OPERATING SYSTEMS

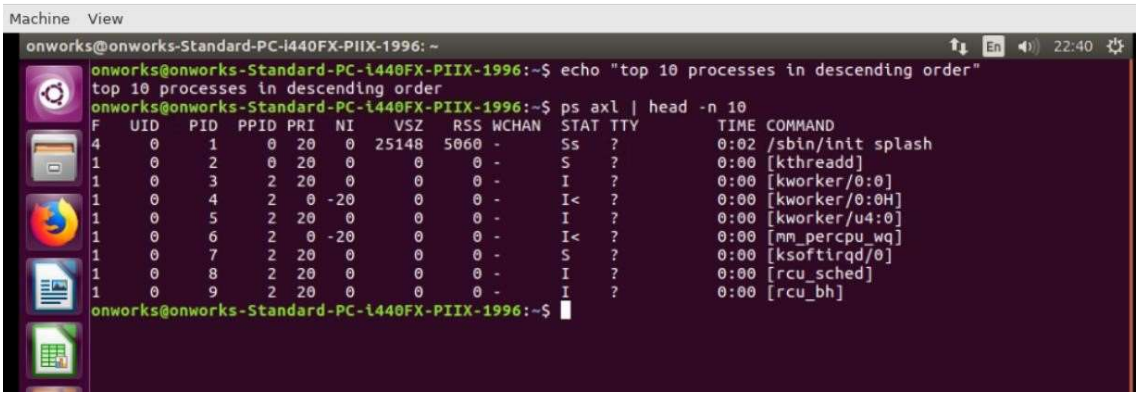
EXPERIMENT – 1

Ayush Jain | 60004200132 | SE – B2

AIM: Explore the internal commands of linux and Write shell scripts to do the following:

1. **Display top 10 processes in descending order**

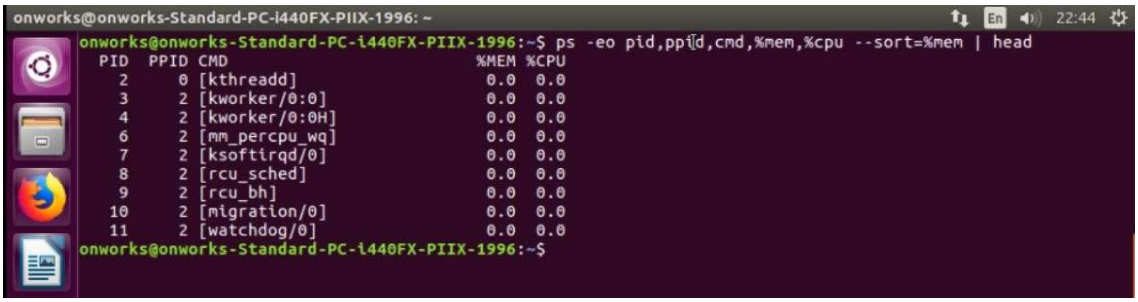
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~$ echo "top 10 processes in descending order" top 10 processes in descending order onworks@onworks-Standard-PC-i440FX-PIIX-1996:~$ ps axl | head -n 10



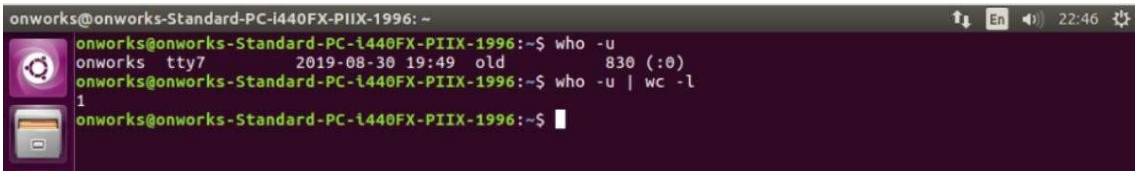
ps

2. **Display processes with highest memory usage.**

onworks@onworks-Standard-PC-i440FX-PIIX-1996:~$ ps -eo pid,ppid,cmd,%mem,%cpu --sort=%mem |head



Display current logged in user and no. of users onworks@onworks-Standard-PC-i440FX-PIIX-1996:~$ who –u onworks@onworks-Standard-PC-i440FX-PIIX-1996:~$ who -u | wc -l



Display current shell, home directory, operating system type, current working directory.

onworks@onworks-Standard-PC-i440FX-PIIX-1996:~$ whoami

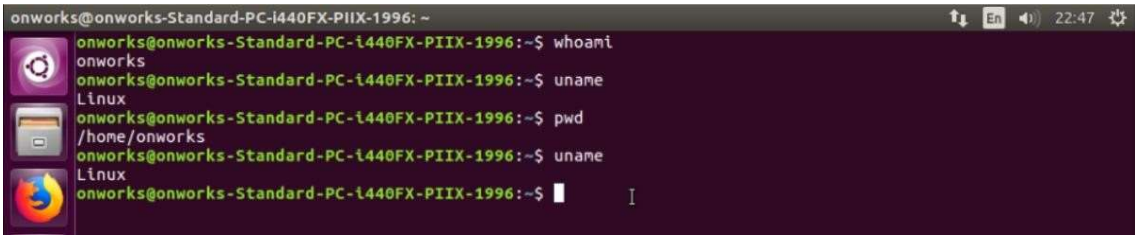
onworks onworks@onworks-Standard-PC-i440FX-PIIX-

1996:~$ uname Linux onworks@onworks-Standard-PC-

i440FX-PIIX-1996:~$ pwd

/home/onworks onworks@onworks-Standard-PC-i440FX-PIIX-

1996:~$ uname Linux



Display OS version, release number. onworks@onworks-Standard-PC-i440FX-

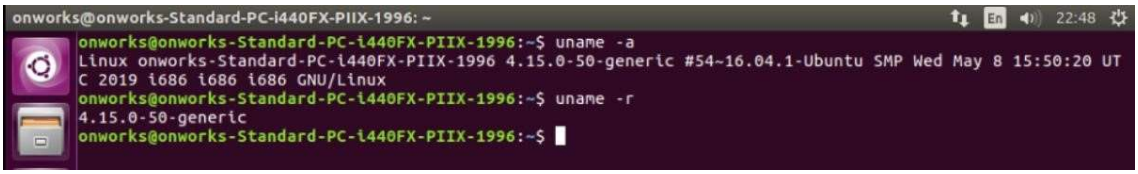
PIIX-1996:~$ uname -a

Linux onworks-Standard-PC-i440FX-PIIX-1996 4.15.0-50-generic

#54~16.04.1Ubuntu SMP Wed May 8 15:50:20 UTC 2019 i686 i686 i686

GNU/Linux onworks@onworks-Standard-PC-i440FX-PIIX-1996:~$ uname -r

4.15.0-50-generic



Illustrate the use of sort, grep, awk, etc onworks@onworks-

Standard-PC-i440FX-PIIX-1996:~$ cat > abc orage kiwi grapes

mangoes

onworks@onworks-Standard-PC-i440FX-PIIX-1996:~$ ls abc

Documents examples.desktop Pictures Templates

Desktop Downloads Music Public Videos

onworks@onworks-Standard-PC-i440FX-PIIX-1996:~$ sort abc

grapes kiwi mangoes orage

onworks@onworks-Standard-PC-i440FX-PIIX-1996:~$ sort abc>lmn.txt

onworks@onworks-Standard-PC-i440FX-PIIX-1996:~$ ls abc

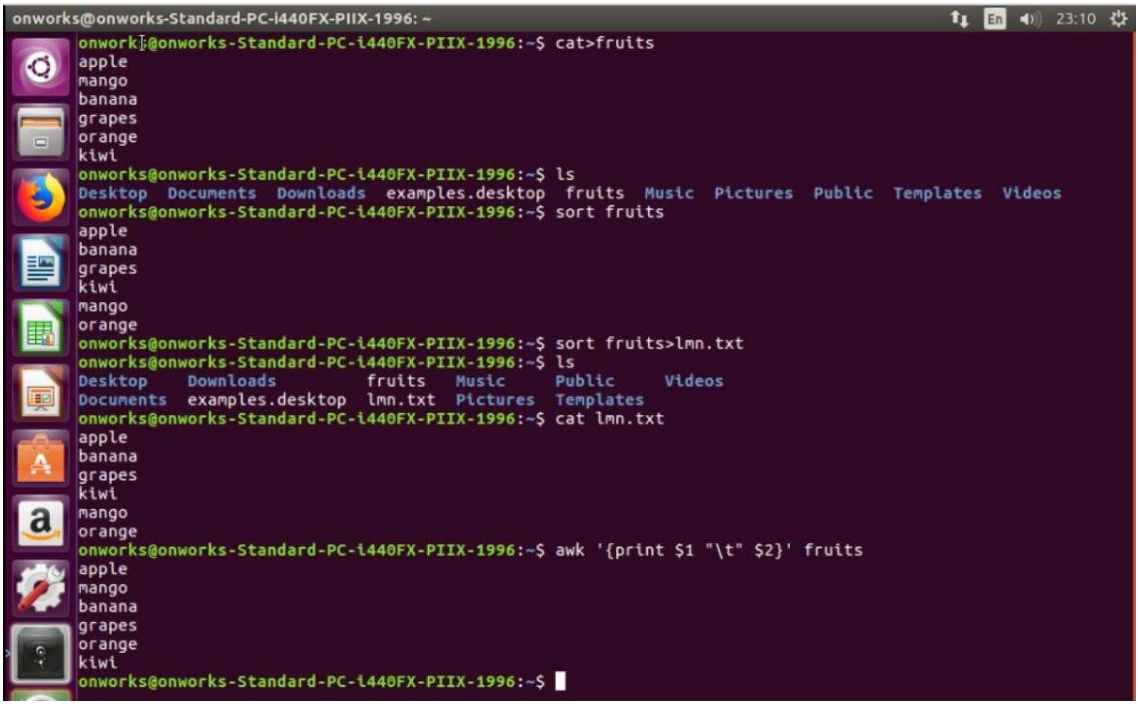
Documents examples.desktop Music Public Videos

Desktop Downloads lmn.txt Pictures Templates

onworks@onworks-Standard-PC-i440FX-PIIX-1996:~$ cat lmn.txt

grapes kiwi mangoes orage

onworks@onworks-Standard-PC-i440FX-PIIX-1996:~$ awk '{print $1 "\t" $2}' abc
orage kiwi grapes
mangoes



OPERATING SYSTEMS

EXPERIMENT – 2

Ayush Jain| 60004200132 | SE – B2

AIM: System calls for file manipulation

PROBLEM STATEMENT:
Try different file manipulation operations provided by linux

1. **pwd Command**

pwd, short for the print working directory, is a command that prints out the current working directory in a hierarchical order, beginning with the topmost root directory (/).
To check your current working directory, simply invoke the pwd command as shown.
$ pwd

2. **mkdir Command**

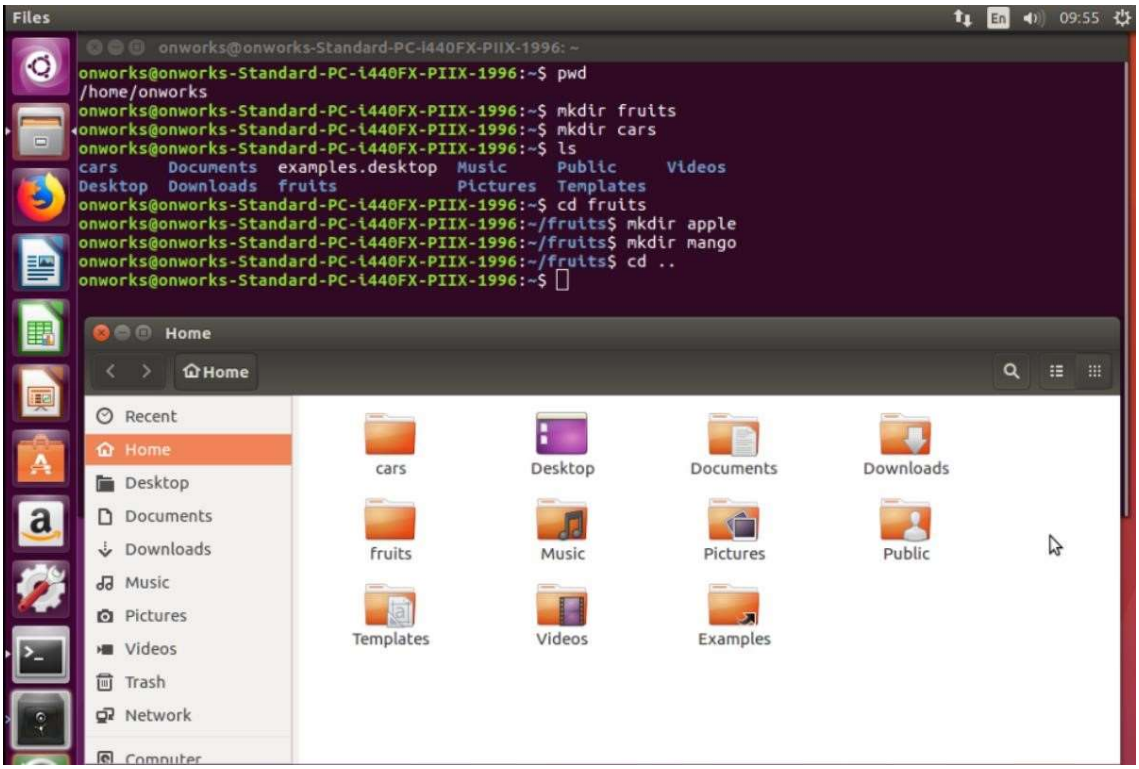
You might have wondered how we created the tutorials directory. Well, it's pretty simple. To create a new directory use the mkdir (make directory) command as follows:
$ mkdir directory_name

3. **ls Command**

The ls command is a command used for listing existing files or folders in a directory. For example, to list all the contents in the home directory, we will run the command.
$ ls

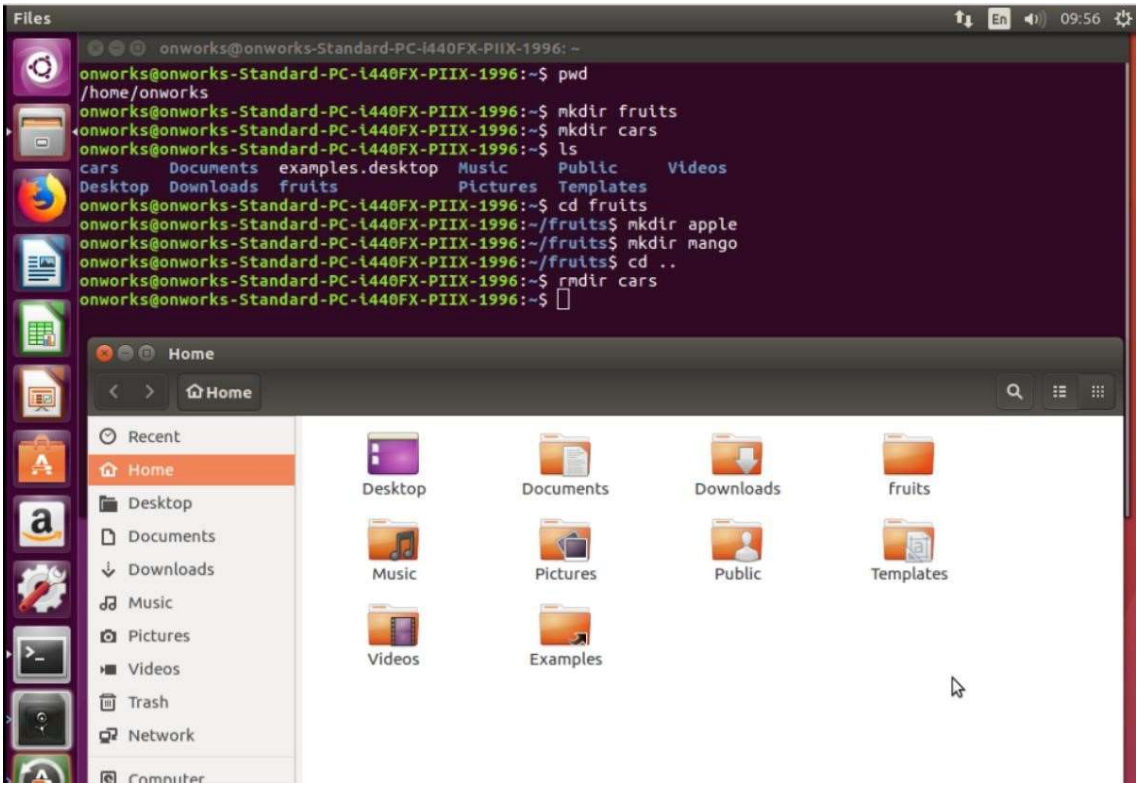
4. **cd Command**

To change or navigate directories, use the cd command which is short for change directory.
For instance, to navigate to particular directory run the command:
$ cd directory_name
To go a directory up append two dots or periods in the end.
$ cd ..
To go back to the home directory run the cd command without any arguments. **$ cd**



5. rmdir Command

The rmdir command deletes an empty directory. For example, to delete or remove the tutorials directory, run the command:



$ rmdir tutorials

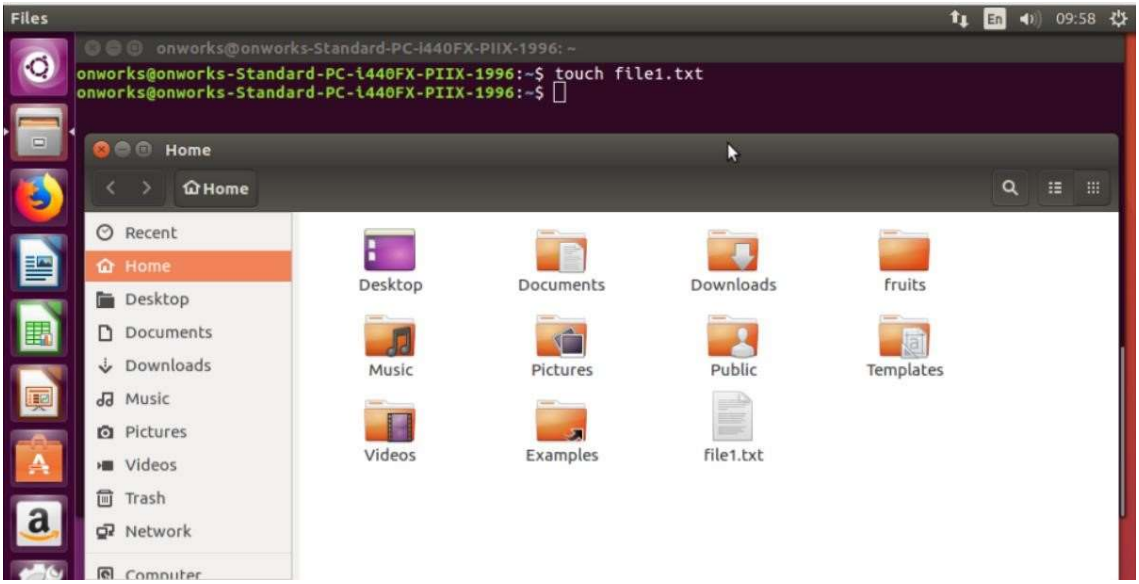
6. touch Command

The touch command is used for creating simple files on a Linux system. To create a file, use the syntax:

$ touch filename

For example, to create a file1.txt file, run the command:

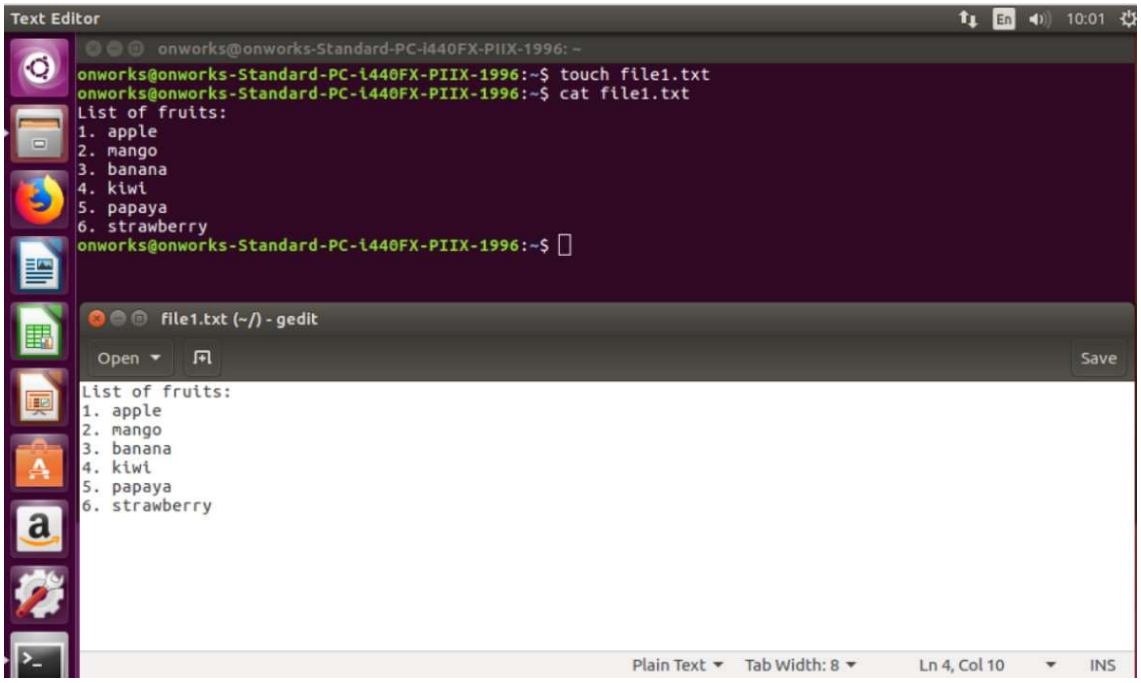
$ touch file1.txt



7. cat Command

To view the contents of a file, use the cat command as follows:

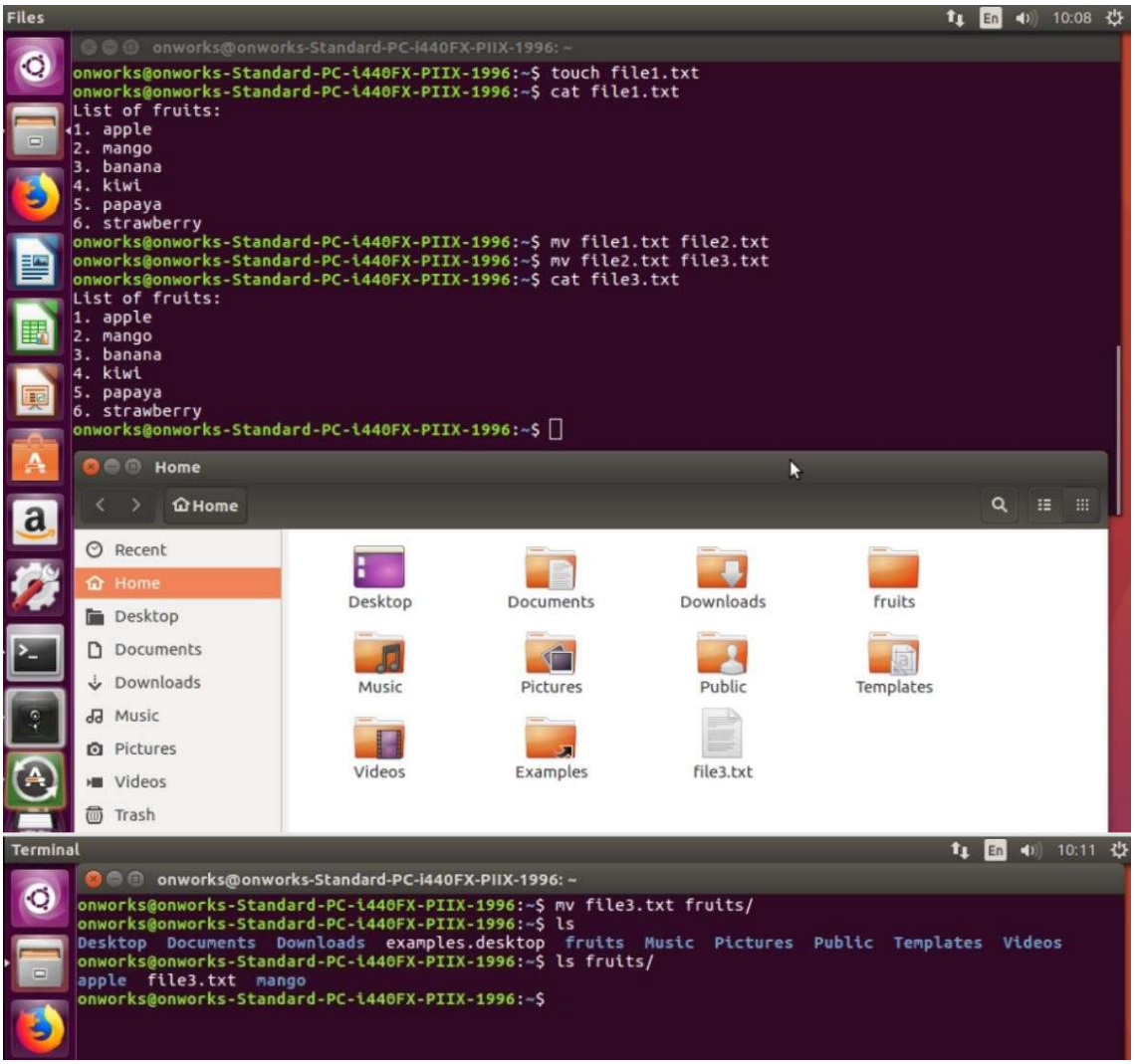
$ cat filename



8. mv Command

The mv command is quite a versatile command. Depending on how it is used, it can rename a file or move it from one location to another. To move the file, use the syntax below:

$ mv filename /path/to/destination/

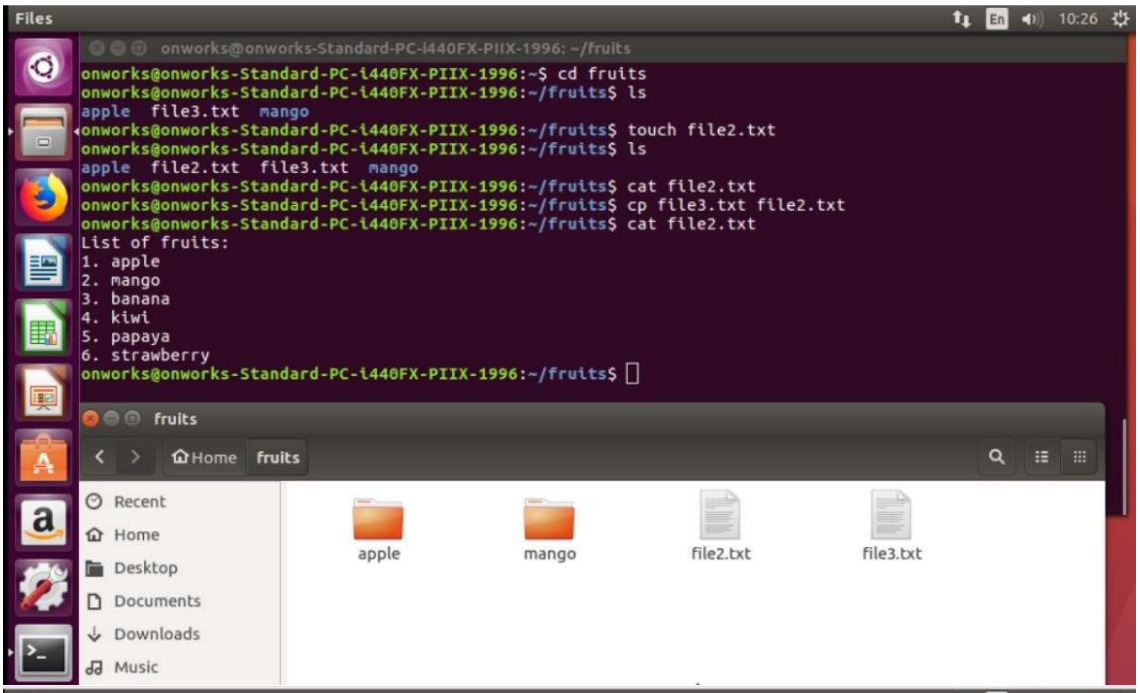


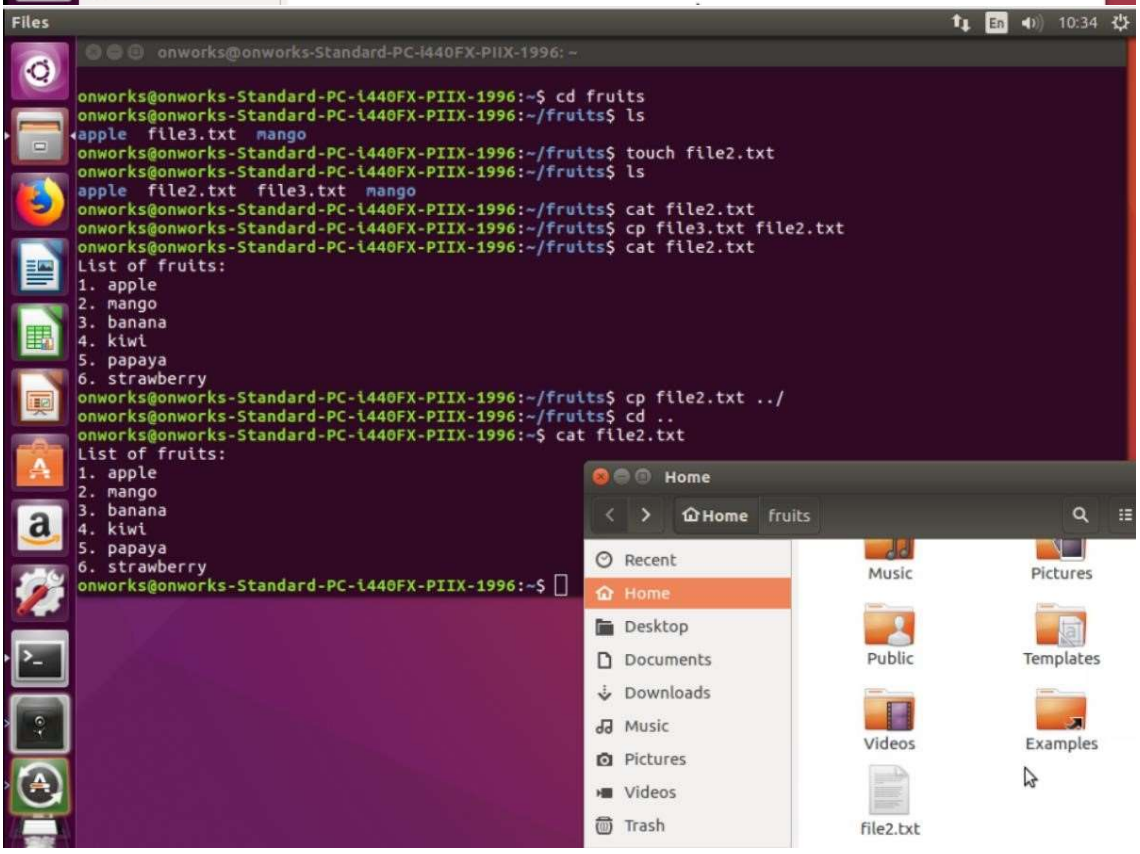
9. cp Command

The cp command, short for copy, copies a file from one file location to another. Unlike the move command, the cp command retains the original file in its current location and makes a duplicate copy in a different directory.

The syntax for copying a file is shown below.

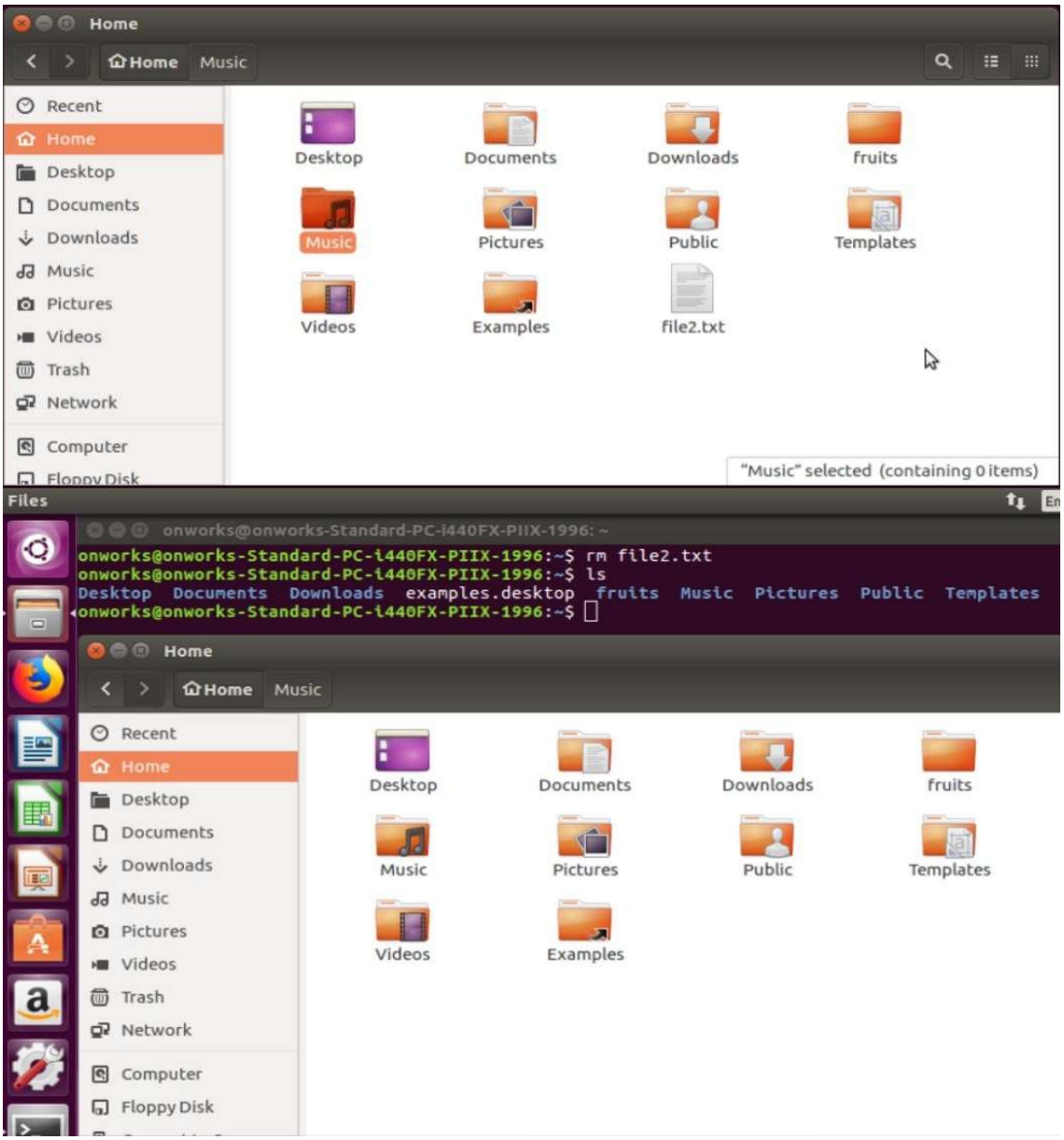
$ cp /file/path /destination/path





10. Deleting a File

rm command could be used to delete a file. It will remove the filename file from the directory.



$rm filename
Also try the following commands

Directory and file commands

cd /home	enter to directory '/ home' [man]
# cd ..	go back one level [man]
# cd ../..	go back two levels [man]
# cd	go to home directory [man]
# cd ~user1	go to home directory [man]

# cd -	go to previous directory [man]

# cp file1 file2	copying a file [man]
# cp dir/* .	copy all files of a directory within the current work directory [man]
# cp -a /tmp/dir1 .	copy a directory within the current work directory [man]
# cp -a dir1 dir2	copy a directory [man]
# cp file file1	outputs the mime type of the file as text [man]
# iconv –l	lists known encodings [man]
# iconv -f fromEncoding -t toEncoding inputFile > outputFile	converting the coding of characters from one format to another [man]
# find . -maxdepth 1 -name *.jpg print -exec convert	batch resize files in the current directory and send them to a thumbnails directory (requires convert from Imagemagick) [man]
# ln -s file1 lnk1	create a symbolic link to file or directory [man]
# ln file1 lnk1	create a physical link to file or directory [man]
# ls	view files of directory [man]
# ls –F	view files of directory [man]
# ls –l	show details of files and directory [man]
# ls –a	show hidden files [man]
# ls *[0-9]*	show files and directory containing numbers [man]
# lstree	show files and directories in a tree starting from root(2) [man]
# mkdir dir1	create a directory called 'dir1' [man]
# mkdir dir1 dir2	create two directories simultaneously [man]
# mkdir -p /tmp/dir1/dir2	create a directory tree [man]

# mv dir1 new_dir	rename / move a file or directory [man]
# pwd	show the path of work directory [man]
# rm -f file1	delete file called 'file1' [man]
# rm -rf dir1	remove a directory called 'dir1' and contents recursively [man]
# rm -rf dir1 dir2	remove two directories and their contents recursively [man]
# rmdir dir1	delete directory called 'dir1' [man]
# touch -t 0712250000 file1	modify timestamp of a file or directory - (YYMMDDhhmm) [man]
# tree	show files and direct

OPERATING SYSTEMS

EXPERIMENT – 3

Ayush Jain| 60004200132 | SE – B2

AIM- Building multi-threaded and multi-process applications.

PROBLEM STATEMENT:

1. Build an instance of bus ticket reservation system using multithreading for the following scenario.
   ABC Bus service has only two seats left for reservations. Two users are trying to book the ticket at the same time.
2. Create separate thread per user. Show how this code is leading to inconsistency.
3. Improve the code by applying synchronization.
4. Conclude the experiment by stating the importance of synchronization in multiprocess and multithread application.

THEORY:

- **Multi-threading:** Multi-threading is a process of executing multiple threads simultaneously. A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking. However, we use multithreading than multiprocessing because threads use a shared memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.
- **Runnable:** Java runnable is an interface used to execute code on a concurrent thread. It is an interface which is implemented by any class if we want that the instances of that class should be executed by a thread. The runnable interface has an undefined method run () with void as return type, and it takes in no arguments.
- **Thread:** Thread class provide constructors and methods to create and perform operations on a thread. Thread class extends Object class and implements Runnable interface. Commonly used Constructors of Thread class: Thread (), Thread (String name), Thread (Runnable r), Thread (Runnable r, String name).

Thread class

CODE:

- **Using multi-threading (No Synchronization):**

```
class Main{

    public static void main(String[] args){ Reservation

        reserve = new Reservation(); Person thread1 = new

        Person(reserve, 2);

        thread1.start();

        Person thread2 = new Person(reserve, 2);

        thread2.start();

    }
}

class Reservation{

    static int availableSeats = 2; void

    reserveSeat(int requestedSeats) {

        System.out.println(Thread.currentThread().getName() + " entered.");

        System.out.println("Availableseats : " + availableSeats + "\n
Requestedseats : " + requestedSeats);

        if (availableSeats >= requestedSeats){ System.out.println("Seat

            Available. Reserve now :-)"); try{

                Thread.sleep(100);


            }

            catch (InterruptedException e){

                System.out.println("Thread interrupted");

            }
```

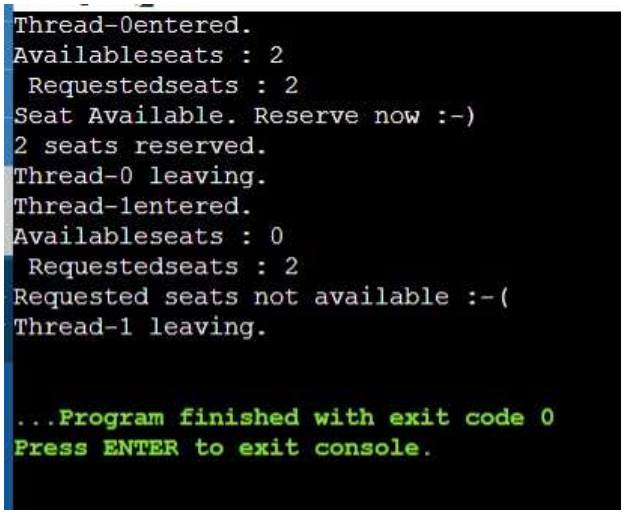
```
            System.out.println(requestedSeats + " seats reserved.");
            availableSeats = availableSeats - requestedSeats;
        }
        else{
            System.out.println("Requested seats not available :-(");
        }
        System.out.println(Thread.currentThread().getName() + " leaving.");
    }
}
class Person extends Thread{
    Reservation reserve;
    int requestedSeats;
    public Person(Reservation reserve, int requestedSeats){ this.reserve
        = reserve;
        this.requestedSeats = requestedSeats;
    }
    public void run() { reserve.reserveSeat(requestedSeats);
    }
}
```

```
Thread-1entered.
Thread-0entered.
Availableseats : 2
 Requestedseats : 2
Seat Available. Reserve now :-)
Availableseats : 2
 Requestedseats : 2
Seat Available. Reserve now :-)
2 seats reserved.
2 seats reserved.
Thread-0 leaving.
Thread-1 leaving.


...Program finished with exit code 0
Press ENTER to exit console.
```

Using multi-threading (With Synchronization):

```
class Main{

    public static void main(String[] args){ Reservation

        reserve = new Reservation(); Person thread1 =

        new Person(reserve, 2); thread1.start(); Person

        thread2 = new Person(reserve, 2);

        thread2.start();

    }

}

class Reservation{

    static int availableSeats = 2; synchronized void

    reserveSeat(int requestedSeats) {

        System.out.println(Thread.currentThread().getName() + " entered.");

        System.out.println("Availableseats : " + availableSeats + "\n
Requestedseats : " + requestedSeats);

        if (availableSeats >= requestedSeats){ System.out.println("Seat

            Available. Reserve now :-)"); try{

                Thread.sleep(100);


            }

            catch (InterruptedException e){

                System.out.println("Thread interrupted");

            }
            System.out.println(requestedSeats + " seats reserved.");

                availableSeats = availableSeats - requestedSeats;

        }
```

```
            else{
                System.out.println("Requested seats not available :-(");
            }
            System.out.println(Thread.currentThread().getName() + " leaving.");
        }
    }
    class Person extends Thread{
        Reservation reserve;
        int requestedSeats;
        public Person(Reservation reserve, int requestedSeats){ this.reserve
            = reserve;
            this.requestedSeats = requestedSeats;
        }
        public void run() { reserve.reserveSeat(requestedSeats);
        }
    }
```



Operating Systems:

Experiment 4

Name: Ayush Jain

SAP ID: 60004200132

Batch: B2

Computer Engineering

Aim: CPU scheduling algorithms like FCFS, SJF, Round Robin etc.

Problem Statement:

1) Perform comparative assessment of various Scheduling Policies like FCFS, SJF (preemptive and non-preemptive), Priority (preemptive and non-preemptive) and Round Robin.
2) Take the input processes, their arrival time, burst time, priority, quantum from user.

Description: Scheduling algorithms are used when more than one process is executable and the OS has to decide which one to run first.

Terms Used:

1) **Submit Time:** The process at which the process is given to CPU
2) **Burst Time:** The amount of time each process takes for execution
3) **Response Time:** The difference between the time when the process starts execution and the submit time.
4) **Turnaround Time:** The difference between the time when the process completes execution and the submit time.

First Come First Serve (FCFS)

The processes are executed in the order in which they have been submitted.

Shortest Job First (SJF)

The processes are checked at each arrival time and the process which have the shortest remaining burst time at that moment gets executed first. This is non-preemptive algorithm.

Priority Scheduling

Each process is assigned a priority and executable process with highest priority is allowed to run

Round Robin

Each process is assigned a time interval called its quantum (time slice)
If the process is still running at the end of the quantum the CPU is preempted and given to another process, and this continues in circular fashion, till all the processes are completely executed.

Code and Output:

1) **First Come First Serve (FCFS)**

```
import java.util.Scanner;
class FCFS
{
	public static void main(String args[])
	{
		Scanner sc = new Scanner(System.in);
		int n,i,j,temp,t1=0,t2=0,tat,wt;
		float atat=0,awt=0;
		System.out.print("Enter number of processes:
");		n=sc.nextInt();			int a[] = new int[n];
	int b[] = new int[n];		int c[] = new int[n];
int t[] = new int[n];		int w[] = new int[n];
		for(i=0; i<n; i++)
	{
			System.out.print("Enter arrival time of process "+(i+1)+": ");
			a[i]=sc.nextInt();
			System.out.print("Enter burst time of process "+(i+1)+": ");
			b[i]=sc.nextInt();
		}
		for(i=0; i<n-1; i++)
	{
			for(j=0; j<n-i-1; j++)
		{
				if(a[j+1]<a[j])
			{
					temp=a[j];
			a[j]=a[j+1];
	a[j+1]=temp;
temp=b[j];				b[j]=b[j+1];
```

```
b[j+1]=temp;                                }
}       }
                System.out.println("Process\t\tStart Time\tFinish Time");
                t1=a[0]; for(i=0;
i<n; i++)               {
                        t2=b[i]+t2;
                        System.out.println("  P"+(i+1)+"\t\t   "+t1+"\t\t   "+t2);
                        t1=t1+b[i];
        c[i]=t2;
                        t[i]=t2-a[i];
                        w[i]=t[i]-b[i];
        }
                System.out.println("Process\tA.T\tB.T\tC.T\tT.A.T\tW.T");
                for(i=0; i<n; i++)
                        System.out.println("
P"+(i+1)+"\t"+a[i]+"\t"+b[i]+"\t"+c[i]+"\t"+t[i]+"\t"+w[i]);
                for(i=0; i<n; i++)
                {
                        atat=atat+t[i];
                        awt=awt+w[i];
        }
                atat=atat/n;
                awt=awt/n;
                System.out.println("Average Turnaround Time: "+atat);
                System.out.println("Average Waiting Time: "+awt);
        }
}
```

```
Enter number of processes: 5
Enter arrival time of process 1: 0
Enter burst time of process 1: 4
Enter arrival time of process 2: 1
Enter burst time of process 2: 3
Enter arrival time of process 3: 2
Enter burst time of process 3: 1
Enter arrival time of process 4: 3
Enter burst time of process 4: 2
Enter arrival time of process 5: 4
Enter burst time of process 5: 5
Process         Start Time      Finish Time
  P1              0               4
  P2              4               7
  P3              7               8
  P4              8               10
  P5              10              15
Process A.T     B.T     C.T     T.A.T   W.T
  P1    0       4       4       4       0
  P2    1       3       7       6       3
  P3    2       1       8       6       5
  P4    3       2       10      7       5
  P5    4       5       15      11      6
Average Turnaround Time: 6.8
Average Waiting Time: 3.8
```

3) **Shortest Job First (SJF)**

```
import java.util.Scanner;
class ShortJobFirst
{
	public static void main(String args[])
	{
		Scanner sc = new Scanner(System.in);
		int n;
		System.out.print("Enter number of processes: ");
		n = sc.nextInt();
		int p[] = new int[n];    //Process id array		int
at[] = new int[n]; //Arrival time array		int bt[] = new
int[n]; //Burst time array		int flag[] = new int[n]; //To check
if process is completed
		int i, j, min1, min2, min3;
		for(i = 0; i < n; i++)
	{
			p[i] = i+1;
			System.out.print("Enter arrival time: ");
			at[i] = sc.nextInt();
  System.out.print("Enter burst time: ");    bt[i] = sc.nextInt();
			flag[i] = 0;
	}
		for(i = 0; i < n; i++)
	{
			min1 = at[i];
		min2 = bt[i];
	min3 = p[i];
j = i-1;
			while(j >= 0 && at[j] > min1)
	{
				at[j+1] = at[j];
			bt[j+1] = bt[j];
		p[j+1] = p[j];
				j--;
			}
			at[j+1] = min1;
		bt[j+1] = min2;
			p[j+1] = min3;
	}
		 int cur_t = 0; //Current time
```

```
                int st[] = new int[n];    //Starting time of each process int
                ct[] = new int[n]; //completion time array
                int tot = 0;       //To count number of processes completed int
                minbt = 1000; //To store the shortest bt and set to highest
value so that all bt values are compared
                int c = 0;         //To track id of process to be scheduled
next            while(tot < n)             {
                        for(i = 0; i < n; i++)
                {
                                if((at[i] <= cur_t) && (flag[i] == 0) && (bt[i] <= minbt))
                                {
                                        minbt = bt[i];
                                        c = i;
                        }
                        }
                        ct[c] = cur_t + minbt;
                st[c] = cur_t;
flag[c] = 1;                    cur_t = ct[c];
                        tot++;
  minbt = 1000; //reset so that bt values of remaining processes are compared   }
                int  tat[] = new int[n]; //Turnaround Time array
        int wt[] = new int[n]; //Waiting Time array
                float sum1 = 0, sum2 = 0; //sum1 for tat and sum2 for wt
System.out.println("\nProcess No.\tA.T\tB.T.\tC.T.\tT.A.T.\tW.T.");
for(i = 0; i < n; i++)                {
                        tat[i] = ct[i] - at[i];
        wt[i] = tat[i] - bt[i];
                        System.out.println("Process
"+p[i]+"\t"+at[i]+"\t"+bt[i]+"\t"+ct[i]+"\t"+tat[i]+"\t"+wt[i]);
                        sum1 += tat[i];
                        sum2 += wt[i];
        }
                System.out.println("Average Turn Around Time: "+(sum1/n));
                System.out.println("Average Waiting Time: "+(sum2/n));
                //To prepare Gantt Chart processes are soted according to start
time            for(i = 0; i < n; i++)                {
                        min1 = st[i];
                        min2 = ct[i];
                        min3 = p[i];
                j = i-1;
                        while(j >= 0 && st[j] > min1)
                        {
```

```
                    st[j+1] = st[j];
                ct[j+1] = ct[j];
                    p[j+1] = p[j];
                    j--;
                }
                st[j+1] = min1;
            ct[j+1] = min2;
                p[j+1] = min3;
        }
 System.out.println("\n\t\tGANTT CHART\nPROCESS \tStart Time\tCompletion Time");
            for(i = 0; i < n; i++)
        {
                System.out.println("PROCESS "+p[i]+"\t\t"+st[i]+"\t\t"+ct[i]);

            }
}
}
```

```
Enter number of processes: 5
Enter arrival time: 0
Enter burst time: 7
Enter arrival time: 2
Enter burst time: 5
Enter arrival time: 3
Enter burst time: 1
Enter arrival time: 4
Enter burst time: 2
Enter arrival time: 5
Enter burst time: 8

Process No.     A.T     B.T.    C.T.    T.A.T.  W.T.
Process 1       0       7       7       7       0
Process 2       2       5       15      13      8
Process 3       3       1       8       5       4
Process 4       4       2       10      6       4
Process 5       5       8       23      18      10
Average Turn Around Time: 9.8
Average Waiting Time: 5.2

                GANTT CHART
PROCESS         Start Time      Completion Time
PROCESS 1               0               7
PROCESS 3               7               8
PROCESS 4               8               10
PROCESS 2               10              15
PROCESS 5               15              23
```

3) Priority Scheduling

```
import java.util.Scanner;
    class Priority
    { public static void main(String args[]) {
                    Scanner sc = new Scanner(System.in); int
                    n;
                    System.out.print("Enter number of processes: "); n
                    = sc.nextInt();
                    int p[] = new int[n];                    //Process id array int
                    at[] = new int[n];              //Arrival time array int bt[]
                    = new int[n];           //Burst time array
                    int flag[] = new int[n];        //To check if process is
    completed           int priority[] = new int[n];            int i, j, min1, min2,
    min3, min4;
                    int cur_t = 0;                              //Current time
                    int st[] = new int[n];                  //Starting time of each process
                    int ct[] = new int[n];                  //completion time array
                    int tot = 0;                                //To count number of
    processes completed
                    int minpri = 1000;                           //To store the shortest bt
                    int c = 0;
                    int b[] = new int[n];
                    for(i = 0; i < n; i++)
            {
                            p[i] = i+1;
                            System.out.print("Enter arrival time: ");
                            at[i] = sc.nextInt();
                            System.out.print("Enter burst time: ");
                            bt[i] = sc.nextInt();
                            System.out.print("Enter priority [smallest number is high]: ");
                            priority[i] = sc.nextInt();
                            flag[i] = 0;
                            st[i] = -1;
                    }
                    for(i = 0; i < n; i++)
            {
                            min1 = at[i];
            min2 = bt[i];
    min3 = p[i];
                            min4 = priority[i];
                            j = i-1;
                            while(j >= 0 && at[j] > min1)
```

```
                {
                        at[j+1] = at[j];
                bt[j+1] = bt[j];
        p[j+1] = p[j];
                        priority[j+1] = priority[j];
                        j--;
                }
                at[j+1] = min1;
                bt[j+1] = min2;
                p[j+1] = min3;
                priority[j+1] = min4;
        }
        for(i = 0; i < n; i++)
                b[i] = bt[i];
        while(tot < n)
{
                for(i = 0; i < n; i++)
        {
                        if((priority[i] <= minpri) && (at[i] <= cur_t) && (flag[i]
== 0))                  {
                                minpri = priority[i];
                                c = i;
                }
                }
                if(st[c] == -1)
        {
                        st[c] = cur_t;
                ct[c] = cur_t;
        }
                b[c]--;
cur_t++;                        ct[c]++;
                for(i = 0; i < n; i++)
        {
                        if((c != i) && (st[i] != -1) && (flag[i] == 0))
                                ct[i]++;
        }
                if(b[c] == 0)
        {
                        flag[c] = 1;
                        tot++;
        }
                minpri = 1000;
```

```
                }
                int  tat = 0;                          //Turnaround Time
                int wt = 0;                                    //Waiting Time array
 float sum1 = 0, sum2 = 0, rt = 0;    //sum1 for tat and sum2 for wt and response time
                System.out.println("\nProcess
No.\tPriority\tA.T\tB.T.\tC.T.\tT.A.T.\tW.T.\tR.T.");
                for(i = 0; i < n; i++)
        {
                        tat = ct[i] - at[i];
                wt = tat - bt[i];
        rt = st[i] - at[i];
                        System.out.println("Process
"+p[i]+"\t"+priority[i]+"\t\t"+at[i]+"\t"+bt[i]+"\t"+ct[i]+"\t"+tat+"\t"+wt+"\t"+rt);
                        sum1 += tat;
                        sum2 += wt;
        }
                System.out.println("Average Turn Around Time: "+(sum1/n));
                System.out.println("Average Waiting Time: "+(sum2/n));
                //To prepare Gantt Chart processes are soted according to start
time             for(i = 0; i < n; i++)                {
                        min1 = st[i];
                min2 = ct[i];
        min3 = p[i];
j = i-1;
                        while(j >= 0 && st[j] > min1)
                {
                                st[j+1] = st[j];
                ct[j+1] = ct[j];
p[j+1] = p[j];
                                j--;
                        }
                        st[j+1] = min1;
                ct[j+1] = min2;
                        p[j+1] = min3;
        }
 System.out.println("\n\t\tGANTT CHART\nPROCESS \tStart Time\tCompletion Time");
                for(i = 0; i < n; i++)
        {
                        System.out.println("PROCESS "+p[i]+"\t\t"+st[i]+"\t\t"+ct[i]);

                }
        }
```

}

```
Enter number of processes: 7
Enter arrival time: 0
Enter burst time: 4
Enter priority [smallest number is high]: 7
Enter arrival time: 1
Enter burst time: 2
Enter priority [smallest number is high]: 6
Enter arrival time: 2
Enter burst time: 3
Enter priority [smallest number is high]: 5
Enter arrival time: 3
Enter burst time: 5
Enter priority [smallest number is high]: 2
Enter arrival time: 4
Enter burst time: 1
Enter priority [smallest number is high]: 4
Enter arrival time: 5
Enter burst time: 4
Enter priority [smallest number is high]: 1
Enter arrival time: 6
Enter burst time: 6
Enter priority [smallest number is high]: 3

Process No.     Priority        A.T     B.T.    C.T.    T.A.T.  W.T.    R.T.
Process 1       7               0       4       25      25      21      0.0
Process 2       6               1       2       22      21      19      0.0
Process 3       5               2       3       21      19      16      0.0
Process 4       2               3       5       12      9       4       0.0
Process 5       4               4       1       19      15      14      14.0
Process 6       1               5       4       9       4       0       0.0
Process 7       3               6       6       18      12      6       6.0
Average Turn Around Time: 15.0
Average Waiting Time: 11.428572
```

```
                GANTT CHART
PROCESS         Start Time      Completion Time
PROCESS 1               0               25
PROCESS 2               1               22
PROCESS 3               2               21
PROCESS 4               3               12
PROCESS 6               5               9
PROCESS 7               12              18
PROCESS 5               18              19
```

4) Round Robin

```
import java.util.Scanner;
    class Queue
    {
            Node front, rear;
    int queueSize;
            class Node
            {               int
    data;
    Node next;
            }
            Queue()         {
            front = null;
```

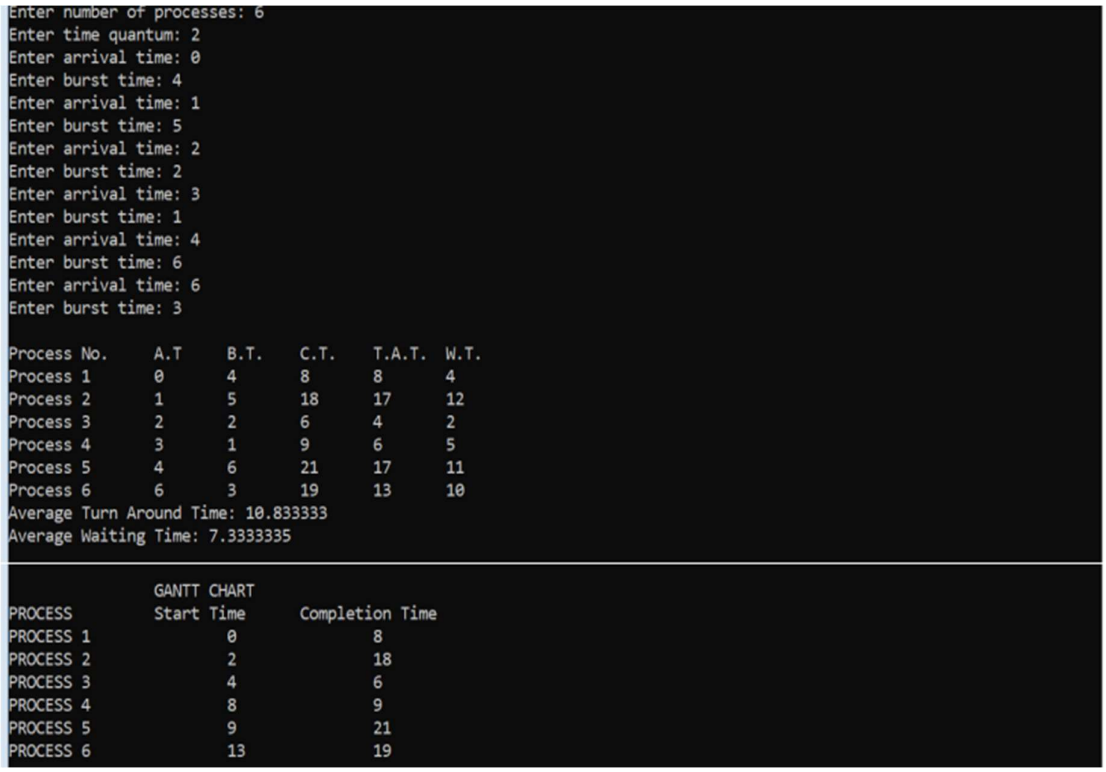
```
rear = null;
queueSize = 0;
        }
        boolean isEmpty()
        {
                return (queueSize == 0);
        }
       int dequeue()
{
 int data = front.data;    front =
front.next;
                if (isEmpty())
        {
                        rear = null;
        }
                queueSize--;
        return data;
}
        void enqueue(int data)
        {
                Node oldRear = rear;
        rear = new Node();
rear.data = data;                rear.next =
null;
                if (isEmpty())
        {
                        front = rear;
                }
        else
                {
                        oldRear.next = rear;
                }
                queueSize++;
        }
}

class RR
{
        public static void main(String args[])
        {
 Scanner sc = new Scanner(System.in);   int n, tq;
                System.out.print("Enter number of processes: ");
```

```
		n = sc.nextInt();
		System.out.print("Enter time quantum: ");
		tq = sc.nextInt();
		Queue q = new Queue();
		int p[] = new int[n];			//Process id array int
		at[] = new int[n];		//Arrival time array
		int bt[] = new int[n];			//Burst time array
		int flag[] = new int[n];		//To check if process is completed
		int f[] = new int[n];	int		//To check if process has arrived
		i, j, min1, min2, min3;
		int cur_t = 0;					//Current time
		int st[] = new int[n];			//Starting time of each process
		int ct[] = new int[n];			//completion time array
		int tot = 0;					//To count number of
processes completed
		int c = 0;					 //To track id of process to
be scheduled next
		int b[] = new int[n];			//Copy of Burst time array
		for(i = 0; i < n; i++)
		{
			p[i] = i+1;
			System.out.print("Enter arrival time: ");
			at[i] = sc.nextInt();
			System.out.print("Enter burst time: ");
			bt[i] = sc.nextInt();
		flag[i] = 0;
f[i] = 0;				st[i] = -1;
	}
		for(i = 0; i < n; i++)
	{
			min1 = at[i];
	min2 = bt[i];
min3 = p[i];
			j = i-1;
			while(j >= 0 && at[j] > min1)
	{
				at[j+1] = at[j];
			bt[j+1] = bt[j];
		p[j+1] = p[j];
				j--;
			}
			at[j+1] = min1;
```

```
                bt[j+1] = min2;
                p[j+1] = min3;
        }
        for(i = 0; i < n; i++)
                b[i] = bt[i];
        while(tot < n)
        {
                for(i = 0; i < n; i++)
                {
                        if((f[i] == 0) && (flag[i] == 0) && (at[i] <= cur_t))
                        {
                                q.enqueue(i);
                                f[i] = 1;
                        }
                }
                c = q.dequeue();
        if(st[c] == -1)
        st[c] = cur_t;                  ct[c]
= cur_t;                        if(b[c] > tq)
                {
                        ct[c] += tq;
        b[c] -= tq;
cur_t += tq;
                        for(i = 0; i < n; i++)
                {
                        if((f[i] == 0) && (flag[i] == 0) && (at[i] <= cur_t))
                        {
                                q.enqueue(i);
                        f[i] = 1;
                }
                        }
                        if(b[c] > 0)
                                q.enqueue(c);
        }
                else if(b[c] <= tq)
                {
                        ct[c] += b[c];
        cur_t += b[c];
b[c] = 0;
                        for(i = 0; i < n; i++)
                {
                        if((f[i] == 0) && (flag[i] == 0) && (at[i] <= cur_t))
```

```
                                        {
                                                q.enqueue(i);
                                                f[i] = 1;
                                        }
                                }
                        }
                        if(b[c] == 0)
                        {
                                flag[c] = 1;
                                tot++;
                        }
                }
                int  tat = 0;                                   //Turnaround Time int
                wt = 0;                                         //Waiting Time array
                float sum1 = 0, sum2 = 0;     //sum1 for tat and sum2 for wt
System.out.println("\nProcess No.\tA.T\tB.T.\tC.T.\tT.A.T.\tW.T.");   for(i = 0; i < n;
i++)   {
                        tat = ct[i] - at[i];
        wt = tat - bt[i];
System.out.println("Process
"+p[i]+"\t"+at[i]+"\t"+bt[i]+"\t"+ct[i]+"\t"+tat+"\t"+wt);
                        sum1 += tat;
                        sum2 += wt;
        }
                System.out.println("Average Turn Around Time: "+(sum1/n));
                System.out.println("Average Waiting Time: "+(sum2/n));

                //To prepare Gantt Chart processes are soted according to start time
        for(i = 0; i < n; i++)                {
                        min1 = st[i];
                min2 = ct[i];
        min3 = p[i];
j = i-1;
                        while(j >= 0 && st[j] > min1)
                {
                                st[j+1] = st[j];
                ct[j+1] = ct[j];
p[j+1] = p[j];
                                j--;
                        }
                        st[j+1] = min1;
                ct[j+1] = min2;
```

```
                        p[j+1] = min3;
        }
                System.out.println("\n\t\tGANTT CHART\nPROCESS \tStart
Time\tCompletion Time");
                for(i = 0; i < n; i++)
                {
                        System.out.println("PROCESS "+p[i]+"\t\t"+st[i]+"\t\t"+ct[i]);
                }
        }
}
```



Conclusion: Hence we have implemented various CPU scheduling algorithms (FCFS, SJF, Priority Scheduling and Round Robin).

Operating Systems

Experiment 5

Ayush Jain|60004200132|SE-B2

Aim: Write a code for First fit, Best fit, Worst fit, Next fit.

Code:

1. **First fit:**

```
#include <stdio.h>
int main()
{

int totalMem = 0;
int part[] = {200, 400, 600, 500, 300, 250}; int
i, j;
printf("\nEnter number of process to be added to main memory:"); int
n ;
scanf("%d", &n);
int mem_p[n];
int flag[6];
for(i = 0; i < n; i++)
{
printf("Enter memory to be assigned to process %d :",(i+1));
scanf("%d", &mem_p[i]);
}
for(i = 0; i < 6; i++)
flag[i] = 0; int id;
for(i = 0; i < n; i++)
{ id = -
1;
for(j = 0; j < 6; j++)
{
if((flag[j] == 0) && (mem_p[i] <= part[j]))
{ id =
j;
break;
}
} if(id != -
1)
```

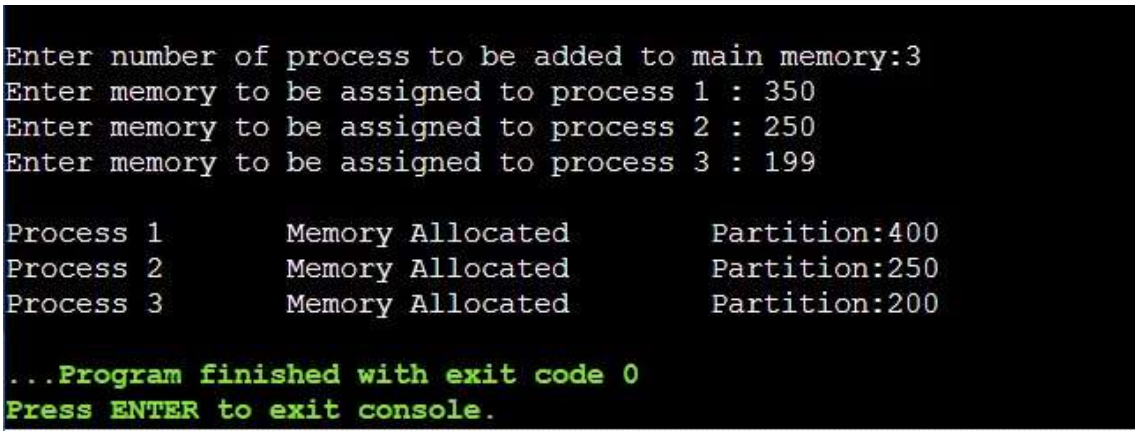
```
{
printf("\nProcess %d\tMemory Allocated\tPartition:%d ",(i+1),part[id]);
flag[id] = 1;
}
else
printf("\nProcess %d\tMemory Not Allocated", (i+1));
}
 return 0;
}
```

```
Enter number of process to be added to main memory:3
Enter memory to be assigned to process 1 :500
Enter memory to be assigned to process 2 :30
Enter memory to be assigned to process 3 :
700

Process 1       Memory Allocated        Partition:600
Process 2       Memory Allocated        Partition:200
Process 3       Memory Not Allocated

...Program finished with exit code 0
Press ENTER to exit console.
```

2. Best fit:

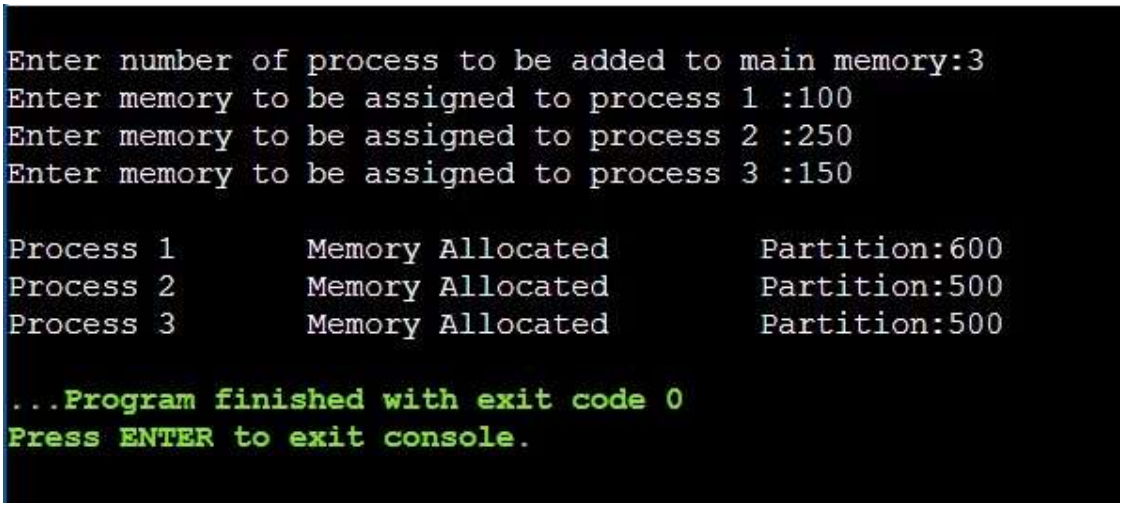
```
#include <stdio.h>
int main()
{
int totalMem = 0;
int part[] = {200, 400, 600, 500, 300, 250}; int
i, j;
printf("\nEnter number of process to be added to main memory:"); int
n ;
scanf("%d", &n); int
mem_p[n]; int
flag[6]; for(i = 0; i <
n; i++)
{
printf("Enter memory to be assigned to process %d : ",(i+1));
scanf("%d", &mem_p[i]);
}
```

```
for(i = 0; i < 6; i++)
flag[i] = 0;  int diff =
10000, id;
for(i = 0; i < n; i++)
{ id = -
1;
for(j = 0; j < 6; j++)
{
if((flag[j] == 0) && (mem_p[i] <= part[j]) && (part[j] - mem_p[i]
< diff))
{
diff = part[j] - mem_p[i]; id
= j;
}
}
if(id != -1)
{
printf("\nProcess %d\tMemory Allocated\tPartition:%d ",(i+1),part[id]);
flag[id] = 1;
}
else
 printf("\nProcess %d\tMemory Not Allocated", (i+1));
diff = 10000;
}
 return 0;
}
```



3. Worst fit

```
#include <stdio.h>
int main()
{
int totalMem = 0;
int part[] = {200, 400, 600, 500, 300, 250}; int
i, j;
printf("\nEnter number of process to be added to main memory:"); int
n ;
scanf("%d", &n); int
mem_p[n];
for(i = 0; i < n; i++)
{
printf("Enter memory to be assigned to process %d :",(i+1));
scanf("%d", &mem_p[i]);
} int diff = 0,
id;
for(i = 0; i < n; i++)
{ id = -
1;
for(j = 0; j < 6; j++)
{
   if((mem_p[i] <= part[j]) && (part[j] - mem_p[i] > diff))
{
diff = part[j] - mem_p[i]; id
= j;
}
}  if(id != -
1)
{
printf("\nProcess %d\tMemory Allocated\tPartition:%d ",(i+1),part[id]);
part[id] = part[id] - mem_p[i];
}
else
 printf("\nProcess %d\tMemory Not Allocated", (i+1));
diff = 0;
 }
 return 0;
}
```



4. Next fit

```
#include <stdio.h>
int main()
{

int part[] = {200, 400, 600, 500, 300, 250}; int
i, j;
printf("\nEnter number of process to be added to main memory:"); int
n ;
scanf("%d", &n); int
mem_p[n];
int flag[6]; for(i =
0; i < n; i++)
{
printf("Enter memory to be assigned to process %d : ",(i+1));
scanf("%d", &mem_p[i]);
}
for(i = 0; i < 6; i++)
flag[i] = 0; int id,
prevId = 0;
for(i = 0; i < n; i++)
{ id = -
1;
for(j = prevId; j < 6; j++)
{
if((flag[j] == 0) && (mem_p[i] <= part[j]))
{ id =
j;
```

```
brea
k;
}
}
 if(id != -1)
{
printf("\nProcess %d\tMemory Allocated\tPartition:%d ",(i+1),part[id]);
flag[id] = 1; prevId = id;
}
else
 printf("\nProcess %d\tMemory Not Allocated", (i+1));
 }
 return 0;
}
```

```
Enter number of process to be added to main memory:3
Enter memory to be assigned to process 1 : 250
Enter memory to be assigned to process 2 : 100
Enter memory to be assigned to process 3 : 350

Process 1       Memory Allocated        Partition:400
Process 2       Memory Allocated        Partition:600
Process 3       Memory Allocated        Partition:500

...Program finished with exit code 0
Press ENTER to exit console.
```

Conclusion:

Learning and executing code for First fit, Best fit, Worst fit, Next fit.

OS:Experiment No. 6

Name: Ayush Jain
SAP ID: 60004200132
Batch: B2
Computer Engineering

Aim- There is a service counter which has a limited waiting queue outside it. It works as follows:

- The counter remains open till the waiting queue is not empty
- If the queue is already full, the new customer simply leaves
- If the queue becomes empty, the outlet doors will be closed (service personnel sleep)
- Whenever a customer arrives at the closed outlet, he/she needs to wake the person at the counter with a wake-up call

Implement the above-described problem using semaphores or mutuexes along with threads. Also show how it works, if there are 2 service personnel, and a single queue. Try to simulate all possible events that can take place, in the above scenario.

Problem Statement-

1) Use Producer Consumer Concept to implement above scenario which treats the queue as buffer.
2) Use Semaphore and mutex for concurrency control and queue status update(full/empty)
3) Use multithreading for implementation

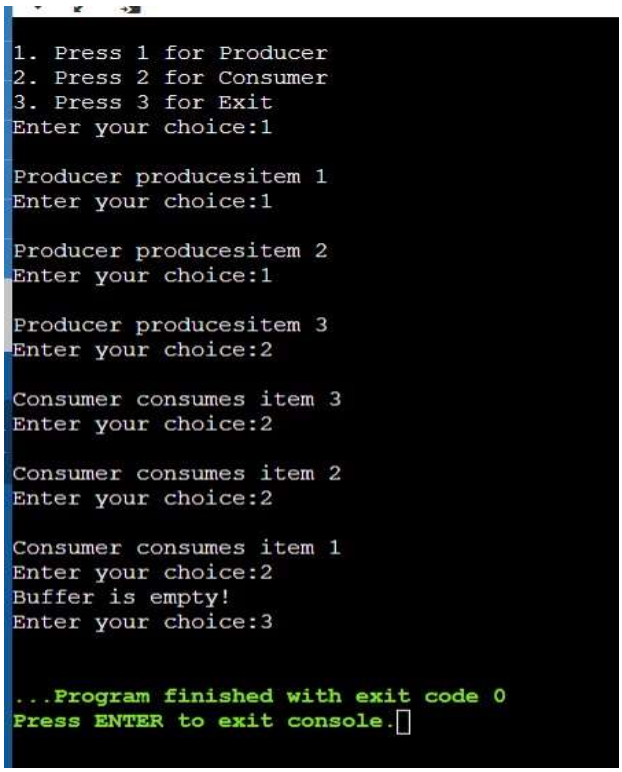
Code-

```
#include <stdio.h> #include
<stdlib.h>
int mutex = 1; int
full = 0; int empty =
5, x = 0;
int wait (int n)
{     n--;
return n;
}
int Signal(int n)
{
n++;
   return n;
}
void producer()
{
```

```
    mutex =wait(mutex);
empty = wait(empty);
x=Signal(x);
printf("\nProducer produces"
"item %d",
          x);
full=Signal(full);
    mutex =Signal(mutex);
}
void consumer()
{
    mutex=wait(mutex);
    full=wait(full);

    printf("\nConsumer consumes "
"item %d",            x);
    x=wait(x);
empty=Signal(empty);
mutex=Signal(mutex);
}

int main()
{
    int n, i;
                    printf("\n1. Press 1 for
  Producer"            "\n2. Press 2 for
                                 Consumer"
          "\n3. Press 3 for Exit");
#pragma omp critical

    for (i = 1; i > 0; i++) {

        printf("\nEnter your choice:");
scanf("%d", &n);         switch (n) {
case 1:
           if ((mutex == 1)
&& (empty != 0)) {
               producer();
           }
else {
               printf("Buffer is full!");
           }
break;

        case 2:
```

```
        if ((mutex == 1)
&& (full != 0)) {
            consumer();
        }
else {
            printf("Buffer is
empty!");
        }
break;
case 3:
exit(0);
break;
      }
   }
}
```

Output:



Conclusion:

Solving the producer consumer problem via use of semaphore and mutex variables.

Operating Systems

Experiment No. 7

Name: Ayush Jain

SAPID: 60004200132

Batch: B1

Branch: Computer Engineering

Aim- Implement order scheduling in supply chain using Banker's Algorithm

Problem Statement-

In supply chain, a deadlock can frequently occur when managing supply and demand between supplier and retailers. Supplier needs to reasonably arrange sequence of order to satisfy retailers. Scheduling management for orders is critical in product supply chain to ensure every retailer to receive commodities from a common supplier. The order contracts between supplier and retailers can specify various payment options such as before or after the delivery of commodities or even several installments. With such a variation in money flow, a supplier has to manage not only the cost for manufacturing commodities but also other operational cost such as management fee, equipment maintenance and storage charge. These situations may cause supplier to reach insufficient money flow to produce commodities, resulting in a low productivity or even a state of financial deficit. Hence, the risk of capital chain rupture for supplier increases with the increase in number of orders or amounts of commodities. Without a careful scheduling of orders, resources allocation of supplier may result in a deadlock situation. Therefore, efficient algorithm is required to avoid such deadlock situations leading to capital chain rupture in supply chain management.

In order to avoid capital chain rupture in OSM and to effectively manage the money flow, it is necessary to adequately allocate requirements of commodities to ensure satisfaction of retailers.

- A single order in OSM as a process and multiple commodities as resources that retailers need.
- If scheduling in OSM is inadequate, it will cause insufficient money flow of supplier and create a risk of capital chain rupture.
- Deadlock will arise when supplier cannot produce commodities for current order and retailers enter an unsafe state of indefinite waiting.
- Since original resources (commodities) do not return but payment is made from retailers to supplier in OSM, define a safe as a state with sufficient money flow for the next production.
- Use Banker's algorithm to acquire a safe sequence in OSM where the money flow remains in a safe state

Code:

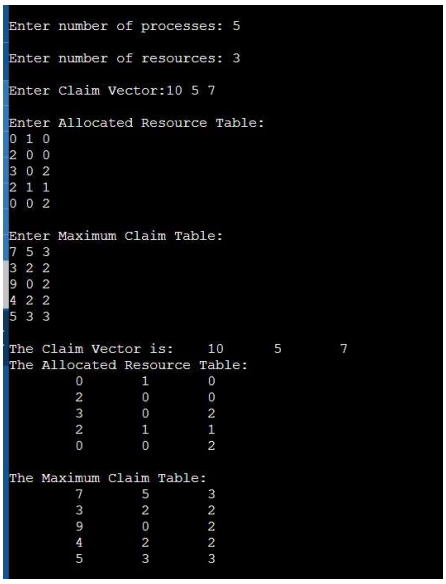
```
#include <stdio.h>

int current[5][5], maximum_claim[5][5], available[5]; int
allocation[5] = {0, 0, 0, 0, 0}; int maxres[5], running[5], safe =
0; int counter = 0, i, j, exec, resources, processes, k = 1;


int main()
{
printf("\nEnter number of processes: ");      scanf("%d",
&processes);


    for (i = 0; i < processes; i++)
{         running[i] = 1;
counter++;
    }

    printf("\nEnter number of resources: ");      scanf("%d",
&resources);


    printf("\nEnter Claim Vector:");
     for (i = 0; i < resources; i++)
```

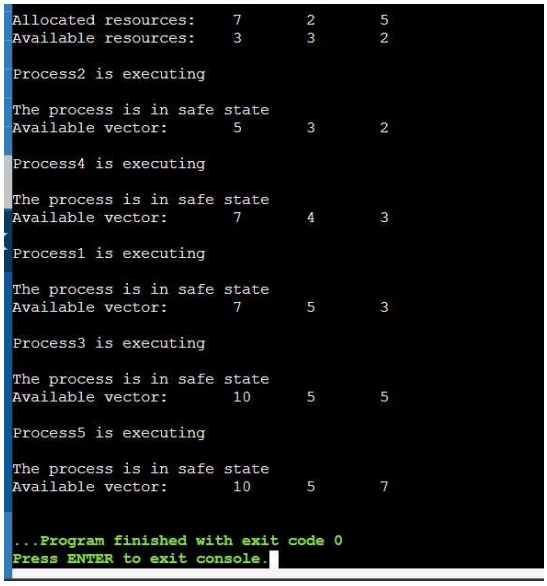
```
{
      scanf("%d", &maxres[i]);
    }

  printf("\nEnter Allocated Resource Table:\n");      for (i =
0; i < processes; i++)
{       for(j = 0; j < resources; j++)
{
  scanf("%d", &current[i][j]);
       }
    }

    printf("\nEnter Maximum Claim Table:\n");      for (i =
0; i < processes; i++)
{         for(j = 0; j < resources; j++)
{
          scanf("%d", &maximum_claim[i][j]);
       }
    }

printf("\nThe Claim Vector is: ");      for (i
= 0; i < resources; i++)
{       printf("\t%d", maxres[i]);
}

              printf("\nThe Allocated Resource Table:\n");
    for (i = 0; i < processes; i++)
{       for (j = 0; j < resources; j++)
{            printf("\t%d", current[i][j]);
       } printf("\n");
    }

    printf("\nThe Maximum Claim Table:\n");      for (i =
0; i < processes; i++)
{         for (j = 0; j < resources; j++)
{
      printf("\t%d", maximum_claim[i][j]);
       }
printf("\n");
    }
```

```
    for (i = 0; i < processes; i++)
{         for (j = 0; j < resources; j++)
{             allocation[j] += current[i][j];
        }
    }

    printf("\nAllocated resources:");
     for (i = 0; i < resources; i++)
{         printf("\t%d", allocation[i]);
    }

    for (i = 0; i < resources; i++)
{        available[i] = maxres[i] - allocation[i];
}

    printf("\nAvailable resources:");      for (i
= 0; i < resources; i++)
{         printf("\t%d", available[i]);
    }
printf("\n");

    while (counter != 0)
{         safe = 0;         for (i = 0; i <
processes; i++)
{             if (running[i])
{                 exec = 1;                 for (j = 0; j <
resources; j++)
{                     if (maximum_claim[i][j] - current[i][j] > available[j]) {
                       exec = 0;
break;
                  }
}                 if (exec)
{                     printf("\nProcess%d is executing\n", i + 1);
running[i] = 0;                     counter--;                     safe = 1;
                    for (j = 0; j < resources; j++)
{                       available[j] += current[i][j];
                  }
break;
```

```
            }
          }       }
if (!safe)
{
          printf("\nThe processes are in unsafe state.\n");            break;
      }
else
{                 printf("\nThe process is in safe state");
printf("\nAvailable vector:");

          for (i = 0; i < resources; i++)
{               printf("\t%d", available[i]);
          }
printf("\n");
      }
   }     return
0;
}
```

Output:





Operating System

Experiment 8

Name: Ayush Jain
SAP ID: 60004200132
Batch: B2
Computer Engineering

Aim- Using the CPU-OS simulator to analyze and synthesize the following:

a. **Process Scheduling algorithms.**
b. **Thread creation and synchronization.**
c. **Deadlock prevention and avoidance.**

Problem Statement:

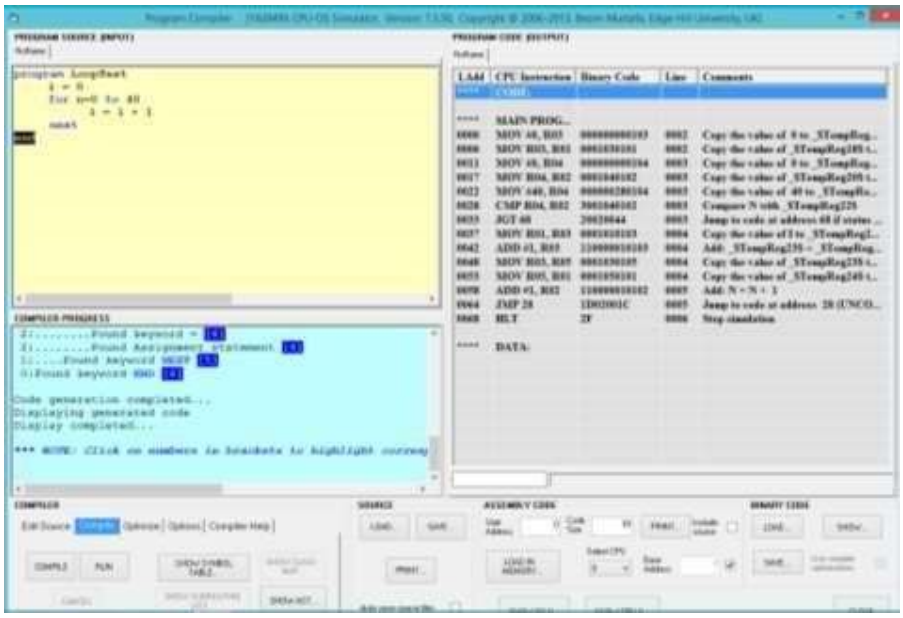
1. Install CPU-OS simulator
1. Perform the following steps

a. Process Scheduling algorithms

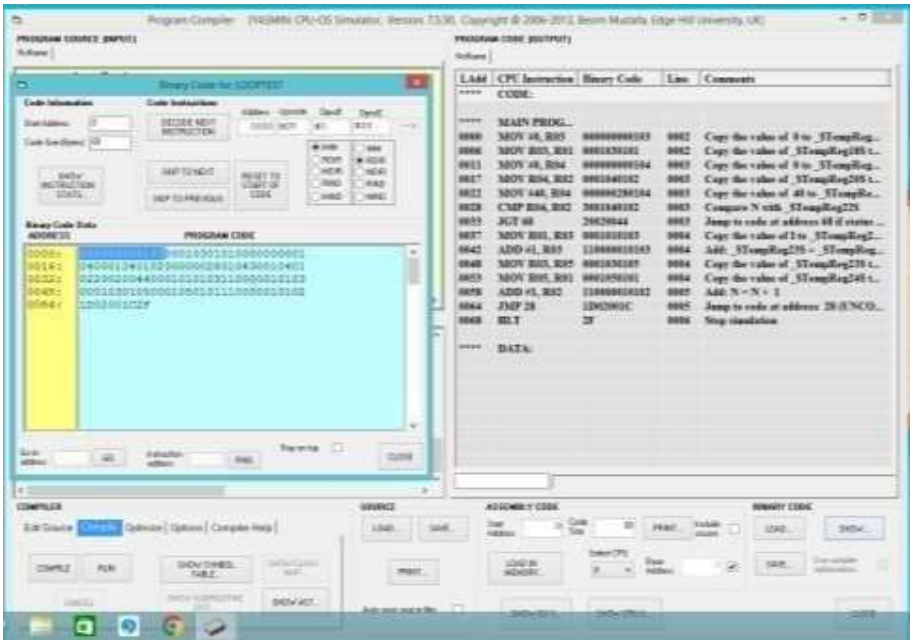
Loading and Compiling Program

You need to create some executable code so that it can be run by the CPU simulator under the control of the OS simulator. In order to create this code, you need to use the compiler which is part of the system simulator. This compiler is able to compile simple high-level source statements similar to Visual Basic. To do this, open the compiler window by selecting the
COMPILER... button in the current window. You should now be looking at the compiler window.
In the compiler window, enter the following source code in the compiler's source editor window (under **PROGRAM SOURCE** frame title):

```
program LoopTest
      i = 0 for n = 0
      to 40 i = i + 1
next end
```



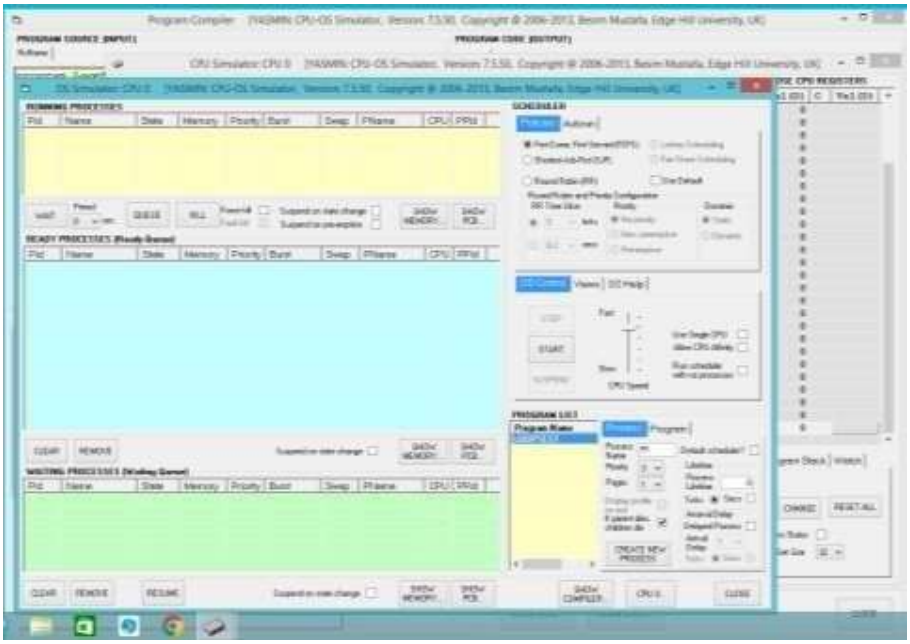
Now you need to compile this in order to generate the executable code. To do this, click on the **COMPILE...** button. You should see the code created on the right in **PROGRAM CODE** view. Make a habit of saving your source code.



Click on the button **SHOW...** in **BINARY CODE** view. You should now see the **Binary Code for LOOPTEST** window. Study the program code displayed in hexadecimal format.

Now, this code needs to be loaded in memory so that the CPU can execute it. To do this, first we need to specify a base address (in **ASSEMBLY CODE** view): uncheck the box next to the edit box with label **Base Address**, and then enter 100 in the edit box.

Now, click on the **LOAD IN MEMORY…** button in the current window. You should now see the code loaded in memory ready to be executed. You are also back in the CPU simulator at this stage. This action is equivalent to loading the program code normally stored on a disc drive into RAM on the real computer systems.



Creating processes from programs in the OS simulator.

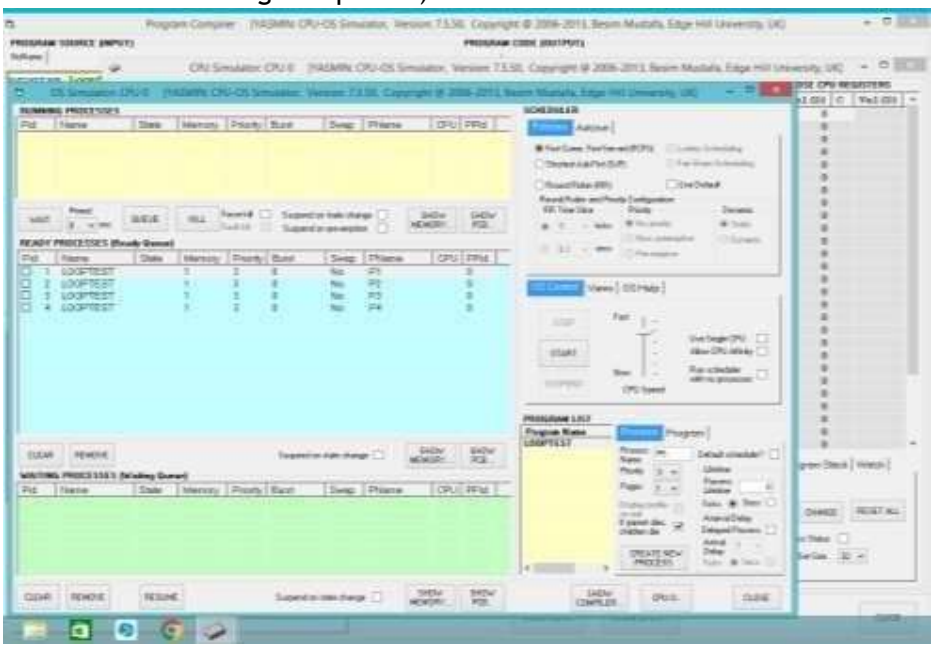
We are now going to use the OS simulator to run this code. To enter the OS simulator, click on the **OS 0…** button in the current window. The OS window opens. You should see an entry, titled **LoopTest**, in the **PROGRAM LIST** view. Now that this program is available to the OS
simulator, we can create as many instances, i.e. processes, of it as we like. You do this by clicking on the **CREATE NEW PROCESS** button. Repeat this four times. Observe the four instances of the program being queued in the ready queue which is represented by the
READY PROCESSES view.

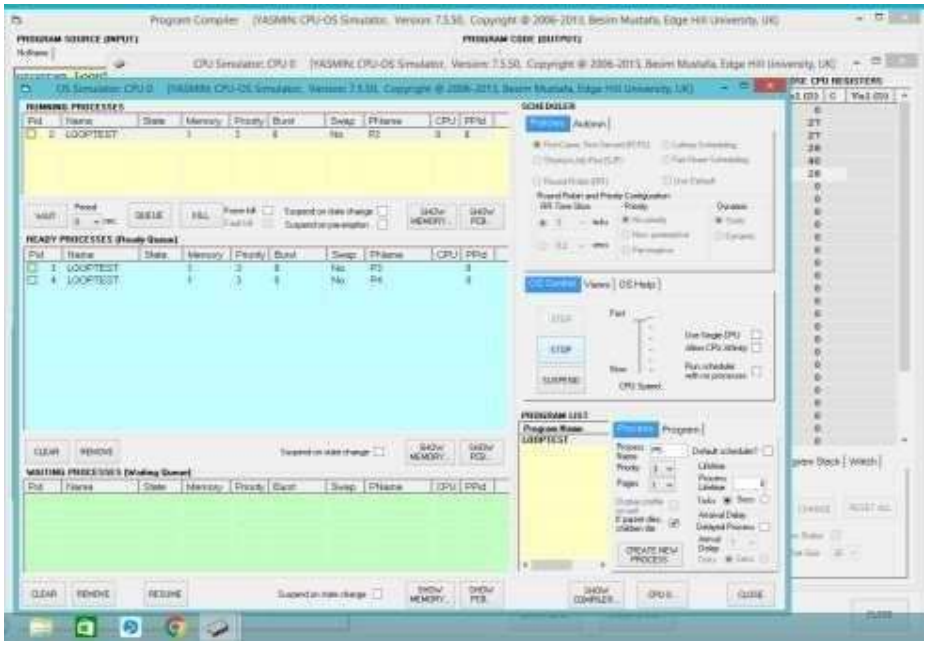
NOTE: it is very important that you follow the instructions below without any deviation. If you do, then you must re-do the exercise from the beginning as any follow-up action(s) may give the wrong results.

Selecting different scheduling policies and run the processes in the OS simulator

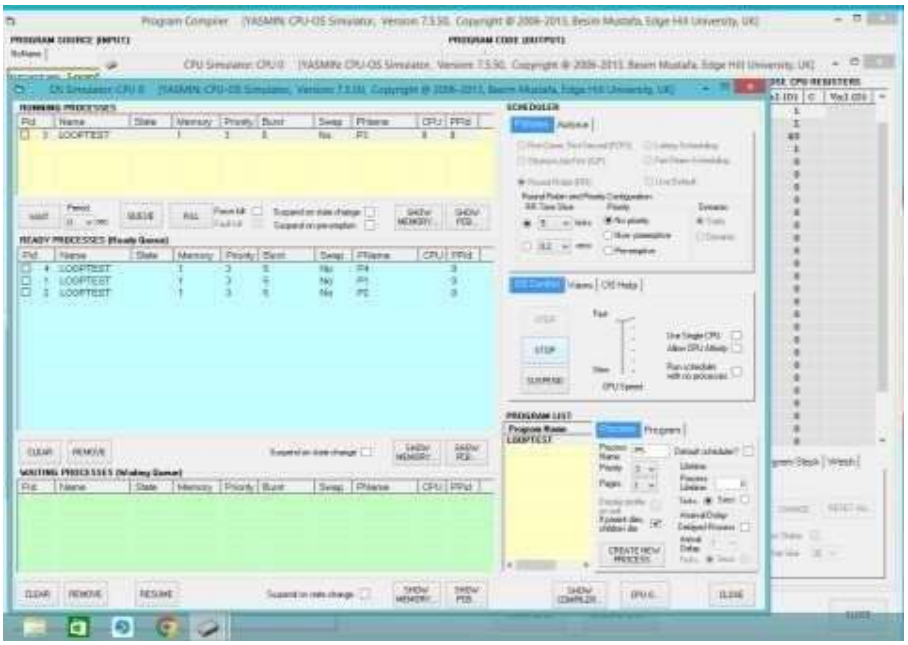
Make sure the **First-Come-First-Served (FCFS)** option is selected in the **SCHEDULER/Policies** view. At this point the OS is inactive. To activate, first move the **Speed** slider to the fastest position, then click on the **START** button. This should start the OS simulator running the processes. Observe the instructions executing in the CPU simulator
window. Make a note of what you observe in the box below as the processes are run (you need

to concentrate on the two views: **RUNNING PROCESSES** and the **READY PROCESSES** during this period).

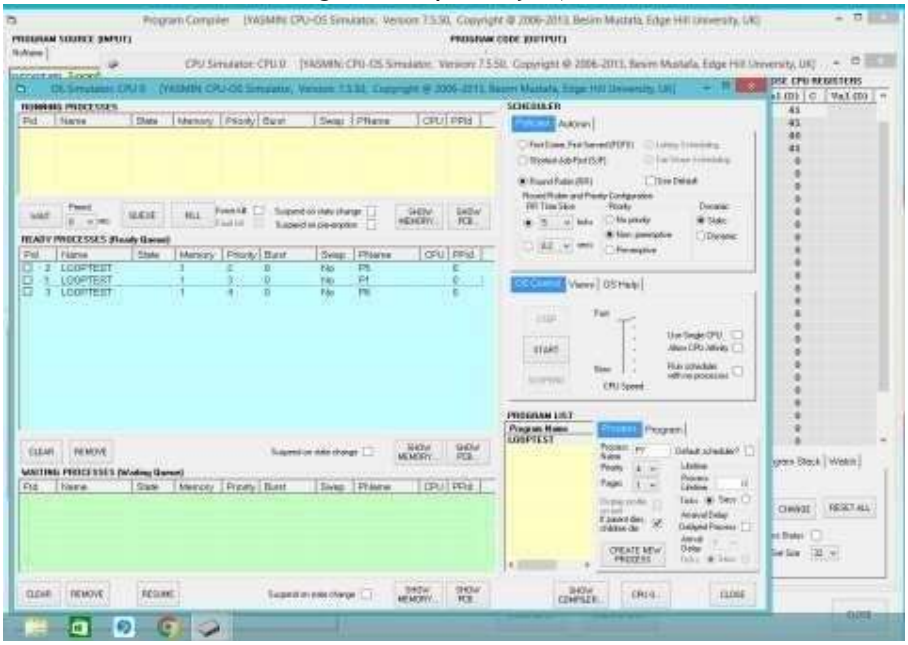




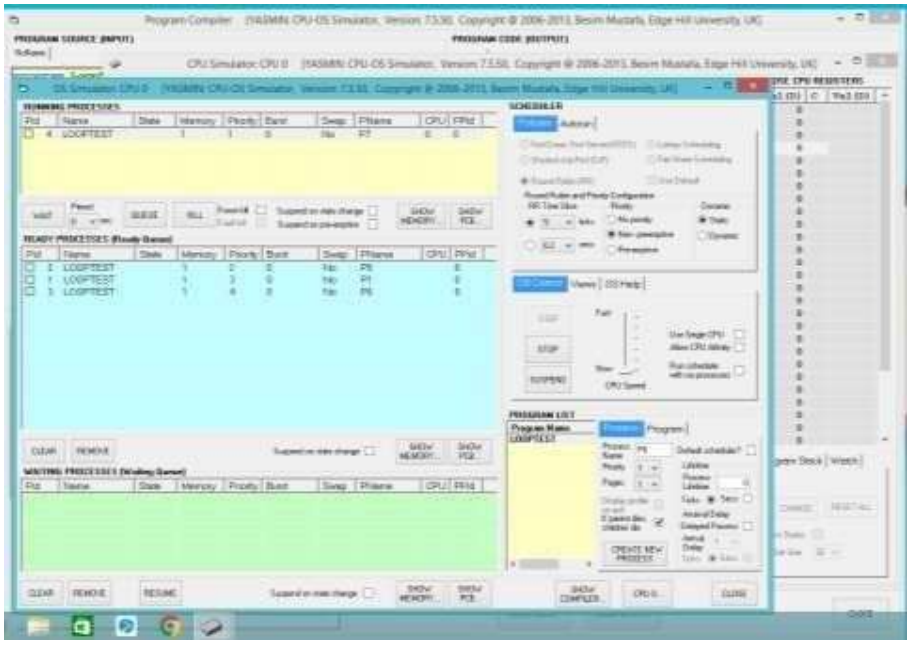
When all the processes finish, do the following. Select **Round Robin (RR)** option in the **SCHEDULER/Policies** view. Then select the **No priority** option in the **SCHEDULER/Policies/Priority** frame. Create three processes. Click on the **START** button and observe the behaviors of the processes until they all complete. You may wish to use speed slider to slow down the processes to better see what is happening. Make a note of what you observed in the box below and compare this with the observation in step 1 above.



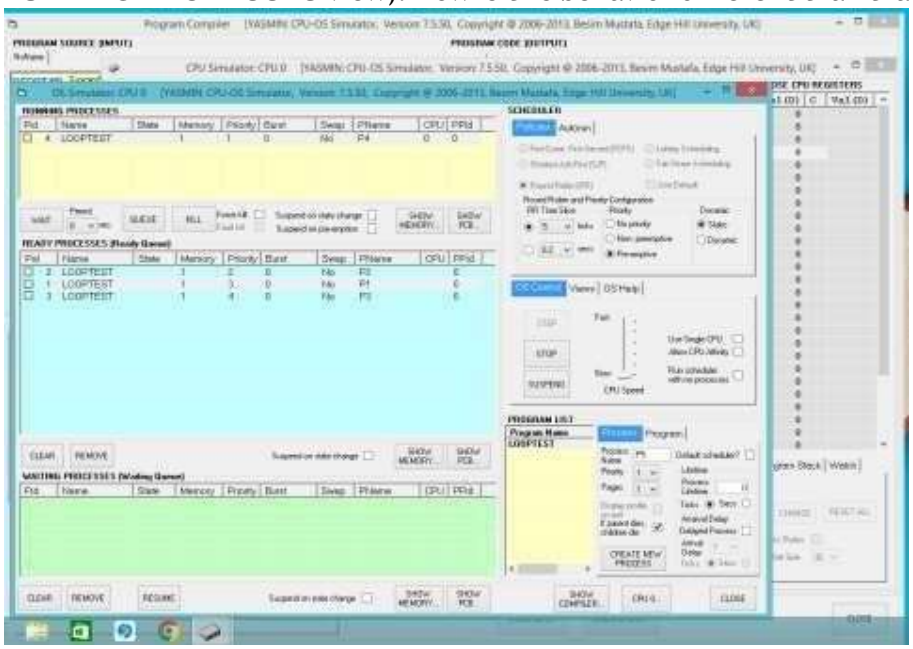
Then select the **Non-preemptive** priority option in the **SCHEDULER/Policies/Priority** frame. Create three processes with the following priorities: 3, 2 and 4. Use the **Priority** drop down list to select priorities. Observe the order in which the three processes are queued in the ready queue represented by the **READY PROCESSES** view and make a note of this in the box below (note that the lower the number the higher the priority is).

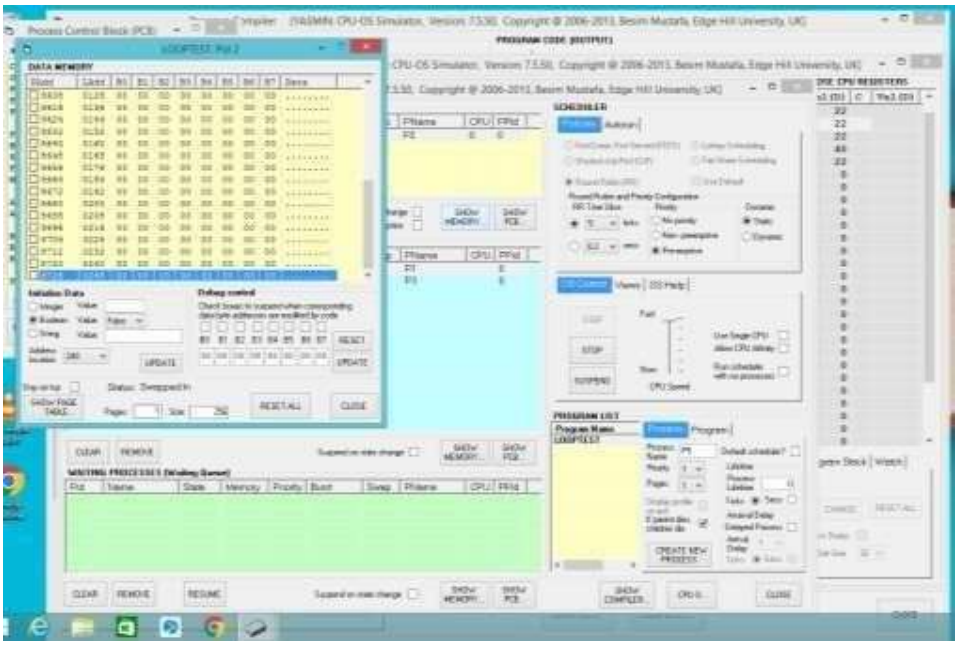


Slide the **Speed** selector to the slowest position and then hit the **START** button. While the first process is being run do the following. Create a fourth process with priority 1. Make a note of what you observe (pay attention to the **READY PROCESSES** view) in the box below.



Now kill all four processes one by one as they start running. Next, select the **Pre-emptive** option in the **SCHEDULER/Policies/Priority** frame. Create the same three processes as in step 3 and then hit the **START** button. While the first process is being run do the following. Create a
fourth process with priority 1. Make a note of what you observe (pay attention to the **RUNNING PROCESSES** view). How is this behavior different than that in step 4 above?





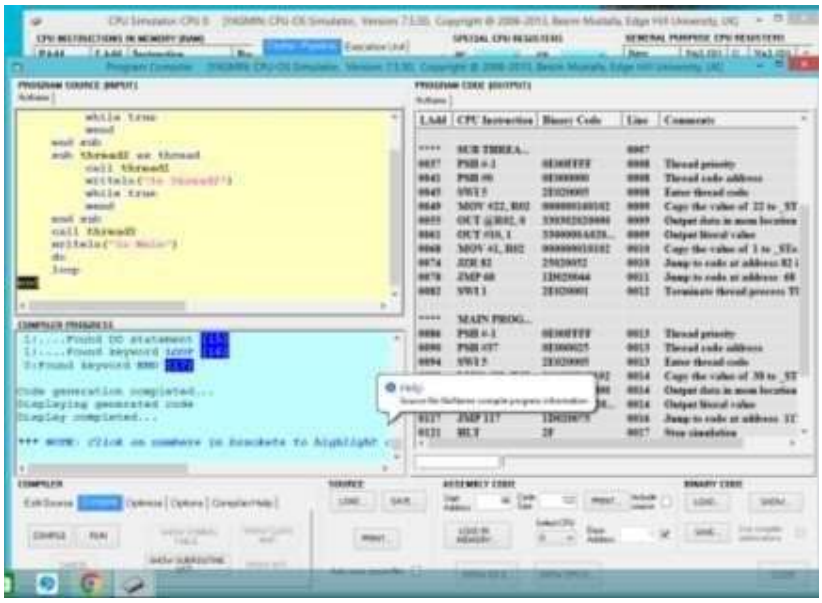
Thread creation and synchronization.

Loading and Compiling a Program

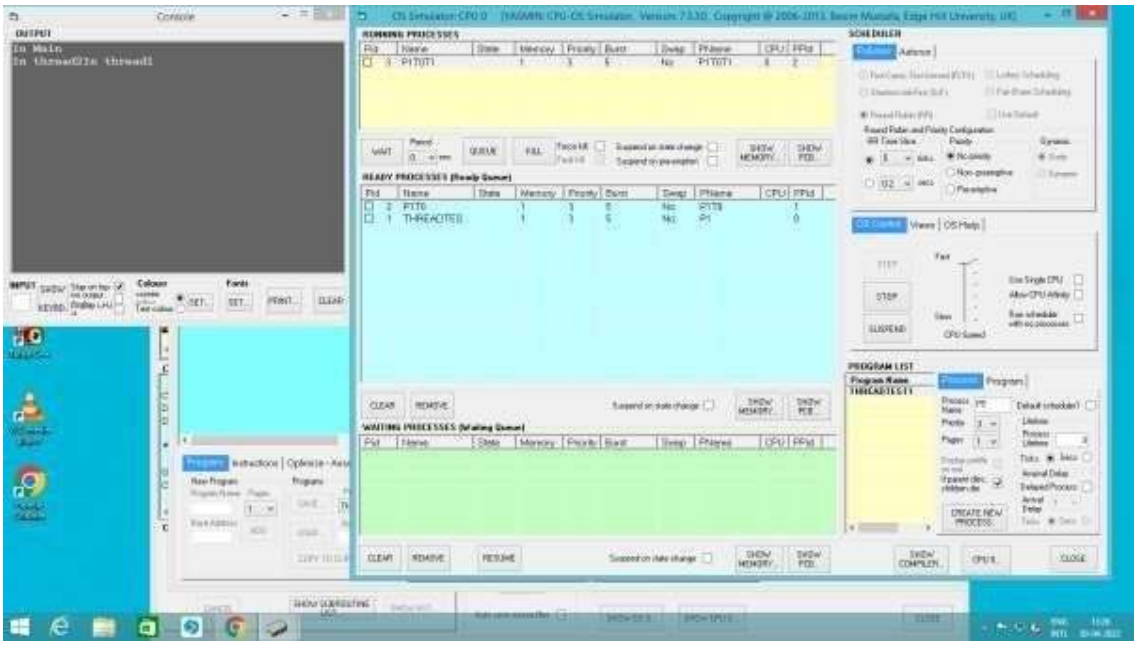
In the compiler window enter the following source code:

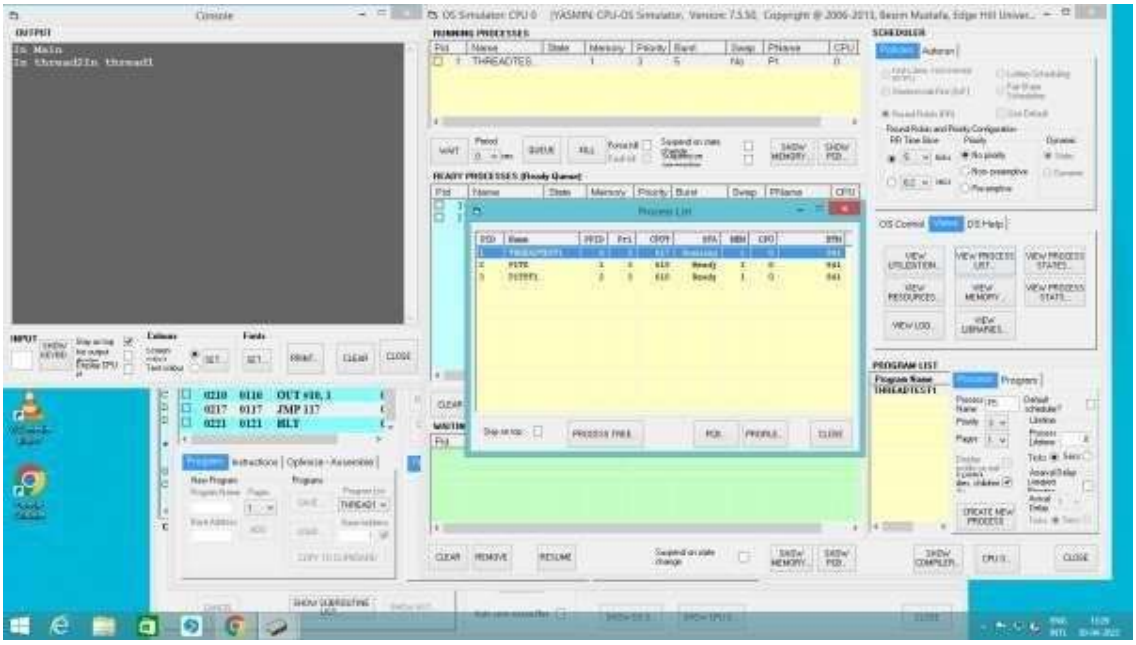
```
program ThreadTest1 sub thread1
as thread writeln("In thread1")
while true wend
     end sub sub thread2 as
     thread call thread1
     writeln("In thread2")
     while true wend
     end      sub   call
     thread2
     writeln("In main")
     do loop
end
```

Compile the above source and load the generated code in memory.



Make the console window visible by clicking on the **INPUT/OUTPUT...** button. Also make sure the console window stays on top by checking the **Stay on top** check box.
Now, go to the OS simulator window (use the **OS...** button in the CPU simulator window) and create a single process of program *ThreadTest1* in the program list view. For this use the
CREATE NEW PROCESS button.
Make sure the scheduling policy selected is **Round Robin** and that the simulation speed is set at maximum.
Hit the **START** button and at the same time observe the displays on the console window. Briefly explain your observations and the no. of processes created in the box below.

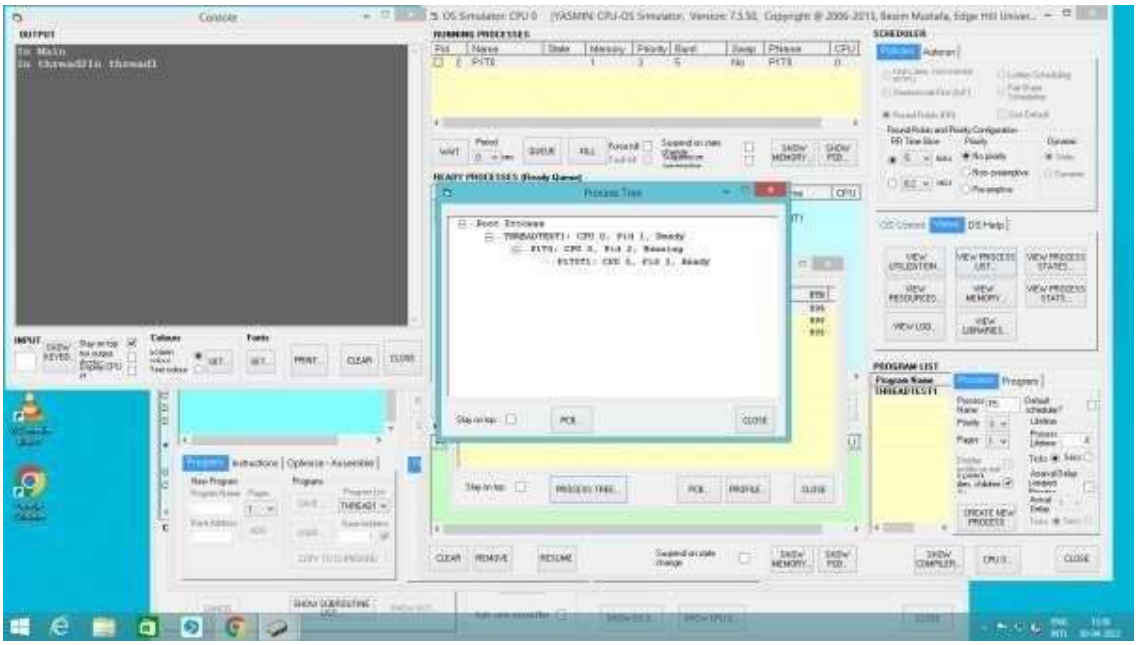




Now, click on the **Views** tab and click on the **VIEW PROCESS LIST...** button. Observe the contents of the window now displaying.

In the Process List window hit the **PROCESS TREE...** button. Observe the contents of the window now displaying.

Briefly explain your observations in the box below:



Stop the running processes by repeatedly using the **KILL** button in the OS simulator window.

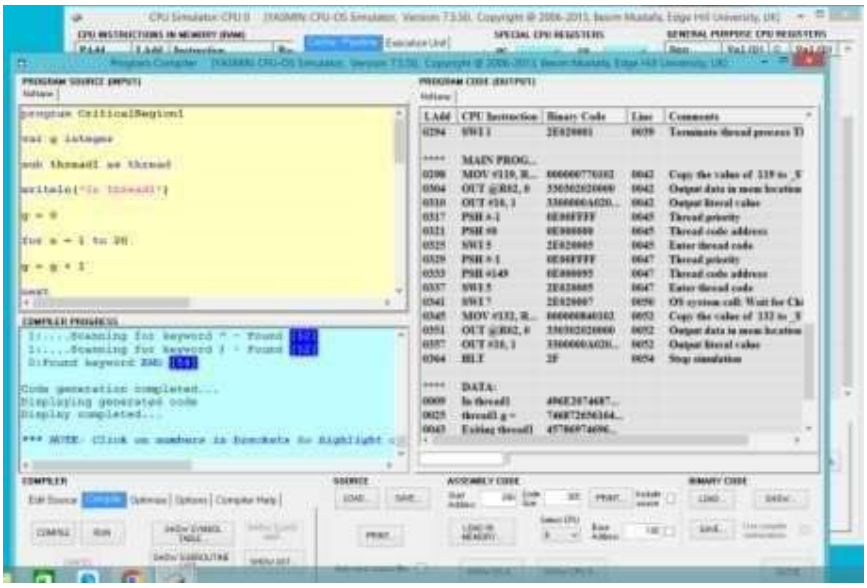
Synchronization

Loading and Compiling a Program

In the compiler window, enter the following source code in the compiler source editor area (under **PROGRAM SOURCE** frame title). Make sure your program is exactly the same as the one below (best to use copy and paste for this).

```
program CriticalRegion1 var g
     integer sub thread1 as
     thread writeln("In
     thread1") g = 0 for n = 1
     to 20
g = g + 1
          next
          writeln("thread1 g = ", g)
          writeln("Exiting thread1")
     end sub
     sub thread2 as thread
          writeln("In thread2")
          g = 0
     for n = 1 to 12
     g = g + 1
          next
          writeln("thread2 g = ", g)
          writeln("Exiting thread2")
```

```
        end sub writeln("In

        main") call thread1

        call thread2


        wait
        writeln("Exiting main")
end
```



The above code creates a main program called *CriticalRegion1*. This program creates two threads thread1 and thread2. Each thread increments the value of the global variable **g** in two separate loops.

- Compile the above code using the **COMPILE...** button.
- Load the CPU instructions in memory using the **LOAD IN MEMORY** button.
- Display the console using the **INPUT/OUTPUT...** button in CPU simulator.
- On the console window check the **Stay on top** check box.

Running the above code

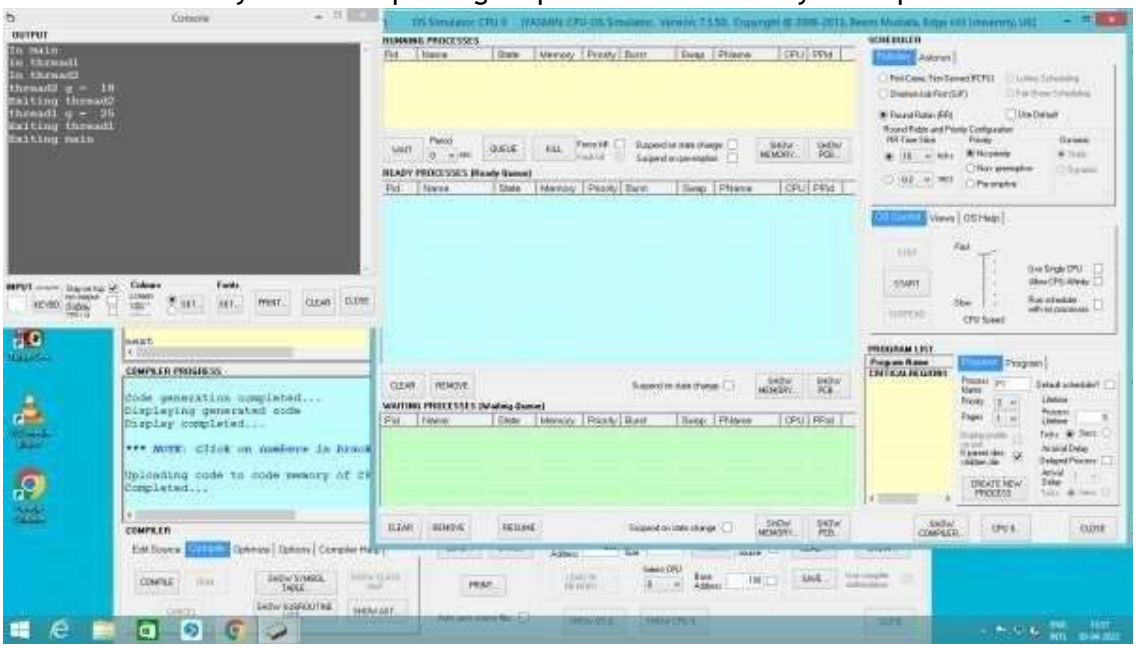
- Enter the OS simulator using the **OS 0...** button in CPU simulator.
- You should see an entry, titled *CriticalRegion1*, in the **PROGRAM** view. **LIST**

- Create an instance of this program using the **NEW PROCESS** button. □ Select **Round Robin** option in the **SCHEDULER/Policies** view.
- Select **10 ticks** from the drop-down list in **RR Time Slice** frame.
- Make sure the console window is displaying (see above).

- Move the **Speed** slider to the fastest position.
- Start the scheduler using the **START** button.

Now, follow the instructions below <u>without</u> any deviations:

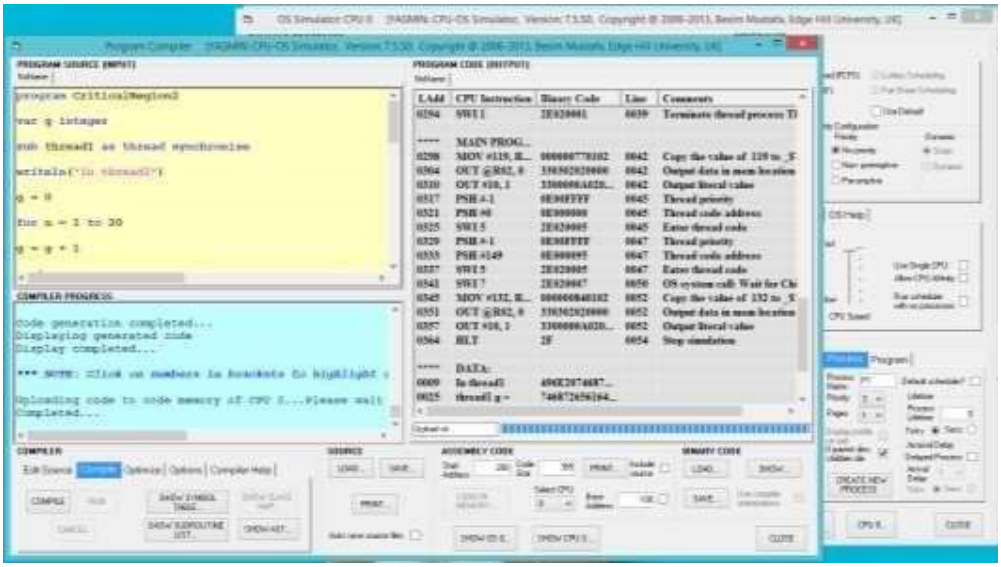
When the program stops running, make a note of the two displayed values of **g**. Are these values what you were expecting? Explain if there are any discrepancies.



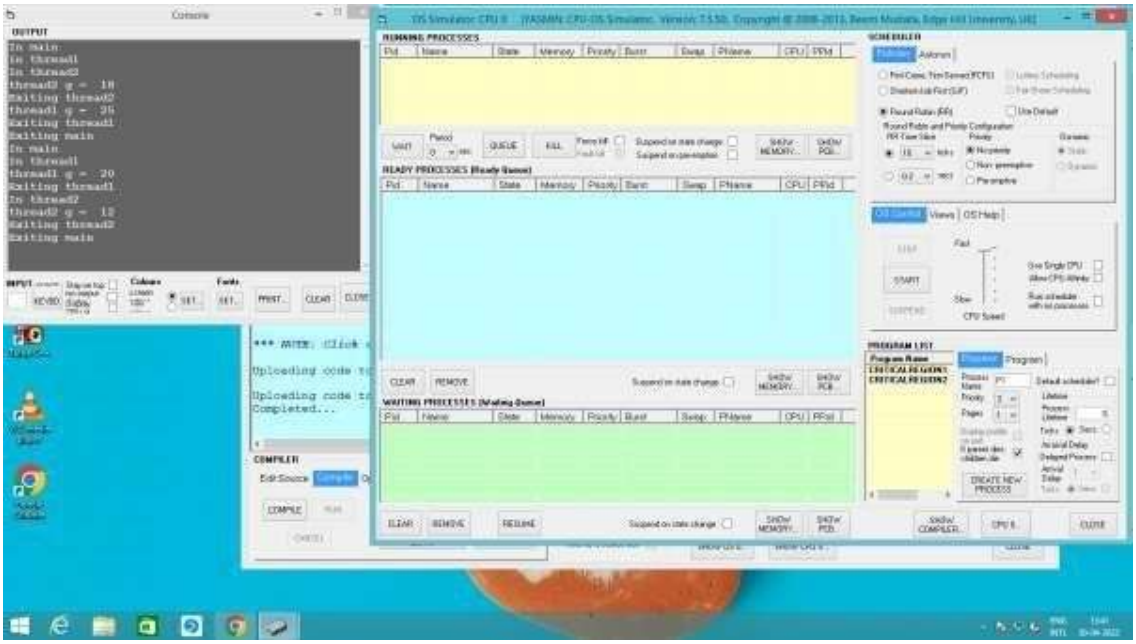
Modify this program as shown below. The changes are in bold and underlined. Rename the program *CriticalRegion2*.

```
     program CriticalRegion2 var
          g integer
     sub thread1 as thread synchronise
writeln("In thread1") g = 0 for n =
               1 to 20 g = g + 1
               next
               writeln("thread1 g = ", g)
               writeln("Exiting thread1")
          end sub
          sub thread2 as thread synchronise
               writeln("In thread2") g = 0
          for n = 1 to 12
          g = g + 1
               next
               writeln("thread2 g = ", g)
               writeln("Exiting thread2")
```

```
        end sub writeln("In

        main")

        call thread1 call
        thread2   wait
        writeln("Exiting main")
end
```



Compile the above program and load in memory as before. Next, run it and carefully observe how the threads behave. Make a note of the two values of variable **g**.



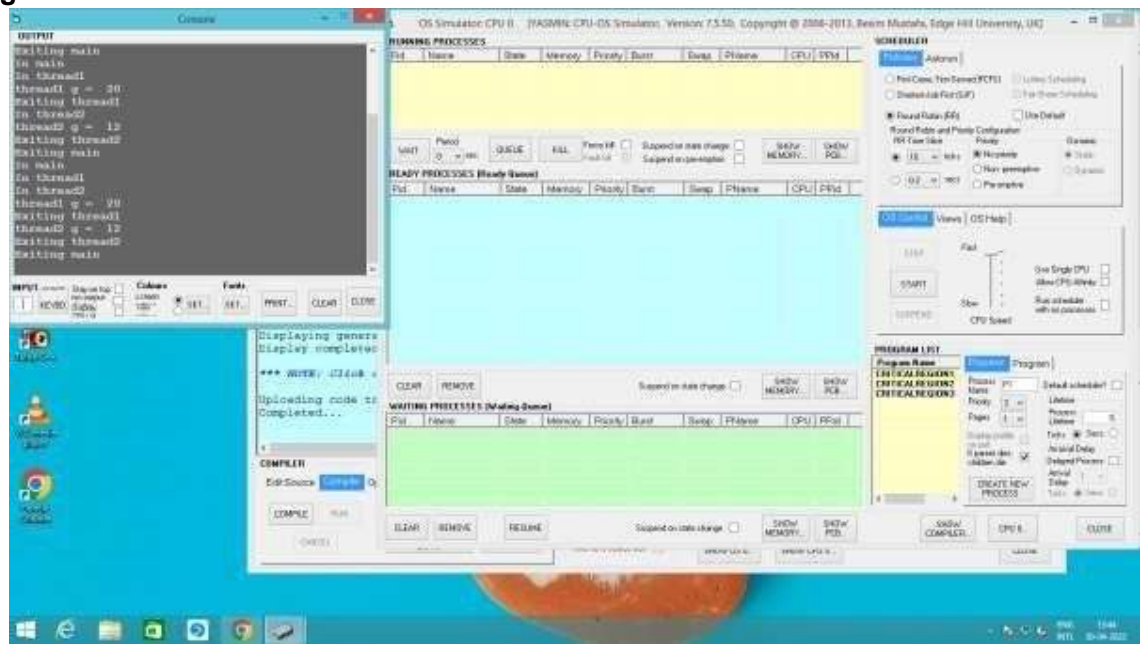
Modify this program for the second time. The new additions are in bold and underlined. Remove the two **synchronise** keywords. Rename it *CriticalRegion3*.

```
program CriticalRegion3 var g
     integer sub thread1 as
     thread_writeln("In
     thread1") enter
          g = 0 for n = 1
          to 20 g = g + 1
          next
          writeln("thread1 g = ", g)
          leave
          writeln("Exiting thread1")
     end sub
     sub thread2 as thread
          writeln("In thread2") enter
          g = 0 for n = 1
          to 12
          g = g + 1
          next
          writeln("thread2 g = ", g)
          leave
          writeln("Exiting thread2")
     end sub

     writeln("In main")

     call thread1 call
     thread2

     wait
     writeln("Exiting main")


end
```

NOTE: The **enter** and **leave** keyword pair protect the program code between them. This makes sure the protected code executes exclusively without sharing the CPU with any other thread.

Locate the CPU assembly instructions generated for the **enter** and **leave** keywords in the compiler's **PROGRAM CODE** view. You can do this by clicking in the source editor on any of the above keywords. Corresponding CPU instruction will be highlighted.:

Compile the above program and load in memory as before. Next, run it. Make a note of the two values of variable
g.



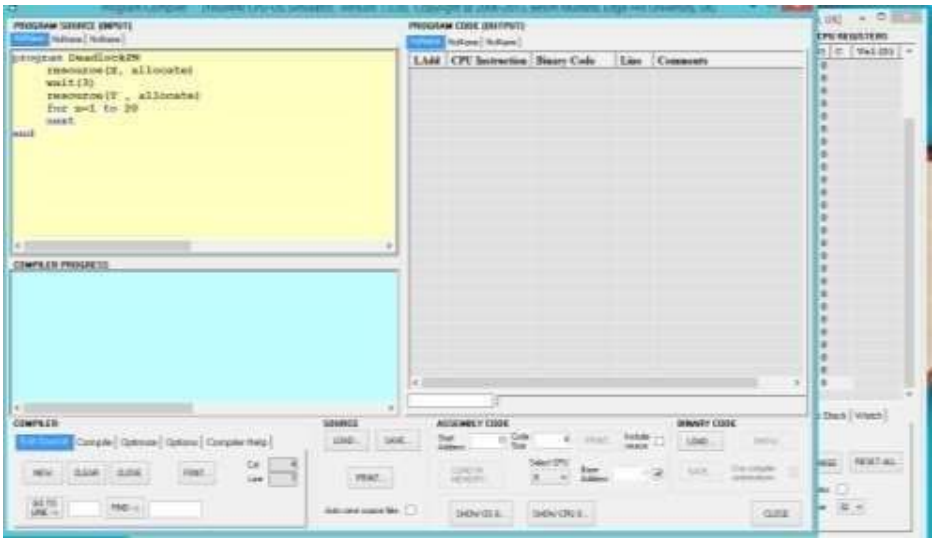
Deadlock prevention and avoidance

Four processes are running. They are called **P1** to **P4**.There are also four resources available (only one instance of each). They are named **R0** to **R3**. At some point of their existence each process allocates a different resource for use and holds it for itself forever. Later each of the processes request another one of the four resources.

Use the Scenario P1 holding R0 and waiting for R1. P2 Holding R1 and waiting for R2. P3 holding R2 and waiting for R3. P4 holding R3 and waiting for R0.
Draw the resource allocation graph for a four process deadlock condition.

In the compiler window, enter the following source code in the compiler source editor area (under **PROGRAM SOURCE** frame title).

```
program DeadlockP N
     resource(X, allocate)
     wait(3)
     resource(Y, allocate)
     for n = 1 to 20 next
end
```



2)

a. Copy the above code and paste it in three more edit windows so that you have a total of four pieces of source code. (Click on New under compiler to create new edit windows)
b. In each case change **N** in the program name to 1 to 4, e.g. DeadlockP**1,** DeadlockP**2,** etc.
c. Look at your graph you constructed in (1) above and using that information fill in the values for each of the **X**s and **Y**s in the four pieces of source code. (X is resource the process is holding and Y is the resource process is waiting for. Eg. For P1, X=0 and Y=1)
d. Compile each one of the four source code.
e. Load in memory the four pieces of code generated.
f. Now switch to the OS simulator.
g. Create a single instance of each of the programs. You can do this by double-clicking on each of the program names in the **PROGRAM LIST** frame under the **Program Name** column.
h. In the **SCHEDULER** frame select **Round Robin (RR)** scheduling policy in the **Policies** tab.
i. In OS Control tab, push the speed slider up to the fastest speed.
j. Select the **Views** tab and click on the **VIEW RESOURCES...** button.
k. Select **Stay on top** check box in the displayed window.
l. Back in the **OS Control** tab use the **START** button to start the OS scheduler and observe the changing process states for few seconds.

m. Have you got a deadlock condition same as you constructed in (1) above? If you haven't then check and if necessary re-do above. Do not proceed to (n) or (3) below until you get a deadlock condition.

n. If you have a deadlock condition then click on the **SHOW DEADLOCKED PROCESSES...** button in the **System Resources** window. Does the highlighted resource allocation graph look like yours?

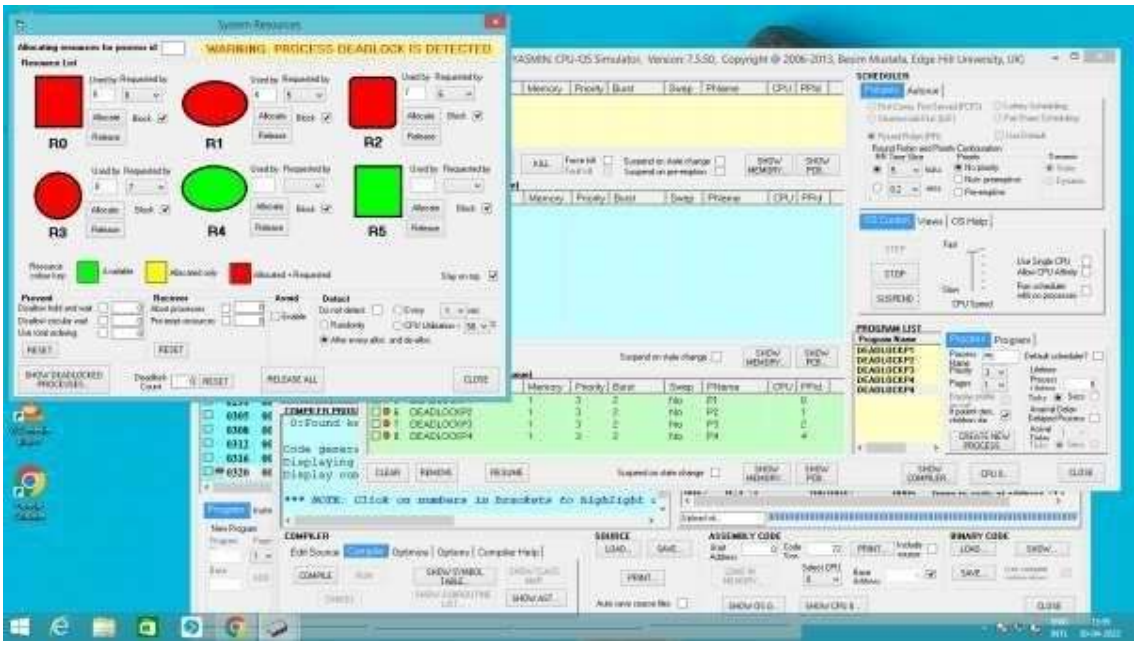
Now that you created a deadlock condition let us try two methods of getting out of this condition:

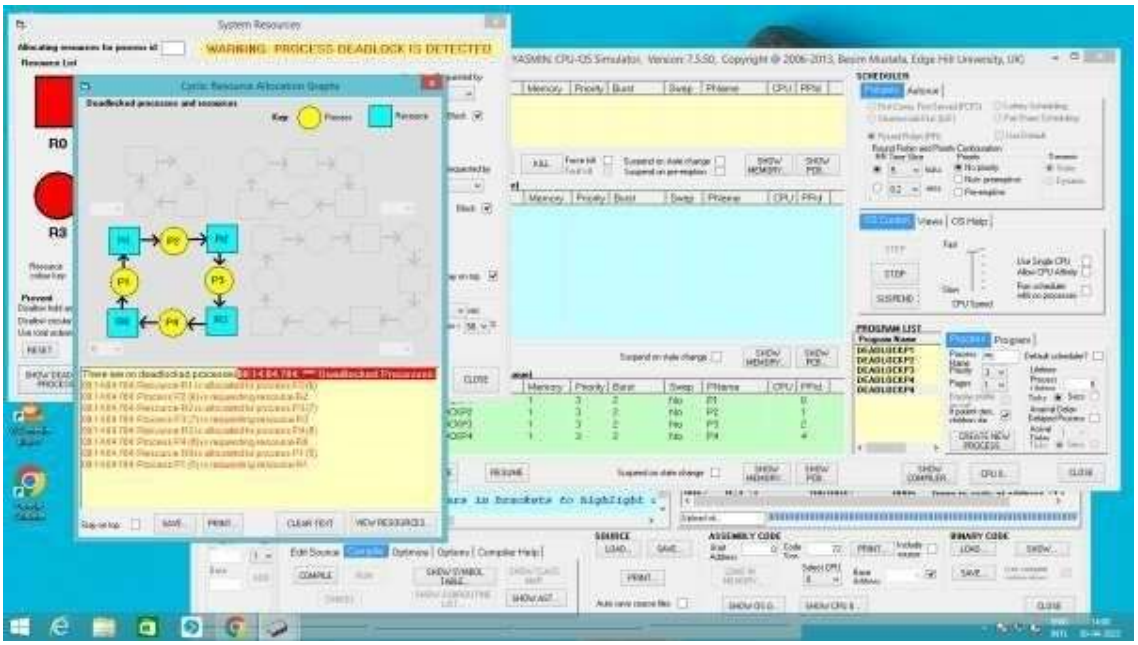
a. In the **System Resources** window, there should be four resource shapes that are in red colour indicating they are both allocated to one process and requested by another.

b. Select one of these resources and click on the **Release** button next to it.

c. Observe what is happening to the processes in the OS Simulator window.

d. Is the deadlock situation resolved? Explain briefly why this helped resolve the deadlock.





e. Re-create the same deadlock condition (steps in 2 above should help).

f. Once the deadlock condition is obtained again do the following: In the OS Simulator window, select a process in the waiting queue in the **WAITING PROCESSES** frame.
g. Click on the REMOVE button and observe the processes.

h. Has this managed to resolve the deadlock? Explain briefly why this helped resolve the deadlock.

This part of the exercises was about two methods of **recovering** from a deadlock condition **after** it happens.

We now look at two methods of **preventing** a deadlock condition **before** it happens.

a. In the **System Resources** window select the **Disallow hold and wait** check box in the **Prevent** frame.

b. Try to re-create the same deadlock condition as before. Have you been successful? What happened? Click on the **SHOW DEADLOCKED PROCESSES...** button and observe the displayed information in the text window for potential clues.

c. Next, uncheck the **Disallow hold and wait** check box and check the **Disallow circular wait** check box.

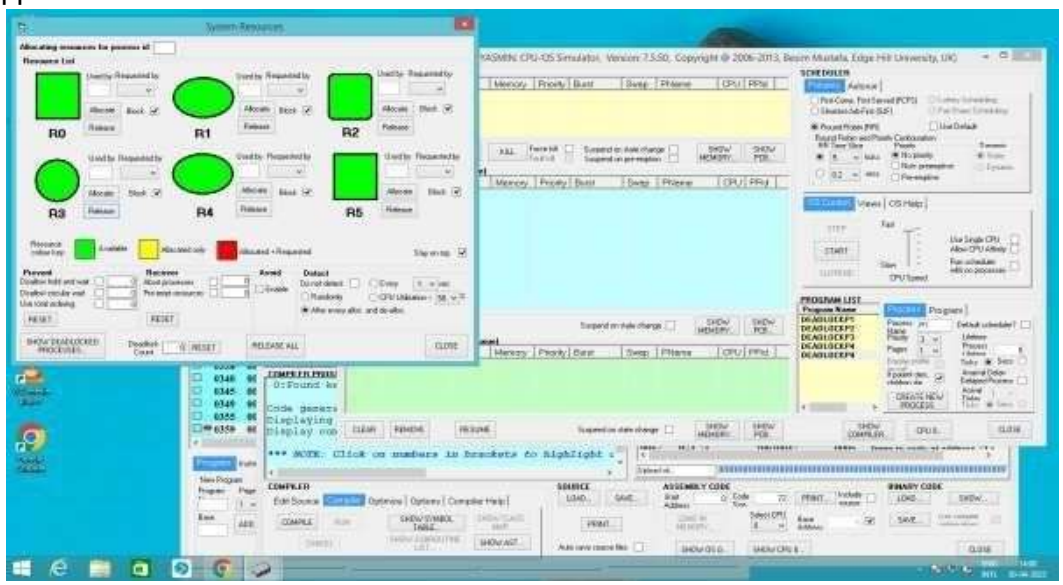
d. Try to re-create the same deadlock condition as before. Have you been successful? What happened? Click on the **SHOW DEADLOCKED PROCESSES...** button and observe the displayed information in the text window for potential clues.

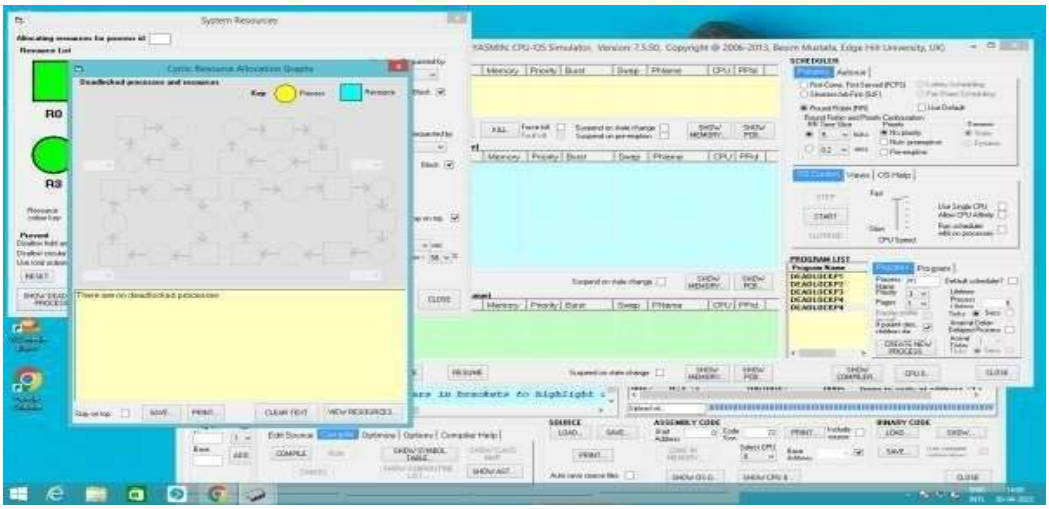
We are now going to try a third method of preventing deadlocking before it happens. It is called "total ordering" method. Here the resources are allocated in increasing resource id numbers only. So, for example, resource R3 must be allocated after resources R0 to R2 and resource R1 cannot be allocated after resource R2 is allocated. Looking at your

resource allocation graph can you see how this ordering can prevent a deadlock? Comment.

a. In the **System Resources** window select the **Use total ordering** check box in the **Prevent** frame. The other options should be unchecked.

b. Try to re-create the same deadlock condition as before. Have you been successful? What happened? Click on the **SHOW DEADLOCKED PROCESSES...** button and observe the displayed information in the text window for potential clues. What happened? Comment.





Conclusion:

Understanding various concepts of OS via simulation.

Operating Systems

Experiment No. 9

Name: Ayush Jain
SAPID: 60004200132
Batch: B1
Branch: Computer Engineering

Aim: Implement various page replacement policies

Problem Statement-
Perform following page replacement policies and perform comparative assessment them.

1) LEAST RECENTLY USED (LRU)
2) FIRST-IN-FIRST-OUT

Description:

In multiprogramming system using dynamic partitioning there will come a time when all of the processes in the main memory are in a blocked state and there is insufficient memory. To avoid wasting processor time waiting for an active process to become unblocked. The OS will swap one of the process out of the main memory to make room for a new process or for a process in Ready-Suspend state. Therefore, the OS must choose which process to replace.
Thus, when a page fault occurs, the OS has to change a page to remove from memory to make room for the page that must be brought in. If the page to be removed has been modified while in memory it must be written to disk to bring the disk copy up to date.
Replacement algorithms can affect the system's performance. Following are the three basic page replacement algorithms:

Least Recently Used Algorithm
This paging algorithm selects a page for replacement that has been unused for the longest time.

First-In-First-Out

Replace the page that has been in memory longest, is the policy applied by FIFO. Pages from memory are removed in round-robin fashion. Its advantage is it's simplicity.

Code-

Least Recently Used

```
#include<stdio.h>

int findLRU (int time[], int n)
{
  int i, minimum = time[0], pos = 0;

  for (i = 1; i < n; ++i)
    {
      if (time[i] < minimum)
         {
           minimum = time[i];
           pos = i;
         }
    }
  return pos;
}

int main ()
{
  int no_of_frames, no_of_pages, frames[10], pages[30], counter =
    0, time[10], flag1, flag2, i, j, pos, faults = 0;
printf ("Enter number of frames: ");   scanf
("%d", &no_of_frames);   printf ("Enter
number of pages: ");   scanf ("%d",
&no_of_pages);   printf ("Enter Pages in
Order: ");
  for (i = 0; i < no_of_pages; ++i)
    {
       printf("\nEnter : ")
      scanf ("%d", &pages[i]);
    }
```

```
  for (i = 0; i < no_of_frames; ++i)
    {
       // empty frames are identified by -1
      frames[i] = -1;
    }

  for (i = 0; i < no_of_pages; ++i)
    {
      flag1 = flag2 = 0;

      for (j = 0; j < no_of_frames; ++j)
         {
           if (frames[j] == pages[i])
             {          counter++;
      time[j] = counter;
flag1 = flag2 = 1;
break;             }
         }

      if (flag1 == 0)
         {
           for (j = 0; j < no_of_frames; ++j)
             {
               if (frames[j] == -1)
                  {
                    counter++;
  faults++;
    frames[j] = pages[i];     time[j] =
counter;
                    flag2 = 1;
  break;
                  }
             }
         }

      if (flag2 == 0)
         {
           pos = findLRU (time, no_of_frames);
           counter++;
faults++;       frames[pos] =
```

```
pages[i];       time[pos] =
counter;
        }

     printf ("\n");

     for (j = 0; j < no_of_frames; ++j)
        {
          printf ("%d\t", frames[j]);
        }
   }
  printf ("\n\nTotal Page Faults = %d", faults);

  return 0;
}
```

FiFO

```
#include <stdio.h>
int main()
{
int referenceString[10], pageFaults = 0, m, n, s, pages, frames;
printf("\nEnter the number of Pages:\t"); scanf("%d", &pages);
printf("\nEnter reference string values:\n");
for( m = 0; m < pages; m++)
{
   printf("Value No. [%d]:\t", m + 1);
   scanf("%d", &referenceString[m]);
}
printf("\n What are the total number of frames:\t");
{
   scanf("%d", &frames);
}
int temp[frames]; for(m = 0;
m < frames; m++)
{
  temp[m] = -1;
}
for(m = 0; m < pages; m++)
{   s =
0;
  for(n = 0; n < frames; n++)
```

```
  {
    if(referenceString[m] == temp[n])
      {          s++;
pageFaults--;
      }
  }
  pageFaults++;
  if((pageFaults <= frames) && (s == 0))
    {
      temp[m] = referenceString[m];
    }
  else if(s == 0)
    {
      temp[(pageFaults - 1) % frames] = referenceString[m];
    }
printf("\n");
    for(n = 0; n < frames; n++)
     {
       printf("%d\t", temp[n]);
     }
}
printf("\nTotal Page Faults:\t%d\n", pageFaults); return
0;
}
```

Output:

Least Recently Used:

FIFO:

```
Enter the number of Pages:      10

Enter reference string values:
Value No. [1]:  1
Value No. [2]:  2
Value No. [3]:  3
Value No. [4]:  4
Value No. [5]:  2
Value No. [6]:  1
Value No. [7]:  5
Value No. [8]:  3
Value No. [9]:  2
Value No. [10]: 4

 What are the total number of frames:   3

1       -1      -1
1       2       -1
1       2       3
4       2       3
4       2       3
4       1       3
4       1       5
3       1       5
3       2       5
3       2       4
Total Page Faults:      9
```

Conclusion: Implemented various page replacement policies.

Operating System

Experiment No. 10

Name: Ayush Jain
SAP ID: 60004200132
Batch: B2
Computer Engineering

Aim- Implement disk scheduling algorithm FCFS, SSTF, SCAN, CSCAN etc.

Problem Statement-

Perform following Disk Scheduling algorithms and also perform the Comparative Assessment of them 1) FCFS

2) SSTF

3) SCAN

4) CSCAN

5) LOOK

Description:

1) FCFS

All incoming requests are placed at the end of the queue. Whatever number that is next in the queue will be the next number served. Using this algorithm doesn't provide the best results. To determine the number of head movements you would simply find the number of tracks it took to move from one request to the next.

2) SSTF

In this case request is serviced according to next shortest distance. Starting at 50, the next shortest distance would be 62 instead of 34 since it is only 12 tracks away from 62 and 16 tracks away from 34. The process would continue until all the process are taken care of.

3) SCAN

This approach works like an elevator does. It scans down towards the nearest end and then when it hits the bottom it scans up servicing the requests that it didn't get going down. If a request comes in after it has been scanned it will not be serviced until the process comes back down or moves back up

4) CSCAN

Circular scanning works just like the elevator to some extent. It begins its scan toward the nearest end and works it way all the way to the end of the system. Once it hits the bottom or top it jumps to the other end and moves in the same direction.

5) LOOK

This is just an enhanced version of C-SCAN. In this the scanning doesn't go past the last request in the direction that it is moving. It too jumps to the other end but not all the way to the end. Just to the furthest request.

SSFT

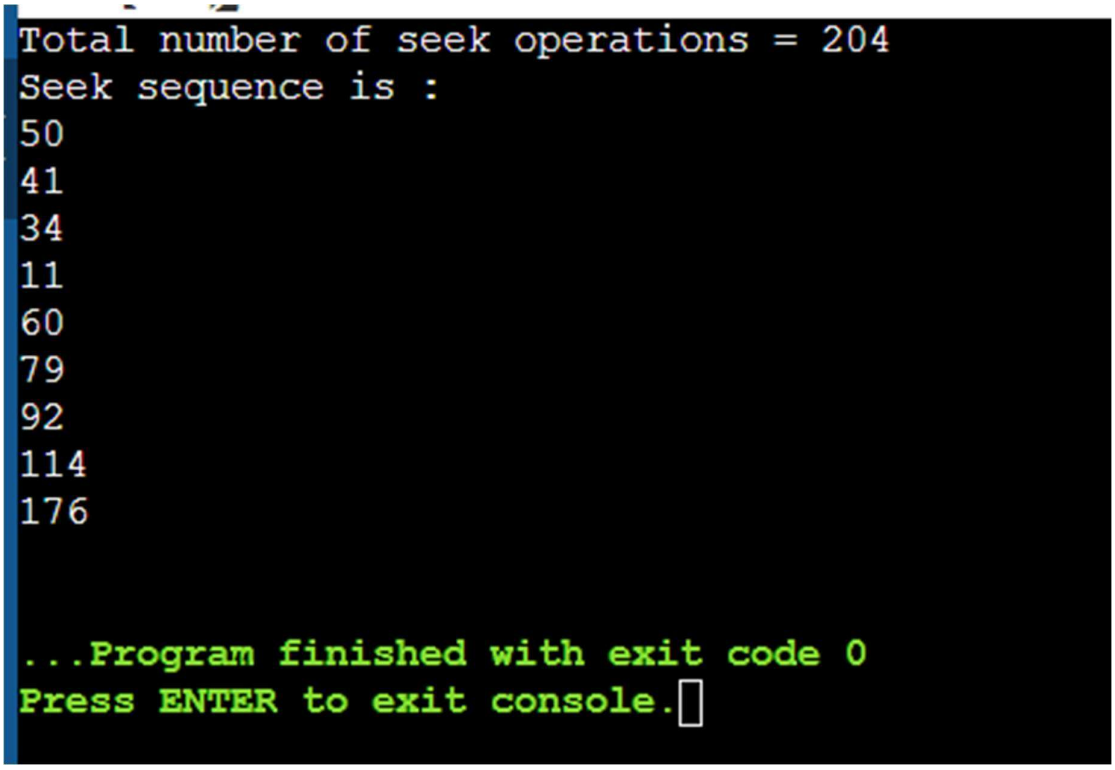
Code:

```
#include <bits/stdc++.h> using namespace std; void

calculatedifference(int request[], int head,int diff[][2], int n)

{     for(int i = 0; i < n;

i++)

    {         diff[i][0] = abs(head -

request[i]);

    }

} int findMIN(int diff[][2], int

n)

{     int index = -1;

int minimum = 1e9;


    for(int i = 0; i < n; i++)

    {

        if (!diff[i][1] && minimum > diff[i][0])

        {

            minimum = diff[i][0];

index = i;

        }

    }     return

index;
```

```
}

void shortestSeekTimeFirst(int request[],
                           int head, int n)
{     if (n == 0)     {         return;
}      int diff[n][2] = { { 0, 0 } };
int seekcount = 0;     int
seeksequence[n + 1] = {0};

   for(int i = 0; i < n; i++)
   {
      seeksequence[i] = head;
calculatedifference(request, head, diff, n);
int index = findMIN(diff, n);
diff[index][1] = 1;         seekcount +=
diff[index][0];         head = request[index];
   }
   seeksequence[n] = head;

   cout << "Total number of seek operations = "
        << seekcount << endl;     cout <<
"Seek sequence is : " << "\n";     for(int i
= 0; i <= n; i++)
   {
      cout << seeksequence[i] << "\n";
   }
} int main() {     int n = 8;     int proc[n] = { 176,
79, 34, 60, 92, 11, 41, 114 };
```

```
    shortestSeekTimeFirst(proc, 50, n);

return 0;

}
```

Output :



SCAN

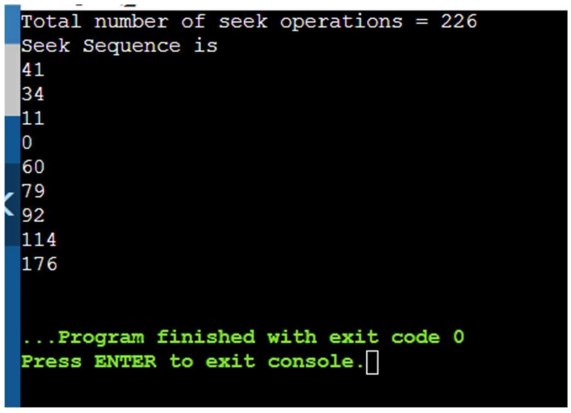
Code :

```
#include <bits/stdc++.h> using

namespace std;

int size = 8; int

disk_size = 200;

void SCAN(int arr[], int head, string direction)

{     int seek_count = 0;     int

distance, cur_track;     vector<int>

left, right;     vector<int>
```

```
seek_sequence;     if (direction ==
"left")         left.push_back(0);
else if (direction == "right")
right.push_back(disk_size - 1);
   for (int i = 0; i < size; i++) {
if (arr[i] < head)
left.push_back(arr[i]);         if
(arr[i] > head)
right.push_back(arr[i]);
   }     std::sort(left.begin(),
left.end());     std::sort(right.begin(),
right.end());     int run = 2;     while
(run--) {         if (direction == "left")
{             for (int i = left.size() - 1; i
>= 0; i--) {                 cur_track =
left[i];
seek_sequence.push_back(cur_track)
;                 distance = abs(cur_track -
head);                 seek_count +=
distance;                 head = cur_track;
           }             direction
= "right";
       }
    else if (direction == "right") {             for (int
i = 0; i < right.size(); i++) {                 cur_track
= right[i];
seek_sequence.push_back(cur_track);
```

```
distance = abs(cur_track - head);
seek_count += distance;                head =
cur_track;
          }
direction = "left";
       }
   }
  cout << "Total number of seek operations = "
        << seek_count << endl;
   cout << "Seek Sequence is" << endl;
   for (int i = 0; i < seek_sequence.size(); i++) {
cout << seek_sequence[i] << endl;
   }
} int main() {     int arr[size] = {
176, 79, 34, 60,                  92,
11, 41, 114 };     int head = 50;
string direction = "left";
   SCAN(arr, head, direction);
   return 0;
}
```

Output:



Name: Ayush Jain
SAP ID: 60004200132
Div: B
Computer Engineering

Operating Systems

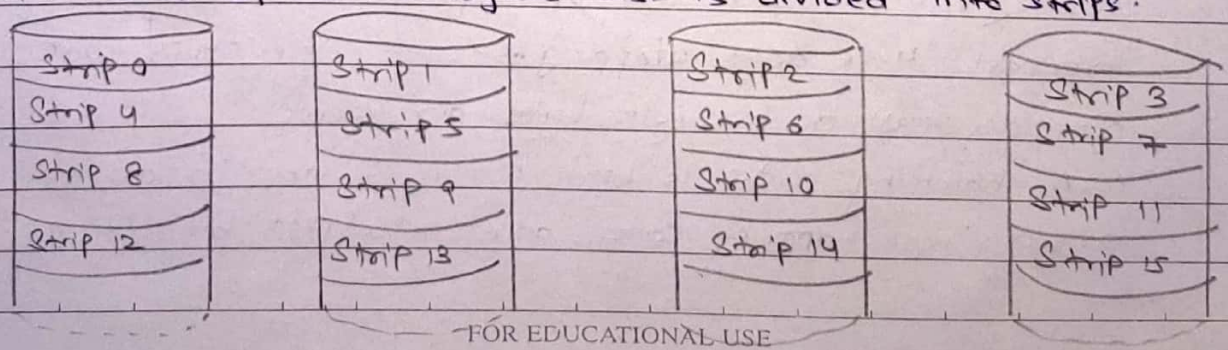
Assignment 1

1) Describe RAID disk management. Provide detailed description of all the levels of RAID.

→ 1) Raid is a set of physical disk drivers viewed by the operating system as a signal logical drive.
2) RAID is the acronym of Redundant Array of Independent DISK or Redundant array of Inexpensive Disks.
3) Here, the data are distributed across the physical drives of an array in a scheme known as stripping.
4) Redundant disk capacity is used to store parity information which guarantees data recoverability in case of disk failure.
5) RAID consists of seven levels:

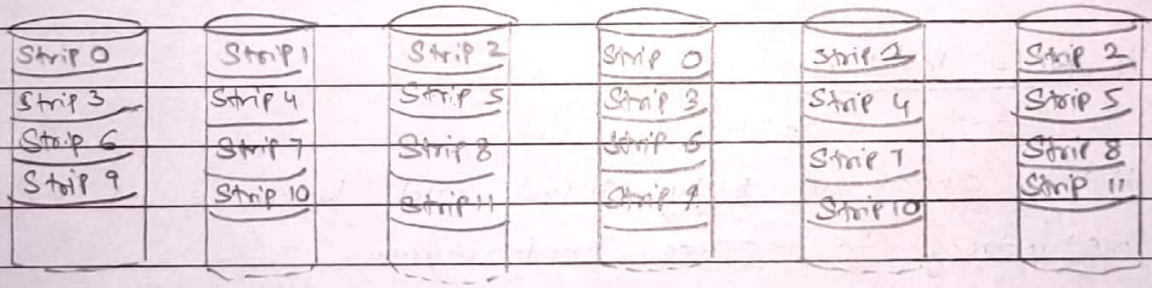
a) **RAID level 0**

i) This cannot be truly called raid because it does not include redundancy to improve performance.
ii) User and system data are distributed across all of the disk in the array.
iii) Here, there is no duplication of data. Hence a block once lost, cannot be recovered. This is used for parallel processing which increases speed. The logical disk is divided into strips.



RAID level 1

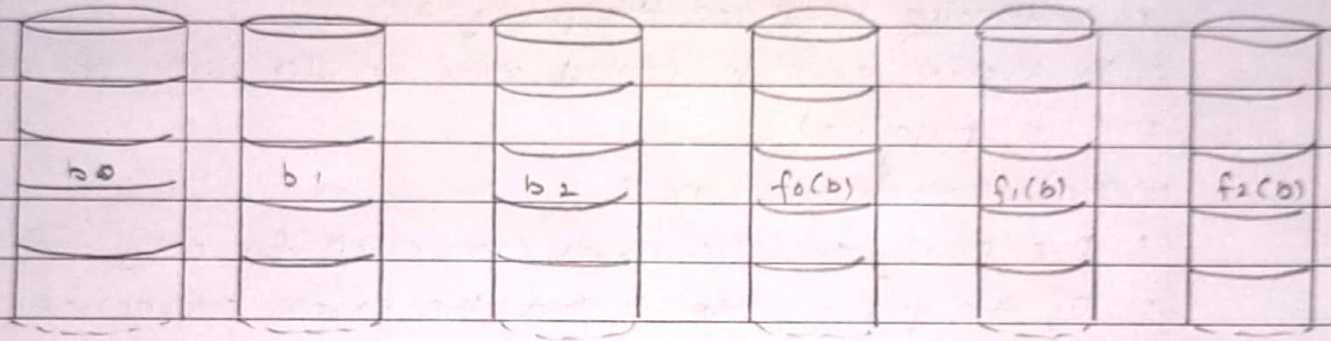
(1) Also known as mirrored disks as redundancy is achieved by the simple expedient of duplicating all the data.
(2) Here if one drive fails the data may be still accessed from the second drive. Here there is no 'write penalty' i.e. a write request requires that both corresponding strips be updated but this can be done in parallel.
(3) The principal disadvantage is cost as it requires twice the disk space of the logical device it supports.
(4) RAID 1 configuration is limited to drives that stores system software and data and other highly critical files. RAID 1 provides real time backup of all data so that is the event of a disk failure, all of the critical data is still immediately available.



RAID level 2

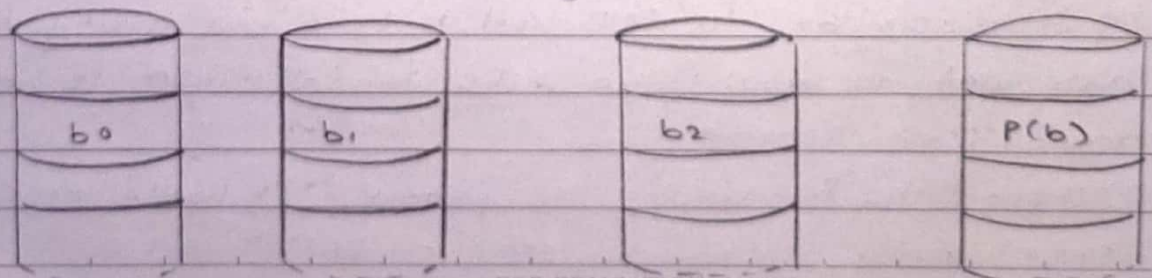
(1) It makes use of parallel access technique. In a parallel access array, all member disk participate in execution of every I/O request. Here data stripping is used. The strips are very small often as small as a single byte or word.
(2) A hamming code is used, which is able to correct single bit error. These hamming code are calculated bit-wise and stored

in the corresponding bit positions or multiple parity disk.
(3) RAID level 2 requires less disk than RAID level 1. If there is a single bit error, the controller can recognize and correct the error instantly so that read access time is not slow.
(4) RAID level 2 will only be effective choice in an environment in which multiple disk error occurs.



RAID level 3

(1) It is similar to RAID level 2. Here, the difference is that it requires only a single redundant disk, no matter the size of disk array.
(2) It computes a single parity bit for the set of individual bits in the same position on all the data disks.
(3) In the event of drive failure, the parity drive is accessed and data is reconstructed from the remaining drives. Once the filled drive is replace the missing data can be restored and operations are resumed. It can achieved very high data transfer rates.



RAID level 4

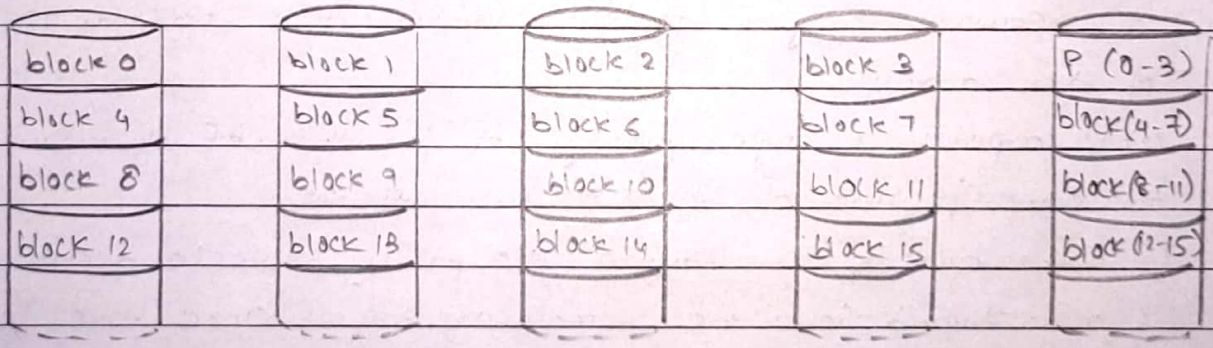
(1) Uses independent access technique. In independent access array each member disk operates seperately. Here the data is striped in blocks.

(2) A parity strip is calculated and this parity is stored in the corresponding strip on the parity disk.

(3) It involves a write penalty when an I/O write request of small size is performed.

(4) The disadvantages are :

1. The parity can give error correction for only one block.
2. If the parity disk is lost the error correction cannot be done.
3. Since the parity is given to one disk only, this disk is repeatedly used. This results in an overload and it may slow down.

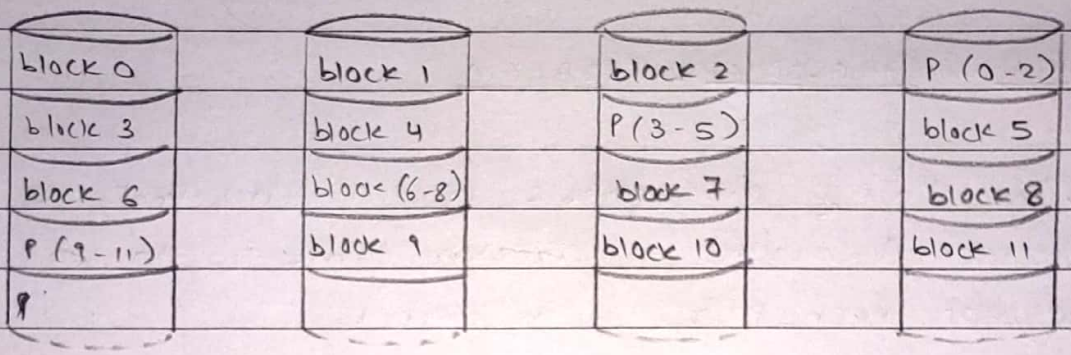


RAID level 5

(1) It is similar to RAID level 4. Here the parity strips are distributed all across the disks. The allocation is done by round-robin scheme.

(2) Those helps in avoiding the potential I/O bottle neck of the single parity disk, as seen in RAID level 4.

(3) It has the characteristic that the loss of any one disk does not result in data loss.

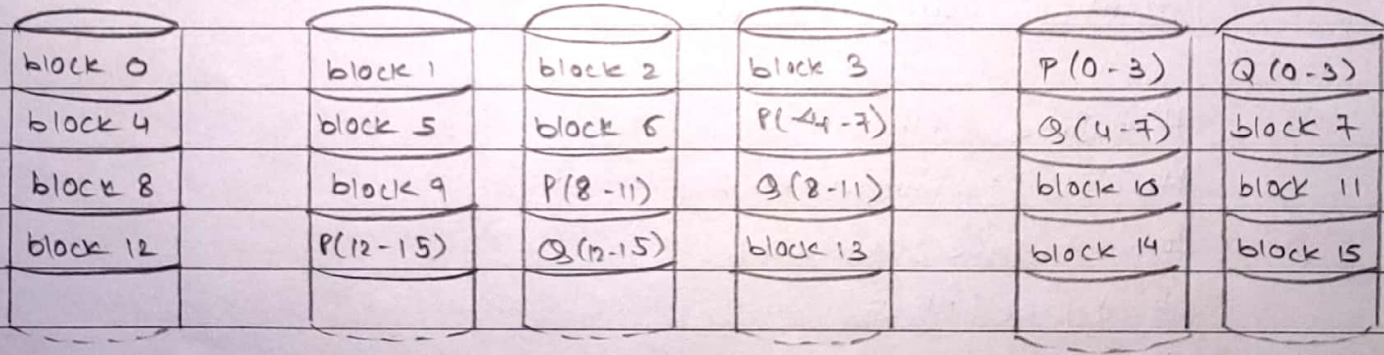


RAID level 6

(1) Here, two different parity calculations are carried out and stored in seperate blocks on different disks. Thus, if user data requires N disks, RAID level 6 will contain N+2 disks.

(2) The main advantage is that it provides extremely high data availability. It incurs a write penalty because each write affects two parity blocks.

(3) Presence of two parity strips helps in calculating for two errors.



Q. 2) Explain Dining Philosophers problem and its solution using semaphores.

→ DINING PHILOSOPHER'S PROBLEM

1) The dining philosopher's prolem states that K philosophers are seated around a circular table with one chopstick between each pair of philosophers.
2) To eat, a philosophers needs both their right and left chopstick. A philosopher can only eat if both immediate left and right chopsticks of the philosophers is available.
3) In case, if both immediate left and right chopsticks of the philosopher are not available then the philosopher puts down their chopstick and starts thinking again.
4) The dining philosophers problem is the classical prolem of synchronization and demostrates a large class of concurrency control problems.

Solution using Semaphores

```
# include <Pthread.h>
# include<semaphore.h>
# include <stdio.h>
# define N 5
# define thinking 2
# define hungry 1
# define eating 0
# define left (phnum + 4) %.N
# define right (phnum + 1) %. N
```

```
int state[N];
int phil[N] = {0,1,2,3,4};

sem_t mutex;
sem_t S[N];

void test (int phnum)
{
  if (state[phnum] == hungry && state[left] != eating
      && state[right] != eating)
  {
    state[phnum] = eating;
    sleep(2);
  Printf("Philosopher %d takes fork %d and %d \n", phnum+1, left+1,
                                                   phnum+1);
  Printf("Philosopher %d is Eating \n", phnum+1);
  sem_post(S[phnum]);
  }
}

 void take_fork (int phnum)
{
  sem_wait(&mutex);
  state[phnum] = hungry;
  Printf("Philosopher %d is hungry \n", phnum+1);
  test(phnum);
  sem_post(&mutex);
  sem_wait(&S[phnum]);
  sleep(1);
}
```

```
void put_fork (int phnum)
{
    sem_wait (&mutex);
    State[phnum] = thinking;
    Printf(" Philosopher %d putting fork %d and %d down \n" ;
                    phnum+1, left+1, phnum+1);
    Printf (" philosopher %d is thinking \n ", phnum+1);
    test (left);
    test (right);
    sem_post (& mutex);
}

void * philosopher (void * num)
{
    while (1)
    {
        int* i = num;
        sleep (1);
        take_fork (*i);
        sleep (0);
        Put_fork (*i);
    }
}

int main ()
{
    int i;
    Pthread_t . thread_id [N];
    sem_init (&mutex, 0, 1);
```

```
for(i=0; i<N; i++)
    sem_init(&S[i],0,0);
for(i=0; i<N; i++)
 {
    Pthread_create(&thread_id[i], NULL, Philosopher, &phil[i]);

   Printf("Philosopher %d is thinking \n"; i+1));
}
 for (i=0; i<N; i++)
    Pthread_join (thread_id[i], NULL);

}
```

Name: Ayush Jain
SAP ID: 60004200132
Div: B
Computer Engineering

OS: Assignment 2

Describe workings of following operating systems.

i) XV6 OS

→ 1) XV6 is a simple, UNIX like teaching operating system developed in the summer of 2006 by MIT.

2) It provides the basic interfaces introduced by Ken Thompson and Dennis Ritchie's UNIX operating system, as well as mimicking UNIX's internal design.

3) It provides Process management, synchronization, memory management, file management functionalities.

4) XV6 takes the traditional form of kernel, a speacial program that provides services to running programs. Each running program called a process, has memory containing instructions, data and a stack. The instructions implement the program's computation. The data are variables on which the computation acts. The stack organizes the program procedure calls.

5) The system call enters the kernel, the kernel performs the services and returns. Thus a process alternates between executing in user space and kernel space.

6) The kernel uses the CPU's hardware protection mechanisms to ensure that each process executing in user space can access only its own memory.

7) The kernel executes with the hardware priviledges required to implement these protections, user program execute without these priviledges.

8) When a user program invokes a system call, the hardware raises the priviledge level and starts executing a pre-

arranged function in the kernel.

9) XV6 does not provides a nation of users or of protecting one user from another, in UNIX terms, all XV6 process run as root.

ii) Real Time OS

→ i) Real Time Os are used in environments where a large number of events, mostly external to computer system, must be accepted and processed in short time or within certain deadlines.

2) Such applications are individual control, telephone, switching equipment, flight control and real time simulations.

3) It can be of 2 types:

① Hard Real Time Operating System.

② Soft Real Time Operating System.

4) Hard Real Time Os: These operating system guarantee that critical task be completed within range of time.

⇒ Ex: air bag controls in cars.

5) Soft Real Time Os: This Os provides some relaxation in time input. Example: multimedia systems.

6) There are different types of basic functionalities of an RTOS are following.

1. Priority based Scheduling.
2. System clock interrupt routine.
3. Deterministic behaviours.
4. Synchronization and Messaging.
5. RTOS services.

→ Priority based Scheduling:

Multitasking operations is accompolished by scheduling process

for execution independently of each other. Each process is assigned a certain level of priority that corresponds to the relative importance of the event that it services. The processor is allocation to the highest priority process. This type of scheduling is called priority based scheduling.

7) System clock Interrupt Routine:

To perform the time sensitive operations the RTOS will provide some sort of system ~~cotts~~ clocks.

8) Deterministic Behaviour:

The RTOS move to great length to protect that whether you have taken 100 tasks or 10 tasks, it does not make any difference in the distance to switch context and it determine the next highest priority tasks.

9) Sychronization and Messaging:

Messaging provides a means of communication with other system and between the tasks. The messaging services includes:

(1) Semaphores.
(2) Event flags.
(3) Mailboxes.
(4) Pipes
(5) Message queues.

10) RTOS services:

The most important part of OS is the Kernel. As tasks cannot acquire CPU attention all the time, the Kernel must also provide some more services. These include:

(1) Interrupt handling services.
(2) Time services.
(3) Device Management services.
(4) Memory Management Services
(5) Input Output Services.

(iii) Mobile OS

→ 1) A mobile OS is an operating system that provides to run other application softwares on mobile devices. It is same kind of software as the famous computer operating system title Linux and windows, but they are light and simple to some extent.

2) A mobile OS typically starts up when a devices powers on presenting a screen with icons or titles that present info and provide application access. Mobile OS also manage cellular and wireless network connectivity, as well as phone access.

3) The OS found on smart phones include symbion OS, iphone OS, RIM's Blackberry, Android, Windows.

4) Mobile operating system delivers various features to users and the distinguishing feature that mobile OS offers is the ~~availability~~ ability to connect to the internet via the smartphones built in modem and a wireless service provider such as verizon and AT&T.

5) Many mobile operating system offers a native web browser application, which allows users to search the internet and visit webpages.

6) Several mobile operating system have native GPS (global positioning system) application that allows users to search to locations, follow step by step directions and share locations with other devices.