ASSIGNMENT 1

NAME: Ananya Prasad

REGNO: 20BCE10093

FACULTY: Dr Abha Trivedi

SLOT: B21+B22+B23

1. Differentiate between Symmetric and Asymmetric Multiprocessing with example.

Multiprocessing is the use of two or more central processing units within a single computer system.

Asymmetric Multiprocessing:

- In asymmetric multiprocessing, the processors are not treated equally.
- Tasks of the operating system are done by master processor.
- No Communication between Processors as they are controlled by the master processor.
- In asymmetric multiprocessing, process is master-slave.
- Asymmetric multiprocessing systems are cheaper.
- Asymmetric multiprocessing systems are easier to design

For example, AMP can be used in assigning specific tasks to CPU based on priority and importance of task completion.

Symmetric Multiprocessing

- In symmetric multiprocessing, all the processors are treated equally.
- Tasks of the operating system are done individual processor
- All processors communicate with another processor by a shared memory.
- In symmetric multiprocessing, the process is taken from the ready queue.
- Symmetric multiprocessing systems are costlier.
- Symmetric multiprocessing systems are complex to design

For example, SMP applies multiple processors to that one problem, known as parallel programming.

2. What do you understand about Multiprogramming and Multitasking systems?

Multi programming:

One of the most important aspects of an Operating System is to multi program.

In a computer system, there are multiple processes waiting to be executed, i.e., they are waiting when the CPU will be allocated to them and they begin their execution. These processes are also known as jobs. Now the main memory is too small to accommodate all of these processes or jobs into it. Thus, these processes are initially kept in an area called job pool. This job pool consists of all those processes awaiting allocation of main memory and CPU.

As soon as one job goes for an I/O task, the Operating System interrupts that job, chooses another job from the job pool (waiting queue), gives CPU to this new job and starts its execution. The previous job keeps doing its I/O operation while this new job does CPU bound tasks. Now say the second job also goes for an I/O task, the CPU chooses a third job and starts

executing it. As soon as a job completes its I/O operation and comes back for CPU tasks, the CPU is allocated to it.

In this way, no CPU time is wasted by the system waiting for the I/O task to be completed.

Therefore, the ultimate goal of multi programming is to keep the CPU busy as long as there are processes ready to execute. This way, multiple programs can be executed on a single processor by executing a part of a program at one time, a part of another program after this, then a part of another program and so on, hence executing multiple programs. Hence, the CPU never remains idle.

Multitasking systems:

Multi-tasking refers to execution of multiple tasks (say processes, programs, threads etc.) at a time. Multitasking is a logical extension of multi programming. The major way in which multitasking differs from multi programming is that multi programming works solely on the concept of context switching whereas multitasking is based on time sharing alongside the concept of context switching.

In a time-sharing system, each process is assigned some specific quantum of time for which a process is meant to execute. Say there are 4 processes P1, P2, P3, P4 ready to execute. So, each of them is assigned some time quantum for which they will execute e.g., time quantum of 5 nanoseconds (5 ns). As one process begins execution (say P2), it executes for that quantum of time (5 ns). After 5 ns the CPU starts the execution of the other process (say P3) for the specified quantum of time.

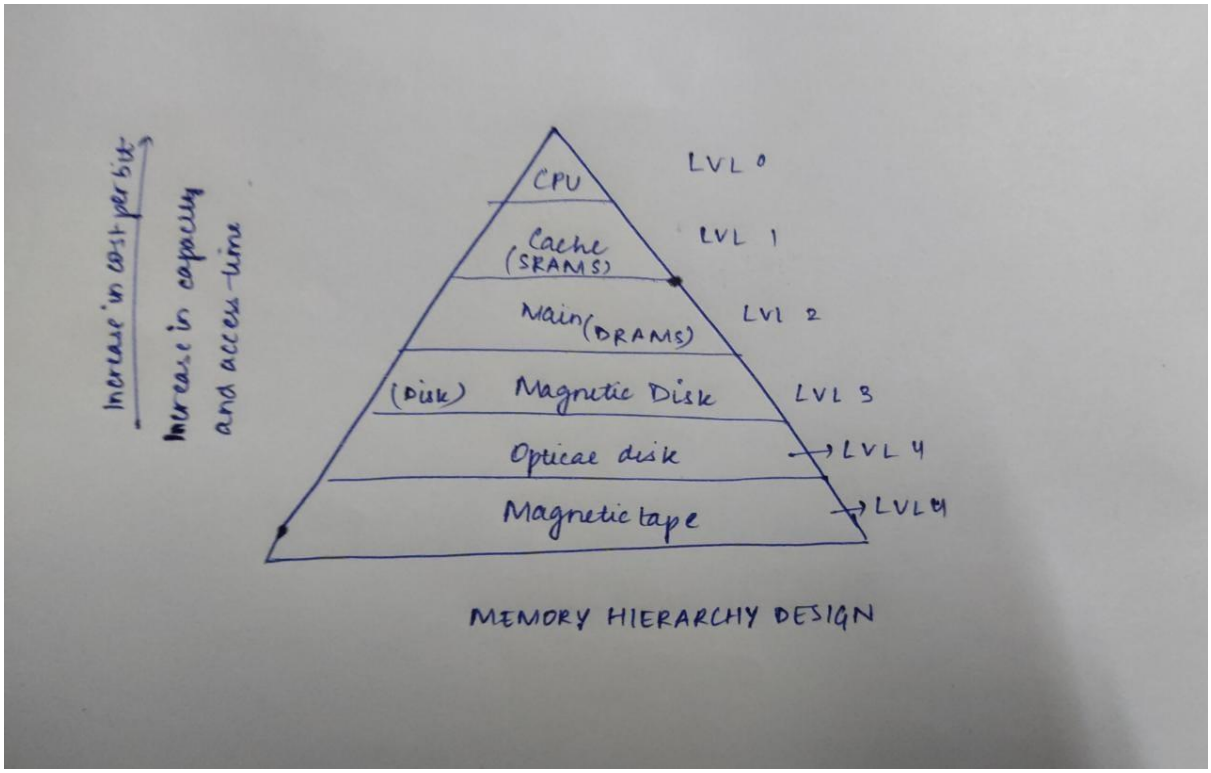
Thus, the CPU makes the processes to share time slices between them and execute accordingly. As soon as time quantum of one process expires, another process begins its execution.

Here also basically a context switch is occurring but it is occurring so fast that the user is able to interact with each program separately while it is running. This way, the user is given the illusion that multiple processes/ tasks are executing simultaneously. But actually, only one process/ task is executing at a particular instant of time. In multitasking, time sharing is best manifested because each running process takes only a fair quantum of the CPU time.

At any time, the CPU is executing only one task while other tasks are waiting for their turn. The illusion of parallelism is achieved when the CPU is reassigned to another task. So, for multitasking to take place, firstly there should be multiprogramming i.e., presence of multiple programs ready for execution. And secondly the concept of time sharing.

3. Discuss Memory Hierarchy.

Memory Hierarchy is an enhancement to organize the memory such that it can minimize the access time. The Memory Hierarchy was developed based on a program behavior known as locality of references.



Cache (MB): Cache is the fastest accessible memory of a computer system. It's access speed is in the order of a few nanoseconds. It is volatile and expensive, so the typical cache size is in the order of megabytes.

Main memory (GB): Main memory is arguably the most used memory. When discussing computer algorithms such as quick sort, balanced binary sorted trees, or fast Fourier transform, one typically assumes that the algorithm operates on data stored in the main memory. The main memory is reasonably fast, with access speed around 100 nanoseconds. It also offers larger capacity at a lower cost. Typical main memory is in the order of 10 GB. However, the main memory is volatile.

Secondary storage (TB): Secondary storage refers to nonvolatile data storage units that are external to the computer system. Hard drives and solid state drives are examples of secondary storage. They offer very large storage capacity in the order of terabytes at very low cost. Therefore, database servers typically have an array of secondary storage devices with data stored distributed and redundantly across these devices. Despite the continuous improvements in access speed of hard drives, secondary storage devices are several magnitudes slower than main memory. Modern hard drives have access speed in the order of a few milliseconds.

Tertiary storage (PB): Tertiary storage refers storage designed for the purpose data backup. Examples of tertiary storage devices are tape drives are robotic driven disk arrays. They are capable of petabyte range storage, but have very slow access speed with data access latency in seconds or minutes.

- External Memory or Secondary Memory – Comprising of Magnetic Disk, Optical Disk, Magnetic Tape i.e. peripheral storage devices which are accessible by the processor via I/O Module.
- Internal Memory or Primary Memory – Comprising of Main Memory, Cache Memory & CPU registers. This is directly accessible by the processor.

We can infer the following characteristics from above figure:

Capacity: It is the global volume of information the memory can store. As we move from top to bottom in the Hierarchy, the capacity increases.

Access Time: It is the time interval between the read/write request and the availability of the data. As we move from top to bottom in the Hierarchy, the access time increases.

Performance: The speed gap increases between the CPU registers and Main Memory due to large difference in access time earlier. This results in lower performance of the system and thus, enhancement was required. This enhancement was made because of which the performance of the system increases. One of the most significant ways to increase system performance is minimizing how far down the memory hierarchy one has to go to manipulate data.

Cost per bit: As we move from bottom to top in the Hierarchy, the cost per bit increases i.e. Internal Memory is costlier than External Memory

4. What are System Calls? Explain the two modes of Operating System.

A system call is the programmatic way in which a computer program requests a service from the kernel of the operating system it is executed on. A system call is a way for programs to interact with the operating system. A computer program makes a system call when it makes a request to the operating system's kernel. System call provides the services of the operating system to the user programs via Application Program Interface (API). It provides an interface between a process and operating system to allow user-level processes to request services of the operating system. System calls are the only entry points into the kernel system. All programs needing resources must use system calls.

Types of System Calls:

- Process Control-These system calls deal with processes such as process creation, process termination etc.
- File Management-These system calls are responsible for file manipulation such as creating a file, reading a file, writing into a file etc.
- Device Management-These system calls are responsible for device manipulation such as reading from device buffers, writing into device buffers etc.
- Information Maintenance-These system calls handle information and its transfer between the operating system and the user program.

- Communication-These system calls are useful for inter-process communication. They also deal with creating and deleting a communication connection.

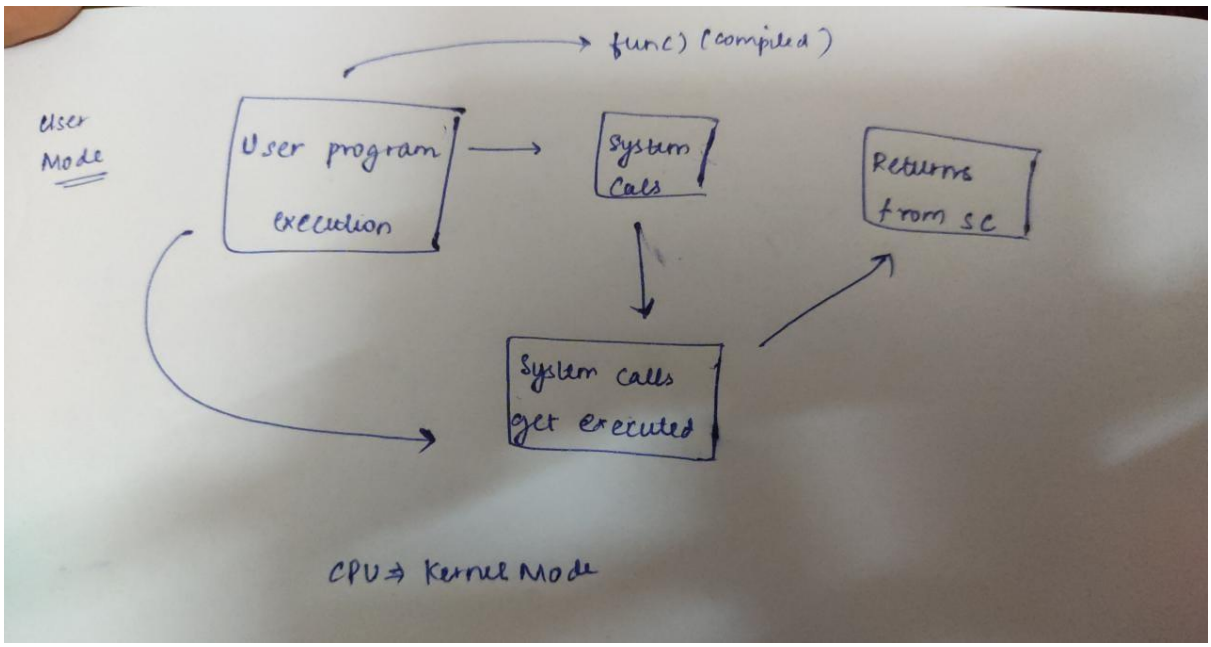
User Mode

The system is in user mode when the operating system is running a user application such as handling a text editor. The transition from user mode to kernel mode occurs when the application requests the help of operating system or an interrupt or a system call occurs. The mode bit is set to 1 in the user mode. It is changed from 1 to 0 when switching from user mode to kernel mode.

Kernel Mode

The system starts in kernel mode when it boots and after the operating system is loaded, it executes applications in user mode. There are some privileged instructions that can only be executed in kernel mode.

These are interrupt instructions, input output management etc. If the privileged instructions are executed in user mode, it is illegal and a trap is generated.The mode bit is set to 0 in the kernel mode. It is changed from 0 to 1 when switching from kernel mode to user mode.



5. What is Operating System and discuss in brief the different services provided by OS.

An Operating System (OS) is a software that acts as an interface between computer hardware components and the user. Every computer system must have at least one operating system to run other programs. It provides services to both the users and to the programs, provides programs an environment to execute and provides users the services to execute the programs in a convenient manner.

A few common services provided by an operating system −

Program execution: Operating systems handle many kinds of activities from user programs to system programs like printer spooler, name servers, file server, etc. Each of these activities is encapsulated as a process.

A process includes the complete execution context (code to execute, data to manipulate, registers, OS resources in use). Following are the major activities of an operating system with respect to program management −

- Loads a program into memory.
- Executes the program.
- Handles program's execution.
- Provides a mechanism for process synchronization.
- Provides a mechanism for process communication.
- Provides a mechanism for deadlock handling.

I/O Operation: An I/O subsystem comprises of I/O devices and their corresponding driver software. Drivers hide the peculiarities of specific hardware devices from the users. An Operating System manages the communication between user and device drivers. I/O operation means read or write operation with any file or any specific I/O device. Operating system provides the access to the required I/O device when required.

File system manipulation: A file represents a collection of related information. Computers can store files on the disk (secondary storage), for long-term storage purpose. Examples of storage media include magnetic tape, magnetic disk and optical disk drives like CD, DVD. Each of these media has its own properties like speed, capacity, data transfer rate and data access methods. A file system is normally organized into directories for easy navigation and usage. These directories may contain files and other directions. Following are the major activities of an operating system with respect to file management −

- Program needs to read a file or write a file.
- The operating system gives the permission to the program for operation on file.
- Permission varies from read-only, read-write, denied and so on.
- Provides an interface to the user to create/delete files.
- Provides an interface to the user to create/delete directories.
- Provides an interface to create the backup of file system.

Communication: In case of distributed systems which are a collection of processors that do not share memory, peripheral devices, or a clock, the operating system manages communications between all the processes. Multiple processes communicate with one another through communication lines in the network. The OS handles routing and connection strategies, and the problems of contention and security. Following are the major activities of an operating system with respect to communication –

- Two processes often require data to be transferred between them

- Both the processes can be on one computer or on different computers, but are connected through a computer network.
- Communication may be implemented by two methods, either by Shared Memory or by Message Passing.

Error handling: Errors can occur anytime and anywhere. An error may occur in CPU, in I/O devices or in the memory hardware. Following are the major activities of an operating system with respect to error handling −

- The OS constantly checks for possible errors.
- The OS takes an appropriate action to ensure correct and consistent computing.

Resource Management: In case of multi-user or multi-tasking environment, resources such as main memory, CPU cycles and files storage are to be allocated to each user or job. Following are the major activities of an operating system with respect to resource management −

- The OS manages all kinds of resources using schedulers.
- CPU scheduling algorithms are used for better utilization of CPU.

Protection: Considering a computer system having multiple users and concurrent execution of multiple processes, the various processes must be protected from each other's activities. Protection refers to a mechanism or a way to control the access of programs, processes, or users to the resources defined by a computer system. Following are the major activities of an operating system with respect to protection −

- The OS ensures that all access to system resources is controlled.
- The OS ensures that external I/O devices are protected from invalid access attempts.
- The OS provides authentication features for each user by means of passwords.

6. The structure of process Pi (i=0 or 1) and Pj (j == 1 or 0) is shown in the above code. Two processes, P0 and P1 share the following variables: boolean flag[2]; initially false int turn; Prove that the algorithm satisfies all three requirements for the critical-section problem.

The three requirements the algorithm should satisfy to be a critical section problem are: Mutually exclusivity, progress and Bounded waiting.

Mutually Exclusive: As the process **Pi** executes the entry section code it first turns its flag value to true and then proceeds to check the flag value of the other process, if the other process is not in its critical state, its process flag will be false and the process to **Pi** will enter the critical section.

But in case the other process is in critical section and flag [j] is true then Pi will enter the while loop, check if turn is equal to j and if turn is j, it will change its flag value to false and stay in the while loop until the process Pj executes the exit section code and makes the turn value not equal to j and flag[j] as false.

Therefore, we can see that the process will not enter the critical section if another process is in its critical section and thus, we can say that its mutually exclusive.

Progress: As we can see the processes upon entering the entry section turn their flag values to true and are free to enter the critical section if the other process is in the critical section. Therefore, we can say that no process will have to wait for another to access the critical section if the critical section is not being used. Progress is non restricted.

Bounded wait: If a process executes the while statement it will change its flag value back to false, this way the other process will not get stuck in a C loop that depends on the other process to come out of the loop. If a process after completion wants to have access of critical section which is in waiting, it will have to wait for the turn value to change which will only happen when the execution of other process is complete. No process has to wait indefinitely. As all the three conditions are satisfied, proved.

7. Case Study:

The Fibonacci sequence is the series of numbers 0, 1, 1, 2, 3, 5, 8, ...

Formally, it can be expressed as:

fib 0 = 0

fibl = 1

jibn = jibn-l + jibn-2

Write a C program using the fork() system call that generates the Fibonacci sequence in the child process. The number of the sequence will be provided in the command line. For example, if 5 is provided, the first five numbers in the Fibonacci sequence will be output by the child process. Because the parent and child processes have their own copies of the data; it will be necessary for the child to output the sequence.

CODE:

```
#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

#include <sys/wait.h>

int main()

{

int a=0, b=1, n=a+b,i;

printf("Enter the number of a Fibonacci Sequence:\n");

scanf("%d ", &i);
```

```
pid_t pid = fork();
if (pid == 0)
{
   printf("Child is make the Fibonacci\n");
   printf("0 %d ",n);
   while (i>0) {
      n=a+b;
      printf("%d ", n);
      a=b;
      b=n;
      i--;
      if (i == 0) {
         printf("\nChild ends\n");
      }
   }
}
   else
   {
      printf("Parent is waiting for child to complete...\n");
      waitpid(pid, NULL, 0);
      printf("Parent ends\n");
   }
   return 0;
}
```

OUTPUT SCREEN:

```
Ananya Prasad@LAPTOP-N1ODGD2F /home
$ ./F2.exe
Enter the number of a Fibonacci Sequence:
7
9
Parent is waiting for child to complete...
Child is make the Fibonacci
0 1 1 2 3 5 8 13 21
Child ends
Parent ends
```