**Assignment 1**

**Operating Systems**

**Q1. What are the main functions of an operating system?**

**Sol.**

There are Many Functions those are Performed by the [Operating System](http://ecomputernotes.com/fundamental/disk-operating-system/what-is-operating-system) But the Main Goal of Operating System is to Provide the Interface between the user and the hardware Means Provides the Interface for Working on the System by the user. The various Functions those are Performed by the Operating System are as Explained below:-

**Operating System as a Resource Manager**

**Operating System Also Known as the Resource Manager** Means Operating System will Manages all the Resources those are Attached to the System means all the Resource like Memory and [Processor](http://ecomputernotes.com/fundamental/introduction-to-computer/what-is-cpu) and all the Input output Devices those are Attached to the System are Known as the Resources of the [Computer](http://ecomputernotes.com/fundamental/introduction-to-computer/what-is-computer" \o "Computer is an advanced electronic device that takes raw data as input from the user and processes these data under the control of set of instructions (called program)." \t "_self)System and the Operating system will Manage all the Resources of the System. The Operating System will identify at which Time the [CPU](http://ecomputernotes.com/fundamental/introduction-to-computer/what-is-cpu) will perform which Operation and in which Time the Memory is used by which Programs. And which [Input Device](http://ecomputernotes.com/fundamental/input-output-and-memory/list-various-input-and-output-devices) will respond to which Request of the user means When the Input and Output Devices are used by the which Programs. So this will manage all the Resources those are attached to the Computer System.

**Operating System also Controls the all the Storage Operations means how the data or files will be Stored into the**[**computers**](http://ecomputernotes.com/fundamental/introduction-to-computer/what-is-computer) and how the Files will be Accessed by the users etc. All the Operations those are Responsible for Storing and Accessing the Files is determined by the Operating System Operating System also Allows us Creation of Files, Creation of Directories and Reading and Writing the data of Files and Directories and also Copy the contents of the Files and the Directories from One Place to Another Place.

1)  **Process Management :** **The Operating System also Treats the Process Management means all the Processes those are given by the user or the Process those are System ‘s own Process are Handled by the Operating System** . The Operating System will Create the Priorities foe the user and also Start or Stops the Execution of the Process and Also Makes the Child Process after dividing the Large Processes into the Small Processes.

2) **Memory Management:** Operating System also Manages the Memory of the Computer System means Provide the Memory to the Process and Also Deallocate the Memory from the Process. And also defines that if a Process gets completed then this will deallocate the Memory from the Processes.

3)   **Extended Machine :** Operating System also behaves like an Extended Machine means Operating system also Provides us Sharing of Files between Multiple Users, also Provides Some Graphical Environments and also Provides Various Languages for Communications and also Provides Many Complex Operations like using Many Hardware’s and Software’s.

4)  **Mastermind:** Operating System also performs Many Functions and for those Reasons we can say that Operating System is a Mastermind. It provides Booting without an Operating System and Provides Facility to increase the Logical Memory of the Computer System by using the Physical Memory of the Computer System and also provides various Types of Formats Like NTFS and FAT File Systems.

Operating System also controls the Errors those have been Occurred into the Program and Also Provides Recovery of the System when the System gets Damaged Means When due to Some Hardware Failure , if System Doesn’t Works properly then this Recover the System and also Correct the System and also Provides us the Backup Facility. And Operating System also breaks the large program into the Smaller Programs those are also called as the threads. And execute those threads one by one.

**Q2. What does the CPU do when it has no program to execute?**

**Sol.**

The CPU is in a halt state. That is, it stops fetching instructions and waits for an interrupt. In normal operation, CPUs continually fetch and execute new instructions, usually from sequential memory locations, with the occasional jump to a new location. Interrupts cause the CPU to save its state, jump to a different location and start doing something completely different. In a halt state, the CPU responds to interrupts but doesn't do anything else.   
  
In a typical desktop or laptop system, the operating system will detect there are no processes needing to run, schedule an interrupt to occur a few milliseconds in the future and then run a special HALT instruction, at which point the CPU stops fetching new instructions. In older CPUs, there was no HALT instruction and they would just execute a busy loop, but this doesn't happen now as its very power inefficient.  
  
Exactly how the halt state is implemented varies. The CPU may just sit with all its circuitry alive but doing nothing, but usually at least the clock will be shut off to everything except the unit responsible for handling interrupts, which will then wake everything up.  In low power processors, the system may turn off the power to the halted CPU and rely on some external circuit to turn it back on if an interrrupt comes in.

**Q3. Int main()**

**{**

**int i, int j;**

**scanf(“%d”, &i);**

**for(j=0; j<i; j++)**

**{**

**sum= j+i;**

**}**

**printf(“%d”, sum);**

**exit(0);**

**}**

**In the above problem, differentiate each and every line of code as CPU execution or I/O execution.**

**Sol.**

In the following program

int main() // CPU execution

{

int i, int j; //CPU execution

scanf(“%d”, &i); // I/O execution

for(j=0; j<i; j++)

CPU execution

{

sum= j+i;

}

printf(“%d”, sum); // I/O execution

exit(0); } // CPU execution

**Q4. Difference between Multi-programming, Multi-tasking and Multi-processing?**

**Sol.**

**Multiprogramming**

In a multiprogramming system there are one or more programs loaded in main memory which are ready to execute. Only one program at a time is able to get the CPU for executing its instructions (i.e., there is at most one process running on the system) while all the others are waiting their turn.  
The main idea of multiprogramming is to maximize the use of CPU time. Indeed, suppose the currently running process is performing an I/O task (which, by definition, does not need the CPU to be accomplished). Then, the OS may interrupt that process and give the control to one of the other in-main-memory programs that are ready to execute (i.e. process context switching). In this way, no CPU time is wasted by the system waiting for the I/O task to be completed, and a running process keeps executing until either it voluntarily releases the CPU or when it blocks for an I/O operation. Therefore, the ultimate goal of multiprogramming is to keep the CPU busy as long as there are processes ready to execute.  
Note that in order for such a system to function properly, the OS must be able to load multiple programs into separate areas of the main memory and provide the required protection to avoid the chance of one process being modified by another one. Other problems that need to be addressed when having multiple programs in memory is fragmentation as programs enter or leave the main memory. Another issue that needs to be handled as well is that large programs may not fit at once in memory which can be solved by using pagination and virtual memory. Please, refer to [this article](https://gabrieletolomei.wordpress.com/virtual-memory-paging-and-swapping/) for more details on that.  
Finally, note that if there are N ready processes and all of those are highly CPU-bound (i.e., they mostly execute CPU tasks and none or very few I/O operations), in the very worst case one program might wait all the other N-1 ones to complete before executing.

**Multiprocessing**  
Multiprocessing sometimes refers to executing multiple processes (programs) at the same time. This might be misleading because we have already introduced the term “multiprogramming” to describe that before.  
In fact, multiprocessing refers to the hardware (i.e., the CPU units) rather than the software (i.e., running processes). If the underlying hardware provides more than one processor then that is multiprocessing. Several variations on the basic scheme exist, e.g., multiple cores on one die or multiple dies in one package or multiple packages in one system.  
Anyway, a system can be both multiprogrammed by having multiple programs running at the same time and multiprocessing by having more than one physical processor.

**Multitasking**  
Multitasking has the same meaning of multiprogramming but in a more general sense, as it refers to having multiple (programs, processes, tasks, threads) running at the same time. This term is used in modern operating systems when multiple tasks share a common processing resource (e.g., CPU and Memory). At any time the CPU is executing one task only while other tasks waiting their turn. The illusion of parallelism is achieved when the CPU is reassigned to another task (i.e. process or thread context switching).  
There are subtle differences between multitasking and multiprogramming. A task in a multitasking operating system is not a whole application program but it can also refer to a “thread of execution” when one process is divided into sub-tasks. Each smaller task does not hijack the CPU until it finishes like in the older multiprogramming but rather a fair share amount of the CPU time called quantum.  
Just to make it easy to remember, both multiprogramming and multitasking operating systems are **(CPU) time sharing** systems. However, while in multiprogramming (older OSs) one program as a whole keeps running until it blocks, in multitasking (modern OSs) time sharing is best manifested because each running process takes only a fair quantum of the CPU time.

**Q5. Difference between a program and a process?**

**Sol.**

**Program**

- A program is a set of instructions that are to perform a designated task, where as the process is an operation which takes the given instructions and perform the manipulations as per the code, called ‘execution of instructions’. A process is entirely dependent of a ‘program’.

**Process**  
- A process is a module that executes modules concurrently. They are separate loadable modules. Where as the program perform the tasks directly relating to an operation of a user like word processing, executing presentation software etc.

**Q6. Categorize the following state: NEW, READY, RUN, BLOCK, TERMINATE, SUSPEND READY, SUSPEND WAIT as main memory or secondary memory.**

**Sol.**

**NEW**: secondary memory

**READY**: main memory

**RUN**: main memory

**BLOCK**: main memory

**TERMINATE**: secondary memory

**SUSPEND READY**: secondary memory

**SUSPEND WAIT** : secondary memory

**Q7. Define the term context switch with a appropriate example.**

**Sol.**

A context switch is a procedure that a computer's [CPU](http://searchcio-midmarket.techtarget.com/definition/CPU) (central processing unit) follows to change from one [task](http://whatis.techtarget.com/definition/task) (or process) to another while ensuring that the tasks do not conflict. Effective context switching is critical if a computer is to provide user-friendly [multitasking](http://searchcio-midmarket.techtarget.com/definition/multitasking). In a CPU, the term "context" refers to the data in the [registers](http://whatis.techtarget.com/definition/register) and program counter at a specific moment in time. A register holds the current CPU [instruction](http://searchcio-midmarket.techtarget.com/definition/instruction). A program counter, also known as an instruction address register, is a small amount of fast [memory](http://searchmobilecomputing.techtarget.com/definition/memory) that holds the [address](http://searchnetworking.techtarget.com/definition/address) of the instruction to be executed immediately after the current one. A context switch can be performed entirely in [hardware](http://searchcio-midmarket.techtarget.com/definition/hardware) (physical media). Older CPUs, such as those in the [x86](http://searchwinit.techtarget.com/definition/x86) series, do it that way. However, most modern CPUs perform context switches by means of [software](http://searchsoa.techtarget.com/definition/software)(programming). A modern CPU can perform hundreds of context switches per second. Therefore, the user gets the impression that the computer is performing multiple tasks in a [parallel](http://searchcio-midmarket.techtarget.com/definition/parallel) fashion, when the CPU actually alternates or rotates between or among the tasks at a high rate of speed.

**Q8. Consider a system with ‘n’ CPU processors and ‘m’ processes, then answer the following queries regarding minimum and maximum number of processes:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Minimum** | | **Maximum** | |
| **Ready** | **?** | | **?** |
| **Running** | **?** | | **?** |
| **Block** | **?** | | **?** |

**Sol.**

|  |  |  |
| --- | --- | --- |
|  | **Minimum** | **Maximum** |
| Ready | 0 | M |
| Running | 0 | N |
| Block | 0 | M |

**Q9. Define the responsibilities of: Short term scheduler, Middle term scheduler and Long term scheduler.**

**Sol.**

**Schedulers** are special system software which handle process scheduling in various ways. Their main task is to select the jobs to be submitted into the system and to decide which process to run. Schedulers are of three types −

* Long-Term Scheduler
* Short-Term Scheduler
* Medium-Term Scheduler

**Long Term Scheduler :**

It is also called a **job scheduler**. A long-term scheduler determines which programs are admitted to the system for processing. It selects processes from the queue and loads them into memory for execution. Process loads into the memory for CPU scheduling

**Short Term Scheduler :**

It is also called as **CPU scheduler**. Its main objective is to increase system performance in accordance with the chosen set of criteria. It is the change of *ready state to running state* of the process.*CPU scheduler selects a process among the processes that are ready to execute and allocates CPU to one of them.*

Short-term schedulers, also known as dispatchers, make the decision of which process to execute next. Short-term schedulers are *faster* than long-term schedulers.

**Medium Term Scheduler :**

Medium-term scheduling is a part of **swapping**. It removes the processes from the memory. It reduces the degree of multiprogramming. The medium-term scheduler is in-charge of handling the swapped out-processes.

A running process may become suspended if it makes an I/O request. A suspended processes cannot make any progress towards completion. In this condition, to remove the process from memory and make space for other processes, the suspended process is moved to the secondary storage. This process is called **swapping**, and the process is said to be swapped out or rolled out. Swapping may be necessary to improve the process mix.

**Q10. State some principles of giving a better user interactivity**.

**Sol.**

**ANSWER 10:**

1. **LEARNABILITY:**  
    Another very important core principle is the ability to easily learn and use an interface after using it for the first time.
2. **CONSISTENCY:**  
   As well as **matching people’s expectations** through terminology, layout and interactions the way in which they are used should be consistent throughout the process and between related applications. By maintaining consistency users learn more quickly, this can be achieved by re-applying in one part of the application their prior experiences from another.
3. **MATCH USER EXPERIENCE AND EXPECTATION:**  
   By matching the sequence of steps, layout of information and terminology used with the expectations and prior experiences of the user, the friction and discomfort of learning a new system will be reduced.

**Q11. Why can’t a virus just take hold of the system and keep running? Think and try to**

**Sol.**

Virus programs out there in the Internet are not that smart to get into a computer hardware. A CPU is a hardware component and that cannot be attacked directly by a software program (Virus). So the Virus program first needs to get into the Motherboard firmware (BIOS) to take control of the whole system. But currently there are no malwares known to the world which could do such complicated things. It doesn’t mean that they do not exist. A CPU is at the lowest end of all computer components. It is coded in a different language then your operating system (CPU is coded in Assembly and OS mainly in C++, the virus has to take over loads of parts before it can reach the CPU. Also a CPU isn't a single component. It has two more units inside it called the Arithmetic Logic Unit and the Control Unit. It is basically super complex to code a virus to do so.

**Q12. What is the responsibility of dispatcher?**

**Sol.**

The dispatcher is the module that gives control of the CPU to the process selected by the short-time scheduler(selects from among the processes that are ready to execute).  
  The function involves :  
  Swithching context  
  Switching to user mode  
  Jumping to the proper location in the user program to restart that program.

**Q13. What are the applications of real time operating system?**

**Sol.**

Real time operating system are known for -  
1.Deterministic- they execute functions in fixed amount of time  
2.Correctness- time at which result produced   
3.Predictability- all constrints related to timing meet  
  
Application based on classification of real time operating system -  
  
\*SOFT RTOS - performance degraded but not destroyed by failure to meet response time constraints.  
Used in Multimedia, interactive video games   
  
\*FIRM RTOS - missing more than few deadline, may lead to catastrophe.  
In robot weed killer  
  
\*Hard RTOS- failure to meet single deadlines may lead to catastrophic failure  
In aircraft control system, aviation, nuclear power systems, cheamical plants ,life support system.

**Q14.What do understand by the term system call?**

**Sol.**

In [computing](https://en.wikipedia.org/wiki/Computing), a **system call** is the programmatic way in which a [computer program](https://en.wikipedia.org/wiki/Computer_program) requests a service from the [kernel](https://en.wikipedia.org/wiki/Kernel_(computing)) of the [operating system](https://en.wikipedia.org/wiki/Operating_system) it is executed on. This may include hardware-related services (for example, accessing a [hard disk drive](https://en.wikipedia.org/wiki/Hard_disk_drive)), creation and execution of new [processes](https://en.wikipedia.org/wiki/Process_(computing)), and communication with integral [kernel services](https://en.wikipedia.org/wiki/Kernel_service) such as [process scheduling](https://en.wikipedia.org/wiki/Process_scheduling). System calls provide an essential interface between a process and the operating system.

In most systems, system calls can only be made from [userspace](https://en.wikipedia.org/wiki/Userspace" \o "Userspace) processes, while in some systems, [OS/360 and successors](https://en.wikipedia.org/wiki/OS/360_and_successors) for example, privileged system code also issues system calls

**Q15. What is the use of Fork and Exec system call?**

**Sol.**

### **Fork**

When you come to metaphorical "fork in the road" you generally have two options to take, and your decision effects your future. Computer programs reach this fork in the road when they hit the fork() system call.

At this point, the operating system will create a new process that is exactly the same as the parent process. This means all the state that was talked about previously is copied, including open files, register state and all memory allocations, which includes the program code.

The return value from the system call is the only way the process can determine if it was the existing process or a new one. The return value to the parent process will be the Process ID (PID) of the child, whilst the child will get a return value of 0.

At this point, we say the process has forked and we have the parent-child relationship as described above.

### **Exec**

Forking provides a way for an existing process to start a new one, but what about the case where the new process is not part of the same program as parent process? This is the case in the shell; when a user starts a command it needs to run in a new process, but it is unrelated to the shell.

This is where the exec system call comes into play. exec will replace the contents of the currently running process with the information from a program binary.

Thus the process the shell follows when launching a new program is to firstly fork, creating a new process, and then exec (i.e. load into memory and execute) the program binary it is supposed to run.