BHARATIYA VIDYA BHAVAN'S

SARDAR PATEL INSTITUTE OF TECHNOLOGY

MUNSHI NAGAR, ANDHERI (WEST), MUMBAI – 400 058.

(Autonomous College Affiliated to University of Mumbai)

MASTER OF COMPUTER APPLICATIONS

ACADEMIC YEAR: 2018-19

Course: Operating System Class: FYMCA Sem: II Course Code: MCA21

Q1. Illustrate different types of system calls

System Call

(Definition=01 mark)

When a program in user mode requires access to RAM or a hardware resource, it must ask the kernel to provide access to that resource. This is done via something called a **system call.**

When a program makes a system call, the mode is switched from user mode to kernel mode. This is called a **context switch.**

Then the kernel provides the resource which the program requested. After that, another context switch happens which results in change of mode from kernel mode back to user mode.

Types: 01 mark

Explanation of each type: 01 mark

Generally, system calls are made by the user level programs in the following situations:

- File Management: Creating, opening, closing and deleting files in the file system.
- Process control: Creating and managing new processes (load, execute, end, and abort).
- Communication: Creating a connection in the network, sending and receiving packets.
- Device Management: Requesting access to a hardware device, like a mouse or a printer ,release device, read, write, get device attributes etc

Q1. Classify different types of OS
Answer: List the different types=01 marks
Functions of each type carries 01 mark(any 4)

1. Simple Batch System
2. Multiprogramming System

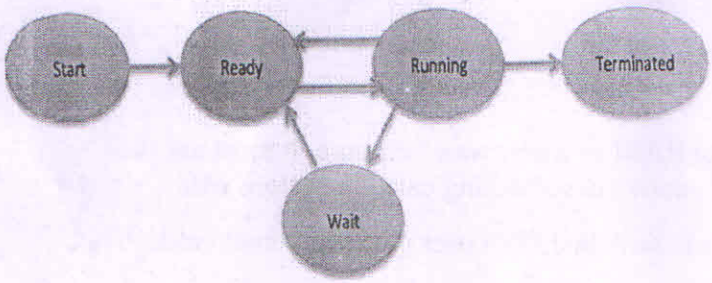
3. Multiprocessor System
4. Desktop System
5. Distributed Operating System
6. Clustered System
7. Real time Operating System

Q2. Demonstrate process state diagram

Process

A process is basically a program in execution. The execution of a process must progress in a Sequential fashion

Diagram=02 mark



Working= 03 marks

Start

This is the initial state when a process is first started/created.

Ready

The process is waiting to be assigned to a processor. Ready processes are waiting to have the processor allocated to them by the operating system so that they can run. Process may come into this state after **Start** state or while running it by but interrupted by the scheduler to assign CPU to some other process

Running

Once the process has been assigned to a processor by the OS scheduler, the process state is set to running and the processor executes its instructions.

Waiting

Process moves into the waiting state if it needs to wait for a resource, such as waiting for user input, or waiting for a file to become available.

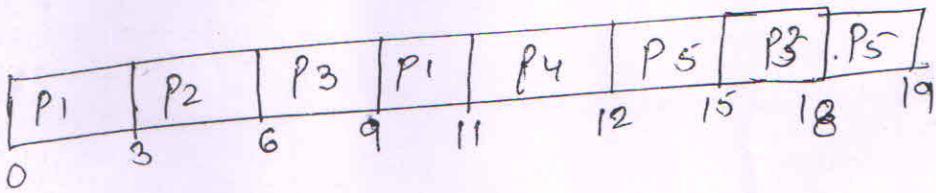
Terminated or Exit

Once the process finishes its execution, or it is terminated by the operating system, it is moved to the terminated state where it waits to be removed from main memory.

PNO	AT	BT	CT	TAT	WT
1	0	5	11	11	6
2	1	3	6	5	2
3	3	6	18	15	09
4	5	1	12	07	06
5	6	4	19	13	09

Process Queue P1 P2 P3 P1 P4 P5 P3 P5

Gantt Chart



Avg WT = (6+2+9+6+9)/5 = 6.4

Avg TAT = (11+5+15+7+13)/5 = 10.2

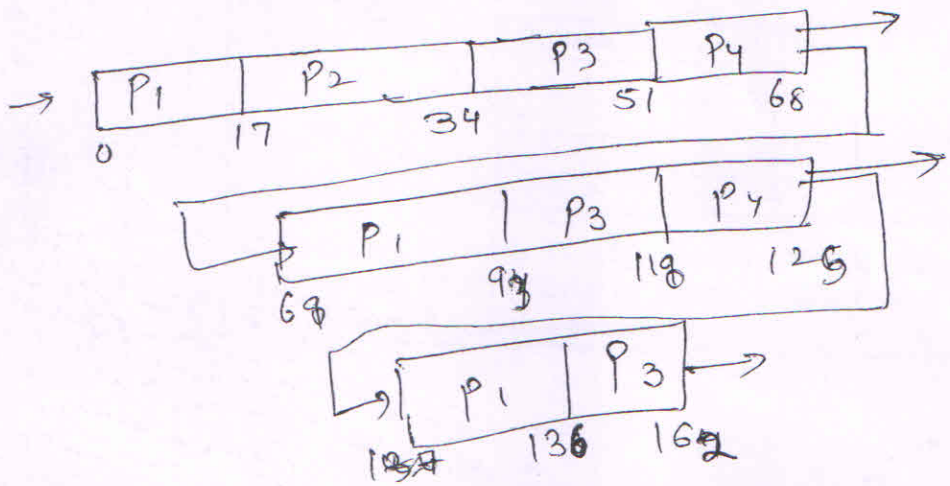
OR

Q2

Processno	BT
P1	53
2	17
3	68
4	24

Avg TAT = (138+17+130+75)/4 = 90

Avg WT = (85+17+96+102)/4 = 75



Q3.

Here, we use one **mutex** `m` and a **semaphore** `w`. An integer variable `read_count` is used to maintain the number of readers currently accessing the resource. The variable `read_count` is initialized to `0`. A value of `1` is given initially to `m` and `w`. (01 mark)

Instead of having the process to acquire lock on the shared resource, we use the mutex `m` to make the process to acquire and release lock whenever it is updating the `read_count` variable.

The code for the **writer** process looks like this: (01mark)

```
while(TRUE)
{
    wait(w);

   /* perform the write operation */

   signal(w);
}
```

And, the code for the **reader** process looks like this (02 marks)

```
while(TRUE)
{
    //acquire lock
    wait(m);
    read_count++;
    if(read_count == 1)
        wait(w);

    //release lock
    signal(m);

    /* perform the reading operation */

    // acquire lock
    wait(m);
    read_count--;
    if(read_count == 0)
        signal(w);

    // release lock
    signal(m);
}
```

Code Explanation =01 mark