8 Jun 2020

KOWSIK NANDAGOPAN D
CSE S4 ROLL NO 31

OPERATING SYSTEM
FIRST SERIES
JUNE 2020

1. Yes, Resoure allocation one of the major services offered by the operating system. ~~B.~~ Different processes require different types of resource in different amount. ① One way of partitioning is dividing resource.
In this approach the resource ~~may be~~ must have the property of divisibility. Some resources are indivisible (eg:- printer). Disks can be divided to 2MB etc... for each processes.
② second Approach is by using the Resource Descriptor Table. Using this table OS verify how many instances are available and how many are allocated etc... If any process comes under that restriction gets allocated to the resource. Till the resource gets allocated the processes are kept in a pool. So it is also called pool based approach.
So it is important for allocate resource for the processes in a way that deadlock won't accour.

~~2.~~

2. Yes, Defenitly, some of the ~~a~~ instruction must be ~~given~~ done in root / admin / privileged mode.
For example.
If the Instruction is to execute ~~to~~ is to get a resource that is owned by OS. ~~For exam~~ for instance deleting a root file in linux must be given root access.
~~To avoid the ~~ ~~a~~ ~~confusion~~

P.T.O

Some process may be malware which ~~so~~ may destroy the whole operating system & hardware.

For allocating hardware resources and doing some of the tasks there is a need of ~~giving the process~~ considering some instruction as privileged ones. Hence that protect the system.

4. This has two answers.

Yes, if the thread is made under kernel mode (level) So the OS will know that the process has two threads and may consider it as 2 processes even though both threads have same memory space. So the process will get chance to get executed multiple time in a cycle. Hence getting more ~~for~~ cpu time means faster execution.

No, if the thread is made under user level. The OS not even know the existance of the thread in the process. The OS will know the process is running and will chance to execute only once (depends on priority) in a cycle. Since this is single process this may take same amount or even more time if as a single threadded process run both threads are not related.

5. Context switching involves saving the data related to a process such as process state, resource, registers ~~data~~, program counters etc...

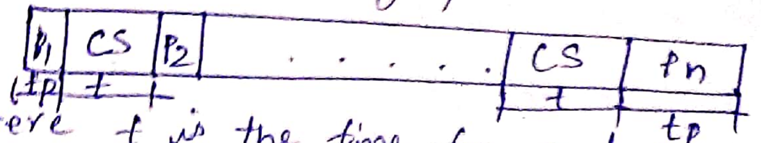
For multitasking/multiprogramming OS each time when the session times out the OS has to save the state and put the ready queue. So each time when there is timeout the OS saves data in PCB or Process descriptor table.

It is actually an overhead for the CPU. ~~[illegible]~~

For example

Case 1

If the process is very small than the quantum of time provided by OS for the process. It is really overhead. Switching continuously for very small process makes CPU under utilised for process and the the CPU will be more used by the context switching operation.



CS → Context switch

Here t is the time for context switch.

If tp < t then the context swich is an overhead.

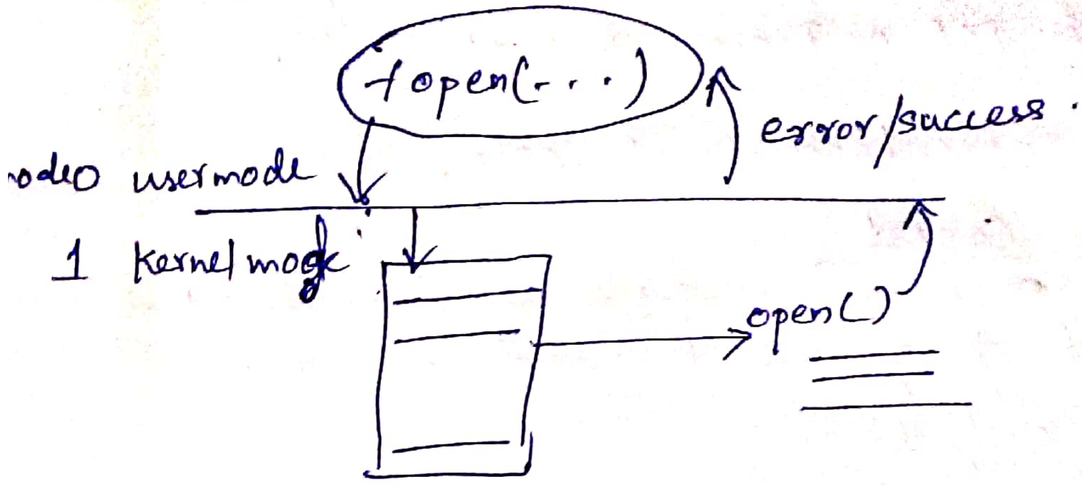
Case 2

If the process is so large tp >> t then context switching is very important else other process won't get chance to execute or have to wait so long. ~~[illegible]~~ This may put ~~[illegible]~~ some process under starvation. So we cannot avoid context switching. A huge process executing for large time is not good for users.

3. Conside the C snippet for opening a file

```
FILE *fp;
fp= fopen ("filename.txt", "[illegible]t");
```

Here when the line ~~[illegible]~~ fopen is executed the C-compiler calls an system API (System call) to open file from the specified directory. The C programmer does not have to be worried for the errors that may arise ie, file not found or file exists or an authorised



Here the fopen asks the kernel to open the file: the ~~[illegible]~~ OS now switches to kernel mode and executes. The kernel has predefined APIs that points to the location where the function is written. Here in this case kernel points to open() and execute open function. If error the error code is passed to the user program running in usermode else the pointer to the file location in main memory is provided by the operating system kernel.

The feature of system calls makes the OS function modular and program language independent. In addition to that this makes C program easy to write and platform independent