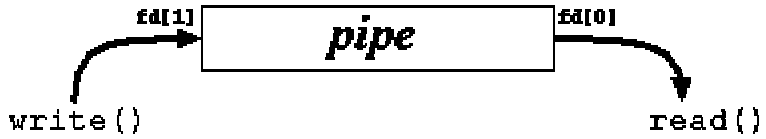


Figure 8.1 A UNIX/Linux pipe with a read end and a write end

The system assures us of one thing: the order in which data is written to the pipe, is the same order as that in which data is read from the pipe. The system also assures that data won't get lost in the middle, unless one of the processes (the sender or the receiver) exits prematurely. The `pipe()` system call is used to create a read-write pipe that may later be used to communicate with a process we'll fork off. The synopsis of the system call is:

```
#include <unistd.h>
int pipe (int fd[2]);
```

Each array element stores a file descriptor. fd[0] is the file descriptor for the read end of the pipe (i.e., the descriptor to be used with the read system call), whereas fd[1] is the file descriptor for the write end of the pipe. (i.e., the descriptor to be used with the write system call).The function returns -1 if the call fails. A pipe is a bounded buffer and the maximum data written is PIPE_BUF, defined in <sys/param.h> in UNIX and in <linux/param.h> in Linux as 5120 and 4096, respectively.

Lets see an example of a two-process system in which the parent process creates a pipe and forks a child process. The child process writes the 'Hello, world!' message to the pipe. The parent process reads this messages and displays it on the monitor screen. Figure 8.2 shows the protocol for this communication and Figure 8.3 shows the corresponding C source code.

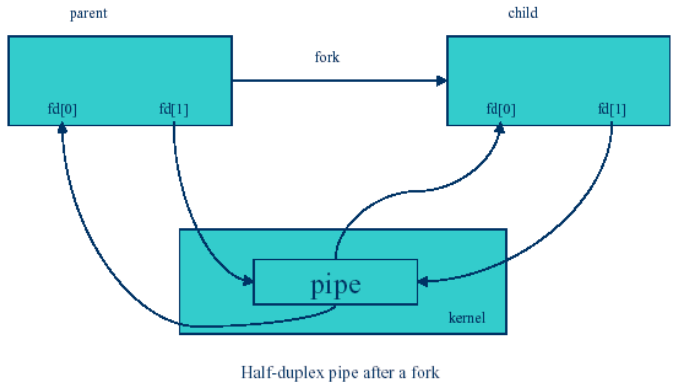


Figure 8.2 Use of UNIX/Linux pipe by parent and child for half-duplex communication

```
/* Parent creates pipe, forks a child, child writes into
   pipe, and parent reads from pipe */
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
main()
{
        int pipefd[2], pid, n, rc, nr, status;
        char *testString = "Hello, world!\n", buf[1024];

        rc = pipe (pipefd);
        if (rc < 0) {
                perror("pipe");
                exit(1);
        }
        pid = fork ();
        if (pid < 0) {
                perror("fork");
                exit(1);
        }
        if (pid == 0) {  /* Child's Code */
            close(pipefd[0]);
            write(pipefd[1], testString, strlen(testString));
            close(pipefd[1]);
            exit(0);
        }
        /* Parent's Code */
        close(pipefd[1]);
        n = strlen(testString);
        nr = read(pipefd[0], buf, nA);
        rc = write(1, buf, nr);
        wait(&status);
        printf("Good work child!\n");
        return(0);
}
```

Figure 8.3 Sample code showing use of UNIX/Linux pipe for IPC between related processes—child write the "Hello, world!" message to the parent, who reads its and displays it on the monitor screen

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.

Operating Systems
Lecture No. 9

Reading Material

- Operating Systems Concepts, Chapter 4
- UNIX/Linux manual pages for `pipe()`, `fork()`, `read()`, `write()`, `close()`, and `wait()` system calls
- Lecture 9 on Virtual TV

Summary

- UNIX/Linux interprocess communication (IPC) tools and associated system calls
- UNIX/Linux standard files and kernel's mechanism for file access
- Use of pipe in a program and at the command line

Unix/Linux IPC Tools

The UNIX and Linux operating systems provide many tools for interprocess communication (IPC). The three most commonly used tools are:

- **Pipe:** Pipes are used for communication between related processes on a system, as shown in Figure 9.1. The communicating processes are typically related by sibling or parent-child relationship.

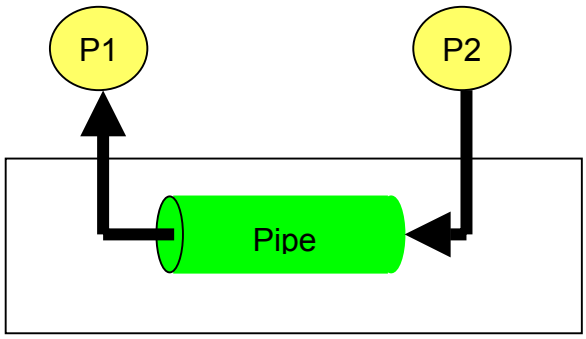


Figure 9.1 Pipes on a UNIX/Linux system

- **Named pipe (FIFO)**: FIFOs (also known as named pipes) are used for communication between related or unrelated processes on a UNIX/Linux system, as shown in Figure 9.2.

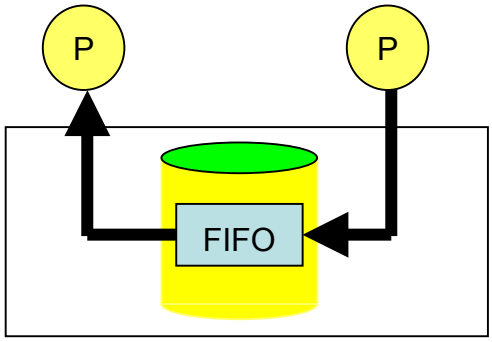


Figure 9.2 Pipes on a UNIX/Linux system

- **BSD Socket**: The BSD sockets are used for communication between related or unrelated processes on the same system or unrelated processes on different systems, as shown in Figure 9.3.

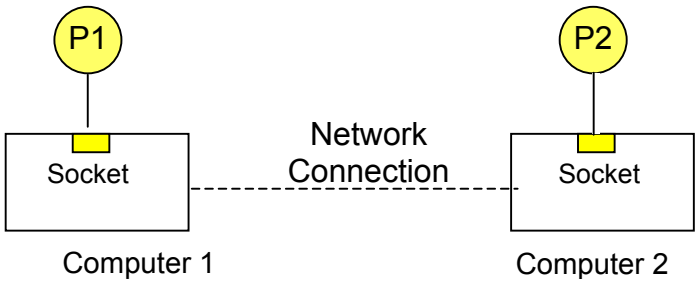


Figure 9.3 Sockets used for IPC between processes on different UNIX/Linux systems

The `open()` System call

The `open()` system call is used to open or create a file. Its synopsis is as follows:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int open(const char *pathname, int flags);
int open(const char pathname, int oflag, /* mode_t mode */);
```

The call converts a pathname into a file descriptor (a small, non-negative integer for use in subsequent I/O as with read, write, etc.). When the call is successful, the file descriptor returned will be the lowest file descriptor not currently open for the process. This system call can also specify whether read or write will be blocking or non-blocking.

The 'oflag' argument specifies the purpose of opening the file and 'mode' specifies permission on the file if it is to be created. 'oflag' value is constructed by ORing various flags: O_RDONLY, O_WRONLY, O_RDWR, O_NDELAY (or O_NONBLOCK), O_APPEND, O_CREAT, etc.

`The open()` system call can fail for many reasons, some of which are:

- Non-existent file
- Operation specified is not allowed due to file permissions

- Search not allowed on a component of pathname
- User's disk quota on the file system has been exhausted

The file descriptor returned by the `open()` system call is used in the `read()` and `write()` calls for file (or pipe) I/O.

The `read()` system call

We discussed the `read()` system call in the notes for lecture 8. The call may fail for various reasons, including the following:

- Invalid 'fildes', 'buf', or 'nbyte'
- Signal caught during read

The `write()` system call

The call may fail for various reasons, including the following:

- Invalid argument
- File size limit for process or for system would exceed
- Disk is full

The `close()` system call

As discussed in the notes for lecture 8, the close() system call is used to close a file descriptor. It takes a file (or pipe) descriptor as an argument and closes the corresponding file (or pipe end).

Kernel Mapping of File Descriptors

Figure 9.4 shows the kernel mapping of a file descriptor to the corresponding file. The system-wide File Table contains entries for all of the open files on the system. UNIX/Linux allocates an inode to every (unique) file on the system to store most of the attributes, including file's location. On a read or write call, kernel traverses this mapping to reach the corresponding file.

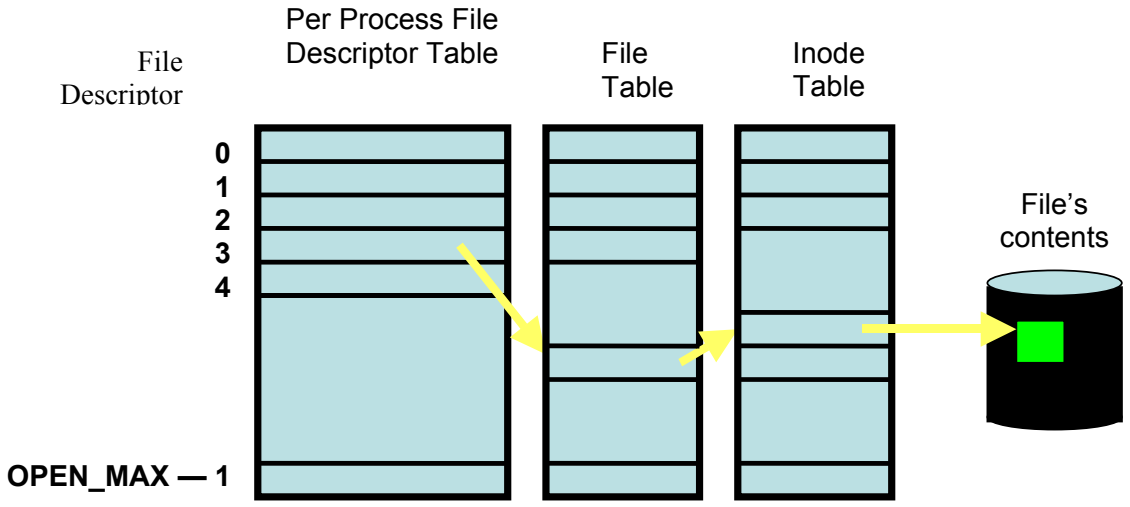


Figure 9.4 File descriptors and their mapping to files

Standard Descriptors in Unix/Linux

Three files are automatically opened by the kernel for every process for the process to read its input from and send its output and error messages to. These files are called **standard files**: standard input, standard output, and standard error. By default, standar d files are attached to the terminal on which a process runs. The descriptors for standard files are known as **standard file descriptors**. Standard files, their descriptors, and their default attachments are:

- Standard input: 0 (keyboard)
- Standard output: 1 (display screen)
- Standard error: 2 (display screen)

The `pipe()` System Call

We discussed the `pipe()` system call in the notes for lecture 8. The `pipe()` system call fails for many reasons, including the following:

- At least two slots are not empty in the PPFDT—too many files or pipes are open in the process
- Buffer space not available in the kernel
- File table is full

Sample Code for IPC with a UNIX/Linux Pipe

We discussed in the notes for lecture 8 a simple protocol for communication between a parent and its child process using a pipe. Figure 9.5 shows the protocol. Code is reproduced in Figure 9.6.

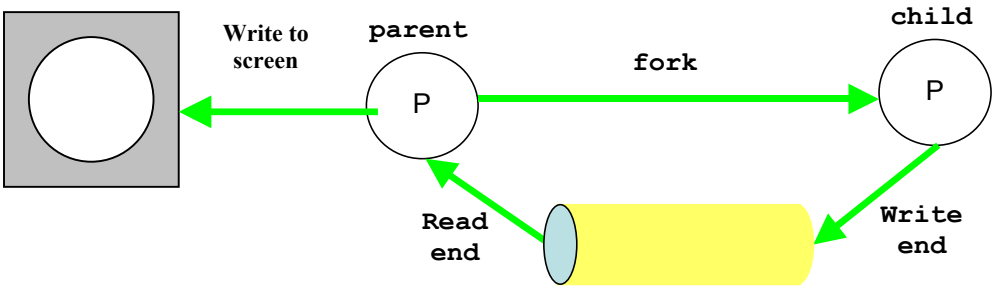


Figure 9.5 IPC between parent and child processes with a UNIX/Linux pipe

```
/* Parent creates pipe, forks a child, child writes into
   pipe, and parent reads from pipe */
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
main()
{
        int pipefd[2], pid, n, rc, nr, status;
        char *testString = "Hello, world!\n", buf[1024];

        rc = pipe (pipefd);
        if (rc < 0) {
                perror("pipe");
```

```
                exit(1);
        }
        pid = fork ();
        if (pid < 0) {
                perror("fork");
                exit(1);
        }
        if (pid == 0) {  /* Child's Code */
            close(pipefd[0]);
            write(pipefd[1], testString, strlen(testString));
            close(pipefd[1]);
            exit(0);
        }
        /* Parent's Code */
        close(pipefd[1]);
        n = strlen(testString);
        nr = read(pipefd[0], buf, nA);
        rc = write(1, buf, nr);
        wait(&status);
        printf("Good work child!\n");
        return(0);
}
```

Figure 9.6 Sample code showing use of UNIX/Linux pipe for IPC between related processes—child write the "Hello, world!" message to the parent, who reads its and displays it on the monitor screen

Command Line Use of UNIX/Linux Pipes

Pipes can also be used on the command line to connect the standard input of one process to the standard input of another. This is done by using the pipe operator which is | and the syntax is as follows:

```
cmd1 | cmd2 | ... | cmdN
```

The semantics of this command line are shown in Figure 9.7.

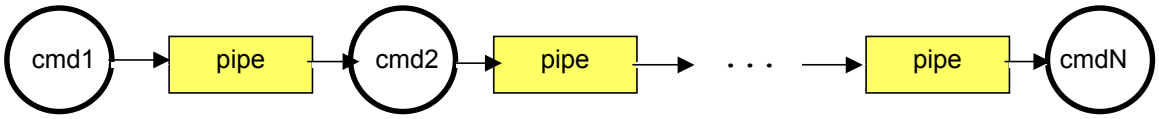


Figure 9.7 Semantics of the command line that connects `cmd1` through `cmdN` via pipes.

The following example shows the use of the pipe operator in a shell command.

```
cat /etc/passwd | grep zaheer
```

The effect of this command is that `grep` command displays lines in the /etc/passwd file that contain the string "zaheer". Figure 9.8 illustrates the semantics of this command.

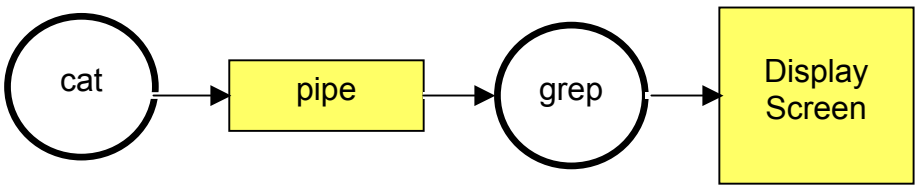


Figure 9.8 Semantics of the cat /etc/passwd | grep zaheer command

The work performed by the above command can be performed by the following sequence of commands without using the pipe operator. The first command saves the /etc/passwd file in the temp1 file and the second command displays those lines in temp1 which contain the string "zaheer". After the temp1 file has been used for the desired work, it is deleted.

```
$ cat /etc/passwd > temp1
$ grep "zaheer" temp1
$ rm temp1
```

Operating Systems
Lecture No. 10

Reading Material

- UNIX/Linux manual pages for the `mknod()` system call, the `mkfifo()` library call, and the `mkfifo` command
- Lecture 10 on Virtual TV

Summary

- Input, output, and error redirection in UNIX/Linux
- FIFOs in UNIX/Linux
- Use of FIFOs in a program and at the command line

Input, output and error redirection in UNIX/Linux

Linux redirection features can be used to detach the default files from stdin, stdout, and stderr and attach other files with them for a single execution of a command. The act of detaching defaults files from stdin, stdout, and stderr and attaching other files with them is known as input, output, and error redirection. In this section, we show the syntax, semantics, and examples of I/O and error redirection.

Input Redirection: Here is the syntax for input redirection:

```
command < input-file
```

or

```
command 0< input-file
```

With this command syntax, keyboard is detached from stdin of 'command' and 'input-file' is attached to it, i.e., 'command' reads input from 'input-file' and not keyboard. Note that 0< operator cannot be used with the C and TC shells. Here is an example use of input redirection. In these examples, the `cat` and `grep` commands read input from the Phones file and not from keyboard.

```
$ cat < Phones
[ contents of Phones ]
$ grep "Nauman" < Phones
[ output of grep ]
$
```

Output Redirection: Here is the syntax for output redirection:

```
command > output-file
```

or

```
command 1> output-file
```

With this command syntax, the display screen is detached from stdout and 'output-file' is attached to it, i.e., 'command' sends output to 'output-file' and not the display screen. Note that 1> operator cannot be used with the C and TC shells. Here is an example use of input redirection. In these examples, the `cat`, `grep`, and `find` commands send their outputs to the Phones, Ali.Phones, and foo.log files, respectively, and not to the display screen.

```
$ cat > Phones
[ your input ]
<Ctrl-D>
$ grep "Ali" Phones > Ali.phones
[ output of grep ]
$ find ~ -name foo -print > foo.log
[ error messages ]
$
```

Error Redirection: Here is the syntax for error redirection:

```
command 2> error-file
```

With this command syntax, the display screen is detached from stderr and 'error-file' is attached to it, i.e., error messages are sent to 'error-file' and not the display screen. Note that 2> cannot be used under C and TC shells. The following are a few examples of error redirection. In these examples, the first `find` command sends its error messages to the errors file and the second `find` command sends its error messages to the /dev/null file. The `ls` command sends its error messages to the error.log file and not to the display screen.

```
$ find ~ -name foo -print 2> errors
[ output of the find command ]
$ ls -l foo 2> error.log
[ output of the find command ]
$ cat error.log
ls: foo: No such file or directory
$ find / -name ls -print 2> /dev/null
/bin/ls
$
```

UNIX/Linux FIFOs

A named pipe (also called a named FIFO, or just FIFO) is a pipe whose access point is a file kept on the file system. By opening this file for reading, a process gets access to the FIFO for reading. By opening the file for writing, the process gets access to the FIFO for writing. By default, a FIFO is opened for blocking I/O. This means that a process reading from a FIFO blocks until another process writes some data in the FIFO. The same goes the other way around. Unnamed pipes can only be used between processes that have an ancestral relationship. And they are temporary; they need to be created every time and are destroyed when the corresponding processes exit. Named pipes (FIFOs) overcome both of these limitations. Figure 10.1 shows two unrelated processes, P1 and P2, communicating with each other using a FIFO.

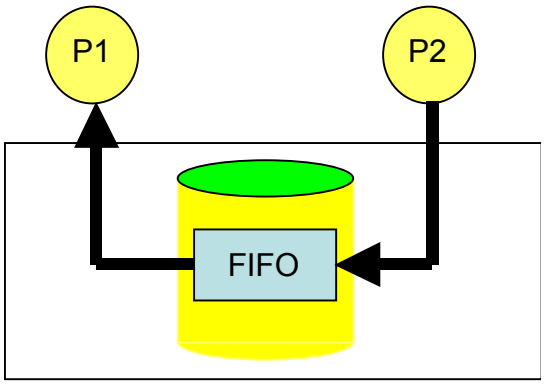


Figure 10.1 Communication between two related or unrelated processes on the same UNIX/Linux machine

Named pipes are created via the `mknod()` system call or `mkfifo()` C library call or by the `mkfifo` command. Here is the synopsis of the mknod() system call.

```
#include <sys/types.h>
#include <sys/stat.h>
int mknod (const char *path, mode_t mode, dev_t dev);
```

The `mknod()` call is normally used for creating special (i.e., device) files but it can be used to create FIFOs too. The 'mode' argument should be permission mode OR-ed with S_IFIFO and 'dev' is set to 0 for creating a FIFO. As is the case with all system calls in UNIX/Linux, `mknod()` returns –1 on failure and *`errno`* is set accordingly. Some of the reasons for this call to fail are:

- File with the given name exists
- Pathname too long
- A component in the pathname not searchable, non-existent, or non-directory
- Destination directory is read-only
- Not enough memory space available
- Signal caught during the execution of `mknod()`

Here is the synopsis of the `mkfifo()` library call.

```
#include <sys/types.h>
#include <sys/stat.h>
int mkfifo (const char *path, mode_t mode)
```

The argument path is for the name and path of the FIFO created, where was the argument mode is for specifying the file permissions for the FIFO. The specification of the mode argument for this function is the same as for the `open()`. Once we have created a FIFO using `mkfifo()`, we open it using `open()`. In fact, the normal file I/O system calls (`close()`, `read()`, `write()`, `unlink()`, etc.) all works with FIFOs. Since `mkfifo()` invokes the `mknod()` system call, the reasons for its failure are pretty much the same as for the `mknod()` call given above.

Unlike a pipe, a FIFO must be opened before using it for communication. A write to a FIFO that no process has opened for reading results in a SIGPIPE signal. When the last process to write to a FIFO closes it, an EOF is sent to the reader. Multiple processes can write to a FIFO are atomic writes to prevent interleaving of multiple writes.

Two common uses of FIFOs are:

- In client-server applications, FIFOs are used to pass data between a server process and client processes
- Used by shell commands to pass data from one shell pipeline to another, without creating temporary files

In client-server software designed for use on the same machine, the server process creates a "well-known" FIFO. Clients communicate send their requests to the server process via the well-known FIFO. Server sends its response to a client via the client-specific FIFO that each client creates and informs the server process about it. Figure 10.2 shows the diagrammatic view of such a software model.

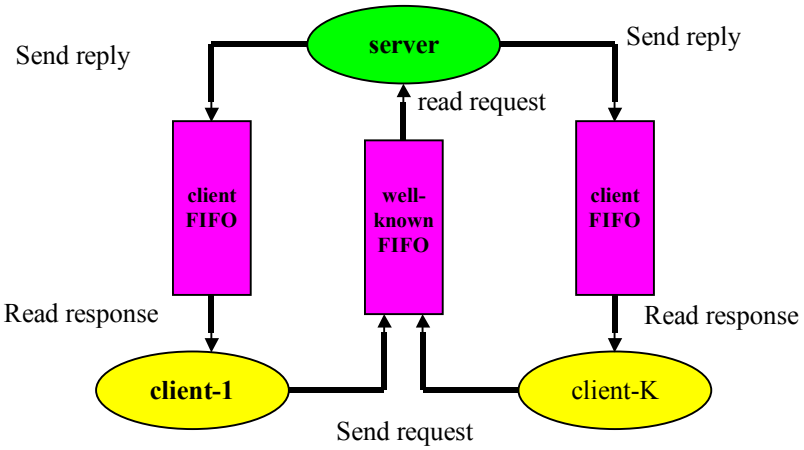


Figure 10.2 Use of FIFOs to implement client-server software on a UNIX/Linux machine

On the command line, `mkfifo` may be used as shown in the following session. As shown in Figure 10.3, the semantics of this session are that prog1 reads its inputs from infile and its output is sent to prog2 and prog3.

```
$ mkfifo fifo1
$ prog3 < fifo1 &
$ prog1 < infile | tee fifo1 | prog2
[ Output ]
$
```

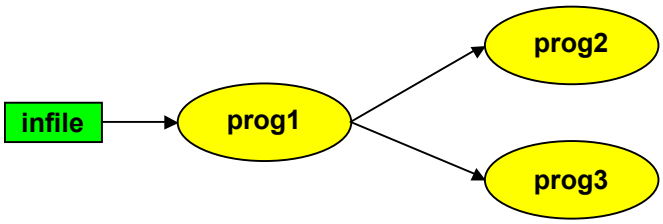


Figure 10.3 Semantics of the above shell session

In the following session, we demonstrate the command line use of FIFOs. The semantics of this session are shown in Figure 10.4. The output of the second command line is the number of lines in the ls.dat file containing ls (i.e., the number of lines in the manual page of the `ls` command containing the string ls) and the output of the third command line is the number of lines in the ls.dat file (i.e., the number of lines in the manual page for the `ls` command).

```
$ man ls > ls.dat
$ cat < fifo1 | grep ls | wc -l &
[1] 21108
$ sort < ls.dat | tee fifo1 | wc -l
  31
  528
$
```

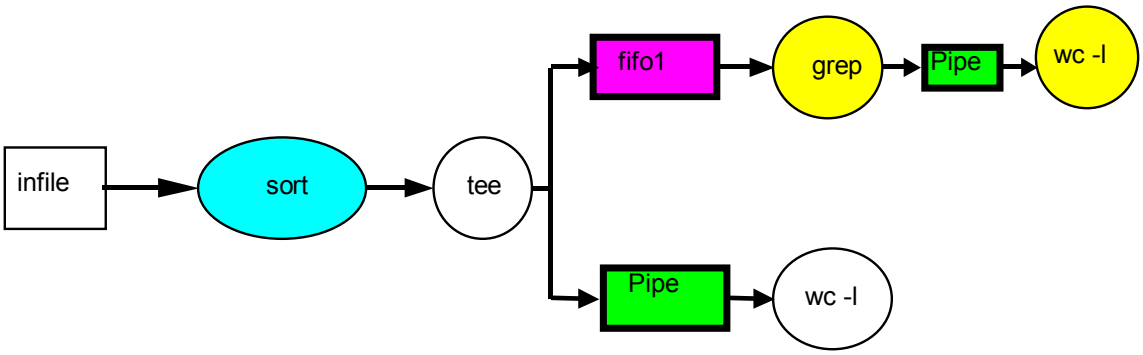


Figure 10.4 Pictorial representation of the semantics of the above shell session

Operating Systems
Lecture No. 11

Reading Material

- UNIX/Linux manual pages for the `mknod()` system call, the `mkfifo()` library call, and the `mkfifo`, `ps`, and `top` commands
- Lecture 11 on Virtual TV

Summary

- More on the use of FIFOs in a program
- Example code for a client-server model
- A few UNIX/Linux process management commands

Use of FIFOs

We continue to discuss the API for using FIFOs for IPC between UNIX/Linux processes. We call these processes client and server. The server process creates two FIFOs, FIFO1 and FIFO2, and opens FIFO1 for reading and FIFO2 for writing. The client process opens FIFO1 for writing and FIFO2 for reading. The client process writes a message to the server process and waits for a response from the server process. The server process reads the message sent by the client process and displays it on the monitor screen. It then sends a message to the client through FIFO2, which the client reads and displays on the monitor screen. The server process then closes the two FIFOs and terminates. The client, after displaying server's message, deletes the two FIFOs and terminates. The protocol for the client-server interaction is shown in Figure 10.1.

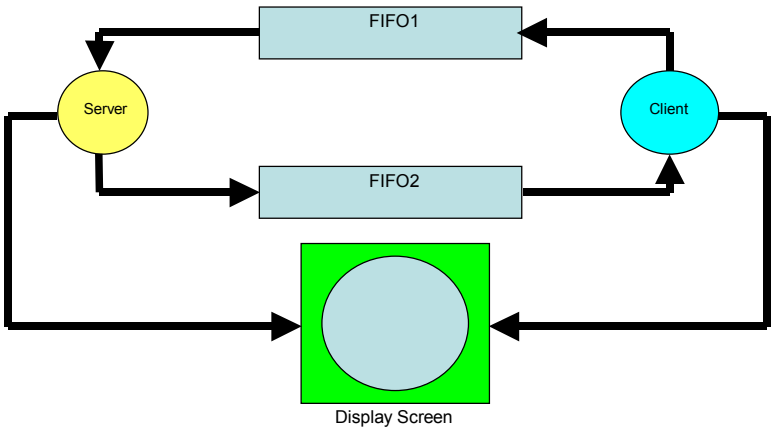


Figure 10.1 Client-server communication using UNIX/Linux FIFOs

The codes for the server and client processes are shown in Figure 10.2 and Figure 10.3, respectively.

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/errno.h>

extern int      errno;

#define FIFO1   "/tmp/fifo.1"
#define FIFO2   "/tmp/fifo.2"
#define PERMS   0666
#define MESSAGE1        "Hello, world!\n"
#define MESSAGE2        "Hello, class!\n"
#include "fifo.h"
main()
{
    char buff[BUFSIZ];
    int readfd, writefd;
    int n, size;

    if ((mknod (FIFO1, S_IFIFO | PERMS, 0) < 0)
                        && (errno != EEXIST)) {
        perror ("mknod FIFO1");
        exit (1);
    }
    if (mkfifo(FIFO2, PERMS) < 0) {
        unlink (FIFO1);
        perror("mknod FIFO2");
        exit (1);
    }
    if ((readfd = open(FIFO1, 0)) < 0) {
        perror ("open FIFO1");
        exit (1);
    }
    if ((writefd = open(FIFO2, 1)) < 0) {
        perror ("open FIFO2");
        exit (1);
    }
    size = strlen(MESSAGE1) + 1;
    if ((n = read(readfd, buff, size)) < 0) {
        perror ("server read"); exit (1);
    }
    if (write (1, buff, n) < n) {
        perror("server write1"); exit (1);
    }
    size = strlen(MESSAGE2) + 1;
    if (write (writefd, MESSAGE2, size) != size) {
        perror ("server write2"); exit (1);
    }
    close (readfd); close (writefd);
}
```

Figure 10.2 Code for the server process

```
#include "fifo.h"
main()
{
    char buff[BUFSIZ];
    int readfd, writefd, n, size;

    if ((writefd = open(FIFO1, 1)) < 0) {
        perror ("client open FIFO1"); exit (1);
    }
    if ((readfd = open(FIFO2, 0)) < 0) {
        perror ("client open FIFO2"); exit (1);
    }
    size = strlen(MESSAGE1) + 1;
    if (write(writefd, MESSAGE1, size) != size) {
        perror ("client write1"); exit (1);
    }
    if ((n = read(readfd, buff, size)) < 0) {
        perror ("client read"); exit (1);
    }
    else
        if (write(1, buff, n) != n) {
            perror ("client write2"); exit (1);
        }
    close(readfd); close(writefd);
     /* Remove FIFOs now that we are done using them */
    if (unlink (FIFO1) < 0) {
        perror("client unlink FIFO1");
        exit (1);
    }
    if (unlink (FIFO2) < 0) {
        perror("client unlink FIFO2");
        exit (1);
    }
    exit (0);
}
```

Figure 10.3 Code for the client process

In the session shown in Figure 10.4, we show how to compile and run the client-server software. We run the server process first so it could create the two FIFOs to be used for communication between the two processes. Note that the server process is run in the background by terminating its command line with an ampersand (&).

```
$ gcc server.c -o server
$ gcc client.c -o client
$ ./server &
[1] 432056
$ ./client
Hello, world!
Hello, class!
$
```

Figure 10.4 Compilation and execution of the client-server software

UNIX/Linux Command for Process Management

We now discuss some of the UNIX/Linux commands for process management, including `ps` and `top`. More commands will be discussed in lecture 12.

`ps` – Display status of processes

`ps` gives a snapshot of the current processes. Without options, ps prints information about processes owned by the user. Some of the commonly used options are `-u`, `-e`, and `-l`.

- `-e` selects all processes
- `-l` formats the output in the long format
- `-u` displays the information in user-oriented format

The shell session in Figure 10.5 shows sample use of the `ps` command. The first command shows the processes running in your current session. The second command shows, page by page, the status of all the processes belonging to root. The last command shows the status of all the processes running on your system.

```
$ ps
  PID TTY          TIME CMD
 1321 pts/0    00:00:00 csh
 1345 pts/0    00:00:00 bash
 1346 pts/0    00:00:00 ps
$ ps -u root | more
  PID TTY          TIME CMD
    1 ?        00:00:04 init
    5 ?        00:00:01 kswapd
  712 ?        00:00:00 inetd
  799 ?        00:00:00 cron
  864 ?        00:00:00 sshd
  934 ?        00:00:00 httpd
 1029 tty1     00:00:00 getty
...
$ ps -e | more
   PID TTY          TIME CMD
     1 ?        00:00:04 init
     2 ?        00:00:00 keventd
     3 ?        00:00:00 ksoftirqd_CPU0
     4 ?        00:00:00 ksoftirqd_CPU1
     5 ?        00:00:01 kswapd
     6 ?        00:00:00 kreclaimd
     7 ?        00:00:00 bdflush
     8 ?        00:00:00 kupdated
...
$
```

Figure 10.5 Use of the `ps` command

top – Display CPU usage of processes

top displays information about the top processes (as many as can fit on the terminal or around 20 by default) on the system and periodically updates this information. Raw CPU percentage is used to rank the processes. A sample run of the top command is shown in Figure 10.6. The output of the command also shows the current time, how long the system has been up and running, number of processes running on the system and their status, number of CPUs in the system and their usage, amount of main memory in the system and its usage, and the size of swap space and its usage. The output also shows a lot of information about each process, including process ID, owner's login name, priority, nice value, and size. Information about processes is updated periodically. See the manual page for the top command for more information by using the man top command.

```
$ top
9:42am  up  5:15,  2 users,  load average: 0.00, 0.00, 0.00
55 processes: 54 sleeping, 1 running, 0 zombie, 0 stopped
CPU0 states:  0.0% user,  0.1% system,  0.0% nice, 99.4% idle
CPU1 states:  0.0% user,  0.2% system,  0.0% nice, 99.3% idle
Mem:   513376K av,  237732K used,  275644K free,      60K shrd,   17944K buff
Swap:  257032K av,       0K used,  257032K free                  106960K cached

  PID USER     PRI  NI  SIZE  RSS SHARE STAT %CPU %MEM   TIME COMMAND
 1406 sarwar    19   0   896  896   700 R     0.3  0.1   0:00 top
 1382 nobody    10   0   832  832   660 S     0.1  0.1   0:00 in.telnetd
    1 root       9   0   536  536   460 S     0.0  0.1   0:04 init
    2 root       9   0     0    0     0 SW    0.0  0.0   0:00 keventd
...
$
```

Figure 10.6 Use of the top command

Operating Systems
Lecture No. 12

Reading Material

- UNIX/Linux manual pages for `fg`, `bg`, `jobs`, and `kill` commands
- Chapter 5 of the textbook
- Lecture 12 on Virtual TV

Summary

- Process Management commands and key presses: `fg`, `bg`, `jobs`, and `kill` commands and `<Ctrl-Z>` and `<Ctrl-C>` command presses
- Thread Concept (thread, states, attributes, etc)

Process Management commands

In the last lecture, we started discussing a few UNIX/Linux process management command. In particular, we discussed the `ps` and `top` commands. We now discuss the `fg`, `bg`, `jobs`, and `kill` commands and `<Ctrl-Z>` and `<Ctrl-C>` key presses.

Moving a process into foreground

You can use the `fg` command to resume the execution of a suspended job in the foreground or move a background job into the foreground. Here is the syntax of the command.

```
fg [%job_id]
```

where, job_id is the job ID (not process ID) of the suspended or background process. If %job_id is omitted, the current job is assumed.

Moving a process into background

You can use the bg command to put the current or a suspended process into the background. Here is the syntax of the command.

```
bg [%job_id]
```

If %job_id is omitted the current job is assumed.

Displaying status of jobs (background and suspended processes)

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

Suspending a process

You can suspend a foreground process by pressing `<Ctrl-Z>`, which sends a STOP/SUSPEND signal to the process. The shell displays a message saying that the job has been suspended and displays its prompt. You can then manipulate the state of this

job, put it in the background with the `bg` command, run some other commands, and then eventually bring the job back into the foreground with the `fg` command.

The following session shows the use of the above commands. The `<Ctrl-Z>` command is used to suspend the `find` command and the `bg` command puts it in the background. We then use the `jobs` command to display the status of jobs (i.e., the background or suspended processes). In our case, the only job is the find command that we explicitly put in the background with the `<Ctrl-Z>` and `bg` commands.

```
$ find / -name foo -print 2> /dev/null
^Z
[1]+  Stopped    find / -name foo -print 2> /dev/null
$ bg
[1]+ find / -name foo -print 2> /dev/null &
$ jobs
[1]+  Running    find / -name foo -print 2> /dev/null &
$ fg
find / -name foo -print 2> /dev/null
[ command output ]
$
```

Terminating a process

You can terminate a foreground process by pressing `<Ctrl-C>`. Recall that this key press sends the SIGINT signal to the process and the default action is termination of the process. Of course, if your foreground process intercepts SIGINT and ignores it, you cannot terminate it with `<Ctrl-C>`. In the following session, we terminate the find command with `<Ctrl-C>`.

```
$ find / -name foo -print 1> out 2> /dev/null
^C
$
```

You can also terminate a process with the kill command. When executed, this command sends a signal to the process whose process ID is specified in the command line. Here is the syntax of the command.

```
kill [-signal] PID
```

where, 'signal' is the signal number and PID is the process ID of the process to whom the specified signal is to be sent. For example, `kill -2 1234` command sends signal number 2 (which is also called SIGINT) to the process with ID 1234. The default action for a signal is termination of the process identified in the command line. When executed without a signal number, the command sends the SIGTERM signal to the process. A process that has been coded to intercept and ignore a signal, can be terminated by sending it the 'sure kill' signal, SIGKILL, whose signal number is 9, as in `kill -9 1234`.

You can display all of the signals supported by your system, along with their numbers, by using the `kill -l` command. On some systems, the signal numbers are not displayed. Here is a sample run of the command on Solaris 2.

```
$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGEMT       8) SIGFPE
 9) SIGKILL     10) SIGBUS      11) SIGSEGV     12) SIGSYS
13) SIGPIPE     14) SIGALRM     15) SIGTERM     16) SIGUSR1
...
$
```

The Thread Concept

There are two main issues with processes:

1. The `fork()` system call is expensive (it requires memory to memory copy of the executable image of the calling process and allocation of kernel resources to the child process)
2. An inter-process communication channel (IPC) is required to pass information between a parent process and its children processes.

These problems can be overcome by using threads.

A thread, sometimes called a **lightweight process (LWP)**, is a basic unit of CPU utilization and executes within the address space of the process that creates it. It comprises a thread ID, a program counter, a register set, errno, and a stack. It shares with other threads belonging to the same process its code sections, data section, current working directory, user and group IDs, signal setup and handlers, PCB and other operating system resources, such as open files and system. A traditional (heavy weight) process has a single thread of control. If a process has multiple threads of control, it can do more than one task at a time. Figure 12.1 shows processes with single and multiple threads. Note that, as stated above, threads within a process share code, data, and open files, and have their own register sets and stacks.

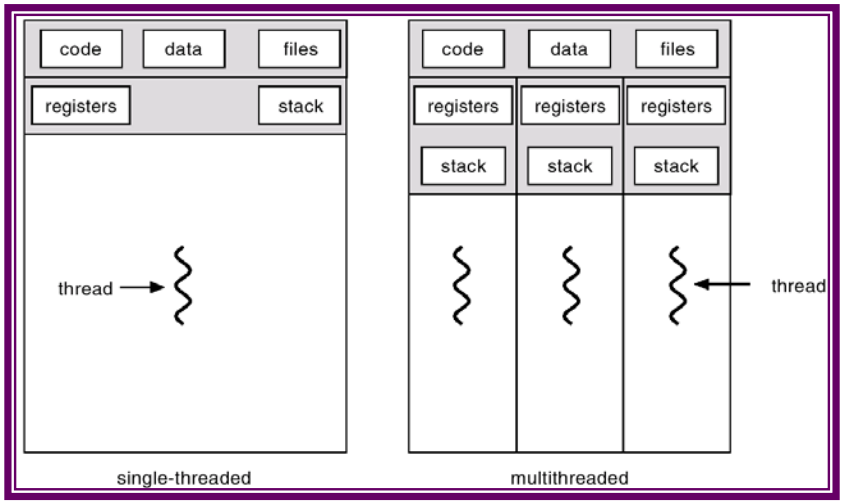


Figure 12.1 Single- and multi-threaded processes

In Figure 12.2, we show the code structure for a sequential (single-threaded) process and how the control thread moves from the main function to the f1 function and back, and from f1 to main and back. The important point to note here is that there is just one thread of control that moves around between various functions.

```
main()
{

     f1(

     f2(…);
     …
}
f1(…)
{  …  }
f2(…)
{  …  }
```

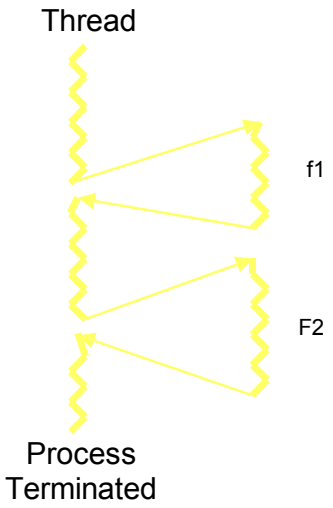


Figure 12.2 Code structure of a single-threaded (sequential) process

In Figure 12.3, we show the code structure for a multi-threaded process and how multiple threads of control are active simultaneously. We use hypothetical function `thread()` to create a thread. This function takes two arguments: the name of a function for which a thread has to be created and a variable in which the ID of the newly created thread is to be stored. The important point to note here is that multiple threads of control are simultaneously active within the same process; each thread steered by its own PC.

```
main()
{
     …
     thread(t1,f1);
     …
     thread(t2,f2);
     …
}
f1(…)
{  …  }
f2(…)
{  …  }
```

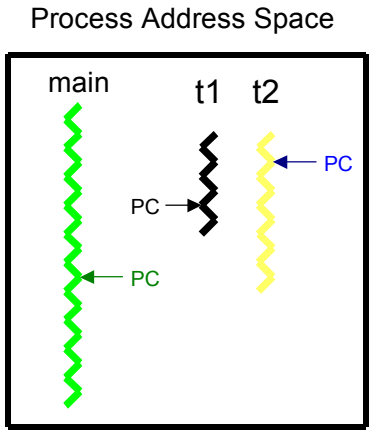


Figure 12.3 Code structure of a multi-threaded process

The Advantages and Disadvantages of Threads

Four main advantages of threads are:

1. Responsiveness: Multithreading an interactive application may allow a program to continue running even if part of it is blocked or is performing a lengthy operation, thereby increasing responsiveness to the user.
2. Resource sharing: By default, threads share the memory and the resources of the process to which they belong. Code sharing allows an application to have several different threads of activity all within the same address space.
3. Economy: Allocating memory and resources for process creation is costly. Alternatively, because threads share resources of the process to which they belong, it is more economical to create and context switch threads.
4. Utilization of multiprocessor architectures: The benefits of multithreading of multithreading can be greatly increased in a multiprocessor environment, where each thread may be running in parallel on a different processor. A single threaded process can only run on one CPU no matter how many are available. Multithreading on multi-CPU machines increases concurrency.

Some of the main disadvantages of threads are:

1. Resource sharing: Whereas resource sharing is one of the major advantages of threads, it is also a disadvantage because proper synchronization is needed between threads for accessing the shared resources (e.g., data and files).
2. Difficult programming model: It is difficult to write, debug, and maintain multi-threaded programs for an average user. This is particularly true when it comes to writing code for synchronized access to shared resources.

Operating Systems
Lecture No. 13

Reading Material

- UNIX/Linux manual pages `pthread_create()`, `pthread_join()`, and `pthread_exit()` calls
- Chapter 5 of the textbook
- Lecture 13 on Virtual TV

Summary

- User- and Kernel –level threads
- Multi-threading models
- Solaris 2 threads model
- POSIX threads (the pthread library)
- Sample code

User and Kernel Threads

Support for threads may be provided at either user level for *user threads* or by kernel for *kernel threads*.

User threads are supported above kernel and are implemented by a thread library at the user level. The library provides support for thread creation, scheduling, and management with no support from the kernel. Since the kernel is unaware of user-level threads, all thread creation and scheduling are done in the user space without the need for kernel intervention, and therefore are fast to create and manage. If the kernel is single threaded, then any user level thread performing a blocking system call will cause the entire process to block, even if other threads are available to run within the application. User thread libraries include POSIX Pthreads , Solaris 2 UI-threads, and Mach C-threads.

Kernel threads are supported directly by the operating system. The kernel performs the scheduling, creation, and management in kernel space; the kernel level threads are hence slower to create and manage, compared to user level threads. However since the kernel is managing threads, if a thread performs a blocking system call, the kernel can schedule another thread in the application for execution. Windows NT, Windowss 2000, Solaris, BeOS and Tru64 UNIX support kernel threads.

Multi-threading Models

There are various models for mapping user-level threads to kernel-level threads. We describe briefly these models, their main characteristics, and examples.

1. **Many-to-One**: In this model, many user-level threads are supported per kernel thread, as shown in Figure 13.1. Since only one kernel-level thread supports many user threads, there is no concurrency. This means that a process blocks when a thread makes a system call. Examples of these threads are Solaris Green threads POSIX Pthreads.

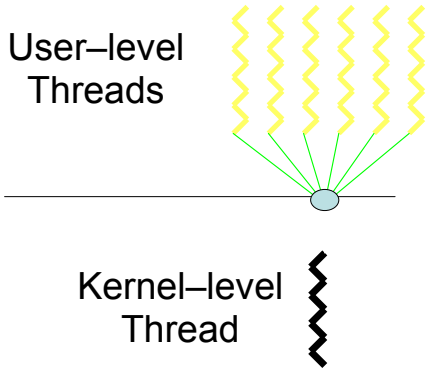


Figure 13.1 Many –to-One Model

2. **One-to-One**: In this model, there is a kernel thread for every user thread, as shown in Figure 13.2. Thus, this model provides true concurrency. This means that a process does not block when a thread makes a system call. The main disadvantage of this model is the overhead of creating a kernel thread per user thread. Examples of these threads are WindowsNT, Windows 2000, and OS/2.

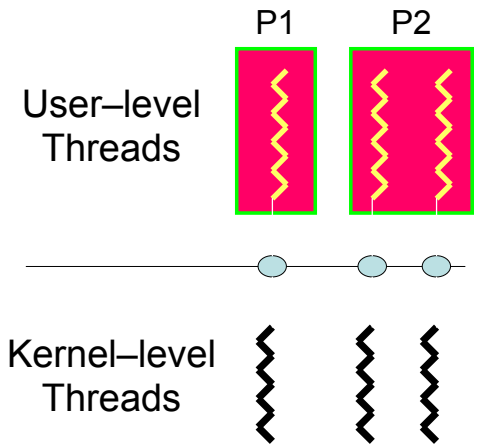


Figure 13.2 One-to-One Model

3. **Many-to-One**: In this model, multiple user-level threads are multiplexed over a smaller or equal number of kernel threads, as shown in Figure 13.2. Thus, true concurrency is not achieved through this model. Examples of these threads are Solais 2 and HP-UX.

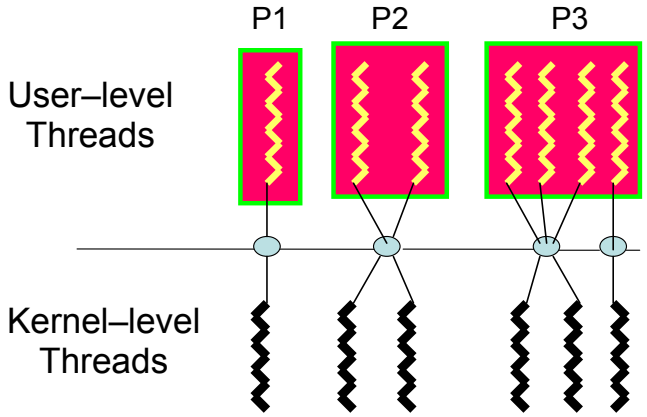


Figure 13.3 Many-to Many Model

Solaris 2 Threads Model

Solaris 2 has threads, lightweight processes (LWPs), and processes, as shown in Figure 13.4. At least one LWP is assigned to every user process to allow a user thread to talk to a kernel thread. User level threads are switched and scheduled among LWPs without kernel's knowledge. One kernel thread is assigned per LWP. Some kernel threads have no LWP associated with them because these threads are not executed for servicing a request by a user-level thread. Examples of such kernel threads are clock interrupt handler, swapper, and short-term (CPU) shceduler.

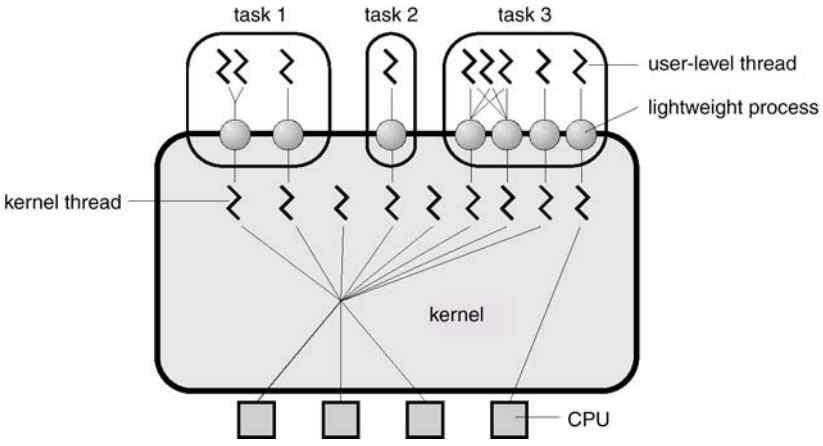


Figure 13.4 Solaris 2 Threads Model

POSIX Threads (the pthread library)

Pthreads refers to the POSIX standard defining an API for thread creation, scheduling, and synchronization. This is a specification for thread behavior not an implementation. OS designers may implement the specification in any way they wish. Generally, libraries implementing the Pthreads specification are restricted to UNIX-based systems such as Solaris 2. In this section, we discuss the Pthreads library calls for creating, joining, and terminating threads and use these calls to write small multi-threaded C programs.

Creating a Thread

You can create a threads by using the `pthread_create()` call. Here is the syntax of this call.

```
int pthread_create(pthread_t *threadp, const pthread_attr_t *attr,
                   void* (*routine)(void *), arg *arg);
```

where, 'threadp' contains thread ID (TID) of the thread created by the call, 'attr' is used to modify the thread attributes (stack size, stack address, detached, joinable, priority, etc.), 'routine' is the thread function, and 'arg' is any argument we want to pass to the thread function. The argument does not have to be a simple native type; it can be a 'struct' of whatever we want to pass in.

The `pthread_create()` call fails and returns the corresponding value if any of the following conditions is detected:

- **EAGAIN** The system-imposed limit on the total number of threads in a process has been exceeded or some system resource has been exceeded (for example, too many LWPs were created).
- **EINVAL** The value specified by 'attr' is invalid.
- **ENOMEM** Not enough memory was available to create the new thread.

You can do error handling by including the <errno.h> file and incorporating proper error handling code in your programs.

Joining a Thread

You can have a thread wait for another thread within the same process by using the pthread_join() call. Here is the syntax of this call.

```
int pthread_join(pthread_t aThread, void **statusp);
```

where, 'aThread' is the thread ID of the thread to wait for and 'statusp' gets the return value of `pthread_exit()` call made in the process for whom wait is being done.

A thread can only wait for a joinable thread in the same process address space; a thread cannot wait for a detached thread. Multiple threads can join with a thread but only one returns successfully; others return with an error that no thread could be found with the given TID

Terminating a Thread

You can terminate a thread explicitly by either returning from the thread function or by using the `pthread_exit()` call. Here is the syntax of the `pthread_exit()` call.

```
void pthread_exit(void *valuep);
```

where, 'valuep' is a pointer to the value to be returned to the thread which is waiting for this thread to terminate (i.e., the thread which has executed `pthread_join()` for this thread).

A thread also terminates when the main thread in the process terminates. When a thread terminates with the exit() system call, it terminates the whole process because the purpose of the exit() system call is to terminate a process and not a thread.

Sample Code

The following code shows the use of the pthread library calls discussed above. The program creates a thread and waits for it. The child thread displays the following message on the screen and terminates.

```
Hello, world! ... The threaded version.
```

As soon as the child thread terminates, the parent comes out of wait, displays the following message and terminates.

```
Exiting the main function.
```

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
/* Prototype for a function to be passed to our thread */
void* MyThreadFunc(void *arg);
int main()
{
   pthread_t aThread;
   /* Create a thread and have it run the MyThreadFunction */
   pthread_create(&aThread, NULL, MyThreadFunc, NULL);
   /* Parent waits for the aThread thread to exit */
   pthread_join(aThread, NULL);
   printf ("Exiting the main function.\n");
   return 0;
}
void* MyThreadFunc(void* arg)
{
        printf ("Hello, world! ... The threaded version.\n");
        return NULL;
}
```

The following session shows compilation and execution of the above program. Does the output make sense to you?

```
$ gcc hello.c –o hello –lpthread –D_REENTRANT
$ hello
Hello, world! ... The threaded version.
Exiting the main function.
$
```

Note that you need to take the following steps in order to be able to use the pthread library.

1. Include <pthread.h> in your program
2. Link the pthread library with your program (by using the –lpthread option in the compiler command)
3. Pass the _REENTRANT macro from the command line (or define it in your program)

Here is another program that uses the pthread library to create multiple threads and have them display certain messages. Read through the code to understand what it does. Then compile and run it on your UNIX/Linux system to display output of the program and to see if you really understood the code.

```
/******************************************************************************
* FILE: hello_arg2.c
* DESCRIPTION:
*   A "hello world" Pthreads program which demonstrates another safe way
*   to pass arguments to threads during thread creation.  In this case,
*   a structure is used to pass multiple arguments.
*
* LAST REVISED: 09/04/02 Blaise Barney
******************************************************************************/
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS   8

char *messages[NUM_THREADS];

struct thread_data
{
  int       thread_id;
  int  sum;
  char *message;
};

struct thread_data thread_data_array[NUM_THREADS];

void *PrintHello(void *threadarg)
{
  int taskid, sum;
  char *hello_msg;
  struct thread_data *my_data;

  sleep(1);
  my_data = (struct thread_data *) threadarg;
  taskid = my_data->thread_id;
  sum = my_data->sum;
  hello_msg = my_data->message;
  printf("Thread %d: %s  Sum=%d\n", taskid, hello_msg, sum);
  pthread_exit(NULL);
}

int main(int argc, char *argv[])
{
```

```
  pthread_t threads[NUM_THREADS];
  int *taskids[NUM_THREADS];
  int rc, t, sum;

  sum=0;
  messages[0] = "English: Hello World!";
  messages[1] = "French: Bonjour, le monde!";
  messages[2] = "Spanish: Hola al mundo";
  messages[3] = "Klingon: Nuq neH!";
  messages[4] = "German: Guten Tag, Welt!";
  messages[5] = "Russian: Zdravstvytye, mir!";
  messages[6] = "Japan: Sekai e konnichiwa!";
  messages[7] = "Latin: Orbis, te saluto!";

  for(t=0; t<NUM_THREADS; t++) {
    sum = sum + t;
    thread_data_array[t].thread_id = t;
    thread_data_array[t].sum = sum;
    thread_data_array[t].message = messages[t];
    printf("Creating thread %d\n", t);
    rc = pthread_create(&threads[t], NULL, PrintHello, (void *) &thread_data_array[t]);
    if (rc) {
      printf("ERROR; return code from pthread_create() is %d\n", rc);
      exit(-1);
    }
  }
  pthread_exit(NULL);
}
```

Reference

The above code was taken from the following website.

`http://www.llnl.gov/computing/tutorials/pthreads/samples/hello_arg2.c`

Operating Systems
Lecture No. 14

Reading Material

- Chapter 6 of the textbook
- Lecture 14 on Virtual TV

Summary

- Basic concepts
- Scheduling criteria
- Preemptive and non-preemptive algorithms
- First-Come-First-Serve scheduling algorithm

Basic Concepts

The objective of multiprogramming is to have some process running at all times, in order to maximize CPU utilization. In a uniprocessor system, only one process may run at a time; any other processes much wait until the CPU is free and can be rescheduled.
In multiprogramming, a process is executed until it must wait, typically for the completion of some I/O request. In a simple computer system, the CPU would then sit idle; all this waiting time is wasted. Multiprogramming entails productive usage of this time. When one process has to wait, the OS takes the CPU away from that process and gives the CPU to another process. Almost all computer resources are scheduled before use.

Life of a Process

As shown in Figure 14.1, process execution consists of a cycle of CPU execution and I/O wait. Processes alternates between these two states. Process execution begins with a **CPU burst.** An I/O burst follows that, and so on**.** Eventually, the last CPU burst will end with a system request to terminate execution, rather than with another I/O burst.

An I/O bound program would typically have many very short CPU bursts. A CPU-bound program might have a few very long CPU bursts. This distribution can help us select an appropriate CPU-scheduling algorithm. Figure 14.2 shows results on an empirical study regarding the CPU bursts of processes. The study shows that most of the processes have short CPU bursts of 2-3 milliseconds.

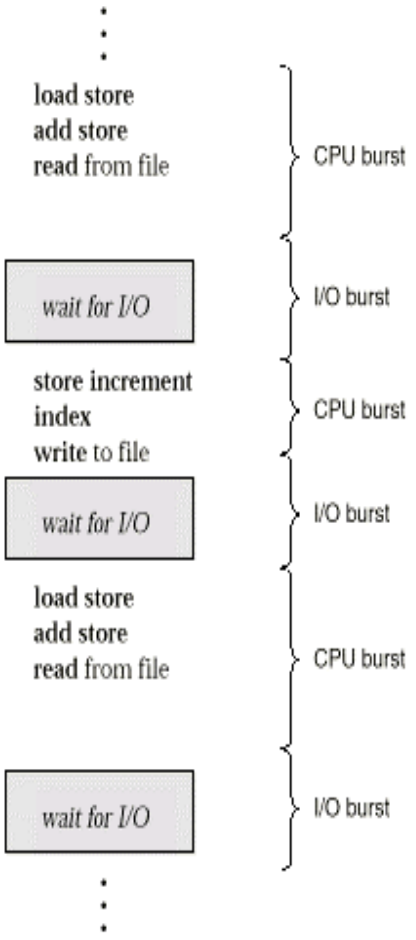


Figure 14.1 Alternating Sequence of CPU and I/O Bursts

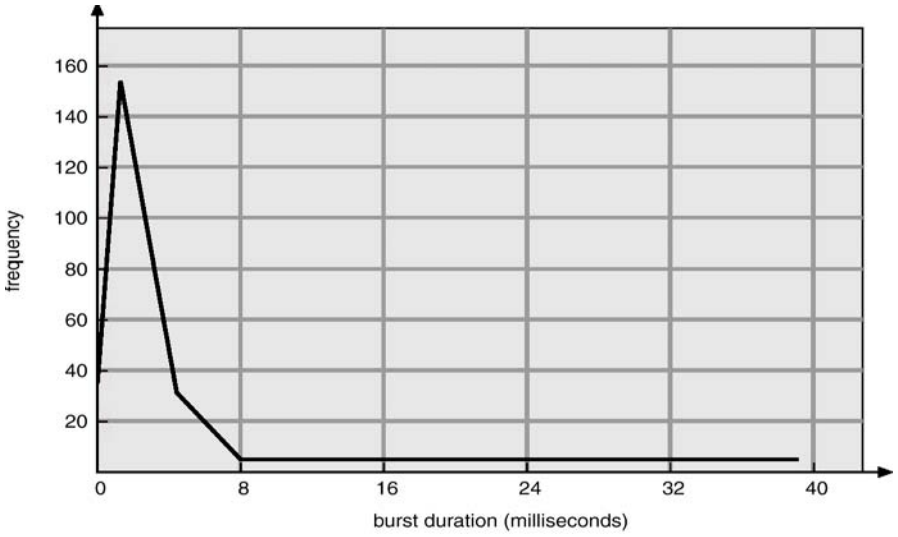


Figure 14.2 Histogram of CPU-burst Times

CPU Scheduler

Whenever the CPU becomes idle, the operating system must select one of the processes in the ready queue to be executed. The **short-term scheduler** (i.e., the CPU scheduler) selects a process to give it the CPU. It selects from among the processes in memory that are ready to execute, and invokes the dispatcher to have the CPU allocated to the selected process.

A ready queue can be implemented as a FIFO queue, a tree, or simply an unordered linked list. The records (nodes) in the ready queue are generally the process control blocks (PCBs) of processes.

Dispatcher

The **dispatcher** is a kernel module that takes control of the CPU from the current process and gives it to the process selected by the short-term scheduler. This function involves:

- Switching the context (i.e., saving the context of the current process and restoring the context of the newly selected process)
- Switching to user mode
- Jumping to the proper location in the user program to restart that program

The time it takes for the dispatcher to stop one process and start another running is known as the **dispatch latency**.

Preemptive and Non-Preemptive Scheduling

CPU scheduling can take place under the following circumstances:

1. When a process switches from the running state to the waiting state (for example, an I/O request is being completed)
2. When a process switches from the running state to the ready state (for example when an interrupt occurs)
3. When a process switches from the waiting state to the ready state (for example, completion of I/O)
4. When a process terminates

In 1 and 4, there is no choice in terms of scheduling; a new process must be selected for execution. There is a choice in case of 2 and 3. When scheduling takes place only under 1 and 4, we say, scheduling is **non-preemptive**; otherwise the scheduling scheme is **preemptive**. Under non-preemptive scheduling once the CPU has been allocated to a process the process keeps the CPU until either it switches to the waiting state, finishes its CPU burst, or terminates. This scheduling method does not require any special hardware needed for preemptive scheduling.

Preemptive scheduling incurs a cost. Consider the case of two processes sharing data. One may be in the midst of updating the data when it is preempted and the second process is run. The second process may try to read the data, which are currently in an inconsistent state. New mechanisms are needed to coordinate access to shared data. We discuss this topic in Chapter 7 of the textbook.

Scheduling Criteria

The scheduling criteria include:

- **CPU utilization**: We want to keep CPU as busy as possible. In a real system it should range from 40 percent (for a lightly loaded system) to 90 percent (for a heavily used system)
- **Throughput:** If CPU is busy executing processes then work is being done. One measure of work is the number of processes completed per time, called, **throughput.** We want to maximize the throughput.
- **Turnaround time:** The interval from the time of submission to the time of completion is the **turnaround time.** Turnaround time is the sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU and doing I/O. We want to minimize the turnaround time.
- **Waiting time:** Waiting time is the time spent waiting in the ready queue. We want to minimize the waiting time to increase CPU efficiency.
- **Response time:** It is the time from the submission of a request until the first response is produced. Thus response time is the amount of time it takes to start responding but not the time it takes to output that response. Response time should be minimized.

Scheduling Algorithms

We will now discuss some of the commonly used short-term scheduling algorithms. Some of these algorithms are suited well for batch systems and others for time-sharing systems. Here are the algorithms we will discuss:

- First-Come-First-Served (FCFS) Scheduling
- Shorted Job First (SJF) Scheduling
- Shortest Remaining Time First (SRTF) Scheduling
- Priority Scheduling
- Round-Robin Scheduling
- Multilevel Queues Scheduling
- Multilevel Feedback Queues Scheduling
- UNIX System V Scheduling

First-Come, First-Served (FCFS) Scheduling

The process that requests the CPU first (i.e., enters the ready queue first) is allocated the CPU first. The implementation of an FCFS policy is managed with a FIFO queue. When a process enters the ready queue, its PCB is linked onto the tail of the queue. When CPU is free, it is allocated to the process at the head of the queue. The running process is removed from the queue. The average waiting time under FCFS policy is not minimal and may vary substantially if the process CPU-burst times vary greatly. FCFS is a non-preemptive scheduling algorithm.

We use the following system state to demonstrate the working of this algorithm. For simplicity, we assume that processes are in the ready queue at time 0.

Process	Burst Time
P1	24
P2	3
P3	3

Suppose that processes arrive into the system in the order: P1, P2, P3. Processes are served in the order: P1, P2, P3.The **Gantt chart** for the schedule is shown in Figure 14.3.

Figure 14.3 Gantt chart showing execution of processes in the order P1, P2, P3

Here are the waiting times for the three processes and the average waiting time per process.

- Waiting times P1 = 0; P2 = 24; P3 = 27
- Average waiting time: (0+24+27)/3 = 17

Suppose that processes arrive in the order: P2, P3, P1. The Gantt chart for the schedule is shown in Figure 14.4:

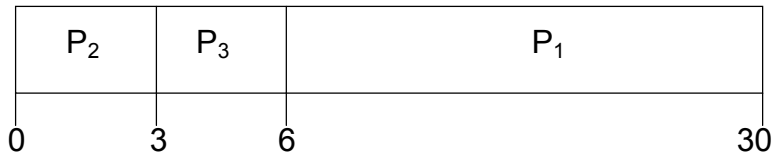


Figure 14.4 Gantt chart showing execution of processes in the order P2, P3, P1

Here are the waiting times for the three processes and the average waiting time per process.

- Waiting time for P1 = 6; P2 = 0; P3 = 3
- Average waiting time: (6 + 0 + 3)/3 = 3

When FCFS scheduling algorithm is used, the **convoy effect** occurs when short processes wait behind a long process to use the CPU and enter the ready queue in a convoy after completing their I/O. This results in lower CPU and device utilization than might be possible if shorter processes were allowed to go first.

In the next lecture, we will discuss more scheduling algorithms.

Operating Systems
Lecture No. 15

Reading Material

- Chapter 6 of the textbook
- Lecture 15 on Virtual TV

Summary

- Scheduling algorithms

Shortest-Job-First Scheduling

This algorithm associates with ach process the length of the latter's next CPU burst. When the CPU is available, it is assigned to the process that has the smallest next CPU burst. If two processes have the same length next CPU burst, FCFS scheduling is used to break the tie. The real difficulty with the SJF algorithm is in knowing the length of the next CPU request. For long term scheduling in a batch system, we can use as the length the process time limit that a user specifies when he submits the job.

For short-term CPU scheduling, there is no way to length of the next CPU burst. One approach is to try to approximate SJF scheduling, by assuming that the next CPU burst will be similar in length to the previous ones, for instance.

The next CPU burst is generally predicted as an exponential average of the measured lengths of previous CPU bursts. Let t_n be the length of the nth CPU burst and let τ_{n+1} be our predicted value for the next CPU burst. We define τ_{n+1} to be

$$\tau_{n+1} = \alpha\, t_n + (1- \alpha)\, \tau_n$$

where, $0 \leq \alpha \leq 1$. We discuss this equation in detail in a subsequent lecture.

The SJF algorithm may either be preemptive or non-preemptive. The choice arises when a new process arrives at the ready queue while a previous process is executing. The new process may have a shorter next CPU burst than what is left of the currently executing process. A preemptive SJF algorithm preempts the currently executing process, whereas a non-preemptive SJF algorithm will allow the currently running process to finish its CPU burst. Preemptive SJF scheduling is sometimes called **shortest-remaining-time-first** scheduling.

We illustrate the working of the SJF algorithm by using the following system state.

Process	Arrival Time	Burst Time
P1	0.0	7
P2	2.0	4
P3	4.0	1
P4	5.0	4

The Gantt chart for the execution of the four processes using SJF is shown in Figure 15.1.

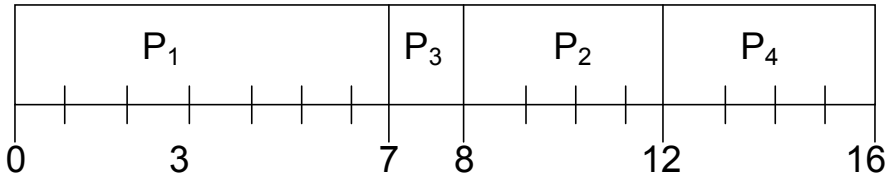


Figure 15.1 Gantt chart showing execution of processes using SJF

Here is the average waiting time per process.

- Average waiting time = (0 + 6 + 3 + 7)/4 = 4 time units

We illustrate the working of the SRTF algorithm by using the system state shown above. The Gantt chart for the execution of the four processes using SRTF is shown in Figure 15.2.

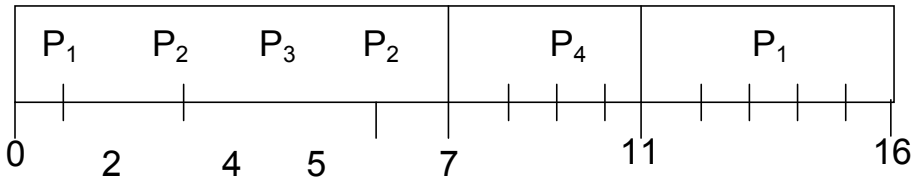


Figure 15.2 Gantt chart showing execution of processes using SRTF

Here is the average waiting time per process.

- Average waiting time = (9 + 1 + 0 +2)/4 = 3 time units

Priority Scheduling

SJF is a special case of the general **priority-scheduling algorithm**. A priority is associated with each process, and the CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority). Equal priority processes are scheduled in FCFS order. The SJF algorithm is simply a priority algorithm where the priority (p) is the inverse of the (predicted) next CPU burst. The larger the CPU burst of a process, the lower its priority, and vice versa.

Priority scheduling can either be preemptive or non-preemptive. When a process arrives at the ready queue, its priority is compared with the priority of the currently running process. A preemptive priority-scheduling algorithm will preempt the CPU if the priority of the newly arrived process is higher than the priority of the currently running process. A non-preemptive priority- scheduling algorithm will simply put the new process at the head of ready queue.

A major problem with priority- scheduling algorithms is **indefinite blocking** (or **starvation**). A process that is ready to run but lacking the CPU can be considered blocked-waiting for the CPU. A priority-scheduling algorithm can leave some low priority processes waiting indefinitely for the CPU. Legend has it that when they were phasing out IBM 7094 at MIT in 1973, they found a process stuck in the ready queue since 1967!

Aging is solution to the problem of indefinite blockage of low-priority processes. It involves gradually increasing the priority of processes that wait in the system for a long time. For example, if priority numbers range from 0 (high priority) to 127 (high priority), we could decrement priority of every process periodically (say every 10 minutes). This would result in every process in the system eventually getting the highest priority in a reasonably short amount of time and scheduled to use the CPU.

Operating Systems
Lecture No. 16

Reading Material

- Chapter 6 of the textbook
- Lecture 16 on Virtual TV

Summary

- Scheduling algorithms

Why is SJF optimal?

SJF is an optimal algorithm because it decreases the wait times for short processes much more than it increases the wait times for long processes. Let's consider the example shown in Figure 16.1, in which the next CPU bursts of P1, P2, and P3 are 5, 3, and 2, respectively. The first Gantt chart shows execution of processes according to the longest-job-first algorithm, resulting in the waiting times for P1, P2, and P3 to be 0, 5, and 8 times units. The second Gantt chart shows execution of processes according to the shortest-job-first algorithm, resulting in the waiting times for P1, P2, and P3 to be 0, 2, and 5. Note that the waiting time for P2 has decreased from 5 to 2 and that of P3 has decreased from 8 to 0. The increase in the wait time for P1 is from 0 to 5, which is much smaller than the decrease in the wait times for P2 and P3.

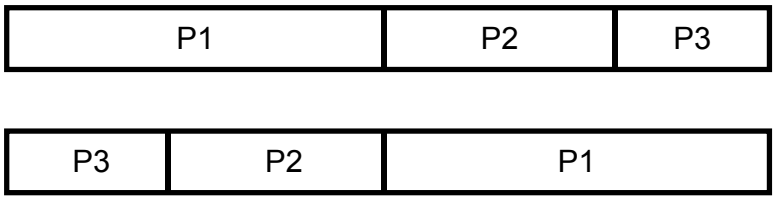


Figure 16.1 Two execution sequences for P1, P2, and P3: longest-job-first and shortest-job-first

Round-Robin Scheduling

The **round-robin (RR) scheduling algorithm** is designed especially for time-sharing systems. It is similar to FCFS scheduling but preemption is added to switch between processes. A small unit of time, called a **time quantum** (or **time slice**) is defined. The ready queue is treated as a circular queue. The CPU scheduler goes around the ready queue, allocating the CPU to each process for a time interval of up to 1 time quantum. To implement RR scheduling, we keep ready queue as a FIFO queue of processes. New processes are added to the tail of the ready queue. The CPU scheduler picks the first process from the ready queue, sets a timer to interrupt after 1 time quantum, and then dispatches the process. One of the two things will then happen. The process may have a CPU burst of less than 1 time quantum, in which case the process itself will release the CPU voluntarily. The scheduler will then proceed to the next process in the ready queue. Otherwise, if the CPU burst of currently running process is longer than one time

quantum, the timer will go off and will cause an interrupt to the operating system. A context switch will happen, the current process will be put at the tail of the ready queue, and the newly scheduled process will be given the CPU.

The average waiting time under the RR policy however is often quite long. It is a preemptive scheduling algorithm. If there are *n* processes n the ready queue, context switch time is t_{cs} and the time quantum is *q* then each process gets 1/*n* of the CPU time in chunks of at most *q* time units. Each process must wait no longer than $(n-1)*(q+t_{cs})$ time units until its next time quantum.

The performance of RR algorithm depends heavily on the size of the time quantum. If the time quantum is very large (infinite), the RR policy remains the same as the FCFS policy. If the time quantum is very small, the RR approach is called the **processor sharing** and appears to the users as though each of *n* processes has its own processor running at 1/*n* the speed of real processor (q must be large with respect to context switch, otherwise the overhead is too high). The drawback of small quantum is more frequent context switches. Since context switching is the cost of the algorithm and no useful work is done for any user process during context switching, the number of context switches should be minimized and the quantum should be chosen such that the ratio of a quantum to context switching is not less than 10:1 (i.e., context switching overhead should not be more than 10% of the time spent on doing useful work for a user process). Figure 16.2 shows increase in the number of context switches with decrease in quantum size.

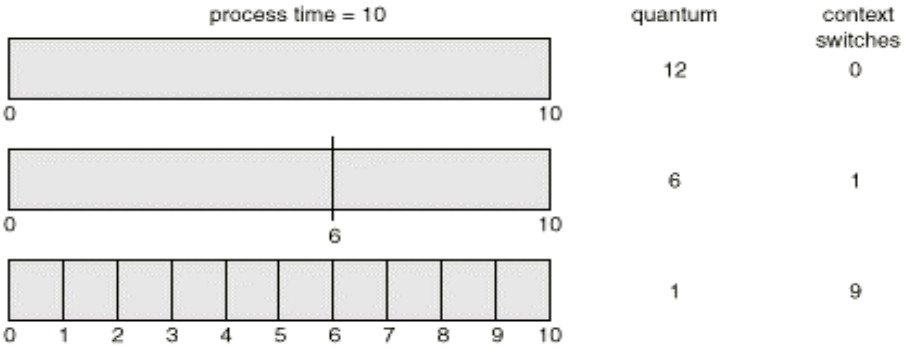
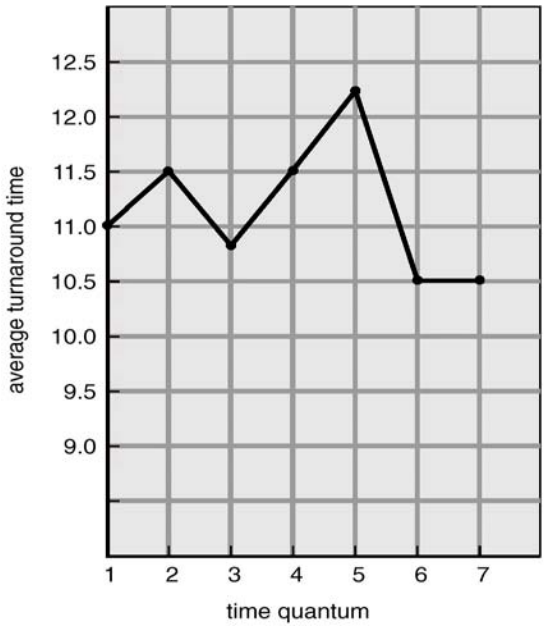


Figure 16.2 Quantum size versus number of context switches

The turnaround time of a process under round robin is also depends on the size of the time quantum. In Figure 16.3 we show a workload of four processes P1, P2, P3, and P4 with their next CPU bursts as 6, 3, 1, and 7 time units. The graph in the figure shows that best (smallest) turnaround time is achieved when quantum size is 6 or greater. Note that most of the given processes finish their next CPU bursts with quantum of 6 or greater. We can make a general statement that the round-robin algorithm gives smallest average turnaround time when quantum value is chosen such that most of the processes finish their next CPU bursts within the quantum.



process	time
P_1	6
P_2	3
P_3	1
P_4	7

Figure 16.3 Turnaround time versus quantum size

We now consider the following system workload to illustrate working of the round-robin algorithm. Execution of P1 though P4 with quantum 20 is shown in Figure 16.4. In the table, original CPU bursts are shown in bold and remaining CPU bursts (after a process has used the CPU for one quantum) are shown in non-bold font.

Process	Burst Time
P1	**53** — 33 — 13
P2	**17**
P3	**68** — 48 — 28 — 8
P4	**24** — 4

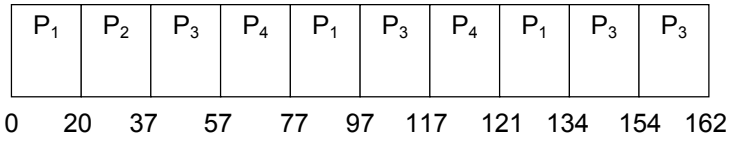


Figure 16.4 Gantt chart showing execution of P1, P2, P3, and P4 with quantum 20 time units

Figure 16.5 shows wait and turnaround times for the four processes. The average wait time for a process comes out to be 73 time units for round robin and 38 for SJF. Typically, RR has a higher average turnaround than SJF, but better response. In time-sharing systems, shorter response time for a process is more important than shorter turnaround time for the process. Thus, round-robin scheduler matches the requirements of time-sharing systems better than the SJF algorithm. SJF scheduler is better suited for batch systems, in which minimizing the turnaround time is the main criterion.

Process	Turnaround Time	Waiting Time
P1	134	134 – 53 = **81**
P2	37	37 – 17 = **20**
P3	162	162 – 68 = **94**
P4	121	121 – 24 = **97**

Figure 16.5 Wait and turnaround times for processes

Multilevel Queue Scheduling

Another class of scheduling algorithms has been created for situations in which processes are easily classified into different groups. For example, a common division is made between **foreground** (or **interactive**) processes and **background** (or **batch) processes**. These two types of processes have different response time requirements and so might have different scheduling needs. In addition, foreground processes may have priority over background processes.

A **multilevel queue-scheduling algorithm** partitions the ready queue into several separate queues, as shown in Figure 16.5. Each queue has its own priority and scheduling algorithm. Processes are permanently assigned to one queue, generally based o some property of the process, such as memory size, process priority or process type. In addition, there must be scheduling among the queues, which is commonly implemented as fixed-priority preemptive scheduling i.e., serve all from foreground then from background. Another possibility is to time slice between queues. Each queue gets a certain portion of the CPU time, which it can then schedule among the various processes in its queue, e.g., 80% to foreground in RR and 20% to background in FCFS. Scheduling across queues prevents starvation of processes in lower-priority queues.

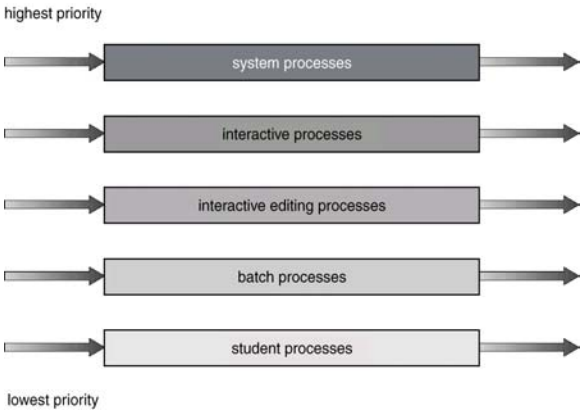


Figure 16.5 Multilevel queues scheduling

Operating Systems
Lecture No. 17

Reading Material

- Chapter 6 of the textbook
- Lecture 16 on Virtual TV

Summary

- Scheduling algorithms
- UNIX System V scheduling algorithm
- Optimal scheduling
- Algorithm evaluation

Multilevel Feedback Queue Scheduling

Multilevel feedback queue scheduling allows a process to move between queues. The idea is to separate processes with different CPU burst characteristics. If a process uses too much CPU time, it will be moved to a lower-priority queue. This scheme leaves I/O bound and interactive processes in the higher-priority queues. Similarly a process that waits too long in a lower-priority queue may be moved o a higher priority queue. This form of aging prevents starvation.

In general, a multi-level feedback queue scheduler is defined by the following parameters:

- Number of queues
- Scheduling algorithm for each queue
- Method used to determine when to upgrade a process to higher priority queue
- Method used to determine when to demote a process
- Method used to determine which queue a process enters when it needs service

Figure 17.1 shows an example multilevel feedback queue scheduling system with the ready queue partitioned into three queues. In this system, processes with next CPU bursts less than or equal to 8 time units are processed with the shortest possible wait times, followed by processes with CPU bursts greater than 8 but no greater than 16 time units. Processes with CPU greater than 16 time units wait for the longest time.

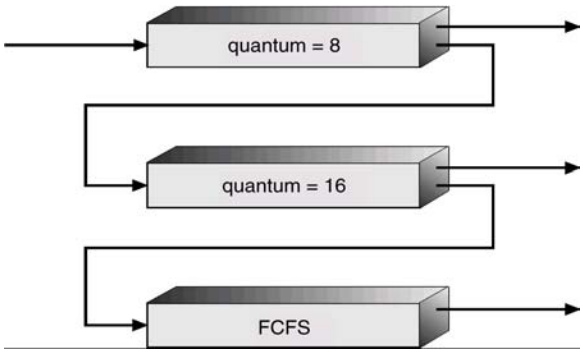


Figure 17.1 Multilevel Feedback Queues Scheduling

UNIX System V scheduling algorithm

UNIX System V scheduling algorithm is essentially a multilevel feedback priority queues algorithm with round robin within each queue, the quantum being equal to1 second. The priorities are divided into two groups/bands:

- Kernel Group
- User Group

Priorities in the Kernel Group are assigned in a manner to minimize bottlenecks, i.e, processes waiting in a lower-level routine get higher priorities than those waiting at relatively higher-level routines. We discuss this issue in detail in the lecture with an example. In decreasing order of priority, the kernel bands are:

- Swapper
- Block I/O device control processes
- File manipulation
- Character I/O device control processes
- User processes

The priorities of processes in the Kernel Group remain fixed whereas the priorities of processes in the User Group are recalculated every second. Inside the User Group, the CPU-bound processes are penalized at the expense of I/O-bound processes. Figure 17.2 shows the priority bands for the various kernel and user processes.

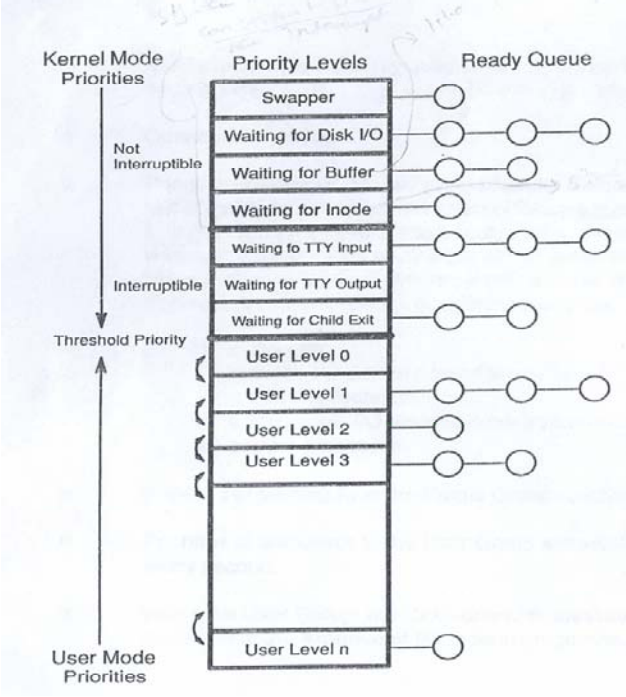


Figure 17.2. UNIX System V Scheduling Algorithm

Every second, the priority number of all those processes that are in the main memory and ready to run is updated by using the following formula:

Priority # = (Recent CPU Usage)/2 + Threshold Priority + nice

The ‘threshold priority’ and ‘nice’ values are always positive to prevent a user from migrating out of its assigned group and into a kernel group. You can change the nice value of your process with the `nice` command.

In Figure 17.3, we illustrate the working of the algorithm with an example. Note that recent CPU usage of the current process is updated every clock tick; we assume that clock interrupt occurs every sixtieth of a second. The priority number of every process in the ready queue is updated every second and the decay function is applied before recalculating the priority numbers of processes.

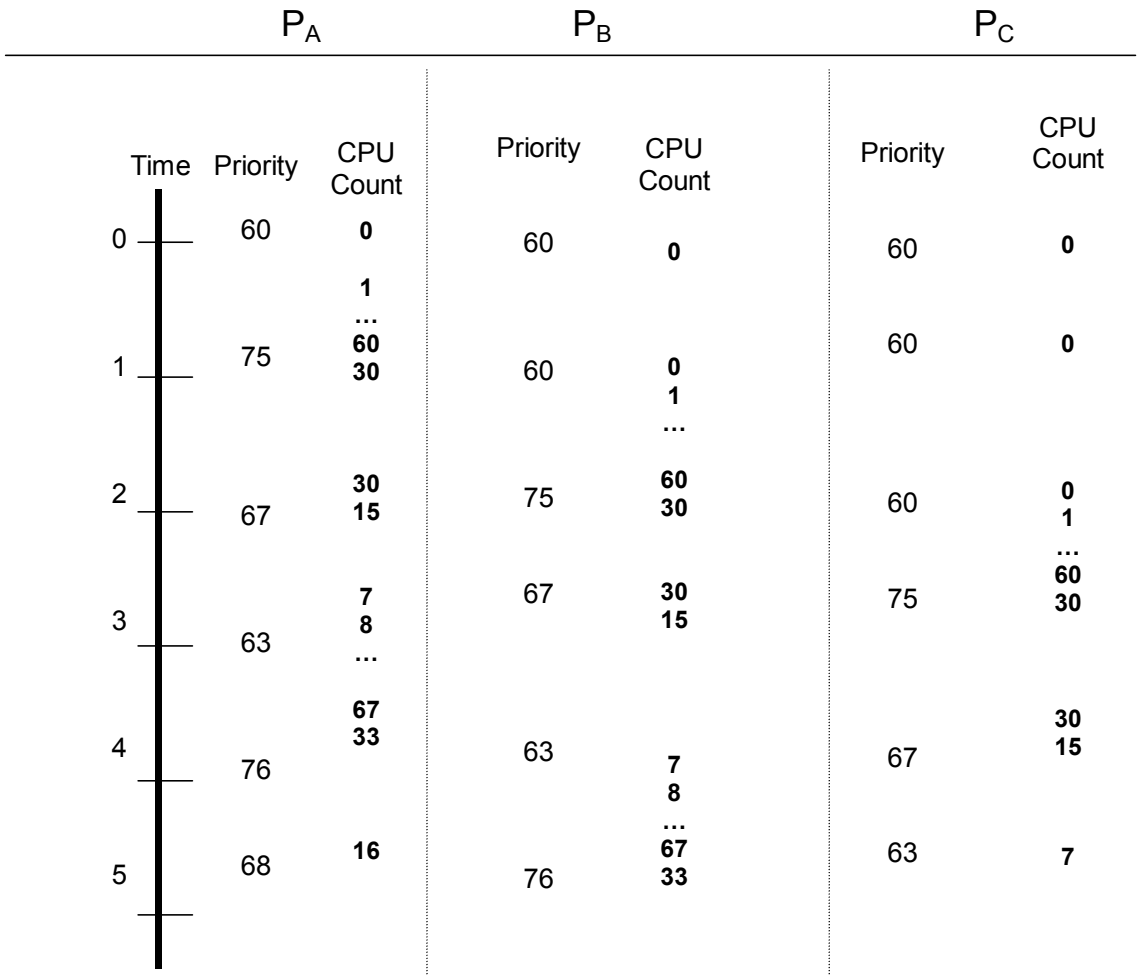


Figure 17.3 Illustration of the UNIX System V Scheduling Algorithm

Figure 17.4 shows that in case of a tie, processes are scheduled on First-Come-First-Serve basis.

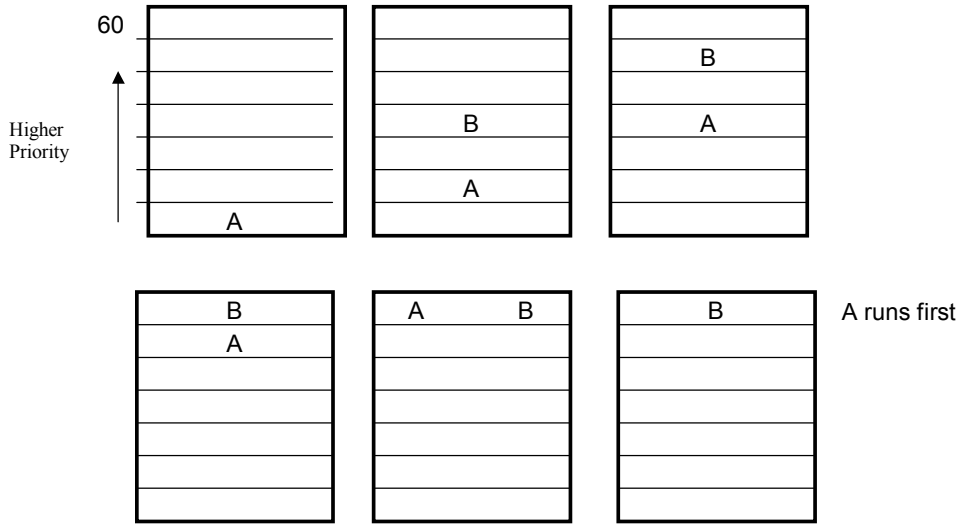


Figure 17.4 FCFS Algorithm is Used in Case of a Tie

Algorithm Evaluation

To select an algorithm, we must take into account certain factors, defining their relative importance, such as:

- Maximum CPU utilization under the constraint that maximum response time is 1 second.
- Maximize throughput such that turnaround time is (on average) linearly proportional to total execution time.

Scheduling algorithms can be evaluated by using the following techniques:

Analytic Evaluation

A scheduling algorithm and some system workload are used to produce a formula or number, which gives the performance of the algorithm for that workload. Analytic evaluation falls under two categories:

Deterministic modeling

Deterministic modeling is a type of analytic evaluation. This method takes a particular predetermined workload and defines the performance of each algorithm for workload in terms of numbers for parameters such as average wait time, average turnaround time, and average response time. Gantt charts are used to show executions of processes. We have been using this technique to explain the working of an algorithm as well as to evaluate the performance of an algorithm with a given workload.

Deterministic modeling is simple and fast. It gives exact numbers, allowing the algorithms to be compared. However it requires exact numbers for input and its answers apply to only those cases.

Queuing Models

The computer system can be defined as a network of servers. Each server has a queue of waiting processes. The CPU is a server with its ready queue, as are I/O systems with their device queues. Knowing the arrival and service rates of processes for various servers, we can compute utilization, average queue length, average wait time, and so on. This kind of study is called **queuing-network analysis**. If n is the average queue length, W is the

average waiting time in the queue, and let λ is the average arrival rate for new processes in the queue, then

$$n = \lambda * W$$

This formula is called the **Little's formula**, which is the basis of **queuing theory**, a branch of mathematics used to analyze systems involving queues and servers.

At the moment, the classes of algorithms and distributions that can be handled by queuing analysis are fairly limited. The mathematics involved is complicated and distributions can be difficult to work with. It is also generally necessary to make a number of independent assumptions that may not be accurate. Thus so that they will be able to compute an answer, queuing models are often an approximation of real systems. As a result, the accuracy of the computed results may be questionable.

The table in Figure 17.5 shows the average waiting times and average queue lengths for the various scheduling algorithms for a pre-determined system workload, computed by using Little's formula. The average job arrival rate is 0.5 jobs per unit time.

Algorithm	Average Wait Time $W = t_w$	Average Queue Length (n)
FCFS	4.6	2.3
SJF	3.6	1.8
SRTF	3.2	1.6
RR (q=1	7.0	3.5
RR (q=4)	6.0	3.0

Figure 17.5 Average Wait Time and Average Queue Length Computed With Little's Equation

Simulations

Simulations involve programming a model of the computer system, in order to get a more accurate evaluation of the scheduling algorithms. Software date structures represent the major components of the system. The simulator has a variable representing a clock; as this variable's value is increased, the simulator modifies the system state to reflect the activities of the devices, the processes and the scheduler. As the simulation executes, statistics that indicate algorithm performance are gathered and printed. Figure 17.6 shows the schematic for a simulation system used to evaluate the various scheduling algorithms.

Some of the major issues with simulation are:

- Expensive: hours of programming and execution time are required
- Simulations may be erroneous because of the assumptions about distributions used for arrival and service rates may not reflect a real environment

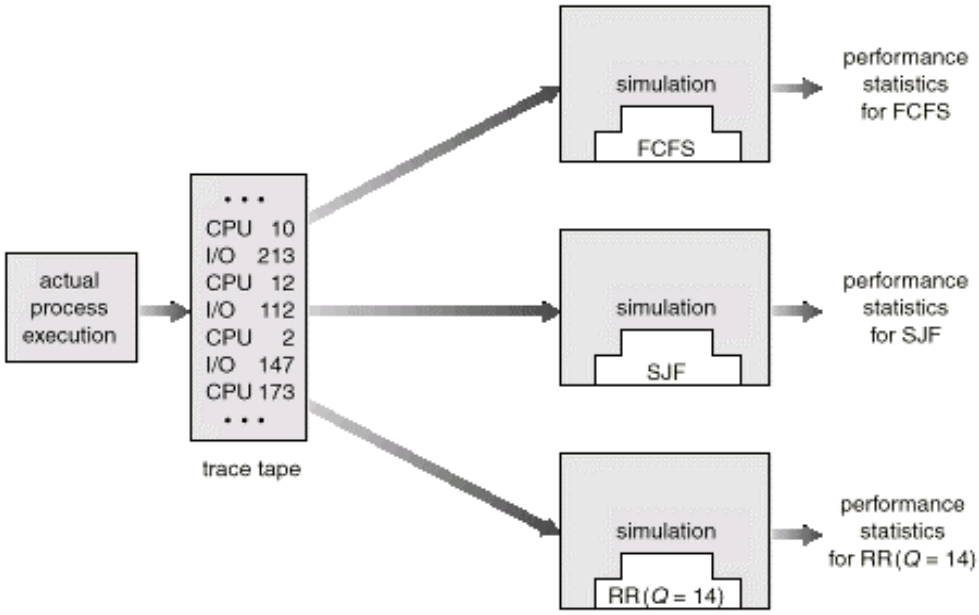


Figure 17.6 Simulation of Scheduling Algorithms

Implementation

Even a simulation is of limited accuracy. The only completely accurate way to evaluate a scheduling algorithm is to code it, put it in the operating system and see how it works. This approach puts the actual algorithm in the real system for evaluation under real operating conditions. The Open Source software licensing has made it possible for us to test various algorithms by implementing them in the Linux kernel and measuring their true performance.

The major difficulty is the cost of this approach. The expense is incurred in coding the algorithm and modifying the operating system to support it, as well as its required data structures. The other difficulty with any algorithm evaluation is that the environment in which the algorithm works will change.

Operating Systems
Lecture No. 18 and 19

Reading Material

- Chapter 7 of the textbook
- Lectures 18 and 19 on Virtual TV

Summary

- Process Synchronization: the basic concept
- The Critical Section Problem
- Solutions for the Critical Section Problem
- 2-Process Critical Section Problem solutions

Process Synchronization

Concurrent processes or threads often need access to shared data and shared resources. If there is no controlled access to shared data, it is often possible to obtain an inconsistent state of this data. Maintaining data consistency requires mechanisms to ensure the orderly execution of cooperating processes, and hence various process synchronization methods are used. In the producer-consumer problem that was discussed earlier, the version only allows one item less than the buffer size to be stored, to provide a solution for the buffer to use its entire capacity of N items is not simple. The producer and consumer share data structure 'buffer' and use other variables shown below:

```
#define BUFFER_SIZE 10
typedef struct
{
  ...
} item;
item buffer[BUFFER_SIZE];
int in=0;
int out=0;
```

The code for the producer process is:

```
while(1)
{
   /*Produce an item in nextProduced*/
   while(counter == BUFFER_SIZE); /*do nothing*/
   buffer[in]=nextProduced;
   in=(in+1)%BUFFER_SIZE;
   counter++;
}
```

The code for the consumer process is:

```
while(1)
{
   while(counter==0); //do nothing
   nextConsumed=buffer[out];
   out=(out+1)%BUFFER_SIZE;
   counter--;
   /*Consume the item in nextConsumed*/
}
```

Both producer and consumer routines may not execute properly if executed concurrently. Suppose that the value of the counter is 5, and that both the producer and the consumer execute the statement counter++ and counter- - concurrently. Following the execution of these statements the value of the counter may be 4,5,or 6! The only correct result of these statements should be counter= =5, which is generated if the consumer and the producer execute separately. Suppose counter++ is implemented in machine code as the following instructions:

```
MOV  R1, counter
INC  R1
MOV  counter, R1
```

whereas counter- - maybe implemented as:

```
MOV  R2, counter
DEC  R2
MOV  counter, R2
```

If both the producer and consumer attempt to update the buffer concurrently, the machine language statements may get interleaved. Interleaving depends upon how the producer and consumer processes are scheduled. Assume counter is initially 5. One interleaving of statements is:

```
producer: MOV  R1, counter   (R1 = 5)
          INC  R1            (R1 = 6)
consumer: MOV  R2, counter   (R2 = 5)
          DEC  R2            (R2 = 4)
producer: MOV  counter, R1   (counter = 6)
consumer: MOV  counter, R2   (counter = 4)
```

The value of `count` will be 4, where the correct result should be 5. The value of `count` could also be 6 if producer executes `MOV counter, R1` at the end. The reason for this state is that we allowed both processes to manipulate the variable counter concurrently. A situation like this, where several processes access and manipulate the same data concurrently and the outcome of the manipulation depends on the particular order in which the access takes place, is called a **race condition**. To guard against such race conditions, we require synchronization of processes.

Concurrent transactions in a bank or in an airline reservation (or travel agent) office are a couple of other examples that illustrates the critical section problem. We show

interleaving of two bank transactions, a deposit and a withdrawal. Here are the details of the transactions:

- Current balance = Rs. 50,000
- Check deposited = Rs. 10,000
- ATM withdrawn = Rs. 5,000

The codes for deposit and withdrawal are shown in Figure 18.1.

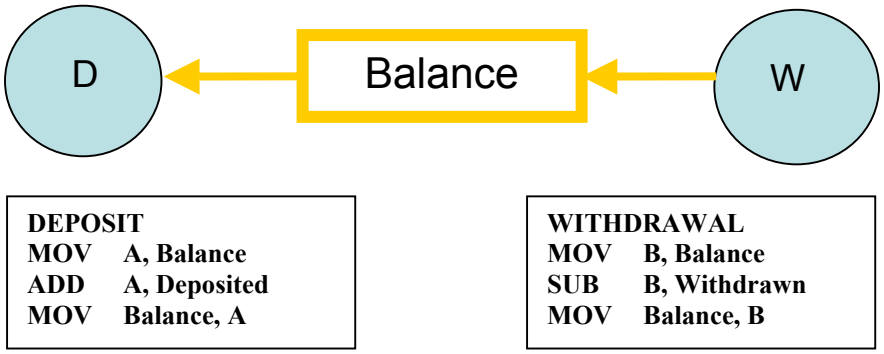


Figure 18.1 Bank transactions—deposit and withdrawal

Here is what may happen if the two transactions are allowed to execute concurrently, i.e., the transactions are allowed to interleave. Note that in this case the final balance will be Rs. 45,000, i.e., a loss of Rs. 5,000. If `MOV Balance,A` executes at the end, the result will be a gain of Rs. 5,000. In both cases, the final result is wrong.

```
Check Deposit:
        MOV   A, Balance        // A = 50,000
        ADD   A, Deposited      // A = 60,000
ATM Withdrawal:
        MOV   B, Balance        // B = 50,000
        SUB   B, Withdrawn      // B = 45,000
Check Deposit:
        MOV   Balance, A        // Balance = 60,000
ATM Withdrawal:
        MOV   Balance, B        // Balance = 45,000
```

The Critical Section Problem

Critical Section: A piece of code in a cooperating process in which the process may updates shared data (variable, file, database, etc.).

Critical Section Problem: Serialize executions of critical sections in cooperating processes.

When a process executes code that manipulates shared data (or resource), we say that the process is in its critical section (for that shared data). The execution of critical sections must be mutually exclusive: at any time, only one process is allowed to execute in its critical section (even with multiple processors). So each process must first request permission to enter its critical section. The section of code implementing this request is

called the **entry section**. The remaining code is the **remainder section**. The critical section problem is to design a protocol that the processes can use so that their action will not depend on the order in which their execution is interleaved (possibly on many processors).

There can be three kinds of solution to the critical section problem:

- Software based solutions
- Hardware based solutions
- Operating system based solution

We discuss the software solutions first. Regardless of the type of solution, the structure of the solution should be as follows. The Entry and Exist sections comprise solution for the problem.

```
do
{
        Entry section

          critical section

        Exit section

          remainder section

} while(1)
```

Solution to the Critical Section Problem

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

1. **Mutual Exclusion**
   If process P_i is executing in its critical section, then no other process can be executing in their critical section.
2. **Progress**
   If no process is executing in its critical section and some processes wish to enter their critical sections, then only those processes that are not executing in their remainder section can participate in the decision on which will enter its critical section next, and this selection cannot be postponed indefinitely.
3. **Bounded Waiting**
   There exists a bound on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

Assumptions

While formulating a solution, we must keep the following assumptions in mind:

- Assume that each process executes at a nonzero speed
- No assumption can be made regarding the relative speeds of the N processes.

2-Process Solutions to the Critical Section Problem

In this section algorithms that are applicable to two processes will be discussed. The processes are P_0 and P_1. When presenting P_i, we use P_j to denote the other process. An assumption is that the basic machine language instructions such as load and store are executed atomically, that is an operation that completes in its entirety without interruption.

Algorithm 1

The first approach is to let the processes share a common integer variable **turn** initialized to 0 or 1. If turn = = i, then process P_i is allowed to execute in its critical section. The structure of the process P_i is as follows:

```
do
{
    while(turn!=j);

       critical section

     turn=j;

       remainder section
} while(1)
```

This solution ensures mutual exclusion, that is only one process at a time can be in its critical section. However it does not satisfy the progress requirement, since it requires strict alternation of processes in the execution of the critical section. For example, if turn= =0 and P_1 is ready to enter its critical section, P_1 cannot do so even though P_0 may be in its remainder section. The bounded wait condition is satisfied though, because there is an alternation between the turns of the two processes.

Algorithm 2

In algorithm two, the variable turn is replaced with an array `boolean flag[2]` whose elements are initialized to false. If flag is true for a process that indicates that the process is ready to enter its critical section. The structure of process P_i is shown:

```
do
{
        flag[i]=true;
        while(flag[j]);

              critical section

        flag[i]=false;

              remainder section
} while(1)
```

In this algorithm P_i sets `flag[i]= true` signaling that it is ready to enter its critical section. Then P_i checks to verify that process P_j is not also ready to enter its critical section. If P_j were ready, then P_i would wait until P_j had indicated that it no longer needed to be in the critical section (that is until `flag[j]=false`). At this point P_i would enter the critical section. On exiting the critical section, P_i would set `flag[i]=false` allowing the other process to enter its critical section. In this solution, the mutual exclusion requirement is satisfied. Unfortunately the progress condition is not met; consider the following execution sequence:

T_0: P_0 sets `flag[0]= true`
T_1: P_1 sets `flag[1]= true`

Now both the processes are looping forever in their respective while statements.

Operating Systems
Lecture No. 20

Reading Material
- Chapter 7 of the textbook
- Lecture 20 on Virtual TV

Summary
- 2-Process Critical Section Problem (continued)
- n-Process Critical Section Problem
- The Bakery Algorithm

2-Process Critical Section Problem (continued)
We discussed two solutions for the 2-process critical section problem in lecture 19 but both were not acceptable because they did not satisfy the progress condition. Here is a good solution for the critical section problem that satisfies all three requirements of a good solution.

Algorithm 3
The processes share two variables:

```
boolean flag[2];
int turn;
```

The boolean array of 'flag' is initialized to false, whereas 'turn' maybe 0 or 1. The structure of the process is as follows:

```
do
{
        flag[i]=true;
        turn=j;
        while(flag[j] && turn==j);

            critical section

        flag[i]=false;

            remainder section

} while(1)
```

To enter its critical section, P_i sets flag[i] to true, and sets 'turn' to j, asserting that if the other process wishes to enter its critical section, it may do so. If both try to enter at the

same time, they will attempt to set 'turn' to i and j. However, only one of these assignments will last, the other will occur but be overwritten instantly. Hence, the eventual value of 'turn' will decide which process gets to enter its critical section.

To prove mutual exclusion, note that P_i enters its critical section only if either flag[j]=false or turn=i. Also, if both processes were executing in their critical sections at the same time, then flag[0]= = flag[1]= = true. These two observations suggest that P_0 and P_1 could not have found both conditions in the while statement true at the same time, since the value of 'turn' can either be 0 or 1. Hence only one process say P_0 must have successfully exited the while statement. Hence mutual exclusion is preserved.

To prove bounded wait and progress requirements, we note that a process P_i can be prevented the critical section only if it is stuck in the while loop with the condition flag[j]= =true and turn=j. If P_j is not ready to enter the critical section, then flag[j]=flase and P_i can enter its critical section. If P_j has set flag[j]=true and is also executing its while statement then either turn=i or turn=j. If turn=i then P_i enters its critical section, otherwise P_j. However, whenever a process finishes executing in its critical section, lets assume P_j, it resets flag[j] to false allowing P_i to enter its critical section. If P_j resets flag[j]=true, then it must also set 'turn' to i, and since P_i does not change the value of 'turn' while executing in its while statement, P_i will enter its critical section (progress) after at most one entry by P_j (bounded waiting).

N-Process Critical Section Problem

In this section we extend the critical section problem of two processes to include n processes. Consider a system of n processes (P_o, P_1 …… P_{n-1}). Each process has a segment of code called a critical section in which the process may be changing common variables, updating a table, writing a file and so on. The important feature of the system in that, when one process enters its critical section, no other process is allowed to execute in its critical section. Thus the execution of critical sections by the processes is mutually exclusive in time. The critical section problem is to design a protocol to serialize executions of critical sections. Each process must request permission to enter its critical section. Many solutions are available in the literature to solve the N-process critical section problem. We will discuss a simple and elegant solution, known as the Bakery algorithm.

The Bakery Algorithm

The bakery algorithm is due to Leslie Lamport and is based on a scheduling algorithm commonly used in bakeries, ice-cream stores, and other locations where order must be made out of chaos. On entering the store, each customer receives a number. The customer with the lowest number is served next. Before entering its critical section, process receives a ticket number. Holder of the smallest ticket number enters its critical section. Unfortunately, the bakery algorithm cannot guarantee that two processes (customers) will not receive the same number. In the case of a tie, the process with the lowest ID is served first. If processes Pi and Pj receive the same number, if i < j, then Pi is served first; else Pj is served first. The ticket numbering scheme always generates numbers in the increasing order of enumeration; i.e., 1, 2, 3, 4, 5 ...

Since process names are unique and totally ordered, our algorithm is completely deterministic. The common data structures are:

```
boolean choosing [n];
int number[n];
```

Initially these data structures are initialized to false and 0, respectively. The following notation is defined for convenience:

- (ticket #, process id #)
- (a,b) < (c,d) if a<c or if a= =c and b<d.
- max($a_{0, \ldots} a_{n-1}$) is a number, k, such that k>= a_i for i=0,…n-1

The structure of process P_i used in the bakery algorithm is as follows:

```
do
{
   choosing[i] = true;
   number[i] = max(number[0],number[1],..number[n-1])+1;
   choosing[i] = false;

   for(j=0; j<n; j++) {
     while(choosing[j]);
     while((number[j]!=0) && ((number[j],j) < (number[i],i)));
   }

              Critical section

   number[i]=0;

              Remainder section

} while(1);
```

To prove that the bakery algorithm is correct, we need to first show that if P_i is in its critical section and P_k has already chosen its number k!=0, then ((number [i],i) < (number[k],k)). Consider P_i in its critical section and P_k trying to enter its critical section. When process P_k executes the second while statement for j= = i it finds that,

- number[i] != 0
- (number[i],i) < (number[k],k)

Thus it keeps looping in the while statement until P_i leaves the P_i critical section. Hence mutual exclusion is preserved. For progress and bounded wait we observe that the processes enter their critical section on a first come first serve basis.

Following is an example of how the Bakery algorithm works. In the first table, we show that there are five processes, P0 through P4. P1's number is 0 because it is not interested in getting into its critical section at this time. All other processes are interested in entering their critical sections and have chosen non-zero numbers by using the `max()` function in their entry sections.

Process	Number
P0	3
P1	0
P2	7
P3	4
P4	8

The following table shows the status of all the processes as they execute the 'for' loops in their entry sections. The gray cells show processes waiting in the second while loops in their entry sections. The table shows that P0 never waits for any process and is, therefore, the first process to enter its critical section, while all other processes wait in their second while loops for j = = 0, indicating that they are waiting for P0 to get out of its critical section and then they would make progress (i.e., they will get out the while loop, increment j by one, and continue their execution).

You can make the following observations by following the Bakery algorithm closely with the help of this table:

- P1 not interested to get into its critical section ⇒ number[1] is 0
- P2, P3, and P4 wait for P0
- P0 gets into its CS, get out, and sets its number to 0
- P3 get into its CS and P2 and P4 wait for it to get out of its CS
- P2 gets into its CS and P4 waits for it to get out
- P4 gets into its CS
- Sequence of execution of processes: <P0, P3, P2, P4>

j	**P0**	**P2**	**P3**	**P4**
0	(3,0) < (3,0)	(3,0) < (7,2)	(3,0) < (4,3)	(3,0) < (8,4)
1	Number[1] = 0	Number[1] = 0	Number[1] = 0	Number[1] = 0
2	(7,2) < (3,0)	(7,2) < (7,2)	(7,2) < (4,3)	(7,2) < (8,4)
3	(4,3) < (3,0)	(4,3) < (7,2)	(4,3) < (4,3)	(4,3) < (8,4)
4	(8,4) < (3,0)	(8,4) < (7,2)	(8,4) < (4,3)	(8,4) < (8,4)

Operating Systems
Lecture No. 21

Reading Material

- Chapter 7 of the textbook
- Lecture 21 on Virtual TV

Summary

- Hardware solutions

Hardware Solutions for the Critical Section Problem

In this section, we discuss some simple hardware (CPU) instructions that can be used to provide synchronization between processes and are available on many systems.

The critical section problem can be solved simply in a uniprocessor environment if we could forbid interrupts to occur while a shared variable is being modified. In this manner, we could be sure that the current sequence of instructions would be run, so no unexpected modifications could be made to the shared variable.

Unfortunately this solution is not feasible in a multiprocessing environment, as disabling interrupts can be time consuming as the message is passed to all processors. This message passing delays entry into each critical section, and system efficiency decreases.

Normally, access to a memory location excludes other accesses to that same location. Designers have proposed machine instructions that perform two operations atomically (indivisibly) on the same memory location (e.g., reading and writing). The execution of such an instruction is also mutually exclusive (even on Multiprocessors). They can be used to provide mutual exclusion but other mechanisms are needed to satisfy the other two requirements of a good solution to the critical section problem.

We can use these special instructions to solve the critical section problem. These instructions are TestAndSet (also known as TestAndSetLock; TSL) and Swap. The semantics of the TestAndSet instruction are as follows:

```
boolean TestAndSet(Boolean &target)
{
  boolean rv=target;
  target=true;
  return rv;
}
```

The semantics simply say that the instruction saves the current value of ‘target’, set it to true, and returns the saved value.

The important characteristic is that this instruction is executed atomically. Thus if two TestAndSet instructions are executed simultaneously, they will be executed sequentially in some arbitrary order.

If the machine supports TestAndSet instruction, then we can implement mutual exclusion by declaring a Boolean variable lock, initialized to false. The structure of process P_i becomes:

```
do
{
      while (TestAndSet(lock)) ;

             Critical section

      lock=false;

             Remainder section

} while(1);
```

The above TSL-based solution is no good because even though mutual exclusion and progress are satisfied, bounded waiting is not.

The semantics of the Swap instruction, another atomic instruction, are, as expected, as follows:

```
boolean Swap(boolean &a, boolean &b)
{
  boolean temp=a;
  a=b;
  b=temp;
}
```

If the machine supports the Swap instruction, mutual exclusion can be implemented as follows. A global Boolean variable lock is declared and is initialized to false. In addition each process also has a local Boolean variable key. The structure of process P_i is:

```
do
{
       key=true;
       while(key == true)
           Swap(lock,key);

             Critical section

       lock=false;

             Remainder section

} while(1);
```

Just like the TSL-based solution shown in this section, the above Swap-based solution is not good because even though mutual exclusion and progress are satisfied, bounded waiting is not. In the next lecture, we will discuss a good solution for the critical section problem by using the hardware instructions.

Operating Systems
Lecture No. 22

Reading Material

- Chapter 7 of the textbook
- Lecture 22 on Virtual TV

Summary

- Hardware based solutions
- Semaphores
- Semaphore based solutions for the critical section problem

Hardware Solutions

In lecture 21 we started discussing the hardware solutions for the critical section problem. We discussed two possible solutions but realized that whereas both solutions satisfied the mutual exclusion and bounded waiting conditions, neither satisfied the progress condition. We now describe a solution that satisfies all three requirements of a solution to the critical section problem.

Algorithm 3

In this algorithm, we combine the ideas of the first two algorithms. The common data structures used by a cooperating process are:

```
boolean waiting[n];
boolean lock;
```

The structure of process P_i is:

```
do
{
        waiting[i] = true;
        key = true;
        while (waiting[i] && key)
            key = TestAndSet(lock);
        waiting[i] = false;

             Critical section

        j = (i+1) % n;
        while ((j!=i) && !waiting[j])
            j = (j+1)% n;
        if (j == i)
            lock = false;
        else
            waiting[j] = false;

             Remainder section

} while(1);
```

These data structures are initialized to false. To prove that the mutual exclusion requirement is met, we note that process P_i can enter its critical section only if either waiting[i]= = false or key = = false. The value of key can become false only if TestAndSet is executed. The first process to execute the TestAndSet instruction will find key= =false; all others must wait. The variable waiting[i] can only become false if another process leaves its critical section; only one waiting[i] is set to false, maintaining the mutual exclusion requirement.

To prove the progress requirement is met, we note that the arguments presented for mutual exclusion also apply here, since a process exiting the critical section either sets lock to false or sets waiting[j] to false. Both allow a process that is waiting to enter its critical section to proceed.

To prove that the bounded waiting requirement is met, we note that, when a process leaves its critical section, it scans the array waiting in the cyclic ordering (i+1, i+2, …, n-1, 0, 1, …, i-1). It designates the first process it sees that is in its entry section with waiting[j]=true as the next one to enter its critical section. Any process waiting to do so will enter its critical section within n-1 turns.

Semaphores

Hardware solutions to synchronization problems are not easy to generalize to more complex problems. To overcome this difficulty we can use a synchronization tool called a semaphore. A **semaphore** S is an integer variable that, apart from initialization is accessible only through two standard atomic operations: wait and signal. These operations were originally termed P (for wait) and V (for signal). The classical definitions of wait and signal are:

```
wait(S) {
  while(S<=0)
  ;// no op
  S--;
}
```

```
signal(S) {
  S++;
}
```

Modifications to the integer value of the semaphore in the wait and signal operations must be executed indivisibly. That is, when one process is updating the value of a semaphore, other processes cannot simultaneously modify that same semaphore value. In addition, in the case of the wait(S), the testing of the integer value of S (S<=0) and its possible modification (S--) must also be executed without interruption.

We can use semaphores to deal with the n-process critical section problem. The n processes share a semaphore, **mutex** (standing for mutual exclusion) initialized to 1. Each process P_i is organized as follows:

```
do
{
     wait(mutex);
          Critical section
     signal(mutex);
          Remainder section
} while(1);
```

As was the case with the hardware-based solutions, this is not a good solution because even though it satisfies mutual exclusion and progress, it does not satisfy bounded wait.

In a uni-processor environment, to ensure atomic execution, while executing wait and signal, interrupts can be disabled. In case of a multi-processor environment, to ensure atomic execution is one can lock the data bus, or use a soft solution such as the Bakery algorithm.

The main disadvantage of the semaphore discussed in the previous section is that it requires **busy waiting**. While a process is in its critical section, any other process that tries to enter its critical section must loop continuously in the entry code. This continual looping is clearly a problem in a real multiprogramming system, where a single CPU is shared among many processes. Busy waiting wastes CPU cycles that some other process may be able to use productively. This type of semaphore is also called a **spinlock** (because the process spins while waiting for the lock). Spinlocks are useful in multiprocessor systems. The advantage of a spinlock is that no context switch is required when a process must wait on a lock, and a context switch may take considerable time. This is, spinlocks are useful when they are expected to be held for short times. The definition of semaphore should be modified to eliminate busy waiting. We will discuss the modified definition of semaphore in the next lecture.

MIDTERM EXAMINATION
Spring 2012
CS604 - Operating System

Question No: 1 (Marks: 1) - Please choose one
_____ command to resume the execution of a suspended job in the foreground

► **fg (Page 68)**
► bg
► jobs
► kill

Question No: 2 (Marks: 1) - Please choose one
_____ commands in Linux is used to copy file
►is
►**cp (Page 30)**
►mv
►mkdir

Question No: 3 (Marks: 1) - Please choose one
The process id returned to the child process after successful fork system call execution is _____.
►**0 (Page 40)**
►1
►2
►3

Question No: 4 (Marks: 1) - Please choose one
In _____addressing, the recipient is not required to name the sender.

► Symmetric
► **Asymmetric (Page 47)**
► Both symmetric and asymmetric
► None of the given options

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

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

Question No: 5 (Marks: 1) - Please choose one
A solution to the critical section problem must satisfy the following requirements
►Progress
►Mutual exclusion
►Bounded Waiting
►All of these (Page 101)

Question No: 6 (Marks: 1) - Please choose one
Typically the execlp system call is used after a fork system call.
►True (Page 39)
►False

Question No: 7 (Marks: 1) - Please choose one
You can create a threads by using the pthread_create() call.
►True (Page 76)
►False

Question No: 8 (Marks: 1) - Please choose one
The interval from the time of submission to the time of completion is the _____

►Turnaround time (Page 83)
►Waiting time
►Response time
►None of all these

Question No: 9 (Marks: 1) - Please choose one
Each process must first request permission to enter its critical section. The section of code implementing this request is called the _____

►entry section (Page 100)
►Critical Section
►remainder section
►None of all these

Question No: 10 (Marks: 1) - Please choose one
IPC provides a mechanism to allow processes to communicate and to synchronize their actions without sharing the same_____

►Address space (Page 46)
►Address Name
►Address ID
►None of all these

خدا کے سوا کسی سے امید مت رکھو

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

MIDTERM EXAMINATION
Spring 2011
CS604 - Operating System

Question No: 1 (Marks: 1) - Please choose one
Linux is a version of _____ operating system.

► OS/2
► Windows
► **Unix** click here for detail
► None of the above

Question No: 2 (Marks: 1) - Please choose one
Current working directory can be accessed using --------- Command.

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

Question No: 3 (Marks: 1) - Please choose one
Mkfifo() is a _____.

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

Question No: 4 (Marks: 1) - Please choose one
____ command gives a snapshot of the current processes.
► **ps (Page 66)**
► top
► who
► ls

Question No: 5 (Marks: 1) - Please choose one
Time interval when the I/O Devices are accessed is known as --------.
► CPU Burst
► **IO Burst** Click here for detail
► Time Slice
► None of Above

بری صحبت سے تنہائی بہتر ہے اور تنہائی سے نیک صحبت بہتر ہے

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

Question No: 6 (Marks: 1) - Please choose one
The process of switching from one process to another is called ----------.

► **Context switching (Page 34)**
► scheduling
► quantum period
► latency

Question No: 7 (Marks: 1) - Please choose one
_____ directory includes essential system boot files including the kernel image.
► /bin
► **/boot (Page 26)**
► /dev
► /etc

Question No: 8 (Marks: 1) - Please choose one
_____ scheduling algorithm is sometimes called shortest remaining time first scheduling algorithm.

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

Question No: 9 (Marks: 1) - Please choose one
A semaphore that cause Busy-Waiting is termed as _____.
► **Spinlock (Page 113)**
► Monitor
► Critical region
► Critical section

Question No: 10 (Marks: 1) - Please choose one
Progress and Bounded Waiting are some of the characteristics to solve the critical section problems.

► **True (Page 101)**
► False

Question No: 11 (Marks: 1) - Please choose one
In ---------addressing; the recipient is not required to name the sender.

► Symmetric
► **Asymmetric (Page 47) rep**
► Both symmetric and asymmetric
► None of the given options

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

Question No: 12 (Marks: 1) - Please choose one
The execution of critical sections must NOT be mutually exclusive

► True
► **False (Page 100)**

Question No: 13 (Marks: 1) - Please choose one
A program in execution is called a _____.

► Command
► **Process (Page 31)**
► Software
► Compiler

Question No: 14 (Marks: 1) - Please choose one
The critical section problem can be solved by the following except

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

Question No: 15 (Marks: 1) - Please choose one
The bottom layer in the layered approach of Operating System is----------

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

Question No: 16 (Marks: 1) - Please choose one
The manual pages can be read in Linux using ____ command.

► **man (Page 27)**
► wan
► desc
► help

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

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

MIDTERM EXAMINATION
Spring 2010
CS604 - Operating System

Question No: 1 (Marks: 1) - Please choose one
The hardware mechanism that enables a device to notify CPU is called an ----------
► **Interrupt** click here for detail
► Signal
► Trap
► Process

Question No: 2 (Marks: 1) - Please choose one
You can display the contents (names of files and directories) of a directory in UNIX/Linux directory structure with the ---------- command.

►ll
► s
► **ls (Page 28)**
► none of the given options

Question No: 3 (Marks: 1) - Please choose one
The ---------- system call suspends the calling process.
► fork
► **wait (Page 42)**
► exec
► exit

Question No: 4 (Marks: 1) - Please choose one
In ---------addressing, the recipient is not required to name the sender.
► Symmetric
► **Asymmetric (Page 47) rep**
► Both symmetric and asymmetric
► None of the given options

Question No: 5 (Marks: 1) - Please choose one
-------- command gives a snapshot of the current processes.
► **ps (Page 66) rep**
► top
► who
► ls

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

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

Question No: 6 (Marks: 1) - Please choose one
----------command to resume the execution of a suspended job in the foreground
► **fg (Page 68) rep**
► bg
► jobs
► kill

Question No: 7 (Marks: 1) - Please choose one
You can use the ---------- command to display the status of suspended and background processes

► fg
► bg
► **jobs (Page 68)**
► kill

Question No: 8 (Marks: 1) - Please choose one
You can terminate a foreground process by pressing ----------

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

Question No: 9 (Marks: 1) - Please choose one
A time sharing system is
► Multi tasking
► Interactive
► Multi user
► **All of these (Page 9)**

Question No: 10 (Marks: 1) - Please choose one
The main characteristic of a Real time system is
► Efficiency
► Large Virtual Memory
► Large secondary storage device
► **Usability** click here for detail

Question No: 11 (Marks: 1) - Please choose one
Shared libraries and kernel modules are stored in directory
► /bin
► /dev
► /boot
► **/lib (Page 26)**

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

Question No: 12 (Marks: 1) - Please choose one
_____scheduler selects the process from the job pool and put them in main memory.

► **Long term (Page 36)**
► Short term
► Medium term
► Swapper

Question No: 13 (Marks: 1) - Please choose one
In indirect inter process communication, a sender_____mention the name of the recipient.
► do
► **do not (Page 47)**

Question No: 14 (Marks: 1) - Please choose one
A_____ is an integer variable that, apart from initialization is accessible only through two standard atomic operations: wait and signal.

► **Semaphore (Page 111)**
► Monitor
► Critical region
► Critical section

Question No: 15 (Marks: 1) - Please choose one
A semaphore that cause Busy-Waiting is termed as .

► **Spinlock (Page 112)**
► Monitor
► Critical region
► Critical section

Question No: 16 (Marks: 1) - Please choose one
The execution of critical sections must NOT be mutually exclusive

► True
► **False (Page 100) rep**

Question No: 17 (Marks: 1) - Please choose one
The performance of Round Robin algorithm does NOT depends heavily on the size of the time quantum.

► **True (Page 89)**
► False

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

Question No: 18 (Marks: 1) - Please choose one
The following requirement for solving critical section problem is known as _____. "There exists a bound on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted."

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

Question No: 19 (Marks: 1) - Please choose one
The critical section problem can be solved by the following except

► Software based solution
► **Firmware based solution (Page 101) rep**
► Operating system based solution
► Hardware based solution

Question No: 20 (Marks: 1) - Please choose one
_____ is also called Swapper.

► Swap space
► **Medium term scheduler (Page 37)**
► Short term scheduler
► Long term scheduler

MIDTERM EXAMINATION
Spring 2010
CS604 - Operating System

Question No: 1 (Marks: 1) - Please choose one
Linux OS can support multiple users at a time

►**True (Page 9)**
►False

جھوٹ انسان اور ایمان دونوں کا دشمن ہے

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

Question No: 2 (Marks: 1) - Please choose one
The Operating system is a layer of software between _____ and _____.

►**hardware, software application (Page 21)**
►Kernel, hardware
►Dos, Windows
►Windows, Kernel

Question No: 3 (Marks: 1) - Please choose one
The major advantage of multi-programming system is

►More than one jobs can be processed at a given time
►**CPU utilization can be increased (Page 8)**
►Jobs can be completed quickly
►All of the options are correct

Question No: 4 (Marks: 1) - Please choose one
The main characteristic of a Real time system is

►Efficiency
►Large Virtual Memory
►Large secondary storage device
►**Usability** click here for detail rep

Question No: 5 (Marks: 1) - Please choose one
Command-line interpreter is also called _____ in some operating systems.
►Kernel
►**Shell (Page 16)**
►Signal
►API

Question No: 6 (Marks: 1) - Please choose one
I/O instructions are Privileged Instructions.
►**True (Page 12)**
►False

Question No: 7 (Marks: 1) - Please choose one
In Linux directory structure, there is _____ root directory.
►**1 (Page 26)**
►2
►3
►4

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

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

Question No: 8 (Marks: 1) - Please choose one
Utilities used for system administration (halt, ifconfig, fdisk, etc.) are stored in _____ directory.

►/dev
►/boot
►/lib
►/sbin (Page 27)

Question No: 9 (Marks: 1) - Please choose one
rm and [r]mkdir commands are used to _____ directory.

►Create
►Move
►Remove (Page 30)
►Modify

Question No: 10 (Marks: 1) - Please choose one
You can use the mv file1 file2 command to move _____

►file1 to file2. (Page 30)
►file 2 to file 1
►this command will not work for moving files
►None of the option is correct.
►Both option a and b are correct

Question No: 11 (Marks: 1) - Please choose one
Taking the CPU from one process and giving the CPU to another process is termed as

►Context Switching **click here for detail**
►Dispatching
►Swapping
►Tracking

Question No: 12 (Marks: 1) - Please choose one
A Process that has finished working, as well as its parent process has also finished its execution. In this state the process A will be called as _____ process.

►Child
►Thread
►Zombie
►Fork

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

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

Question No: 13 (Marks: 1) - Please choose one
Bounded Buffer is a buffer of _____ size
►variable
►**fixed (Page 44)**

Question No: 14 (Marks: 1) - Please choose one
In _____ communication the process which wants to communicate with the other process must explicitly name the recipient and the sender.
►**Direct (Page 46)**
►Indirect
►Automatic
►Self

Question No: 15 (Marks: 1) - Please choose one
In indirect inter process communication, a sender _____ mention the name of the recipient.
►does
►**does not (Page 47) rep**

Question No: 16 (Marks: 1) - Please choose one
The returned code to the child process after successful fork system call execution is

►1
►2
►3
►**0 (Page 40) rep**

Question No: 17 (Marks: 1) - Please choose one
If the fork system call fails, it returns

►1
►**-1 (Page 40)**
►2
►0

Question No: 18 (Marks: 1) - Please choose one
When a process opens its first file explicitly it will get descriptor number _____

►1
►2
►**3** click here for detail
►4

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

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

MIDTERM EXAMINATION
Spring 2009
CS604 - Operating System

Question No: 1 (Marks: 1) - Please choose one
I MB or 1 megabyte is equivalent to----

►1024 bytes
►**1024^2 bytes** click here for detail
►1024^3 bytes
►1000000 bytes

Question No: 2 (Marks: 1) - Please choose one
The bottom layer in the layered approach of Operating System is----------

►Hardware
►User interface
►**Hardware (Page 21) rep**
►Kernel
►None of the given options

Question No: 3 (Marks: 1) - Please choose one
----------has a hierarchical file system structure.

►DOS
►Windows
►**UNIX (Page 25)**
►None of the given options

Question No: 4 (Marks: 1) - Please choose one
You can use the ---------command in UNIX to create a directory.

►rmdir
►**mkdir (Page 29)**
►cp
►gcc

بد صورت چہرہ بد صورت دماغ سے بہتر ہے

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

Question No: 5 (Marks: 1) - Please choose one
Files that start with a -------- in UNIX/Linux directory structure are known as hidden files .

►**. (dot) (Page 28)**
►# (hash)
►/ (slash)
►~ (tilt)

Question No: 6 (Marks: 1) - Please choose one
The creating process is called a---------- process while the new processes are called the ---------- of that process

►None of the given options
►Children, parent
►**Parent, children (Page 38)**
►Zombie, single

Question No: 7 (Marks: 1) - Please choose one
_____buffer places no practical limit on the size of the buffer

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

Question No: 8 (Marks: 1) - Please choose one
The _____ are used for communication between related or unrelated processes on the same system or unrelated processes on different systems.

►Pipes
►**BSD Sockets (Page 53)**
►Named pipe (FIFO)
►None of the given options

Question No: 9 (Marks: 1) - Please choose one
A_____ is an abstract key for accessing a file.

►**File descriptor** **click here for detail**
►Input Redirection
►Output Redirection
►FIFO

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

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

Question No: 10 (Marks: 1) - Please choose one
-------- command gives a snapshot of the current processes.

►**ps (Page 66) rep**
►top
►who
►ls

Question No: 11 (Marks: 1) - Please choose one
You can display all of the signals supported by your system, along with their numbers, by using the ---------- command

►<Ctrl-A>
►fg
►jobs
►**kill -l (Page 69)**

Question No: 12 (Marks: 1) - Please choose one
The time it takes for the dispatcher to stop one process and start another running is known as the----------.

►**Dispatch latency (Page 82)**
►Scheduling
►Context switching
►None of the given options

Question No: 13 (Marks: 1) - Please choose one
First-Come, First-Served (FCFS) is a ----------scheduling algorithm.

►preemptive
►**non-preemptive (Page 83)**
►both preemptive and non- preemptive
►none of the given options

Question No: 14 (Marks: 1) - Please choose one
The Shortest-Job-First Scheduling algorithm can be

►Preemptive only
►non-preemptive only
►**preemptive or non-preemptive. (Page 85)**
►None of the given options

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

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

Question No: 15 (Marks: 1) - Please choose one
Preemptive ----------scheduling is sometimes called shortestremaining-time-first scheduling.

►First-Come-First-Served (FCFS)
►Round-Robin
►Sorted Job First (SJF) (Page 85)
►Priority

Question No: 16 (Marks: 1) - Please choose one
The basic purpose of _____is to help the users to utilize the hardware resources for completing different tasks in a simplified manner

►Operating system (Page 6)
►Application software
►All Software
►All of the given

Question No: 17 (Marks: 1) - Please choose one
OS helps manages the following except

►Application software
►Bus speed of the system Click here for detail
►Memory
►Virtual memory

Question No: 18 (Marks: 1) - Please choose one
_____ scheduler selects the process from the job pool and put them in main memory.

►Long term (Page 36) rep
►Short term
►Medium term
►Swapper

Question No: 19 (Marks: 1) - Please choose one
A parent process calling _____ system call will be suspended until children process terminates.

►wait click here for detail
►fork
►exit
►exec

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

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

Question No: 20 (Marks: 1) - Please choose one
n-process critical section problem can be solved by using

►**The bakery algorithm (Page 105)**
►Deterministing modeling
►Analytic evaluation
►None of above

بہترین تجربہ وہ ہے جس سے نصیحت حاصل ہو

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

CS604 - Operating System
Solved Quizzes (1 and 2)

Quiz No.1 Dated Dec 04, 2012

Question No: 1 of 10 (Marks: 1) - Please choose one
_____ is a piece of code in a cooperating process in which the process may updates shared data (variable, file, database, etc.).

►Critical analysis
►**Critical section (Page 100)**
►Critical path
►Critical code

Question No: 2 of 10 (Marks: 1) - Please choose one
Round Robin algorithm is similar to _____ scheduling but preemption is added to switch between processes.

►Shortest job first
►Shortest Remaining Time First
►**First Come First Server (Page 88)**
►None of these

Question No: 3 of 10 (Marks: 1) - Please choose one
DOS is single user operating system.

►**True (Page 7)**
►False

Question No: 4 of 10 (Marks: 1) - Please choose one
A process is said to be in critical section if it executes code that manipulates shared data

►**True (Page 100)**
►False

Question No: 5 of 10 (Marks: 1) - Please choose one
When process opens its first file explicitly it will get descriptor number _____

►**1**
►2
►**3** Click here for detail
►4

خوبصورتی علم وادب سے ہوتی ہے لباس و حسن سے نہیں

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

Question No: 6 of 10 (Marks: 1) - Please choose one
A major problem with priority scheduling algorithms is _____.

►Deadlock
►Aging
►**Starvation (Page 86)**
►None of the these

Question No: 7 of 10 (Marks: 1) - Please choose one
All threads within a process share the _____ address space.

►Same
►**Different (Page 71)**

Question No: 8 of 10 (Marks: 1) - Please choose one
_____ displays information about the top processes.

►Is
►Cs
►**Top (Page 67)**
►Cd

Question No: 9 of 10 (Marks: 1) - Please choose one
The scheduling of _____ are done by the operating system.

►**Kernel threads (Page 73)**
►User level threads
►Both kernel and user level thread
►None of the give option

Question No: 10 of 10 (Marks: 1) - Please choose one
In Unix/ Linux, by default the standard output file is attached to the _____

►File
►**Screen (Page 59)**
►Printer
►Scanner

Question No: 1 of 10 (Marks: 1) - Please choose one
POSIX is a standard developed by ANSI

►IEEE (not sure)
►**ISO**
►ACM

تم اچھا کرو زمانہ تم کو برا سمجھے یہ اس سے بہتر ہے کہ تم برا کرو اور زمانہ تم کو اچھا سمجھے

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

Question No: 2 of 10 (Marks: 1) - Please choose one
_____ is the basis of queuing theory which is branch of mathematics used to analyze systems involving queues and servers.

►**Little's Formula (Page 96)**
►Deterministic modeling
►Queuing Theory
►Queuing Analysis

Question No: 3 of 10 (Marks: 1) - Please choose one
_____ is a solution to the problem of indefinite blockage of low-priority processes.

►Starvation
►Deadlock
►**Aging (Page 87)**
►None of the these

Question No: 4 of 10 (Marks: 1) - Please choose one
The priority of a process can be changed using _____ command.

►**nice (Page 94)**
►cmd
►Cat
►grep

Question No: 5 of 10 (Marks: 1) - Please choose one
Batch programs are usually _____ programs.

►Interactive
►**Non-interactive** click here for detail
►Foreground
►Preemptive

Question No: 6 of 10 (Marks: 1) - Please choose one
A process consists of _____
►One or more threads
►Code
►Data
►**All of the given** click here for detail

Question No: 7 of 10 (Marks: 1) - Please choose one
/usr/X11R6 is used by the X Window System.

►**True (Page 27)**
►False

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

Question No: 8 of 10 (Marks: 1) - Please choose one
command displays the contents of current working directory.

►**Is (Page 28)**
►Cs
►Mv

Question No: 9 of 10 (Marks: 1) - Please choose one
Linux uses _____ directory to store system configuration files.

►/bin
►/dev
►/boot
►**/etc (Page 26)**

Question No: 10 of 10 (Marks: 1) - Please choose one
User mode can run the Privileged instructions.

►**1 (Page 11)**
►0

Question No: 1 of 10 (Marks: 1) - Please choose one
If your processor does not have two slots empty in Per Process File Descriptor Table, then your _____ system call will fail.

►**Pipe (Page 55)**
►read
►write
►open

Question No: 2 of 10 (Marks: 1) - Please choose one
First _____ entries in Per Process File Descriptor Table are used as soon as the process is created.
►1
►2
►3
►**4 (Page 54)**

Question No: 3 of 10 (Marks: 1) - Please choose one
The number of processes completed per unit time is called _____.

►Turn around time
►**Throughput (Page 83)**
►Response time
►Dispatch latency

Question No: 4 of 10 (Marks: 1) - Please choose one
The procedure "The time at which the process finished working MINUS the arrival time of the process MINUS CPU burst for that process" will help calculate the _____.

►on-preemptive Shortest Job First scheduling.
►**Preemptive Shortest Job First scheduling. (Page 85)**
► FCFS
►RR Scheduling

Question No: 5 of 10 (Marks: 1) - Please choose one
/opt is used for storage of large applications.
►**True (Page 27)**
►False

Question No: 6 of 10 (Marks: 1) - Please choose one
____ is a virtual directory in Linux and Unix.
►**/proc (Page 27)**
►/temp
►/ver
►/boot

Question No: 7 of 10 (Marks: 1) - Please choose one
The Home Directory for superuser in Linux and Unix is
►/home
►**/root (Page 27)**
►None of the given

Question No: 8 of 10 (Marks: 1) - Please choose one
Linux Treats Devices as Files.
►**True (Page 26)**
►False

Question No: 9 of 10 (Marks: 1) - Please choose one
An absolute pathname starts with the root directory (/) and a relative pathname starts with your home directory.

►**True (Page 25)**
►False

Question No: 10 of 10 (Marks: 1) - Please choose one
A pathname is the list of directories separated by _____.

►#
►$
►&
►/ **(Page 25)**

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

Question No: 1 of 10 (Marks: 1) - Please choose one
_____ determines How to do something.

►**Mechanism (Page 24)**
►Policy
►Mechanism and Policy:
►None of the given

Question No: 2 of 10 (Marks: 1) - Please choose one
User Goal of OS is that It easy to use, reliable, safe and fast.

►**True (Page 24)**
►False

Question No: 3 of 10 (Marks: 1) - Please choose one
We can install and run multiple OS by using VMWare.

►**True** **click here for detail**
►False

Question No: 4 of 10 (Marks: 1) - Please choose one
Mach, MacOS X Server, QNX, OS/2 and Windows NT are examples of OS Based on _____.

►Layered
►**Micro Kernal (Page 22)**
►Virtual Machine
►None of The Given

Question No: 5 of 10 (Marks: 1) - Please choose one
In Layered Approach of OS, the Layer highest Layer is User Interface layer.

►**True (Page 21)**
►False

Question No: 6 of 10 (Marks: 1) - Please choose one
In Layered approach of OS, Lowest Layer is known as _____.

►Software Layer
►**Hardware Layer (Page 21)**
►Lower Level Layer
►None of The Given

Question No: 7 of 10 (Marks: 1) - Please choose one
Operating System is the Manager of Hardware Resources.
►**True (Page 6)**
►False

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

Question No: 8 of 10 (Marks: 1) - Please choose one
An operating system is a control program that manages the execution of user programs to prevent errors and improper use of a computer.

►**True (Page 6)**
►False

Question No: 9 of 10 (Marks: 1) - Please choose one
The bottom-up view is that operating system is a resource manager who manages the hardware and software resources in the computer system.

►**True (Page 6)**
►False

Question No: 10 of 10 (Marks: 1) - Please choose one
_____ determines What will be done.

►Mechanism
►**Policy (Page 24)**
►Mechanism and Policy
►None of the given

Question No: 1 of 10 (Marks: 1) - Please choose one
copy file1 file2 is an example of _____ OS view.

►**Top down (Page 6)**
►Bottum Up

Question No: 2 of 10 (Marks: 1) - Please choose one
The Top-down view is that it is a program that acts as an intermediary between a user of a computer and the computer hardware, and makes the computer system convenient to use.

►**True (Page 6)**
►False

Question No: 3 of 10 (Marks: 1) - Please choose one
Managing Secondary Storage Involves all of the Following except

►Allocating storage space
►Deallocating Storage
►**Prevent Overwritting (Page 5)**
►**Insure integrity of shared data**

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

Question No: 4 of 10 (Marks: 1) - Please choose one
The Purpose of Operating System is to generate Executable Programs and to _____ them.

►Regenetrate
►**Execute (Page 5)**
►Store
►Remove

Question No: 5 of 10 (Marks: 1) - Please choose one
Users are the People, machines or computers that uses the Hardware resources.

►**True (Page 4)**
►False

Question No: 6 of 10 (Marks: 1) - Please choose one
Database, Complier, Video games are examples of _____.

►Hardware
►**Application (Page 4)**
►Operating System
►Users

Question No: 7 of 10 (Marks: 1) - Please choose one
Which of the Following is not an Operating System.

►Linux
►Unix
►Windows Xp
►**Datebase (Page 7)**

Question No: 8 of 10 (Marks: 1) - Please choose one
Operating system enables the user to use the Hardware Resources.

►**True (Page 4)**
►False

Question No: 9 of 10 (Marks: 1) - Please choose one
Which of the following is NOT a Hardware Resource.

►CPU
►**OS (Page 4)**
►I/O Devices
►Memory

خاموشی غصے کا بہترین علاج ہے

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

Question No: 10 of 10(Marks: 1) - Please choose one
Hardware provide basic computing resource.

►**True (Page 4)**
►False

Question No: 1 of 10 (Marks: 1) - Please choose one
_____ is a preemptive scheduling algorithm.

►First Come First Serve
►Shortest Job First
►**Round Robin (Page 89)**
►None of these

Question No: 2 of 10 (Marks: 1) - Please choose one
The priorities of processes in the _____ group remain fixed.

►**Kernel (Page 93)**
►User

Question No: 3 of 10 (Marks: 1) - Please choose one
The process of switching from one process to another is called latency.
►True
►**False (Page 34)**

Question No: 4 of 10 (Marks: 1) - Please choose one
In Unix/ Linux, by default the standard input file is attached to the _____
►Mouse
►**Keyboard (Page 55)**
►Light pen
►Joystick

Question No: 5 of 10 (Marks: 1) - Please choose one
The nice value helps in assigning _____ to a process.

►**Priority (Page 94)**
►Weight
►Time
►Scheduling

Question No: 10 of 10 (Marks: 1) - Please choose one
_____ integer shows the highest priority of a process in CPU scheduling

►**Small (Page 86)**
►Large

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

Quiz No.2

Question No: 1 of 10 (Marks: 1) - Please choose one
If a system is not in a safe state, there can be No deadlocks.

►True
►**False (Page 137)**

Question No: 1 of 10 (Marks: 1) - Please choose one
A dashed line is used to represent a _____ in Resource Allocation Graph.

►**Claim edge (Page 138)**
►Request edge
►Assignment edge
►Allocation edge

Question No: 1 of 10 (Marks: 1) - Please choose one
The process of holding at least one resource and waiting to acquire additional resources that are currently being held by other processes is known as_____.

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

Question No: 1 of 10 (Marks: 1) - Please choose one
In Resource Allocation Graph**,** A _____ Pi →Rj indicates that process Pi may request resource Rj at some time in the future.

►**Claim edge (Page 138)**
►Request edge
►Assignment edge
►Allocation edge

جھوٹ رزق کو کھا جاتا ہے

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

Question No: 1 of 10 (Marks: 1) - Please choose one
If the system can allocate resources to each process in some order and still avoid a deadlock then it said to be in _____ state.

►**Safe (Page 137)**
►Unsafe
►Mutual
►Starvation

Question No: 1 of 10 (Marks: 1) - Please choose one
A condition where a set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set is termed as _____.
►**Deadlock (Page 130)**
►Starvation

Question No: 1 of 10 (Marks: 1) - Please choose one
The following is NOT a classical problem of synchronization

►Bounded buffer problem
►Reader writer problem
►Dining philosopher's problem
►**Counting semaphore problem (Page 118)**

Question No: 1 of 10 (Marks: 1) - Please choose one
The integer value of _____semaphores can range over an unrestricted integer domain.
►**Counting (Page 117)**
►Binary
►Mutex
►Bounded buffer

Question No: 1 of 10 (Marks: 1) - Please choose one
The condition in which a set {P0, P1… Pn} of waiting processes must exist such that P0 is waiting for a resource that is held by P1, P1 is waiting for a resource that is held by P2, and so on, Pn-1 is waiting for a resource held by Pn, and Pn is waiting for a resource held by P0. This condition is known as _____.

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

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

Question No: 1 of 10 (Marks: 1) - Please choose one
You can use the rm file1 command to _____ file1

►Retrieve
►**Remove (Page 30)**
►Make
►modify

Question No: 2 of 10 (Marks: 1) - Please choose one
The correct command for compiling C program named program.c in Linux environment is

►**gcc program.c –o FirstPrgram (Page 31)**
►gcc –o FirstProgram program.c
►gcc –z FirstProgram program.c
►gcc program.c –m FirstPrgram

Question No: 3 of 10 (Marks: 1) - Please choose one
Using _____system, we can create a new process in Linux.

►**Fork (Page 39)**
► exec
►wait
►exit

Question No: 4 of 10 (Marks: 1) - Please choose one
Cooperating processes never share any data, code, memory or state.

►True
►**False (Page 5)**

Question No: 5 of 10 (Marks: 1) - Please choose one
____ command display the status of a process.

►ls
►**ps (Page 66)**
►gcc
►cat

Question No: 6 of 10 (Marks: 1) - Please choose one
Swapper is also termed as Short term scheduler.

►True
►**False (Page 36)**

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

Question No: 7 of 10 (Marks: 1) - Please choose one
_____ system call is used to write to a file or FIFO or any other IPC channel.

►read
►**write (Page 48)**
►open
►fork

Question No: 8 of 10 (Marks: 1) - Please choose one
A Process 'A' that has finished working but its parent process has also finished its execution. In this state the process 'A' will be called as _____ process.

►Child
►Thread
►**Zombie (Page 42)**
►Fork

Question No: 9 of 10 (Marks: 1) - Please choose one
_____ scheduling allows a process to move between queues.
►Round Robin
►First Come First Serve
►**Multilevel Feedback Queue (Page 92)**
►Shortest Remaining Time First

Question No: 10 of 10 (Marks: 1) - Please choose one
Round Robin algorithm is most suitable for _____.

Time sharing system (Page 88)
Real time systems and batch systems
Running Batch programs
Expert system

Question No: 1 of 10 (Marks: 1) - Please choose one
Kernel is responsible for scheduling the user level threads.
►True
►**False (Page 73)**

Question No: 2 of 10 (Marks: 1) - Please choose one
A ---------- (or an *exception)* is a software-generated interrupt caused either by an error (division by zero or invalid memory access) or by a user request for an operating system service.

► Interrupt
► **Trap (Page 10)**
► Signal
► Process

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

Question No: 3 of 10 (Marks: 1) - Please choose one
Which register holds the smallest legal physical memory address for a process?

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

Question No: 4 of 10 (Marks: 1) - Please choose one
The ----------semaphore provides mutual exclusion for accesses to the buffer pool and is initialized to the value 1.
► **mutex (Page 118)**
► binary
► couting
► none of the given options

Question No: 5 of 10 (Marks: 1) - Please choose one
Binary semaphores are those that have only two values---------
► 0 and n
► 0 and 0
► **0 and 1 (Page 117)**
► None of the given options

Question No: 6 of 10 (Marks: 1) - Please choose one
Physical memory is broken down into fixed-sized blocks, called---------- and Logical memory is divided into blocks of the same size, called ----------

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

Question No: 7 of 10 (Marks: 1) - Please choose one
A page table needed for keeping track of pages of the page table is called ----------
► 2-level paging
► **Page directory (Page 173)**
► Page size
► Page table size

Question No: 8 of 10 (Marks: 1) - Please choose one
The address generated by the CPU, after any indexing or other addressing-mode arithmetic, is called a --------*address*, and the address it gets translated to by the MMU is called a ---------*address*.

►**Virtual, physical** **click here for detail**
► Hexadecimal, Binary,
► Valid, invalid
► Physical, Virtual

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

Question No: 9 of 10 (Marks: 1) - Please choose one
Each page is a power of -------- bytes long in paging scheme.

► 2
► 3
► 4 (Page 167)
► 5

Question No: 10 of 10 (Marks: 1) - Please choose one
Which part of the computer system helps in managing the file and memory management system?

► Operating System (Page 5)
► Device Drivers
► Application Software
► Hardware

Question No: 1 of 10 (Marks: 1) - Please choose one
Which of the following is correct definition for wait operation?
► wait(S) { (Page 111)
while(S<=0)
;// no o
S--;
}
► wait(S) {
S++;
}
► wait(S) {
while(S>=0)
;// no op
S--;
}
► wait(S) {
S--;
}

Question No: 3 of 10 (Marks: 1) - Please choose one
In deadlock detection and recovery algorithm, a deadlock exists in the system if and only if the wait for graph contains a _____

► Cycle (Page 147)
► Graph
► Edge
► Node

اطمینان قلب چاہتے ہو تو حسد سے دور رہو

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

Question No: 4 of 10 (Marks: 1) - Please choose one
---------- register contains the size of the process

►Base register
►Index register
►**Limit register (Page 13)**
►Stack pointers register

Question No: 5 of 10 (Marks: 1) - Please choose one
The ----------scheme is not applicable to a resource allocation system with multiple instances of each resource type.

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

Question No: 6 of 10 (Marks: 1) - Please choose one
_____ algorithm is used in Deadlock avoidance.

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

Question No: 7 of 10 (Marks: 1) - Please choose one
What do we name to an address that is loaded into the memory-address register of the memory?

►Logical address
►**Physical address (Page 155)**
►Binary addresses
►None of the given options

Question No: 8 of 10 (Marks: 1) - Please choose one
Cache is non-volatile memory.

►True
►**False (Page 153)**

Question No: 9 of 10 (Marks: 1) - Please choose one
A system call_____

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

Question No: 10 of 10 (Marks: 1) - Please choose one
The condition where a set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set, is termed as _____.

►**Deadlock (Page 130)**
►Starvation

Question No: 1 of 10 (Marks: 1) - Please choose one
Banker's algorithm is used for _____

►**Deadlock avoidance (Page 140)**
►Deadlock detection
►Deadlock prevention
►Deadlock removal

Question No: 2 of 10 (Marks: 1) - Please choose one
The _____ requires that once a writer is ready, that writer performs its write as soon as possible , if a writer waiting to access the object, no new readers may start reading.

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

Question No: 3 of 10 (Marks: 1) - Please choose one
_____ is an integer variable accessible through wait and signal which are atomic operations.

►**Semaphore (Page 111)**
►Mutex
►Busy w aiting
►Signal

Question No: 4 of 10 (Marks: 1) - Please choose one
The integer value of _____ semaphores can not be greater than 1.

►Counting
►**Binary (Page 117)**
►Mutex
►Bounded buffer

Question No: 5 of 10 (Marks: 1) - Please choose one
Starvation is infinite blocking caused due to unavailability of resources.

►**True (Page 115)**
►False

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

Question No: 4 of 10 (Marks: 1) - Please choose one
Operating System provides services such as Managing Primary and Secondary Storage, Processes and Allowing user to manage his/her files and directories.

►**True (Page 5)**
►False

Question No: 5 of 10 (Marks: 1) - Please choose one
_____ is used in real time operating systems.

►**Non-preemptive scheduling** **Click here for detail**
►Preemptive scheduling
►Dispatching scheduling
►FCFS scheduling

Question No: 1 of 10 (Marks: 1) - Please choose one
Preventing a condition of _____ to happen, deadlocks can be prevented to happen.

►Critical region
►**Circular wait (Page 136)**
►Monitors
►Critical section

اپنی مرضی اور اللہ کی مرضی میں فرق کا نام غم ہے

اس سے پہلے کہ تمہیں شہوت فتنے میں ڈالے نکاح کر لو

وہ لوگ مبارک ہیں جو الفاظ سے نصیحت نہیں کرتے بلکہ عمل سے کرتے ہیں

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

CS604- OPERATING SYSTEM
(SOLVED MCQs)
FROM MIDTERM PAPERS
LECTURE (1-22)

BC190202640@vu.edu.pk
Junaidfazal08@gmail.com

For More Visit: vulmshelp.com

JUNAID MALIK
(0304-1659294)

1. When a process has undivided access to a shared piece of code than no other process can execute code, this state is called_.
   - Race condition
   - Progress
   - Mutual exclusion Page 98
   - Bounded waiting
2. Critical section is a place where certain shared structure is updated. Its solution required certain precaution: one of them is the access to critical section by one process at a time only. What this condition is termed as?
   - Progress
   - Entry section
   - Mutual exclusion Page 97
   - Bounded waiting
3. Which of the following is not a major advantage of multiprogramming system is
   - More than one jobs can be processed at a given time
   - CPU utilization can be increases
   - Jobs can be completed quickly

     If more process can exist, the rest must wait until the CPU is free and can be rescheduled page 31
4. We can suspend a foreground process by pressing___which sends a STOP/SUSPEND signal to the process.
   - Ctrl-w
   - Ctrl-x
   - Ctrl-y

   □□□□□Ctrl-z pg#65
5. _____command in LINUX is used to copy file.
   - ls
   - cp pg#27

- mv
- mkdir

6. System calls provide the interface between a___ and the Operating System.
   - User
   - Machine
   - Process pg#15
   - Kernel

7. In critical section problem___requirement illustrates that, "If no process is executing in its critical section and some processes wish to enter their critical sections, then only those processes that are not executing in their remainder section can participate in the decision on which will enter its critical section next, and this selection cannot be postponed indefinitely. "
   - Mutual exclusion
   - □□□□□Progress pg#98
   - Bounded waiting
   - Slow execution

8. Batch programs are usually _____ programs.
   - Interactive
   - Non-interactive My Point of View
   - Foreground
   - Preemptive

9. You can use the mv file1 file2 command to move_____
   - File1 to file 2 (Page 27)
   - File 2to file 1
   - This command will not work for moving files
   - None of the option correct

10. The_____ defines an operating system as a bridge between computer user and hardware for a user's convenience.
   - System-view
   - Bottom-up view
   - Top-down view (page 03)
   - Layered-view

11. The section of code before the critical section is called____.
    - **Entry section (Page 97)**
    - Remainder section
    - Exit section
    - Crystal section AL-JUNAID INSTITUTE OF GROUP

12. _____are used by shell commands to pass data from one shell pipeline to another, without creating temporary files.
    - Pipes
    - Message queues
    - BSD sockets
    - **FIFOs (Page 58)**

13. system call is used to create a child process.
- **Fork (page 48)**
- Parent
- Child
- None of them

13. The problem with priority scheduling algorithm is___.
    - Deadlock
    - **Starvation (page 83)**
    - Aging
    - Nice value

14. The basic purpose of___is to help the users to utilize the hardware resources for completing different task in a simplified manner.
    - Operating system (page 01)
    - Application software
    - All software
    - All of the given

15. If your processor does not have two slots empty in Pre Process file Descriptor Table, then your_system call will fail.

□□□□□**Pipe (page 52)**

- Read
- Write
- Open

16. Which of the following are TRUE for direct communication?
    - A communication link can be associated with N number of process(N=maximum number of processes supported by system)
    - **A communication link can be associated with exactly two processes (pg 44)**
    - Exactly N/2 links exist between each pair of processes(N=maximum number of processes supported by system)
    - Exactly two links exist between each pair of processes

17. POSIX is a standard develop by__.
    - ANSI
    - **IEEE My Point of View**
    - ISO
    - ACM

18. Keeping in mind scheduling algorithm, when you go to any famous fast food franchise. On entering the store, each customer receive a number. The customer with a lowest number is served next. This algorithm is called_____.
    - Dijkstra's algorithm
    - Simple algorithm
    - Deadlock algorithm AL-JUNAID INSTITUTE OF GROUP
    - **Bakery algorithm (page 102)**

19. Which scheduling algorithm allocates the CPU first to the process that request the CPU first?
    - **First come First served scheduling (page 80)**
    - Shortest job scheduling
    - Priority scheduling

- None of the given

20. Bounded buffer is a buffer of___size.
    - Variable
    - **Fixed (page 41)**
21. The wait operation of the Semaphore basically works on the basic_____system call.
    - Stop()
    - Block()
    - Hold()
    - **Wait() (page 111)**

23. Priority scheduling cannot be pre-emptive.
    - True
    - False Page 83
24. When a process P1 switches from the running state to the waiting state because a I/O request is being completed. This scheduling is called .
    - Non preemptive pg#79
    - Switching
    - Termination
    - Preemptive
25. The priority of a process can be changed using_____command.
    - cmd
    - cat
    - grep
    - Nice
26. In_____addressing, the recipient is not required to name the sender.
    - Symmetric
    - **Asymmetric (page 44)**
    - Both(Symmetric and Asymmetric)
    - None
27. An operating system is easily portable between varying hardware designs in structural approach.

- Virtual Machines
- Simple

- Layered
- **Micro kernels Page**

28. The_____approach include the ease of extending the OS.

- Macro Kernels AL-JUNAID INSTITUTE OF GROUP
- Small Kernel
- Monolithic Kernel
- **Micro Kernel (page 19)**

29. Which of the following information is not displayed with top command?

- **Number of threads in a process (page 64)**
- CPU Usage
- Top processes
- Nice value

30. _____processes or thread often need access to shared data and shared resources.

- Parallel
- **□□□□□Concurrent (page 95)**
- Single
- Temporary

31. To terminate a process <ctrl-c> is pressed, which signal it actually passes to process for termination?

- SIGTERM
- **□□□□□SIGINT (page 66)**
- SIGKILL
- SIGTRAP

32. UNIX System V scheduling uses queues, which run_algorithm.

- First Come First Serve
- Round Robin Page 90
- Shortest Job First

- Shortest Remaining Time First

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

- **Operating system (page 12 & 13)**
- Device Drivers
- Application Software
- Hardware

34. CPU bound processes are scheduled before short or I/O bound processes in _____scheduling, therefore, resulting in less CPU and device utilization.

- **First Come First Serve (Page 81)**
- Round Robin
- Shortest Job First
- Shortest Remaining Time First

35. Performance measures of scheduling algorithms are calculated with charts, in order to evaluate an algorithm for a particular workload.

- Simulation
- Analytic evaluation
- **Deterministic modeling (Page 92)**
- Queuing Models

36. In multi-threaded process thread () take two argument, they are used to take _____and____.

- Program counter value, address space limit.
- New thread ID, process name
- Register count, program counter
- New thread function name, new thread ID Page 68

37. In the Bakery algorithm, processes are prioritized based on highest ticks among computing processes.

- True
- False Page 102

38. The solution type where in critical section we use "entry section and "exit section" is
called kind of____.
- Hardware solution
- Assumption

□□□□□Software Solution (Page 98)

- Operating System Solution

39. How many modes are supported in operating system?
- 1
- **2 My point of View**
- 3
- 4

40. ___Scheduler salects the process from the JOB Pool and put them in main memory

- **Long term (Page 33)**
- Short term
- Medium term
- Swapper

AL-JUNAID INSTITUTE OF GROUP

41. A parent process calling_system call will be suspended until children processes terminate.

□□□□□Wait (Page 36)

- Fork
- Exit
- Exec

42. A time-sharing system is

- Multi user

- Multitasking
- Interactive

- **All of these (Page 06)**

43. Pipes simply used on the command line to connect the standard input of one process to the standard input of another. Which of the following syntax is correct use of command line Linux/UNIX pipes?

- Cmd1 % cmd2 %.....%cmdN
- **Cmd1 | cmd2 |……| cmdN Page 53**
- Cmd1 $ cmd2 $.....$ cmdN
- Cmd1 & cmd &……..&cmdN

44. Which is not basic computing hardware?

- CPU
- Memory
- Compact Disc (Page 1)
- I/O Devices

45. Consider a system of N processes (Po, P1……..Pn-1). Each process in its critical section and process may be changing common, updating a table. No other process is allowed to execute in its section. This problem is called_____.

- Bakery algorithm

- **N-Process Critical Section (Page 102)**
- N-Mutual exclusion
- Deadlock algorithm

46. _____Scheduling can be made into a processor sharing approach.

- First Come First serve
- Round Robin (Page 86)
- Shortest remaining time

- Shortest Job First

47. Linux uses___directory to store system configuration files.
    - /bin
    - /dev
    - /boot
    - /etc (page 23)

48. A thread shares its resources (like data section, code section, open files, signal) with____
    - Other process similar to the one that the thread belongs to
    - Other threads that belong to similar processes
    - □□□□□Other thread that belong to the same processes pg#69+Google
    - All of the mentioned

49. _____is a piece of code in a cooperating process in which the process may update shared data (variable, file, database, etc).
    - Critical Analysis
    - □□□□ Critical Section pg#97
    - Critical Path
    - Critical Code

50. A program in execution is called a ___
    - Command
    - □□□□□Process pg#12
    - Software
    - Compiler

51. This layered approach is taken to its logical conclusion in the concept of a
    - Virtual Machine pg#19
    - Logical Machine AL-JUNAID INSTITUTE OF GROUP
    - Physical Machine
    - None of the given

52. CPU bound processes are scheduled before short or I/O bound processes in scheduling therefore, resulting in less CPU

and device utilization.

- Shortest Job First
- Shortest Remaining Time first

□□□□□First Come First Serve (page 80)

- Round-Robin

53. Using_____system, we can create a new process in Linux.
    - Fork pg#36
    - Exec
    - Wait
    - Exit
54. A shell command mkfifo can be used to create a/an _
    - Named pipe pg57
    - Message queue
    - Pipe
    - Unnamed pipe

55. While using the read /write system call which data type is used to return the size of file to buffer from file descriptor fd?
    - Pipefd
    - SSIZE-MaX pg#45
    - FIFO
    - Ssize_t
56. For reading input, which of the following system call is used?

- write
- rd

□□□□□read pg#45

- change

57. _____multi-threading model provides full concurancy.

- One-to-Many
- Many-to-Many
- One-to-One pg#71

- Many-to-One

58. A solution to the critical section must satisfy the following requirements except
    - Progress
    - Mutual Exclusion
    - Bounded Waiting
    - Race Condition pg#98

59. Which is not a parameter of pthread_create().
    - Function
    - Return status pg#73
    - Thread ID
    - Thread attributes

60. Preemptive SJF (Shortest Job First) scheduling is sometimes called scheduling.
    - shortest time-last
    - shortest remaining-time-first pg#82
    - smallest time-first
    - biggest time-first

61. The file descriptor for Standard Input (stdin) is
    - 0 pg#52
    - 1
    - 2
    - 3

62. Rather than maximizing CPU utilization and use of peripheral devices, ____ systems are for maximizing user convenience and responsiveness.
    - Single-user Page 4
    - Batch
    - Multi-user Batch
    - Time-sharing

63. The Operating system is a layer of software between_____and___.
    - Hardware, software application My Point Of View

- Kernel, hardware
- Dos, Windows
- Windows, Kernel

64. You can have a thread wait for another thread with the same process by using the system call.

□□□□ pthread_join() pg#73

- pthread_exit()
- pthread_create()
- pthread_terminate()

65. In indirect communication between processes P and Q___

- There is another process R to handle and pass on the messages between P and Q.
- There is another machine between the two processes to help communication
- There is mailbox to help communication between P and Q Page 44
- A link is associated with exactly two processes

66. Which process can be affected by other processes executing in the system?

□□□□ cooperating process pg#41+Google

- child process
- parent process
- init process

67. _____are used to show executions of processes.

- Gantt charts pg#85
- Time slice
- Quantum
- Sequence

68. The ready queue in RR (Round Robin) scheduling algorithm is referred to as _____.

- Time Quantum
- Time Slice
- LIFO (Last in First Out)
- Circular Queue pg#85

69. Linux redirection features can be used to_and attach other files with them for a single execution of a command.
    - detach the default files from stdin and stdout
    - detach the default files from stdin and stderr
    - □□□□□detach the default files from stdin, stdout and stderr pg 55
    - detach the default files from stdout, and stderr

70. Linux redirection features can be used to_and attach other files with them for a single execution of a command.
    - detach the default files from stdin and stdout
    - detach the default files from stdin and stderr
    - □□□□□detach the default files from stdin, stdout and stderr pg 55
    - detach the default files from stdout, and stderr

71. A____is an integer variable that, apart from initialization is accessible only through standard atomic operations.
    - Mutual Exclusion
    - Semaphore pg#108
    - Mutex
    - Busy waiting

72. _____displays information about the top processes.
    - Is
    - Cs
    - Top Page 64
    - Cd

73. In critical section problem, each process must first request permission to enter its critical section. The section of code implementing this request is called the_____.
    - Remainder section
    - □□□□□Entry section page 97
    - Exit section
    - Sub section

74. _____scheduling allows a process to move between queues.
    - Round Robin
    - First Come First Serve
    - □□□□□Multilevel Feedback Queue page 89

- Shortest Remaining Time First

75. To terminate a process <ctrl-c> is pressed, which signal it actually pass to process for termination?
    - SIGTERM
    - SIGINT pg#66
    - SIGKILL
    - SIGTRAP
76. The time it takes for the dispatcher to stop one process and start another running is known as the_
    - Dispatch latency Page 79
    - Dispatch time
    - Stop time
    - Sleep time
77. In_____inter process communication, a sender mention the name of a recipient.
    - Indirect
    - Direct pg#43
    - Automatic
    - Synchronous
78. Consider three processes in scheduling. Here are the waiting time for three processes P1=5,P2=10,P3=3.Which one of the following is correct average waiting time per process?
    - 3
    - 2
    - 18
    - **6 My Point of View**
79. The priorities of processes in the_____group remain fixed
    - Kernel pg#90
    - User
80. _____algorithm is used for solving n-process critical section problem.

- Bankers
- Bakery page 102
- Babbles
- None of the given

81. The situation in which no context switching is required in multiprocessor system is referred to as_____.
    - Busy waiting
    - Spin lock pg#110
    - Interrupt
    - Scheduler
82. _system call is used to write to a file or FIFO or any other IPC channel.
    - Read
    - Write pg#45
    - Open
    - Fork
83. In LINUX/UNIX environment Ali want to know the number of processes running on the system and their status, number of CPUs in the system and their usage, amount of main memory and its usage. Which of the following command will help in this regard?
    - $ ps
    - $ gcc
    - $ fifo
    - $ top Page 64
84. IN the result displayed by ls –l command.5th column shows
    - Group of the owner
    - File size in bytes page 26
    - Time last upated
    - Owner of the file
85. In Unix/Linux by default the standard input file is attached to the
    - Mouse

□□□□□Keyboard pg#52

- Light pen
- Joystick

86. In which of the following system multiple user are allowed to used the computer simultaneously?

- Single user
- Two user
- Three user
- Multi user Page 6

87. In the layered approach of operating system___.

- Bottom Layer(0) is the user interface

□□□□ Highest layer(N)is the user interface pg#18

- Bottom Layer(N) is the hardware
- Highest Layer(N) is the hardwar

88. A heavy weight process.

- Has multiple threads of execution

□□□□□Has a single thread of execution pg#67

- Multiple or single thread
- None of the mentioned

89. Is a solution of the problem of indefinite blockage of low-priority processes.

- Starvation
- Deadlock

□□Aging pg#84

- None of these

90. The child process can

- Be a duplicate of the parent process Page 36
- Never be a duplicate of the parent process
- Cannot have another program loaded into it
- Never have another program loaded into it

91. FIFO's(also known as named pipes)are used for communication between UNIX/Linux system.

- Related and Unrelated
- Parent-child

- Unrelated processes
- Related processes Page 49

92. _____Command is used to change the directory.
    - Is
    - Cp
    - Cd Page 219
    - Mv

93. In Unix/Linux environment, Ayesha wants to know the complete picture of current processes in her session. Which of the following command will help her in this regard?

$ ps Page 63

- $ gcc
- $ top
- $ fifo

94. Which of the following conditions must be satisfied to solve the critical section problem?
    - Mutual Exclusion
    - Progress
    - Bounded waiting
    - All of the mentioned Page 98

95. The_____process creates two FIFOs, FIFO2, and opens FIFO1 for reading and FIFO2 writing
    - Server Page 60
    - Client
    - Both server and client AL-JUNAID INSTITUTE OF GROUP
    - None of the given options

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

- semaphore S
- spinlock Page 109
- locking semaphore
- mutex

97. Kernel is responsible for scheduling the user level threads
- True
- False Page 70

98. Which of the following of two operations are provided by the IPC facility?
- Write & Delete message
- Delete & Receive message
- Send & Delete message
- Receive & send message My Point of View

99. Access of variable semaphore is possible only through two atomic operation _____and
- TestAndSet, Swap
- Lock, Key
- Boolean, integer
- □□□□□Wait, signal pg#108

100. Kill command is used to terminate a process. The syntax of the command is kill {- signal}PID when executed without a signal number, the command sends the_____ signal to the process.
- SIGINT
- SIGKILL
- SIGTERM pg#66
- None of the given option

101. Given below to statement can be categorized in some sort of message passing technique. This type is named as___

1.Send (A message)

2.Receive (B message)

- Synchronization

□□□□ Direct communication pg#43

- Explicit buffering
- Send by copy communication

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

- True

- **False (Page 41)**

103. _____Command display the status of the process.

- Is

□□□□□Ps pg#63

- Gcc
- Cat

104. _____Command display the status of the process.

- Is

□□□□□Ps pg#63

- Gcc
- Cat

105. For undivided and uninterrupted testing and setting of semaphore, uni-processor systems tend to____.

- **Disable interrupts (Page 109)**
- Use bakery algorithm
- Use spinlock
- Use signal

106. A___enables a user process to request the operating system to execute a privileged instruction for it

- Interrupt
- Library call
- System Call

- **Trap (page 8)**

107. We can terminate a thread explicitly by either returning from the thread function or by using the___call.

- express()
- pthread_exit() (page 73)
- win()
- slow()

AL-JUNAID INSTITUTE OF GROUP

108. Ali is an operating system designer. One user requirement regarding OS is easy to debug and modify. In your opinion which one of the following OS structure Ali needs to follow?

- Simple OS Approach
- Micro kernels
- **Layered approach My Point of View+G**
- Virtual machine

109. Is not system call in Linux.

- **Mkfifo (Page 57)**
- Mknod
- Read
- Open

110. _____is also called swapper .

- Long term
- Short term
- **Medium term pg#34**
- Swap space

111. After fork() system call is made, parent and child process have their separate copy of_____.

- Environment
- Nice value

- File mode creation mask
- **File descriptors pg#38**

112. Critical section problem is to___ the concurrent execution of cooperating process.
    - Update
    - Srialize My Point of View
    - Stop
    - Start

113. Round Robin algorithm is most suitable for___.
- **Time sharing system (Page 85)**
- Real time system and batch systems
- Running batch progress
- Expert system

114. Concurrent processes must be synchronized to prevent_.
- Mutual exclusion
- **Race condition (Page 96)**
- Critical Section
- Waiting

115. In instruction TestAndSet mutual exclusion implementation is done by declaring a

Boolean variable lock_____
- Initialized as zero
- Initialized as true
- Initialized as 1
- **Initialized to false (Page 106)**

116. The kernel is a computer program that manages___request from software
- **Input/output My Point of View**
- Process
- Hardware
- Software

117. _____is used in real time operating systems.
- Non-preemptive scheduling
- **Preemptive scheduling My Point of View**
- Dispatching scheduling

- FCFS scheduling

118. Which of the following provide interface to access the services of the operating system?

- API

□□□□□System call (PAGE 15+Google)

- Library call
- Assembly Instructions

119. In process management some of the jobs can be suspended for some time in order to give other job a chance to be in execution. Which command can be used to place back a suspended command?

□□□□□fg (Page 65)

- top
- ps
- bg

120. ~symbol in the pathname ~/docs/cs604/handouts represented___.

- Root directory
- **Login directory pg#22**
- Parent directory
- Child directory

121. Which of the following statement is not true regarding the cooperating processes?

- It provides an environment to allow concurrent user to access same resources.
- It provide an environment to run parallel processes to speedup computation.
- It construct the system in a modular fashion, dividing the system functions into separate processes or threads.

□□□□□It may affect or be affected by any other process executing in the system. (Page 41)

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

- Child
- Thread

□□□□□Zombie My Point of View

- Fork

123. The nice value helps in assigning___to process
- **Priority My Point of View**
- Weight
- Time
- Scheduling

124. In critical section problem ____requirement illustrates that "There exists a bound on the number of times that other processes are allowed their critical sections after a process has made a request to enter its critical section and before that request is granted."
- Progress
- Mutual Exclusion
- **□□□□□Bounded Waiting Page 98**
- Race Condition

125. When process opens its first file explicitly it will get descriptor number_____
- 1
- 2
- **□□□□□3 My Point of View**
- 4

126. _____is a solution to critical section problem.
- **Lamport's bakery algorithm My Point Of View**
- Safety algorithm
- Dijikstra's algorithm
- Banker's algorithm

125. Which of the following is used to show the end of the file in UNIX?
- Ctrl+A
- **□□□□□Ctrl+D My Point of View**
- Ctrl+O
- Ctrl+E

126. _____is used to request the OS by the process to take an I/O or initiating child process
- Interrupt
- Trap
- Signal

- **System call(Page 195)**

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

- Multithreading
- Context switching
- Critical section My Point of View
- Pipelining

130. Under_____once the CPU has been allocated to a process the process keeps the CPU until either it switches to the waiting state, finishes its CPU burst, or terminates.

- **Non-preemptive scheduling pg#79**
- Preemptive scheduling
- Force scheduling
- Dynamic scheduling

131. The round-robin RR scheduling algorithm is designed especially for____

- Single-user systems
- Nice systems
- Batch system
- **Time-sharing system pg#85**

132. In producer-consumer problem synchronization is required. On which shared area this synchronization actually effect?

- Entry section
- Exit section
- Counter
- Buffer Page 94

133. A process is_____if it cannot affect or be affected by any other process executing in the system.

- Both(independent and dependent)

- None of the given
- Dependent
- **Independent** **Page 41**

134. Command-line interpreter is also called____ in some operating system
- **Shell** **My Point of View**
- Signal
- API
- Kernel

135. When processes are generated using fork () system call and then after they are coordinated with each other using IPC channel. They are utilizing a separate address space for each process (parent. child) .Kernel resources and IPC channel. This makes it a more heavy. Which strategy can be used to make it light?
- Termination of child process when these become more
- Scheduling of each process
- **Use** of batch systems
- **Use of threads (Page 67)**

136. All UNIX and LINUX systems have one thing in common which is___.
- Set of system calls My Point Of View
  - Set of commands
  - Set of instructions
  - Set of text editors

137. First_____entries in pre-process file descriptor table are used as soon as the process is created.
- 1
- 2
- 3 Page 52
- 4

138. _____directory includes essential system boot files including the kernel image
- /bin
- /boot Page 23
- /dev

- /etc

139. A__signal is generated when a write performed to fifo that no process is opened for reading.

- **SIGPIPE (Page 58)**
- Named PIPE
- Unnamed PIPE
  - SIGFIFO

140. Consider a scenario of CPU protection,__is added to the operating system in order to detect and avoid loop in a user program.

- I/O
- **Timer (Page 11)**
- Base and limit register
- Turning interrupt enable off

141. The process id returned to the child process after successful fork system call execution is____.

- **0 (Page 37)**
- 1
- 2
- 3

142. Semaphore S is a/an____ type of variable to use as synchronization tool.

- **Integer (Page 108)**
- Boolean
- Double
- Float

143. In the bakery algorithm to solve the critical section problem_____.

- Each process is put into a queue and picked up in an ordered manner
- **Each process receives a number(may or may not be unique) and the one with the lowest number is served next (Page 102)**
- Each process gets a unique number and the one with the highest

number is served next

- Each process gets a unique number and the one with the lowest number is served next

144. The BSD sockets are used for communication between related or unrelated processes on the same system or_on different systems.

□□□□ Unrelated processes (Page 50)

- Related processes
- Both(related and unrelated)
- Sibling process

145. While a process executes its critical section, other entering processes loop in a continual function in their entry sections, thus causing .

- Bounded waiting
- Race condition
- Mutual exclusion
- **Busy waiting Page 109**

146. A major problem with priority-scheduling algorithm is

- Conclusion
- Aging
- Termination
- **Starvation (Page 83)**

147. The manual pages can be read in Linux using__command.

□□□□□Man (Page 24)

- wan
- desc
- help

148. The creating process is called a___process while the new process are called the___of the process.

- Children, parent
- Zombie, single
- None of the given
- Parent,children Page 35

149. UNIX system V Scheduling algorithm in every second, the priority number of all

those processes that are in the main memory and ready to run is updated by using the following formula:

- Priority # = Threshold priority + nice
- Priority # = (Recent CPU Usage)/2 + nice
- Priority # = (Recent CPU Usage)/2 + Threshold priority
- Priority #=(Recent CPU Usage)/2+Threshold priority+nice Page 90

150. An operating system is a /an___ that manages the execution of user programs to prevent errors and improper use of a computer
   - Disk
   - Application program
   - **Control program Page 03**
     - Physical unit

151. State of process transits from running to ready because of__.
- Fork()
- Scheduler dispatch
- I/O
- Interrupt Page 79

152. Taking the CPU from one process and giving the CPU to another process is termed as
   - **Context Switching (Page 31)**
   - Dispatching
   - Swapping
   - Tracking

153. In priority-scheduling algorithm equal priority processes are scheduled in order.
   - **FCFS (First Come First Serve) (Page 83)**
   - LIFO (Last in First Out)
   - SJF (Shorted Job First)
   - SRTF (Shortest Remaining Time First)

154. A/an___ is software generated call caused by error, or when a user request for

an operating system service

- **Trap (Page 7)**
- Interrupt
- Signal
- System call

155. Critical section problem can be solved by using how many ways?
   - 4
   - **3 Page 97**
   - 2
   - 1

156. The write() system call may not fail for____reason(s)
   - Invalid argument
   - File size limit for process or for system would exceed
   - Disk is full
   - Valid argument Page 51

157. If a parent exits terminating its children processes, these children are handed over
process.
   - Init Page 37
   - Root
   - Daemon
   - Sibling

159. The_____system call suspends the calling process until one of the immediate children terminate.

□□□□□**Wait (Page 39)**

- Signal
- Trap
- Interrupt

160. The correct command when pipes are used on the command line to connect the standard input of one process to the standard input of another is.

- Cmd1 % cmd2% …… %cmdN

□□□□□**Cmd1 | cmd2 | ……. | cmdN (Page 53)**

- Cmd1 $ cmd2 $ …… $ cmdN
- Cmd1 & cmd2 & …… & cmdN

161. A_____system has well defined, fixed time constraints, and if the system does not produce output for an input within the time constraints, the system fail.

- Operating
- Real-time Page 7
- Limited
- Andriod

162. In round-robin (RR) scheduling algorithm the CPU scheduler goes around the ready queue, allocating the CPU to each process for time interval of up to__time quantum.

- 4 AL-JUNAID INSTITUTE OF GROUP
- 3
- 2
- 1 **(Page 85)**

163. In hardware multiprocessor environment two instructions are executed _____ which are swap and TestAndSet.

- **Atomically (Page 105)**
- Automatically
- Manually
- By force

164. We can perform the solution to critical section problem by allowing only one process to enter at a time but another solution hold some instruction use. One of them is TSL, how it is written in program?

- **TestAndSet (Boolean &target) (Page 105)**
- Testandset(Bolean &target)
- TSL (Bolean &target)
- TSL ()

165. The segment of code in which the process may change common variables, update tables, write into files is known as _____

- Program

- **Critical section (Page 102)**
- Non-critical section
- Synchronizing

166. When sender never block because it has an infinite length storage area, then it means it is holding a queue of____capacity.

- Zero
- **Unbounded (Page 45)**
- Defined
- Bounded

167. _____are lowest prioritized processes for scheduling in Kernel Group.

- **User processes (Page 90)**
- Character I/O device control processes
- Block I/O control processes
- File manipulation processes

168. In round-robin (RR) scheduling algorithm, in case the time quantum is very large (infinite), the RR policy remains the same as the___policy.

- LIFO (Last in First Out)
- **FCFS (First Come First Served) (Page 86)**
- SJF(Shorted Job First)
- SRTF (Shortest Remaining Time First

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

- **Mutual exclusion (page 98)**
- Context switching
- Multithreading
- Progress

170. Multilevel feedback_____scheduling allows a process to move between queues.

- List

- **Queue (page 89)**
- Array
- Time

171. The link between two processes P and Q to send and receive messages is called

Communication link (page 43)

- Message-Passing link
- Synchronization link
- All of the mentioned

172. _____command in Linux helps to create a new directory

- ls
- cp
- mv

- **mkdir (page 26)**

173. _____is the basis of queuing theory which is branch of mathematics used to analyze systems involving queues and servers.

- **Little's Formula (page 93)**
- Deterministic Modeling
- Queuing Theory
- Queuing Analysis

174. TSL based solution can implement mutual exclusion, it abbreviates ___

- **TestAndSet Instruction (page 106)**
- Test Swap Instruction
- Trap Set Solution
- Test Set Instrument

175. You can have a thread wait for another thread within the same process by using the _____system call

- **Pthread_join() (page 73)**
- Pthread_exit()
- Pthread_create()

- Pthread_terminate()

176. A signal is an event generated to get attention of
- Hard disk
- **Process (page 7)**
- Trap
- Interrupts

177. Both in two atomic instructions TestAndSet and swap two critical section requirements are satisfied but not _____
- Mutual exclusion
- Progress
- **Bounded waiting (page 106)**
- Assumptions

178. You can use the rm file 1 command to__file 1.
- Retrieve
- **Remove (page 27)**
- Make
- Modify

179. Shared libreries and kernel modules are stored in____directory.
- /bin
- /dev
- /boot
- /lib (page 23)

180. Use of the semaphore create a problem of busy waiting, this wastes CPU cycle that some other process may be able to use productively. This type of semaphore is also called_____
- Semaphore S
- **Spinlock (page 109)**
- Locking Semaphore
- Mutex

181. As a result of <Ctrl-C>, a SIGINT signal is sent to a process. Signal number for SIGINT is _____

- 15
- 1
- 9
- **2 (page 66)**

182. The section of code after the critical section is called_.

- Crystal section
- Entry section
- **Remainder section (page 98)**
- Exit section

183. To prevent a process from keeping CPU for too long, control is shifted to another ready process by invoking a timer .

- When running process finishes its task
- Every time a free space is available in main memory
- **Every time a process finishes its time slice (page 11)**
- When running process makes a system call

184. ---------- View is that operating system is resource manager.

- Top-down
- Single-up
- Down-up
- **bottom-up (page 3)**

185. Co-operating process sharing a piece of code are executed periodically to solve

- Waiting problem
  - Non-concurrency problem
- Aging problem
  - **Critical section problem** **Page 97**

186. Software section to critical section problem can run only in environment __
- Multiprocessor
- Multithreading
- Uniprocessor Page 105
- Separate address spacing

187. command displays the contents of current working directory.
- Is Page 25
- Cs
- Mv
- Cp

188. When processes communicate with each other, they perform communication through
synchronization and utilizing separate address spaces. This action is termed as ___
- Process management
- Synchronization
- Direct/indirect Communication
- Inter Process Communication Page 43

189. We can use semaphore to deal with the number of___process critical section problem. AL-JUNAID INSTITUTE OF GROUP
- 1
- **n (Page 108)**
- n-1
- 2n

190. Which is not considered a role of an operating system?
- Memory allocation and de-allocation of a process
- **Synchronization of cooperating processes My Point of View**
- Program execution
- Protection of a user's file and directories

191. In system call that creates a pipe for IPC. If one of the operation from read/write
would fail then what will be returned as result?
- -1+1

- 1
- -1 Page 46
- 0

192. Which of the following is not true for a time-sharing system?

- Multi user
- Iterative
- CPU sit idle when process requires I/O operation
- Rigid time requirements My Point of View

193. Which could not be the advantage of thread?

- **Separate address space (Page 69)**
- Quick response
- Economical
- Best in multiprocessing environment

194. OS helps manages the following except

- Application software
- Memory
- Virtual memory
- **Bus speed of the system My Point of View**

195. In operating system___ command is used to copy files to same location or different location.

- Copy
- Copi
- Cp
- Both Copy and CP My Point Of View

196. scheduling algorithm can be preemptive or non-preemptive.

- First Come First Serve
- **Shortest Job First (Page 82)**
- Round Robin
- Priority

197. The main characteristic of real time system is.

- Efficiency
- Large virtual memory
- Large secondary storage device

□□□□□Usability My Point of View

198. Processes P1,P2,P3,P4 enter the ready queue at time 0,3,3,11 with burst times

P1=8, P2=2, P3=1, P4=1. With shortest remaining time first ,__process is given CPU at time=3.

- P1
- P2

□□□□□P3 My Point of View

- P4

199. If size of a process is 376052 bytes and its smallest physical memory address is

242785.Its address space cannot exceed beyond___.

- □ 376053
- □ 376051
- □ 618836

□ 618837 My Point of View

200. Which of the following is a correct command line for running in background?

- cat code.c | grep print %

□□□□□cat code.c | grep print & (Page 62)

- cat code.c | grep print #
- cat code.c | grep print @

201. Which of the following command is used to read data from file named academics?

- con<academics
- socket>academics
- ran>academics
- **cat<academics (Page 55)**

202. Using hardware solution to synchronization for complex problems, introduce a new synchronization tool known as___.

- Trap
- **Semaphore (Page 108)**
- SWAP
- TestAndSet

203. In LINUX directory structure, there is_root directory.

- 1 Page 23
- 2
- 4 AL-JUNAID INSTITUTE OF GROUP
- 3

204. The time interval from submission of a process to the time when its output is generated is termed as.

- **Turnaround time (Page 80)**
- CPU burst
- Responsive time
- Throughput

205. The correct command for compiling C program named program.c in LINUX environment is

- gcc -z FirstProgram program.c
- gcc program.c -m FirstProgram
- gcc -o FirstProgram program.c
- **gcc program.c -o First Program (Page 28)**

206. The Kernel is_____user threads

- Is part of
- Unaware of Page 70
- Aware of
- The creater of

207. _____scheduling algorithm is sometimes called shortest remaining time first scheduling algorithm.

- Priority scheduling
- Non-preemptive SJF
- **Preemptive Shortest Job First (Page 82)**

- FCFS

208. You can use the bg command to put the current or a suspended process into the background. What is the correct syntax of bg command from the following?

- bg [%id]
- bg [job#]
- bg [id#]

□□□□□bg [%job_id] (Page 65)

209. The region in the memory that a process is allowed to access is known as_____.

□□□□□Address space (Page 10)

- Instruction
- Operation
- Register

210. In Unix/Linux by default the standard output file is attached to the

- File
- **Screen (Page 52)**
- Printer
- Scanner

211. The procedure "The time at which the process finished working MINUS the arrival time of the process MINUS CPU burst for that process" will help calculate the

.Non-preemptive Shortest Job First scheduling.

- **Preemptive Shortest Job First scheduling. (Page 82**
- FCFS
- RR Scheduling

212. Processes P1, P2, P3 having burst time P1=3, P2=4, P3=3 and turnaround times P1=7, P2=10, P3=9 enter the ready queue at Time=0. With RR scheduling waiting time for process P2 is__.

□ 4 AL-JUNAID INSTITUTE OF GROUP

□ 0

□ 3

□□□□□6 **My Point of View**

213. _____Scheduler takes the process from the ready queue and assigns the CPU with the help of Dispatcher.

- Swapper
- **Short term (Page 79)**
- Long term
- Medium term

214. To display all processes____option is used with ps command/

- -l
- -a
- **-e (Page 63)**
- -u

215. Clock interrupt handler is an example of kernel level thread that is not executed for serving a request by a user thread. It is named as_____.

- EINVAL
- EAGAIN
- □□□□□**LWP (Page 72)**
- POSIX

216. Consider three processes in scheduling. Here are the waiting times for three process P1=4,P2=2,P3=6.Which one of the following is correct average waiting time.

- 2
- 4 My Point of View
- 3
- 6

217. Processes P1,P2,P3 having burst time P1=3,P2=4,P3=3 and turnaround time P1=7,P2=10,P3=9 enter the ready queue at T=0 waiting time for process P2 is__.

- 3

- 4
- 6 My Poit of View
- 0

218. P1,P2,P3,P4,P5 are five processes with tick numbers P1=1,P2=2,P3=4,P4=2,P5=4
then considering Lamport's bakery algorithm_____ process enters critical section after P1.

□□□□□**P2** **My Point of View**

- P3
- P4
- P5

219. Four processes P1, P2, P3, P4 enter the ready queue at time 0 with burst times P1=3, P2=10, P3=4. Waiting time of P3 would be___.

□ 12
□□□□□**13** **My Point of View**
□ 14
□ 17

Question No : 1 of 26 Marks: 1 (Budgeted Time 1 Min)

I MB or 1 megabyte is equivalent to----

Answer (Please select your correct option)

- 1024 bytes
- 1024^2 bytes 
- 1024^3 bytes
- 1000000 bytes

Question No : 2 of 26 Marks: 1 (Budgeted Time 1 Min)

A ---------system collects physically separate, possibly heterogeneous, systems into a single coherent system, providing the user with access to the various resources that the system maintains

Answer (Please select your correct option)

- Distributed 
- Real-time
- Single user
- Time-sharing

Question No : 3 of 26 Marks: 1 (Budgeted Time 1 Min)

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

Answer (Please select your correct option)

- DOS
- Windows
- UNIX 
- None of the given options

Question No : 4 of 26 Marks: 1 (Budgeted Time 1 Min)

Round-Robin Scheduling is a ---------- scheduling algorithm

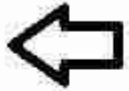
Answer (Please select your correct option)

- Preemptive 
- non-preemptive
- Preemptive or non-preemptive.
- None of the given options

Question No : 5 of 26 Marks: 1 (Budgeted Time 1 Min)

A piece of code in a cooperating process in which the process may update shared data is called----------

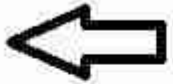
Answer (Please select your correct option)

- Buffer
- Critical section 
- Semaphore
- None of the given options

Question No : 6 of 26 Marks: 1 (Budgeted Time 1 Min)

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

Answer (Please select your correct option)

- Progress
- Bounded waiting 
- Mutual exclusion
- None of the given

Question No : 7 of 26 Marks: 1 (Budgeted Time 1 Min)

A _____ is a piece of code in a cooperating process in which the process may updates shared data (variable, file, database, etc.).

Answer (Please select your correct option)

- critical section 
- critical region
- monitor
- all of the given are correct

Question No : 8 of 26 Marks: 1 (Budgeted Time 1 Min)

Semaphore is a _____ solution to the synchronization problems.

Answer (Please select your correct option)

- Operating system
- Software
- None of the given
- Hardware

Question No : 9 of 26 Marks: 1 (Budgeted Time 1 Min)

In Unix/ Linux, by default the standard input file is attached to the _____.

Answer (Please select your correct option)

- Mouse
- Keyboard
- Light pen
- Joystick

Question No : 10 of 26 Marks: 1 (Budgeted Time 1 Min)

The ---------- are used for communication between related or unrelated processes on the same system or unrelated processes on different systems.

Answer (Please select your correct option)

- None of the given
- SSD socket
- DSD socket
- BSD sockets

Question No : 11 of 26 Marks: 1 (Budgeted Time 1 Min)

The files which are automatically opened by the kernel for every process for the process to read its input from and send its output and error messages are called.....

Answer (Please select your correct option)

- None of the given
- Standard File descriptors
- File descriptors
- Standard files

Question No : 12 of 26 Marks: 1 (Budgeted Time 1 Min)

Bounded Buffer is a buffer of _____ size

Answer (Please select your correct option)

- Variable
- Fixed
- Both
- None of the given

Question No : 13 of 26 Marks: 1 (Budgeted Time 1 Min)

The goal of single user operating system is

Answer (Please select your correct option)

- To maximize utilization of CPU
- To maximize the user responsiveness
- To maximize utilization of peripheral devices
- To maximize throughput

Question No : 14 of 26 Marks: 1 (Budgeted Time 1 Min)

Which type of operating system guarantees that critical tasks must be completed on time?

Answer (Please select your correct option)

- Timesharing system
- Multitasking system
- Batch system
- Realtime system

Question No : 15 of 26 Marks: 1 (Budgeted Time 1 Min)

Which of the following command is used to read data from file named academics?

Answer (Please select your correct option)

- con < academics
- cat < academics
- socket > academics
- ran > academics

Question No : 16 of 26 Marks: 1 (Budgeted Time 1 Min)

_____command puts a foreground process into background.

Answer (Please select your correct option)

- fg
- bg
- Ctrl z
- None of the given

Question No : 17 of 26 Marks: 1 (Budgeted Time 1 Min)

Given the gantt chart, which one of the following is average waiting time

Answer (Please select your correct option)

- 35.66
- 1.33
- 34.66
- 2.33

Question No : 18 of 26 Marks: 1 (Budgeted Time 1 Min)

_____ is solution to the problem of indefinite blockage of low-priority processes.

Answer (Please select your correct option)

- Aging 
- Starvation
- Deadlock
- None of the given

Question No : 19 of 26 Marks: 1 (Budgeted Time 1 Min)

The part of machine level instruction, which tells the central processor what has to be done, is

Answer (Please select your correct option)

- Address
- Locator
- Flip-Flop
- Operation code

Question No : 20 of 26 Marks: 1 (Budgeted Time 1 Min)

Which of the following statements is false?

Answer (Please select your correct option)

- Two different critical sections may be executed concurrently if they do not use the same shared variables.
- Programs with critical sections can never be used simultaneously by more than one process.
- None of the given
- A process wanting to enter a critical section currently in use must wait for the process utilizing the critical section to terminate.

Question No : 1 of 26 Marks: 1 (Budgeted Time 1 Min)

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

Answer (Please select your correct option)

- 1024 bytes
- 1024^2 bytes
- 1024^3 bytes
- 1000000 bytes

Question No : 2 of 26 Marks: 1 (Budgeted Time 1 Min)

A ---------system collects physically separate, possibly heterogeneous, systems into a single coherent system, providing the user with access to the various resources that the system maintains

Answer (Please select your correct option)

- Distributed

- Real-time
- Single user
- Time-sharing

Question No : 3 of 26 Marks: 1 (Budgeted Time 1 Min)

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

Answer (Please select your correct option)

- DOS
- Windows
- UNIX
- None of the given options

Question No : 4 of 26 Marks: 1 (Budgeted Time 1 Min)

Round-Robin Scheduling is a ---------- scheduling algorithm

Answer (Please select your correct option)

- Preemptive

- non-preemptive
- Preemptive or non-preemptive.
- None of the given options

Question No : 5 of 26 Marks: 1 (Budgeted Time 1 Min)

A piece of code in a cooperating process in which the process may update shared data is called----------

Answer (Please select your correct option)

- Buffer
- Critical section

- Semaphore
- None of the given options

Question No : 6 of 26 Marks: 1 (Budgeted Time 1 Min)

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

Answer (Please select your correct option)

- Progress
- Bounded waiting
- Mutual exclusion
- None of the given

Question No : 7 of 26 Marks: 1 (Budgeted Time 1 Min)

A _____ is a piece of code in a cooperating process in which the process may updates shared data (variable, file, database, etc.).

Answer (Please select your correct option)

- critical section

- critical region
- monitor
- all of the given are correct

Question No : 8 of 26 Marks: 1 (Budgeted Time 1 Min)

Semaphore is a _____ solution to the synchronization problems.

Answer (Please select your correct option)

- Operating system

- Software
- None of the given
- Hardware

Question No : 9 of 26 Marks: 1 (Budgeted Time 1 Min)

In Unix/ Linux, by default the standard input file is attached to the _____.

Answer (Please select your correct option)

- Mouse
- Keyboard
- Light pen
- Joystick

Question No : 10 of 26 Marks: 1 (Budgeted Time 1 Min)

The ---------- are used for communication between related or unrelated processes on the same system or unrelated processes on different systems.

Answer (Please select your correct option)

- None of the given
- SSD socket
- DSD socket
- BSD sockets

Question No : 11 of 26 Marks: 1 (Budgeted Time 1 Min)

The files which are automatically opened by the kernel for every process for the process to read its input from and send its output and error messages are called.....

Answer (Please select your correct option)

- None of the given
- Standard File descriptors
- File descriptors
- Standard files

Question No : 12 of 26 Marks: 1 (Budgeted Time 1 Min)

Bounded Buffer is a buffer of _____ size

Answer (Please select your correct option)

- Variable
- Fixed
- Both
- None of the given

Question No : 13 of 26 Marks: 1 (Budgeted Time 1 Min)

The goal of single user operating system is

Answer (Please select your correct option)

- To maximize utilization of CPU
- To maximize the user responsiveness
- To maximize utilization of peripheral devices
- To maximize throughput

Question No : 14 of 26 Marks: 1 (Budgeted Time 1 Min)

Which type of operating system guarantees that critical tasks must be completed on time?

Answer (Please select your correct option)

- Timesharing system
- Multitasking system
- Batch system
- Realtime system

Question No : 15 of 26 Marks: 1 (Budgeted Time 1 Min)

Which of the following command is used to read data from file named academics?

Answer (Please select your correct option)

- con < academics
- cat < academics
- socket > academics
- ran > academics

Question No : 16 of 26 Marks: 1 (Budgeted Time 1 Min)

_____command puts a foreground process into background.

Answer (Please select your correct option)

- fg
- bg
- Ctrl z
- None of the given

Question No : 17 of 26 Marks: 1 (Budgeted Time 1 Min)

Given the gantt chart, which one of the following is average waiting time

Answer (Please select your correct option)

- 35.66
- 1.33
- 34.66
- 2.33

Correct Answer Solved By Hadi
usmanraj20@gmail.com
03228043306

Made By: Waqar Siddhu

Question No : 18 of 26 Marks: 1 (Budgeted Time 1 Min)

_____ is solution to the problem of indefinite blockage of low-priority processes.

Answer (Please select your correct option)

- Aging

- Starvation
- Deadlock
- None of the given

Question No : 19 of 26 Marks: 1 (Budgeted Time 1 Min)

The part of machine level instruction, which tells the central processor what has to be done, is

Answer (Please select your correct option)

- Address
- Locator
- Flip-Flop
- Operation code

Question No : 20 of 26 Marks: 1 (Budgeted Time 1 Min)

Which of the following statements is false?

Answer (Please select your correct option)

- Two different critical sections may be executed concurrently if they do not use the same shared variables.
- Programs with critical sections can never be used simultaneously by more than one process.
- None of the given
- A process wanting to enter a critical section currently in use must wait for the process utilizing the critical section to terminate.

Question No : 1 of 26 Marks: 1 (Budgeted Time 1 Min)

Main memory is a large array of ---------called memory locations ranging in size from hundreds of thousands to billions

Answer (Please select your correct option)

- Interrupts
- Registers
- Digits
- Words

Question No : 2 of 26 Marks: 1 (Budgeted Time 1 Min)

In ---------- model, there is a kernel thread for every user thread

Answer (Please select your correct option)

- Many-to-Many
- None of the given options
- One-to-One
- Many-to-One

Question No : 3 of 26 Marks: 1 (Budgeted Time 1 Min)

A time interval when a process uses CPU only is called

Answer (Please select your correct option)

- CPU burst

- IO burst
- Scheduling
- None of the given options

Question No : 4 of 26 Marks: 1 (Budgeted Time 1 Min)

The part of the program where the shared memory is accessed is called ----------

Answer (Please select your correct option)

- Buffer

- Critical Section
- Semaphore
- None of the given options

Question No : 5 of 26 Marks: 1 (Budgeted Time 1 Min)

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

Answer (Please select your correct option)

- /dev
- /boot
- /sbin
- /lib

Question No : 6 of 26 Marks: 1 (Budgeted Time 1 Min)

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

Answer (Please select your correct option)

- Progress
- Bounded waiting
- Mutual exclusion
- None of the given

Question No : 7 of 26 Marks: 1 (Budgeted Time 1 Min)

All ---------- within a process share the same address space

Answer (Please select your correct option)

- Threads

- Processes
- CPU burst
- IO Burst

Question No : 8 of 26 Marks: 1 (Budgeted Time 1 Min)

In a Multilevel Queue, the foreground queue has ------- scheduling algorithm and background queue has ---------- scheduling algorithm

Answer (Please select your correct option)

- First Come First Serve, Round-Robin
- Round-Robin, First Come First Serve
- Round-Robin, Round-Robin
- First Come First Serve, First Come First Serve

Question No : 9 of 26 Marks: 1 (Budgeted Time 1 Min)

Priority of a process is ---------- to the priority number of the process.

Answer (Please select your correct option)

- Directly proportional
- Equal
- Inversely proportional
- Greater

i think but not sure

Question No : 10 of 26 Marks: 1 (Budgeted Time 1 Min)

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

Answer (Please select your correct option)

- Top-down

- Bottom-up
- Alternate
- None of the above

Question No : 11 of 26 Marks: 1 (Budgeted Time 1 Min)

_____ commands in Linux helps to create a new directory.

Answer (Please select your correct option)

- ls
- cp
- mv
- mkdir

Question No : 12 of 26 Marks: 1 (Budgeted Time 1 Min)

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

Answer (Please select your correct option)

- Long term

- Short term
- Medium term
- Swapper

Question No : 13 of 26 Marks: 1 (Budgeted Time 1 Min)

----------are used for communication between unrelated processes on a system.

Answer (Please select your correct option)

- Pipes
- Named pipe (FIFO)
- BSD Sockets
- none of the given options

Question No : 14 of 26 Marks: 1 (Budgeted Time 1 Min)

The syntax for output redirection is

Answer (Please select your correct option)

- command < output-file
- command >= output-file
- command > output-file
- command =< output-file

Question No : 15 of 26 Marks: 1 (Budgeted Time 1 Min)

The shared buffer can be implemented as a circular array with ..... logical pointers.

Answer (Please select your correct option)

- Five
- Two
- Three
- Four

Question No : 16 of 26 Marks: 1 (Budgeted Time 1 Min)

Limit register contains

Answer (Please select your correct option)

- The size of the process
- Starting address of the process
- Address of the next instruction
- Operand to be fetched

Question No : 17 of 26 Marks: 1 (Budgeted Time 1 Min)

Which of the following command is used to read data from file named academics?

Answer (Please select your correct option)

- con < academics
- cat < academics
- socket > academics
- ran > academics

Question No : 18 of 26 Marks: 1 (Budgeted Time 1 Min)

_____command puts a foreground process into background

Answer (Please select your correct option)

- fg
- bg

- Ctrl z
- None of the given

Question No : 19 of 26 Marks: 1 (Budgeted Time 1 Min)

_____ is solution to the problem of indefinite blockage of low-priority processes.

Answer (Please select your correct option)

- Aging

- Starvation
- Deadlock
- None of the given

Question No : 20 of 26 Marks: 1 (Budgeted Time 1 Min)

What is the requirement for solution to critical section problem?

Answer (Please select your correct option)

- Mutual Exclusion
- Progress
- Bounded Waiting
- All of the given

Question No : 1 of 26 Marks: 1 (Budgeted Time 1 Min)

Main memory is a large array of ---------called memory locations ranging in size from hundreds of thousands to billions

Answer (Please select your correct option)

- Interrupts
- Registers
- Digits
- Words

Question No : 2 of 26 Marks: 1 (Budgeted Time 1 Min)

In ---------- model, there is a kernel thread for every user thread

Answer (Please select your correct option)

- Many-to-Many
- None of the given options
- One-to-One 
- Many-to-One

Question No : 3 of 26 Marks: 1 (Budgeted Time 1 Min)

A time interval when a process uses CPU only is called

Answer (Please select your correct option)

- CPU burst 
- IO burst
- Scheduling
- None of the given options

Question No : 4 of 26 Marks: 1 (Budgeted Time 1 Min)

The part of the program where the shared memory is accessed is called ---------

Answer (Please select your correct option)

- Buffer 
- Critical Section
- Semaphore
- None of the given options

Question No : 5 of 26 Marks: 1 (Budgeted Time 1 Min)

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

Answer (Please select your correct option)

- /dev
- /boot
- /sbin 
- /lib

Question No : 6 of 26 Marks: 1 (Budgeted Time 1 Min)

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

Answer (Please select your correct option)

- Progress
- Bounded waiting 
- Mutual exclusion
- None of the given

Question No : 7 of 26 Marks: 1 (Budgeted Time 1 Min)

All ---------- within a process share the same address space

Answer (Please select your correct option)

- Threads
- Processes
- CPU burst
- IO Burst

Question No : 8 of 26 Marks: 1 (Budgeted Time 1 Min)

In a Multilevel Queue, the foreground queue has ------- scheduling algorithm and background queue has ---------- scheduling algorithm

Answer (Please select your correct option)

- First Come First Serve, Round-Robin
- Round-Robin, First Come First Serve 
- Round-Robin, Round-Robin
- First Come First Serve, First Come First Serve

Question No : 9 of 26 Marks: 1 (Budgeted Time 1 Min)

Priority of a process is ---------- to the priority number of the process.

Answer (Please select your correct option)

- Directly proportional
- Equal
- Inversely proportional
- Greater

Question No : 10 of 26 Marks: 1 (Budgeted Time 1 Min)

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

Answer (Please select your correct option)

- Top-down
- Bottom-up
- Alternate
- None of the above

Question No : 11 of 26 Marks: 1 (Budgeted Time 1 Min)

_____ commands in Linux helps to create a new directory.

Answer (Please select your correct option)

- ls
- cp
- mv
- mkdir

Question No : 12 of 26 Marks: 1 (Budgeted Time 1 Min)

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

Answer (Please select your correct option)

- Long term 
- Short term
- Medium term
- Swapper

Question No : 13 of 26 Marks: 1 (Budgeted Time 1 Min)

----------are used for communication between unrelated processes on a system.

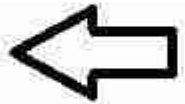
Answer (Please select your correct option)

- Pipes
- Named pipe (FIFO)
- BSD Sockets 
- none of the given options

Question No : 14 of 26 Marks: 1 (Budgeted Time 1 Min)

The syntax for output redirection is

Answer (Please select your correct option)

- command < output-file
- command >= output-file
- command > output-file 
- command =< output-file

Question No : 15 of 26 Marks: 1 (Budgeted Time 1 Min)

The shared buffer can be implemented as a circular array with ..... logical pointers.

Answer (Please select your correct option)

- Five
- Two 
- Three
- Four

Question No : 16 of 26 Marks: 1 (Budgeted Time 1 Min)

Limit register contains

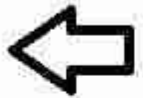
Answer (Please select your correct option)

- The size of the process 
- Starting address of the process
- Address of the next instruction
- Operand to be fetched

Question No : 17 of 26 Marks: 1 (Budgeted Time 1 Min)

Which of the following command is used to read data from file named academics?

Answer (Please select your correct option)

- con < academics
- cat < academics 
- socket > academics
- ran > academics

Question No : 18 of 26 Marks: 1 (Budgeted Time 1 Min)

_____command puts a foreground process into background

Answer (Please select your correct option)

- fg
- bg
- Ctrl z
- None of the given

Question No : 19 of 26 Marks: 1 (Budgeted Time 1 Min)

_____ is solution to the problem of indefinite blockage of low-priority processes.

Answer (Please select your correct option)

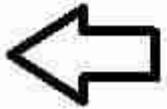
- Aging 
- Starvation
- Deadlock
- None of the given

Question No : 20 of 26 Marks: 1 (Budgeted Time 1 Min)

What is the requirement for solution to critical section problem?

Answer (Please select your correct option)

- Mutual Exclusion
- Progress
- Bounded Waiting
- All of the given



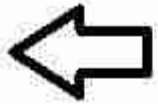
Question: 1 (Marks: 1) Attempted Questions: 22 Total Questions

When process opens its first file explicitly it will get descriptor number _____

Choices:

2 **standard file descriptors**.
Standard input: 0 (keyboard)
Standard output: 1 (display screen)
Standard error: 2 (display screen)
page 52

3



1

4

Question: 2 (Marks: 1) Attempted Questions: 22 Total Questions: 2

Using _____system call, we can create a new process in Unix.

Choices:

- fork
- exec
- wait
- exit

Question: 3 (Marks: 1) Attempted Questions: 22 Total Questions: 26

Which one is not the state of the process?

Choices:

- Waiting
- Running
- Privileged
- New

Operating Systems (CS604)

Question: 4 (Marks: 1) Attempted Questions: 22 Total Questio

To avoid the race condition, the number of processes that may be simultaneously inside their critical section is

Choices:

8

1

16

0

_____ loads, verifies, and executes programs that have been translated into Java byte code.

Choices:

- Java Virtual Machine
- VMWare
- VM Solution
- VM PC

Question: 6 (Marks: 1) Attempted Questions: 22 Total Questions:

A time interval when a process uses CPU only is called _____.

Choices:

- Scheduling
- I/O burst
- CPU burst
- Dispatch latency

Question: 7 (Marks: 1) Attempted Questions: 22 Total Questions: 2

Round robin scheduling usually increases the average waiting time so its performance heavily depends upon _____.

Choices:

- No. of processes
- Time quantum
- Switching
- No. of queues

Question: 8 (Marks: 1) Attempted Questions: 22 Total Questions:

Due to which of the following reason, a parent may terminate the execution of one of its children.

Choices:

- Use more resources than allocated
- Busy other than assigned task
- Start abnormal behavior
- Many jobs in a system

Question: 9 (Marks: 1) Attempted Questions: 22 Total Questio

Which of the following tool is used for inter process communication between related or unrelated processes on the same system or unrelated processes on different system over the network?

Choices:

- Pipe
- BSD socket
- Named pipe (FIFO)
- Message Queue

Question: 10 (Marks: 1) Attempted Questions: 22 Total Questions: 26

The files which are automatically opened by the kernel for every process to read its input from and send its output and error messages are called_____.

Choices:

- File identifiers
- Standard File descriptors
- File descriptors
- Standard files

Question: 11 (Marks: 1) Attempted Questions: 22 Total Questions: 26

In Unix/Linux client server process communication, server process runs in the background by terminating its command line with ----------.

Choices:

- & (selected)
- @
- %
- $

Question: 12 (Marks: 1) Attempted Questions: 22 Total Questio

Each thread has its own ----------.

Choices:

- Process ID (PID)
- Address Space
- Thread ID (TID)
- Open file descriptors

Operating Systems (CS604)

Question: 13 (Marks: 1) Attempted Questions: 22 Total Questions: 26

Preemptive scheme of Short job first scheduling is known as -------.

Choices:

- First Come First Serve Scheduling
- Shortest-Remaining-Time-First
- Priority Scheduling
- Round-Robin Scheduling

Operating Systems (CS604)

Question: 14 (Marks: 1) Attempted Questions: 22 Total Questions: 2

A process terminates by calling the ----------system call.

Choices:

wait

fork

exec

exit

Question: 15 (Marks: 1) Attempted Questions: 22 Total Questions:

The ----------system call is used to open or create a file.

Choices:

open()

read()

write()

close()

Question: 16 (Marks: 1) Attempted Questions: 22 Total Questions:

The syntax for input redirection is

Choices:

- command < input-file
- command > input-file
- command >= input-file
- command =< input-file

Question: 17 (Marks: 1) Attempted Questions: 22 Total Questions: 26

The syntax for output redirection is

Choices:

- command < output-file
- command >= output-file
- command > output-file
- command =< output-file

Question: 18 (Marks: 1) Attempted Questions: 22 Total Questions:

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

Choices:

- Shortest Job First scheduling
- First Come First Serve
- Priority Scheduling
- Round Robin Scheduling

Question: 19 (Marks: 1) Attempted Questions: 22 Total Questions: 2

Process synchronization is a process of helping processes to _____

Choices:

- update the shared data simultaneously
- update the shared data and resources in an orderly fashion
- not to update the shared data or memory
- access data and memory at first come first serve basis

Question: 20 (Marks: 1) Attempted Questions: 23 Total Questions: 26

A parent process calling _____ system call will be suspended until any imediate child terminates.

Choices:

- wait
- fork
- exit
- exec

Virtual University Overseas Examination System V2.0 (x64)

Operating Systems (CS604)

Question: 1 (Marks: 1) Attempted Questions: 25 Total Questions: 26

_____ commands in Linux is used to copy file

Choices:

ls

cp

mv

mkdir

TIME LEFT
17

Inter process communication (IPC) allows process to communicate and to synchronize their actions without sharing the same _____

Choices:

- (•) Address space
- ( ) Messages
- ( ) Pipes
- ( ) Message Queue

Virtual University Overseas Examination System V2.0 (x64)

Operating Systems (CS604)

Question: 3 **(Marks: 1)** **Attempted Questions: 25** **Total Questions: 26**

Pipe system call is used to _____ a pipe

Choices:

- Destroy
- Create
- Modify
- Clear

TIME LEFT
17

Which of the following command is used to read data from file named academics?

Choices:

- con < academics
- cat < academics
- socket > academics
- ran > academics

In a multithreaded model, Which one of the following provides true concurrency?

Choices:

- Many-to-one
- One-to-one
- Many-to-many
- One-to-many

Question: 6 (Marks: 1) Attempted Questions: 25 Total Questions: 26

In _____model, there is a kernel thread for every user thread.

Choices:

- (•) One-to-one
- ( ) Many –to-one
- ( ) Many-to-many
- ( ) One-to-many

_____ loads, verifies, and executes programs that have been translated into Java byte code.

Choices:

- (•) Java Virtual Machine
- ( ) VMWare
- ( ) VM Solution
- ( ) VM PC

_____ is the time taken for the dispatcher to stop one process and start another process.

Choices:

- (•) Dispatch latency
- ( ) Scheduling
- ( ) Context switching
- ( ) Preemption

A time interval when a process uses CPU only is called _____.

Choices:

- Scheduling
- I/O burst
- CPU burst
- Dispatch latency

Question: 10 (Marks: 1) Attempted Questions: 25 Total Questions: 26

A time interval when a process uses I/O devices only is called _____.

Choices:

- CPU burst
- I/O burst
- Scheduling
- Context switching

While two processes in execution, only one can be in critical section at a time , this requirement is named as _____.

Choices:

- Race condition
- Mutual exclusion
- Progress
- Bounded waiting

The file descriptor for Standard Output (stdout) is ----------.

Choices:

0

1

2

3

The file descriptor for Standard error (stderr) is ----------.

Choices:

0

1

2

3

---------- are used by shell commands to pass data from one shell pipeline to another, without creating temporary files.

Choices:

- Pipes
- Message queues
- BSD sockets
- FIFOs

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

Choices:

- First-Come-First-Served (FCFS)
- Round-Robin
- Sorted Job First (SJF)
- Priority

In a Multilevel Queue, the foreground queue has ------- scheduling algorithm and background queue has ---------- scheduling algorithm.

Choices:

- First Come First Serve, Round-Robin
- Round-Robin, First Come First Serve
- Round Robin, Round Robin
- First Come First Serve, First Come First Serve

A process terminates by calling the ----------system call.

Choices:

- (•) wait
- ( ) fork
- ( ) exec
- ( ) exit

The syntax for input redirection is

Choices:

- (●) command < input-file
- ( ) command > input-file
- ( ) command >= input-file
- ( ) command =< input-file

A _____ is a piece of code in a cooperating process in which the process may update shared data (variable, file, database, etc.).

Choices:

- critical section
- critical region
- monitor
- semaphore

A parent process calling _____ system call will be suspended until any imediate child terminates.

Choices:

- wait
- fork
- exit
- exec

Virtual University Overseas Examination System V2.0 (x64)

Operating Systems (CS604)

Question: 1 (Marks: 1) Attempted Questions: 25 Total Questions: 26

_____ commands in Linux is used to copy file

Choices:

ls

cp

mv

mkdir

TIME LEFT
17

Inter process communication (IPC) allows process to communicate and to synchronize their actions without sharing the same _____

Choices:

- (•) Address space
- ( ) Messages
- ( ) Pipes
- ( ) Message Queue

Virtual University Overseas Examination System V2.0 (x64)

Operating Systems (CS604)

Question: 3 **(Marks: 1)** **Attempted Questions: 25** **Total Questions: 26**

Pipe system call is used to _____ a pipe

Choices:

- Destroy
- Create
- Modify
- Clear

TIME LEFT
17

Virtual University Overseas Examination System V2.0 (x64)

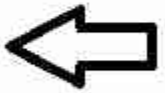
Operating Systems (CS604)

Question: 4 (Marks: 1) Attempted Questions: 25 Total Questions: 26

Which of the following command is used to read data from file named academics?

Choices:

- con < academics
- cat < academics
- socket > academics
- ran > academics



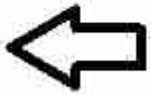
Correct Answer Solved By Hadi
usmanraj20@gmail.com
03228043306

TIME LEFT
17

In a multithreaded model, Which one of the following provides true concurrency?

Choices:

- Many-to-one
- One-to-one



Correct Answer Solved By Hadi
usmanraj20@gmail.com
03228043306

- Many-to-many
- One-to-many

In _____model, there is a kernel thread for every user thread.

Choices:

- (●) One-to-one

- ( ) Many –to-one
- ( ) Many-to-many
- ( ) One-to-many

_____ loads, verifies, and executes programs that have been translated into Java byte code.

Choices:

- (•) Java Virtual Machine
- ( ) VMWare
- ( ) VM Solution
- ( ) VM PC

_____ is the time taken for the dispatcher to stop one process and start another process.

Choices:

- Dispatch latency

- Scheduling
- Context switching
- Preemption

A time interval when a process uses CPU only is called _____.

Choices:

- Scheduling
- I/O burst
- CPU burst
- Dispatch latency

Question: 10 (Marks: 1) Attempted Questions: 25 Total Questions: 26

A time interval when a process uses I/O devices only is called _____.

Choices:

- CPU burst
- I/O burst
- Scheduling
- Context switching

While two processes in execution, only one can be in critical section at a time , this requirement is named as _____.

Choices:

- Race condition
- Mutual exclusion
- Progress
- Bounded waiting

The file descriptor for Standard Output (stdout) is ----------.

Choices:

0

1

Correct Answer Solved By Hadi
usmanraj20@gmail.com
03228043306

2

3

Virtual University Overseas Examination System V2.0 (x64)

Operating Systems (CS604)

Question: 13 (Marks: 1) Attempted Questions: 25 Total Questions: 26

The file descriptor for Standard error (stderr) is ----------.

Choices:

0

1

2

Correct Answer Solved By Hadi
usmanraj20@gmail.com
03228043306

3

TIME LEFT
16

---------- are used by shell commands to pass data from one shell pipeline to another, without creating temporary files.

Choices:

- Pipes
- Message queues
- BSD sockets
- FIFOs

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

Choices:

- First-Come-First-Served (FCFS)
- Round-Robin
- Sorted Job First (SJF)
- Priority

In a Multilevel Queue, the foreground queue has ------- scheduling algorithm and background queue has ---------- scheduling algorithm.

Choices:

- First Come First Serve, Round-Robin
- Round-Robin, First Come First Serve
- Round Robin, Round Robin
- First Come First Serve, First Come First Serve

A process terminates by calling the ----------system call.

Choices:

- (•) wait
- ( ) fork
- ( ) exec
- ( ) exit

The syntax for input redirection is

Choices:

- (●) command < input-file
- ( ) command > input-file
- ( ) command >= input-file
- ( ) command =< input-file

A _____ is a piece of code in a cooperating process in which the process may update shared data (variable, file, database, etc.).

Choices:

- critical section
- critical region
- monitor
- semaphore

A parent process calling _____ system call will be suspended until any imediate child terminates.

Choices:

- wait

- fork
- exit
- exec

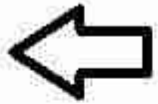
Question: 1 (Marks: 1) Attempted Questions: 22 Total Questions

When process opens its first file explicitly it will get descriptor number _____

Choices:

2 **standard file descriptors**.
Standard input: 0 (keyboard)
Standard output: 1 (display screen)
Standard error: 2 (display screen)
page 52

3



1

4

Question: 2 (Marks: 1) Attempted Questions: 22 Total Questions: 2

Using _____system call, we can create a new process in Unix.

Choices:

- fork
- exec
- wait
- exit

Question: 3 (Marks: 1) Attempted Questions: 22 Total Questions: 26

Which one is not the state of the process?

Choices:

- Waiting
- Running
- Privileged
- New

Operating Systems (CS604)

Question: 4 (Marks: 1) Attempted Questions: 22 Total Questio

To avoid the race condition, the number of processes that may be simultaneously inside their critical section is

Choices:

8

1

16

0

_____ loads, verifies, and executes programs that have been translated into Java byte code.

Choices:

- Java Virtual Machine
- VMWare
- VM Solution
- VM PC

Question: 6 (Marks: 1) Attempted Questions: 22 Total Questions:

A time interval when a process uses CPU only is called _____.

Choices:

- Scheduling
- I/O burst
- CPU burst
- Dispatch latency

Question: 7 (Marks: 1) Attempted Questions: 22 Total Questions: 2

Round robin scheduling usually increases the average waiting time so its performance heavily depends upon _____.

Choices:

- No. of processes
- Time quantum
- Switching
- No. of queues

Question: 8 (Marks: 1) Attempted Questions: 22 Total Questions:

Due to which of the following reason, a parent may terminate the execution of one of its children.

Choices:

- Use more resources than allocated
- Busy other than assigned task
- Start abnormal behavior
- Many jobs in a system

Operating Systems (CS604)

Question: 9 (Marks: 1) Attempted Questions: 22 Total Questio

Which of the following tool is used for inter process communication between related or unrelated processes on the same system or unrelated processes on different system over the network?

Choices:

Pipe

BSD socket

Correct Answer Solved By Hadi
usmanraj20@gmail.com
03228043306

Named pipe (FIFO)

Message Queue

TIME LEFT

Question: 10 (Marks: 1) Attempted Questions: 22 Total Questions: 26

The files which are automatically opened by the kernel for every process to read its input from and send its output and error messages are called_____.

Choices:

- File identifiers
- Standard File descriptors
- File descriptors
- Standard files

Question: 11 (Marks: 1) Attempted Questions: 22 Total Questions: 26

In Unix/Linux client server process communication, server process runs in the background by terminating its command line with ----------.

Choices:

- & (selected)
- @
- %
- $

Question: 12 (Marks: 1) Attempted Questions: 22 Total Questio

Each thread has its own ----------.

Choices:

- Process ID (PID)
- Address Space
- Thread ID (TID)
- Open file descriptors

Operating Systems (CS604)

Question: 13 (Marks: 1) Attempted Questions: 22 Total Questions: 26

Preemptive scheme of Short job first scheduling is known as -------.

Choices:

- First Come First Serve Scheduling
- Shortest-Remaining-Time-First
- Priority Scheduling
- Round-Robin Scheduling

Operating Systems (CS604)

Question: 14 (Marks: 1) Attempted Questions: 22 Total Questions: 2

A process terminates by calling the ----------system call.

Choices:

wait

fork

exec

exit

Question: 15 (Marks: 1) Attempted Questions: 22 Total Questions:

The ----------system call is used to open or create a file.

Choices:

open()

read()

write()

close()

Question: 16 (Marks: 1) Attempted Questions: 22 Total Questions:

The syntax for input redirection is

Choices:

- command < input-file
- command > input-file
- command >= input-file
- command =< input-file

Question: 17 (Marks: 1) Attempted Questions: 22 Total Questions: 26

The syntax for output redirection is

Choices:

- command < output-file
- command >= output-file
- command > output-file
- command =< output-file

Question: 18 (Marks: 1) Attempted Questions: 22 Total Questions:

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

Choices:

- Shortest Job First scheduling
- First Come First Serve
- Priority Scheduling
- Round Robin Scheduling

Question: 19 (Marks: 1) Attempted Questions: 22 Total Questions: 2

Process synchronization is a process of helping processes to _____

Choices:

- update the shared data simultaneously
- update the shared data and resources in an orderly fashion
- not to update the shared data or memory
- access data and memory at first come first serve basis

Operating Systems (CS604)

Question: 20 (Marks: 1) Attempted Questions: 23 Total Questions: 26

A parent process calling _____ system call will be suspended until any imediate child terminates.

Choices:

wait

fork

exit

exec

CS604- OPERATING SYSTEM
(SOLVED Current Subjective)
FROM MIDTERM PAPERS
LECTURE (1-22)

BC190202640@vu.edu.pk

Junaidfazal08@gmail.com

For More Visit: vulmshelp.com

JUNAID MALIK
(0304-1659294)

AL-JUNAID TECH INSTITUTE

1. **Exit system call Wait system call**

ANSWER:

A call to wait() blocks the calling process until one of its child processes exits or a signal is received. After child process terminates, parent continues its execution after wait system call instruction. Child process may terminate due to any of these: It calls exit();

2. **why mknod() call fail give two reason.**

ANSWER:

There are several reasons why a call to the mknod() function could fail:

The process does not have sufficient permissions to create the file. In order to create a file using mknod(), the process must have the appropriate permissions for the directory in which the file is being created.The file name is invalid. There are certain characters and conventions that are not allowed in file names, and if the name specified in the mknod() call contains such characters or does not follow the correct conventions, the call will fail.The file name is too long. There is a maximum length for file names on most systems, and if the name specified in the mknod() call exceeds this maximum length, the call will fail.The file system is full. If the file system does not have sufficient free space to create the file, the mknod() call will fail.The file already exists. If a file with the same name already exists in the specified directory, the mknod() call will fail unless the O_EXCL flag is set, in which case the call will fail with the EEXIST error.

3. **CPU schedulers**
   - **Long term**
   - **Short term**
   - **Middle term**

ANSWER:

In a computer operating system, the CPU scheduler is a component that manages the execution of processes on the CPU. There are three types of CPU schedulers:

Long-term scheduler (also called the job scheduler): This scheduler determines which processes should be admitted to the ready queue for execution. It is

responsible for balancing the trade-off between CPU utilization and response time. The long-term scheduler is responsible for deciding which programs will be executed in the future and for how long. It is a slower-acting scheduler and is usually not invoked very often.

Short-term scheduler (also called the CPU scheduler or dispatcher): This scheduler selects which process should be executed next and allocates the CPU to it. It is a faster-acting scheduler that is invoked very frequently (e.g., every few milliseconds).

Middle-term scheduler: This is a hybrid of the long-term and short-term schedulers. It is invoked less frequently than the short-term scheduler, but more frequently than the long-term scheduler. The middle-term scheduler is responsible for adjusting the priority of processes that have been in the system for a while and have not yet completed. This can help prevent processes that have been in the system for a long time (e.g., "starving" processes) from being starved of CPU resources.

4. ps command And working of Ps-e Ps-u Ps-l

ANSWER:

The ps command is a command-line utility that displays information about the processes running on a system. It can be used to display the processes of a specific user, the processes with a specific terminal, or all processes on the system.

The ps command has many options that can be used to modify its behavior. Some common options include:

-e: This option tells ps to display information about all processes on the system, including processes that have been terminated but have not yet been cleaned up by the system.

-u: This option tells ps to display information about processes owned by a specific user. The user's name or user ID can be specified as an argument to the -u option.

-l: This option tells ps to display information in a "long" format, which includes additional details such as the process's priority, nice value, and process ID.

For example, to display information about all processes on the system in a long format, you could use the command ps -el. To display information about processes owned by the user john in a long format, you could use the command ps -ul john.

5. Similarities between threads and process

ANSWER:

There are several similarities between threads and processes:

- Both threads and processes are independent sequences of execution that can run concurrently on a computer.
- Both threads and processes can have their own set of resources, such as memory, file handles, and network sockets.
- Both threads and processes can be scheduled by the operating system to run on a CPU.
- Both threads and processes can be created and terminated by the operating system or by other threads or processes.
- Both threads and processes can communicate with each other through various means, such as shared memory, message passing, or signaling.
- Both threads and processes can be used to divide a larger task into smaller units of work that can be executed concurrently.
- Both threads and processes can be used to improve the performance and scalability of an application by allowing it to make better use of multiple CPU cores.

6. Differentiate between absolute and relative path

ANSWER:

An absolute path is the full, exact path to a file or directory, starting from the root directory of the file system. It includes all directories and subdirectories that must be traversed to reach the file or directory. An absolute path is not affected by the current working directory of the process or the location of the file that is being executed.

A relative path, on the other hand, is a path to a file or directory relative to the current working directory of the process. It does not include the root directory of the file system, and it specifies only the directories and subdirectories that must be traversed from the current working directory to reach the file or directory.

For example, consider the following file system layout:

```
/
|-home
  |-user1
    |-file1.txt
  |-user2
    |-file2.txt
```

If the current working directory is /home/user1, then the absolute path to file1.txt is /home/user1/file1.txt, and the relative path to file1.txt is simply file1.txt. The absolute path to file2.txt is /home/user2/file2.txt, and the relative path to file2.txt from the current working directory is ../user2/file2.txt.

7. Can user process use the kernal code,justify.

ANSWER:

No, user processes cannot use kernel code directly. The kernel is the core of the operating system and provides services to user processes through system calls and other interfaces. It is protected from direct access by user processes, and user processes are not allowed to execute kernel code or access kernel data structures directly.

This separation between the kernel and user processes is important for several reasons:

- It helps to ensure the stability and security of the system. If user processes were allowed to execute kernel code or access kernel data structures directly, they could potentially crash the system or gain unauthorized access to sensitive information.
- It allows the kernel to be more efficient. By isolating the kernel from user processes, the kernel can be optimized for its specific tasks and does not need to worry about the unpredictable behavior of user processes.
- It allows for the development of multiple user-level libraries and programming languages. Because user processes cannot execute kernel code directly, they must rely on user-level libraries and programming

languages to perform certain tasks. This allows for greater flexibility and innovation in the development of user-level software.

8. Purpose of following commands

- **Fg**
- **Bg**
- **Ps**
- **Kill**
- **Jobs**

ANSWER:

fg: The fg command stands for "foreground." It is used to bring a background or suspended process to the foreground, allowing it to receive input from and display output to the terminal.

bg: The bg command stands for "background." It is used to resume a suspended process in the background, allowing it to continue running without taking up the terminal.

ps: The ps command stands for "process status." It is used to display information about the processes running on a system, including the process ID, command name, and status.

kill: The kill command is used to send a signal to a process, causing it to terminate. It can be used to terminate processes that are stuck or behaving unexpectedly.

jobs: The jobs command is used to display a list of jobs that are running or stopped in the background or foreground of the current shell. It can be used to monitor and manage background processes.

9. Why we use execple() system call after fork() system call

ANSWER:

The exec family of functions (e.g., execve(), execvp(), execl(), etc.) is used to replace the current process image with a new process image. This can be used to execute a different program, or to execute the same program with different arguments or environment variables.

The fork() system call is used to create a new process, which is a copy of the calling process. It is often used in combination with one of the exec functions to create a new process that runs a different program.

After a call to fork(), the new process (child process) and the original process (parent process) both continue to execute from the point at which the fork() call was made. However, the child process typically calls an exec function immediately after the fork() call to replace its process image with a new program. The parent process may continue to execute other code, or it may wait for the child process to complete using the wait() system call.

The combination of fork() and exec() is often used in Unix-like operating systems to create new processes that run different programs. It allows a process to create a child process that runs a specific program, while the parent process can continue to perform other tasks or monitor the child process.

10. Kernel Bottleneck

ANSWEWR:

A kernel bottleneck is a situation where the kernel (the core of the operating system) becomes a limiting factor in the performance of the system. This can occur when the kernel is unable to keep up with the demand for its services, or when it consumes too many resources and leaves insufficient resources for user processes.

There are several potential causes of kernel bottlenecks:

- High system load: If the system is under heavy load, with many processes competing for the same resources, the kernel may become a bottleneck as it tries to manage the resource contention.
- Inefficient kernel design: If the kernel is not well-optimized for the hardware and workload of the system, it may become a bottleneck as it performs poorly compared to other components of the system.
- Resource contention: If the kernel and user processes are competing for the same resources (e.g., CPU time, memory, I/O bandwidth), the kernel may become a bottleneck as it tries to allocate resources fairly among the processes.
- Hardware limitations: If the hardware of the system is not sufficient to support the workload of the kernel and user processes, the kernel may become a bottleneck as it tries to compensate for the limited resources.

To address a kernel bottleneck, it may be necessary to optimize the kernel, upgrade the hardware, reduce the system load, or change the workload of the system.

11.Kernel Level Threads

ANSWER:

Kernel-level threads, also known as kernel threads or system threads, are threads that are managed and scheduled by the operating system kernel. They are implemented directly in the kernel and are managed as part of the kernel's process control block (PCB) data structure.

In contrast, user-level threads are implemented in user space and are managed by a thread library, such as the pthreads library. They are scheduled by the operating system, but they do not appear as distinct entities to the kernel. Instead, they are multiplexed onto a smaller number of kernel-level threads, which are then scheduled by the kernel.

Kernel-level threads have several advantages over user-level threads:

- They can be scheduled more efficiently by the kernel, which has a better view of the system's resources and can make more informed decisions about thread scheduling.
- They can take advantage of multiple CPU cores and processors more effectively, as they can be scheduled independently by the kernel.
- They can make use of special hardware features, such as real-time scheduling or advanced debugging features, that are not available to user-level threads.

However, kernel-level threads also have some disadvantages. They require more overhead to create and manage, as they are part of the kernel's data structures. They also require more system resources, as each kernel-level thread requires its own kernel stack and PCB. This can make it more difficult to create and manage large numbers of threads.

12.Critical solution problem and it's approches are necessary

ANSWER:

A critical section problem is a situation where multiple processes or threads need to access shared resources, and the integrity of the system depends on how

these accesses are coordinated. The critical section problem occurs when two or more processes attempt to access a shared resource simultaneously, which can lead to race conditions or other synchronization issues.

There are several approaches to solving the critical section problem:

- Mutual exclusion: This approach ensures that only one process or thread can access the critical section at a time. It can be implemented using locks, semaphores, or other synchronization mechanisms.
- Priority inversion: This approach allows processes or threads with higher priority to access the critical section first, preventing lower-priority processes from indefinitely waiting for the resource. It can be implemented using priority inheritance or priority ceiling protocols.
- Deadlock prevention: This approach prevents processes or threads from entering into a state where they are waiting indefinitely for a resource that is held by another process or thread. It can be implemented using lock ordering, resource allocation graphs, or other techniques.
- Deadlock detection and recovery: This approach allows processes or threads to detect when they are in a deadlock state and take corrective action to break the deadlock. It can be implemented using a deadlock detection algorithm or a watchdog process.
- Timing constraints: This approach allows processes or threads to access the critical section only within certain time intervals or with certain periods of separation. It can be implemented using time-based semaphores or other timing mechanisms.

13.Linux commands

- Fg
- Kill
- Ps
- Bg
- Cp1

ANSWER:

- fg: The fg command stands for "foreground." It is used to bring a background or suspended process to the foreground, allowing it to receive input from and display output to the terminal.

- kill: The kill command is used to send a signal to a process, causing it to terminate. It can be used to terminate processes that are stuck or behaving unexpectedly.
- ps: The ps command stands for "process status." It is used to display information about the processes running on a system, including the process ID, command name, and status.
- bg: The bg command stands for "background." It is used to resume a suspended process in the background, allowing it to continue running without taking up the terminal.
- cp: The cp command stands for "copy." It is used to copy files and directories from one location to another. It can also be used to create copies of files with different names or in different directories. For example, to copy the file file.txt from the current directory to the /tmp directory, you could use the command cp file.txt /tmp.

MIDTERM EXAMINATION
Fall 2012
CS604- Operating Systems

The given code is as following;

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

Explain if the given code satisfies Mutual Exclusion or not. Justify your answer. (5)

Answer:- (Page 105)

No, it does not satisfy Mutual Exclusion because too prove mutual exclusion, note that P_i enters its critical section only if either flag[j]=false or turn=i. Also, if both processes were executing in their critical sections at the same time, then flag[0]= = flag[1]= = true. These two observations suggest that P_0 and P_1 could not have found both conditions in the while statement true at the same time, since the value of 'turn' can either be 0 or 1. Hence only one process say P_0 must have successfully exited the while statement. Hence mutual exclusion is preserved.

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

3. write down the software based solution for critical section problem (2)
Answer:- (Page 101)

do
{
Entry section
critical section
Exit section
remainder section
} while(1)

5. while working on linux, by which command you can show information about a process (2)
Answer:- (Page 66)
ps gives a snapshot of the current processes. Without options, ps prints information about processes owned by the user.

MIDTERM EXAMINATION
Fall 2012
CS604- Operating Systems

1) Difference between "progress" and " bounded time: in critical section. 2 marks
Answer:- (Page 98)
Progress:-If no process is executing in its critical section and some processes wish to enter their critical sections, then only those processes that are not executing in their remainder section can participate in the decision on which will enter its critical section next, and this selection cannot be postponed indefinitely.

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

3) If process in background then we want its move in foreground then what unix linux command is use to moving. 3 marks
Answer:- (Page 65)
Moving a process into foreground
You can use the fg command to resume the execution of a suspended job in the foreground or move a background job into the foreground. Here is the syntax of the command.
fg [%job_id]
Where, job_id is the job ID (not process ID) of the suspended or background process. If %job_id is omitted, the current job is assumed.

خدا کے سوا کسی سے امید مت رکھو

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

4) How The open source help us to test the algorithm 3 marks

Answer:- (Page 94)

The Open Source software licensing has made it possible for us to test various algorithms by implementing them in the Linux kernel and measuring their true performance.

MIDTERM EXAMINATION
Fall 2012
CS604- Operating Systems

what is difference b/w preemptive and non-preemptive (2)

Answer:- Click here for detatail

Preemptive scheduling allows a process to be interrupted in the middle of its execution, taking the CPU away and allocating it to another process. Non Preemptive scheduling ensures that a process relinquishes control of the CPU only when it finishes with its current CPU burst.

difference b/w progress and bound waiting

Answer:- Rep

MIDTERM EXAMINATION
Fall 2012
CS604- Operating Systems

Question No 1:-

Preemptive Short Job First scheduling algorithm is best algorithm for minimizing the waiting time for the process. How can you calculate the average time in preemptive Short Job First scheduling algorithm?

Answer:- (Page 80)

Example

Process	Burst time	Arrival
P_1	24	0
P_2	3	0
P_3	3	0

Gantt chart: Order P_1, P_2, P_3

Average waiting time: (0+24+27)/3 = 17

بری صحبت سے تنہائی بہتر ہے اور تنہائی سے نیک صحبت بہتر ہے

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

Question No 2:-
Critical section has hardware based or software based solution. You have to write the structure of the software base solution for critical section problem?
Answer:- Rep

Question No 3:-
You may imagine the multi level feedback queue (MLFQ) scheduling algorithm is same as Short Job First scheduling algorithm. Justify your answer either yes or no?

Question No 4:-
Do you feel that reason for mknod system call and mkfifo library call failure are both same? Give reason to support your answer?
Answer:- (Page 57)
In fact, the normal file I/O system calls (close(), read(), write(), unlink(), etc.) all works with FIFOs. Since mkfifo() invokes the mknod() system call, the reasons for its failure are pretty much the same as for the mknod() call given above.

Question No 5:-
You have to explain the working of Semaphore Algorithm?
Answer:- (Page 108)
Hardware solutions to synchronization problems are not easy to generalize to more complex problems. To overcome this difficulty we can use a synchronization tool called a semaphore. A **semaphore** S is an integer variable that, apart from initialization is accessible only through two standard atomic operations: wait and signal. These operations were originally termed P (for wait) and V (for signal). The classical definitions of wait and signal are:

```
wait(S) {
while(S<=0)
;// no op
S--;
}
signal(S) {
S++;
}
```

Modifications to the integer value of the semaphore in the wait and signal operations must be executed indivisibly. That is, when one process is updating the value of a semaphore, other processes cannot simultaneously modify that same semaphore value. In addition, in the case of the wait(S), the testing of the integer value of S (S<=0) and its possible modification (S--) must also be executed without interruption.

We can use semaphores to deal with the n-process critical section problem. The n processes share a semaphore, **mutex** (standing for mutual exclusion) initialized to 1. Each process P_i is organized as follows:

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

```
do
{
wait(mutex);
Critical section
signal(mutex);
Remainder section
} while(1);
```

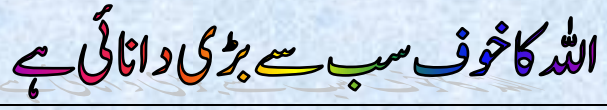
Question No 6:-
Write difference between preemptive and non preemptive scheduling algorithm? Give one name each for preemptive and non preemptive scheduling algorithm?
Answer:- Click here for detatail
Preemptive scheduling allows a process to be interrupted in the middle of its execution, taking the CPU away and allocating it to another process. Non Preemptive scheduling ensures that a process relinquishes control of the CPU only when it finishes with its current CPU burst.
FCFS is a non-preemptive scheduling algorithm.
The SJF algorithm may either be preemptive or non-preemptive. Preemptive SJF scheduling is sometimes called **shortest-remaining-time-first** scheduling.

MIDTERM EXAMINATION
Fall 2012
CS604- Operating Systems

1) Similarities between process and thread execution? 2 marks
Answer:- Click here for detail
Similarities
1) Share cpu.
2)sequential execution
3)create child
4) If one thread is blocked then the next will be start to run like process.

Dissimilarities:
1) Threads are not independent like process.
2) All threads can access every address in the task unlike process.
3)threads are design to assist one another and process might or not might be assisted on one another

2)SJF preemptive average waiting time ?2 marks
Answer:- Rep



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

3) Define any three queues that are used in multifeedback scheduling?

4) Process synchronization of concurrent process or thread for shared data and shared resource? 3 marks

Answer:- (Page 95)

Concurrent processes or threads often need access to shared data and shared resources. If there is no controlled access to shared data, it is often possible to obtain an inconsistent state of this data. Maintaining data consistency requires mechanisms to ensure the orderly execution of cooperating processes, and hence various process synchronization methods are used.

5) Define data race detection is static and dynamic if it is define in detail? marks 5

Answer:- Click here for detail

Data race detection can be broadly classified as either static or dynamic. Static data race detectors [5, 28] analyze the program source code without executing it. They scale to large code bases, providing results in a short amount of time. For instance, RELAY can analyze and report races in the Linux kernel (4 MLOC) in around 5 hours. Static race detectors typically have fewer false negatives (i.e., do not miss real races) than dynamic race detectors [23]. However, static race detectors tend to have many false positives (i.e., produce reports that do not actually correspond to real races). For example, 84% of the races reported by RELAY [28] are not true races.

Dynamic data race detectors typically do not have false positives, but only detect data races in executions they can directly observe; therefore they have many false negatives. They also have high runtime overhead (e.g., 200× in the case of Intel ThreadChecker [11] and up to 8.5× in the case of FastTrack [6]), because they typically need to monitor all memory accesses and synchronization primitives. Detectors that employ sampling [13] decrease runtime overhead at the expense of introducing both false positives and false negatives. High runtime overhead indirectly decreases the coverage of dynamic detectors: they cannot be enabled in production, so they are only used by developers to detect races in executions of the program's test suite.

6) Priority scheduling algorithm is indefinite blocking or starvation define it why? marks 5

Answer:- (Page 83)

A major problem with priority- scheduling algorithms is **indefinite blocking** (or **starvation**). A process that is ready to run but lacking the CPU can be considered blocked-waiting for the CPU. A priority-scheduling algorithm can leave some low priority processes waiting indefinitely for the CPU. Legend has it that when they were phasing out IBM 7094 at MIT in 1973, they found a process stuck in the ready queue since 1967!

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

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

MIDTERM EXAMINATION
Fall 2012
CS604- Operating Systems

Q.2.Diff b/w bounded and progress in respect to critical section?
Answer:- Rep

Q.3.Explain Semaphore Algorithm?
Answer:- Rep

Q.4: For terminating a process which command you will use ?
Answer:- (Page 69)
You can terminate a foreground process by pressing <Ctrl-C>.
You can also terminate a process with the kill command. When executed, this command sends a signal to the process whose process ID is specified in the command line.
Here is the syntax of the command.
kill [-signal] PID

Q.5: Every operating system creates an identified operation or or the process.which includes (data copying, the code ,heap , When UNIX fork () system cal is make at the kernel level as well as at the User level ?
Answer:- (Page 73)
Support for threads may be provided at either user level for *user threads* or by kernel for *kernel threads.*

User threads are supported above kernel and are implemented by a thread library at the user level. The library provides support for thread creation, scheduling, and management with no support from the kernel. Since the kernel is unaware of user-level threads, all thread creation and scheduling are done in the user space without the need for kernel intervention, and therefore are fast to create and manage. If the kernel is single threaded, then any user level thread performing a blocking system call will cause the entire process to block, even if other threads are available to run within the application. User thread libraries include POSIX Pthreads , Solaris 2 UI-threads, and Mach Cthreads.

Kernel threads are supported directly by the operating system. The kernel performs the scheduling, creation, and management in kernel space; the kernel level threads are hence slower to create and manage, compared to user level threads. However since the kernel is managing threads, if a thread performs a blocking system call, the kernel can schedule another thread in the application for execution. Windows NT, Windowss 2000, Solaris, BeOS and Tru64 UNIX support kernel threads.

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

Q:6 Diff b/w FIFO and Pipe ?

Answer:- Click here for detail

A FIFO is similar to a pipe. A FIFO (First In First Out) is a one-way flow of data. FIFOs have a name, so unrelated processes can share the FIFO. FIFO is a named pipe. difference between pipes and FIFOs is the manner in which they are created and opened. Once these tasks have been accomplished, I/O on pipes and FIFOs has exactly the same semantics.

The difference between fifos and pipes is that the former is identified in the file system with a name, while the latter is not. That is, fifos are named pipes. Fifos are identified by an access point which is a file within the file system, whereas pipes are identified by an access point which is simply an allotted inode. Another major difference between fifos and pipes is that fifos last throughout the life-cycle of the system, while pipes last only during the life-cycle of the process in which they were created. To make it more clear, fifos exist beyond the life of the process. Since they are identified by the file system, they remain in the hierarchy until explicitly removed using unlink, but pipes are inherited only by related processes, that is, processes which are descendants of a single process.

MIDTERM EXAMINATION
Fall 2012
CS604- Operating Systems

Q#3: if processor is run in the background we want to move in foreground write the LINIX/UNIX command for this process?(3 M)

Answer:- Rep

Q#5: Semaphore is variable and abstract data type that provide a simple but useful abstraction for controlling access by multiple excess to common resource in parallel environment A semaphore has record how many units are particular source are available, coupled with operations to safely adjust the records as units are required or became free and its necessary waits until a unit of the resources becomes available. You have to identify the drawbacks which are due to using semaphore?(5M)

Answer:- (Page 109)

The main disadvantage of the semaphore discussed in the previous section is that it requires **busy waiting**. While a process is in its critical section, any other process that tries to enter its critical section must loop continuously in the entry code. This continual looping is clearly a problem in a real multiprogramming system, where a single CPU is shared among many processes. Busy waiting wastes CPU cycles that some other process may be able to use productively. This type of semaphore is also called a **spinlock** (because the process spins while waiting for the lock). Spinlocks are useful in multiprocessor systems. The advantage of a spinlock is that no context switch is required when a process must wait on a lock, and a context switch may take considerable time. This is, spinlocks are useful when they are expected to be held for short times. The definition of semaphore should be modified to eliminate busy waiting.

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

Q#6: Out of (SRTF) and round robin scheduling algorithm which one is best suited to be used in Time-sharing-system where response time is an important performance criteria? Give reasons in your answer?(5M)

Answer:- (Page 87)

Preemptive SJF scheduling is sometimes called **shortest-remaining-time-first** scheduling.

Typically, round robin has a higher average turnaround than SJF, but better response. In timesharing systems, shorter response time for a process is more important than shorter turnaround time for the process. Thus, round-robin scheduler matches the requirements of time-sharing systems better than the SJF algorithm. SJF scheduler is better suited for batch systems, in which minimizing the turnaround time is the main criterion.

MIDTERM EXAMINATION
Spring 2012
CS604- Operating Systems

1disadvantages of One to one threads

Answer:- (Page 71)

The main disadvantage of this model is the overhead of creating a kernel thread per user thread.

2. Assume

i)that each process executes at a nonzero speed

ii)No assumption can be made regarding the relative speeds of the N processes.

These two assumptions can be used for ? (2 marks)

Answer:- (Page 98)

These are used for Solution to the Critical Section Problem.

3 a job suspended or running in back ground to move it to the fore ground which command is used?

Answer:- Rep

5 when a process makes its copy which things are that must be copy to new new process(5)

Answer:- (Page 37)

When the fork system call is executed, a new process is created. The original process is called the parent process whereas the process is called the child process. The new process consists of a copy of the address space of the parent. This mechanism allows the parent process to communicate easily with the child process. On success, both processes continue execution at the instruction after the fork call, with one difference, the return code for the fork system call is zero for the child process, while the process identifier of the child is returned to the parent process. On failure, a -1 will be returned in the parent's context, no child process will be created, and an error number will be set appropriately.

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

MIDTERM EXAMINATION
Spring 2012
CS604- Operating Systems

Q.21: differentiate Entry Section and Reminder Section. (2 Marks)

Answer:- (Page 97)

When a process executes code that manipulates shared data (or resource), we say that the process is in its critical section (for that shared data). The execution of critical sections must be mutually exclusive: at any time, only one process is allowed to execute in its critical section (even with multiple processors). So each process must first request permission to enter its critical section. The section of code implementing this request is called the **entry section**. The remaining code is the **remainder section**.

Q.23: UNIX System V scheduling algorithm (3 marks)

Answer:- (Page 90)

UNIX System V scheduling algorithm is essentially a multilevel feedback priority queues algorithm with round robin within each queue, the quantum being equal to1 second. The priorities are divided into two groups/bands:

- Kernel Group
- User Group

اپنی مرضی اور اللہ کی مرضی میں فرق کا نام غم ہے

اس سے پہلے کہ تمہیں شہوت فتنے میں ڈالے نکاح کر لو

وہ لوگ مبارک ہیں جو الفاظ سے نصیحت نہیں کرتے بلکہ عمل سے کرتے ہیں

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

Question No : 21 of 26 Marks: 2 (Budgeted Time 4 Min)

Preemptive Shortest Job First scheduling algorithm is best algorithm for minimizing the waiting time for the processes. How can you calculate the average waiting time in preemptive Shortest Job First scheduling algorithm?

Answer (Please click here to Add Answer)

Question No : 22 of 26 Marks: 2 (Budgeted Time 4 Min)

Critical section has hardware based or software based solutions. You have to write the structure of the software based solution for the critical section problem.

Answer (Please click here to Add Answer)

Question No : 23 of 26 Marks: 3 (Budgeted Time 6 Min)

Consider this algorithm for mutual exclusion.

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

```
Process 2 {
while true {
      while (turn == 2)
            ; /* do nothing */
```

Answer (Please click here to Add Answer)

Question No : 23 of 26 Marks: 3 (Budgeted Time 6 Min)

```
        ; /* do nothing */
    critical section
    turn = 2;
    other non-critical code
  }
}
```

```
        ; /* do nothing */
    critical section
    turn = 1;
    other non-critical code
  }
}
```

Answer (Please click here to Add Answer)

Question No : 23 of 26 Marks: 3 (Budgeted Time 6 Min)

```
        other non-critical code
    }
}
```

```
        other non-critical code
    }
}
```

Does this solution have any problem? If yes then justify your answer.

Answer (Please click here to Add Answer)

Question No : 24 of 26 Marks: 3 (Budgeted Time 6 Min)

Process can run either as background process or as foreground process. If the process is running in background and you wanted to move a background process into foreground in a UNIX/LINUX shell. Which command will you use for it?

Answer (Please click here to Add Answer)

Question No : 25 of 26 Marks: 5 (Budgeted Time 10 Min)

Semaphore is a variable or abstract data type that provides a simple but useful abstraction for controlling access by multiple processes to a common resource in a parallel environment. We can use semaphores to deal with the n-process critical section problem. You have to explain the working of Semaphore Algorithm.

Answer (Please click here to Add Answer)

Question No : 26 of 26 Marks: 5 (Budgeted Time 10 Min)

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

Answer (Please click here to Add Answer)

Let us consider a situation in which each process gets a small unit of CPU time, called *time slice* or *quantum*, which is usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue. Which scheduling algorithm is best suited for above situation?

Answer:

Process can run either as background process or as foreground process. If there is a process which is in the suspended state and working in background, you wanted to display the status of suspended and background process in a UNIX/LINUX shell. Which command will you use?

Answer:

Moving a process into foreground: fg [%job_id]
where, job_id is the job ID (not process ID) of the suspended or background process. If %job_id is omitted, the current job is assumed.
Moving a process into background: bg [%job_id]
If %job_id is omitted the current job is assumed.

Open sources software's are continuously upgraded by different developers. How Open Source Software has made it possible for us to test various algorithms instead of pirated software's?

Answer:

What is the function of following command in Linux/ Unix:

a) command ls> abc.txt

b) cat < fileName.txt

Answer:

Answer: (a) command ls > abc.txt :
Without an argument of ls commands it assumes your current working directory.So if we run the ls command right after login, it will displays names of files and directories in our home directory.
(b) cat < fileName. txt:

Operating Systems (CS604)

Question: 25 (Marks: 5) **Attempted Questions: 25** **Total Questions: 26**

There are five processes given below with their run time units. Assume that all processes arrive in numerical order at time 0 then answer the questions given below:

Process ID	CPU Requirements
P_1	5
P_2	4
P_3	2
P_4	1
P_5	8

Show the scheduling order for these processes under first-come-first-served, shortest-job first, and round-robin scheduling (quantum = 2).

Answer:

Question No : 21 of 26 Marks: 2 (Budgeted Time 4 Min)

Preemptive Shortest Job First scheduling algorithm is best algorithm for minimizing the waiting time for the processes. How can you calculate the average waiting time in preemptive Shortest Job First scheduling algorithm?

Answer (Please click here to Add Answer)

Question No : 22 of 26 Marks: 2 (Budgeted Time 4 Min)

Critical section has hardware based or software based solutions. You have to write the structure of the software based solution for the critical section problem.

Answer (Please click here to Add Answer)

Question No : 23 of 26 Marks: 3 (Budgeted Time 6 Min)

Consider this algorithm for mutual exclusion.

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

```
Process 2 {
while true {
      while (turn == 2)
            ; /* do nothing */
```

Answer (Please click here to Add Answer)

Question No : 23 of 26 Marks: 3 (Budgeted Time 6 Min)

```
        ; /* do nothing */
    critical section
    turn = 2;
    other non-critical code
  }
}
```

```
        ; /* do nothing */
    critical section
    turn = 1;
    other non-critical code
  }
}
```

Answer (Please click here to Add Answer)

Question No : 23 of 26 Marks: 3 (Budgeted Time 6 Min)

```
        other non-critical code
    }
}
```

```
        other non-critical code
    }
}
```

Does this solution have any problem? If yes then justify your answer.

Answer (Please click here to Add Answer)

Question No : 24 of 26 Marks: 3 (Budgeted Time 6 Min)

Process can run either as background process or as foreground process. If the process is running in background and you wanted to move a background process into foreground in a UNIX/LINUX shell. Which command will you use for it?

Answer (Please click here to Add Answer)

Question No : 25 of 26 Marks: 5 (Budgeted Time 10 Min)

Semaphore is a variable or abstract data type that provides a simple but useful abstraction for controlling access by multiple processes to a common resource in a parallel environment. We can use semaphores to deal with the n-process critical section problem. You have to explain the working of Semaphore Algorithm.

Answer (Please click here to Add Answer)

Question No : 26 of 26 Marks: 5 (Budgeted Time 10 Min)

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

Answer (Please click here to Add Answer)

Let us consider a situation in which each process gets a small unit of CPU time, called *time slice* or *quantum*, which is usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue. Which scheduling algorithm is best suited for above situation?

Answer:

Process can run either as background process or as foreground process. If there is a process which is in the suspended state and working in background, you wanted to display the status of suspended and background process in a UNIX/LINUX shell. Which command will you use?

Answer:

Moving a process into foreground: fg [%job_id]
where, job_id is the job ID (not process ID) of the suspended or background process. If %job_id is omitted, the current job is assumed.
Moving a process into background: bg [%job_id]
If %job_id is omitted the current job is assumed.

Open sources software's are continuously upgraded by different developers. How Open Source Software has made it possible for us to test various algorithms instead of pirated software's?

Answer:

What is the function of following command in Linux/ Unix:

a) command ls> abc.txt

b) cat < fileName.txt

Answer:

Answer: (a) command ls > abc.txt :
Without an argument of ls commands it assumes your current working directory.So if we run the ls command right after login, it will displays names of files and directories in our home directory.
(b) cat < fileName. txt:

Operating Systems (CS604)

Question: 25 (Marks: 5) **Attempted Questions: 25** **Total Questions: 26**

There are five processes given below with their run time units. Assume that all processes arrive in numerical order at time 0 then answer the questions given below:

Process ID	CPU Requirements
P_1	5
P_2	4
P_3	2
P_4	1
P_5	8

Show the scheduling order for these processes under first-come-first-served, shortest-job first, and round-robin scheduling (quantum = 2).

Answer:

Question No : 21 of 26 Marks: 2 (Budgeted Time 4 Min)

A number of queues can be used in the Multi Level Queue and Multilevel Feedback Queue. You need to identify any three queues that you can use in a Multilevel Feedback Queue.

Answer (Please click here to Add Answer)

Question No : 22 of 26 Marks: 2 (Budgeted Time 4 Min)

Which term is best suited for a process that is a background system process and that provide you some service.

Answer (Please click here to Add Answer)

Question No : 23 of 26 Marks: 3 (Budgeted Time 6 Min)

Sort out the given below Scheduling algorithms with respect to there best utilization of CPU (arrange in order low to high).

0. First In First Out
1. Priority based scheduling
2. Multilevel Queue scheduling

Answer (Please click here to Add Answer)

Question No : 24 of 26 Marks: 3 (Budgeted Time 6 Min)

Which term is best suited for the situation where several processes access and manipulate shared data concurrently, the final value of the data depends on which process finishes last?

Answer (Please click here to Add Answer)

Question No : 25 of 26 Marks: 5 (Budgeted Time 10 Min)

Semaphore is a variable or abstract data type that provides a simple but useful abstraction for controlling access by multiple processes to a common resource in a parallel environment. We can use semaphores to deal with the n-process critical section problem. You have to explain the working of Semaphore Algorithm.

Answer (Please click here to Add Answer)

Question No : 26 of 26 Marks: 5 (Budgeted Time 10 Min)

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

Answer (Please click here to Add Answer)

Question No : 21 of 26 Marks: 2 (Budgeted Time 4 Min)

A number of queues can be used in the Multi Level Queue and Multilevel Feedback Queue. You need to identify any three queues that you can use in a Multilevel Feedback Queue.

Answer (Please click here to Add Answer)

Question No : 22 of 26 Marks: 2 (Budgeted Time 4 Min)

Which term is best suited for a process that is a background system process and that provide you some service.

Answer (Please click here to Add Answer)

Question No : 23 of 26 Marks: 3 (Budgeted Time 6 Min)

Sort out the given below Scheduling algorithms with respect to there best utilization of CPU (arrange in order low to high).

0. First In First Out
1. Priority based scheduling
2. Multilevel Queue scheduling

Answer (Please click here to Add Answer)

Question No : 24 of 26 Marks: 3 (Budgeted Time 6 Min)

Which term is best suited for the situation where several processes access and manipulate shared data concurrently, the final value of the data depends on which process finishes last?

Answer (Please click here to Add Answer)

Question No : 25 of 26 Marks: 5 (Budgeted Time 10 Min)

Semaphore is a variable or abstract data type that provides a simple but useful abstraction for controlling access by multiple processes to a common resource in a parallel environment. We can use semaphores to deal with the n-process critical section problem. You have to explain the working of Semaphore Algorithm.

Answer (Please click here to Add Answer)

Question No : 26 of 26 Marks: 5 (Budgeted Time 10 Min)

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

Answer (Please click here to Add Answer)

Question: 21 (Marks: 2)

Write the code to create a thread, take thread ID in thread named as newThread and run Mythread function.

Question: 22 (Marks: 2) Atten

A CPU scheduler has different evaluation matrices and average waiting time is one of them. State how **Waiting time** relates to the evaluation of matrices with respect to CPU Scheduling?

.....

Operating Systems (CS604)

Question: 23 (Marks: 3)

The mknod() system call may fail due to different reasons. You are required to identify at least three failure scenarios in which mknod() system call fails to execute.

.....

Answer:

Some of the scenarios are :
When the file with given name is exists

Operating Systems (CS604)

Question: 24 (Marks: 3)

What will happen if exit () is called instead of pthread_exit () in multithreaded system?

Answer:

Operating Systems (CS604)

Question: 25 (Marks: 5)

While processing a job it is possible that each process is executed and managed independently but sometimes it gives overhead to OS. Highlight the issues that can come across while managing a process independently and how the threads can help to overcome these issues? Write at least three aspects used by thread to overcome these issues.

Operating Systems (CS604)

Question: 26 (Marks: 5)

Consider performance of FCFS algorithm for three computer-bound processes. If process P1 takes 24 seconds, P2 takes 3 seconds and P3 takes 3 seconds and processes arrive in the given order P1, P2, P3. You need to calculate the following:

a) **Turnaround Time per process.**
b) **Average Turnaround time of processes**