Name: Shyed Shahriar Housaini
ID: MCE 079 05536
Ans to the q.no- 1(a)

Operating system is a resource allocator and control program/application making efficient use of hardware and managing execution of user or system or application programs.

OS provides controlled allocation of the resources, processes, memories, I/O devices. OS decides how much time should be given to particular program to be executed by processor. OS also manages memory and I/O devices.

The main goal of OS is to manage, allocate and keep trae of which programs are using which resources, grant resource request, resolve conflicting requests for different user accounts and programs. So, OS can be called a resource allocator.

Ans to the q. no- 1 (b)

CPU scheduling aims to optimize the utilization of CPU. It is a process that allows system to carry out multiple processes at once. CPU keeps a process on hold while other process is beinge executed, to maximize limited resources. The scheduling ensures that all the Process in the CPU are being executed in a timely manner and system utilize~~the~~ the full capacity of limited resources, makes the system efficient and speedy - which is optimization. CPU scheduling reduces the average load into the ready queue and reduce the turnaround time for particular processes. CPU scheduling increases throughput or the number of tasks done in per unit of time and reduces the idle time of resources by switching between processes - All these are optimization problems being solved by CPU scheduling.

Ans to the q. no - 1 (c)

System call: System calls are programming interfaces provided by OS, typically written in C or C++ and they are mostly accessed ~~by~~ programs via API. Most common API are Win32, POSIX API for Unix, Linux or MacOS and Java API for JVM. Typically a number is associated with each system call and table index is maintained by Syscall interface. Sys call interface invokes the sys call in OS kernel and return status and return values. The caller need to know nothing about how the call is implemented, just need to obey the API. Sys call implementation are hidden and managed by runtime library functions in compilers.

Ans. to the q. no- 1 (d)

Tasks of system calls:-

Fork(): Unix, process control, create a process; create copies of process.

exit(): Unix, process control, Exit current process, terminate the process.

wait(): ~~The parent~~ Unix, process control, ~~the~~ parent process is on hold until the child process is completed.

Ans. to the q. no-2 (b)

The negative side of priority scheduling is – "Starvation" – low priority processes may never get executed.

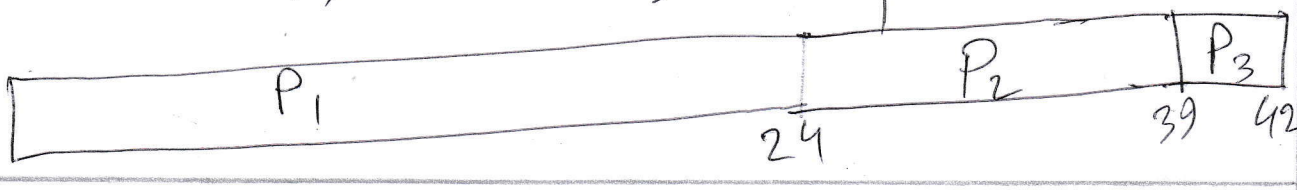
Solution for negative of priority scheduling (Starvation) is 'Aging' – as time progresses, we have to increase the priority of processes.

Ans. to the q. no-2 (a)

Convoy effect: It is short process behind long process; like one CPU bound and many I/O bound ~~or stv~~ processes.

Process	Burst Time
P_1	24
P_2	15
P_3	3

Waiting time for $P_3 = 39$



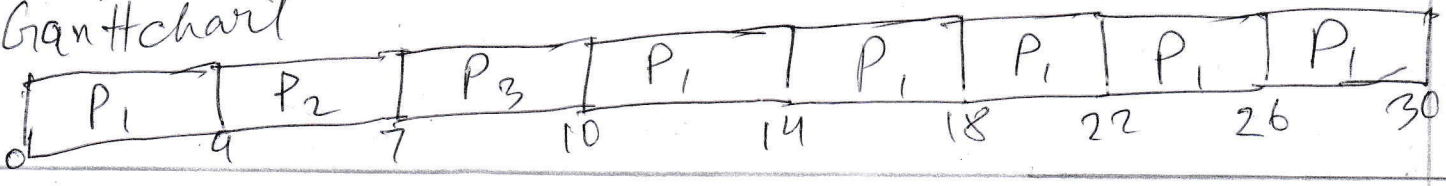
Ans to the q. no-2(d)

Dispatch Latency: Time it takes for the dispatcher to stop one process and start another running.

Time quantum in RR (Round Robin) scheduling:

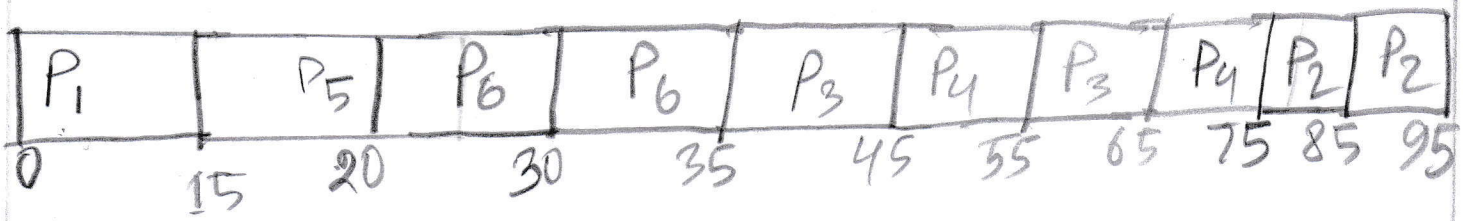
1) Each process gets a small unit of CPU time time quantum q, after this time has passed (usually 10-100 milliseconds) the process is preempted and added to the end of ready queue. for n processes and TS, q;
each process geth 1/n of the CPU and at most q time at once, max waiting time is (n-1)q. Example of RR, TS = 4
Process P_1, P_2 P_3 with time 24, 3 & 3 seconds

Ganttchart



Ans to the q. no-3

① Gantt Chart with time quantum = 10 units



② Waiting time for: $P_1 = 0$; $P_5 = 15$; $P_6 = 20$; $P_3 = 35$; $P_4 = 45$; $P_2 = 75$; (units)

③ We know that Turn around time
= Burst time + Waiting time

Turnaround time for

$P_1 = 15 + 0 = 15$ units
$P_2 = 75 + 20 = 95$ units
$P_3 = 35 + 20 = 55$ units
$P_4 = 20 + 45 = 65$ units
$P_5 = 5 + 15 = 20$ units
$P_6 = 15 + 20 = 35$ units.

Ans. to the q. no-2 (c)

ICP: It means Inter Process Communication. it is an area of memory shared among the processes that want to communicate, that communication is under the control of user processes. Have to provide mechanism that will allow the ~~us~~ user process to synchronize their actions when they access shared memory.

Bounded-Buffer - Shared memory Solution

```
#define ~~Buff~~ BUFFER_SIZE 10
    typedef struct {
        ----
    } item;
item buffer[BUFFER_SIZE];
    int in=0; int out=0;
// Producer Process; shared memory
item next_produced;
while(true){ /*produce an item in next produced*/
    while(((in+1)% BUFFER_SIZE)== out);
    /* do nothing */
buffer[in] = next_produced;
        in = (in+1)% BUFFER_SIZE}
```

```
// Consumer process - shared memory
item next_consumed;
while (true) {
    while (in == out); /* do nothing */
    next_consumed = buffer[out];
    out = (out+1) % BUFFER_SIZE;
// Consume the item next consumed }
```