****OPERATING SYSTEM****

**ASSIGNMENT -1**

1. **Define System calls?List various system calls and their functions.**

A ****system call**** is a mechanism that provides the interface between a process and the operating system. It is a programmatic method in which a computer program requests a service from the kernel of the OS.

System call offers the services of the operating system to the user programs via API (Application Programming Interface). System calls are the only entry points for the kernel system.

## **Types of System calls**

Here are the five types of system calls used in OS:

* Process Control
* File Management
* Device Management
* Information Maintenance
* Communications

### **Process Control**

This system calls perform the task of process creation, process termination, etc.

Functions:

* End and Abort
* Load and Execute
* Create Process and Terminate Process
* Wait and Signed Event
* Allocate and free memory

### **File Management**

File management system calls handle file manipulation jobs like creating a file, reading, and writing, etc.

Functions:

* Create a file
* Delete file
* Open and close file
* Read, write, and reposition
* Get and set file attributes

### **Device Management**

Device management does the job of device manipulation like reading from device buffers, writing into device buffers, etc.

Functions

* Request and release device
* Logically attach/ detach devices
* Get and Set device attributes

### **Information Maintenance**

It handles information and its transfer between the OS and the user program.

Functions:

* Get or set time and date
* Get process and device attributes

### **Communication:**

These types of system calls are specially used for interprocess communications.

Functions:

* Create, delete communications connections
* Send, receive message
* Help OS to transfer status information
* Attach or detach remote devices

1. **Define System programs.List all of them.**

System programs provide an environment where programs can be developed and executed. In the simplest sense, system programs also provide a bridge between the user interface and system calls. In reality, they are much more complex. For example, a compiler is a complex system program.

## **Types of System Programs**

System programs can be divided into seven parts. These are given as follows:

### **Status Information**

The status information system programs provide required data on the current or past status of the system. This may include the system date, system time, available memory in system, disk space, logged in users etc.

### **Communications**

These system programs are needed for system communications such as web browsers. Web browsers allow systems to communicate and access information from the network as required.

### **File Manipulation**

These system programs are used to manipulate system files. This can be done using various commands like create, delete, copy, rename, print etc. These commands can create files, delete files, copy the contents of one file into another, rename files, print them etc.

### **Program Loading and Execution**

The system programs that deal with program loading and execution make sure that programs can be loaded into memory and executed correctly. Loaders and Linkers are a prime example of this type of system programs.

### **File Modification**

System programs that are used for file modification basically change the data in the file or modify it in some other way. Text editors are a big example of file modification system programs.

### **Application Programs**

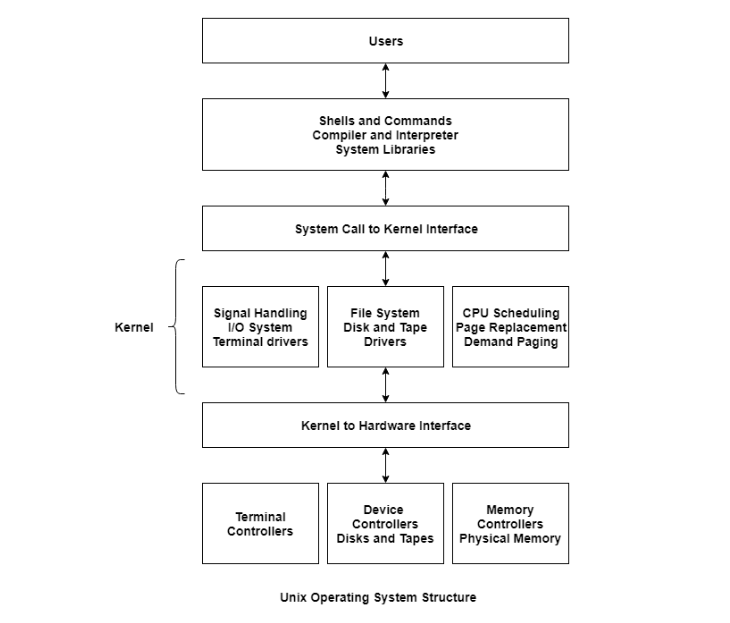
Application programs can perform a wide range of services as per the needs of the users. These include programs for database systems, word processors, plotting tools, spreadsheets, games, scientific applications etc.

### **Programming Language Support**

These system programs provide additional support features for different programming languages. Some examples of these are compilers, debuggers etc. These compile a program and make sure it is error free respectively.

1. **Draw the structure of following OS: i) UNIX ii) LINUX. Why is Kernel needed in OS?**

**i)**



**ii)**

The **operating system kernel**represents the highest level of privilege in a modern general purpose computer. The**kernel** arbitrates access to protected hardware and controls how limited resources such as running time on the CPU and physical memory pages are used by processes on the **system**.

1. **What is a Thread? List its advantages and disadvantages. What are the benefits of multithreaded programming? Explain thread structure in Solaries OS.**

A thread is a path of execution within a process. A process can contain multiple threads.

**Advantages:**

1. Sometimes it is very useful to make a program appear to do two things at once. The classic example is to perform a real-time word count on a document while still editing the text. One thread can manage the user’s input and perform editing. The other, which can see the same document content, can continuously update a word count variable. The first thread (or even a third one) can use this shared variable to keep the user informed. Another example is a multithreaded database server where an apparent single process serves multiple clients, improving the overall data throughput by servicing some requests while blocking others, waiting for disk activity. For a database server, this apparent multitasking is quite hard to do efficiently in different processes, because the  requirements for locking and data consistency cause the different processes to be very tightly coupled. This can be done much more easily with multiple threads than with multiple processes.
2. The performance of an application that mixes input, calculation, and output may be improved by running these as three separate threads. While the input or output thread is waiting for a connection, one of the other threads can continue with calculations. A server application pro-cessing multiple network connects may also be a natural fit for a multithreaded program.
3. Now that multi-cored CPUs are common even in desktop and laptop machines, using multiple threads inside a process can, if the application is suitable, enable a single process to better utilize the hardware resources available.
4. In general, switching between threads requires the operating system to do much less work than switching between processes. Thus, multiple threads are much less demanding on resources than multiple processes, and it is more practical to run programs that logically require many threads of execution on single-processor systems. That said, the design difficulties of writing a multithreaded program are significant and should not be taken lightly.

**Disadvantages:**

1. Writing multithreaded programs requires very careful design. The potential for introducing subtle timing faults, or faults caused by the unintentional sharing of variables in a multithreaded program is considerable. Alan Cox (the well respected Linux guru) has commented that threads are also known as “how to shoot yourself in both feet at once.”
2. Debugging a multithreaded program is much, much harder than debugging a single-threaded one, because the interactions between the threads are very hard to control.
3. A program that splits a large calculation into two and runs the two parts as different threads will not necessarily run more quickly on a single processor machine, unless the calculation truly allows multiple parts to be calculated simultaneously and the machine it is executing on has multiple processor cores to support true multiprocessing.

The benefits of multi threaded programming can be broken down into four major categories:

* ****Resource Sharing****

All the threads of a process share its resources such as memory, data, files etc. A single application can have different threads within the same address space using resource sharing.

* ****Responsiveness****

Program responsiveness allows a program to run even if part of it is blocked using multithreading. This can also be done if the process is performing a lengthy operation. For example - A web browser with multithreading can use one thread for user contact and another for image loading at the same time.

* ****Utilization of Multiprocessor Architecture****

In a multiprocessor architecture, each thread can run on a different processor in parallel using multithreading. This increases concurrency of the system. This is in direct contrast to a single processor system, where only one process or thread can run on a processor at a time.

* ****Economy****

It is more economical to use threads as they share the process resources. Comparatively, it is more expensive and time-consuming to create processes as they require more memory and resources. The overhead for process creation and management is much higher than thread creation and management.

1. **Explain different File Management commands in UNIX.**

****#1) touch****: Create a new file or update its timestamp.

* ****Syntax****: touch [OPTION]…[FILE]
* ****Example****: Create empty files called ‘file1’ and ‘file2’
  + $ touch file1 file2

****#2) cat****: Concatenate files and print to stdout.

* ****Syntax****: cat [OPTION]…[FILE]
* ****Example****: Create file1 with entered cotent
  + $ cat > file1
  + Hello
  + ^D

****#3) cp****: Copy files

* ****Syntax****: cp [OPTION]source destination
* ****Example****: Copies the contents from file1 to file2 and contents of file1 is retained
  + $ cp file1 file2

****#4) mv****: Move files or rename files

* ****Syntax****: mv [OPTION]source destination
* ****Example****: Create empty files called ‘file1’ and ‘file2’
  + $ mv file1 file2

****#5) rm****: Remove files and directories

* ****Syntax****: rm [OPTION]…[FILE]
* ****Example****: Delete file1
  + $ rm file1

****#6) mkdir****: Make directory

* ****Syntax****: mkdir [OPTION] directory
* ****Example****: Create directory called dir1
  + $ mkdir dir1

****#7) rmdir****: Remove a directory

* ****Syntax****: rmdir [OPTION] directory
* ****Example****: Create empty files called ‘file1’ and ‘file2’
  + $ rmdir dir1

****#8) cd****: Change directory

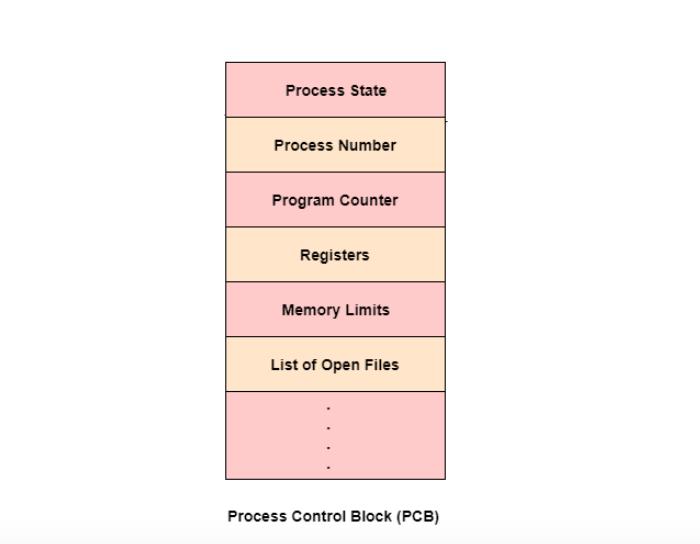
* ****Syntax****: cd [OPTION] directory
* ****Example****: Change working directory to dir1
  + $ cd dir1

****#9) pwd****: Print the present working directory

* ****Syntax****: pwd [OPTION]
* ****Example****: Print ‘dir1’ if a current working directory is dir1
  + $ pwd

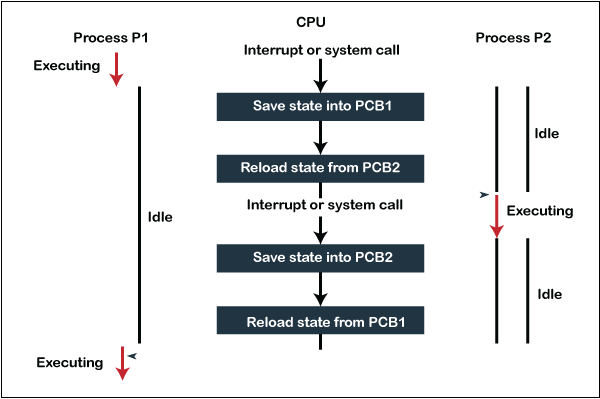
1. **Explain the structure of PCB.Also,explain the role of PCB in context switching.Also,Draw the flowchart for this event.**

## **Structure of the Process Control Block**



A **context switching** is the procedure to restore and store the state of **context** of CPU in process control block(**PCB**) so that the process execution can be resumed from the same point at a later time. This **context switching** technique enables multiple processes to share a single CPU.

**FLOWCHART**



1. **What is the need of CPU Scheduling algorithms?How do you measure the complexity of these algorithms.Give one example of Operating System where these algorithms is used.**

****CPU Scheduling**** is a process of determining which process will own CPU for execution while another process is on hold. The main task of CPU scheduling is to make sure that whenever the CPU remains idle, the OS at least select one of the processes available in the ready queue for execution. The selection process will be carried out by the CPU scheduler. It selects one of the processes in memory that are ready for execution.

Different [CPU scheduling algorithms](https://www.geeksforgeeks.org/cpu-scheduling-in-operating-systems/) have different properties and the choice of a particular algorithm depends on the various factors. Many criteria have been suggested for comparing CPU scheduling algorithms.

The criteria include the following:

1. ****CPU utilisation** –**   
   The main objective of any CPU scheduling algorithm is to keep the CPU as busy as possible. Theoretically, CPU utilisation can range from 0 to 100 but in a real-time system, it varies from 40 to 90 percent depending on the load upon the system.
2. ****Throughput** –**   
   A measure of the work done by CPU is the number of processes being executed and completed per unit time. This is called throughput. The throughput may vary depending upon the length or duration of processes.
3. ****Turnaround time** –**   
   For a particular process, an important criteria is how long it takes to execute that process. The time elapsed from the time of submission of a process to the time of completion is known as the turnaround time. Turn-around time is the sum of times spent waiting to get into memory, waiting in ready queue, executing in CPU, and waiting for I/O.
4. ****Waiting time** –**   
   A scheduling algorithm does not affect the time required to complete the process once it starts execution. It only affects the waiting time of a process i.e. time spent by a process waiting in the ready queue.
5. ****Response time** –**   
   In an interactive system, turn-around time is not the best criteria. A process may produce some output fairly early and continue computing new results while previous results are being output to the user. Thus another criteria is the time taken from submission of the process of request until the first response is produced. This measure is called response time.
6. **An operating system uses i) pre-emptive Shortest Job first (SJF) ii) Round-robin(with time quantum 5 process scheduling algorithm. Consider the arrival times and execution times for the following processes**:

Process Execution time Arrival time

P1 20 0

P2 25 15

P3 10 30

P4 15 45

**What is the total waiting time for process P2?**

1. **Consider three CPU-intensive processes, which require 10, 20 and 30 time units and arrive at times 0, 2 and 6, respectively. Assume context switching time is 2 time units .Now calculate Average Turn Around Time,Average Waiting Time using i) Pre-emptive SJF ii)Round –robin(time quantum 2 time units)**
2. **Using Semaphore,provide solution to the i)producer-consumer problem ii) Read –write problem when severa processes are reading and writing on shared file and Dining-Philosophers problem**
3. **Write the implementation details of spin-lock semaphore.**

A *spinlock* is one possible implementation of a lock, namely one that is implemented by busy waiting ("spinning"). A semaphore is a generalization of a lock (or, the other way around, a lock is a special case of a semaphore). Usually, *but not necessarily*, spinlocks are only valid within one process whereas semaphores can be used to synchronize between different processes, too.

A lock works for mutual exclusion, that is ****one**** thread at a time can acquire the lock and proceed with a "critical section" of code. Usually, this means code that modifies some data shared by several threads.  
A *semaphore* has a counter and will allow itself being acquired by ****one or several**** threads, depending on what value you post to it, and (in some implementations) depending on what its maximum allowable value is.

1. **Write short note on multiprocessor scheduling.**

In multiple-processor scheduling **multiple CPU’s** are available and hence **Load Sharing** becomes possible. However multiple processor scheduling is more **complex** as compared to single processor scheduling. In multiple processor scheduling there are cases when the processors are identical i.e. HOMOGENEOUS, in terms of their functionality, we can use any processor available to run any process in the queue.

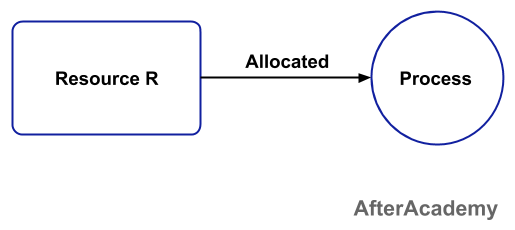
1. **What is a deadlock?What are the necessary conditions for deadlock occurrence?Discuss all deadlock handling strategies.**

Deadlock is a situation where two or more processes are waiting for each other. For example, let us assume, we have two processes P1 and P2. Now, process P1 is holding the resource R1 and is waiting for the resource R2. At the same time, the process P2 is having the resource R2 and is waiting for the resource R1. So, the process P1 is waiting for process P2 to release its resource and at the same time, the process P2 is waiting for process P1 to release its resource. And no one is releasing any resource. So, both are waiting for each other to release the resource. This leads to infinite waiting and no work is done here. This is called Deadlock.

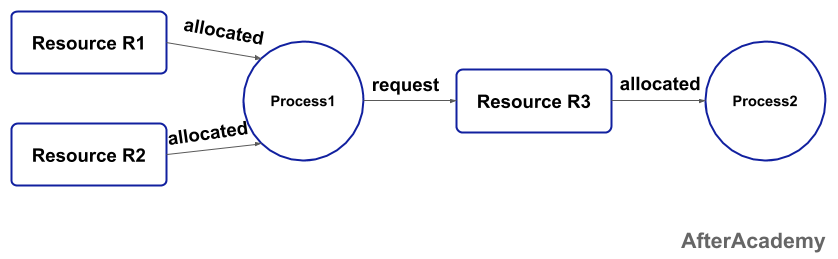
#### **Necessary Conditions of Deadlock**

There are four different conditions that result in Deadlock. These four conditions are also known as Coffman conditions and these conditions are not mutually exclusive. Let's look at them one by one.

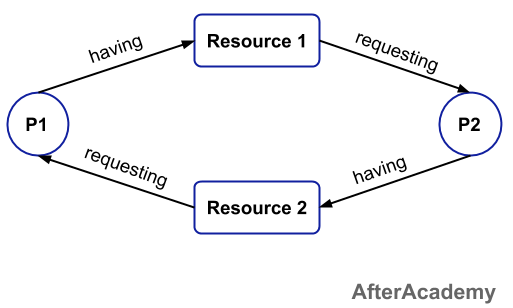
* **Mutual Exclusion:** A resource can be held by only one process at a time. In other words, if a process P1 is using some resource R at a particular instant of time, then some other process P2 can't hold or use the same resource R at that particular instant of time. The process P2 can make a request for that resource R but it can't use that resource simultaneously with process P1.



* **Hold and Wait:** A process can hold a number of resources at a time and at the same time, it can request for other resources that are being held by some other process. For example, a process P1 can hold two resources R1 and R2 and at the same time, it can request some resource R3 that is currently held by process P2.



* **No preemption:** A resource can't be preempted from the process by another process, forcefully. For example, if a process P1 is using some resource R, then some other process P2 can't forcefully take that resource. If it is so, then what's the need for various scheduling algorithm. The process P2 can request for the resource R and can wait for that resource to be freed by the process P1.
* **Circular Wait:** Circular wait is a condition when the first process is waiting for the resource held by the second process, the second process is waiting for the resource held by the third process, and so on. At last, the last process is waiting for the resource held by the first process. So, every process is waiting for each other to release the resource and no one is releasing their own resource. Everyone is waiting here for getting the resource. This is called a circular wait.



Deadlock will happen if all the above four conditions happen simultaneously.

1. **A system is having 3 user processes P1, P2 and P3 where P1 requires 2 units of resource R, P2 requires 3 units of resource R, P3 requires 4 units of resource R. Find the minimum number of units of R that ensures no deadlock.**
2. **1.5,1.6, 1.7, 1.10, 1.11, 4.2,4.3, 4.6,4.9**
3. **Two case studies**