**Assignment 1**

**Submitted to: Siddharth Gupta**

**Submitted by: Anushka Khattri(RollNo.-37 Sec-A(CS))**

Ans1) The main functions of an OS are:

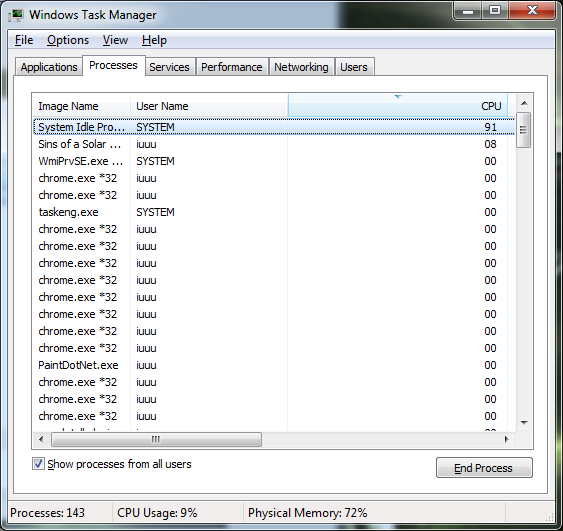
* **Process Management**: It manages all the process from start to shut down of a computer. It handles creation and deletion of user and system processes.
* **Memory Management:** It decide which process are loaded in the memory when memory space becomes available. It allocate and deallocate the memory space as needed.
* **File Management:** It create files and directories ,rename ,move, copy and delete files.
* **Security Management:** It maintain the security of each processes by providing a mutual exclusion.
* **Command interpreter:** A command interpreter is an interface between system and the user. There are two types of interface:

i)Command Line Interface

ii)Graphic User Interface

With a command line user interface the user interact with the OS by typing command to perform specific tasks whereas with a GUI the user interacts with the OS by using a mouse to access windows icons and menus.

### Ans2)

[](http://www.extremetech.com/wp-content/uploads/2014/11/CPU-Idle.png)

For x86 chips running Windows, this function takes the form of the halt (HLT) instruction and the Windows System Idle Process. Windows schedules the idle process to run on a CPU core only when there’s no other threads eligible for execution on that CPU. If you look at the list of running processes on a Windows system, you’ll often see the System Idle Process running at a high percentage. Unlike the other processes on that list, however, this is actually a measure of how little the CPU is working, not how much. The HLT instruction is designed to consume as little power as possible and dropping the CPU into this state saves significant amounts of energy. A [computer processor](https://en.wikipedia.org/wiki/Computer_processor) is described as idle when it is not being used by any [program](https://en.wikipedia.org/wiki/Computer_program). Every program or task that runs on a computer system occupies a certain amount of processing time on the CPU. If the CPU has completed all tasks it is idle. Modern processors use idle time to save power. Common methods are reducing the clock speed along with the CPU voltage and sending parts of the processor into a sleep state. On processors that have a halt instruction that stops the CPU until an interrupt occurs, such as [x86](https://en.wikipedia.org/wiki/X86)'s [HLT](https://en.wikipedia.org/wiki/HLT_(x86_instruction)) instruction, it may save significant amounts of power and heat if the idle task consists of a loop which repeatedly executes HLT instructions.

Most [operating systems](https://en.wikipedia.org/wiki/Operating_system)will display an idle task, which is a special task loaded by the OS [scheduler](https://en.wikipedia.org/wiki/Scheduling_(computing)) only when there is nothing for the computer to do. The idle task can be hard-coded into the scheduler, or it can be implemented as a separate task with the lowest possible priority.

Ans3) 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

Ans4)Multiprogramming:

* More than one program that is to be executed by the processor loaded into the memory.
* Suppose we have two programs loaded into the memory . The first program that load will get execute. At one point of time it requires input from the user or waiting for some, during that time CPU is idle.Instead of wasting its time, the CPU will begin to execute the second program.

Multitasking:

* Both the memory and CPU time is shared among the tasks.
* It gives an illusion that all the processes or tasks are performed simultaneously.
* In desktop e.g we can listen to music, download things simultaneously.
* CPU time is shared equally among the processes or tasks.

Multiprocessing:

* More than one processor in a single computer eg octa, quad, dual core processor.
* These processors share two things in common:

i)Memory

ii)Peripherals.

By sharing the memory and peripherals they are able to execute different task simultaneously.

Ans5)  **Program:**

Program is an executable file containing set of instructions written to perform a specific job on your computer.

Programs are stored on a disk or secondary memory and they are read into the primary memory and executed by kernel. Therefore programs are referred as passive entity as it resides on a secondary memory.

Eg. Notepad.exe is an executable file containing the set of instructions which help us to edit and print the text files.

Process1

Process2

Process3

Program

**Process:**

Process is an executing instance of a program.

A process is sometimes referred as active entity as it resides on the primary memory and leave the memory if the system is .Several process may related to same program.eg you can run multiple instances of a notepad program. Each instance is referred as a process.

E.g When you double click on a notepad icon on your computer, a process is started that will run the notepad program.

Thread1

Thread2

Thread3

Process

Ans6)

**NEW**: secondary memory

**READY**: main memory

**RUN**: main memory

**BLOCK**: main memory

**TERMINATE**: secondary memory

**SUSPEND READY**: secondary memory

**SUSPEND WAIT** : secondary memory

Ans7) Context switching is the procedure of storing the state of an active process for the CPU when it has to start executing a new one. For example, process A with its address space and stack is currently being executed by the CPU and there is a system call to jump to a higher priority process B; the CPU needs to remember the current state of the process A so that it can suspend its operation, begin executing the new process B and when done, return to its previously executing process A.The major need for a context switch arises when CPU has to switch between user mode and kernel mode but some OS designs may obviate it.

A common approach to context switching is making use of a separate stack per switchable entity (thread/process), and using the stack to store the context itself. This way the context itself is merely the stack pointer. For example:

push a ;push all registers

mov old\_esp, saved\_location

mov new\_esp, esp

pop a

Here, the act of context switching is done by changing the stack pointer to a new location, and the registers are stored on the stack itself.

Ans8) n-> CPU , m-> process

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

Ans9 **•Long-term scheduler (or job scheduler)** -

–selects which processes should be brought into the ready queue.

–invoked very infrequently (seconds, minutes); may be slow.

–controls the degree of multiprogramming

**•Short term scheduler (or CPU scheduler) -**

–selects which process should execute next and allocates CPU.

–invoked very frequently (milliseconds) - must be very fast

**•Medium Term Scheduler**

–swaps out process temporarily

–balances load for better throughput

In short, LTS decide the number of process going to execute. The work of STS is to improve the performance of the CPU by minimizing/remove the indolence of the CPU by processing as many process it can so it load as many process it can. then it comes MTS ,as short term scheduler load many process so it may cause CPU overloading so here Middle Term Scheduler comes in to picture by removing process (It is doing the opposite work of Short Term Scheduler) So, the processor can run the processes smoothly.

Ans10) **1. Match User Experience and Expectations**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.

**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.

**3. Functional Minimalism**

The range of possible actions should be no more than is absolutely necessary. Providing too many options can detract from the primary functions and reduce usability by overwhelming the user with choices.

* Avoid unnecessary features and functions
* Break complex tasks into manageable sub-tasks
* Limit functions rather than the user experience.

**4. Control, Trust and Explorability**These three elements are fundamentally important to any system. If users feel in **control of the process** they will be more comfortable using the system. If the user is comfortable and in control they will **trust that the system will protect them** from making unrecoverable or unrecognized errors or from feeling stupid. Trust inspires confidence and **with confidence the user is free to explore** further.

**5. Error Prevention, Detection and Recovery.**The best way to reduce the amount of errors a user makes is to anticipate possible mistakes and prevent them from happening in the first place. If the errors are unavoidable we need to make them easy to spot and help the user to recover from them quickly and without unnecessary friction.

**Error Prevention**  
Prevent errors by:

* Disabling functions that aren’t relevant to the user
* Using appropriate controls to constrain inputs (e.g. radio buttons, dropdowns)
* Providing descriptive, clear instructions and considering preemptive help
* As a last resort provide clear warning messages

**Error Detection**  
Anticipate possible errors and provide feedback that helps users verify that:

* They’ve done what they intended to do
* What they intended to do was correct

It is important to remember that providing feedback by changing the visual state of an object or item is more noticeable than a written message.

**Error Recovery**  
If the error is unavoidable provide clearly marked ways for the user to recover from it. For example provide “back”, “undo” or “cancel” commands.

Ans11) 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.

Ans12)Dispatcher is the module that gives control of the CPU to the process selected by the short term scheduler. It receives control in kernel mode as the result of an interrupt or system call.

Ans13) Applications of Real Time operating system:

* Almost all the modern telecommunication systems make use of RTOS.
* Radar systems ,network switching control systems, satellite monitoring systems, satellite launch-control and maneuvering mechanisms , global positioning systems all have their roots in RTOS.
* Now a days RTOs are increasingly finding use in strategic and military operations. These are used in guided missile launching units, track-and-trace spy satellites, etc.

Ans14) 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. System call provides an essential interface between a process and the operating system. It may include a hardware related services, creation and execution of new processes and communication with integral kernel services such as process scheduling. E.g. open () 🡪is a system call which is used to open and create a file and it return a descriptor.

Ans15)**fork() system call:**

When fork() is called ,the OS will create a new process that is exactly the same as the parent process .This means all the state that was there previously is copied including open files, register state and all memory allocations which include the program code. The return value from the system call decide whether the new process is created or not. The return value from the parent process will the process ID of the child.

**exec() system call:**

Through fork() the existing process can create a new process but with exec() 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. 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 then execute the program.