

---

**NAME: JUNAID GIRKAR**

**SUBJECT: OPERATING SYSTEMS**

**ANNUAL YEAR: 2020 - 2021**

**SAP ID: 60004190057**

**BRANCH: COMPUTER ENGINEERING**

**DIVISION: A (A-3)**

**SEMESTER: 4**

---

# INDEX

EXP No.	AIM	PAGE No.
1	Explore the internal commands of linux and Write shell scripts	2
2	System calls for file manipulation	14
3	Building multi-threaded and multi-process applications	19
4	CPU scheduling algorithms like FCFS, SJF, Round Robin etc	28
5	Process and Thread Synchronisation using client server mechanism	57
6	Implement the above-described problem using semaphores or mutexes along with threads.	65
7	Implement order scheduling in supply chain using Banker's Algorithm	75
8	Using the CPU-OS simulator analyze and synthesize the following	85
9	Implement various page replacement policies	110
10	Implement disk scheduling algorithm FCFS, SSTF, SCAN, CSCAN etc.	128

# EXPERIMENT 1

---

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

1. Display top 10 processes in descending order
2. Display processes with highest memory usage.
3. Display current logged in user and no. of users.
4. Display current shell, home directory, operating system type, current working directory.
5. Display OS version, release number.
6. Illustrate the use of sort, grep, awk, etc.

## **LINUX:**

Linux is a community of open-source Unix like operating systems that are based on the Linux Kernel. It was initially released by **Linus Torvalds** on September 17, 1991. It is a free and open-source operating system and the source code can be modified and distributed to anyone commercially or non commercially under the GNU General Public License.

Initially, Linux was created for personal computers and gradually it was used in other machines like servers, mainframe computers, supercomputers, etc. Nowadays, Linux is also used in embedded systems like routers, automation controls, televisions, digital video recorders, video game consoles, smartwatches, etc. The biggest success of Linux is Android(operating system) it is based on the Linux kernel that is running on smartphones and tablets. Due to android Linux has the largest installed base of all general-purpose operating systems. Linux is generally packaged in a Linux distribution.

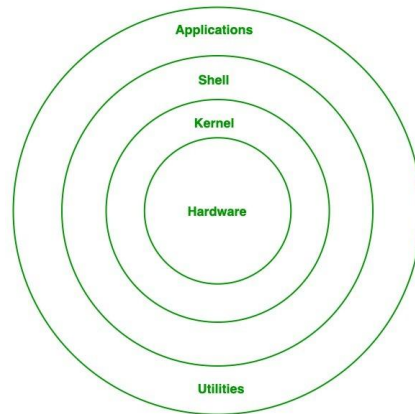
## **Linux Distribution**

Linux distribution is an operating system that is made up of a collection of software based on Linux kernel or you can say distribution contains the Linux kernel and supporting libraries and software. And you can get a Linux based operating system by downloading one of the Linux distributions and these distributions are available for different types of devices like embedded devices, personal computers, etc. Around 600 + Linux Distributions are available and some of the popular Linux distributions are:

- MX Linux
- Manjaro
- Linux Mint
- elementary
- Ubuntu
- Debian
- Solus
- Fedora
- openSUSE
- Deepin

## **Architecture of Linux**

Linux architecture has the following components:



1. **Kernel:** Kernel is the core of the Linux based operating system. It virtualizes the common hardware resources of the computer to provide each process with its virtual resources. This makes the process seem as if it is the sole process running on the machine. The kernel is also responsible for preventing and mitigating conflicts between different processes. Different types of the kernel are:
  - Monolithic Kernel
  - Hybrid kernels
  - Exo kernels
  - Micro kernels
2. **System Library:** Is The special types of functions that are used to implement the functionality of the operating system.
3. **Shell:** It is an interface to the kernel which hides the complexity of the kernel's functions

from the users. It takes commands from the user and executes the kernel's functions.

4. **Hardware Layer:** This layer consists all peripheral devices like RAM/ HDD/ CPU etc.
5. **System Utility:** It provides the functionalities of an operating system to the user.

### **Advantages of Linux**

- The main advantage of Linux, is it is an open-source operating system. This means the source code is easily available for everyone and you are allowed to contribute, modify and distribute the code to anyone without any permissions.
- In terms of security, Linux is more secure than any other operating system. It does not mean that Linux is 100 percent secure it has some malware for it but is less vulnerable than any other operating system. So, it does not require any anti-virus software.
- The software updates in Linux are easy and frequent.
- Various Linux distributions are available so that you can use them according to your requirements or according to your taste.

- Linux is freely available to use on the internet.
- It has large community support.
- It provides high stability. It rarely slows down or freezes and there is no need to reboot it after a short time.
- It maintain the privacy of the user.
- The performance of the Linux system is much higher than other operating systems. It allows a large number of people to work at the same time and it handles them efficiently.
- It is network friendly.
- The flexibility of Linux is high. There is no need to install a complete Linux suit; you are allowed to install only required components.
- Linux is compatible with a large number of file formats.
- It is fast and easy to install from the web. It can also install on any hardware even on your old computer system.
- It performs all tasks properly even if it has limited space on the hard disk.

### **Disadvantages of Linux**

- It is not very user-friendly. So, it may be confusing for beginners.

- It has small peripheral hardware drivers as compared to windows.

## 1. Display top 10 processes in descending order

**ps command** is used to list the currently running processes and their PIDs along with some other information depends on different options. It reads the process information from the virtual files in /proc file-system. /proc contains virtual files, this is the reason it's referred as a virtual file system.

### ps [options]

```

jarvis@linuxconfig: ~/Desktop/OS-Exp1
jarvis@linuxconfig:~/Desktop$ mkdir OS-Exp1
jarvis@linuxconfig:~/Desktop$ cd OS-Exp1/
jarvis@linuxconfig:~/Desktop/OS-Exp1$ echo "Top 10 processes in descending order"
Top 10 processes in descending order
jarvis@linuxconfig:~/Desktop/OS-Exp1$ ps axl|head -n 10

```

F	UID	PID	PPID	PRI	NI	VSZ	RSS	WCHAN	STAT	TTY	TIME	COMMAND
4	0	1	0	20	0	167788	11792	-	Ss	?	0:01	/sbin/init splash
1	0	2	0	20	0	0	0	-	S	?	0:00	[kthreadd]
1	0	3	2	0	-20	0	0	-	I<	?	0:00	[rcu_gp]
1	0	4	2	0	-20	0	0	-	I<	?	0:00	[rcu_par_gp]
1	0	6	2	0	-20	0	0	-	I<	?	0:00	[kworker/0:0H-kblockd]
1	0	8	2	0	-20	0	0	-	I<	?	0:00	[mm_percpu_wq]
1	0	9	2	20	0	0	0	-	S	?	0:00	[ksoftirqd/0]
1	0	10	2	20	0	0	0	-	I	?	0:00	[rcu_sched]
1	0	11	2	-100	-	0	0	-	S	?	0:00	[migration/0]

```

jarvis@linuxconfig:~/Desktop/OS-Exp1$

```

## 2. Display processes with highest memory usage.



```
jarvis@linuxconfig: ~/Desktop/OS-Exp1
jarvis@linuxconfig:~/Desktop/OS-Exp1$ echo " Display Processes with highest memory usage"
Display Processes with highest memory usage
jarvis@linuxconfig:~/Desktop/OS-Exp1$ ps -eopid,ppid,cmd,%mem,%cpu --sort=%mem |head
  PID   PPID  CMD      %MEM %CPU
    2      0 [kthreadd]    0.0  0.0
    3      2 [rcu_gp]      0.0  0.0
    4      2 [rcu_par_gp]  0.0  0.0
    6      2 [kworker/0:0H-kblockd] 0.0  0.0
    8      2 [mm_percpu_wq] 0.0  0.0
    9      2 [ksoftirqd/0] 0.0  0.0
   10      2 [rcu_sched]   0.0  0.0
   11      2 [migration/0] 0.0  0.0
   12      2 [idle_inject/0] 0.0  0.0
jarvis@linuxconfig:~/Desktop/OS-Exp1$
```

3. Display current logged in user and no. of users.

**who** command is used to find out the following information :

1. Time of last system boot
2. Current run level of the system
3. List of logged in users and more.

Syntax : **\$who** [options] [filename]

```
jarvis@linuxconfig: ~/Desktop/OS-Exp1
jarvis@linuxconfig:~/Desktop/OS-Exp1$ echo "Display Current logged in User"
Display Current logged in User
jarvis@linuxconfig:~/Desktop/OS-Exp1$ who -u
jarvis  :0                2021-04-21 11:18  ?                1043 (:0)
jarvis@linuxconfig:~/Desktop/OS-Exp1$
jarvis@linuxconfig:~/Desktop/OS-Exp1$ echo "Display Number of logged in users"
Display Number of logged in users
jarvis@linuxconfig:~/Desktop/OS-Exp1$ who -u|wc -l
1
jarvis@linuxconfig:~/Desktop/OS-Exp1$
```

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

1. **whoami** command is used both in *Unix Operating System* and as well as in *Windows Operating System*.
  - It is basically the concatenation of the strings **"who", "am", "i"** as **whoami**.
  - It displays the username of the current user when this command is invoked.
  - It is similar as running the **id** command with the options **-un**.

Syntax : **\$whoami**

2. The command '**uname**' displays the information about the system.

Syntax: **\$uname** [OPTION]

3. **pwd** stands for **Print Working Directory**. It prints the path of the working directory, starting from the root.

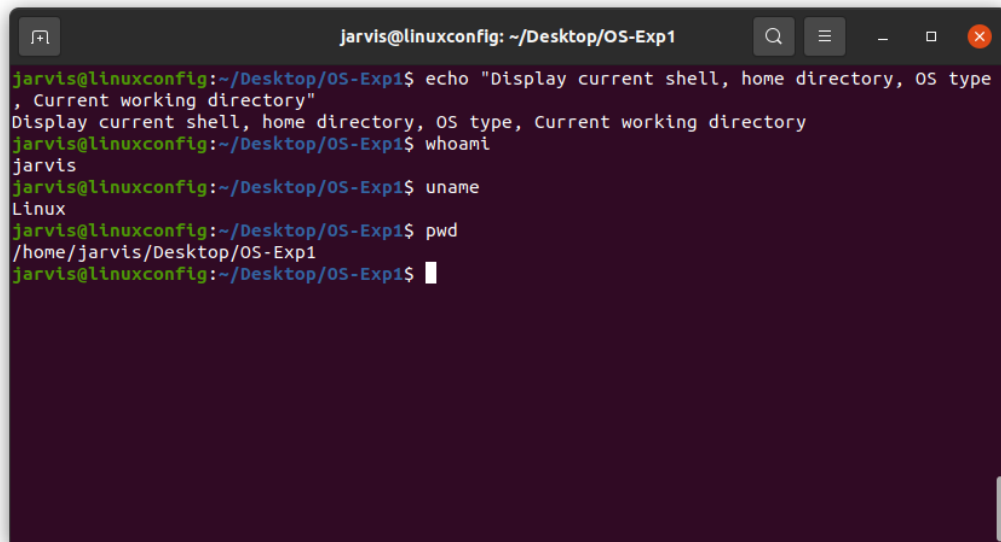
**pwd** is shell built-in command(**pwd**) or an actual binary(**/bin/pwd**).

**\$PWD** is an [environment variable](#) which stores the path of the current directory.

This command has two flags.

**\$pwd -L**: Prints the symbolic path.

**\$pwd -P**: Prints the actual path.

A terminal window titled 'jarvis@linuxconfig: ~/Desktop/OS-Exp1' with standard window controls. The terminal shows a series of commands and their outputs: 'echo "Display current shell, home directory, OS type, Current working directory"' outputs the same text; 'whoami' outputs 'jarvis'; 'uname' outputs 'Linux'; and 'pwd' outputs '/home/jarvis/Desktop/OS-Exp1'. The prompt is always 'jarvis@linuxconfig:~/Desktop/OS-Exp1\$'.

```
jarvis@linuxconfig:~/Desktop/OS-Exp1$ echo "Display current shell, home directory, OS type, Current working directory"
Display current shell, home directory, OS type, Current working directory
jarvis@linuxconfig:~/Desktop/OS-Exp1$ whoami
jarvis
jarvis@linuxconfig:~/Desktop/OS-Exp1$ uname
Linux
jarvis@linuxconfig:~/Desktop/OS-Exp1$ pwd
/home/jarvis/Desktop/OS-Exp1
jarvis@linuxconfig:~/Desktop/OS-Exp1$
```

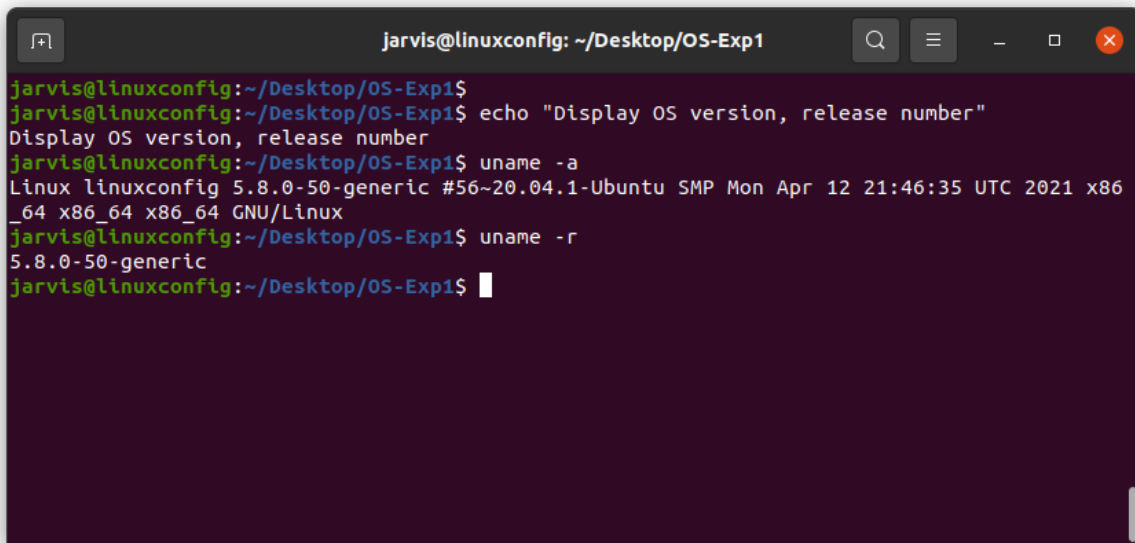
5. Display OS version, release number.

The command 'uname' displays the information about the system.

**Syntax: uname [OPTION]**

#### OPTIONS:

- a option: It prints all the system information
- s option: It prints the kernel name.
- n option: It prints the hostname of the network node
- r option: It prints the kernel release date
- v option: It prints the version of the current kernel.
- m option: It prints the machine hardware name.
- p option: It prints the **type** of the processor.
- i option: It prints the platform of the hardware.
- o option: It prints the name of the operating system.

A terminal window titled 'jarvis@linuxconfig: ~/Desktop/OS-Exp1' with standard window controls. The terminal shows a series of commands and their outputs: a prompt, an echo command, the 'uname -a' command showing detailed system information, and the 'uname -r' command showing the kernel version.

```
jarvis@linuxconfig:~/Desktop/OS-Exp1$  
jarvis@linuxconfig:~/Desktop/OS-Exp1$ echo "Display OS version, release number"  
Display OS version, release number  
jarvis@linuxconfig:~/Desktop/OS-Exp1$ uname -a  
Linux linuxconfig 5.8.0-50-generic #56~20.04.1-Ubuntu SMP Mon Apr 12 21:46:35 UTC 2021 x86_64 x86_64 x86_64 GNU/Linux  
jarvis@linuxconfig:~/Desktop/OS-Exp1$ uname -r  
5.8.0-50-generic  
jarvis@linuxconfig:~/Desktop/OS-Exp1$
```

6. Illustrate the use of sort, grep, awk, etc.

#### **SORT:**

SORT command is used to sort a file, arranging the records in a particular order. By default, the sort command sorts file assuming the contents are ASCII. Using options in sort command, it can also be used to sort numerically.

**The sort command follows these features as stated below:**

- 1.Lines starting with a number will appear before lines starting with a letter.
- 2.Lines starting with a letter that appears earlier in the alphabet will appear before lines starting with a letter that appears later in the alphabet.
- 3.Lines starting with a lowercase letter will appear before lines starting with the same letter in uppercase.

```
Syntax : $ sort filename.txt
```

### **GREP:**

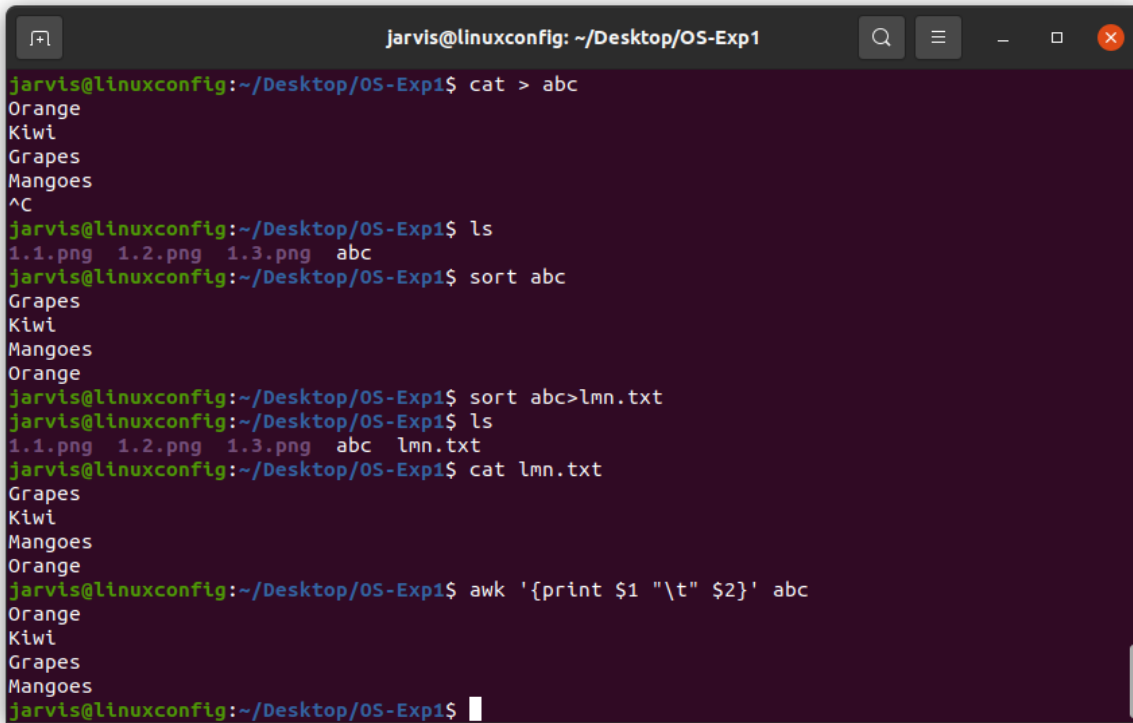
The grep filter searches a file for a particular pattern of characters, and displays all lines that contain that pattern. The pattern that is searched in the file is referred to as the regular expression (grep stands for globally search for regular expression and print out).

```
Syntax: grep [options] pattern [files]
```

### **AWK:**

Awk is a utility that enables a programmer to write tiny but effective programs in the form of statements that define text patterns that are to be searched for in each line of a document and the action that is to be taken when a match is found within a line. Awk is mostly used for pattern scanning and processing. It searches one or more files to see if they contain lines that matches with the specified patterns and then performs the associated actions.

Syntax: `awk options 'selection _criteria {action }'`  
`input-file > output-file`



```
jarvis@linuxconfig: ~/Desktop/OS-Exp1
jarvis@linuxconfig:~/Desktop/OS-Exp1$ cat > abc
Orange
Kiwi
Grapes
Mangoes
^C
jarvis@linuxconfig:~/Desktop/OS-Exp1$ ls
1.1.png 1.2.png 1.3.png abc
jarvis@linuxconfig:~/Desktop/OS-Exp1$ sort abc
Grapes
Kiwi
Mangoes
Orange
jarvis@linuxconfig:~/Desktop/OS-Exp1$ sort abc>lmn.txt
jarvis@linuxconfig:~/Desktop/OS-Exp1$ ls
1.1.png 1.2.png 1.3.png abc lmn.txt
jarvis@linuxconfig:~/Desktop/OS-Exp1$ cat lmn.txt
Grapes
Kiwi
Mangoes
Orange
jarvis@linuxconfig:~/Desktop/OS-Exp1$ awk '{print $1 "\t" $2}' abc
Orange
Kiwi
Grapes
Mangoes
jarvis@linuxconfig:~/Desktop/OS-Exp1$
```

**CONCLUSION:** We learned a few linux commands, their syntax and implemented them from the linux terminal.

# EXPERIMENT - 2

---

## 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:

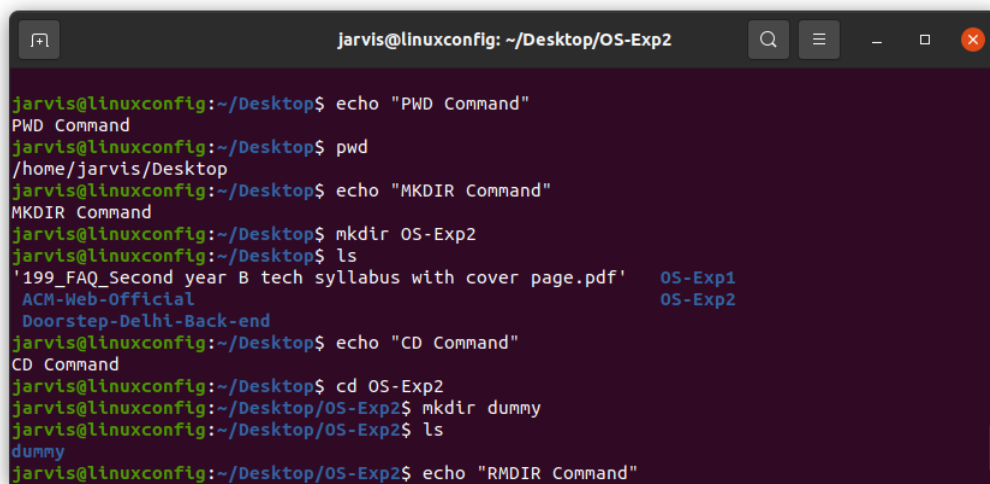
```
$ cd directory_name
```

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
```



```
jarvis@linuxconfig: ~/Desktop/OS-Exp2
jarvis@linuxconfig:~/Desktop$ echo "PWD Command"
PWD Command
jarvis@linuxconfig:~/Desktop$ pwd
/home/jarvis/Desktop
jarvis@linuxconfig:~/Desktop$ echo "MKDIR Command"
MKDIR Command
jarvis@linuxconfig:~/Desktop$ mkdir OS-Exp2
jarvis@linuxconfig:~/Desktop$ ls
'199_FAQ_Second year B tech syllabus with cover page.pdf'  OS-Exp1
ACM-Web-Official                                           OS-Exp2
Doorstep-Delhi-Back-end
jarvis@linuxconfig:~/Desktop$ echo "CD Command"
CD Command
jarvis@linuxconfig:~/Desktop$ cd OS-Exp2
jarvis@linuxconfig:~/Desktop/OS-Exp2$ mkdir dummy
jarvis@linuxconfig:~/Desktop/OS-Exp2$ ls
dummy
jarvis@linuxconfig:~/Desktop/OS-Exp2$ echo "RMDIR Command"
```



## 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
```



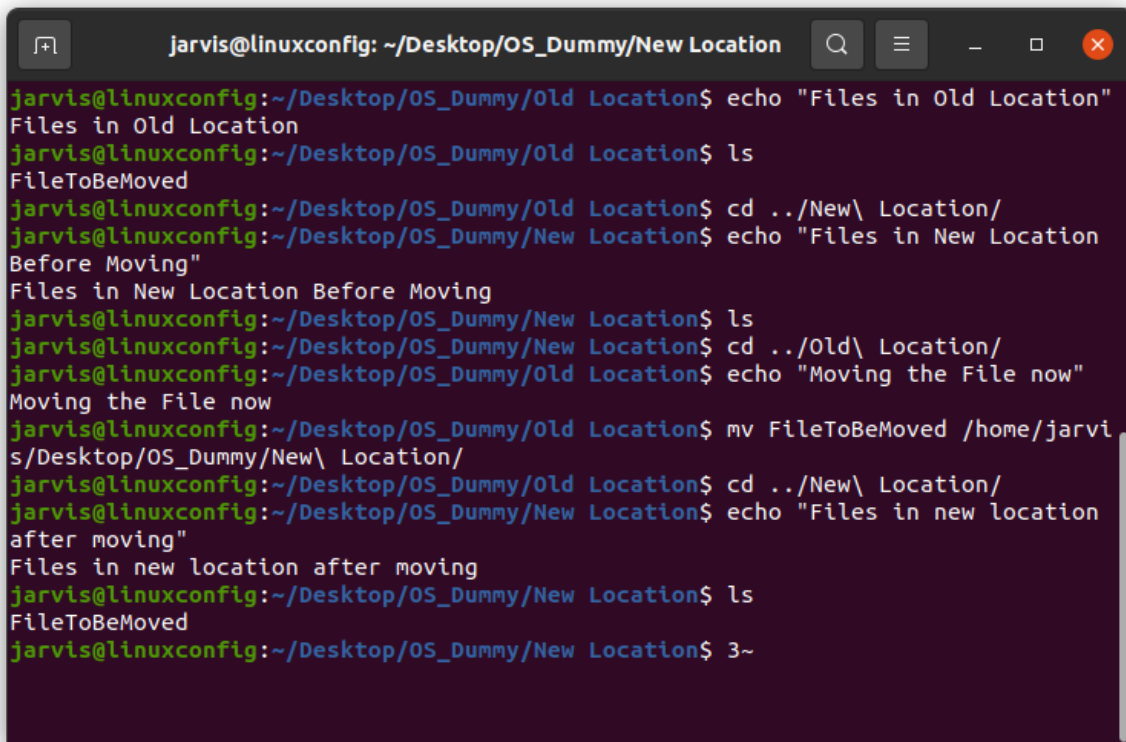
```
jarvis@linuxconfig: ~/Desktop/OS-Exp2
jarvis@linuxconfig:~/Desktop/OS-Exp2$ echo "RMDIR Command"
RMDIR Command
jarvis@linuxconfig:~/Desktop/OS-Exp2$ rmdir dummy
jarvis@linuxconfig:~/Desktop/OS-Exp2$ ls
jarvis@linuxconfig:~/Desktop/OS-Exp2$ echo "Touch Command"
Touch Command
jarvis@linuxconfig:~/Desktop/OS-Exp2$ touch dummy.txt
jarvis@linuxconfig:~/Desktop/OS-Exp2$ cat dummy.txt
This
Is
Dummy
Data
jarvis@linuxconfig:~/Desktop/OS-Exp2$
```

## 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/
```

A terminal window titled 'jarvis@linuxconfig: ~/Desktop/OS\_Dummy/New Location' showing a series of commands and their outputs. The user first navigates to the 'Old Location' and lists its contents, showing a file named 'FileToBeMoved'. Then, they navigate to the 'New Location' and list its contents, which are empty. They then execute the 'mv' command to move 'FileToBeMoved' from the old location to the new location. Finally, they navigate back to the 'New Location' and list its contents, showing that 'FileToBeMoved' has been successfully moved there.

```
jarvis@linuxconfig:~/Desktop/OS_Dummy/Old Location$ echo "Files in Old Location"
Files in Old Location
jarvis@linuxconfig:~/Desktop/OS_Dummy/Old Location$ ls
FileToBeMoved
jarvis@linuxconfig:~/Desktop/OS_Dummy/Old Location$ cd ../New\ Location/
jarvis@linuxconfig:~/Desktop/OS_Dummy/New Location$ echo "Files in New Location
Before Moving"
Files in New Location Before Moving
jarvis@linuxconfig:~/Desktop/OS_Dummy/New Location$ ls
jarvis@linuxconfig:~/Desktop/OS_Dummy/New Location$ cd ../Old\ Location/
jarvis@linuxconfig:~/Desktop/OS_Dummy/Old Location$ echo "Moving the File now"
Moving the File now
jarvis@linuxconfig:~/Desktop/OS_Dummy/Old Location$ mv FileToBeMoved /home/jarvis/Desktop/OS_Dummy/New\ Location/
jarvis@linuxconfig:~/Desktop/OS_Dummy/Old Location$ cd ../New\ Location/
jarvis@linuxconfig:~/Desktop/OS_Dummy/New Location$ echo "Files in new location
after moving"
Files in new location after moving
jarvis@linuxconfig:~/Desktop/OS_Dummy/New Location$ ls
FileToBeMoved
jarvis@linuxconfig:~/Desktop/OS_Dummy/New Location$ 3~
```

## 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.

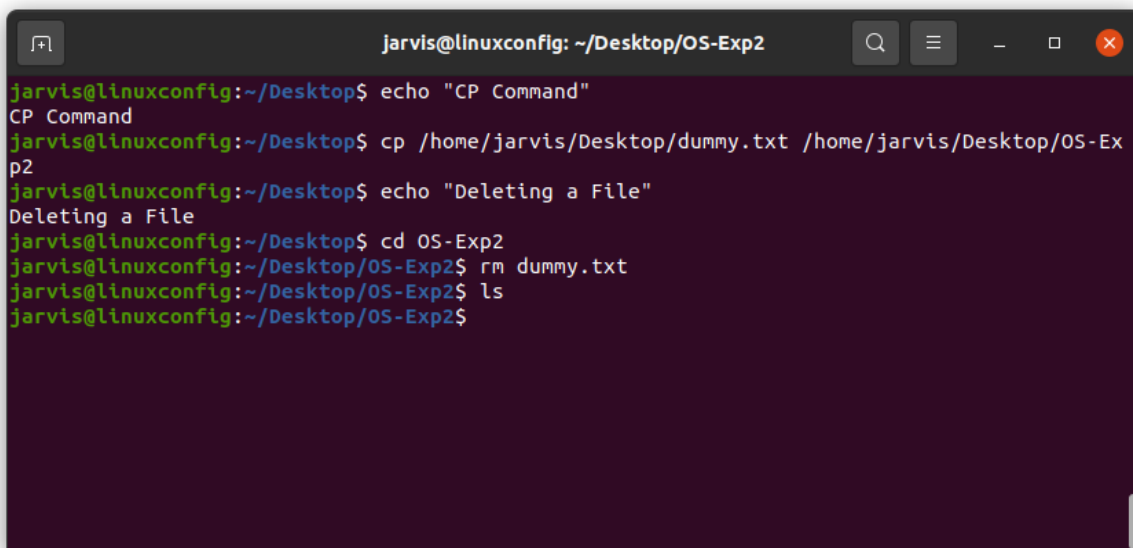
The syntax for copying a file is shown below.

```
$ 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
```

A terminal window titled 'jarvis@linuxconfig: ~/Desktop/OS-Exp2' with standard window controls. The terminal shows a sequence of commands and their outputs: 'echo "CP Command"' outputs 'CP Command'; 'cp /home/jarvis/Desktop/dummy.txt /home/jarvis/Desktop/OS-Exp2' is executed; 'echo "Deleting a File"' outputs 'Deleting a File'; 'cd OS-Exp2' changes the directory; 'rm dummy.txt' deletes the file; and 'ls' shows the directory contents. The prompt changes from '~/' to '~/Desktop/OS-Exp2' after the 'cd' command.

```
jarvis@linuxconfig:~/Desktop$ echo "CP Command"
CP Command
jarvis@linuxconfig:~/Desktop$ cp /home/jarvis/Desktop/dummy.txt /home/jarvis/Desktop/OS-Exp2
jarvis@linuxconfig:~/Desktop$ echo "Deleting a File"
Deleting a File
jarvis@linuxconfig:~/Desktop$ cd OS-Exp2
jarvis@linuxconfig:~/Desktop/OS-Exp2$ rm dummy.txt
jarvis@linuxconfig:~/Desktop/OS-Exp2$ ls
jarvis@linuxconfig:~/Desktop/OS-Exp2$
```

**CONCLUSION:** We learnt about linux commands for file management, their syntax and also implemented these commands.

# EXPERIMENT - 3

---

**AIM:** Building multi-threaded and multi-process applications

**THEORY:-**

**Multiprocessing:**

A multiprocessing system has more than two processors. The CPUs are added to the system that helps to increase the computing speed of the system. Every CPU has its own set of registers and main memory.

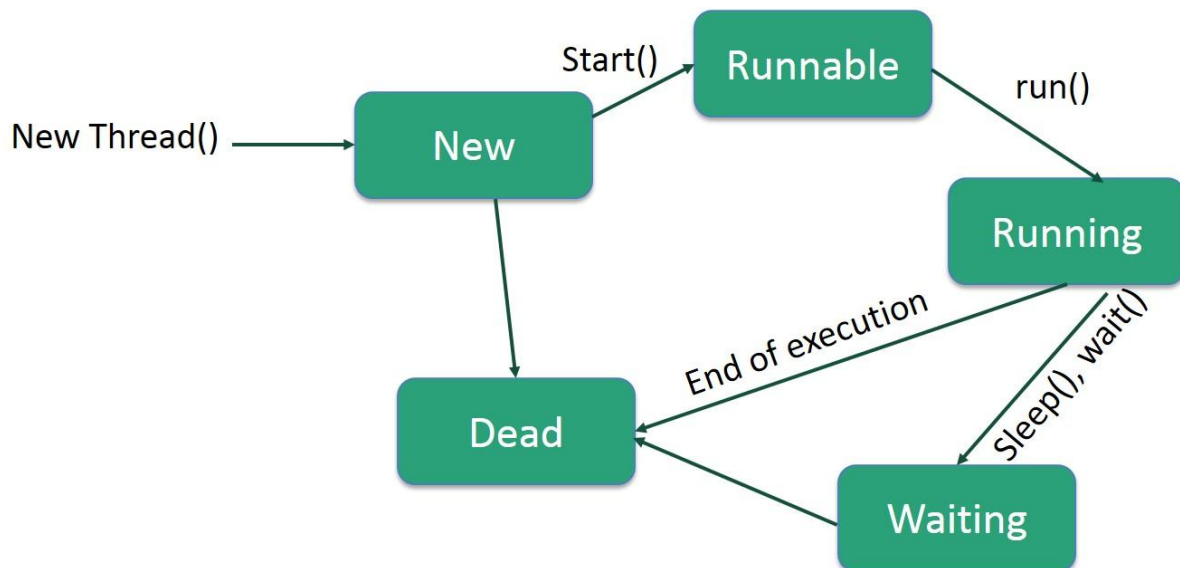
However, because each CPU are separate, it may happen that one CPU may not have anything to process. One processor may sit idle, and the other may be overloaded with the specific processes. In such a case, the process and resources are shared dynamically among the processors.

**Multithreading:**

Multithreading is a program execution technique that allows a single process to have multiple code segments (like threads). It also runs concurrently within the "context" of that process. Multi-threaded applications are applications that have two or more threads that run concurrently. Therefore, it is also known as concurrency.

**Life Cycle of a Thread**

A thread goes through various stages in its life cycle. For example, a thread is born, starts, runs, and then dies. The following diagram shows the complete life cycle of a thread.



Following are the stages of the life cycle -

- New - A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a born thread.
- Runnable - After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task.
- Waiting - Sometimes, a thread transitions to the waiting state while the thread waits for another thread to perform a task. A thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.
- Timed Waiting - A runnable thread can enter the timed waiting state for a specified interval of time. A thread in this state transitions back to the runnable state when that time interval expires or when the event it is waiting for occurs.
- Terminated (Dead) - A runnable thread enters the terminated state when it completes its task or otherwise terminates.

## Thread Priorities

Every Java thread has a priority that helps the operating system determine the order in which threads are scheduled.

Threads with higher priority are more important to a program and should be allocated processor time before lower-priority threads. However, thread priorities cannot guarantee the order in which threads execute and are very much platform dependent.

## CREATING A THREAD:

A thread can be created in two ways:-

1. **Extending Thread class:** This approach provides more flexibility in handling multiple threads created using available methods in Thread class.
2. **Implementing a Runnable Interface:** This method provides an entry point for the thread and you will put your complete business logic inside this method.

## Characteristics of Multiprocessing

Here are the essential features of Multiprocessing:

- Multiprocessing is classified according to the way their memory is organized.
- Multiprocessing improves the reliability of the system
- Multiprocessing can improve performance by decomposing a program into parallel executable tasks.

## Characteristics of Multithreading

Here are important aspects of multithreading:

- In the multithreading process, each thread runs parallel to each other.
- Threads do not allow you to separate the memory area. Therefore it saves memory and offers a better application performance

## Advantage of Multiprocessing

- The biggest advantage of a multiprocessor system is that it helps you to get more work done in a shorter period.
- The code is usually straightforward.
- Takes advantage of multiple CPU & cores
- Helps you to avoid GIL limitations for CPython
- Remove synchronization primitives unless if you use shared memory.
- Child processes are mostly interruptible/killable
- It helps you to get work done in a shorter period.
- These types of systems should be used when very high speed is required to process a large volume of data.
- Multiprocessing systems save money compared to single processor systems as processors can share peripherals and power supplies.

## Disadvantage of Multiprocessing

- IPC(Inter-Process Communication) a quite complicated with more overhead
- Has a larger memory footprint

## Advantage of Multithreading

- Threads share the same address space
- Threads are lightweight which has a low memory footprint
- The cost of communication between threads is low.
- Access to memory state from another context is easier
- It allows you to make responsive UIs easily
- An ideal option for I/O-bound applications
- Takes lesser time to switch between two threads within the shared memory and time to terminate
- Threads are faster to start than processes and also faster in task-switching.
- All Threads share a process memory pool that is very beneficial.
- Takes lesser time to create a new thread in the existing process than a new process

## Disadvantage of multithreading

- Multithreading system is not interruptible/killable
- If not following a command queue and message pump model then manual use of synchronization needed which becomes a necessity
- Code is usually harder to understand and increases the potential for race conditions increases dramatically.

Parameter	Multiprocessing	Multithreading
Basic	Multiprocessing helps you to increase computing power.	Multithreading helps you to create computing threads of a single process to increase computing power.



Execution	It allows you to execute multiple processes concurrently.	Multiple threads of a single process are executed concurrently.
CPU switching	In Multiprocessing, CPU has to switch between multiple programs so that it looks like that multiple programs are running simultaneously.	In multithreading, CPU has to switch between multiple threads to make it appear that all threads are running simultaneously.
Creation	The creation of a process is slow and resource-specific.	The creation of a thread is economical in time and resource.
Classification	Multiprocessing can be symmetric or asymmetric.	Multithreading is not classified.
Memory	Multiprocessing allocates separate memory and resources for each process or program.	Multithreading threads belonging to the same process share the same memory and resources as that of the process.
Pickling objects	Multithreading avoids pickling.	Multiprocessing relies on pickling objects in memory to send to other processes.
Program	Multiprocessing system allows executing multiple programs and tasks.	Multithreading system executes multiple threads of the same or different processes.
Time taken	Less time is taken for job processing.	A moderate amount of time is taken for job processing.

## CODE:

```
class TicketCounter {
    private int availableSeats;

    TicketCounter(int availableSeats){
        this.availableSeats = availableSeats;
    }

    public synchronized void bookTicket(String pname, int
numberOfSeats) {
        System.out.println("Welcome to XYZ bus Service");
        System.out.println("No. of seats available :
"+availableSeats);
        if ((availableSeats >= numberOfSeats) && (numberOfSeats >
0)) {
            System.out.println("CONGRATS Mr./Ms. "+pname+" The
number of seats requested (" +numberOfSeats + " seats) are BOOKED");
            availableSeats = availableSeats- numberOfSeats;
        }
        else System.out.println("SORRY Mr./Ms. "+pname+" The
number of seats requested (" +numberOfSeats + " seats) are not
available");
        System.out.println();
    }
}

class TicketBookingThread extends Thread {

    private TicketCounter ticketCounter;
    private String passengerName;
    private int noOfSeatsToBook;

    public TicketBookingThread(TicketCounter ticketCounter,String
passengerName, int noOfSeatsToBook) {
        this.ticketCounter = ticketCounter;
        this.passengerName = passengerName;
        this.noOfSeatsToBook = noOfSeatsToBook;
    }
}
```

```

        public void run() {
            ticketCounter.bookTicket(passengerName, noOfSeatsToBook);
        }
    }

    public class Multithreading {

        public static void main(String[] args) {
            TicketCounter ticketCounter = new TicketCounter(6);
            TicketBookingThread t1 = new
TicketBookingThread(ticketCounter, "Person1", 2);
            TicketBookingThread t2 = new
TicketBookingThread(ticketCounter, "Person2", 3);
            TicketBookingThread t3 = new
TicketBookingThread(ticketCounter, "Person3", 2);

            t1.start();
            t2.start();
            t3.start();
        }
    }
}

```

## OUTPUT:

```

Welcome to XYZ bus Service
No. of seats available : 6
CONGRATS Mr./Ms. Person1 The number of seats requested (2 seats) are BOOKED

Welcome to XYZ bus Service
No. of seats available : 4
CONGRATS Mr./Ms. Person2 The number of seats requested (3 seats) are BOOKED

Welcome to XYZ bus Service
No. of seats available : 1
SORRY Mr./Ms. Person3 The number of seats requested (2 seats) are not available

```

## **CONCLUSION:**

We learnt the difference between multithreading and multiprocessing and the different methods that can be implemented in multithreading. For our experiment, I have extended the Thread class to implement multithreading in our bus ticket booking system.

# EXPERIMENT - 4

---

## Shortest Job First Non\_Preemptive Scheduling

### AIM:

Write a program to demonstrate Shortest Job First Non-Preemptive scheduling with gantt diagram.

### THEORY:

Shortest job first (SJF) or shortest job next, is a scheduling policy that selects the waiting process with the smallest execution time to execute next. SJN is a non-preemptive algorithm.

- Shortest Job first has the advantage of having a minimum average waiting time among all scheduling algorithms.
- It is a Greedy Algorithm.
- It may cause starvation if shorter processes keep coming. This problem can be solved using the concept of ageing.
- It is practically infeasible as the Operating System may not know burst time and therefore may not sort them. While it is not possible to predict execution time, several methods can be used to estimate the execution time for a job, such as a weighted average of previous execution times. SJF

can be used in specialized environments where accurate estimates of running time are available.

**Algorithm:**

1. Sort all the process according to the arrival time.
2. Then select that process which has minimum arrival time and minimum Burst time.
3. After completion of process make a pool of process which after till the completion of previous process and select that process among the pool which is having minimum Burst time.

**Advantages :**

Here, are pros/benefits of Non-preemptive Scheduling method:

- Offers low scheduling overhead
- Tends to offer high throughput
- It is conceptually very simple method
- Less computational resources need for Scheduling

**Disadvantages :**

Here, are cons/drawback of Non-Preemptive Scheduling method:

- It can lead to starvation especially for those real-time tasks
- Bugs can cause a machine to freeze up
- It can make real-time and priority Scheduling difficult
- Poor response time for processes

CODE :

```
import java.util.*;
import java.util.Scanner;
import java.io.*;
import java.time.*;

public class sjfNonPreemptive {

    static int[] at= new int[50] ;
    static int[] bt= new int[50] ;
    static int[] tat= new int[50] ;
    static int[] wt= new int[50] ;
    static int[] process_order = new int[50] ;
    static boolean[] process_status= new boolean[50] ; // false =>
executed

    static int[] processID = new int[50] ;
    static int num , total_tat=0 , total_wt=0 ;
    static double avg_tat=0.00 , avg_wt=0.00;

    public static void sortArrayByAT(int processID[] ,int at[] , int
bt[]){

        for (int i = 1; i <num; ++i) {
            int at_key = at[i];
            int bt_key = bt[i];
            int process_key = processID[i];

            int hole = i - 1;
            while (hole >= 0 && at[hole] > at_key) {
```

```

        at[hole + 1] = at[hole];
        bt[hole + 1] = bt[hole];
        processID[hole + 1] = processID[hole];
        hole = hole - 1;
    }
    at[hole + 1] = at_key;
    bt[hole + 1] = bt_key;
    processID[hole + 1] = process_key ;

}

}

    public static void sortArrayByProcessID(int processID[] ,int at[]
, int bt[] , int tat[] , int wt[]){

        for (int i = 1; i < num; ++i) {
            int at_key = at[i];
            int bt_key = bt[i];
            int tat_key = tat[i];
            int wt_key = wt[i];
            int process_key = processID[i];

            int hole = i - 1;
            while (hole >= 0 && processID[process_order[hole]] >
process_key) {
                at[hole + 1] = at[hole];
                bt[hole + 1] = bt[hole];
                wt[hole + 1] = wt[hole];
                tat[hole + 1] = tat[hole];
                processID[hole + 1] = processID[hole];
                hole = hole - 1;
            }
            at[hole + 1] = at_key;
            bt[hole + 1] = bt_key;
            wt[hole + 1] = wt_key;
            tat[hole + 1] = tat_key;
            processID[hole + 1] = process_key ;

        }

    }
}

```



```

    public static void printInputTable(int processID[] ,int at[] ,
int bt[]){

        System.out.println("\n \n*** The Input Table ***\n ") ;
        System.out.println("\nPID \t AT\t BT") ;

        for(int j=0 ; j<num ; j++){
            System.out.println(processID[j]+" \t " +at[j]+" \t " +
bt[j] ) ;
        }
        System.out.println() ;
    }

    public static void printOutputTable(int processID[] ,int at[] ,
int bt[] , int tat[], int wt[], int process_order[]){

        System.out.println("\n \n*** The Final Output Table *** \n ")
;
        System.out.println("\nPID \t AT\t BT\t TAT\t WT") ;

        for(int j=0 ; j<num ; j++){
            System.out.println(processID[process_order[j]]+" \t
"+at[process_order[j]]+" \t " + bt[process_order[j]]+" \t " +
tat[process_order[j]]+" \t " + wt[process_order[j]] ) ;
        }
        System.out.println() ;

        System.out.println("Average TurnAround Time : " + avg_tat) ;
        System.out.println("Average Waiting Time : " +avg_wt) ;

    }

    public static void ganttChart(int processID[] ,int switch_time[],
int process_order[]){
        int j=0 ;
        System.out.println("** GANTT CHART **\n") ;
        if(at[0] != 0){
            System.out.print(switch_time[j++]+" \t[NA]\t") ;
        }

        for(int i=0 ; i<num ; i++){
            // System.out.print(switch_time[j++]+" \t"+ processID[i] +" |
") ;

```

```

        System.out.print(switch_time[j++]+"\\t["+
processID[process_order[i]] +"]\\t") ;
    }
    System.out.print(switch_time[j]) ;

}

public static void CalcTAT_and_WT(int switch_time[] ,int at[] ,
int bt[], int process_order[]){

    int j=1 ;

    if(at[0] != 0){
        j++ ;
    }
    for(int i=0 ; i<num ; i++){
        tat[process_order[i]] = switch_time[j++] -
at[process_order[i]] ;
        wt[process_order[i]] =tat[process_order[i]] -
bt[process_order[i]] ;
        total_tat += tat[process_order[i]];
        total_wt += wt[process_order[i]];
    }

    avg_tat = (double)total_tat/num ;
    avg_wt = (double)total_wt/num ;
}

public static void SJFNP(int processID[],int at[] , int bt[]){
    int total_time=0 , j=0;
    int switch_time[] = new int[50] ;
    switch_time[j++] = 0;

    if(at[0] != 0){
        total_time = total_time + at[0] ;
        switch_time[j++] = total_time ;
    }

    int min_burst ;
    int index =0 ;

    for(int i=0 ; i<num ; i++){
        min_burst =999 ;
        for(int k=0 ; k<num ; k++){

```

```

        if((process_status[k] == true) && (at[k] <= total_time)
&& (bt[k]<min_burst)){
            min_burst= bt[k];
            index = k;
        }

    }

    total_time =total_time + min_burst;
    switch_time[j++] = total_time ;
    process_status[index] = false ;
    process_order[i] =index ;
    // System.out.print(processID[i]+\t" + index +"\n") ;

}
System.out.println() ;

gantChart(processID ,switch_time , process_order) ;
CalcTAT_and_WT(switch_time ,at , bt , process_order) ;
// sortArrayByProcessID(processID, at, bt, tat, wt);
printOutputTable(processID, at, bt ,tat , wt , process_order)
;
}

```

```

public static void main(String args[]){
    Scanner sc = new Scanner(System.in) ;

    System.out.println("Enter the number of processes : ") ;
    num =sc.nextInt() ;

    System.out.println("Enter the Arrival Time & Burst Time of
the Processes :) " ) ;

    for(int i=0 ; i<num ; i++){
        // System.out.println("Arrival Time & Burst Time of "+
(i+1)+" : " ) ;
        processID[i] =i+1 ;
        at[i] =sc.nextInt() ;
        bt[i] =sc.nextInt() ;
        process_status[i]= true;
    }
}

```

```

    }

    //Sorting of the Processes wrt AT
    sortArrayByAT(processID,at,bt) ;

    //Print Array
    printInputTable(processID,at,bt) ;

    //SJF NP
    SJFNP(processID,at,bt) ;

}
}

```

OUTPUT :

```

Enter the number of processes :
5
Enter the Arrival Time & Burst Time of the Processes :
1 7
2 5
3 1
4 2
5 8

*** The Input Table ***

PID    AT    BT
1       1    7
2       2    5
3       3    1
4       4    2
5       5    8

** GANTT CHART **

0       [NA]  1       [1]    8       [3]    9       [4]    11      [2]    16      [5]    24

*** The Final Output Table ***

PID    AT    BT    TAT    WT
1       1    7     7     0
3       3    1     6     5
4       4    2     7     5
2       2    5    14     9
5       5    8    19    11

Average TurnAround Time : 10.6
Average Waiting Time : 6.0

```

## **CONCLUSION :**

Thus, we were able to successfully use the SJF Non-Preemptive Scheduling Algorithm and understand its advantages and disadvantages.

## Shortest Job First Preemptive Scheduling

**AIM:** Write a program to demonstrate the Shortest Job First Preemptive Scheduling program with Gantt diagram.

### THEORY:

In the Shortest Remaining Time First (SRTF) scheduling algorithm, the process with the smallest amount of time remaining until completion is selected to execute. Since the currently executing process is the one with the shortest amount of time remaining by definition, and since that time should only reduce as execution progresses, processes will always run until they complete or a new process is added that requires a smaller amount of time.

### Advantages:

Here, are pros/benefits of Preemptive Scheduling method:

- Preemptive scheduling method is more robust, approach so one process cannot monopolize the CPU
- Choice of running task reconsidered after each interruption.
- Each event cause interruption of running tasks
- The OS makes sure that CPU usage is the same by all running process.
- In this, the usage of CPU is the same, i.e., all the running processes will make use of CPU equally.
- This scheduling method also improvise the average response time.

- Preemptive Scheduling is beneficial when we use it for the multiprogramming environment.

### **Disadvantages :**

Here, are cons/drawback of Preemptive Scheduling method:

- Need limited computational resources for Scheduling
- Takes a higher time by the scheduler to suspend the running task, switch the context, and dispatch the new incoming task.
- The process which has low priority needs to wait for a longer time if some high priority processes arrive continuously.

CODE:

```
import java.util.*;

class P {
    int id;
    int burstTime;
    int arrivalTime;

    public P(int id, int arrivalTime, int burstTime) {
        this.id = id;
        this.burstTime = burstTime;
        this.arrivalTime = arrivalTime;
    }
}

class P_SRT {

    static void findWaitingTime(P process_arr[], int waitingTime[],
int finish_time[], ArrayList<Integer> Grant) {

        int length = process_arr.length;
        int remainingTime[] = new int[length];
```

```

for (int i = 0; i < length; i++)
    remainingTime[i] = process_arr[i].burstTime;

int complete = 0;
int time = 0;
int minimum = Integer.MAX_VALUE;
int shortest = 0;
boolean check = false;

while (complete != length) {

    for (int j = 0; j < length; j++) {
        if ((process_arr[j].arrivalTime <= time) &&
(remainingTime[j] < minimum) && remainingTime[j] > 0) {
            minimum = remainingTime[j];
            shortest = j;
            check = true;
        }
    }

    if (check == false) {
        Grant.add(0);
        time++;
        continue;
    }

    Grant.add(process_arr[shortest].id);
    remainingTime[shortest]--;

    minimum = remainingTime[shortest];
    if (minimum == 0)
        minimum = Integer.MAX_VALUE;

    if (remainingTime[shortest] == 0) {

        complete++;
        check = false;
        finish_time[shortest] = time + 1;

        // waiting time
        waitingTime[shortest] = finish_time[shortest] -
process_arr[shortest].burstTime

```



```

        - process_arr[shortest].arrivalTime;
    }
    time++;
}

static void findTurnAroundTime(P process_arr[], int
waitingTime[], int turnAroundTime[]) {
    int length = process_arr.length;
    for (int i = 0; i < length; i++)
        turnAroundTime[i] = process_arr[i].burstTime +
waitingTime[i];
}

static void findavgTime(P process_arr[], int waitingTime[], int
turnAroundTime[], int finishTime[],
    ArrayList<Integer> Grant) {

    int length = process_arr.length;
    int total_waitTime = 0;
    int total_tATime = 0;
    System.out.println("Processes " + " Arrival Time " + " Burst
time " + "Finish Time" + " Waiting time "
        + " Turn around time");

    // Calculate total waiting time and
    // total turnaround time
    for (int i = 0; i < length; i++) {
        total_waitTime = total_waitTime + waitingTime[i];
        total_tATime = total_tATime + turnAroundTime[i];
        System.out.println(" " + process_arr[i].id + "\t\t" + " "
+ process_arr[i].arrivalTime + "\t\t"
            + +process_arr[i].burstTime + "\t" +
finishTime[i] + "\t\t" + waitingTime[i] + "\t\t"
            + turnAroundTime[i]);
    }

    System.out.println("Average waiting time = " + (double)
total_waitTime / (double) length);
    System.out.println("Average turn around time = " + (double)
total_tATime / (double) length);
    System.out.println("Grant Chart: " + Grant);
}

```

```

    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter number of processes");
        int n = sc.nextInt();
        sc.nextLine();
        ArrayList<Integer> Grant = new ArrayList<Integer>();
        int arrivalTime = 0, bursttime = 0;
        int waitingTime[] = new int[n];
        int turnAroundTime[] = new int[n];
        int finishTime[] = new int[n];
        P process_arr[] = new P[n];
        for (int i = 0; i < n; i++) {
            System.out.println("Enter arrival time and burst time ");
            arrivalTime = sc.nextInt();
            bursttime = sc.nextInt();
            process_arr[i] = new P(i + 1, arrivalTime, bursttime);
        }
        sc.close();

        findWaitingTime(process_arr, waitingTime, finishTime, Grant);
        findTurnAroundTime(process_arr, waitingTime, turnAroundTime);
        findavgTime(process_arr, waitingTime, turnAroundTime,
        finishTime, Grant);
    }
}

```

OUTPUT :

```

Enter number of processes
5
Enter arrival time and burst time
1 7
Enter arrival time and burst time
2 5
Enter arrival time and burst time
3 1
Enter arrival time and burst time
4 2
Enter arrival time and burst time
5 8
Processes  Arrival Time  Burst time  Finish Time  Waiting time  Turn around time
1           1             7           16           8           15
2           2             5           10           3           8
3           3             1            4           0           1
4           4             2            6           0           2
5           5             8           24          11          19
Average waiting time = 4.4
Average turn around time = 9.0
Grant Chart: [0, 1, 2, 3, 4, 4, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 5, 5, 5, 5, 5, 5, 5, 5]
PS C:\Users\junai\OneDrive - Shri Vile Parle Kelavani Mandal\Desktop\DJSCF\SEM 4\PRACTICE

```

## CONCLUSION :

Thus, we were able to successfully use the SJF Preemptive Scheduling Algorithm and understand its advantages and disadvantages.

## Preemptive Priority Scheduling

**AIM:** Write a program to demonstrate Preemptive Priority Scheduling with gantt diagram

### THEORY:

**In this problem smaller numbers denote higher priority.**

The following functions are used in the given code below:

- **void quicksort(process array[], low, high)**– This function is used to arrange the processes in ascending order according to their arrival time.
- **int partition(process array[], int low, int high)**– This function is used to partition the array for sorting.
- **void insert(process Heap[], process value, int \*heapsize, int \*currentTime)**– It is used to include all the valid and eligible processes in the heap for execution. heapsize defines the number of processes in execution depending on the current time currentTime keeps record of the current CPU time.
- **void order(process Heap[], int \*heapsize, int start)**– It is used to reorder the heap according to priority if the processes after insertion of new process.
- **void extractminimum(process Heap[], int \*heapsize, int \*currentTime)**– This function is used to find the process with highest priority from the heap. It also reorders the heap after extracting the highest priority process.
- **void scheduling(process Heap[], process array[], int n, int \*heapsize, int \*currentTime)**– This function is responsible for executing the highest priority extracted from Heap[].

- **void process(process array[], int n)**– This function is responsible for managing the entire execution of the processes as they arrive in the CPU according to their arrival time.

### **Advantages:**

Here, are benefits/pros of using priority scheduling method:

- Easy to use scheduling method
- Processes are executed on the basis of priority so high priority does not need to wait for long which saves time
- This method provides a good mechanism where the relative importance of each process may be precisely defined.
- Suitable for applications with fluctuating time and resource requirements.

### **Disadvantages:**

Here, are cons/drawbacks of priority scheduling

- If the system eventually crashes, all low priority processes get lost.
- If high priority processes take lots of CPU time, then the lower priority processes may starve and will be postponed for an indefinite time.
- This scheduling algorithm may leave some low priority processes waiting indefinitely.
- A process will be blocked when it is ready to run but has to wait for the CPU because some other process is running currently.

- If a new higher priority process keeps on coming in the ready queue, then the process which is in the waiting state may need to wait for a long duration of time.

CODE:

```
import java.util.*;

class Prow {

    int id;
    int burstTime;
    int arrivalTime;
    int priority;

    public Prow(int id, int arrivalTime, int burstTime, int priority)
    {
        this.id = id;
        this.burstTime = burstTime;
        this.arrivalTime = arrivalTime;
        this.priority = priority;
    }
}

public class Preemptive_Priority {
    public static void main(String[] args) {
        int n;
        System.out.println("Enter Number of Processes:");
        Scanner ob = new Scanner(System.in);
        n = ob.nextInt();
        ArrayList<Integer> Grant = new ArrayList<Integer>();
        int arrivalTime = 0, bursttime = 0, priority;
        int waitingTime[] = new int[n];
        int turnAroundTime[] = new int[n];
        int finishTime[] = new int[n];
        Prow process_arr[] = new Prow[n];
        for (int i = 0; i < n; i++) {
            System.out.println("Enter Arrival Time, Burst Time and
Priority for the Process:");
            arrivalTime = ob.nextInt();
            bursttime = ob.nextInt();
        }
    }
}
```

```

        priority = ob.nextInt();
        process_arr[i] = new Prow(i + 1, arrivalTime, bursttime,
priority);
    }
    findWaitingTime(process_arr, waitingTime, finishTime, Grant);
    findTurnAroundTime(process_arr, waitingTime, turnAroundTime);
    findavgTime(process_arr, waitingTime, turnAroundTime,
finishTime, Grant);

    ob.close();

}

```

```

static void findWaitingTime(Prow process_arr[], int
waitingTime[], int finish_time[], ArrayList<Integer> Grant) {

```

```

    int length = process_arr.length;
    int min = 0;
    int priorityList[] = new int[length];
    int remainingTime[] = new int[length];

    for (int i = 0; i < length; i++)
        priorityList[i] = process_arr[i].priority;

    for (int i = 0; i < length; i++)
        remainingTime[i] = process_arr[i].burstTime;

    int complete = 0;
    int time = 0;
    int minimum = Integer.MAX_VALUE;
    int shortest = 0;
    boolean check = false;

    while (complete != length) {

        for (int j = 0; j < length; j++) {

            if ((process_arr[j].arrivalTime <= time) &&
(priorityList[j] < minimum && remainingTime[j] != 0) &&
(priorityList[j] > 0)) {
                minimum = priorityList[j];
                shortest = j;

```

```

        check = true;
    }
}

if (check == false) {
    Grant.add(0);
    time++;
    continue;
}

Grant.add(process_arr[shortest].id);
remainingTime[shortest]--;

//System.out.println("Remaining Time : " +
remainingTime[shortest]);

minimum = priorityList[shortest];

min = remainingTime[shortest];
if (min <= 0)
    minimum = Integer.MAX_VALUE;

if (remainingTime[shortest] <= 0) {

    complete++;
    check = false;
    finish_time[shortest] = time + 1;

    // waiting time
    waitingTime[shortest] = finish_time[shortest] -
process_arr[shortest].burstTime
        - process_arr[shortest].arrivalTime;
    }
    time++;
}

}

static void findTurnAroundTime(Prow process_arr[], int
waitingTime[], int turnAroundTime[]) {
    int length = process_arr.length;
    for (int i = 0; i < length; i++)
        turnAroundTime[i] = process_arr[i].burstTime +
waitingTime[i];
}

```



```

    }

    static void findavgTime(Prow process_arr[], int waitingTime[],
int turnAroundTime[], int finishTime[],
        ArrayList<Integer> Grant) {

        int length = process_arr.length;
        int total_waitTime = 0;
        int total_tATime = 0;
        System.out.println("Processes " + " Arrival Time " + " Burst
time " + "Priority" + " Finish Time" + " Waiting time "
            + " Turn around time");

        // Calculate total waiting time and
        // total turnaround time
        for (int i = 0; i < length; i++) {
            total_waitTime = total_waitTime + waitingTime[i];
            total_tATime = total_tATime + turnAroundTime[i];
            System.out.println(" " + process_arr[i].id + "\t\t" + " "
+ process_arr[i].arrivalTime + "\t\t"
                + +process_arr[i].burstTime + "\t" +
process_arr[i].priority + "\t" + finishTime[i] + "\t\t" +
waitingTime[i] + "\t\t"
                + turnAroundTime[i]);
        }

        System.out.println("Average waiting time = " + (double)
total_waitTime / (double) length);
        System.out.println("Average turn around time = " + (double)
total_tATime / (double) length);
        System.out.println("Grant Chart: " + Grant);

    }
}

```

OUTPUT :

```

Enter Number of Processes:
7
Enter Arrival Time, Burst Time and Priority for the Process:
0 4 2
Enter Arrival Time, Burst Time and Priority for the Process:
1 2 4
Enter Arrival Time, Burst Time and Priority for the Process:
2 3 6
Enter Arrival Time, Burst Time and Priority for the Process:
3 5 10
Enter Arrival Time, Burst Time and Priority for the Process:
4 1 8
Enter Arrival Time, Burst Time and Priority for the Process:
5 4 12
Enter Arrival Time, Burst Time and Priority for the Process:
6 6 9
Processes  Arrival Time  Burst time  Priority  Finish Time  Waiting time  Turn around time
1           0             4           2         4             0             4
2           1             2           4         6             3             5
3           2             3           6         9             4             7
4           3             5          10        21            13            18
5           4             1           8        10             5             6
6           5             4          12        25            16            20
7           6             6           9        16             4            10
Average waiting time = 6.428571428571429
Average turn around time = 10.0
Grant Chart: [1, 1, 1, 1, 2, 2, 3, 3, 3, 5, 7, 7, 7, 7, 7, 7, 4, 4, 4, 4, 4, 4, 6, 6, 6, 6]

```

## CONCLUSION :

Thus, we were able to successfully use the Priority Preemptive Scheduling Algorithm and understand its advantages and disadvantages.

## Round Robin Scheduling

**AIM:** Write a program to demonstrate Round Robin Scheduling with Gantt diagram.

### THEORY:

Round Robin is a CPU scheduling algorithm where each process is assigned a fixed time slot in a cyclic way.

- It is simple, easy to implement, and starvation-free as all processes get fair share of CPU.
- One of the most commonly used technique in CPU scheduling as a core.
- It is preemptive as processes are assigned CPU only for a fixed slice of time at most.
- The disadvantage of it is more overhead of context switching.

### ADVANTAGES:

- There is fairness since every process gets equal share of CPU.
- The newly created process is added to end of ready queue.
- A round-robin scheduler generally employs time-sharing, giving each job a time slot or quantum.
- While performing a round-robin scheduling, a particular time quantum is allocated to different jobs.

- Each process gets a chance to reschedule after a particular quantum time in this scheduling.

#### DISADVANTAGES:

- There is Larger waiting time and Response time.
- There is Low throughput.
- There are Context Switches.
- Gantt chart seems to come too big (if quantum time is less for scheduling. For Example: 1 ms for big scheduling.)
- Time consuming scheduling for small quanta .

#### CODE:

```
import java.util.*;
import java.util.Scanner;

public class RoundRobin {
    static ArrayList<chartItem> ganttChart = new
    ArrayList<>();

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter number of
Process: ");
        int n =sc.nextInt();
        Process3[] process = new Process3[n];
        System.out.println("ID AT BT");
        for(int i=0;i<n;i++)
        {
            int id = sc.nextInt();
            int AT = sc.nextInt();
            int BT = sc.nextInt();
            process[i] = new Process3(id,AT,BT);
        }
    }
}
```

```

        System.out.println("Enter Quantum Time :
");
        int QT =sc.nextInt();
        sortProcess(process);
        completeProcess(process,QT);
        float wait =0.0f;
        float Ttime =0.0f;
        System.out.println("Id\tAT\tBT\tCT\tWT\tTT
");
        for(int i=0;i<process.length;i++)
        {
            System.out.println(process[i].id+"\t"+process[i].AT
+" \t"+process[i].BT+"\t"+process[i].CT+"\t"+process
[i].WT+"\t"+process[i].TT);
            wait+=process[i].WT;
            Ttime+=process[i].TT;
        }
        System.out.print("\nProcess: ");
        for(int i=0;i<ganttChart.size();i++)
        {
            System.out.print("|
"+ganttChart.get(i).id+"\t");

        }
        System.out.println("| \n");
        System.out.print("Time:-\t");
        int time=0;
        System.out.print(" "+time+"\t");
        for(int i=0;i<ganttChart.size();i++)
        {
            time+=ganttChart.get(i).time;
            System.out.print(" "+time+"\t");

        }
        System.out.println("\n");
        System.out.println("AVG Waiting time :
"+(wait/process.length)+"\nAVG Turn around time :
"+(Ttime/process.length));

    }

```

```

        static void completeProcess(Process3[]
process,int QT) {
            ArrayList<Process3> readyQueue = new
ArrayList<>();
            int time = 0 ;
            int pt =0;
            while(pt<process.length)
            {
                if(readyQueue.isEmpty())
                {
                    readyQueue.add(new
Process3(process[pt]));
                    time = process[pt].AT>=time
?process[pt].AT:time;
                }
                else {
                    if(process[pt].AT>time)
                    {
                        break;
                    }
                    else
                    {
                        readyQueue.add(new
Process3(process[pt]));
                    }
                }
                pt++;
            }

            while(!readyQueue.isEmpty()) {
                boolean completed = false;
                Process3 p = readyQueue.remove(0);

                if(p.BT>QT) {
                    p.BT-=QT;
                    time+=QT;
                    chartItem c=new chartItem(p.id,QT);
                    ganttChart.add(c);
                    completed=false;
                }
            }

```

```

        else {
            time += p.BT;
            int id = getId(process,p);
            process[id].CT=time;
            process[id].TT=time-process[id].AT;

process[id].WT=process[id].TT-process[id].BT;
            chartItem c=new
chartItem(p.id,p.BT);
            ganttChart.add(c);
            completed=false;
            completed=true;

        }

        while(pt<process.length)
        {
            if(readyQueue.isEmpty())
            {
                readyQueue.add(new
Process3(process[pt]));
                time = process[pt].AT>=time
?process[pt].AT:time;
            }
            else {
                if(process[pt].AT>time)
                {
                    break;
                }
                else
                {
                    readyQueue.add(new
Process3(process[pt]));
                }
            }
            pt++;
        }

        if(!completed) {
            readyQueue.add(p);
        }

```

```

    }
}

static void sortProcess(Process3[] process)
{
    for (int i = 0; i < process.length; i++) {
        for (int j = 0; j < process.length - i
- 1; j++) {
            if (process[j].AT > process[j+1].AT) {
                Process3 Temp = process[j];
                process[j]=process[j+1];
                process[j+1]=Temp;
            }
        }
    }
}

static int getId(Process3[] ps,Process3 p)
{
    int id =0;
    for(id=0;id<ps.length;id++)
    {
        if(ps[id].id==p.id)
            break;
    }
    return id;
}

}

class Process3{
    int id,AT,BT,WT=0,TT=0,CT=0;
    Process3(int id,int AT,int BT)
    {
        this.id=id;
        this.AT=AT;
        this.BT=BT;
    }
    Process3(Process3 p){
        this.id=p.id;

```



```

        this.AT=p.AT;
        this.BT=p.BT;
    }
}

class chartItem{
    int id,time;
    chartItem(int id,int time){
        this.id=id;
        this.time = time;
    }
}

```

OUTPUT:

```

Enter number of Process:
6
ID AT BT
1 0 4
2 1 5
3 2 2
4 3 1
5 4 6
6 6 3
Enter Quantum Time :
2
Id    AT    BT    CT    WT    TT
1      0     4     8     4     8
2      1     5    18    12    17
3      2     2     6     2     4
4      3     1     9     5     6
5      4     6    21    11    17
6      6     3    19    10    13

Process: | 1 | 2 | 3 | 1 | 4 | 5 | 2 | 6 | 5 | 2
         | 5 |   |   |   |   |   |   |   |   |
Time:-   0     2     4     6     8     9    11    13    15    17
18      19    21

AVG Waiting time : 7.333333
AVG Turn around time : 10.833333
PS C:\Users\jijunai\OneDrive - Shri Vile Parle Kelavani Mandal\Desktop\DS&CF\SEM 4\PPA\CTCA

```

**CONCLUSION:**

Thus, we were able to successfully use the Round Robin Scheduling Algorithm and understand its advantages and disadvantages.

# EXPERIMENT – 5

**AIM:** Implementation of process and thread synchronization using Client Server architecture

## **THEORY:**

### **Synchronized Keyword**

Java provides a keyword “Synchronized” that can be used in a program to mark a Critical section. The critical section can be a block of code or a complete method. Thus, only one thread can access the critical section marked by the Synchronized keyword.

We can write the concurrent parts (parts that execute concurrently) for an application using the Synchronized keyword. We also get rid of the race conditions by making a block of code or a method Synchronized.

When we mark a block or method synchronized, we protect the shared resources inside these entities from simultaneous access and thereby corruption.

### **Types of Synchronization**

**There are 2 types of synchronization as explained below:**

## 1) Process Synchronization

Process Synchronization involves multiple processes or threads executing simultaneously. They ultimately reach a state where these processes or threads commit to a specific sequence of actions.

## 2) Thread Synchronization

In Thread Synchronization, more than one thread is trying to access a shared space. The threads are synchronized in such a manner that the shared space is accessed only by one thread at a time.

**In Java, we can use the synchronized keyword with:**

- A block of code
- A method

The above types are the mutually exclusive types of thread synchronization. Mutual exclusion keeps the threads accessing shared data from interfering with each other.

The other type of thread synchronization is “InterThread communication” that is based on cooperation between threads.

## Client-Server Model:

The Client-server model is a distributed application structure that partitions task or workload between the providers of a resource or service, called servers, and service requesters called clients. In the client-server architecture, when the client computer sends a request for data to the server through the internet, the server accepts the requested process and deliver the data packets requested back to the client. Clients do not share any of their resources. Examples of Client-Server Model are Email, World Wide Web, etc.

## Working of Client-Server Model:

In this article we are going to take a dive into the **Client-Server** model and have a look at how the **Internet** works via, web browsers. This article will help us in having a solid foundation of the WEB and help in working with WEB technologies with ease.

- **Client:** When we talk the word **Client**, it mean to talk of a person or an organization using a particular service. Similarly in the digital world a **Client** is a computer (**Host**) i.e. capable of receiving information or using a particular service from the service providers (**Servers**).
- **Servers:** Similarly, when we talk the word **Servers**, It mean a person or medium that serves something. Similarly in this digital world a **Server** is a remote computer which provides information (data) or access to particular services.

So, its basically the **Client** requesting something and the **Server** serving it as long as its present in the database.

### **Advantages of Client-Server model:**

- Centralized system with all data in a single place.
- Cost efficient requires less maintenance cost and Data recovery is possible.
- The capacity of the Client and Servers can be changed separately.

### **Disadvantages of Client-Server model:**

- Clients are prone to viruses, Trojans and worms if present in the Server or uploaded into the Server.
- Server are prone to Denial of Service (DOS) attacks.
- Data packets may be spoofed or modified during transmission.
- Phishing or capturing login credentials or other useful information of the user are common and MITM(Man in the Middle) attacks are common.

## **Server-side Programming :**

It is the program that runs on server dealing with the generation of content of web page.

- 1) Querying the database
- 2) Operations over databases
- 3) Access/Write a file on server.
- 4) Interact with other servers.
- 5) Structure web applications.
- 6) Process user input. For example if user input is a text in search box, run a search algorithm on data stored on server and send the results.

## **Client-side Programming :**

It is the program that runs on the client machine (browser) and deals with the user interface/display and any other processing that can happen on client machine like reading/writing cookies.

- 1) Interact with temporary storage
- 2) Make interactive web pages
- 3) Interact with local storage
- 4) Sending request for data to server
- 5) Send request to server
- 6) work as an interface between server and user

CODE:

## Client Side Code -

```
import java.io.*;
import java.net.*;

class Client {
    public static void main(String[] args) throws Exception {
        String msg;
        Socket s2 = new Socket("localhost", 80);

        String names[] = { "JUNAID", "JUNAID_GIRKAR" };
        int i = (int) (Math.random() * (names.length));
        msg = names[i];

        PrintStream d1 = new PrintStream(s2.getOutputStream());
        d1.println(msg);

        BufferedReader d2 = new BufferedReader(new
        InputStreamReader(s2.getInputStream()));
        String resp = d2.readLine();

        System.out.println("\nServer response: " + resp);
    }
}
```

## Server Side Code -

```
import java.io.*;
import java.net.*;

class manageClient extends Thread {
    private Socket s;
    private String name;

    manageClient(Socket s) {
        this.s = s;
    }
}
```

```

        public void run() {
            try {
                BufferedReader d2 = new BufferedReader(new
InputStreamReader(s.getInputStream()));
                this.name = d2.readLine();

                System.out.println("\nClient Name: " + this.name);

                PrintStream d1 = new PrintStream(s.getOutputStream());
                d1.println("You are connected " + this.name);
            } catch (Exception e) {
                System.out.println("\n [!] Thread of client: " +
this.name + " was interrupted.");
            }
        }
    }

class Server {
    public static void main(String[] args) throws Exception {
        System.out.println("\nStarting Server...\n");
        ServerSocket socket = new ServerSocket(80);
        try {
            while (true) {
                Socket s1 = socket.accept();

                System.out.println("\nClient connected");

                Thread T1 = new manageClient(s1);
                T1.start();
            }
        } catch (Exception e) {
            System.out.println(e);
            socket.close();
        }
    }
}

```

OUTPUT:



## Server Output

```
PS C:\Users\junai\OneDrive - Shri Vile Parle K...  
TICALS\OS\Practical-3 10-03-2021> java Server  
  
Starting Server...  
  
Client connected  
Client Name: JUNAID
```

## Client Output

```
PS C:\Users\junai\OneDrive - Shri Vile Parle K...  
TICALS\OS\Practical-3 10-03-2021> java Client  
  
Server response: You are connected JUNAID
```

## CONCLUSION:

We learnt the use of the synchronization keyword in java. The different types of synchronization. We also learnt about the Client-Server model in Java and its implementation with the use of Sockets from java.net package.

# EXPERIMENT - 6

---

**AIM:** Case Study based on Producer Consumer problem using Semaphores

## **THEORY:**

About Producer-Consumer problem:

The Producer-Consumer problem is a classic problem this is used for multi-process synchronization i.e. synchronization between more than one processes.

In the producer-consumer problem, there is one Producer that is producing something and there is one Consumer that is consuming the products produced by the Producer. The producers and consumers share the same memory buffer that is of fixed-size.

The job of the Producer is to generate the data, put it into the buffer, and again start generating data. While the job of the Consumer is to consume the data from the buffer.

## About the problem:

The following are the problems that might occur in the Producer-Consumer:

- The producer should produce data only when the buffer is not full. If the buffer is full, then the producer shouldn't be allowed to put any data into the buffer.
- The consumer should consume data only when the buffer is not empty. If the buffer is empty, then the consumer shouldn't be allowed to take any data from the buffer.
- The producer and consumer should not access the buffer at the same time.

## Solution:

The above three problems can be solved with the help of semaphores

A semaphore S is an integer variable that can be accessed only through two standard operations : wait() and signal().

The wait() operation reduces the value of semaphore by 1 and the signal() operation increases its value by 1.

```
wait(S){  
while(S<=0);    // busy waiting  
S--;  
}
```

```
signal(S){  
S++;  
}
```

Semaphores are of two types:

1. **Binary Semaphore** – This is similar to mutex lock but not the same thing. It can have only two values – 0 and 1. Its value is initialized to 1. It is used to implement the solution of critical section problem with multiple processes.
2. **Counting Semaphore** – Its value can range over an unrestricted domain. It is used to control access to a resource that has multiple instances.

In the producer-consumer problem, we use three semaphore variables:

1. **Semaphore S:** This semaphore variable is used to achieve mutual exclusion between processes. By using this variable, either Producer or Consumer will be allowed to use or access the shared buffer at a particular time. This variable is set to 1 initially.
2. **Semaphore E:** This semaphore variable is used to define the empty space in the buffer. Initially, it is set to the whole space of the buffer i.e. "n" because the buffer is initially empty.

3. **Semaphore F:** This semaphore variable is used to define the space that is filled by the producer. Initially, it is set to "0" because there is no space filled by the producer initially.

By using the above three semaphore variables and by using the *wait()* and *signal()* function, we can solve our problem(the *wait()* function decreases the semaphore variable by 1 and the *signal()* function increases the semaphore variable by 1). So. let's see how.

**The following is the pseudo-code for the producer:**

```
void producer() {  
    while(T) {  
        produce()  
        wait(E)  
        wait(S)  
        append()  
        signal(S)  
        signal(F)  
    }  
}
```

The above code can be summarized as:

- **while()** is used to produce data, again and again, if it wishes to produce, again and again.
- **produce()** function is called to produce data by the producer.
- **wait(E)** will reduce the value of the semaphore variable "E" by one i.e. when the producer produces something then there is a decrease in the value of the empty space in the buffer. If the buffer is full i.e. the value of the semaphore variable "E" is "0", then the program will stop its execution and no production will be done.
- **wait(S)** is used to set the semaphore variable "S" to "0" so that no other process can enter into the critical section.
- **append()** function is used to append the newly produced data in the buffer.
- **signal(s)** is used to set the semaphore variable "S" to "1" so that other processes can come into the critical section now because the production is done and the append operation is also done.
- **signal(F)** is used to increase the semaphore variable "F" by one because after adding the data into the buffer, one space is filled in the buffer and the variable "F" must be updated.

This is how we solve the produce part of the producer-consumer problem. Now, let's see the consumer solution. The following is the code for the consumer:

```
void consumer() {  
    while(T) {  
        wait(F)  
        wait(S)  
        take()  
        signal(S)  
        signal(E)  
        use()  
    }  
}
```

The above code can be summarized as:

- **while()** is used to consume data, again and again, if it wishes to consume, again and again.
- **wait(F)** is used to decrease the semaphore variable "F" by one because if some data is consumed by the consumer then the variable "F" must be decreased by one.
- **wait(S)** is used to set the semaphore variable "S" to "0" so that no other process can enter into the critical section.
- **take()** function is used to take the data from the buffer by the consumer.
- **signal(S)** is used to set the semaphore variable "S" to "1" so that other processes can come into the critical section now because the consumption is done and the take operation is also done.
- **signal(E)** is used to increase the semaphore variable "E" by one because after taking the data from the buffer, one space is freed from the buffer and the variable "E" must be increased.

- **use()** is a function that is used to use the data taken from the buffer by the process to do some operation.

## CODE :

```
import java.util.concurrent.Semaphore;

class Q {

    int item;

    // semCon initialized with 0 permits
    // to ensure put() executes first
    static Semaphore semCon = new Semaphore(0);

    static Semaphore semProd = new Semaphore(1);

    // to get an item from buffer
    void get()
    {
        try {
            // Before consumer can consume an item,
            // it must acquire a permit from semCon
            semCon.acquire();
        }
        catch (InterruptedException e) {
            System.out.println("InterruptedException caught");
        }

        // consumer consuming an item
        System.out.println("CONSUMER consumed item : " + item);

        // After consumer consumes the item,
        // it releases semProd to notify producer
        semProd.release();
    }

    // to put an item in buffer
    void put(int item)
```



```

{
    try {
        // Before producer can produce an item,
        // it must acquire a permit from semProd
        semProd.acquire();
    }
    catch (InterruptedException e) {
        System.out.println("InterruptedException caught");
    }

    // producer producing an item
    this.item = item;

    System.out.println("PRODUCER produced item : " + item);

    // After producer produces the item,
    // it releases semCon to notify consumer
    semCon.release();
}
}

// Producer class
class Producer implements Runnable {
    Q q;
    Producer(Q q)
    {
        this.q = q;
        new Thread(this, "PRODUCER").start();
    }

    public void run()
    {
        for (int i = 0; i < 5; i++)
            // producer put items
            q.put(i);
    }
}

// Consumer class
class Consumer implements Runnable {
    Q q;
    Consumer(Q q)
    {

```

```

        this.q = q;
        new Thread(this, "CONSUMER").start();
    }

    public void run()
    {
        for (int i = 0; i < 5; i++)
            // consumer get items
            q.get();
    }
}

// Driver class
class Producer_Consumer {
    public static void main(String args[])
    {
        // creating buffer queue
        Q q = new Q();

        // starting consumer thread
        new Consumer(q);

        // starting producer thread
        new Producer(q);
    }
}

```

## OUTPUT:

```

C:\Users\junit\Documents> java Producer_Consumer
PRODUCER produced item : 0
CONSUMER consumed item : 0
PRODUCER produced item : 1
CONSUMER consumed item : 1
PRODUCER produced item : 2
CONSUMER consumed item : 2
PRODUCER produced item : 3
CONSUMER consumed item : 3
PRODUCER produced item : 4
CONSUMER consumed item : 4

```

## **CONCLUSION:**

We learnt about Semaphores and how they can be implemented to solve the Producer-Consumer problem which is used for multi-process synchronization.

# EXPERIMENT – 7

---

**AIM:** Working and Implementation Bankers Algorithm

## **THEORY:**

The banker's algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an "s-state" check to test for possible activities, before deciding whether allocation should be allowed to continue.

Banker's algorithm is named so because it is used in the banking system to check whether a loan can be sanctioned to a person or not. Suppose there are n number of account holders in a bank and the total sum of their money is S. If a person applies for a loan then the bank first subtracts the loan amount from the total money that bank has and if the remaining amount is greater than S then only the loan is sanctioned. It is done because if all the account holders come to withdraw their money then the bank can easily do it.

In other words, the bank would never allocate its money in such a way that it can no longer satisfy the needs of all its customers. The bank would try to be in safe state always.

When working with a banker's algorithm, it requests to know about three things:

1. How much each process can request for each resource in the system. It is denoted by the [MAX] request.
2. How much each process is currently holding each resource in a system. It is denoted by the [ALLOCATED] resource.

3. It represents the number of each resource currently available in the system. It is denoted by the **[AVAILABLE]** resource.

Following are the important data structures terms applied in the banker's algorithm as follows:

Suppose  $n$  is the number of processes, and  $m$  is the number of each type of resource used in a computer system.

1. **Available:** It is an array of length ' $m$ ' that defines each type of resource available in the system. When  $\text{Available}[j] = K$ , means that ' $K$ ' instances of Resources type  $R[j]$  are available in the system.
2. **Max:** It is a  $[n \times m]$  matrix that indicates each process  $P[i]$  can store the maximum number of resources  $R[j]$  (each type) in a system.
3. **Allocation:** It is a matrix of  $m \times n$  orders that indicates the type of resources currently allocated to each process in the system. When  $\text{Allocation}[i, j] = K$ , it means that process  $P[i]$  is currently allocated  $K$  instances of Resources type  $R[j]$  in the system.
4. **Need:** It is an  $M \times N$  matrix sequence representing the number of remaining resources for each process. When the  $\text{Need}[i][j] = k$ , then process  $P[i]$  may require  $K$  more instances of resources type  $R_j$  to complete the assigned work.  
$$\text{Need}[i][j] = \text{Max}[i][j] - \text{Allocation}[i][j].$$

5. **Finish**: It is the vector of the order **m**. It includes a Boolean value (true/false) indicating whether the process has been allocated to the requested resources, and all resources have been released after finishing its task.

The Banker's Algorithm is the combination of the safety algorithm and the resource request algorithm to control the processes and avoid deadlock in a system:

## Safety Algorithm

It is a safety algorithm used to check whether or not a system is in a safe state or follows the safe sequence in a banker's algorithm:

1. There are two vectors **Work** and **Finish** of length **m** and **n** in a safety algorithm.

```
Initialize: Work = Available  
Finish[i] = false; for I = 0, 1, 2, 3, 4... n - 1.
```

2. Check the availability status for each type of resources [i], such as:

```
Need[i] <= Work  
Finish[i] == false  
If the i does not exist, go to step 4.
```

- 3.

```
Work = Work + Allocation(i) // to get new resource allocation  
Finish[i] = true
```

Go to step 2 to check the status of resource availability for the next process.

4. If `Finish[i] == true`; it means that the system is safe for all processes.

# Resource Request Algorithm

A resource request algorithm checks how a system will behave when a process makes each type of resource request in a system as a request matrix.

Let's create a resource request array  $R[i]$  for each process  $P[i]$ . If the Resource Request  $[j]$  equals 'K', which means the process  $P[i]$  requires 'k' instances of Resources type  $R[j]$  in the system.

1. When the number of **requested resources** of each type is less than the **Need** resources, go to step 2 and if the condition fails, which means that the process  $P[i]$  exceeds its maximum claim for the resource. As the expression suggests:

```
If Request(i) <= Need
Go to step 2;
```

2. And when the number of requested resources of each type is less than the available resource for each process, go to step (3). As the expression suggests:

```
If Request(i) <= Available
```

Else Process  $P[i]$  must wait for the resource since it is not available for use.

3. When the requested resource is allocated to the process by changing state:

```
Available = Available - Request
```

```
Allocation(i) = Allocation(i) + Request (i)
```

```
Needi = Needi - Requesti
```

When the resource allocation state is safe, its resources are allocated to the process  $P(i)$ . And if the new state is unsafe,

the Process P (i) has to wait for each type of Request R(i) and restore the old resource-allocation state.

## Advantages

Following are the essential characteristics of the Banker's algorithm:

1. It contains various resources that meet the requirements of each process.
2. Each process should provide information to the operating system for upcoming resource requests, the number of resources, and how long the resources will be held.
3. It helps the operating system manage and control process requests for each type of resource in the computer system.
4. The algorithm has a Max resource attribute that indicates each process can hold the maximum number of resources in a system.

## Disadvantages

1. It requires a fixed number of processes, and no additional processes can be started in the system while executing the process.
2. The algorithm no longer allows the processes to exchange its maximum needs while processing its tasks.



3. Each process has to know and state their maximum resource requirement in advance for the system.
4. The number of resource requests can be granted in a finite time, but the time limit for allocating the resources is one year.

## CODE:

```
import java.util.*;
class Bankers
{
    static int P=5;
    static int R=3;
    static void printMatrix(int m[][])
    {
        for(int i=0;i<P;i++)
        {
            for(int j=0;j<R;j++)
            {
                System.out.print(m[i][j] + " ");
            }
            System.out.println();
        }
    }
    static void calculateNeed(int need[][], int max[][],int
alloc[][])
    {
        for (int i = 0 ; i < P ; i++)
        {
            for (int j = 0 ; j < R ; j++)
            {
                need[i][j] = max[i][j] - alloc[i][j];
            }
        }
        System.out.println("\nNeed Matrix :");
        printMatrix(need);
    }
    static void isSafe(int processes[], int avail[], int max[][],int
alloc[][])
}
```

```

{
    int [][]need = new int[P][R];
    calculateNeed(need, max, alloc);
    boolean []finish = new boolean[P];
    int []safeSeq = new int[P];
    int []work = new int[R];
    for (int i = 0; i < R ; i++)
    {
        work[i] = avail[i];
    }
    int count = 0;
    while (count < P)
    {
        boolean found = false;
        for (int p = 0; p < P; p++)
        {
            if (finish[p] == false)
            {
                int j;
                for (j = 0; j < R; j++)
                {
                    if (need[p][j] > work[j])
                        break;
                }
                if (j == R)
                {
                    for (int k = 0 ; k < R ; k++)
                    {
                        work[k] += alloc[p][k];
                    }
                    safeSeq[count++] = processes[p];
                    finish[p] = true;
                    found = true;
                }
            }
        }
        if (found == false)
        {
            System.out.print("\nSystem is not in safe state.
Deadlock can occur.");
            return;
        }
    }
}

```

```

        System.out.print("\nSystem is in safe state.\nSafe sequence
is: ");
        for (int i = 0; i < P-1 ; i++)
        {
            System.out.print("P" + safeSeq[i] + "->");
        }
        System.out.print(safeSeq[P-1]);
    }
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the no of processes : ");
        P = sc.nextInt();
        int processes[] = new int [P];
        for(int i=0;i<P;i++)
        {
            processes[i]=i+1;
        }
        System.out.println("Enter total no of resources : ");
        R = sc.nextInt();
        int totalResources[] = new int[R];
        System.out.println("Enter total no of resources present : ");
        for(int i=0;i<R;i++)
        {
            totalResources[i]=sc.nextInt();
        }
        int avail[] = new int[R];
        int max[][] = new int[P][R];
        int alloc[][] = new int[P][R];
        System.out.println("Enter Maximum need Matrix :");
        for(int i=0;i<P;i++)
        {
            for(int j=0;j<R;j++)
            {
                max[i][j]=sc.nextInt();
            }
        }
        System.out.println("Enter Allocation Matrix :");
        for(int i=0;i<P;i++)
        {
            for(int j=0;j<R;j++)
            {
                alloc[i][j]=sc.nextInt();
            }
        }
    }
}

```

```

    }
}
for(int i=0;i<R;i++)
{
    avail[i]=0;
    for(int j=0;j<P;j++)
    {
        avail[i]=avail[i] + alloc[j][i];
    }
    avail[i]=totalResources[i]-avail[i];
}
System.out.println("Available Matrix :");
for(int i=0;i<R;i++)
{
    System.out.print(avail[i] + " ");
}
System.out.println("\nMaximum Need Matrix :");
printMatrix(max);
System.out.println("\nAllocated Matrix :");
printMatrix(alloc);
isSafe(processes, avail, max, alloc);
sc.close();
}
}

```

**OUTPUT:**

```

Enter the no of processes :
5
Enter total no of resources :
3
Enter total no of resources present :
10
5
7
Enter Maximum need Matrix :
7 5 3
3 2 2
9 0 2
4 2 2
5 3 3
Enter Allocation Matrix :
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Available Matrix :
3 3 2
Maximum Need Matrix :
7 5 3
3 2 2
9 0 2
4 2 2
5 3 3

Allocated Matrix :
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2

Need Matrix :
7 4 3
1 2 2
6 0 0
2 1 1
5 3 1

System is in safe state.
Safe sequence is: P2->P4->P5->P1->3

```

**CONCLUSION:** We learnt implementation of Bankers Algorithm which is used to avoid deadlocks in an operating system by simulating the allocation for predetermined maximum possible amounts of all resources before deciding whether allocation should be allowed to continue.

# EXPERIMENT - 8

---

**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
- 2) 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).

The first process moves from the ready processes section to the running processes. Meanwhile I can see the registers changing rapidly in another window. After about 20 seconds, Process 1 leaves the Running Processes section and Process 2 takes it place. Again 20 seconds then Process 3 and then Process 4. After that since there are no ready processes both Ready Processes and Running Processes sections remain empty

OS Simulator: CPU 0 [YASMIN: CPU-OS Simulator, Version: 7.5.50, Copyright © 2006-2013, Besim Mustafa, Edge Hill University, UK]

**RUNNING PROCESSES**

Pid	Name	State	Memory	Priority	Burst	Swap	PName	CPU	PPid
10	LOOPTEST1		1	3	0	No	P1	0	9

WAIT Period: 0 sec. QUEUE KILL Force kill ☐ Suspend on state change ☐ SHOW MEMORY... SHOW PCB...

**READY PROCESSES (Ready Queue)**

Pid	Name	State	Memory	Priority	Burst	Swap	PName	CPU	PPid
11	LOOPTEST1		1	3	0	No	P2	9	
12	LOOPTEST1		1	3	0	No	P3	9	
13	LOOPTEST1		1	3	0	No	P4	9	

CLEAR REMOVE Suspend on state change ☐ SHOW MEMORY... SHOW PCB...

**WAITING PROCESSES (Waiting Queue)**

Pid	Name	State	Memory	Priority	Burst	Swap	PName	CPU	PPid
4	DEADLOCKP1		1	3	3	No	P1	0	
5	DEADLOCKP2		1	3	3	No	P2	1	
6	DEADLOCKP3		1	3	3	No	P3	2	
7	DEADLOCKP4		1	3	3	No	P4	3	

CLEAR REMOVE RESUME Suspend on state change ☐ SHOW MEMORY... SHOW PCB... SHOW COMPILER... CPU 0... CLOSE

**SCHEDULER**

Policies: ☒ First-Come, First-Served (FCFS) ☐ Lottery Scheduling ☐ Shortest-Job-First (SJF) ☐ Fair-Share Scheduling ☐ Round Robin (RR) ☐ Use Default

Round Robin and Priority Configuration

RR Time Slice: ☒ 5 ticks ☐ 0.2 secs

Priority: ☒ No priority ☐ Non-preemptive ☐ Pre-emptive

Dynamic: ☒ Static ☐ Dynamic

OS Control Views OS Help

STEP STOP SUSPEND

Fast Slow CPU Speed

Use Single CPU ☐ Allow CPU Affinity ☐ Run scheduler with no processes ☐

**PROGRAM LIST**

Program Name	Process	Program
DEADLOCKP1	P5	
DEADLOCKP2		
DEADLOCKP3		
DEADLOCKP4		
LOOPTEST		
LOOPTEST1		

Process Name: P5 Priority: 3 Pages: 1

Display profile on exit ☐ If parent dies, children die ☒

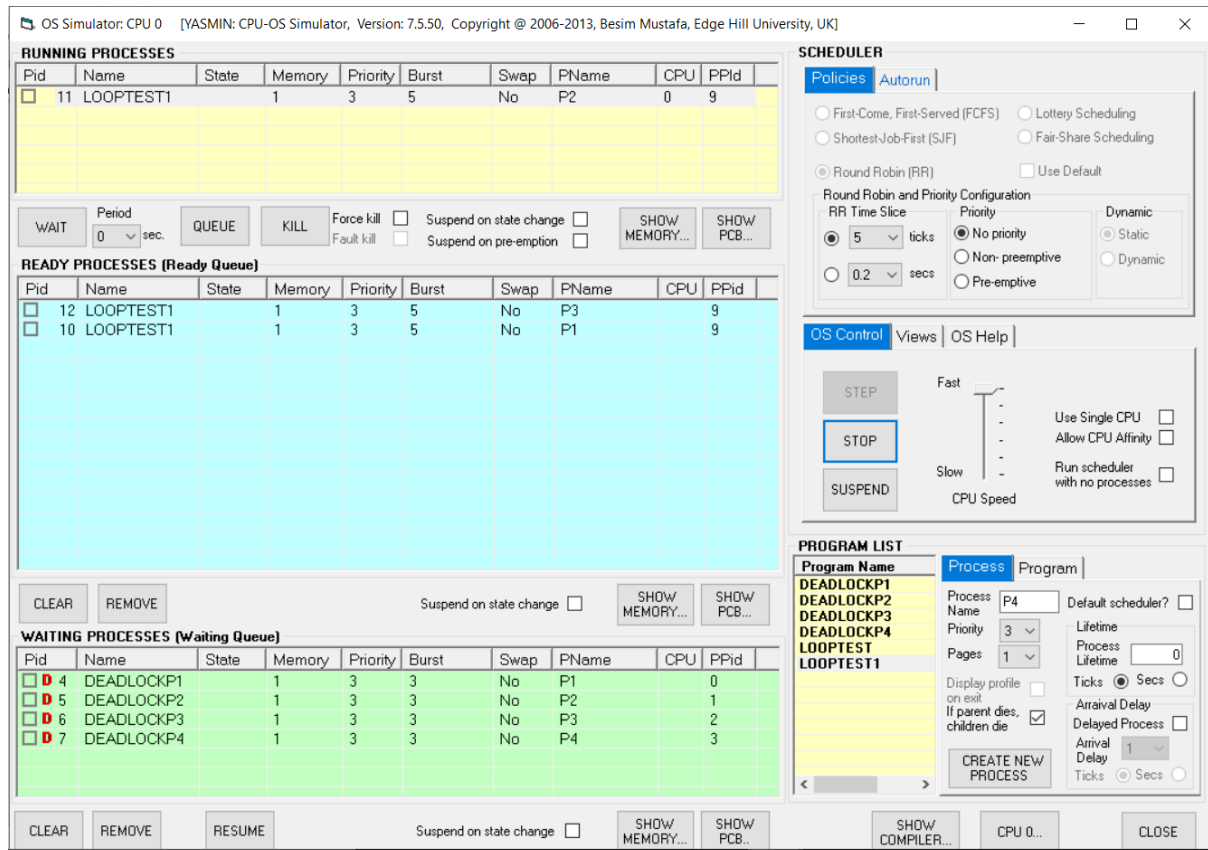
CREATE NEW PROCESS

Lifetime: Process Lifetime: 0 Ticks: ☒ Secs: ☐ Arrival Delay: Delayed Process ☐ Arrival Delay: 1 Ticks: ☐ Secs: ☐

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.

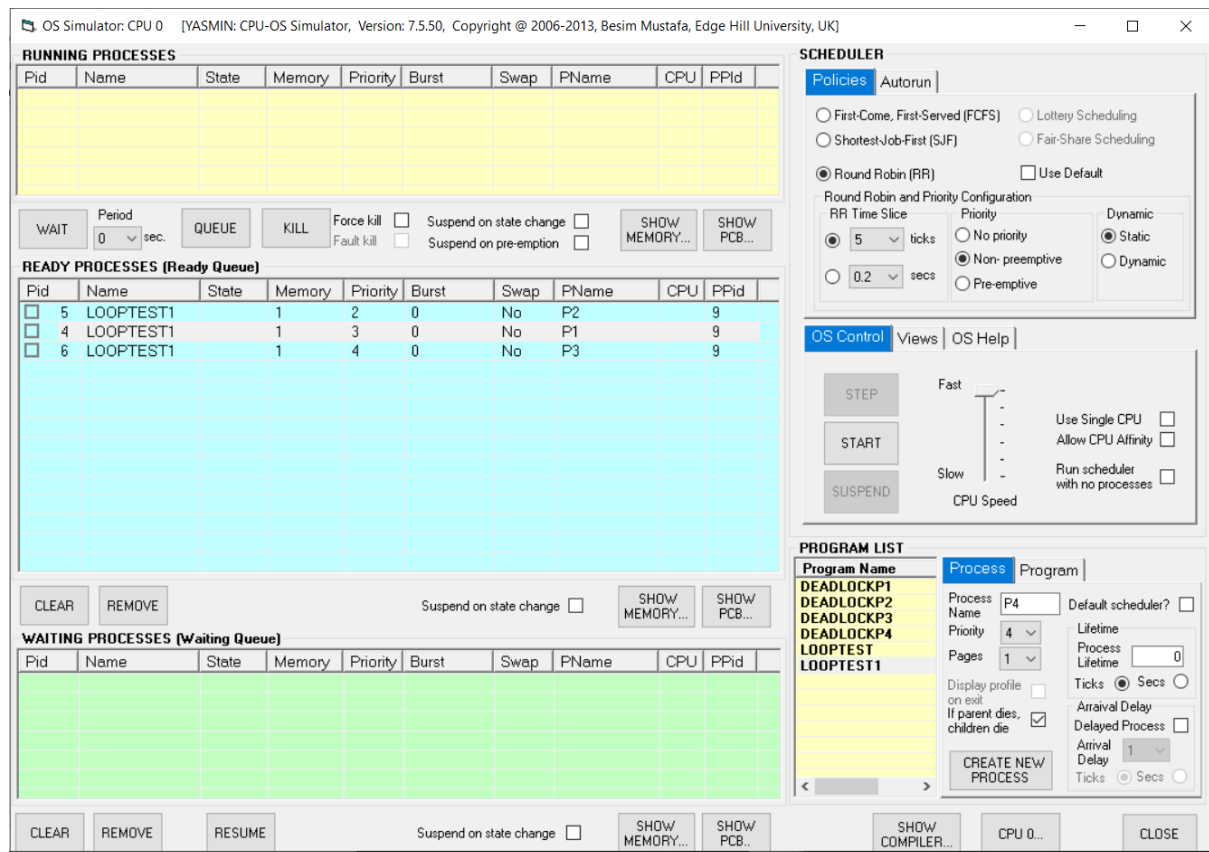
First Process 1 moves from READY to RUNNING. It is there for a short time and we can see the General Purpose CPU registers value changing. It P1 then comes back to READY and P2 moves to RUNNING. Similar to P1, its there for some time and then comes back to READY STATE and P3 takes its place. After P3 has run for some time, it comes back to READY state and P1 moves to RUNNING again. This cycle keeps on for some time then all the three processes finish their running and both the READY and RUNNING states become empty again.





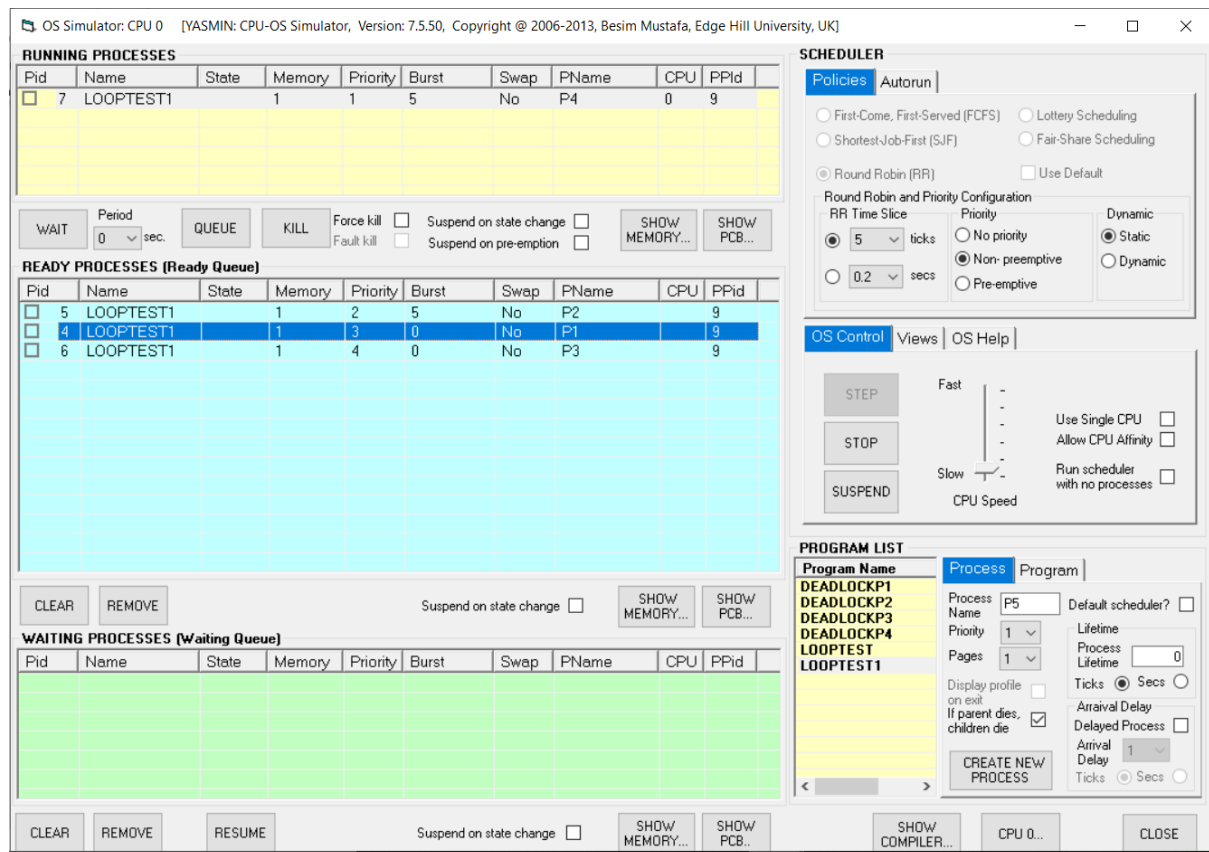
Then select the **Non-preemptive** priority option in the **SCHEDULER/Polices/Priority** frame. Create three processes with the following priorities: 3, 2 and 4. Use the **Priority** drop down list to select priorities. Observe the order in which the three processes are queued in the ready queue represented 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).

The 3 processes are queued in the READY PROCESSES in ascending order of priority.



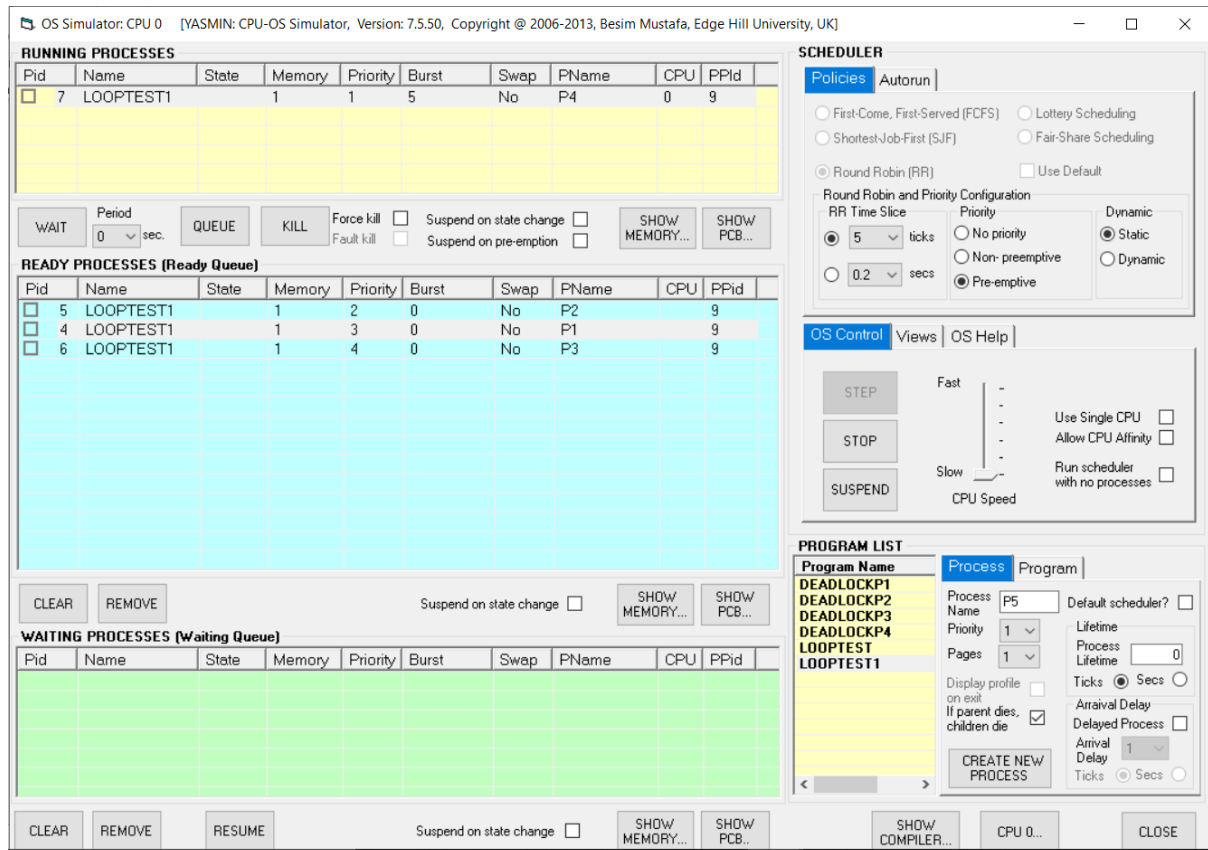
Slide the **Speed** selector to the slowest position and then hit the **START** button. While the first process is being run do the following. Create a fourth process with priority 1. Make a note of what you observe (pay attention to the **READY PROCESSES** view) in the box below.

P1 moves to the top in the READY list and soon moves to the RUNNING PROCESSES after the burst time of currently running process where it gets executed completely although with small pauses where it moves to READY and back to RUNNING. The rest of the processes occur similarly in the increasing order of their priority



Now kill all four processes one by one as they start running. Next, select the **Pre-emptive** option in the **SCHEDULER/Polices/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?

P4 immediately enter RUNNING and is executed till completion. Then P2 enters and is executed till completion. This cycle continuous is ascending order of the priority



## 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.

Total 3 processes get created having the same priority as the main. One is the main program and the remaining two are threads. All 3 are executed partially in a cycle such that they finish at the same time one after the other.



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:

How are the parent/child process relationships represented?

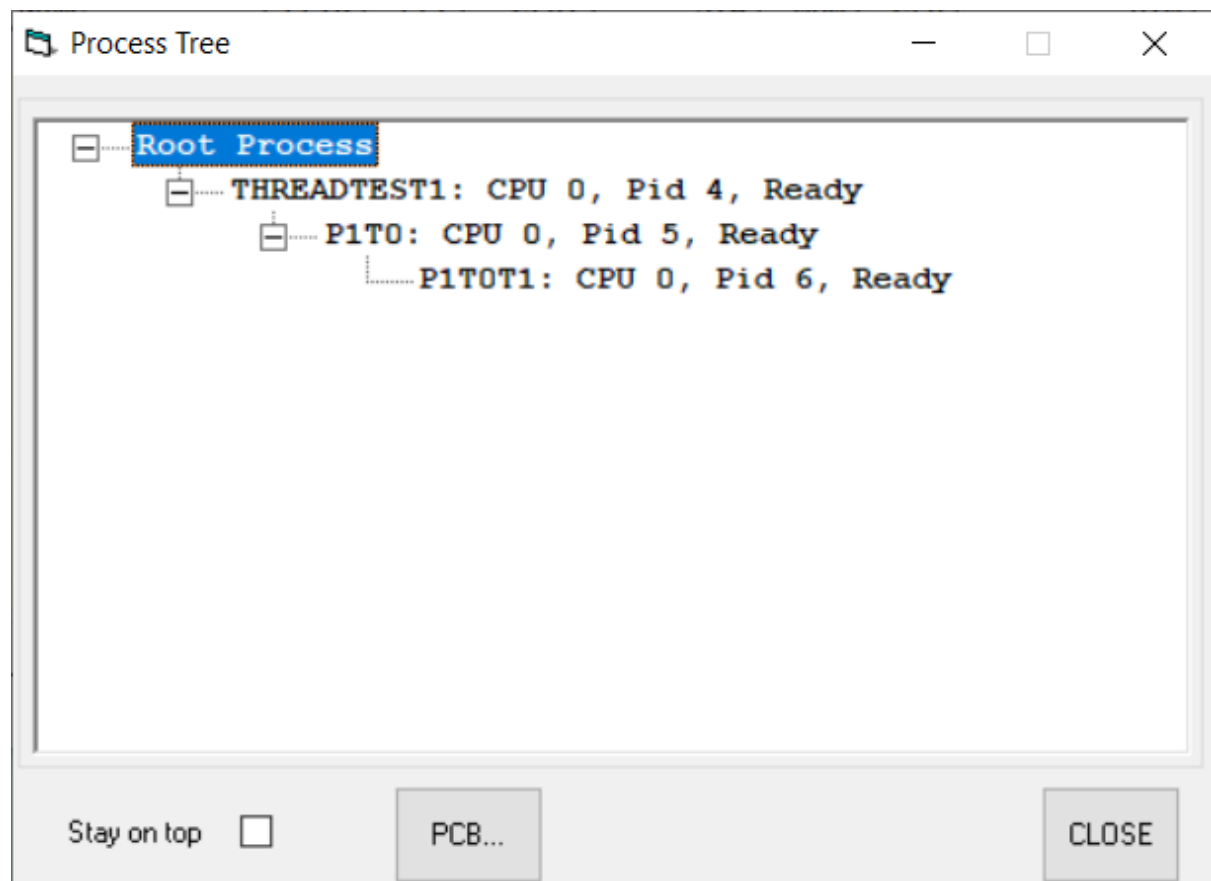
The parent/child process are represented in the form of a tree with the main parent [Root] at the top and the smallest child process at the bottom [P1T0T1]

Identify the parent and the children processes:

The main parent is Root Process. Our program ThreadTest1 is its child. Its state keeps changing from ready to running and back.

ThreadTest1's child is named P1T0 whose state also keeps fluctuating.

P1T0's child is P1T0T1 who state also keeps fluctuating.



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 LIST** view.
- 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 without 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.

```
Thread1 = 25
Thread2 = 18
No, these are not what I expected.
```

```
In main
In thread2In thread1

In main
In thread1
In thread2
thread2 g = 18
Exiting thread2
thread1 g = 25
Exiting thread1
Exiting main
```

<b>INPUT</b>		<b>Colours</b>		<b>Fonts</b>			
<input type="checkbox"/> SHOW KEYBD...	<input type="checkbox"/> Stay on top	<input type="checkbox"/> Screen colour	<input checked="" type="radio"/> SET...	<input type="checkbox"/> SET...	<input type="button" value="PRINT..."/>	<input type="button" value="CLEAR"/>	<input type="button" value="CLOSE"/>
<input type="checkbox"/> No output display	<input type="checkbox"/> Display CPU id	<input type="radio"/> Text colour	<input type="radio"/>				

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

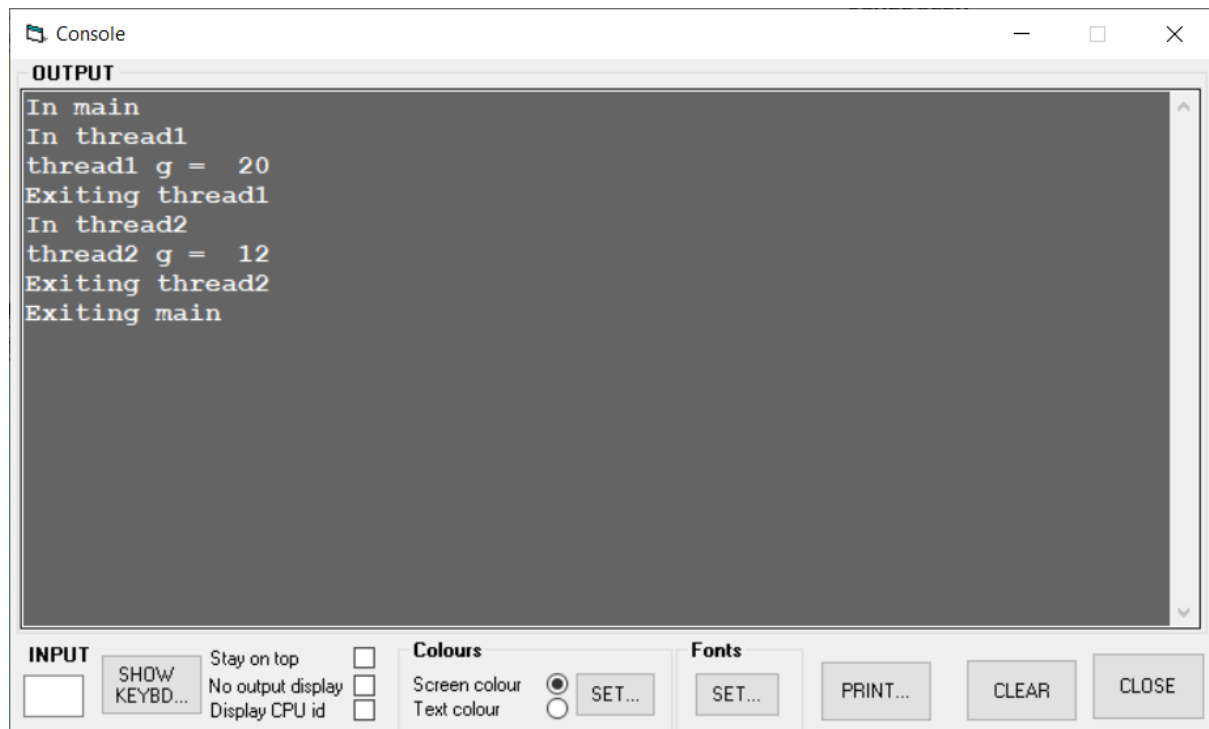
  writeln("In main")

  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**.

Thread 1 = 20 Thread 2 = 12
--------------------------------



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

```
program CriticalRegion2
  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")

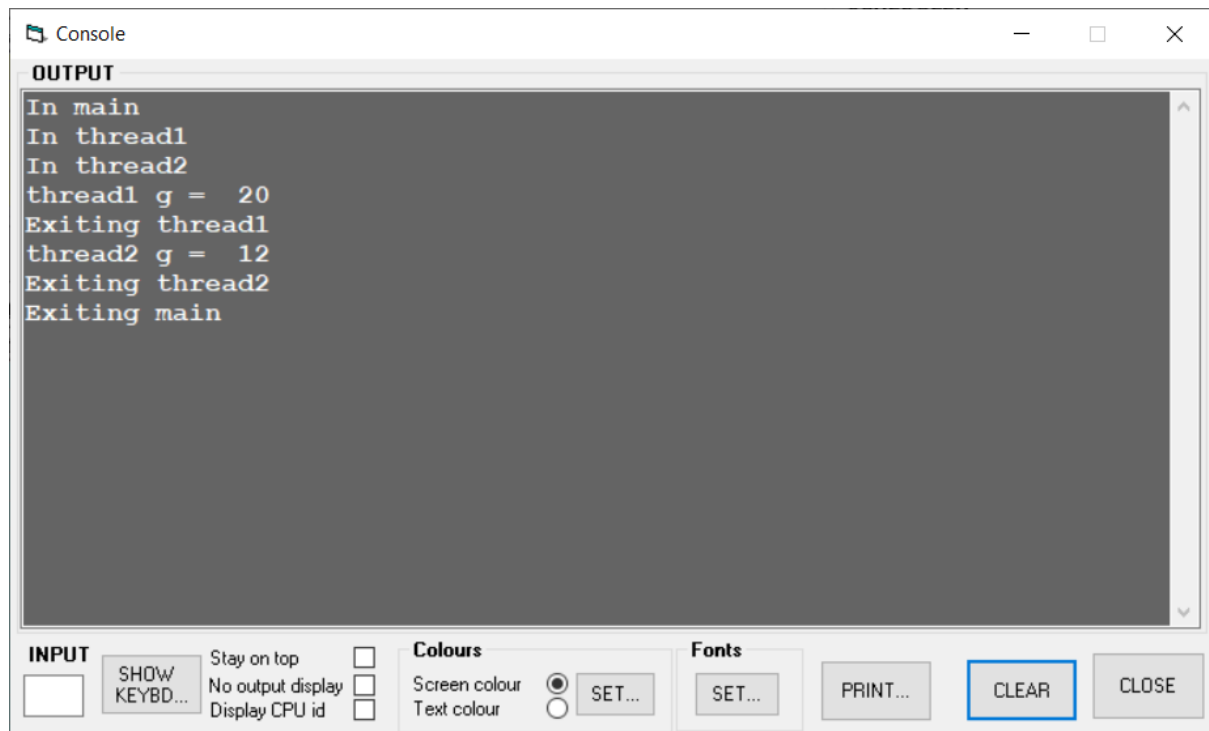
```

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**.

Thread 1 = 20 Thread 2 = 12
--------------------------------



## 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 DeadlockPN
  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. DeadlockP1, DeadlockP2, etc.
- c. Look at your graph you constructed in (1) above and using that information fill in the values for each of the **Xs** and **Ys** 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?

System Resources

Allocating resources for process id

Resource List

Resource	Used by	Requested by	Allocate	Block	Release
R0 (Red Square)	13	17	<input type="button" value="Allocate"/>	<input checked="" type="checkbox"/>	<input type="button" value="Release"/>
R1 (Red Oval)	15	13	<input type="button" value="Allocate"/>	<input checked="" type="checkbox"/>	<input type="button" value="Release"/>
R2 (Red Square)	16	15	<input type="button" value="Allocate"/>	<input checked="" type="checkbox"/>	<input type="button" value="Release"/>
R3 (Red Circle)	17	16	<input type="button" value="Allocate"/>	<input checked="" type="checkbox"/>	<input type="button" value="Release"/>
R4 (Green Oval)			<input type="button" value="Allocate"/>	<input checked="" type="checkbox"/>	<input type="button" value="Release"/>
R5 (Green Square)			<input type="button" value="Allocate"/>	<input checked="" type="checkbox"/>	<input type="button" value="Release"/>

Resource colour key: ■ Available ■ Allocated only ■ Allocated + Requested

Stay on top ☐

**Prevent**

Disallow hold and wait ☐

Disallow circular wait ☒

Use total ordering ☐

**Recover**

Abort processes ☐

Pre-empt resources ☐

**Avoid**

☐ Enable

**Detect**

Do not detect ☐ ☐ Every  sec

☐ Randomly ☐ CPU Utilization <  %

☒ After every alloc. and de-alloc.

Deadlock Count

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

- 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.
- Select one of these resources and click on the **Release** button next to it.
- Observe what is happening to the processes in the OS Simulator window.
- Is the deadlock situation resolved? Explain briefly why this helped resolve the deadlock.

Yes, the deadlock got resolved. This is because that process released whatever resources it was holding which in turn was required by another. This process continued and all the process came out of 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 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.

Yes, this again deleted the process and released the resources it was holding which was required by another process and so on thus the deadlock got resolved

**System Resources**

Allocating resources for process id

**Resource List**

Resource	Used by	Requested by	Allocate	Block	Release
R0	<input type="text"/>	<input type="text"/>	<input type="button" value="Allocate"/>	<input checked="" type="checkbox"/>	<input type="button" value="Release"/>
R1	<input type="text"/>	<input type="text"/>	<input type="button" value="Allocate"/>	<input checked="" type="checkbox"/>	<input type="button" value="Release"/>
R2	<input type="text"/>	<input type="text"/>	<input type="button" value="Allocate"/>	<input checked="" type="checkbox"/>	<input type="button" value="Release"/>
R3	<input type="text"/>	<input type="text"/>	<input type="button" value="Allocate"/>	<input checked="" type="checkbox"/>	<input type="button" value="Release"/>
R4	<input type="text"/>	<input type="text"/>	<input type="button" value="Allocate"/>	<input checked="" type="checkbox"/>	<input type="button" value="Release"/>
R5	<input type="text"/>	<input type="text"/>	<input type="button" value="Allocate"/>	<input checked="" type="checkbox"/>	<input type="button" value="Release"/>

Resource colour key: ■ Available ■ Allocated only ■ Allocated + Requested

**Prevent**

Disallow hold and wait ☐   
 Disallow circular wait ☒   
 Use total ordering ☐

**Recover**

Abort processes ☐   
 Pre-empt resources ☐

**Avoid**

☐ Enable

**Detect**

Do not detect ☐ ☐ Every  sec  
☐ Randomly ☐ CPU Utilization <  %  
☒ After every alloc. and de-alloc.

Deadlock Count

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.

Yes, we are managing the resources such that any resource that will be requested by a process is not held up by another deadlocked process and thus avoids the deadlock at all.

System Resources

Allocating resources for process id

Resource List

Resource	Used by	Requested by	Allocate	Block	Release
R0	<input type="text"/>	<input type="text"/>	<input type="button" value="Allocate"/>	<input checked="" type="checkbox"/>	<input type="button" value="Release"/>
R1	<input type="text"/>	<input type="text"/>	<input type="button" value="Allocate"/>	<input checked="" type="checkbox"/>	<input type="button" value="Release"/>
R2	<input type="text"/>	<input type="text"/>	<input type="button" value="Allocate"/>	<input checked="" type="checkbox"/>	<input type="button" value="Release"/>
R3	<input type="text"/>	<input type="text"/>	<input type="button" value="Allocate"/>	<input checked="" type="checkbox"/>	<input type="button" value="Release"/>
R4	<input type="text"/>	<input type="text"/>	<input type="button" value="Allocate"/>	<input checked="" type="checkbox"/>	<input type="button" value="Release"/>
R5	<input type="text"/>	<input type="text"/>	<input type="button" value="Allocate"/>	<input checked="" type="checkbox"/>	<input type="button" value="Release"/>

Resource colour key: ■ Available ■ Allocated only ■ Allocated + Requested

Stay on top ☐

Prevent	Recover	Avoid	Detect
Disallow hold and wait <input type="checkbox"/> <input type="text" value="0"/> Disallow circular wait <input checked="" type="checkbox"/> <input type="text" value="0"/> Use total ordering <input type="checkbox"/> <input type="text" value="0"/> <input type="button" value="RESET"/>	Abort processes <input type="checkbox"/> <input type="text" value="0"/> Pre-empt resources <input type="checkbox"/> <input type="text" value="0"/> <input type="button" value="RESET"/>	<input type="checkbox"/> Enable	Do not detect <input type="checkbox"/> <input type="radio"/> Every <input type="text" value="1"/> sec <input type="radio"/> Randomly <input type="radio"/> CPU Utilization < <input type="text" value="50"/> % <input checked="" type="radio"/> After every alloc. and de-alloc.

Deadlock Count

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

As seen in the resource allocation Graph, the OS does not allocate a resource if it would lead to breaking of total ordering

System Resources

Allocating resources for process id

**Resource List**

Resource	Used by	Requested by	Used by	Requested by	Used by	Requested by	
	5			6			7
<b>R0</b>	<input type="button" value="Allocate"/>	<input checked="" type="checkbox"/> Block	<b>R1</b>	<input type="button" value="Allocate"/>	<input checked="" type="checkbox"/> Block	<b>R2</b>	<input type="button" value="Allocate"/>
<input type="button" value="Release"/>			<input type="button" value="Release"/>			<input type="button" value="Release"/>	
	8	7					
<b>R3</b>	<input type="button" value="Allocate"/>	<input checked="" type="checkbox"/> Block	<b>R4</b>	<input type="button" value="Allocate"/>	<input checked="" type="checkbox"/> Block	<b>R5</b>	<input type="button" value="Allocate"/>
<input type="button" value="Release"/>			<input type="button" value="Release"/>			<input type="button" value="Release"/>	

Resource colour key:  Available  Allocated only  Allocated + Requested

Stay on top ☐

**Prevent**  
 Disallow hold and wait ☐ 0  
 Disallow circular wait ☐ 0  
 Use total ordering ☒ 9

**Recover**  
 Abort processes ☐ 0  
 Pre-empt resources ☐ 0

**Avoid**  
☐ Enable

**Detect**  
 Do not detect ☐ ☐ Every 1 sec  
☐ Randomly ☐ CPU Utilization < 50 %  
☒ After every alloc. and de-alloc.

Deadlock Count

System Resources

Allocating resources for process id

**Resource List**

Resource	Used by	Requested by	Used by	Requested by	Used by	Requested by	
							
<b>R0</b>	<input type="button" value="Allocate"/>	<input checked="" type="checkbox"/> Block	<b>R1</b>	<input type="button" value="Allocate"/>	<input checked="" type="checkbox"/> Block	<b>R2</b>	<input type="button" value="Allocate"/>
<input type="button" value="Release"/>			<input type="button" value="Release"/>			<input type="button" value="Release"/>	
							
<b>R3</b>	<input type="button" value="Allocate"/>	<input checked="" type="checkbox"/> Block	<b>R4</b>	<input type="button" value="Allocate"/>	<input checked="" type="checkbox"/> Block	<b>R5</b>	<input type="button" value="Allocate"/>
<input type="button" value="Release"/>			<input type="button" value="Release"/>			<input type="button" value="Release"/>	

Resource colour key:  Available  Allocated only  Allocated + Requested

Stay on top ☐

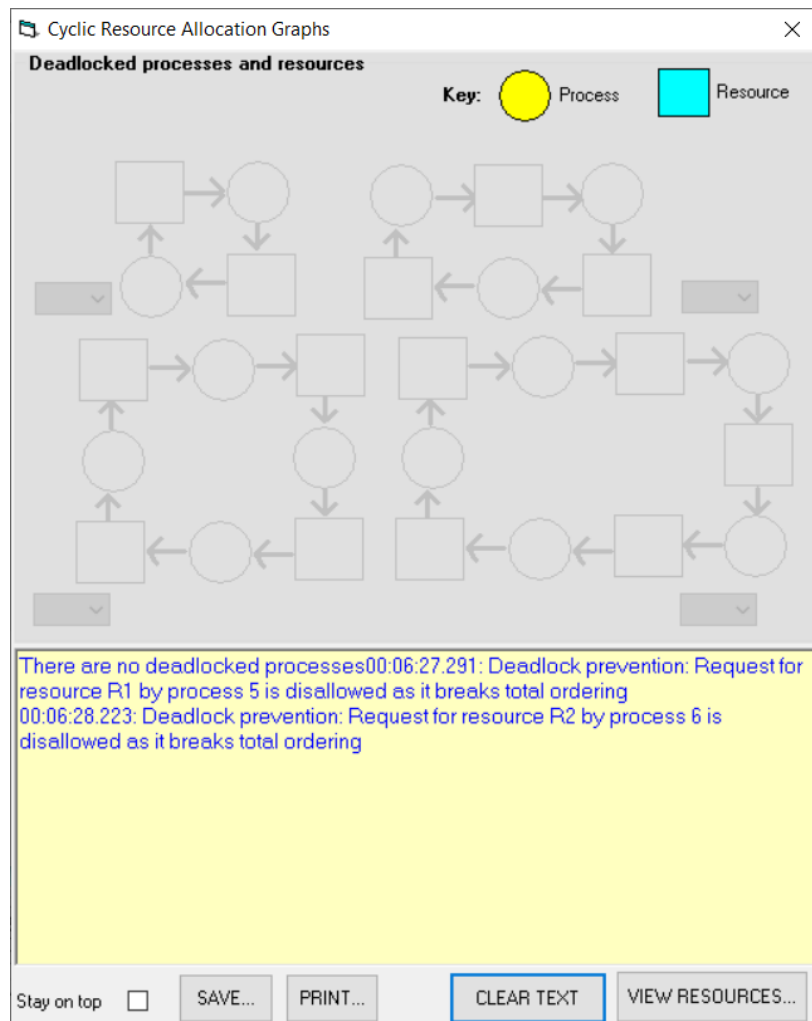
**Prevent**  
 Disallow hold and wait ☐ 0  
 Disallow circular wait ☐ 0  
 Use total ordering ☒ 9

**Recover**  
 Abort processes ☐ 0  
 Pre-empt resources ☐ 0

**Avoid**  
☐ Enable

**Detect**  
 Do not detect ☐ ☐ Every 1 sec  
☐ Randomly ☐ CPU Utilization < 50 %  
☒ After every alloc. and de-alloc.

Deadlock Count



**CONCLUSION:** We used a CPU Simulator to analyse and synthesize process scheduling algorithm, thread creation and synchronization, Deadlock prevention and avoidance algorithms.

# EXPERIMENT - 9

---

## **AIM:**

To write a program to implement the following page replacement policies :-

1. First In First Out [FIFO]
2. Least Recently Used [LRU]
3. Optimal Page Replacement [OPR]

## **DESCRIPTION:**

Page Replacement Algorithm decides which page to remove, also called swap out when a new page needs to be loaded into the main memory. Page Replacement happens when a requested page is not present in the main memory and the available space is not sufficient for allocation to the requested page.

## **TERMS USED:**

1. **Page** - A page is a fixed-length contiguous block of virtual memory, described by a single entry in the page table
2. **Frame** - A frame is the smallest fixed-length contiguous block of physical memory into which memory pages are mapped by the operating system
3. **Paging** - Paging is a memory management scheme that eliminates the need for contiguous allocation of physical memory
4. **Page Table** - A page table is the data structure used by a virtual memory system in a computer operating system to store the mapping between virtual addresses and physical addresses

**5. Page Fault** - A page fault happens when a running program accesses a memory page that is mapped into the virtual address space but not loaded in physical memory

## FIRST IN FIRST OUT (FIFO):

This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.

**Example-1** Consider page reference string 1, 3, 0, 3, 5, 6, 3 with 3 page frames. Find number of page faults.

Page reference						
1, 3, 0, 3, 5, 6, 3						
1	3	0	3	5	6	3
		0	0	0	0	3
	3	3	3	3	6	6
1	1	1	1	5	5	5
Miss	Miss	Miss	Hit	Miss	Miss	Miss

Total Page Fault = 6

Initially all slots are empty, so when 1, 3, 0 came they are allocated to the empty slots -> **3 Page Faults**.



when 3 comes, it is already in memory so -> **0 Page Faults.**

Then 5 comes, it is not available in memory so it replaces the oldest page slot i.e 1. ->**1 Page Fault.**

6 comes, it is also not available in memory so it replaces the oldest page slot i.e 3 ->**1 Page Fault.**

Finally when 3 come it is not available so it replaces 0  
**1 page fault**

- Advantages –

- 1.It is simple and easy to understand & implement.

- Disadvantages –

- 1.The process effectiveness is low.
- 2.When we increase the number of frames while using FIFO, we are giving more memory to processes. So, page fault should decrease, but here the page faults are increasing. This problem is called Belady's Anomaly.
- 3.Every frame needs to be taken into account off.

## LEAST RECENTLY USED (LRU):

In this algorithm page will be replaced which is least recently used.

**Example-3** Consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 with 4 page frames. Find number of page faults.

Page reference	7,0,1,2,0,3,0,4,2,3,0,3,2,3													No. of Page frame - 4
7	0	1	2	0	3	0	4	2	3	0	3	2	3	
			2	2	2	2	2	2	2	2	2	2	2	
		1	1	1	1	1	4	4	4	4	4	4	4	
	0	0	0	0	0	0	0	0	0	0	0	0	0	
7	7	7	7	7	3	3	3	3	3	3	3	3	3	
Miss	Miss	Miss	Miss	Hit	Miss	Hit	Miss	Hit	Hit	Hit	Hit	Hit	Hit	
Total Page Fault = 6														

Here LRU has same number of page fault as optimal but it may differ according to question.

Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots -> **4 Page faults**

0 is already there so -> **0 Page fault.**

when 3 came it will take the place of 7 because it is least recently used -> **1 Page fault**

0 is already in memory so -> **0 Page fault.**

4 will take place of 1 -> **1 Page Fault**

Now for the further page reference string -> **0 Page fault** because they are already available in the memory.

- Advantages –

1. It is open for full analysis.
2. In this, we replace the page which is least recently used, thus free from Belady's Anomaly.
3. Easy to choose page which has faulted and hasn't been used for a long time.

- Disadvantages –

1. It requires additional Data Structure to be implemented.
2. Hardware assistance is high.

## OPTIMAL PAGE REPLACEMENT (OPR):

In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.

Example-2: Consider the page references 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, with 4 page frame. Find number of page fault.

Page reference	7,0,1,2,0,3,0,4,2,3,0,3,2,3														No. of Page frame - 4	
7	0	1	2	0	3	0	4	2	3	0	3	2	3			
			2	2	2	2	2	2	2	2	2	2	2			
		1	1	1	1	1	4	4	4	4	4	4	4			
	0	0	0	0	0	0	0	0	0	0	0	0	0			
7	7	7	7	7	3	3	3	3	3	3	3	3	3			
Miss	Miss	Miss	Miss	Hit	Miss	Hit	Miss	Hit	Hit	Hit	Hit	Hit	Hit			
Total Page Fault = 6																

Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots -> 4 Page faults

0 is already there so -> 0 Page fault.

when 3 came it will take the place of 7 because it is not used for the longest duration of time in the future.->1 Page fault.

0 is already there so -> 0 Page fault..

4 will takes place of 1 -> 1 Page Fault.

Now for the further page reference string -> 0 Page fault because they are already available in the memory.

Optimal page replacement is perfect, but not possible in practice as the operating system cannot know future requests. The use of Optimal Page replacement is to set up a benchmark so that other replacement algorithms can be analyzed against it.

- Advantages –

1. Complexity is less and easy to implement.
2. Assistance needed is low i.e Data Structure used are easy and light.

- Disadvantages –

1. OPR is perfect, but not possible in practice as the operating system cannot know future requests.
2. Error handling is tough.

- FIFO:

CODE:

```
import java.io.*;
public class FIFO {

    public static void main(String[] args) throws IOException
    {
        BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
        int frames, pointer = 0, hit = 0, fault = 0, ref_len;
        int buffer[];
        int reference[];
        int mem_layout[][];

        System.out.println("Please enter the number of Frames: ");
        frames = Integer.parseInt(br.readLine());
```

```

        System.out.println("Please enter the length of the
Reference string: ");
        ref_len = Integer.parseInt(br.readLine());

        reference = new int[ref_len];
        mem_layout = new int[ref_len][frames];
        buffer = new int[frames];
        for(int j = 0; j < frames; j++)
            buffer[j] = 99;

        System.out.println("Please enter the reference string: ");
        for(int i = 0; i < ref_len; i++)
        {
            reference[i] = Integer.parseInt(br.readLine());
        }
        System.out.println();
        for(int i = 0; i < ref_len; i++)
        {
            int search = -1;
            for(int j = 0; j < frames; j++)
            {
                if(buffer[j] == reference[i])
                {
                    search = j;
                    hit++;
                    break;
                }
            }
            if(search == -1)
            {
                buffer[pointer] = reference[i];
                fault++;
                pointer++;
                if(pointer == frames)
                    pointer = 0;
            }
            for(int j = 0; j < frames; j++)
                mem_layout[i][j] = buffer[j];
        }
    }

```

```

    for(int i = 0; i < frames; i++)
    {
        for(int j = 0; j < ref_len; j++)
            System.out.printf("%3d ", mem_layout[j][i]);
        System.out.println();
    }

    float hit_ratio = ((float)hit/ref_len);
    System.out.println("The number of Hits: " + hit);
    System.out.println("The number of Faults: " + fault);
    System.out.println("Hit Ratio: " + (float)hit_ratio);
    System.out.println("Miss Ratio: "+(float)(1-hit_ratio));

}

}

```

OUTPUT :

```
IFO'
Please enter the number of Frames:
3
Please enter the length of the Reference string:
11
Please enter the reference string:
1
2
3
4
2
1
5
3
2
4
6

    1   1   1   4   4   4   4   3   3   3   6
99   2   2   2   2   1   1   1   2   2   2
99 99   3   3   3   3   5   5   5   4   4
The number of Hits: 1
The number of Faults: 10
Hit Ratio: 0.09090909
Miss Ratio: 0.9090909
PS: C:\Users\juna1\OneDrive - Shri Vile Parle Kelavani M
```



- LRU

### CODE:

```
import java.io.*;
import java.util.*;

public class LRU {

    public static void main(String[] args) throws IOException
    {
        BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
        int frames, pointer = 0, hit = 0, fault = 0, ref_len;
        Boolean isFull = false;
        int buffer[];
        ArrayList<Integer> stack = new ArrayList<Integer>();
        int reference[];
        int mem_layout[][];

        System.out.println("Please enter the number of Frames: ");
        frames = Integer.parseInt(br.readLine());

        System.out.println("Please enter the length of the
Reference string: ");
        ref_len = Integer.parseInt(br.readLine());

        reference = new int[ref_len];
        mem_layout = new int[ref_len][frames];
        buffer = new int[frames];
        for(int j = 0; j < frames; j++)
            buffer[j] = 99;

        System.out.println("Please enter the reference string: ");
        for(int i = 0; i < ref_len; i++)
        {
            reference[i] = Integer.parseInt(br.readLine());
        }
    }
}
```

```

System.out.println();
for(int i = 0; i < ref_len; i++)
{
    if(stack.contains(reference[i]))
    {
        stack.remove(stack.indexOf(reference[i]));
    }
    stack.add(reference[i]);
    int search = -1;
    for(int j = 0; j < frames; j++)
    {
        if(buffer[j] == reference[i])
        {
            search = j;
            hit++;
            break;
        }
    }
    if(search == -1)
    {
        if(isFull)
        {
            int min_loc = ref_len;
            for(int j = 0; j < frames; j++)
            {
                if(stack.contains(buffer[j]))
                {
                    int temp = stack.indexOf(buffer[j]);
                    if(temp < min_loc)
                    {
                        min_loc = temp;
                        pointer = j;
                    }
                }
            }
        }
        buffer[pointer] = reference[i];
        fault++;
        pointer++;
        if(pointer == frames)
        {

```

```

        pointer = 0;
        isFull = true;
    }
}
for(int j = 0; j < frames; j++)
    mem_layout[i][j] = buffer[j];
}

for(int i = 0; i < frames; i++)
{
    for(int j = 0; j < ref_len; j++)
        System.out.printf("%3d ", mem_layout[j][i]);
    System.out.println();
}

float hit_ratio = ((float)hit/ref_len);

System.out.println("The number of Hits: " + hit);
System.out.println("Hit Ratio: " + (float)hit_ratio);
System.out.println("Miss Ratio: "+(float)(1 - hit_ratio));
System.out.println("The number of Faults: " + fault);
}

}

```

OUTPUT:

```

Please enter the number of Frames:
3
Please enter the length of the Reference string:
20
Please enter the reference string:
7
0
1
2
0
3
0
4
2
3
0
3
2
1
2
0
1
7
0
1

  7  7  7  2  2  2  2  4  4  4  0  0  0  1  1  1  1  1  1  1
-1  0  0  0  0  0  0  0  0  3  3  3  3  3  3  0  0  0  0  0
-1 -1  1  1  1  3  3  3  2  2  2  2  2  2  2  2  7  7  7

The number of Hits: 8
Hit Ratio: 0.4
Miss Ratio: 0.6
The number of Faults: 12

```

- OPTIMAL

CODE :

```

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
public class OptimalReplacement {

    public static void main(String[] args) throws IOException
    {
        BufferedReader br = new BufferedReader(new
        InputStreamReader(System.in));
        int frames, pointer = 0, hit = 0, fault = 0, ref_len;
        boolean isFull = false;
        int buffer[];
    }
}

```

```

int reference[];
int mem_layout[][];

System.out.println("Please enter the number of Frames: ");
frames = Integer.parseInt(br.readLine());

System.out.println("Please enter the length of the
Reference string: ");
ref_len = Integer.parseInt(br.readLine());

reference = new int[ref_len];
mem_layout = new int[ref_len][frames];
buffer = new int[frames];
for(int j = 0; j < frames; j++)
    buffer[j] = 99;

System.out.println("Please enter the reference string: ");
for(int i = 0; i < ref_len; i++)
{
    reference[i] = Integer.parseInt(br.readLine());
}
System.out.println();
for(int i = 0; i < ref_len; i++)
{
    int search = -1;
    for(int j = 0; j < frames; j++)
    {
        if(buffer[j] == reference[i])
        {
            search = j;
            hit++;
            break;
        }
    }
    if(search == -1)
    {
        if(isFull)
        {
            int index[] = new int[frames];
            boolean index_flag[] = new boolean[frames];
            for(int j = i + 1; j < ref_len; j++)

```

```

        {
            for(int k = 0; k < frames; k++)
            {
                if((reference[j] == buffer[k]) && (index_flag[k] ==
false))
                {
                    index[k] = j;
                    index_flag[k] = true;
                    break;
                }
            }
        }
        int max = index[0];
        pointer = 0;
        if(max == 0)
            max = 200;
        for(int j = 0; j < frames; j++)
        {
            if(index[j] == 0)
                index[j] = 200;
            if(index[j] > max)
            {
                max = index[j];
                pointer = j;
            }
        }
        buffer[pointer] = reference[i];
        fault++;
        if(!isFull)
        {
            pointer++;
            if(pointer == frames)
            {
                pointer = 0;
                isFull = true;
            }
        }
    }
    for(int j = 0; j < frames; j++)
        mem_layout[i][j] = buffer[j];

```

```

    }

    for(int i = 0; i < frames; i++)
    {
        for(int j = 0; j < ref_len; j++)
            System.out.printf("%3d ", mem_layout[j][i]);
        System.out.println();
    }
    float hit_ratio = ((float)hit/ref_len);
    System.out.println("The number of Hits: " + hit);
    System.out.println("Hit Ratio: " + (float)hit_ratio);
    System.out.println("The number of Faults: " + fault);
    System.out.println("Miss Ratio: "+(1-hit_ratio));
}
}

```

## OUTPUT :

```

Please enter the number of Frames:
3
Please enter the length of the Reference string:
11
Please enter the reference string:
1
2
3
4
2
1
5
3
2
4
6

    1   1   1   1   1   1   5   3   3   3   6
99   2   2   2   2   2   2   2   2   2   2
99 99   3   4   4   4   4   4   4   4   4

The number of Hits: 4
Hit Ratio: 0.363637
The number of Faults: 7
Miss Ratio: 0.636363
PS: C:\Users\juna\OneDrive - Shri Vile Parle Kelaver

```

## **CONCLUSION:**

We have implemented various page replacement policies and have tested them on standard input. This helps to highlight their strengths and weaknesses thus allowing us to differentiate between them. We have learned about the 'Optimal' policy which cannot be implemented in real systems as it requires knowledge of future events and its use as a comparative baseline for evaluating other policies.



# EXPERIMENT – 10

---

**AIM:** Implement Disk Scheduling Algorithms

## **THEORY:**

Disk scheduling is done by operating systems to schedule I/O requests arriving for the disk. Disk scheduling is also known as I/O scheduling.

## **Importance of Disk Scheduling:**

Multiple I/O requests may arrive by different processes and only one I/O request can be served at a time by the disk controller. Thus other I/O requests need to wait in the waiting queue and need to be scheduled.

Two or more requests may be far from each other so can result in greater disk arm movement.

Hard drives are one of the slowest parts of the computer system and thus need to be accessed in an efficient manner.

## **Important Terms in a Disk Scheduling Algorithm:**

- **Seek Time:** Seek time is the time taken to locate the disk arm to a specified track where the data is to be read or write. So the disk scheduling algorithm that gives minimum average seek time is better.

- **Rotational Latency:** Rotational Latency is the time taken by the desired sector of disk to rotate into a position so that it can access the read/write heads. So the disk scheduling algorithm that gives minimum rotational latency is better.
- **Transfer Time:** Transfer time is the time to transfer the data. It depends on the rotating speed of the disk and number of bytes to be transferred.
- **Disk Access Time:** Disk Access Time is:

$$\text{Disk Access Time} = \text{Seek Time} + \text{Rotational Latency} + \text{Transfer Time}$$

- **Disk Response Time:** Response Time is the average of time spent by a request waiting to perform its I/O operation. Average Response time is the response time of all requests. Variance Response Time is the measure of how individual requests are serviced with respect to average response time. So the disk scheduling algorithm that gives minimum variance response time is better.

## DISK SCHEDULING ALGORITHMS:

The different types of disk scheduling algorithm are:-

- FCFS scheduling algorithm
- SSTF (shortest seek time first) algorithm
- SCAN scheduling
- C-SCAN scheduling
- LOOK Scheduling
- C-LOOK scheduling

### **SSTF Disk Scheduling Algorithm-**

- SSTF stands for Shortest Seek Time First.
- This algorithm services that request next which requires least number of head movements from its current position regardless of the direction.
- It breaks the tie in the direction of head movement.

### **Advantages-**

- It reduces the total seek time as compared to FCFS.
- It provides increased throughput.
- It provides less average response time and waiting time.

## **Disadvantages-**

- There is an overhead of finding out the closest request.
- The requests which are far from the head might starve for the CPU.
- It provides high variance in response time and waiting time.
- Switching the direction of head frequently slows down the algorithm.

## **C-SCAN Disk Scheduling Algorithm-**

- Circular-SCAN Algorithm is an improved version of the SCAN Algorithm.
- Head starts from one end of the disk and move towards the other end servicing all the requests in between.
- After reaching the other end, head reverses its direction.
- It then returns to the starting end without servicing any request in between.
- The same process repeats.

## **Advantages-**

- The waiting time for the cylinders just visited by the head is reduced as compared to the SCAN Algorithm.

- It provides uniform waiting time.
- It provides better response time.

## **Disadvantages-**

- It causes more seek movements as compared to SCAN Algorithm.
- It causes the head to move till the end of the disk even if there are no requests to be serviced.

## **SSTF ALGORITHM:**

### **CODE:**

```
import java.util.Scanner;

class node {

    int distance = 0;
    boolean accessed = false;
}

public class SSTF {

    // Calculates difference of each track number with the head
    position
    public static void calculateDifference(int queue[],
                                         int head, node diff[])

    {
        for (int i = 0; i < diff.length; i++)
            diff[i].distance = Math.abs(queue[i] - head);
    }

    public static int findMin(node diff[])
    {
```

```

        int index = -1, minimum = Integer.MAX_VALUE;

        for (int i = 0; i < diff.length; i++) {
            if (!diff[i].accessed && minimum > diff[i].distance) {

                minimum = diff[i].distance;
                index = i;
            }
        }
        return index;
    }

    public static void shortestSeekTimeFirst(int request[], int head)
    {
        if (request.length == 0)
            return;

        // create array of objects of class node
        node diff[] = new node[request.length];

        for (int i = 0; i < diff.length; i++)

            diff[i] = new node();

        int seek_count = 0;

        // stores sequence in which disk access is done
        int[] seek_sequence = new int[request.length + 1];

        for (int i = 0; i < request.length; i++) {

            seek_sequence[i] = head;
            calculateDifference(request, head, diff);

            int index = findMin(diff);

            diff[index].accessed = true;
            seek_count += diff[index].distance;
            head = request[index];
        }

        // for last accessed track

```

```

        seek_sequence[seek_sequence.length - 1] = head;

        System.out.println("Total number of seek operations = "
                                + seek_count);

        System.out.println("Seek Sequence is");

        // print the sequence
        for (int i = 0; i < seek_sequence.length; i++)
            System.out.print(seek_sequence[i] + " -> ");

    }

    public static void main(String[] args)
    {
        int n;
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter the number of elements you want to
store: ");
        n=sc.nextInt();

        int[] arr = new int[n];
        System.out.print("Enter the elements of the array: ");

        for(int i=0; i<n; i++)
        {
            arr[i]=sc.nextInt();
        }

        System.out.print("Enter Initial Head Position: ");
        int head_pos = sc.nextInt();
        shortestSeekTimeFirst(arr, head_pos);
    }
}

```

## OUTPUT:

```
java SSTF
Enter the number of elements you want to store: 8
Enter the elements of the array: 176 79 34 60 92 11 41 114
Enter Initial Head Position: 50
Total number of seek operations = 204
Seek Sequence is
50 -> 41 -> 34 -> 11 -> 60 -> 79 -> 92 -> 114 -> 176 ->
```

## CSCAN ALGORITHM:

### CODE:

```
import java.util.*;

class CscanDiskSchedulingAlgo {

    static int size = 8;
    static int disk_size = 200;

    public static void CSCAN(int arr[], int head)
    {
        int seek_count = 0;
        int distance, cur_track;

        Vector<Integer> left = new Vector<Integer>();
        Vector<Integer> right = new Vector<Integer>();
        Vector<Integer> seek_sequence
            = new Vector<Integer>();

        left.add(0);
        right.add(disk_size - 1);

        for (int i = 0; i < size; i++) {
            if (arr[i] < head)
                left.add(arr[i]);
            if (arr[i] > head)
```



```

        right.add(arr[i]);
    }

    // Sorting left and right vectors
    Collections.sort(left);
    Collections.sort(right);

    // First service the requests
    // on the right side of the
    // head.
    for (int i = 0; i < right.size(); i++) {
        cur_track = right.get(i);

        seek_sequence.add(cur_track);

        distance = Math.abs(cur_track - head);

        seek_count += distance;

        head = cur_track;
    }

    // Once reached the right end
    // jump to the beggining.
    head = 0;

    // adding seek count for head returning from 199 to
    // 0
    seek_count += (disk_size - 1);

    // Now service the requests again
    // which are left.
    for (int i = 0; i < left.size(); i++) {
        cur_track = left.get(i);

        seek_sequence.add(cur_track);

        distance = Math.abs(cur_track - head);

        seek_count += distance;
    }

```

```

        head = cur_track;
    }

    System.out.println("Total number of seek "+ "operations =
" + seek_count);

    System.out.println("Seek Sequence is");

    for (int i = 0; i < seek_sequence.size(); i++) {
        System.out.print(seek_sequence.get(i) + " -> ");
    }
}

public static void main(String[] args) throws Exception
{
    int n;
    Scanner sc=new Scanner(System.in);
    System.out.print("Enter the number of elements you want to
store: ");
    n=sc.nextInt();

    int[] arr = new int[n];
    System.out.print("Enter the elements of the array: ");

    for(int i=0; i<n; i++)
    {
        arr[i]=sc.nextInt();
    }

    System.out.print("Enter Initial Head Position: ");
    int head_pos = sc.nextInt();

    CSCAN(arr, head_pos);
}
}

```

## OUTPUT:

```
ava CscanDiskSchedulingAlgo
Enter the number of elements you want to store: 8
Enter the elements of the array: 176 79 34 60 92 11 41 114
Enter Initial Head Position: 50
Total number of seek operations = 389
Seek Sequence is
60 -> 79 -> 92 -> 114 -> 176 -> 199 -> 0 -> 11 -> 34 -> 41 ->
```

**CONCLUSION:** We learnt about different Disk Scheduling Algorithms, the advantages and disadvantages of each and implemented SSTF and CSCAN in a java program.