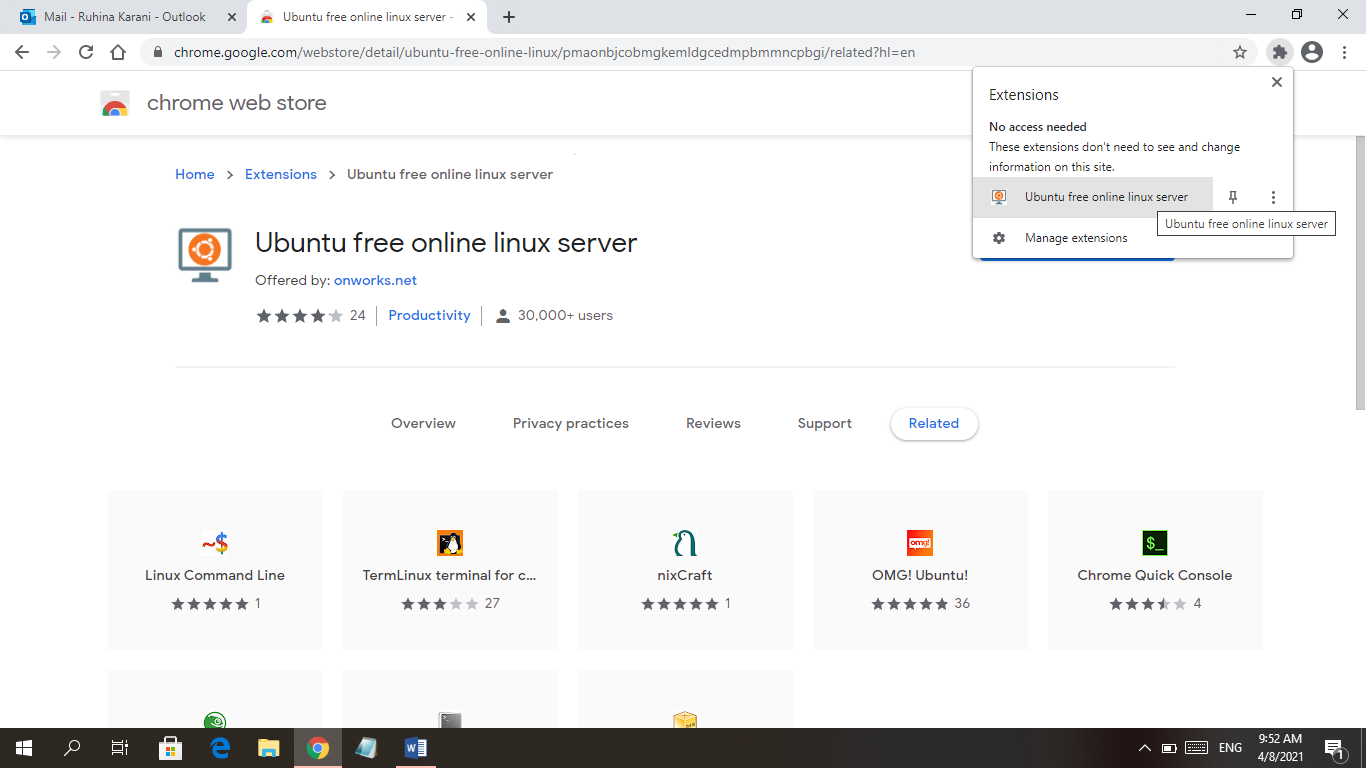
**Adding Linux Terminal as Extension in Google Chrome**

**Step 1- Click on the below link to add UBUNTU Free online linux server**

[**https://chrome.google.com/webstore/detail/ubuntu-free-online-linux/pmaonbjcobmgkemldgcedmpbmmncpbgi?hl=en**](https://chrome.google.com/webstore/detail/ubuntu-free-online-linux/pmaonbjcobmgkemldgcedmpbmmncpbgi?hl=en)

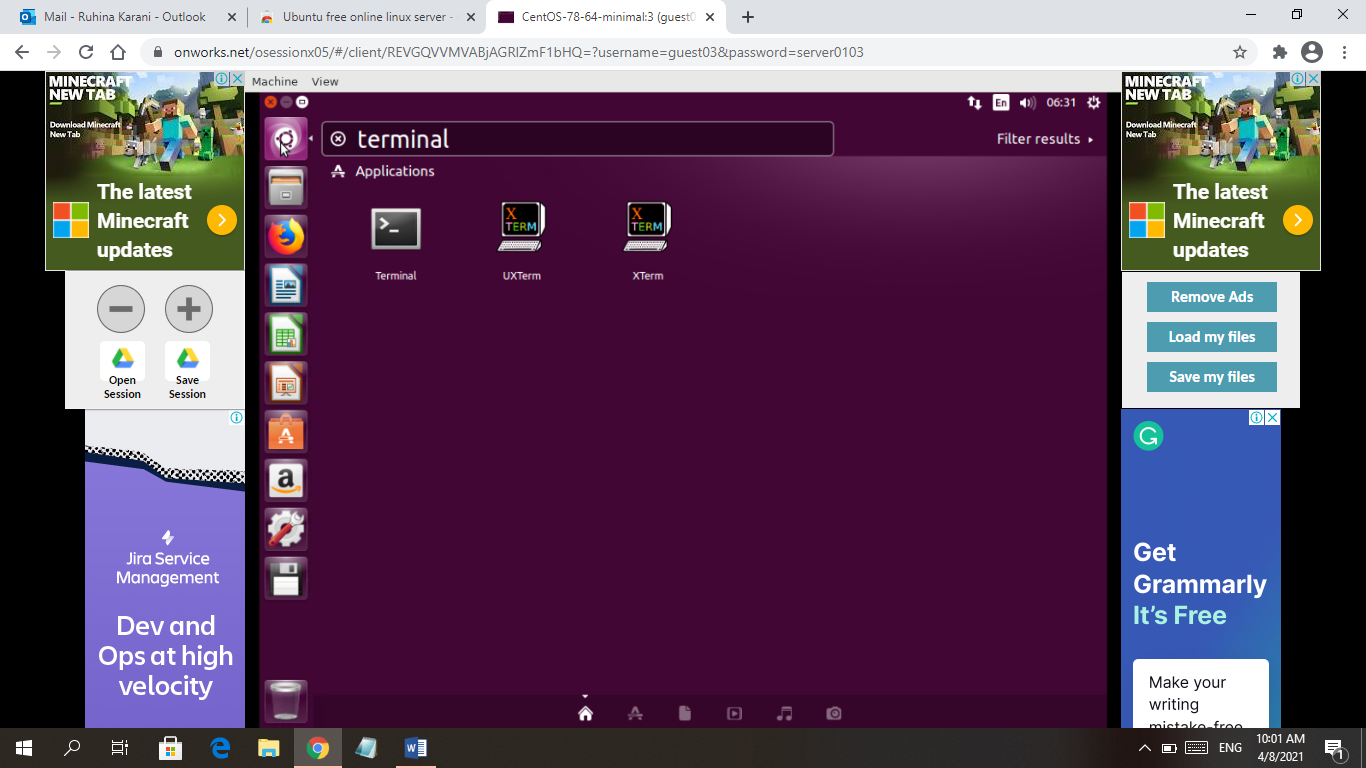
**Step 2- Click Add to chrome-> Add extension**

**Step 3- Click on extensions on topmost right hand corner-> UBUNTU Free online linux server**

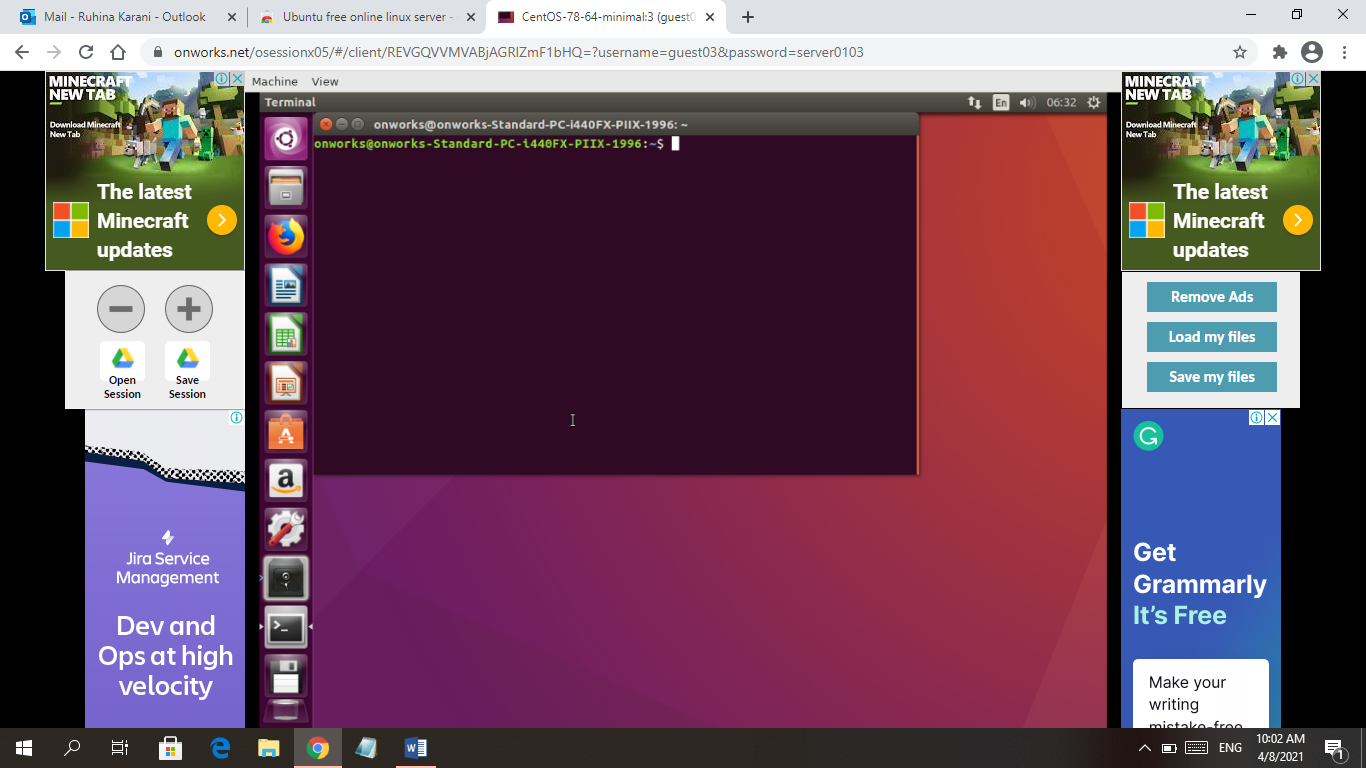


**Step 4- Click Ubuntu GNOME-> Start-> Enter**

**Step 5- Click on Ubuntu ICON-> type terminal**



**Step 6- Click on terminal and start practicing your commands.**



**Experiment No. 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.
7. Display top 10 processes in descending order

**onworks@onworks-Standard-PC-i440FX-PIIX-1996:~$ echo "top 10 processes in descending order"**

top 10 processes in descending order

**onworks@onworks-Standard-PC-i440FX-PIIX-1996:~$ ps axl | head -n 10**

F UID PID PPID PRI NI VSZ RSS WCHAN STAT TTY TIME COMMAND

4 0 1 0 20 0 25084 5032 - Ss ? 0:02 /sbin/init splash

1 0 2 0 20 0 0 0 - S ? 0:00 [kthreadd]

1 0 3 2 20 0 0 0 - I ? 0:00 [kworker/0:0]

1 0 4 2 0 -20 0 0 - I< ? 0:00 [kworker/0:0H]

1 0 6 2 0 -20 0 0 - I< ? 0:00 [mm\_percpu\_wq]

1 0 7 2 20 0 0 0 - S ? 0:00 [ksoftirqd/0]

1 0 8 2 20 0 0 0 - I ? 0:03 [rcu\_sched]

1 0 9 2 20 0 0 0 - I ? 0:00 [rcu\_bh]

1. 0 10 2 -100 - 0 0 - S ? 0:00 [migration/0]
2. Display processes with highest memory usage.

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

PID PPID CMD %MEM %CPU

2 0 [kthreadd] 0.0 0.0

3 2 [kworker/0:0] 0.0 0.0

4 2 [kworker/0:0H] 0.0 0.0

6 2 [mm\_percpu\_wq] 0.0 0.0

7 2 [ksoftirqd/0] 0.0 0.0

8 2 [rcu\_sched] 0.0 1.1

1. 2 [rcu\_bh] 0.0 0.0
2. Display current logged in user and no. of users

**onworks@onworks-Standard-PC-i440FX-PIIX-1996:~$ who –u**

onworks tty7 2019-08-30 19:49 old 830 (:0)

**onworks@onworks-Standard-PC-i440FX-PIIX-1996:~$ who -u | wc -l**

1

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

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

onworks

**onworks@onworks-Standard-PC-i440FX-PIIX-1996:~$ uname**

Linux

**onworks@onworks-Standard-PC-i440FX-PIIX-1996:~$ pwd**

/home/onworks

**onworks@onworks-Standard-PC-i440FX-PIIX-1996:~$ uname**

Linux

1. Display OS version, release number.

**onworks@onworks-Standard-PC-i440FX-PIIX-1996:~$ uname -a**

Linux onworks-Standard-PC-i440FX-PIIX-1996 4.15.0-50-generic #54~16.04.1-Ubuntu SMP Wed May 8 15:50:20 UTC 2019 i686 i686 i686 GNU/Linux

**onworks@onworks-Standard-PC-i440FX-PIIX-1996:~$ uname -r**

4.15.0-50-generic

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

**onworks@onworks-Standard-PC-i440FX-PIIX-1996:~$ cat > abc**

orage

kiwi

grapes

mangoes

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

abc Documents examples.desktop Pictures Templates

Desktop Downloads Music Public Videos

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

grapes

kiwi

mangoes

orage

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

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

abc Documents examples.desktop Music Public Videos

Desktop Downloads lmn.txt Pictures Templates

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

grapes

kiwi

mangoes

orage

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

orage

kiwi

grapes

mangoes

**Experiment No. 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**

1. **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**

1. **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**

1. **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**

1. **rmdir Command**

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

**$ rmdir tutorials**

1. **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**

1. **cat Command**

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

$ cat filename

1. **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/**

1. **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**

1. **Deleting a File**

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

**$rm filename**

Also try the following commands

**Directory and file commands**

|  |  |
| --- | --- |
| cd /home | enter to directory '/ home'   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=cd)] |
| # cd .. | go back one level   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=cd)] |
| # cd ../.. | go back two levels   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=cd)] |
| # cd | go to home directory   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=cd)] |
| # cd ~user1 | go to home directory   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=cd)] |
| # cd - | go to previous directory   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=cd)] |
| # cp file1 file2 | copying a file   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=cp)] |
| # cp dir/\* . | copy all files of a directory within the current work directory   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=cp)] |
| # cp -a /tmp/dir1 . | copy a directory within the current work directory   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=cp)] |
| # cp -a dir1 dir2 | copy a directory   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=cp)] |
| # cp file file1 | outputs the mime type of the file as text   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=file)] |
| # iconv –l | lists known encodings   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=iconv)] |
| # iconv -f fromEncoding -t toEncoding inputFile > outputFile | converting the coding of characters from one format to another   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=iconv)] |
| # find . -maxdepth 1 -name \*.jpg -print -exec convert | batch resize files in the current directory and send them to a thumbnails directory (requires convert from Imagemagick)   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=iconv)] |
| # ln -s file1 lnk1 | create a symbolic link to file or directory   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=ln)] |
| # ln file1 lnk1 | create a physical link to file or directory   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=ln)] |
| # ls | view files of directory   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=ls)] |
| # ls –F | view files of directory   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=ls)] |
| # ls –l | show details of files and directory   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=ls)] |
| # ls –a | show hidden files   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=ls)] |
| # ls \*[0-9]\* | show files and directory containing numbers   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=ls)] |
| # lstree | show files and directories in a tree starting from root(2)   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=lstree)] |
| # mkdir dir1 | create a directory called 'dir1'   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=mkdir)] |
| # mkdir dir1 dir2 | create two directories simultaneously   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=mkdir)] |
| # mkdir -p /tmp/dir1/dir2 | create a directory tree   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=mkdir)] |
| # mv dir1 new\_dir | rename / move a file or directory   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=mv)] |
| # pwd | show the path of work directory   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=pwd)] |
| # rm -f file1 | delete file called 'file1'   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=rm)] |
| # rm -rf dir1 | remove a directory called 'dir1' and contents recursively   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=rm)] |
| # rm -rf dir1 dir2 | remove two directories and their contents recursively   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=rm)] |
| # rmdir dir1 | delete directory called 'dir1'   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=rmdir)] |
| # touch -t 0712250000 file1 | modify timestamp of a file or directory - (YYMMDDhhmm)   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=touch)] |
| # tree | show files and direct |

**Experiment No. 3**

**Aim-** Building multi-threaded and multi-process applications

**Problem Statement-**

1. Build an instance of bus ticket reservation system using multithreading for the following scenario.

ABC Bus service has only two seats left for reservations. Two users are trying to book the ticket at the same time.

1. Create separate thread per user. Show how this code is leading to inconsistency
2. Improve the code by applying synchronization.
3. Conclude the experiment by stating the importance of synchronization in multiprocess and multithread application

**CODE-**

**Using Multithreading( NO Synchronization)**

//This represent a Passenger. The reasons for extending the Thread class is that this should carry the data of number of seats.

**public class PassengerThread extends Thread**

{

private int seatsNeeded;

public PassengerThread(int seats, Runnable target, String name)

{

super(target, name);

this.seatsNeeded = seats;

}

public int getSeatsNeeded()

{

return seatsNeeded;

}

}

**public class BusReservation1 implements Runnable{**

private int totalSeatsAvailable=2;

public void run()

{

//BusReservation br = new BusReservation();

PassengerThread pt = (PassengerThread) Thread.currentThread();

boolean ticketsBooked = this.bookTickets(pt.getSeatsNeeded(), pt.getName());

if(ticketsBooked==true)

{

System.out.println("CONGRATS Mr./Ms. "+Thread.currentThread().getName()+" The number of seats requested "+pt.getSeatsNeeded()+" are BOOKED");

}

else

{

System.out.println("Sorry Mr/Ms. "+Thread.currentThread().getName()+" The number of seats requested " +pt.getSeatsNeeded()+" are not available");

}

}

public boolean bookTickets(int seats, String name)

{

System.out.println("Welcome to ABC Bus Service "+Thread.currentThread().getName()+"\nNumber of Tickets Available are:" +this.getTotalSeatsAvailable());

if (seats>this.getTotalSeatsAvailable())

{

return false;

}

else

{

totalSeatsAvailable = totalSeatsAvailable-seats;

return true;

}

}

private int getTotalSeatsAvailable()

{

return totalSeatsAvailable;

}

}

**public class ThreadSafetyDemo {**

public static void main(String[] args) {

BusReservation1 br = new BusReservation();

PassengerThread pt1 = new PassengerThread(2, br, "JAY");

PassengerThread pt2 = new PassengerThread(2, br, "KUSHAL");

pt1.start();

pt2.start();

}

}

**/\* OUTPUT OF MULTITHREADING \*/**

**D:\>java ThreadSafetyDemo**

Welcome to ABC Bus Service KUSHAL

Number of Tickets Available are:2

Welcome to ABC Bus Service JAY

Number of Tickets Available are:2

CONGRATS Mr./Ms. KUSHAL The number of seats requested 2 are BOOKED

Sorry Mr/Ms. JAY The number of seats requested 2 are not available

**Using Multithreading ( With Synchronization)**

//This represent a Passenger. The reasons for extending the Thread class is that this should carry the data of number of seats.

**public class PassengerThread extends Thread**

{

private int seatsNeeded;

public PassengerThread(int seats, Runnable target, String name)

{

super(target, name);

this.seatsNeeded = seats;

}

public int getSeatsNeeded()

{

return seatsNeeded;

}

}

**public class BusReservation implements Runnable{**

private int totalSeatsAvailable=2;

public void run()

{

//BusReservation br = new BusReservation();

PassengerThread pt = (PassengerThread) Thread.currentThread();

boolean ticketsBooked = this.bookTickets(pt.getSeatsNeeded(), pt.getName());

if(ticketsBooked==true)

{

System.out.println("CONGRATS Mr./Ms. "+Thread.currentThread().getName()+" The number of seats requested "+pt.getSeatsNeeded()+" are BOOKED");

}

else

{

System.out.println("Sorry Mr/Ms. "+Thread.currentThread().getName()+" The number of seats requested " +pt.getSeatsNeeded()+" are not available");

}

}

public **synchronized** boolean bookTickets(int seats, String name)

{

System.out.println("Welcome to ABC Bus Service "+Thread.currentThread().getName()+"\nNumber of Tickets Available are:" +this.getTotalSeatsAvailable());

if (seats>this.getTotalSeatsAvailable())

{

return false;

}

else

{

totalSeatsAvailable = totalSeatsAvailable-seats;

return true;

}

}

private int getTotalSeatsAvailable()

{

return totalSeatsAvailable;

}

}

**public class ThreadSafetyDemo {**

public static void main(String[] args) {

BusReservation br = new BusReservation();

PassengerThread pt1 = new PassengerThread(2, br, "JAY");

PassengerThread pt2 = new PassengerThread(2, br, "KUSHAL");

pt1.start();

pt2.start();

}

}

**/\* OUTPUT Using Multithreading ( With Synchronization) \*/**

**D:\>javac PassengerThread.java**

**D:\>javac BusReservation.java**

**D:\>javac ThreadSafetyDemo.java**

**D:\>java ThreadSafetyDemo**

Welcome to ABC Bus Service JAY

Number of Tickets Available are:2

CONGRATS Mr./Ms. JAY The number of seats requested 2 are BOOKED

Welcome to ABC Bus Service KUSHAL

Number of Tickets Available are:0

Sorry Mr/Ms. KUSHAL The number of seats requested 2 are not available

**Experiment No. 4**

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

**Problem Statement-**

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

**Description:**

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

Terms used

1)Submit time :The process at which the process is given to CPU

2)Burst time : The amount of time each process takes for execution

3)Response time :The difference between the time when the process

starts execution and the submit time.

4)Turnaround time :The difference between the time when the

process completes execution and the submit time.

First Come First Serve(FCFS)

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

Shortest Job First(SJF)

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

Shortest Remaining Time Next(SRTN)

This is preemptive version of SJF. In this scheduling algorithm, the [process](http://en.wikipedia.org/wiki/Process_(computing)) with the smallest amount of time remaining until completion is selected to execute.

Round Robin

Each process is assigned a time interval called its quantum(time slice)

If the process is still running at the end of the quantum the CPU is preempted and given to another process, and this continues in circular fashion, till all the processes are completely executed.

Priority Scheduling

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

**Experiment No. 5**

**Aim- Process and Thread Synchronization using client server mechanism**

**Problem Statement-**

1. Use socket programming to construct client and server program.
2. Use multithreading which will enable server to communicate with multiple clients simultaneously in synchronous manner.
3. Run multiple instances of client program to communicate with server

**Code-**

**// Server Code**

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.net.ServerSocket;

import java.net.Socket;

public class Server\_X\_Client {

public static void main(String args[]){

Socket s=null;

ServerSocket ss2=null;

System.out.println("Server Listening......");

try{

ss2 = new ServerSocket(4445); // can also use static final PORT\_NUM , when defined

}

catch(IOException e){

e.printStackTrace();

System.out.println("Server error");

}

while(true){

try{

s= ss2.accept();

System.out.println("connection Established");

ServerThread st=new ServerThread(s);

st.start();

}

catch(Exception e){

e.printStackTrace();

System.out.println("Connection Error");

}

}

}

}

class ServerThread extends Thread{

String line=null;

BufferedReader is = null;

PrintWriter os=null;

Socket s=null;

public ServerThread(Socket s){

this.s=s;

}

public void run() {

try{

is= new BufferedReader(new InputStreamReader(s.getInputStream()));

os=new PrintWriter(s.getOutputStream());

}catch(IOException e){

System.out.println("IO error in server thread");

}

try {

line=is.readLine();

while(line.compareTo("QUIT")!=0){

os.println(line);

os.flush();

System.out.println("Response to Client : "+line);

line=is.readLine();

}

} catch (IOException e) {

line=this.getName(); //reused String line for getting thread name

System.out.println("IO Error/ Client "+line+" terminated abruptly");

}

catch(NullPointerException e){

line=this.getName(); //reused String line for getting thread name

System.out.println("Client "+line+" Closed");

}

finally{

try{

System.out.println("Connection Closing..");

if (is!=null){

is.close();

System.out.println(" Socket Input Stream Closed");

}

if(os!=null){

os.close();

System.out.println("Socket Out Closed");

}

if (s!=null){

s.close();

System.out.println("Socket Closed");

}

}

catch(IOException ie){

System.out.println("Socket Close Error");

}

}//end finally

}

}

**// Client for Echo Server**

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.net.InetAddress;

import java.net.Socket;

public class NetworkClient {

public static void main(String args[]) throws IOException{

InetAddress address=InetAddress.getLocalHost();

Socket s1=null;

String line=null;

BufferedReader br=null;

BufferedReader is=null;

PrintWriter os=null;

try {

s1=new Socket(address, 4445); // You can use static final constant PORT\_NUM

br= new BufferedReader(new InputStreamReader(System.in));

is=new BufferedReader(new InputStreamReader(s1.getInputStream()));

os= new PrintWriter(s1.getOutputStream());

}

catch (IOException e){

e.printStackTrace();

System.err.print("IO Exception");

}

System.out.println("Client Address : "+address);

System.out.println("Enter Data to echo Server ( Enter QUIT to end):");

String response=null;

try{

line=br.readLine();

while(line.compareTo("QUIT")!=0){

os.println(line);

os.flush();

response=is.readLine();

System.out.println("Server Response : "+response);

line=br.readLine();

}

}

catch(IOException e){

e.printStackTrace();

System.out.println("Socket read Error");

}

finally{

is.close();os.close();br.close();s1.close();

System.out.println("Connection Closed");

}

}

}

**Experiment No. 6**

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

follows:

• The counter remains open till the waiting queue is not empty

• If the queue is already full, the new customer simply leaves

• If the queue becomes empty, the outlet doors will be closed (service personnel sleep)

• Whenever a customer arrives at the closed outlet, he/she needs to wake the person at the counter with a wake-up call

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

**Problem Statement-**

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

**Code-**

import java.util.Scanner;

import java.util.concurrent.Semaphore;

import java.util.concurrent.locks.ReentrantLock;

class prodcons{

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MAIN\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public static int MAX=5; // buffersize

public static final int cnum=9; //total consumers for servicing

public static final String ANSI\_RESET = "\u001B[0m";

public static final String ANSI\_BLACK = "\u001B[30m";

public static final String ANSI\_RED = "\u001B[31m";

public static final String ANSI\_GREEN = "\u001B[32m";

public static final String ANSI\_YELLOW = "\u001B[33m";

public static final String ANSI\_BLUE = "\u001B[34m";

public static final String ANSI\_PURPLE = "\u001B[35m";

public static final String ANSI\_CYAN = "\u001B[36m";

public static final String ANSI\_WHITE = "\u001B[37m";

public static int service\_time[]=new int[cnum]; //initializing array for service time

public static void printQ(int[] Q,int front,int rear){

int f=front;int r=rear;

while(f!=r){

System.out.print(Q[f]+" ");

f+=1;

f=f%MAX;

}

}

public static void main(String args[]) throws InterruptedException

{

Scanner sc=new Scanner(System.in);

int choice = 0;int z=0;

while(choice!=1 && choice!=2) // Total service personnel required

{

if(z==0)

{

System.out.println("Enter the number of Service personnel required(1/2): ");

choice=sc.nextInt();

z=1;

}

else

{

System.out.println(ANSI\_RED+"Please enter correct value(either 1 or 2)!!"+ANSI\_RESET);

System.out.println("Enter the number of Service personnel required(1/2): ");

choice=sc.nextInt();

}

}

sc.close();

for(int i=0;i<cnum;i++) // Generating random servicing time required by each customer

service\_time[i]=getRandomNumberInRange(1, 5);

int max=MAX-1;

System.out.println(ANSI\_BLUE+"Waiting Queue length: "+ANSI\_RESET+max);

System.out.println(ANSI\_RED+"Total customers: "+cnum+ANSI\_RESET);

//System.out.println(ANSI\_BLUE+"Customer No.\t"+"Service time\t"+"Service Personnel\t"+"Curr queue\t"+"Remarks\t"+ANSI\_RESET);

System.out.println(ANSI\_YELLOW+"-----------------Waiting Queue is Empty, Now the Counter is CLOSED---------------"+ANSI\_RESET);

switch(choice)

{

case 1: System.out.println("SERVICING WITH SINGLE PERSONNEL");

System.out.println(ANSI\_BLUE+"Customer No.\t"+"Service time\t"+"Service Personnel\t"+"Curr queue\t"+"Remarks"+ANSI\_RESET);

final prodcons pc1 = new prodcons();

Thread p1 = new Thread(new Runnable() // Creating producer thread for 1 service personnel case

{

@Override

public void run()

{

try

{

pc1.producer1();

}

catch(InterruptedException e)

{

e.printStackTrace();

}

}

});

Thread c1 = new Thread(new Runnable() // Creating consumer thread for 1 service personnel case

{

@Override

public void run()

{

try

{

pc1.consumer1();

}

catch(InterruptedException e)

{

e.printStackTrace();

}

}

});

//System.out.println(ANSI\_GREEN+"Customer 1 arrives and wakes up the Service Personnel"+ANSI\_RESET);

System.out.println(ANSI\_GREEN+"\t\t"+"\t\t"+"\t\t\t"+"--"+"\t\t"+"Customer 1 arrives and wakes up the Service Personnel"+ANSI\_RESET);

p1.start();

c1.start();

p1.join();

c1.join();

break;

case 2: System.out.println("SERVICING WITH 2 SERVICE PERSONNEL");

System.out.println(ANSI\_BLUE+"Customer No.\t"+"Service time\t"+"Service Personnel\t"+"Curr queue\t"+"Remarks"+ANSI\_RESET);

final prodcons pc2 = new prodcons();

Thread p2 = new Thread(new Runnable() // Creating producer thread for 2 service personnel case

{

@Override

public void run()

{

try

{

pc2.producer2();

}

catch(InterruptedException e)

{

e.printStackTrace();

}

}

});

Thread c21 = new Thread(new Runnable() // Creating consumer thread for 2 service personnel case

{

@Override

public void run()

{

try

{

pc2.consumer21();

}

catch(InterruptedException e)

{

e.printStackTrace();

}

}

});

Thread c22 = new Thread(new Runnable() // Creating consumer thread for 2 service personnel case

{

@Override

public void run()

{

try

{

pc2.consumer22();

}

catch(InterruptedException e)

{

e.printStackTrace();

}

}

});

//System.out.println(ANSI\_GREEN+"Customer 1 arrives and wakes up the Service Personnel"+ANSI\_RESET);

System.out.println(ANSI\_GREEN+"\t\t"+"\t\t"+"\t\t\t"+"--"+"\t\t"+"Customer 1 arrives and wakes up the Service Personnel"+ANSI\_RESET);

p2.start();

c21.start();

c22.start();

p2.join();

c21.join();

c22.join();

break;

default: System.out.println(ANSI\_RED+"Please restart the program as wrong input is given."+ANSI\_RESET);

}// Switch ends

System.out.println(ANSI\_YELLOW+"--------------------------------COUNTER IS CLOSED--------------------------------"+ANSI\_RESET);

}

// Function for Random number generation

private static int getRandomNumberInRange(int min, int max) {

if (min >= max) {

throw new IllegalArgumentException("max must be greater than min");

}

return (int)(Math.random() \* ((max - min) + 1)) + min;

}

/////////////////////////////////////////////////////////

public static final int FULL = 0;

public static final int EMPTY=MAX;

public static int last=0,flag=0,slept=0,front=0,rear=0;

static Semaphore semFull = new Semaphore(FULL); // Semaphore for checking how many slots are FULL

static Semaphore semEmp = new Semaphore(EMPTY); // Semaphore for checking how many slots are EMPTY

ReentrantLock lock = new ReentrantLock(); // MUTEX LOCK

int queue[]=new int[MAX];

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*Consumer Method for 1 service personnel case\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

public void consumer1() throws InterruptedException

{

int item=0;

do

{

semFull.acquire();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MUTEX region\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

synchronized (this)

{

if(front==rear)

{

//System.out.println(ANSI\_RED+"Waiting queue is empty, Service personnel sleeps"+ANSI\_RESET);

System.out.println(ANSI\_RED+"\t\t"+"\t\t"+"\t\t\t"+"--"+"\t\t"+"Waiting queue is empty, Service personnel sleeps"+ANSI\_RESET);

slept=1;

}

else

{

item=queue[front];

//System.out.println(ANSI\_GREEN+"Customer "+item+" is being serviced having service time "+service\_time[item-1]+"sec"+ANSI\_RESET);

System.out.print(ANSI\_GREEN+item+"\t\t"+service\_time[item-1]+" sec\t\t"+"\t1\t\t");

printQ(queue,front,rear);

System.out.println("\t\tBeing serviced"+ANSI\_RESET);

front=(front+1)%MAX;

if(last==1 && item==cnum-1)

flag=1;

Thread.sleep(service\_time[item-1]\*1000);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MUTEX region ENDS here\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

semEmp.release();

}while(item!=cnum && flag!=1);

}

//\*\*\*\*\*\*\*1st Consumer Method for 2 Service personnel case\*\*\*\*\*\*\*/

public void consumer21() throws InterruptedException

{

int item=0;

do

{

semFull.acquire();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MUTEX region\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

lock.lock();

if(front==rear)

{

//System.out.println(ANSI\_RED+"Waiting Queue is empty and Service Personnel sleeps"+ANSI\_RESET);

System.out.println(ANSI\_RED+"\t\t"+"\t\t"+"\t\t\t"+"--"+"\t\t"+"Waiting queue is empty, Service personnel sleeps"+ANSI\_RESET);

slept=1;

lock.unlock();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MUTEX region ENDS here\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

}

else

{

item=queue[front];

//System.out.println(ANSI\_GREEN+"Customer "+item+" is being serviced by service personnel 1 for time "+service\_time[item-1]+"sec"+ANSI\_RESET);

System.out.print(ANSI\_GREEN+item+"\t\t"+service\_time[item-1]+" sec\t\t"+"\t1\t\t");

printQ(queue,front,rear);

System.out.println("\t\tBeing serviced"+ANSI\_RESET);

front=(front+1)%MAX;

lock.unlock();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ORRR MUTEX region ENDS here\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if(item==cnum)

flag=1;

Thread.sleep(service\_time[item-1]\*1000);///////service\_time[item]

}

semEmp.release();

}while(item!=cnum && flag!=1);

flag=1;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*2nd Consumer Method for 2 Service personnel case\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

public void consumer22() throws InterruptedException

{

int item=0;

do

{

semFull.acquire();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Mutex region\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

lock.lock();

if(front==rear)

{

//System.out.println(ANSI\_RED+"Waiting queue is empty, Service personnel sleeps"+ANSI\_RESET);

System.out.println(ANSI\_RED+"\t\t"+"\t\t"+"\t\t\t"+"--"+"\t\t"+"Waiting queue is empty, Service personnel sleeps"+ANSI\_RESET);

lock.unlock();

slept=1;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MUTEX region ENDS here\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

}

else

{

item=queue[front];

//System.out.println(ANSI\_CYAN+"Customer "+item+" is being serviced by service personnel 2 for time "+service\_time[item-1]+"sec"+ANSI\_RESET);

System.out.print(ANSI\_CYAN+item+"\t\t"+service\_time[item-1]+" sec\t\t"+"\t2\t\t");

printQ(queue,front,rear);

System.out.println("\t\tBeing serviced"+ANSI\_RESET);

front=(front+1)%MAX;

lock.unlock();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ORRR MUTEX region ENDS here\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if(item==cnum)

flag=1;

Thread.sleep(service\_time[item-1]\*1000);

}

semEmp.release();

}while(item!=cnum && flag!=1);

flag=1;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Producer Class for 1 Service personnel case\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

public void producer1() throws InterruptedException

{

for(int i=1;i<=cnum;i++)

{

semEmp.acquire();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Mutex Region\*\*\*\*\*\*\*\*\*\*\*/

synchronized (this)

{

// consumer thread waits while list

// is empty

if ((front==0 && rear==MAX-1)||(rear==(front-1)%MAX))

{

//System.out.println(ANSI\_RED+"Queue is full, Customer "+i+" leaves"+ANSI\_RESET);

System.out.print(ANSI\_RED+"\t\t"+"\t\t"+"\t\t\t");

printQ(queue,front,rear);

System.out.println("\t\tQueue is full, Customer "+i+" leaves"+ANSI\_RESET);

}

else

{

queue[rear]=i;

rear=(rear+1)%MAX;

if(slept==1)

{

//System.out.println(ANSI\_PURPLE+"Customer "+ i +" arrives and wakes up service personnel"+ANSI\_RESET);

System.out.println(ANSI\_PURPLE+"\t\t"+"\t\t"+"\t\t\t"+"--"+"\t\t"+"Customer "+ i +" arrives and wakes up service personnel"+ANSI\_RESET);

slept=0;

}

else

{

//System.out.println(ANSI\_BLUE+"Customer "+i +" is added to the queue"+ANSI\_RESET);

System.out.print(ANSI\_BLUE+"\t\t"+"\t\t"+"\t\t\t");

printQ(queue,front,rear);

System.out.println("\t\tCustomer "+ i +" is added to the queue"+ANSI\_RESET);

}

}

}//mutex ends

semFull.release();

}

semFull.release();

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Producer Class for 2 Service personnel case\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

public void producer2() throws InterruptedException

{

for(int i=1;i<=cnum;i++)

{

semEmp.acquire();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MUTEX region\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

lock.lock();

// consumer thread waits while queue

// is empty

if ((front==0 && rear==MAX-1)||(rear==(front-1)%MAX))

{

//System.out.println(ANSI\_RED+"Queue is full, Customer "+i+" leaves"+ANSI\_RESET);

System.out.print(ANSI\_RED+"\t\t"+"\t\t"+"\t\t\t");

printQ(queue,front,rear);

System.out.println("\t\tQueue is full, Customer "+i+" leaves"+ANSI\_RESET);

}

else

{

queue[rear]=i;

rear=(rear+1)%MAX;

if(slept==1)

{

//System.out.println(ANSI\_PURPLE+"Customer "+ i +" arrives and wakes up service personnel"+ANSI\_RESET);

System.out.println(ANSI\_PURPLE+"\t\t"+"\t\t"+"\t\t\t"+"--"+"\t\t"+"Customer "+ i +" arrives and wakes up service personnel"+ANSI\_RESET);

slept=0;

}

else

{

//System.out.println(ANSI\_BLUE+"Customer "+i +" is added to the queue"+ANSI\_RESET);

System.out.print(ANSI\_BLUE+"\t\t"+"\t\t"+"\t\t\t");

printQ(queue,front,rear);

System.out.println("\t\tCustomer "+ i +" is added to the queue"+ANSI\_RESET);

}

}

lock.unlock();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MUTEX region ENDS here\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

semFull.release();

}

semFull.release();

}

}

**/\* OUTPUT\*/**

**D:\>java prodcons**

**Enter the number of Service personnel required(1/2): 1**

**Waiting Queue length: 4**

**Total customers: 9**

**Waiting Queue is Empty, Now the Counter is CLOSED---------**

**SERVICING WITH SINGLE PERSONNEL**

**Customer No. Service time Service Personnel Curr queue Remarks**

**Customer 1 arrives and wakes up the Service Personnel**

**1 Customer 1 is added to the queue**

**12 Customer 2 is added to the queue**

**1 1 sec 1 1 2 Being serviced**

**2 3 Customer 3 is added to the queue**

**2 1 sec 1 2 3 Being serviced**

**3 4 Customer 4 is added to the queue**

**3 2 sec 1 3 4 Being serviced**

**4 5 Customer 5 is added to the queue**

**4 5 6 Customer 6 is added to the queue**

**4 4 sec 1 4 5 6 Being serviced**

**5 6 7 Customer 7 is added to the queue**

**5 6 7 8 Customer 8 is added to the queue**

**5 6 7 8 Queue is full, Customer 9 leaves**

**5 5 sec 1 5 6 7 8 Being serviced**

**6 5 sec 1 6 7 8 Being serviced**

**7 1 sec 1 7 8 Being serviced**

**8 3 sec 1 8 Being serviced**

**Waiting queue is empty, Service personnel sleeps**

**--------------------------------COUNTER IS CLOSED--------------------------**

**D:\>java prodcons**

**Enter the number of Service personnel required(1/2):**

**2**

**Waiting Queue length: 4**

**Total customers: 9**

**-------Waiting Queue is Empty, Now the Counter is CLOSED---------**

**SERVICING WITH 2 SERVICE PERSONNEL**

**Customer No. Service time Service Personnel Curr queue Remarks**

**Customer 1 arrives and wakes up the Service Personnel**

**1 Customer 1 is added to the queue**

**1 2 Customer 2 is added to the queue**

**1 2 3 Customer 3 is added to the queue**

**1 2 3 4 Customer 4 is added to the queue**

**1 2 3 4 Queue is full, Customer 5 leaves**

**1 1 sec 2 1 2 3 4 Being serviced**

**2 3 sec 1 2 3 4 Being serviced**

**3 1 sec 2 3 4 Being serviced**

**4 6 Customer 6 is added to the queue**

**4 5 sec 2 4 6 Being serviced**

**6 7 Customer 7 is added to the queue**

**6 5 sec 1 6 7 Being serviced**

**7 8 Customer 8 is added to the queue**

**7 5 sec 2 7 8 Being serviced**

**8 9 Customer 9 is added to the queue**

**8 4 sec 1 8 9 Being serviced**

**9 5 sec 2 9 Being serviced**

**Waiting queue is empty, Service personnel sleeps**

**--------------------------------COUNTER IS CLOSED--------------------------**

**Experiment No. 7**

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

**Problem Statement-**

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

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

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

**Experiment No. 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
3. ***Process Scheduling algorithms***

**Loading and Compiling Program**

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

In the compiler window, enter the following source code in the compiler’s source editor window (under **PROGRAM SOURCE** frame title):

program LoopTest

i = 0

for n = 0 to 40

i = i + 1

next

end

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

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

Now, this code needs to be loaded in memory so that the CPU can execute it. To do this, first we need to specify a base address (in **ASSEMBLY CODE** view): uncheck the box next to the edit box with label **Base Address**, and then enter 100 in the edit box. Now, click on the **LOAD IN MEMORY…** button in the current window. You should now see the code loaded in memory ready to be executed. You are also back in the CPU simulator at this stage. This action is equivalent to loading the program code normally stored on a disc drive into RAM on the real computer systems.

**Creating processes from programs in the OS simulator.**

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

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

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

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

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

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

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

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

***Thread creation and synchronization.***

**Loading and Compiling a Program**

In the compiler window enter the following source code:

program ThreadTest1

sub thread1 as thread

writeln("In thread1")

while true

wend

end sub

sub thread2 as thread

call thread1

writeln("In thread2")

while true

wend

end sub

call thread2

writeln("In main")

do

loop

end

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

Make the console window visible by clicking on the **INPUT/OUTPUT…** button. Also make sure the console window stays on top by checking the **Stay on top** check box.

Now, go to the OS simulator window (use the **OS…** button in the CPU simulator window) and create a single process of program *ThreadTest1* in the program list view. For this use the **CREATE NEW PROCESS** button.

Make sure the scheduling policy selected is **Round Robin** and that the simulation speed is set at maximum.

Hit the **START** button and at the same time observe the displays on the console window.

Briefly explain your observations and the no. of processes created in the box below.

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

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

Briefly explain your observations in the box below:

How are the parent/child process relationships represented?

Identify the parent and the children processes:

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.

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

program CriticalRegion2

var g integer

sub thread1 as thread **synchronise**

writeln("In thread1")

g = 0

for n = 1 to 20

g = g + 1

next

writeln("thread1 g = ", g)

writeln("Exiting thread1")

end sub

sub thread2 as thread **synchronise**

writeln("In thread2")

g = 0

for n = 1 to 12

g = g + 1

next

writeln("thread2 g = ", g)

writeln("Exiting thread2")

end sub

writeln("In main")

call thread1

call thread2

wait

writeln("Exiting main")

end

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

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

program 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**.

**Deadlock prevention and avoidance**

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

Use the Scenario P1 holding R0 and waiting for R1. P2 Holding R1 and waiting for R2. P3 holding R2 and waiting for R3. P4 holding R3 and waiting for R0.

Draw the resource allocation graph for a four process deadlock condition.

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

program DeadlockP**N**

resource(**X**, allocate)

wait(3)

resource(**Y**, allocate)

for n = 1 to 20

next

end

2)

a. Copy the above code and paste it in three more edit windows so that you have a total of four pieces of source code. ( Click on New under compiler to create new edit windows)

b. In each case change **N** in the program name to 1 to 4, e.g. DeadlockP**1,** DeadlockP**2,** etc.

c. Look at your graph you constructed in (1) above and using that information fill in the values for each of the **X**s and **Y**s in the four pieces of source code. ( X is resource the process is holding and Y is the resource process is waiting for. Eg. For P1, X=0 and Y=1)

d. Compile each one of the four source code.

e. Load in memory the four pieces of code generated.

f. Now switch to the OS simulator.

g. Create a single instance of each of the programs. You can do this by double-clicking on each of the program names in the **PROGRAM LIST** frame under the **Program Name** column.

h. In the **SCHEDULER** frame select **Round Robin (RR)** scheduling policy in the **Policies** tab.

i. In OS Control tab, push the speed slider up to the fastest speed.

j. Select the **Views** tab and click on the **VIEW RESOURCES…** button.

k. Select **Stay on top** check box in the displayed window.

l. Back in the **OS Control** tab use the **START** button to start the OS scheduler and observe the changing process states for few seconds.

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

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

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

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

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

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

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

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

f. Once the deadlock condition is obtained again do the following: In the OS Simulator window, select a process in the waiting queue in the **WAITING PROCESSES** frame.

g. Click on the REMOVE button and observe the processes.

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

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

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

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

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

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

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

We are now going to try a third method of preventing deadlocking before it happens. It is called “total ordering” method. Here the resources are allocated in increasing resource id numbers only. So, for example, resource R3 must be allocated after resources R0 to R2 and resource R1 cannot be allocated after resource R2 is allocated. Looking at your resource allocation graph can you see how this ordering can prevent a deadlock? Comment.

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

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

**Experiment No. 9**

**Aim- Implement various page replacement policies**

**Problem Statement-**

Perform following page replacement policies and perform comparative assessment them.

1) OPTIMAL

2) LEAST RECENTLY USED (LRU)

3) FIRST-IN-FIRST-OUT

**Description:**

In multiprogramming system using dynamic partitioning there will come a time when all of the processes in the main memory are in a blocked state and there is insufficient memory. To avoid wasting processor time waiting for an active process to become unblocked. The OS will swap one of the process out of the main memory to make room for a new process or for a process in Ready-Suspend state.

Therefore, the OS must choose which process to replace.

Thus, when a page fault occurs, the OS has to change a page to remove from memory to make room for the page that must be brought in. If the page to be removed has been modified while in memory it must be written to disk to bring the disk copy up to date.

Replacement algorithms can affect the system's performance. Following are the three basic page replacement algorithms:

Optimal Page Replacement Policy

The idea is to replace the page that will not be referenced for the longest period of time.

Least Recently Used Algorithm

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

First-In-First\_Out

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

**Experiment No. 10**

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

**Problem Statement-**

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

1) FCFS

2) SSTF

3) SCAN

4) CSCAN

5) LOOK

**Description:**

1) FCFS

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

2) SSTF

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

3) SCAN

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

4) CSCAN

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

5) LOOK

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