Name : Sumit kumar Yadav   Roll No . 18CS30042

① (a) Not priviliged mode :-

To enter priviledged mode from non privileged user app, we need to CPU in privileged mode for some operation like I/O.

(b) Priviliged mode :- Bound register are typical contents of PCB that stores size of instruction which must be protected for normal user.

(c) Not priviliged Mode :-

CPU register are free to use for every user program so no need to put in priviliged mode.

But some register like program counter are need to be put in priviliged mode

(d) Priviliged mode :-

~~Dis~~ Disabling all interrupt requries when some high priority program is running.

(e) Priviliged mode :-

~~[illegible]~~ Read the status of an external devices after some time later pooling devices use system timers to have Kernal call routine.

Ans2 :- "If your program contains many requests for memory. you can speed up its execution by combining all these requests into a single system call". This is because when a software interrupt occurs at system call occurs, then the interrupt hardware transfer the control to the kernel and after processing the interrupt the kernel gives control to an user programme.

both of these action cause switching between programs. Thus to reduce this overhead we make a single request to obtain a large chunk of memory instead of making several requests for small areas of memory.

Ans 3:-

(a) Since the degree of multiprogramming does not change, there may be some periods of time when CPU has no work. hence throughput may be limited by the availability of memory and thus almost double the speed.

(b) Due to expansion of main memory the m also increases, but the throughput would be limited by the speed of CPU hence throughput may not double. (it is same as earlier)

(c) little increase in throughput because involvement of DMA will give free time for CPU, but it will not ~~~~ double because DMA is not as fast as CPU

d) Yes, it should double as it does not have the drawbacks of proposals (a) & (b).

Ans 4:- No. of processes $= 2^n$

$= 2^4 \quad (n = 4)$

$= 16$

i.e, if current no. of process is a then after fork(); function call the no. of processes becomes 2a.

Ans 5:-

Syscall

→ Syscall are made by the program and syscall return control to program

→ Syscall are synchronous, program-initiated control transfer from user to the OS to obtain service from the OS

ie: syscall

Exception

→ Exception raised by the kernel and are unintentional and don't return to the user program

→ Exceptions are synchronous, program-initiated control transfer from user to the OS in response to an exceptional event

eg: Page fault, divide by zero

Ans 6:-

lets define c, cost as the sum of cost of service time and cost of writing time per user

ie; $$C = \frac{S}{TN} + \frac{WN}{M}$$

Now, to find the optimal batch size that minimizes the total cost, c

i.e; $$\frac{d(C)}{dN} = 0$$

$$\Rightarrow \frac{d}{dN}\left(\frac{S}{TN} + \frac{WN}{M}\right) = 0$$

$$\Rightarrow -\frac{S}{TN^2} + \frac{W}{M} = 0 \Rightarrow \frac{S}{TN^2} = \frac{W}{M}$$

$$\Rightarrow N^2 = \frac{SM}{WT} \Rightarrow \boxed{N = \sqrt{\frac{M}{T} \cdot \frac{S}{W}}}$$

(b) ∵ $M = 5$ min, $T = 1$ min, $S = \$ 200/\text{hour}$, $N = 50$

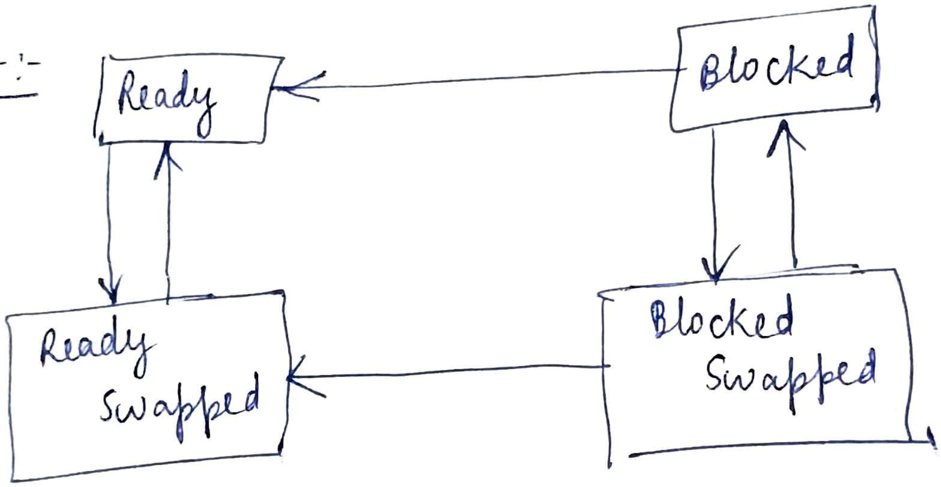
$$\therefore W = \frac{MS}{TN^2}$$

$$= \frac{\cancel{5} \times 200}{1 \times \cancel{50} \times 50} \ \$/\text{hour}$$

$$= \frac{20}{50} \ \$/\text{hour}$$

$$= \frac{2}{5} = 0.4 \ \$/\text{hour}.$$

Ans 7 :-



Blocked → Blocked Swapped : In blocked state, due to lack
(swap out)
of memory a lower priority process is sent to blocked swapped.

Blocked swapped → Blocked : In blocked state, when there is
(swap in)
enough memory a or when a much higher priority process is there in blocked swapped state.

Blocked → Ready : whenever input/output or any event wait completed or resources granted.

Block swapped → Ready Swapped : whenever input/output or event wait completed or resources granted.

Ready → Ready Swapped : In Ready state, due to lack of memory a lower priority process is sent to ready swapped.

Ready Swapped → Ready : whenever ready state has enough memory to incorporated a process of the ready swapped State.

Ans 8 :- Typical entries in the PCB :-
Process state, Program counter, CPU scheduling information, memory - management information, accounting information, I/O status information.

- Context switching imposed overhead due to its time requirements.
∴ The overhead can be reduced by migrating kernel services such as scheduling, time tick processing and interrupt handling to hardware.

2 design approaches :-
→ by increasing main memory, ensure minimized transition to Ready-swapped state & blocked-swapped state as sufficient memory is there for multiple programs to stays in Ready & Blocked State
→ Reduce overhead by ~~mody~~ modifying scheduling pointer queues in PCB by ensuring high priority processes are scheduled more frequently & earlier.

Ans 9 :- CPU-bound :- A task that performs calculations on a small set of number eg, multiplying smalle matrices

I/O bound :- A task that processes data from disk eg; counting the ~~numb~~ number of lines in a file.

Ans 10 :-

Short term scheduler	Medium term Scheduler	Long term Scheduler
→ It is a CPU scheduler	→ It is swapping based	→ It is a Job scheduler
→ make a decision about which process to be executed next and then it call the dispatcher	→ Moving the process from main memory to secondary & vice-versa	→ Makes a decision about how many processes should be made to stay in ready state
→ Minimal in a time-sharing system	→ Time-sharing system uses medium-term scheduler	→ Absent or minimal in a time sharing system
→ less control over the degree of multiprogramming	→ Reduce the degree of multiprogramming	→ It controls the degree of multiprogramming

Ans 11 :-

```
# include <stdio.h>
# include < sys | ipc.h>
# includ < sys | shm.h>
int main ( )
{
    int shmid, status ;
    int  *a , *b ;
    int  i ;
    sh mid = shmget (IPC - PRIVATE , 2* sizeof (int), 0777 | IPC-CREAT);
    if ( forK( ) = = 0)
    {
        b = (int *)
        for (i = 0; i < 10; i ++)
        {
            sleep (1)
            printf (" child 1 Reads ", %.d %.d", b[0], b[1]);
        }
        shmdt (d);
    }
    else { if (forkC) = = 0) { int *c = (int *) shmat(shmid,0,0)
        for (i=0; i <10; i++)
        { sleep (1);
            printf(" child 2 Loads " c[0], c[1]);
        }
        shmdt(c);
    }   a = (int *) shmat ( shmid, 0, 0);
    a[0] = 0;
    a[1] = 1;
    for (i=0; i<10; i++){ sleep(1); a[0] = a[0] + a[1];
        a[1] += a[0] }  wait (&status); shmdt(a);
        shmdt ( shmid, IPC_RMID, 0); } } }
```

Ans12 :-

~~The reason why the instruct~~

The reasons is that we need to keep the context switch time as short as possible, so keeping the part of the OS that dols with context switches always in the fixed location ~~of~~ in the memory rather than bring it in from disk every time. ~~or ew~~ that means the OS instructions will execute faster and the context switch time will be predictable.

Ans13 :- Data Structures :-

① Arrays :-

Pros : OS can directly access any PCB with the index number and here there is no need of traversal.

Cons : It can run out of memory if a huge amount of processes gets allocated.

② Doubly Linked List :-

Pros : It uses dynamic memory allocation that implies memory overflow ~~will~~ won't occur here.

Cons : OS can not directly access any PCB without traversing the whole linked list.

③ Circular Doubly linked list :-

Pros : OS can insert a PCB within a circular linked list ~~thus~~ instead of appending new processes which ultimately save memory.

Cons : OS can not directly access any PCB without traversing the whole linked list.