Rupinder Goyal 19CS10050

[redacted]

1

1 The following are the three ways:-

a) The process executes a system call
b) An interupt occurs
c) Synchronous exceptions like traps occur

2) Advantages:-

a) User level threads are fast and more efficient than kernel level threads

b) User level threads can also run on OS that doesnt support threads

Disadvantages:-

~~The~~ The entire process gets blocked if the user level thread performs blocking operation. In the kernel process, ~~other~~ ~~threads~~ only that thread gets ~~p~~ blocked.

4 ① Ready to Running

Yes, it is possible.

When the context switch occurs, ~~ex~~ one of the processes in the ready queue is selected and executed. Therefore state changes from Ready to Running.

2) Running to Ready state

Yes, it is ~~process~~ possible
If ~~Pop~~ the running process has a timer interupt, the process state changes from running to ready state

3) Running to Waiting

Yes, it is ~~pro~~ possible

When the ~~process run~~ running process needs an I/O, it ~~is~~ state is changed to Running to waiting state

4) Ready to Waiting

No, it is not possible
~~when a running process.~~
Before going to the waiting queue, process must be in the runing state

⑤ Waiting state to Ready State

Yes, it is possible.

~~hte~~ When an IO gets completed, the process which requested that IO, is shifted from waiting to ready state.

⑤ Preemptive Shortest Job first

P1	P2	P4	P1	P6	P3	P~~4~~5
0	2	6	8	13	18	24

P1 waiting time :- 8 − 2 = 6 ~~sec~~ msec

P2 waiting time = 0

P3 " " = 18 − 3 = 15 ~~sec~~ msec

P4 " " = 6 − 5 = 1 ~~sec~~ msec

P5 " " = ~~235~~ 24 − 6~~8~~ = ~~18sec~~ 18 ~~sec~~ msec

P6 " " = 13 − 8 = 5 ~~sec~~ msec

Total time :- 45

avg. waiting time = $\frac{45}{6}$ = 7.5 msec ii)

P1	P2	P3	P1	P4	P5	P2	P6	P3	P1	P5	P6	P5
0	3	6	9	12	14	17	18	21	24	25	28	30

Waiting times:-

P1 = 0 + (9-3) + (24-12) = 18 msec

P2 = (3-2) + 17 - 6 = 12 msec

P3 = (6-3) + (21-9) = 15 msec

P4 = (12-5) = 7 msec

P5 = (14-6) + (25-17) + (30-28)

= 18 msec

P6 = (18-8) + (28-21) = 17 msec

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

= 14.5 msec

(c)

1) There are 2 modes of operations in OS:-

a) User mode b) Kernel Mode

ii) OS enters kernel mode for executing a system call

iii)

a) User ~~sto~~ must be allowed to perform ~~such~~ certain functions that can cause harm to the system / hog the resources.

b) Users must not access certain part of memory / or memory of other programs.

Due to these reasons, we require other mode that can restrict the user mode. ~~from~~ ~~do~~

Q:7 i) True,
The syscalls are previleged functions so it cannot be run in kernel mode
So it is necessary to shift the mode

ii) True, ~~this~~ ~~case~~ This case can occur in case of any interupt or an exception.

7)

~~a) True~~,

c) False,
All the common things like Global variables are stored in heaps therefore it is common between threads

d) True,
Local variables are stored in stacks which can differ for different threads. ~~Here~~ Hence, Seperate stack for each thread must be stored.

8) a) Each ~~[illegible]~~ time, fork call will double the number of processes.
Initially, there is 1 parent process.
At the end, 2^6 processes will be there

Total ~~chid~~ child processes = $2^6 - 1$ = 63 new processes

b) first fork will create 1 ~~chid~~ child process
Child process goes in if body.
If body contains 2 fork calls.
Each will double up the processes
=> 4 processes here
Parent process goes in else body. Processes double up => 2 processes here
In total, 6 processes are there
5 new processes will be created

Q:9

```
# include <sys/ shm.h>

Key_t    key = 1234;

int  shmid  = shmget ( key, 100 , 0666| IPC_CREAT);

void* shm  = (void *) shmat (shmid, NULL ,0);
```

Q:3

a) If the time slice is 50 ms, first iteration computation gets over in first scheduling

⇒ Total response Time = 53 ms

For ~~the~~ remaining iterations, other 9 processes ~~run~~ and current process run, and takes 53 secs

Request comes 200 ms late due to IO

⇒ Time = 530 − 200 = 330 ms

b) First, all processes run for 20 sec each, 230 time consumed. ~~Again~~, It will get repeated. once

Then, 3rd time, the process gets executed for 10 sec and requests IO

⇒ ~~2300~~ 23 × 10 + 23 × 10 + 13
= 473 ms

For other iterations, 9 × 13 ~~ti~~ ms time for other processes first. Then 20 time slices for 10 processes (this completes 40 ms of IO for each process)

23~~0~~ × 20 ms

13 ms for the 3rd time for our process

200 time was taken by the first IO

⇒ 9 × 13 + 23 × 10 + 23 × 10 + 13 − 200
= 390 ms