OS Theory 06/03/2022

Take Home Assignment 1

Q.1 (a) When the process executes a system call. (Ex. printf eventually calls write which is a privileged instruction)

(b) Whenever there is a software interrupt (trap) due to exceptions, system calls etc. the processor transits from user mode to kernel mode

(c) When there is a hardware interrupt, like I/O buffer is ready to be filled, the processor transmits from user mode to kernel mode

(d) When we do a arithmetic operation like $\left(\frac{10}{0}\right)$ the processor transits from user mode to kernel mode to solve the operation. [Here, a signal SIGFPE will be generated which will be handled by kernel]

Q.2 advantage

(i) Thread switching is fast as the user-level threads lie in user space

(ii) memory efficient as there is sharing of resources. In kernel, there is no easy and efficient way of doing this

(iii) OS independent. Wherever the library exists, we can do easily do the API calls.

disadvantage

Mapping of user thread to kernel thread makes them slower when there is a blocking call on kernel thread, all

the other the user threads mapped to it are blocked.

Q.3 All the calculation are done for process P1

Every process will have same answers as they have same arrival and burst time and IO time

Since all process are identical, their calculations are similar

(a)

10 times { 53 ... 53 } I 200

53

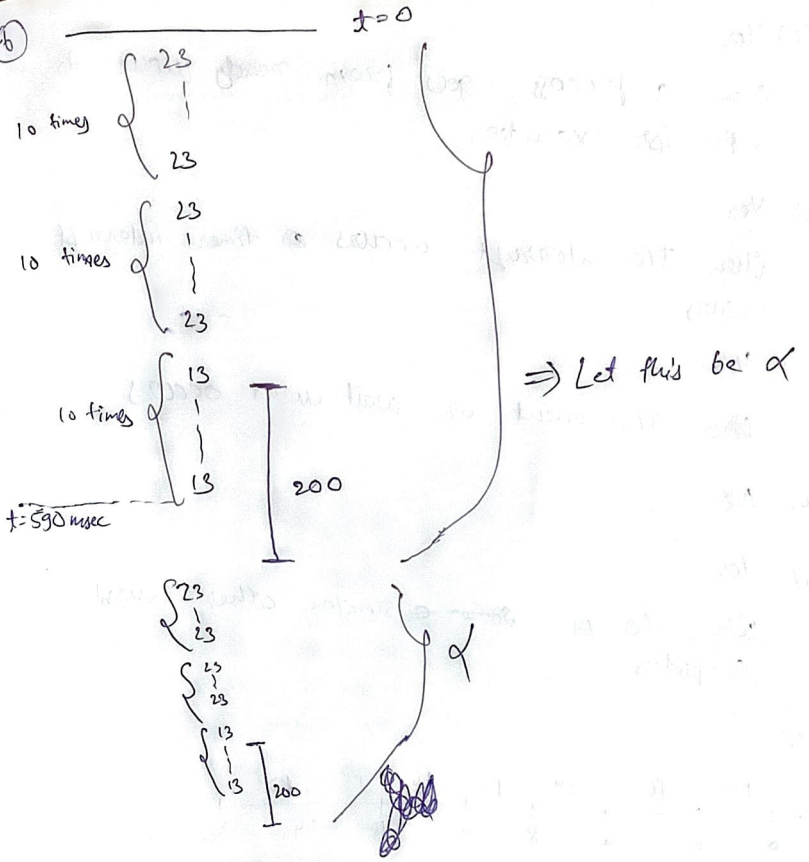
CPU Burst = 50 msec (as after that process goes to IO)
Time slice = 50 msec

Response time of 1st Iteration = 53 msec

" " " 2nd " = (10×53 + 53) − (53 + 200)
= 530 − 200
= 330 msec

" " " 3rd to 10th " = 330 msec
(as the cycle repeats because the burst time is within the time slice and after doing IO it becomes available to CPU

(b)



CPU burst = 50 msec

Time Quantum = 20 msec

Response time for each iteration = 230 + 230 + 13
= 473 msec

Logic :- at $t = \overset{590}{\cancel{525}}$ msec, all process are in I/O.
So process P1 becomes available again at
473 + 200 = 673 msec

(4) (i) Yes

When a process goes from ready queue to CPU for execution

(ii) Yes

When I/O interrupt occurs or timer interrupt occurs

(iii) Yes

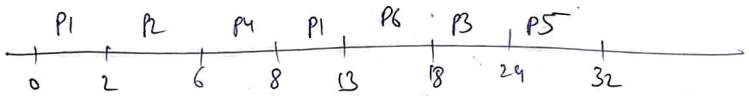
When I/O event or wait event occurs

(iv) No, process only goes from waiting to ready and the other way round. After ready it only goes to running or swapped out.

(v) Yes

When I/O or ~~some~~ similar other event completes

(5) (i)

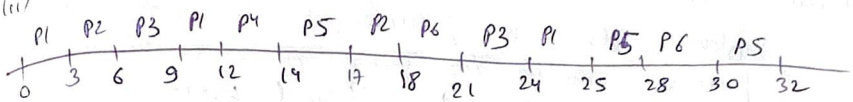


Waiting time = Time at which process completed − Time at which it would comple if there were no other process, i.e. it didn't had to wait

$$\text{avg. wait time} = \frac{(13-7)+(6-6)+(24-9)+(8-7)+(32-14)+(18-13)}{6}$$

$$= 7.5 \text{ msec}$$

(ii)



$$A.W.T = \frac{(25-7) + (8-6) + (24-9) + (14-7) + (32-14) + (30-13)}{6}$$

$$= \frac{18 + 12 + 15 + 7 + 18 + 17}{6}$$

$$= 14.5 \text{ msec}$$

(6) (i) User mode, Kernel mode

(ii) Kernel mode

(iii) (a) User does not need to know the details of hardware of his/her computer. The user application can just call the API functions provided by the kernel

(b) Kernel mode executes privileged instructions, giving them to user mode will be catastrophic as user can execute any instruction he/she wants (like forcing the OS to schedule only his/her process)

(7) (a) True

System calls are priviledged instruction to provide smooth functioning of OS, thus the processor switches to kernel mode

(b) True

Some other process may cause an interrupt or exception and the processor switches to kernel mode to handle that.

(c) False

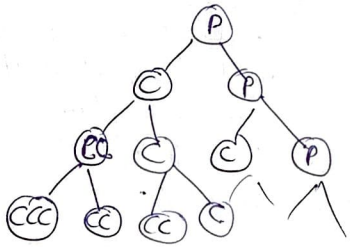
Heaps are shared between threads, as the threads of a process group have same global variables

(d) True

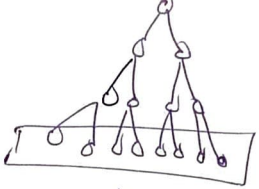
Each thread has its own copy of registers and stack. This stack space is the space where the thread execute its instruction with the ~~code~~ help of values stored in registers, stack and heap

(8) (a) $2^6 - 1 = 63$

As they are in for loop, each fork will create a child and then in the next iteration the parent and child will again both calls fork and this process will continue 6 times



Complete binary tree of height 6.



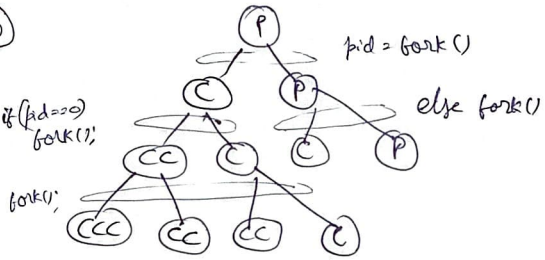
for representation of 3 height

Finally, all the ~~process~~ process ~~the~~ will be the leaves of the binary tree and there will be total $2^6 = 64$ process.

So new process = 64 - 1
= 63

(as there was 1 original process)

(b)



Total 6 leaves, so finally 6 process, but 1 original process

So, new process = 5

(9)

```
# include <sys/shm.h>
# include <sys/types.h>
.
.
.
key_t key = 1234;
int shmid = shmget (key, 100, IPC_CREAT|0666);
char *myseg = shmat (shmid, NULL, 0)
.
.
.
```