OPERATING SYSTEM MANUAL



OPERATING SYSTEM

Lab Manual

[Fall/ Spring 20\_\_]

|  |
| --- |
| Student Name: BINYAMIN |
| Student Id: 13402 |
|  |

|  |
| --- |
| Prepared By: *Dr. Noman Islam* |
| Instructor: *Dr. Noman Islam* |

**LIST OF EXPERIMENTS**

|  |  |  |
| --- | --- | --- |
| **S. No** | **Date** | **Experiment** |
| **1** | \_\_/\_\_/\_\_ | To study and implement socket programming in Java |
| **2** | \_\_/\_\_/\_\_ | To study and implement file I/O in Java |
| **3** | \_\_/\_\_/\_\_ | To study and implement multi-threading in Java |
| **4** | \_\_/\_\_/\_\_ | To study and execute basic Linux commands on a terminal |
| **5** | \_\_/\_\_/\_\_ | To study and execute system administration commands on a terminal |
| **6** | \_\_/\_\_/\_\_ | To study and implement shell programming in Linux |
| **7** | \_\_/\_\_/\_\_ | To study and implement information security techniques in Linux |
| **8** | \_\_/\_\_/\_\_ | To study and implement concurrency control techniques in Java |
| **9** | \_\_/\_\_/\_\_ | To study and implement process scheduling algorithms in Java |
| **10** | \_\_/\_\_/\_\_ | To study and implement file allocation techniques in Java |
| **11** | \_\_/\_\_/\_\_ | To study and implement page replacement algorithms in Java |
| **12** | \_\_/\_\_/\_\_ | To study and implement disk scheduling algorithm in Java |

Find the IP address of a local host using java program using inetAddress class

package task1;

import java.net.InetAddress;

import java.net.UnknownHostException;

/\*\*

\*

\* @author Binyamin

\*/

public class Task1 {

/\*\*

\* @param args the command line arguments

\*/

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

// TODO code application logic here

InetAddress myIP=InetAddress.getLocalHost();

/\* public String getHostAddress(): Returns the IP

\* address string in textual presentation.

\*/

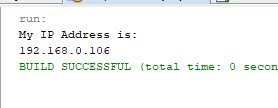
System.out.println("My IP Address is:");

System.out.println(myIP.getHostAddress());

}

}

**OUTPUT**



**Question 2 ) Write a small port scanner application.The program usage as follows**

E:\ >java PortScanner 132 137

Port not in use : 132

Port not in use : 133

Port not in use : 134

Port in use : 135

Port not in use : 136

Port not in us : e 137

package task2;

import java.net.\*;

import java.io.IOException;

import java.net.InetSocketAddress;

import java.net.Socket;

/\*\*

\*

\* @author Binyamin

\*/

public class Task2 {

/\*\*

\* @param args the command line arguments

\*/

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

// TODO code application logic here

for(int i=1;i<=65000;i++){

try{

Socket s=new Socket("127.0.0.1",i);

System.out.println("port in use "+i);

s.close();}

catch(IOException s){

}

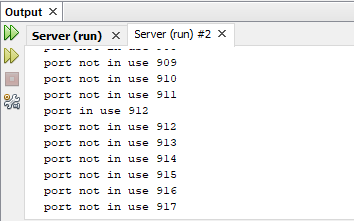
System.out.println("port not in use "+i);

}

}

}

**OUTPUT**



Q3 write a small server that accepts socket connection on port 2020.develop a client application that connects to server

1. Using buffered output stream write to server the “hello”
2. The server should respond with hello

Server

import java.io.IOException;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.lang.ClassNotFoundException;

import java.net.ServerSocket;

import java.net.Socket;

/\*\*

\*

\* @author Binyamin

\*/

public class SocketServer {

/\*\*

\* @param args the command line arguments

\*/

private static ServerSocket server;

//socket server port on which it will listen

private static int port = 9876;

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

//create the socket server object

server = new ServerSocket(port);

//keep listens indefinitely until receives 'exit' call or program terminates

while(true){

System.out.println("Waiting for the client request");

//creating socket and waiting for client connection

Socket socket = server.accept();

//read from socket to ObjectInputStream object

ObjectInputStream ois = new ObjectInputStream(socket.getInputStream());

//convert ObjectInputStream object to String

String message = (String) ois.readObject();

System.out.println("Message Received: " + message);

//create ObjectOutputStream object

ObjectOutputStream oos = new ObjectOutputStream(socket.getOutputStream());

//write object to Socket

oos.writeObject("Hello "+message);

//close resources

ois.close();

oos.close();

socket.close();

//terminate the server if client sends exit request

if(message.equalsIgnoreCase("exit")) break;

}

System.out.println("Shutting down Socket server!!");

//close the ServerSocket object

server.close();

}

}

Socket client

package socketclient;

import java.io.IOException;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.net.InetAddress;

import java.net.Socket;

import java.net.UnknownHostException;

/\*\*

\*

\* @author Binyamin

\*/

public class SocketClient {

/\*\*

\* @param args the command line arguments

\*/

public static void main(String[] args) throws UnknownHostException, IOException, ClassNotFoundException, InterruptedException{

InetAddress host = InetAddress.getLocalHost();

Socket socket = null;

ObjectOutputStream oos = null;

ObjectInputStream ois = null;

for(int i=0; i<5;i++){

//establish socket connection to server

socket = new Socket(host.getHostName(), 9876);

//write to socket using ObjectOutputStream

oos = new ObjectOutputStream(socket.getOutputStream());

System.out.println("Sending request to Socket Server");

if(i==4)oos.writeObject("exit");

else oos.writeObject(""+i);

//read the server response message

ois = new ObjectInputStream(socket.getInputStream());

String message = (String) ois.readObject();

System.out.println("Message: " + message);

//close resources

ois.close();

oos.close();

Thread.sleep(100);

}

}

}

**OUTPUT**

**Sending request to Socket Server**

**Message: Hello 0**

**Sending request to Socket Server**

**Message: Hello 1**

**Sending request to Socket Server**

**Message: Hello 2**

**Sending request to Socket Server**

**Message: Hello 3**

**Sending request to Socket Server**

**Message: Hello exit**

Q) Develop an echo server

import java.io.\*;

import java.net.ServerSocket;

import java.net.Socket;

import java.util.Scanner;

public class Myserver {

/\*\*

\* @param args the command line arguments

\*/

public static void main(String[] args) {

// TODO code application logic here

connectToServer();

}

public static void connectToServer() {

try(ServerSocket serverSocket = new ServerSocket(9991)) {

Socket connectionSocket = serverSocket.accept();

InputStream inputToServer = connectionSocket.getInputStream();

OutputStream outputFromServer = connectionSocket.getOutputStream();

Scanner scanner = new Scanner(inputToServer, "UTF-8");

PrintWriter serverPrintOut = new PrintWriter(new OutputStreamWriter(outputFromServer, "UTF-8"), true);

serverPrintOut.println("Hello World! Enter Peace to exit.");

//Have the server take input from the client and echo it back

//This should be placed in a loop that listens for a terminator text e.g. bye

boolean done = false;

while(!done && scanner.hasNextLine()) {

String line = scanner.nextLine();

serverPrintOut.println("Echo from <Your Name Here> Server: " + line);

if(line.toLowerCase().trim().equals("peace")) {

done = true;

}

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

**OUTPUT**

Binyamin :$ nc $ 127.0.0.1 9991

Hello world ! Enter peace to exit

**TO study and implement file I/O in java**

the following program and save.

//FileTest.java

import java.io.\*;

class FileTest

{

public void fileWrite()

{

File dstFile = new File("K:\\myOutput\\outputfile.txt");

PrintWriter out = new PrintWriter

(new BufferedWriter(new FileWriter(dstFile)));

out.print("Hello ");

out.println("world");

out.close();

}

}

//FileTestMain.java

import java.io.\*;

class FileTestMain

{

public static void main(String[] args)

{

FileTest fileTest = new FileTest();

fileTest.fileWrite();

}

}

1. Try to compile the class FileTest. What goes wrong? This is because opening up a file could throw an IOException, which is a checked exception. This means you have to tell Java how to deal with it, or the program won't compile

import java.io.IOException;

import java.io.PrintWriter;

/\*\*

\*

\* @author 7500

\*/

public class File\_Test2 {

public void fileWrite() throws IOException

{

try{

File dstFile = new File("C:\\Users\\Binyamin");

PrintWriter out = new PrintWriter

(new BufferedWriter(new FileWriter(dstFile)));

out.print("Hello ");

out.println("world");

out.close();

}

catch(Exception s){

}

}

/\*\*

\* @param args the command line arguments

\*/

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

// TODO code application logic here

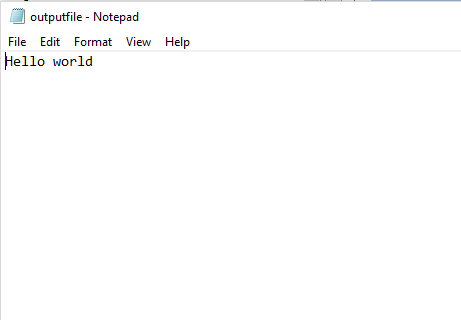
File\_Test2 fileTest = new File\_Test2();

fileTest.fileWrite();

}

}

**OUTPUT**



3.Modify your program to write to the file five lines, each of which contains your name or a friend's name, followed by a space and then an age, then another space and a gpa. For example:

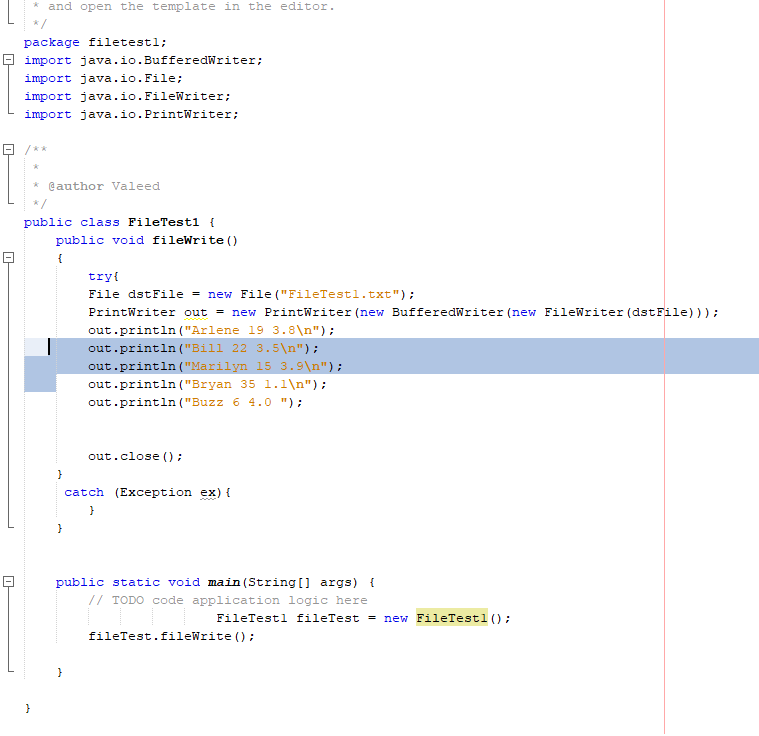
Arlene 19 3.8

Bill 22 3.5

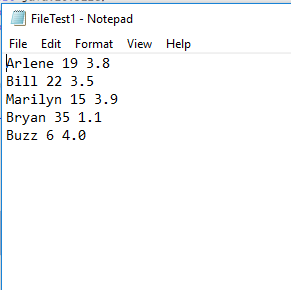
Marilyn 15 3.9

Bryan 35 1.1

Buzz 6 4.0



**OUTPUT**



4.Add the following method to your FileTest class:

public void consoleRead() throws IOException

{

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

System.out.print("What is your first name? ");

String first = in.readLine();

System.out.print("What is your last name? ");

String last = in.readLine();

System.out.println("Your name is " + last + ", " + first + ".");

}

Compile it. Add "throws" statements as necessary. Modify your main to run the consoleRead method, and recompile. Run your program. What does it do?

package filetest2;

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.File;

import java.io.FileWriter;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintWriter;

/\*\*

\*

\* @author Binyamin

\*/

public class FileTest2 {

public void fileWrite()

{

try{

File dstFile = new File("FileTest2.txt");

PrintWriter out = new PrintWriter(new BufferedWriter(new FileWriter(dstFile)));

out.println("Arlene 19 3.8\n");

out.println("Bill 22 3.5\n");

out.println("Marilyn 15 3.9\n");

out.println("Bryan 35 1.1\n");

out.println("Buzz 6 4.0 ");

out.close();

}

catch (Exception ex){

}

}

public void consoleRead() throws IOException

{

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

System.out.print("What is your first name? ");

String first = in.readLine();

System.out.print("What is your last name? ");

String last = in.readLine();

System.out.println("Your name is " + last + ", " + first + ".");

}

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

// TODO code application logic here

FileTest2 fileTest = new FileTest2();

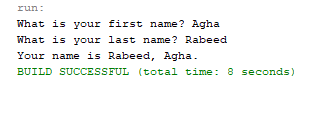
fileTest.fileWrite();

fileTest.consoleRead();

}

}

**OUTPUT**



1. Add the following method to FileTest.

public void fileRead() throws IOException

{

File srcFile = new File("K:\\FilePractice\\outputfile.txt");

BufferedReader in = new BufferedReader(new FileReader(srcFile));

String text = in.readLine();

System.out.println(text);

in.close();

}

Run the method. What do you see? Modify this method to print out the names, ages, and gpas of the five people you stored back in Task 1.

**package filetest2;**

**import java.io.BufferedReader;**

**import java.io.BufferedWriter;**

**import java.io.File;**

**import java.io.FileReader;**

**import java.io.FileWriter;**

**import java.io.IOException;**

**import java.io.InputStreamReader;**

**import java.io.PrintWriter;**

**/\*\***

**\***

**\* @author Binyamin**

**\*/**

**public class FileTest2 {**

**public void fileWrite()**

**{**

**try{**

**File dstFile = new File("FileTest2.txt");**

**PrintWriter out = new PrintWriter(new BufferedWriter(new FileWriter(dstFile)));**

**out.println("Agha 19 3.8\n");**

**out.println("Bin Yamin 22 3.5\n");**

**out.println("Adeel 15 3.9\n");**

**out.println("Shoaib 35 1.1\n");**

**out.println("Adnan 6 4.0 ");**

**out.close();**

**}**

**catch (Exception ex){**

**}**

**}**

**public void consoleRead() throws IOException**

**{**

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

**System.out.print("What is your first name? ");**

**String first = in.readLine();**

**System.out.print("What is your last name? ");**

**String last = in.readLine();**

**System.out.println("Your name is " + last + ", " + first + ".");**

**}**

**public void fileRead() throws IOException**

**{**

**File srcFile = new File("FileTest2.txt");**

**BufferedReader in = new BufferedReader(new FileReader(srcFile));**

**String text = in.readLine();**

**System.out.println(text);**

**in.close();**

**}**

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

**// TODO code application logic here**

**FileTest2 fileTest = new FileTest2();**

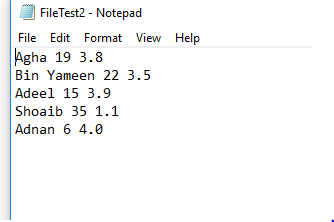
**fileTest.fileWrite();**

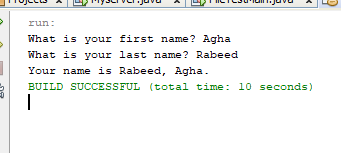
**fileTest.consoleRead();**

**}**

**}**

**OUTPUT**





6.In reality, you would want to be able to separate each item on each line into different variables, rather than keeping all the information on name, age, and gpa in one string. To break it up, use a StringTokenizer.

**package filetest3;**

**import java.io.BufferedReader;**

**import java.io.BufferedWriter;**

**import java.io.File;**

**import java.io.FileReader;**

**import java.io.FileWriter;**

**import java.io.IOException;**

**import java.io.InputStreamReader;**

**import java.io.PrintWriter;**

**import static java.lang.System.in;**

**import java.util.StringTokenizer;**

**/\*\***

**\***

**\* @author Binyamin**

**\*/**

**public class FileTest3 {**

**public void fileWrite()**

**{**

**try{**

**File dstFile = new File("FileTest2.txt");**

**PrintWriter out = new PrintWriter(new BufferedWriter(new FileWriter(dstFile)));**

**out.println("Arlene 19 3.8\n");**

**out.println("Bill 22 3.5\n");**

**out.println("Marilyn 15 3.9\n");**

**out.println("Bryan 35 1.1\n");**

**out.println("Buzz 6 4.0 ");**

**out.close();**

**}**

**catch (Exception ex){**

**}**

**}**

**public void consoleRead() throws IOException**

**{**

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

**System.out.print("What is your first name? ");**

**String first = in.readLine();**

**System.out.print("What is your last name? ");**

**String last = in.readLine();**

**System.out.println("Your name is " + last + ", " + first + ".");**

**}**

**public void fileRead() throws IOException**

**{**

**try{**

**File srcFile = new File("FileTest2.txt");**

**BufferedReader in = new BufferedReader(new FileReader(srcFile));**

**String text;**

**while((text=in.readLine()) != null){**

**StringTokenizer st= new StringTokenizer(text);**

**System.out.println("Name: "+ st.nextToken());**

**System.out.println("Age: "+ st.nextToken());**

**System.out.println("CGPA: "+ st.nextToken());**

**}}**

**catch(Exception s){**

**}**

**in.close();**

**}**

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

**// TODO code application logic here**

**FileTest3 fileTest = new FileTest3();**

**fileTest.fileWrite();**

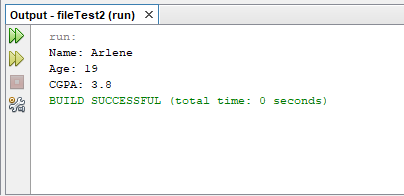
**//fileTest.consoleRead();**

**fileTest.fileRead();**

**}**

**}**

**OUTPUT**



**To study and implement multi-threading in Java**

**Instructions:**

1. A thread is an independent unit of execution.
2. In Java, the Runnable interface and Thread class of package java.lang are used for implementation of thread
3. To implement a thread, the desired class must implement the Runnable interface and provide the run() method.

public class MyThread implements Runnable {

public void run() {

//implementation of thread  
 }

}

1. The Thread class can then be used to start a thread as follows:

public class TestThread

{

public static void main( String[] args )

{

MyThread m = new MyThread();

Thread t = new Thread(m);

m.start();

}

}

**Lab Tasks:**

1. Write a class that implements Runnable. Define a constructor that takes the name of the thread as argument. The thread upon execution will print the name of the thread in a while loop. Define and run 5 thread objects. What output do you see?
2. In task 1, modify the run method to randomly sleep the thread for few milliseconds. Observe the output.
3. Create a multi-threaded client server application in Java.

Write a class that implements Runnable. Define a constructor that takes the name of the thread as argument. The thread upon execution will print the name of the thread in a while loop. Define and run 5 thread objects. What output do you see?

\*/

package multithreaded;

import java.lang.Package;

/\*\*

\*

\* @author Binyamin

\*/

public class MyThread {

String name;

public MyThread(String n){

this.name = n;

}

public void run(){

while (true){

System.out.println("running " + System.currentTimeMillis() + name);

try{

Thread.sleep(1000);

}catch(InterruptedException e){

e.printStackTrace();

}

}}

}

**Main class**

**package multithreaded;**

**import java.lang.Package;**

**public class main {**

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

**// TODO code application logic here**

**for(int i= 0; i<5; i++){**

**MyThread m = new MyThread("Thread " +i);**

**Thread tl = new Thread(m);**

**tl.start();**

**}**

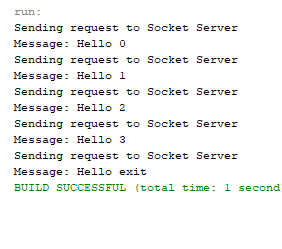
**}}**

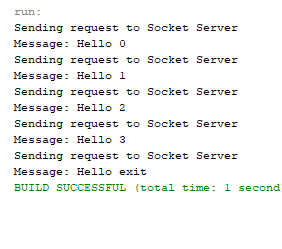
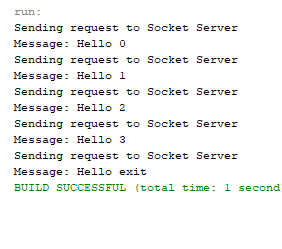
**OUTPUT**

****

In task 1, modify the run method to randomly sleep the thread for few milliseconds. Observe the output

****



1. Create a multi-threaded client server application in Java.

**USER SIDE**

**import java.io.DataInputStream;**

**import java.io.DataOutputStream;**

**import java.net.ServerSocket;**

**import java.io.IOException;**

**import java.net.InetAddress;**

**import java.net.Socket;**

**import java.net.UnknownHostException;**

**/\*\***

**\***

**\* @author Binyamin**

**\*/**

**public class User {**

**/\*\***

**\* @param args the command line arguments**

**\*/**

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

**// TODO code application logic here**

**{**

**try**

**{**

**ServerSocket ss = new ServerSocket(100);**

**while(true){**

**Socket s = ss.accept();**

**ServerThread st = new ServerThread(s);**

**Thread t = new Thread(st);**

**t.start();**

**}**

**}**

**catch (Exception e)**

**{**

**}**

**}**

**}**

**}**

**CLASS**

**import java.io.DataInputStream;**

**import java.io.DataOutputStream;**

**import java.net.ServerSocket;**

**import java.net.Socket;**

**/\*\***

**\***

**\* @author Binyamin**

**\*/**

**public class ServerThread implements Runnable{**

**private Socket s;**

**public ServerThread(Socket s)**

**{**

**this.s=s;**

**}**

**@Override**

**public void run()**

**{**

**try**

**{**

**DataOutputStream dos = new DataOutputStream(s.getOutputStream());**

**DataInputStream dis = new DataInputStream(s.getInputStream());**

**System.out.println(dis.readUTF());**

**s.close();**

**}**

**catch (Exception e)**

**{**

**}**

**}**

**}**

**OFFICER**

**package officer;**

**import java.io.DataInputStream;**

**import java.io.DataOutputStream;**

**import java.io.IOException;**

**import java.net.Socket;**

**/\*\***

**\***

**\* @author Binyamin**

**\*/**

**public class Officer {**

**/\*\***

**\* @param args the command line arguments**

**\*/**

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

**// TODO code application logic here**

**try{**

**Socket s = new Socket("192.168.109.56", 101);**

**DataOutputStream dos = new DataOutputStream(s.getOutputStream());**

**DataInputStream dis = new DataInputStream(s.getInputStream());**

**dos.writeUTF("Salam");**

**System.out.println(dis.readUTF());**

**s.close();**

**}**

**catch(Exception ex)**

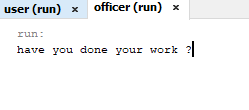
**{**

**}**

**}**

**}**

**OUTPUT**

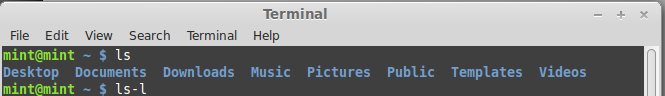
****

**To study and execute basic Linux commands on a terminal**

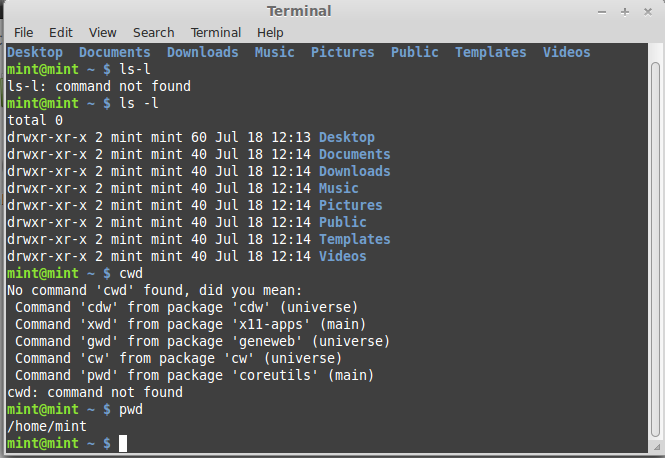
Linux is a Unix-like and mostly POSIX-compliant computer operating system (OS) assembled under the model of free and open-source software development and distribution. In this lab, we will work on Ubuntu, one of the flavors of Linux. For this purpose, we will use virtualization environment.

**Lab Tasks:**

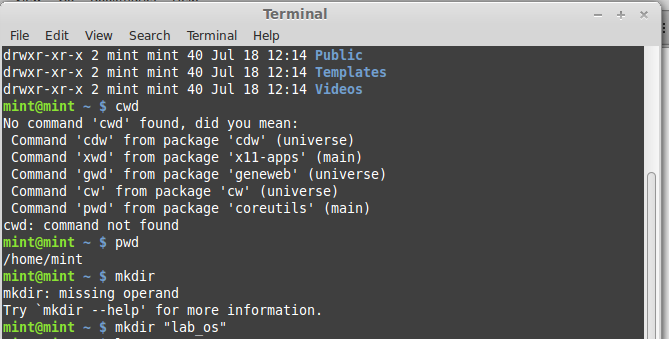
1. Using ls command find out the contents of current directory



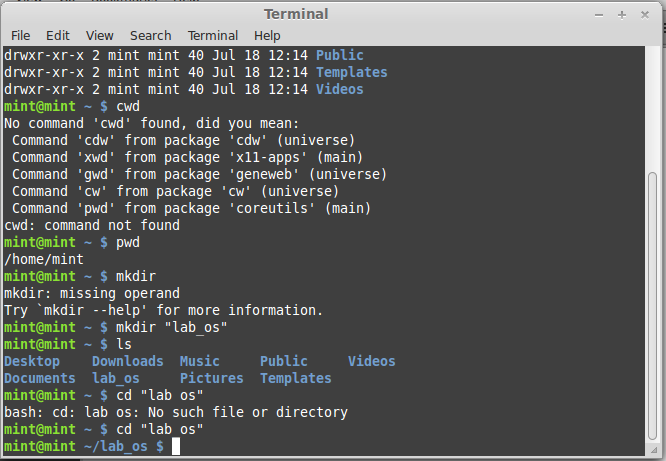
1. What are the permissions for normal user, group and world for each file
2. User - Read, Write, Execute
3. Group - Read, Write, Execute
4. World - Read, Write, Execute
5. Find out the name of current working directory



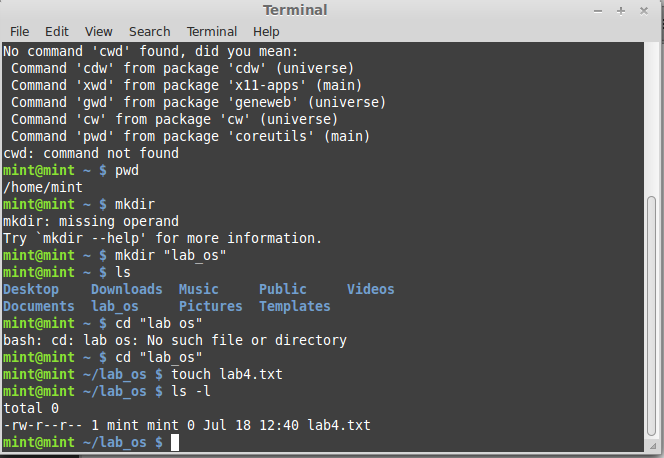
1. Create a new folder named “lab os” using the mkdir command



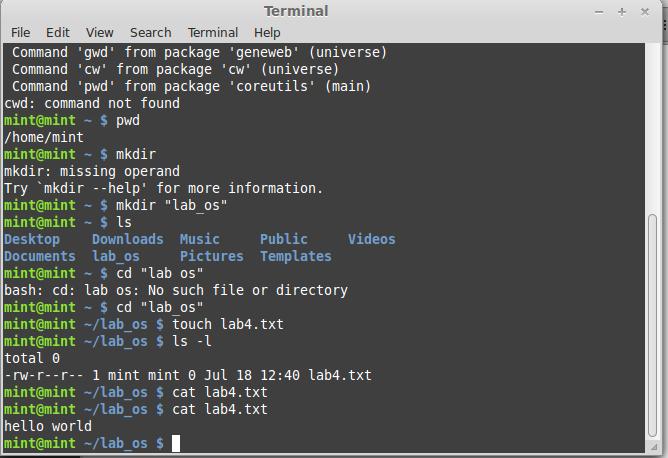
1. Switch to the directory “lab os”



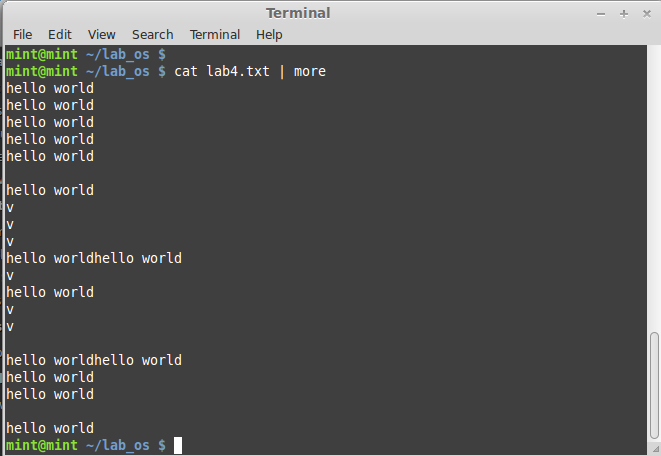
1. Create a file in the directory named “lab4.txt” using touch command



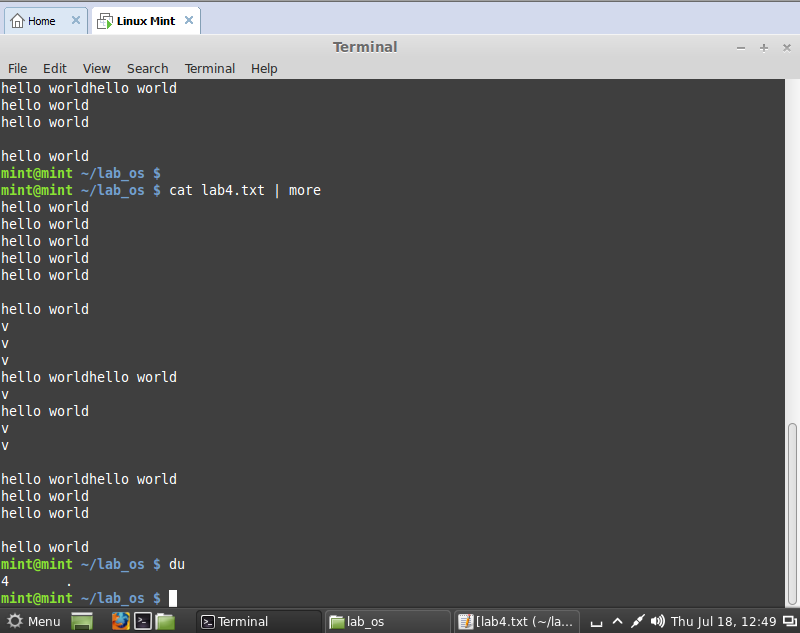
1. List down the contents of file using cat command. Try using “more” and “less” option



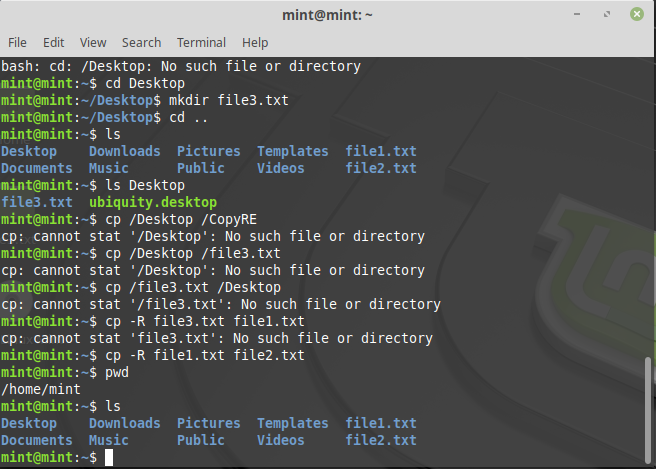
More:



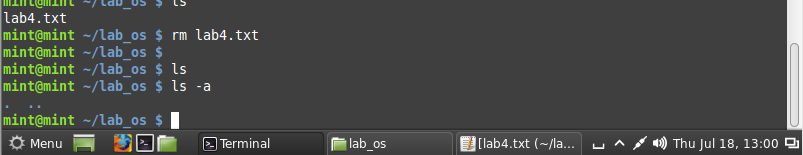
1. Find out the space consumed by directory using “du” command



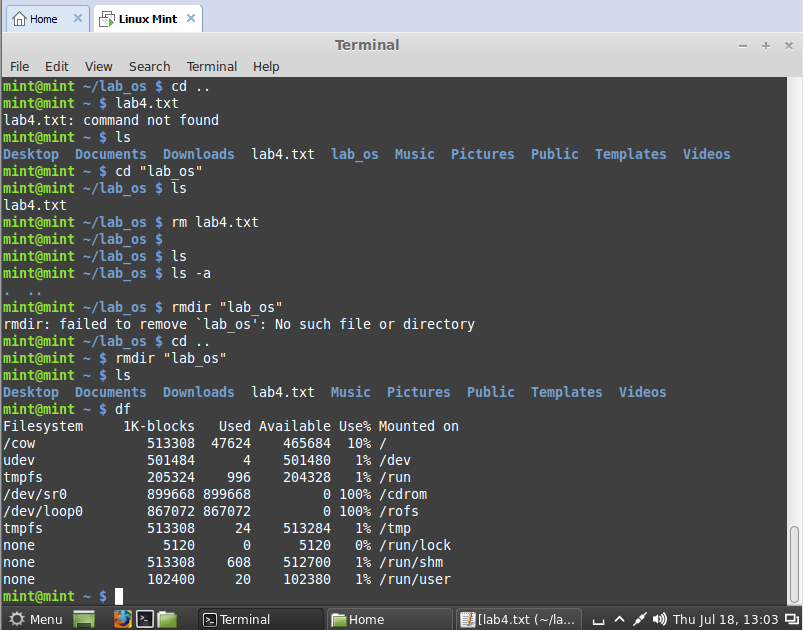
1. Copy the file to parent directory using cp command



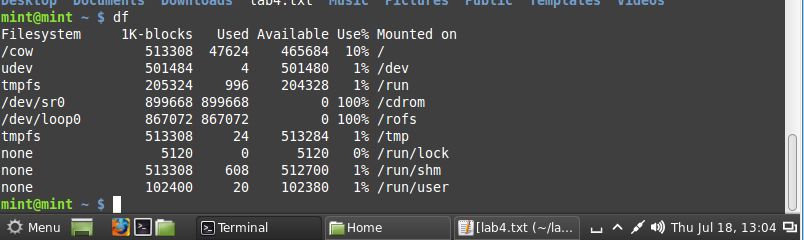
Remove the file using rm command



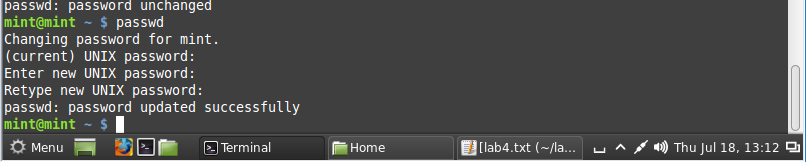
1. Remove the directory using rmdir command



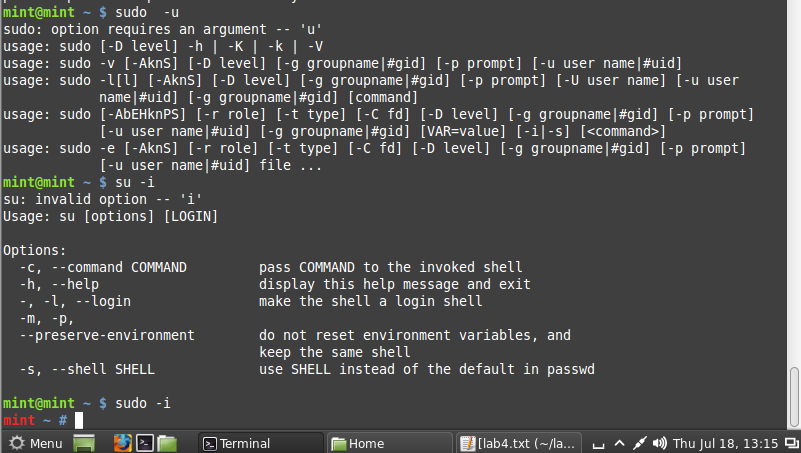
1. Check the free space on disk using df command



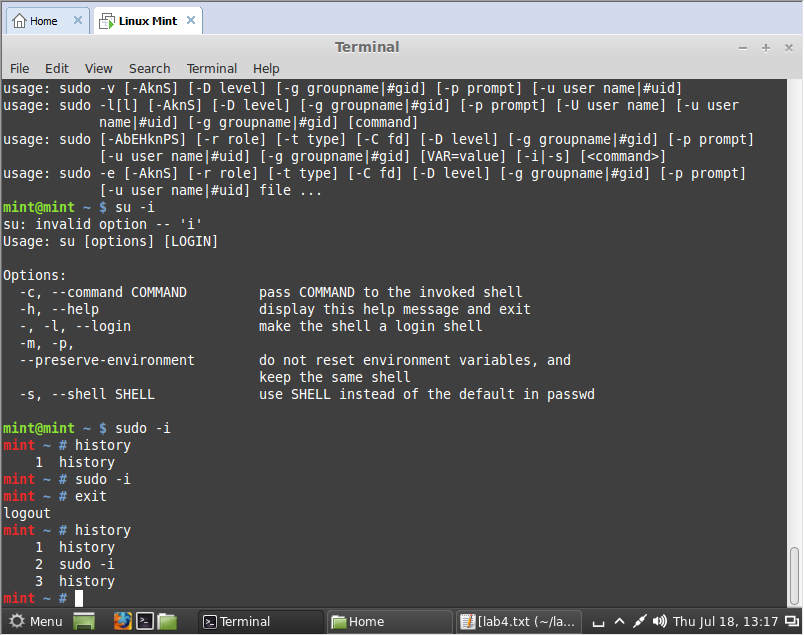
1. Change the password of the user using passwd command



1. Switch to super user, using the command “su”



1. Using the history command, list down the commands run on the terminal window



**To study and execute system administration commands on a terminal**

**Instructions:**

Linux comprises a set of commands for basic system administration. In this lab, we will study these commands.

**Lab Tasks:**

1. Using the ‘uptime' command, since how long your system is running and the number of users that are currently logged in.



1. Using the ‘w’, display the users currently logged in and their process along-with load averages



1. Using the ‘users’ command, display the currently logged in users.



1. Using the ‘top’ command, display processor activity of your system and also displays tasks managed by kernel in real-time.



Using ‘tar’ command, compress your home directory in Linux



1. ‘lsof’ command to list all open files



1. Using the ‘last’ command, watch activity of ‘mint’ user in the system



1. Using the ‘env’ command, lists all the environment variables of your system. Use ‘echo’ command to print values of $HOME and $PATH



1. The ‘ps’ command displays about processes running in the system. Try option –ax, -u.



The ‘kill’ command can be used to terminate process. Using this command terminate some processes of your system



1. ‘ifconfig’ command is used to show the configuration of internet on LINUX. Use this command to find IP and MAC address of your computer



Using the ‘netstat’ command, show the status of your network



1. Using the ping command, to ping your localhost



1. Create a group named ‘student’ using groupadd



1. Create a file named ‘hello.txt’



1. Using the ‘useradd’ command create a user with your name in the group student



1. Change the owner of hello.txt to user you just created



1. Change the group owner of hello.txt to group student



|  |  |
| --- | --- |
| Lab 6 | To study and implement shell programming in Linux |

**To study and implement shell programming in Linux**

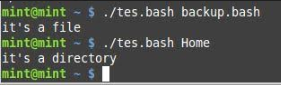
**Instructions:**

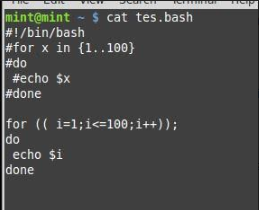
1. A shell script is a computer program designed to be run by the Unix shell, a command-line interpreter
2. The various dialects of shell scripts are considered to be scripting languages.

Typical operations performed by shell scripts include file manipulation, program execution, and printing text

**Lab Tasks:**

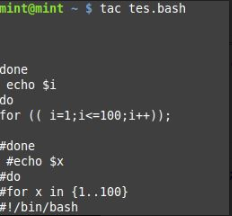
1. Write a script that backs itself up, that is, copies itself to a file named backup.sh.   
   Hint: Use the cat command





1. Write a script that echoes itself to stdout, but backwards.

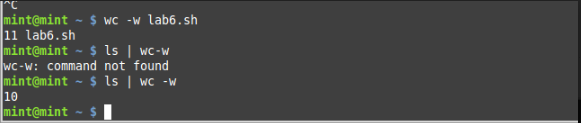
Hint: Use the tac command

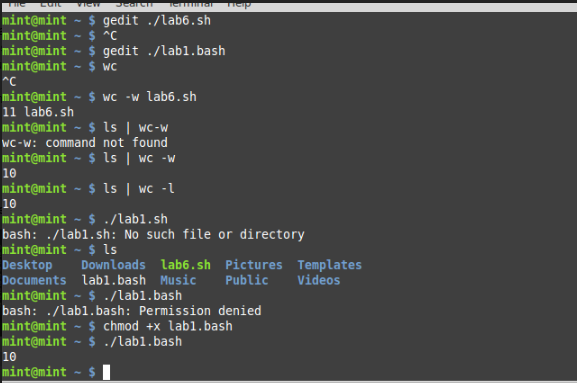


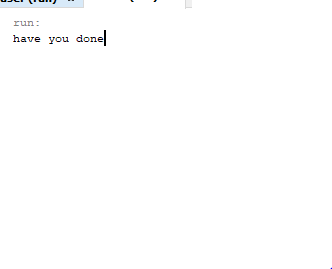
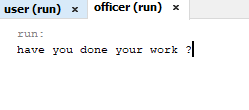
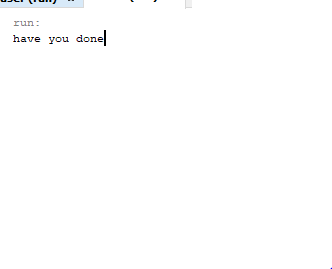
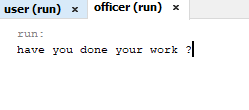
1. Perform a recursive directory listing on the user's home directory and save the information to a file.

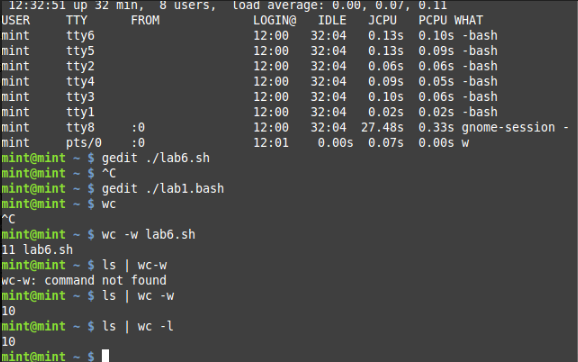


1. Write a script that reads each line of a target file, then writes the line back to stdout, but with an extra blank line following. This has the effect of double-spacing the file.

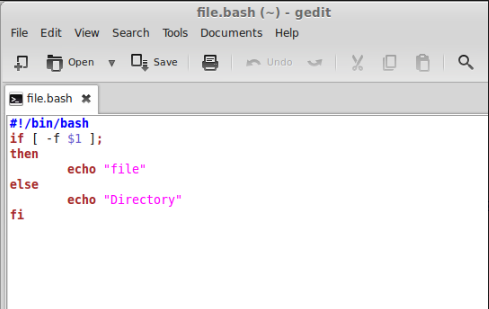




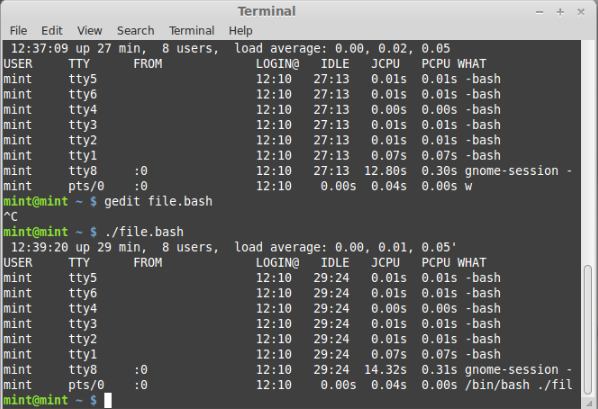
  



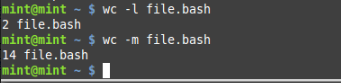
1. Write a shell script that takes a command –line argument and reports on whether it is directory, a file



Write a shell script program to display list of user currently logged in



1. Shell script program to count number of files in a Directory.





**To study and implement concurrency control techniques in Java**

Java provides the synchronized key word for implementing concurrency control while using multi-threaded applications. In this lab, you will learn how to implement these techniques.

Create the following program in Java:

public class UnsynchronizedExample {

public static void main(String[] args) {

new PrintStringsThread("Hello ", "there.");

new PrintStringsThread("How are ", "you?");

new PrintStringsThread("Thank you ", "very much!");

}

}

public class PrintStringsThread implements Runnable {

Thread thread;

String str1, str2;

PrintStringsThread(String str1, String str2) {

this.str1 = str1;

this.str2 = str2;

thread = new Thread(this);

thread.start();

}

public void run() {

TwoStrings.print(str1, str2);

}

}

public class TwoStrings {

// This method is not synchronized

static void print(String str1, String str2) {

System.out.print(str1);

try {

Thread.sleep(500);

} catch (InterruptedException ie) {

}

System.out.println(str2);

}

}

What output do you see? Explain the output.

**package unsynchronized;**

**/\*\***

**\***

**\* @author Binyamin**

**\*/**

**public class Unsynchronized {**

**/\*\***

**\* @param args the command line arguments**

**\*/**

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

**// TODO code application logic here**

**new printstring("Hello ", "there.");**

**new printstring("How are ", "you?");**

**new printstring("Thank you ", "very much!");**

**}**

**}**

**Another class**

**public class printstring implements Runnable {**

**Thread thread;**

**String str1, str2;**

**printstring(String str1, String str2) {**

**this.str1 = str1;**

**this.str2 = str2;**

**thread = new Thread(this);**

**thread.start();**

**}**

**@Override**

**public void run() {**

**two\_strings.print(str1, str2);**

**}**

**}**

**public class two\_strings {**

**// This method is not synchronized**

**static void print(String str1, String str2) {**

**System.out.print(str1);**

**try {**

**Thread.sleep(500);**

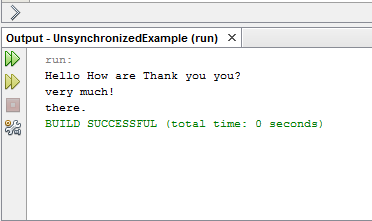
**} catch (InterruptedException ie) {**

**}**

**System.out.println(str2);**

**}}**

**Output**



1. Now use the synchronized methods to display the desired result.

**public class Synchronized {**

**/\*\***

**\* @param args the command line arguments**

**\*/**

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

**// TODO code application logic here**

**new printstring("Hello ", "there.");**

**new printstring("How are ", "you?");**

**new printstring("Thank you ", "very much!");**

**}**

**}**

**public class printstring implements Runnable{**

**Thread thread;**

**String str1, str2;**

**printstring(String str1, String str2) {**

**this.str1 = str1;**

**this.str2 = str2;**

**thread = new Thread();**

**thread.start();**

**}**

**@Override**

**public void run() {**

**two\_strings.print(str1, str2);**

**}**

**}**

**public class two\_strings {**

**static synchronized void print(String str1, String str2) {**

**System.out.print(str1);**

**try {**

**Thread.sleep(500);**

**} catch (InterruptedException ie) {**

**}**

**System.out.println(str2);**

**}**

**}**

**Output**



**To study and implement process scheduling algorithms in Java**

**Instructions:**

In this lab, we will implement different CPU scheduling techniques.

**Lab Tasks**

1. **Shortest Job First:** The number of processes and burst time is input from the user. The program should then print total access time, burst time and wait time for every process. Also print the average wait time.

Hint: Sort the element based on their burst time

1. Simulate the First Come First Serve and Priority scheduling algorithm.

package shotest\_job\_first;

import java.util.\*;

import java.util.ArrayList;

import java.util.Scanner;

/\*\*

**\***

**\* @author 7500**

**\*/**

**public class Shotest\_job\_first {**

**public int id;**

**public int bursttime ;**

**public int waittime;**

**public int priority;**

**public Shotest\_job\_first(int id, int bursttime, int priority){**

**this.id =id;**

**this.bursttime =bursttime;**

**this.priority =priority;**

**}**

**@Override**

**public String toString(){**

**return "Process : " + id + " Burst Time : " +this.bursttime + " Priority : " + this.priority + " Wait Time : " + this.waittime;**

**}**

**/\*\***

**\* @param args the command line arguments**

**\*/**

**}**

**//main class**

**/\***

**\* To change this license header, choose License Headers in Project Properties.**

**\* To change this template file, choose Tools | Templates**

**\* and open the template in the editor.**

**\*/**

**package shotest\_job\_first;**

**import java.util.ArrayList;**

**import java.util.Scanner;**

**/\*\***

**\***

**\* @author 7500**

**\*/**

**public class Process {**

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

**// TODO code application logic here**

**Scanner sc = new Scanner (System.in);**

**ArrayList< Shotest\_job\_first> al = new ArrayList< Shotest\_job\_first>();**

**System.out.println("Enter the Number of Process : ");**

**int n = sc.nextInt();**

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

**System.out.printf("Enter the burst Time and Priority %d : " , i);**

**Shotest\_job\_first p = new Shotest\_job\_first(i,sc.nextInt(),sc.nextInt());**

**al.add(p);**

**}**

**System.out.println(al);**

**System.out.println("Based on SJF");**

**sjf(new ArrayList(al));**

**System.out.println("Based on priority");**

**priorityBased(new ArrayList(al));**

**System.out.println("Based on first come fist serve");**

**fcfs(new ArrayList(al));**

**}**

**public static void sjf (ArrayList al){**

**int currentTime = 0;**

**while (al.size()!= 0){**

**//Pick the shortest Job**

**int s = 0;**

**for (int i = 1;i< al.size(); i++){**

**Shotest\_job\_first p = ( Shotest\_job\_first)al.get(i);**

**Shotest\_job\_first shortest = ( Shotest\_job\_first)al.get(s);**

**if(p.bursttime<shortest.bursttime){**

**s=i;**

**}**

**}**

**Shotest\_job\_first shortest = ( Shotest\_job\_first)al.get(s);**

**shortest.waittime=currentTime;**

**System.out.println(shortest);**

**currentTime += shortest.bursttime;**

**al.remove(s);**

**}**

**}**

**public static void priorityBased (ArrayList al){**

**int currentTime = 0;**

**while (al.size()!= 0)**

**//Pick the shortest Job**

**int s = 0;**

**for (int i = 1;i< al.size(); i++){**

**Shotest\_job\_first p = ( Shotest\_job\_first)al.get(i);**

**Shotest\_job\_first highest = ( Shotest\_job\_first)al.get(s);**

**if(p.priority>highest.priority){**

**s=i;**

**}**

**}**

**Shotest\_job\_first shortest = ( Shotest\_job\_first)al.get(s);**

**shortest.waittime=currentTime;**

**System.out.println(shortest);**

**currentTime += shortest.bursttime;**

**al.remove(s);**

**}**

**}**

**public static void fcfs(ArrayList al)**

**{**

**int currentTime = 0;**

**for (int i = 1;i< al.size(); i++){**

**Shotest\_job\_first p = ( Shotest\_job\_first)al.get(i);**

**p.waittime=currentTime;**

**System.out.println(p);**

**currentTime += p.bursttime;**

**}**

**}**

**}**

