Subrato Tapaswi - 60 - D6AD

OS - Assignment 1

1) · System calls provide an interface between the process & the operating system. System calls allow user-level processes to request some services from the operating system which process itself is not allowed to do.
In handling the trap, the OS will enter in kernel mode, where it has access to privileged instructions, and can perform the desired service on behalf of user-level process.
It is because of the critical nature of operations that the OS itself does them every time they're needed.

Types of System calls:

a1) Process control (mainly used for process)
- end, abort
- load, execute
- create process, terminate process
- wait for time
- wait event, signal event
- allocate and free memory
- get process attributes, set process attributes

a2) File management (used for manipulation of files)
- create file, delete file
- open, close
- read, write, reposition
- get & set file attributes

a3) Device management (used for managing devices)
- request & release device
- read, write, reposition
- get & set device attributes
- logically attach or detach devices

a4) Information maintenance (to do info. maintenance)
- get & set time or date
- get, set system data
- get & set process, file or device attributes

a5) Communication (for communication among calls)
- create, delete communication connection
- send, receive msgs
- transfer status info
- attach or detach remote devices

2) (i) chmod - used to change file permissions after they're created
umask - sets default permissions for files when they're created
(ii) adduser - sets up new user, their folders, directories and other necessary functions easily
useradd - creates a new user without adding directories
(iii) chown - used to change the owner of a file
chmod - handles what users can do with a file
(iv) grep - used for finding particular patterns in a file and to display all fields matching that pattern
awk - more of scripting language used in manipulating data & generating reports. It enables writing of simple & effective prgms in statement forms to search through a file for a specific pattern & performs action when a match is found.

3) • To make the computer system convenient to use
• To make the use of computer hardware in efficient way.
OS may be viewed as collection of software consisting of procedures for operating the computer and providing an environment for execution of programs. It is an interface between user and computer. So, an OS makes everything in the computer to work together smoothly and efficiently.

4) • OS is system software, which acts as an interface between a user of the computer and the computer hardware. The main purpose of an OS is to provide an environment in which we can execute prgms.
Functions: (a) Perform basic tasks such as recognizing input from the keyboard, sending output to the display screen, keeping track of files and directories on the disk, and controlling peripheral devices such as disk drives and printers.
(b) Ensure that different programs and users running at the same time do not interfere with each other.

(c) Provide a software platform on top of which other programs can run.

5) • For 'n' processes to be scheduled on one processor, there can be n! different schedules possible.
• If an OS has 4 processes - P1, P2, P3, P4, for scheduling 1st process it has 4 choices, for 2nd process - 3 choices and so on.
So in total, $4 \times 3 \times 2 \times 1 = 4!$

6) Max^m no. of processes = n

7) Min^m no. of processes = 0

8) a) Run → ready ⇒ This transition is possible
The most common reason for this transition is that the running process has reached the max^m allowable time for uninterrupted execution, i.e. time-out occurs.
Other reasons can be imposition of priority levels as determined by the scheduling policy used for low level Scheduler, and arrival of a higher priority process into Ready state.

b) Run → Blocked ⇒ This transition is possible
A process is put into the Blocked state if it requests something for which it must wait. A request to OS is usually in form of system call.

d) Blocked → run: not possible

e) Run → Terminated: A process terminated from running state by completing its execution or by explicitly being killed.

c) not possible

9) SJF - Favours short processes as it will process short jobs first.
FCFS - Discriminates against short jobs since any short jobs arriving after long jobs will have a longer waiting time.

RR - treats all jobs equally (giving them equal bursts of CPU time) so short jobs will be able to leave the system faster since they will finish 1st.
Multilevel feedback queues - works similar to RR algo, discriminate favourably toward short jobs.
SRTF - Process having smallest execution time is chosen for next execution. favours short processes.

10) A race cond^n occurs when multiple processes or threads read and write data items so that the final result depends on the order of exec^n of inste. in multiple processes.
- Let us consider 2 processes, P1 & P2 share the global variable num. At some point in its exec^n, P1 updates num to value "10" and P2 updates num to value 20. Thus, these 2 processes are in a race to write num. The process that updates last determines final value of num.

There's 2 kinds of approaches to fight race cond^{n}s:
- Avoiding shared state
- Using synchronizations and atomic operations.

11) - To leave the responsibility with the processes that wish to execute concurrently. Thus, processes, whether they are system programs or application programs would be required to coordinate with one another to enforce mutual exclusion, with no support from programming language, or OS. This can be called as software approaches.

- But this approach is prone to high processing overhead and bugs, in spite it is useful to examine such approaches to gain a better understanding of complexity of concurrent processing.
- Use of special-purpose machine inst. These have adv of reducing overhead but nevertheless will be shown to be unattractive as a general purpose sol^n.

• Semaphore is OS and programming lang. software approach or mechanisms that are used to provide concurrency. It is simply a variable which is non-negative and shared between threads. This variable is used to solve critical sec^n prblm & to achieve process synchronization in the multiprocessing environment.

12) • In computing, the producer-consumer problem (a.k.a. bounded buffer prblm) is a classic eg. of multiprocess synchronization prblm. It describes 2 processes, the producer and consumer, who share a common, fixed-size buffer as a queue. The producer's job is to generate data, put it into the buffer & start again. At same time, consumer is consuming the data (removing from buffer), one piece at a time.
• The problem is to make sure that producer won't try to add data if buffer is full and consumer won't try to remove data if its empty.
• Sol^n: For the producer is either go to sleep or discard data if buffer is full. Next time the consumer removes data from buffer, it notifies producer, who fills buffer again.
• In the same way, consumer can go to sleep if it finds buffer to be empty. Next time producer puts data into buffer, it wakes up sleeping consumer. Sol^n can be reached by means of inter-process comm^n, typically using semaphores.
• An inadequate sol^n could result in a deadlock where both processes are waiting to be awakened. Problem can also be generalized to have multiple ~~producer~~ producers & consumers.

13) • A context switching is a process that involves switching of the CPU from one process or task to another. In this phenomenon, the exec^n of process that is present in running state is suspended by kernel and another process that is present in ready state is executed by CPU.
• It's one of the essential features of multitasking OS. Processes are switched so fastly that it gives an illusion to user that all processes are being executed at same time.

A context is the contents of CPUs registers and prgm counter at any point in time. Context switching can happen due to fol. reasons:
- When a process of high priority comes in ready state. In this case, the exec" of running process should be stopped & higher priority process should be given CPU for exec".
- When an interruption occurs then the process in running state should be stopped & CPU should handle interrupt before doing something el
- When a transition between user mode & kernel mode is req. then you have to perform context switching.

- Disadvantages: It requires some time for context switching.
- Time is required to save context of one process that is in the running state and then getting context of another process that is about to come in running state. During that time, there's no useful work done by CPU from user perspective. So, context switching is pure overhead in this cond".

14) - Multithreading in an interactive application may allow a prgm to continue running even if a part of it is blocked or is performing a lengthy operation, thereby increasing responsiveness to the user.
- Some OS provide a combined user level thread and kernel level thread facility. In a combined system, multiple threads within the same application can run in parallel on multiple processors and a blocking system need not block the entire process.

Multithreading models are 3 types:
- Many to many relationship
- Many to one
- One to one

User level Thread	Kernel level thread
• Faster to create and manage	• Slower to create & manage
• Implementation is by a thread library at user level.	• OS supports creation of Kernel threads.
• Generic & can run on any OS.	• Specific to the OS
• Can't take adv of multithreading.	• Can be multithreaded.
• Context switch time is less & req. no hardware support.	• Context switch time is more & hardware support is needed.
• If one user level thread perform blocking op., entire process is blocked.	• If one performs blocking op, then another thread can continue exec^n.
• Eg. Java, POSIX threads.	• Eg. Window Solaris.

15) For implementation mutual exec^n many software algos are developed. But there's high processing overhead and the risk of logical errors in these software algos. Several interesting hardware approaches to mutual exec^n. Some of these are:
- Special Machine; Compare and Swap; Exchange instruction
- Properties of Machine-inst. approach

(a) Interrupt disabling: Concurrent processes cannot have overlapped exec^n but can be interleaved in uniprocessor system. Furthermore, a process will continue to run until OS invokes a service or until it is interrupted. Therefore, to have mutual exec^n the process shouldn't be interrupted. In OS Kernel some primitives can be defined to enable or disable the interrupt.

(b) Special Machine Inst - In a multiprocessor config, several processors share access to common main memory. In this case, there's not a master/slave rel^n, rather processors behave independently in a peer rel^n. There's no interrupt mechanism b/w processors on which mutual exec^n can be based.

At the hardware level, access to a memory loc^n excludes any other access to same loc^n. With this as foundation, processor designers have proposed several machine instructions that carry out 2 actions atomically.

(c) Compare and Swap inst: One version of inst. checks a memory loc^n (*word) against a test value (testval). If memory location's current value is testval, it is replaced with newval; otherwise it is left unchanged. The old memory value is always returned; thus, the memory loc^n has been updated if the returned value is same as the test value.
This atomic instruction therefore has 2 parts:
1. A compare is made between a memory value and
2. A test value; if the values differ a swap occurs.

(d) Exchange instruction: The inst. exchanges contents of a reg. with that of a memory loc^n. Intel Pentium IA-32 architecture and the intel Itanium IA-64 architecture contain an XCHG instruction.
- Mutual exec^n protocol is based on the use of an exchange inst^a. A shared variable status is initialized to 0. Each process uses a local variable key that is initialized to 1. A process with status = 0 can enter the critical section and rest all other process are excluded.

(e) Machine-instruction approach: It has no. of advantages:
- It is applicable to any no. of processes on either a single processor or multiple processors sharing main memory.
- It is simple and therefore easy to verify.
- It can be used to support multiple critical sections; each critical section can be defined by its own variable.

14) ~~(A)~~

11) (B) 2

- let P_0, P_1 & P_2 be the 3 processes with arrival times 0, 2 & 6 resp. and CPU burst times 10, 20 & 30 resp. At time 0, P_0 is only available process so it runs. At time 2, P_1 arrives, but P_0 has shortest rem. time, so it continues. At time 6, P_2 arrives but P_0 has shortest rem. time, so it continues. At time 10, P_1 is scheduled as it's shortest rem. time process. At time 30, P_2 is scheduled.

∴ Only 2 context switches are needed. P_0 to P_1 & P_1 to P_2.

12) (B) 10.6%

→ let 3 processes be P_0, P_1 & P_2. Execn time - 10, 20 & 30 resp.
P_0 spends 1st 2 time units in I/O, 4 units of CPU time & finally 1 unit in I/O.
P_1 spends 1st 4 units in I/O, 14 units of CPU time & finally 2 unit in I/O.
P_2 - 1st 6 units in I/O, 21 units of CPU time & finaly 3 units in I/O.

PID	AT	ID	BT	IO
P_0	0	2	7	1
P_1	0	4	14	2
P_2	0	6	21	3

idle	P_0	P_1	P_2	idle
0	2	9	23	44	47

Total time = 47
Idle time = 2+3 = 5
Percentage of idle time = 5/47 × 100 = 10.6 %

13) (A) Mutual execution but not progress

The given mechanism is based on strict alternation, which guarantees always mutual exclusion and never progress.

In this mutual exec^n is satisfied because at any point of time, either $S_1 = S_2$ or $S_1 != S_2$ but not both.

But here progress isn't satisfied because suppose $S_1 = 1$ and $S_2 = 0$ and P_1 isn't interested to enter into critical sec^n, but P_2 wants to enter the critical sec^n & P_2 will not be able enter as P_1 isn't entering it.

$\therefore S_1 \neq S_2$.

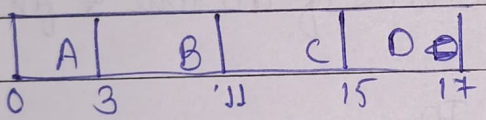
So if 1 process isn't interested in entering critical sec^n, it will not allow the other process to enter the critical sec^n which is interested.

So, progress isn't satisfied.

16)

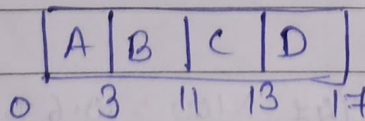
Process	Arrival time	Burst time	Priority
A	0.001	3	3
B	1.001	8	4
C	4.001	4	6
D	6.001	2	5

*FCFS:



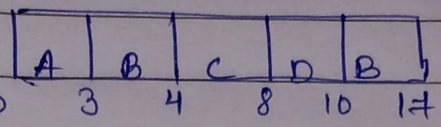
Job	Arrival time	BT	Finish time	Turn around time	Waiting time
A	0.001	3	3	3 ~~3-999~~	0
B	1.001	8	11	10 ~~9-999~~	2
C	4.001	4	15	11	7
D	6.001	2	17	11	9
		Avg		35/4 = 8.75	18/4 = 4.5

*SJF:



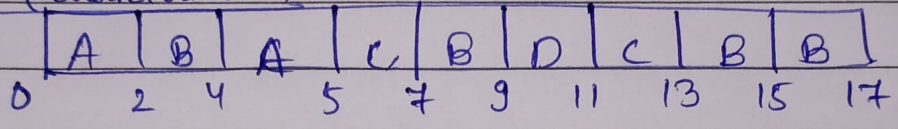
Job	AT	BT	FT	TAT	WT
A	0	3	3	3	0
B	1	8	11	10	2
C	4	4	17	13	9
D	6	2	13	7	5
			Avg =	33/4 = 8.25	16/4 = 4

* SRTF:



Job	AT	BT	FT	TaT	WT
A	0	3	3	3	0
B	1	8	17	16	8
C	4	4	8	4	0
D	6	2	10	4	2
			Avg. → 27/4 = 6.75	10/4 = 2.5	

* RR (Quantum = 2)



Job	AT	BT	FT	TaT	WT
A	0	3	5	5	2
B	1	8	17	16	8
C	4	4	13	9	5
D	6	2	11	5	3
			Avg.	35/4 = 8.75	18/4 = 4.5