

**OPERATING SYSTEM**

Lab Manual

[Summer 2019]

|  |  |
| --- | --- |
| Student Name: Ali Zain Aziz |  |
| Student Id: 12253 |  |
|  |  |

|  |  |
| --- | --- |
| 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 containers and dockers |  |
| **11** | \_\_/\_\_/\_\_ | To study and implement page replacement algorithms in Java |  |
| **12** | \_\_/\_\_/\_\_ | To study and implement disk scheduling algorithm in Java |  |

**To study and implement File I/O in Java**

**Instructions:**

* Try to compile the class File Test. 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

ANSWER: 01

package filetest;

import java.io.BufferedWriter;

import java.io.File;

import java.io.FileWriter;

import java.io.IOException;

import java.io.PrintWriter;

/\*\*

\*

\* @author Alizain Aziz

\*/

public class FileTest {

public void fileWrite() throws IOException

{

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

PrintWriter out = new PrintWriter

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

out.print("Hello ");

out.println("world");

out.close();

}

/\*\*

\* @param args the command line arguments

\*/

public static void main(String[] args) {

FileTest fileTest = new FileTest();

fileTest.fileWrite();

} }

* Run your program again. If all went successfully, open up "My Computer", and find your FilePractice folder on your K drive. You should be able to find the file "outputfile.txt". Double click on it, and take a look. What do you see?

package file\_test2;

import java.io.BufferedWriter;

import java.io.File;

import java.io.FileWriter;

import java.io.IOException;

import java.io.PrintWriter;

/\*\*

\*

\* @author Alizain Aziz

\*/

public class File\_Test2 {

public void fileWrite() throws IOException

{

try{

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

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

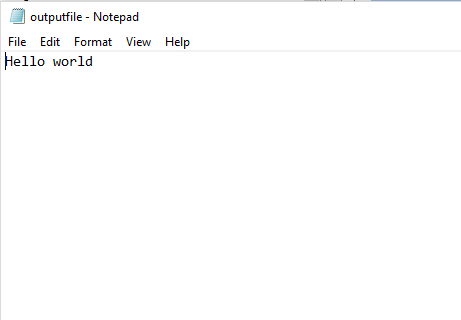
\*/

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

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

fileTest.fileWrite();

} }



* 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

ANSWER: 3

package filetest2;

import java.io.BufferedWriter;

import java.io.File;

import java.io.FileWriter;

import java.io.PrintWriter;

/\*\*

\*

\* @author Alizain Aziz

\*/

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){

} }

/\*\*

\* @param args the command line arguments

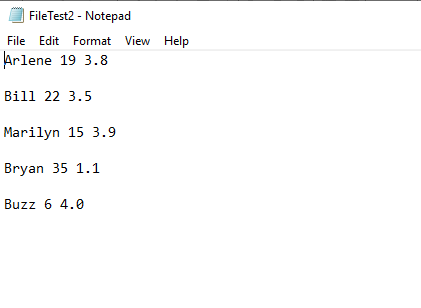
\*/

public static void main(String[] args) {

FileTest2 fileTest = new FileTest2();

fileTest.fileWrite();

} }



* 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?

ANSWER : 4

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 Alizain Aziz

\*/

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 + ".");

}

/\*\*

\* @param args the command line arguments

\*/

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

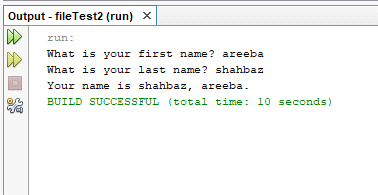
FileTest2 fileTest = new FileTest2();

fileTest.fileWrite();

fileTest.consoleRead();

}

}



* 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 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;

import static java.lang.System.in;

import java.util.StringTokenizer;

/\*\*

\*

\* @author Alizain Aziz

\*/

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 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();

}

/\*\*

\* @param args the command line arguments

\*/

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

FileTest2 fileTest = new FileTest2();

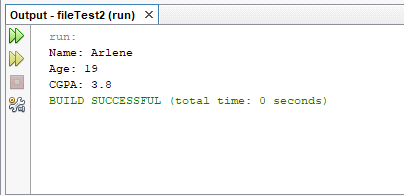
fileTest.fileWrite();

//fileTest.consoleRead();

fileTest.fileRead();

}

}



**To study and implement socket programming in Java**

Sockets provide the communication mechanism between two computers using Transmission Control Protocol (TCP) or User Datagram Protocol (UDP). This lab will demonstrate how to implement TCP sockets using Java. Before starting the lab, download and install Java and Eclipse IDE by following the instructions below:

* Download and Install Java Development Kit (JDK)’s latest version
* Download ‘Eclipse’ on your computer
* Go to Eclipse folder and Run eclipse.exe file
* The Eclipse environment will start. Now perform the lab tasks.

**Lab Tasks:**

* Find the IP address of a local host using java program. Use the InetAddress class.

package getip;

import java.net.InetAddress;

import java.net.UnknownHostException;

/\*\*

\*

\* @author Alizain Aziz

\*/

public class Getip {

/\*\*

\* @param args the command line arguments

\*/

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

InetAddress myIP=InetAddress.getLocalHost();

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

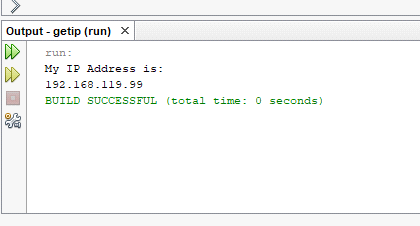
\* address string in textual presentation.

\*/

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

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

}}



* Write a small port scanner application. The program usage is 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 use : 137

package server;

import java.io.IOException;

import java.net.InetSocketAddress;

import java.net.Socket;

/\*\*

\*

\* @author Alizain Aziz

\*/

public class Server {

/\*\*

\* @param args the command line arguments

\*/

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

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

try{

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

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

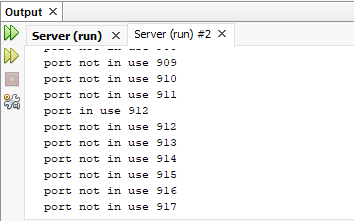
s.close();}

catch(IOException s){

}

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

}}}



* Write a small server that accepts socket connection on port 2020. Develop a client application that connects to the server.
* Using BufferedOutputStream, write to the server “Hello”
* The server should respond with the word Hello

package javaapplication4;

import java.io.DataInputStream;

import java.io.DataOutputStream;

import java.io.IOException;

import java.io.\*;

import java.\*;

import java.net.Socket;

import java.io.OutputStream;

/\*\*

\*

\* @author Alizain Aziz

\*/

public class Socketlist {

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

Socket ss =new Socket("192.168.119.92",130);// server connectivity from client

String message1=("salam");

String message2=("salassssssssssm");

// comunication

OutputStream ostream = ss.getOutputStream();

DataOutputStream dos= new DataOutputStream(ostream);

dos.writeBytes(message1);

dos.writeBytes(message2);

System.out.println();

dos.close();

ostream.close();

ss.close();

}

}

* Modify the Task 3 to develop an echo server

try

{

ServerSocket s=new ServerSocket(2020); //allowing port to connect

Socket ss=s.accept(); //accept client

DataInputStream dis=new DataInputStream(ss.getInputStream());

DataOutputStream dos=new DataOutputStream(ss.getOutputStream());

String msg=dis.readUTF();

System.out.println(msg);

dos.writeUTF(msg);

s.close();

ss.close();

}

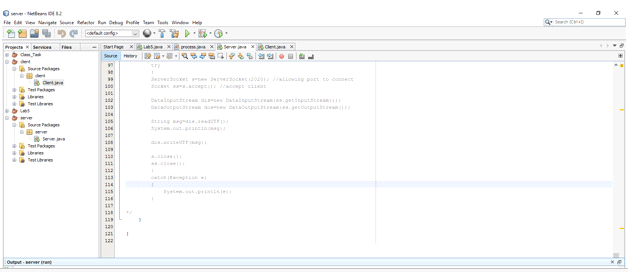
catch(Exception e)

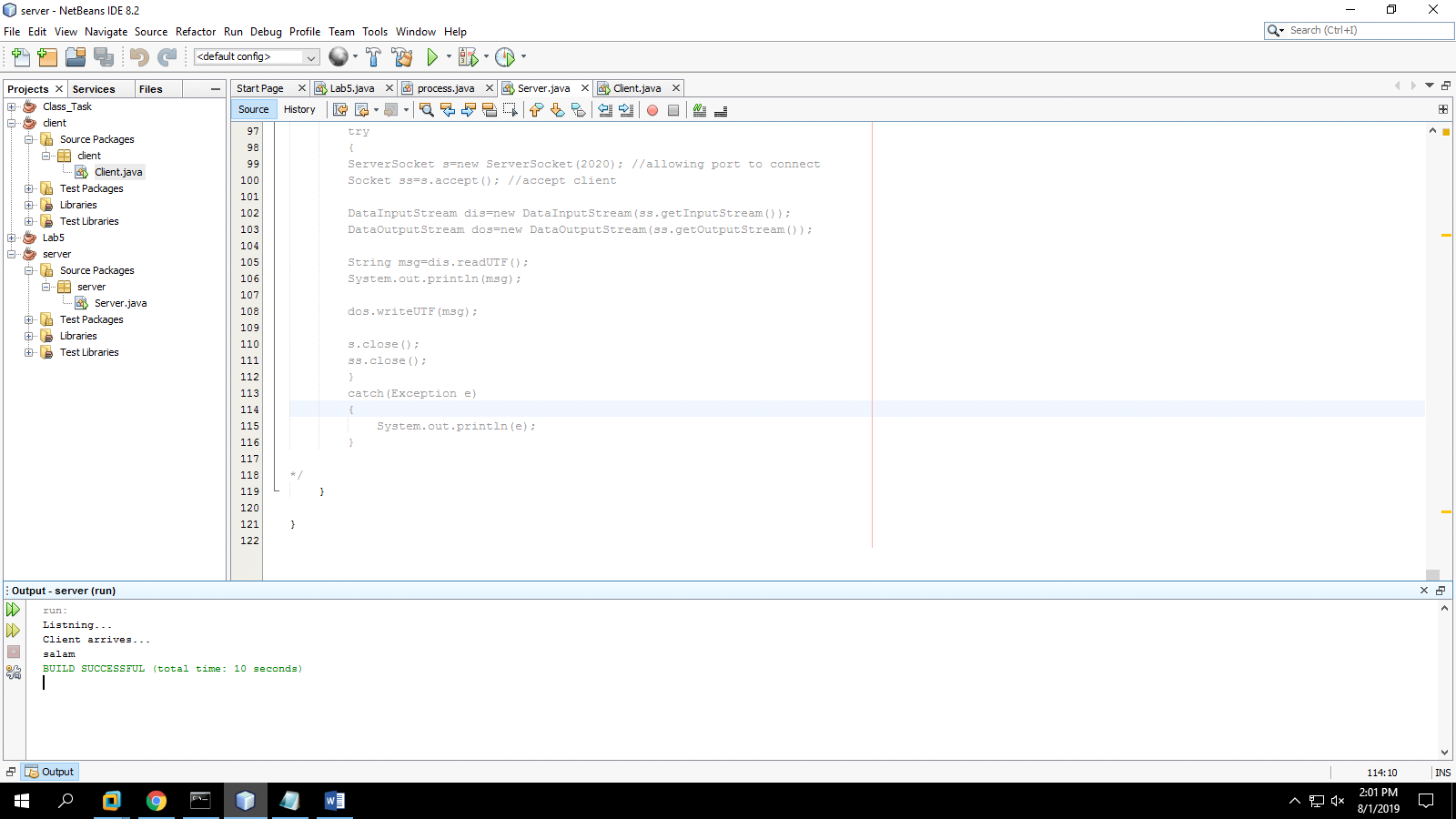
{

System.out.println(e);

}

}  
}





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

**Instructions:**

* A thread is an independent unit of execution.
* In Java, the Runnable interface and Thread class of package java.lang are used for implementation of thread
* 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  
 }

}

* 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:**

* 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?

import java.util.logging.Level;

import java.util.logging.Logger;

/\*\*

\*

\* @author Alizain Aziz

\*/

public class MyThread implements Runnable {

String name;

public MyThread(String n){

this.name=n;

}

@Override

public void run() {

while(true){

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

try {

Thread.sleep(100);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

Main class code

/\*\*

\*

\* @author 7500

\*/

public class main {

public static void main (String[] args){

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

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

Thread t=new Thread(m);

t.start();

}}}

TASK#03:

package multithreadserver;

import java.io.IOException;

import java.io.PrintStream;

import java.net.ServerSocket;

import java.net.Socket;

/\*\*

\*

\* @author Alizain Aziz

\*/

public class MultiThreadServer implements Runnable{

Socket csocket;

MultiThreadServer(Socket csocket) {

this.csocket = csocket;

}

/\*\*

\* @param args the command line arguments

\*/

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

// TODO code application logic here

ServerSocket ssock = new ServerSocket(1234);

System.out.println("Listening");

while (true) {

Socket sock = ssock.accept();

System.out.println("Connected");

new Thread(new MultiThreadServer(sock)).start();

}

}

@Override

public void run() {

try {

PrintStream pstream = new PrintStream(csocket.getOutputStream());

for (int i = 100; i >= 0; i--) {

pstream.println(i + " bottles of beer on the wall");

}

pstream.close();

csocket.close();

} catch (IOException e) {

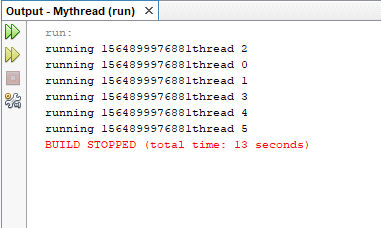
System.out.println(e);

}

}

}

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



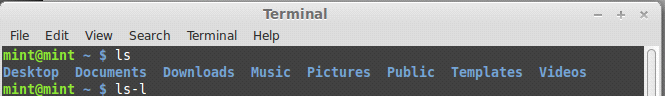
* Create a multi-threaded client server application in Java.

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

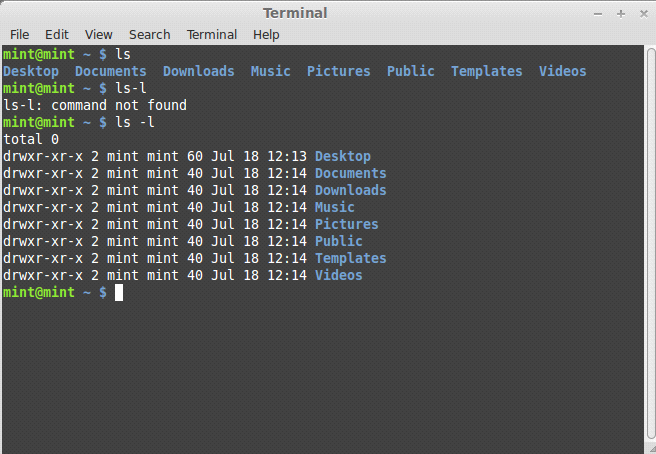
Linux is a Unix-like and mostly POSIX-compliant computer operating system (OS) assembled under the model of free and open-source software development and distribution. In this lab, we will work on Ubuntu, one of the flavors of Linux. For this purpose, we will use virtualization environment.

**Lab Tasks:**

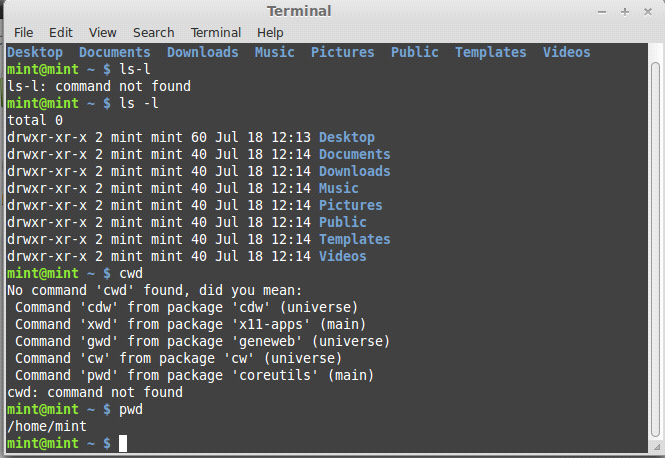
* Using ls command find out the contents of current directory



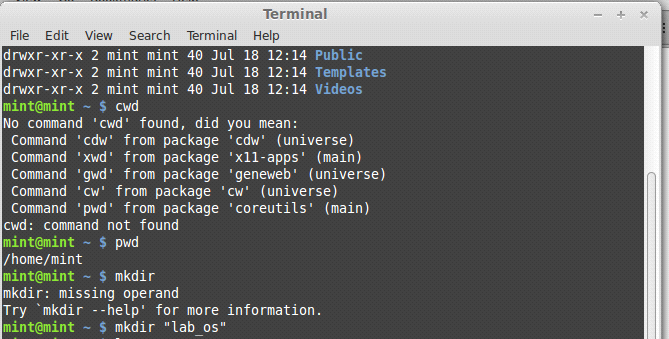
* What are the permissions for normal user, group and world for each file.



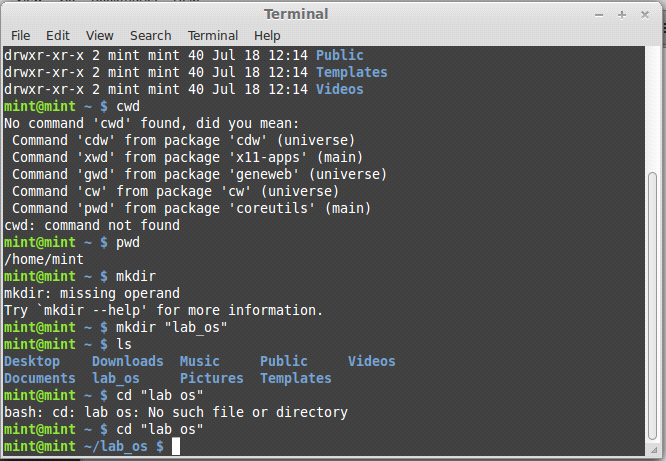
* Find out the name of current working directory.



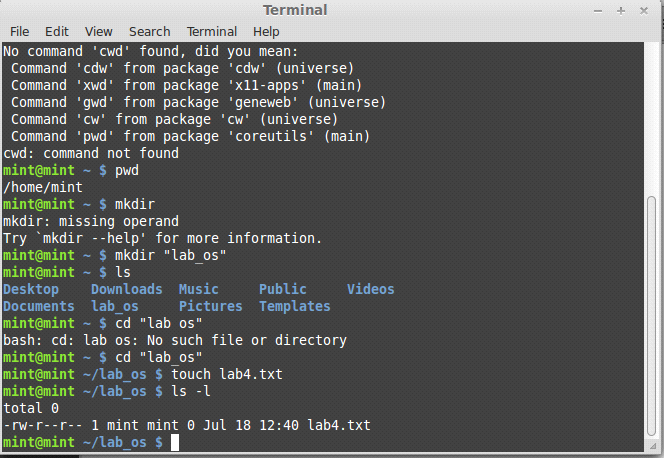
* Create a new folder named “lab os” using the mkdir command.



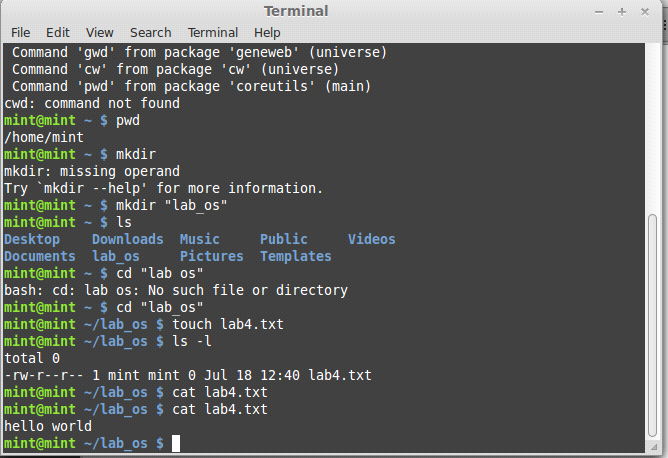
* Switch to the directory “lab os”



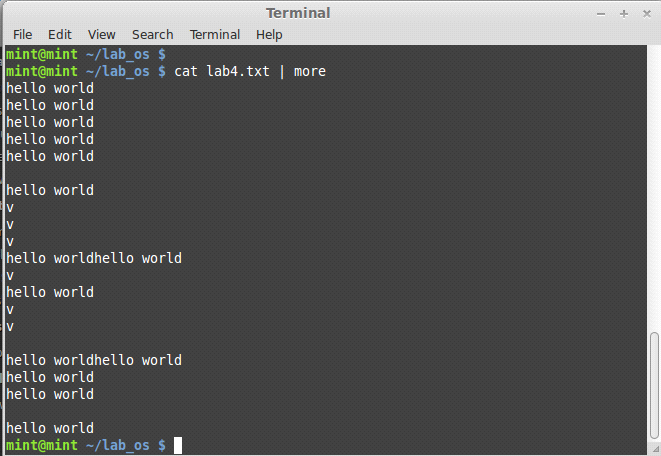
* Create a file in the directory named “lab4.txt” using touch command



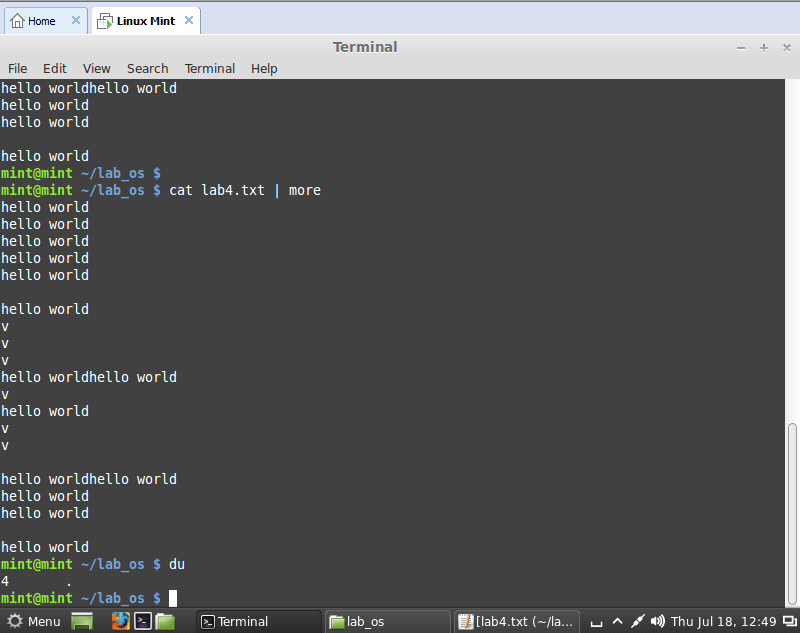
* List down the contents of file using cat command. Try using “more” and “less” option



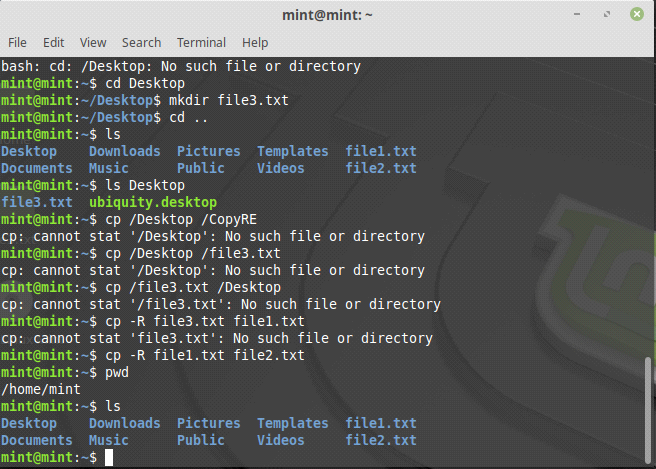
More:



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



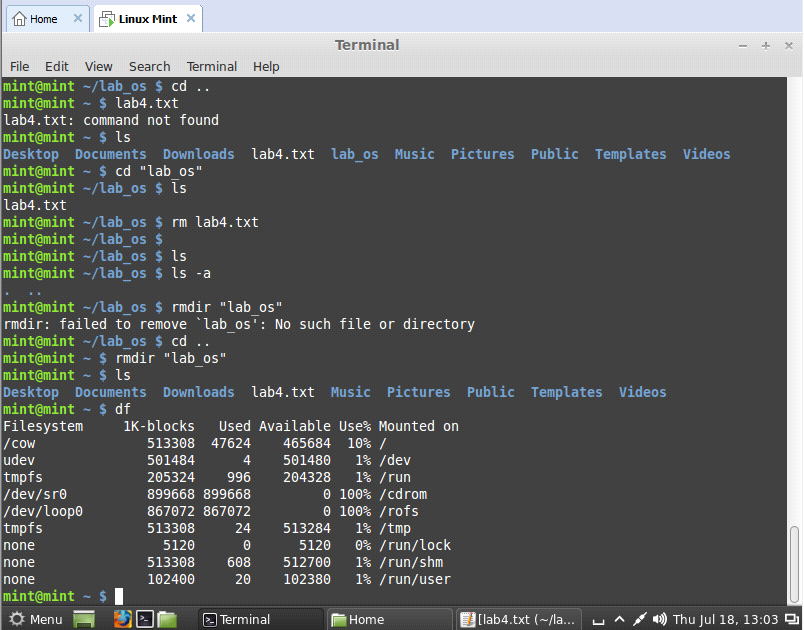
* Copy the file to parent directory using cp command



* Remove the file using rm command



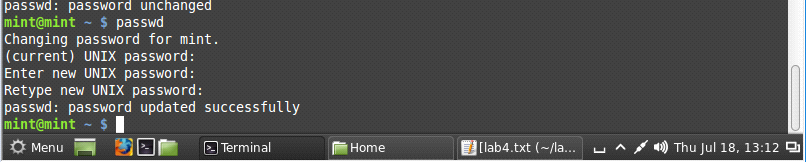
* Remove the directory using rmdir command



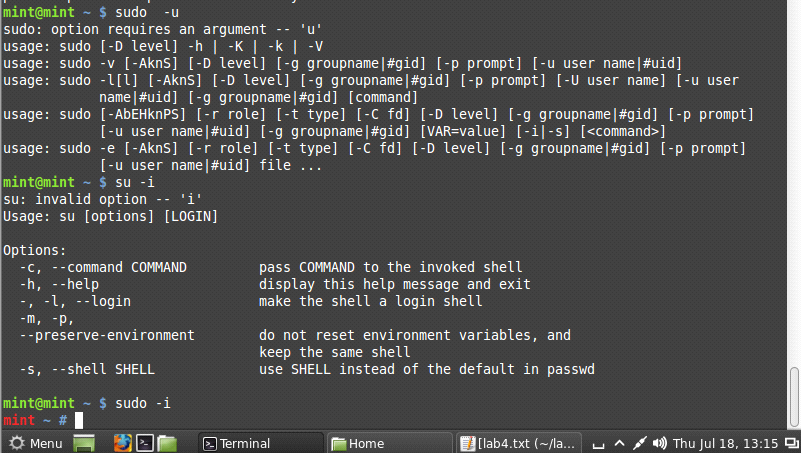
* Check the free space on disk using df command



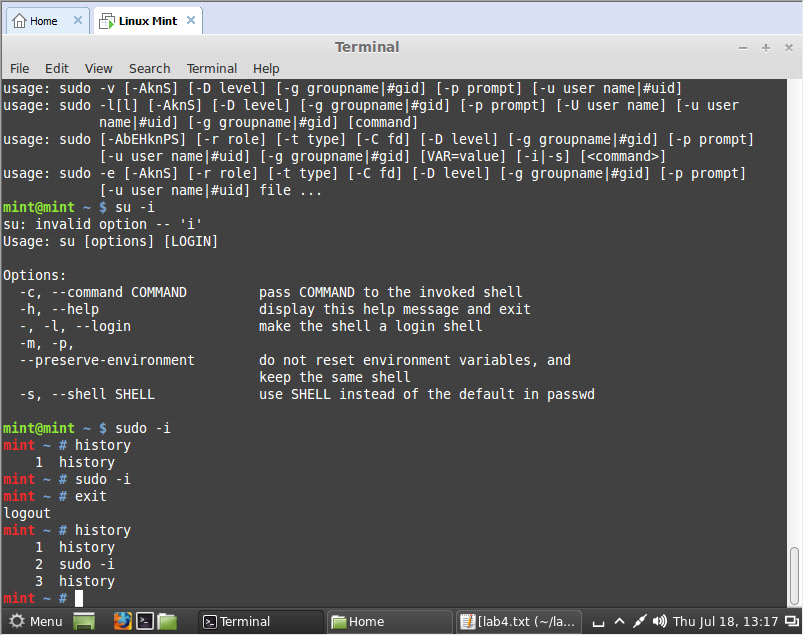
* Change the password of the user using passwd command



* Switch to super user, using the command “su”



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

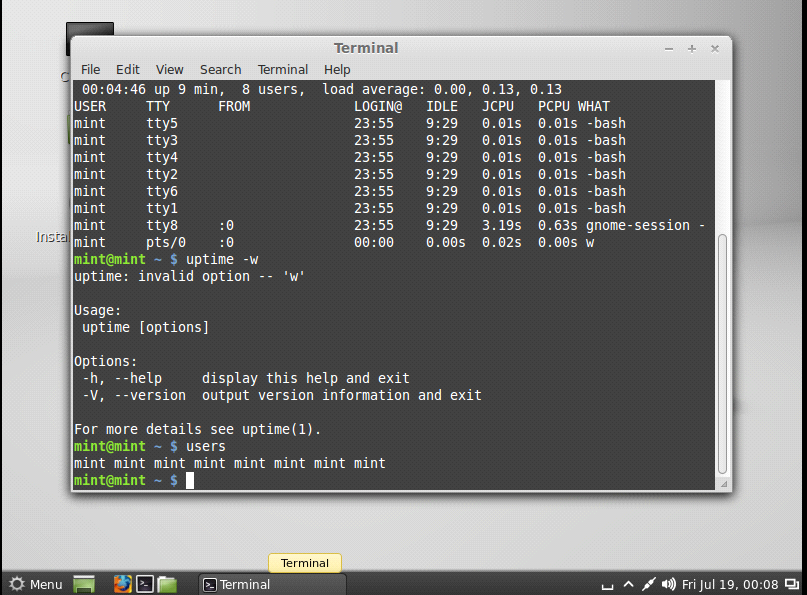


**To study and execute system administration commands on a terminal**

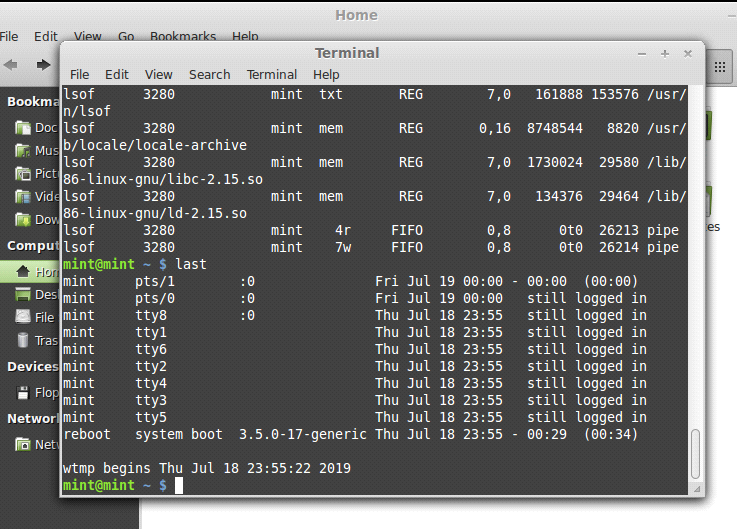
**Instructions:**

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

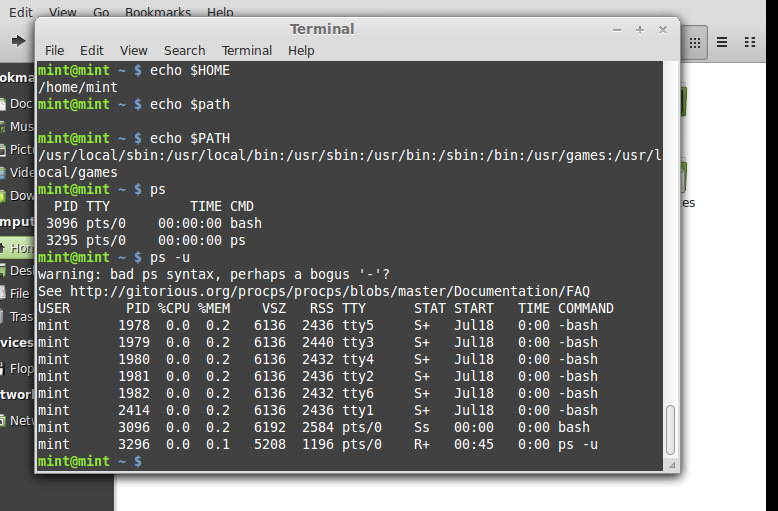
* Using the ‘users’ command, display the currently logged in users.



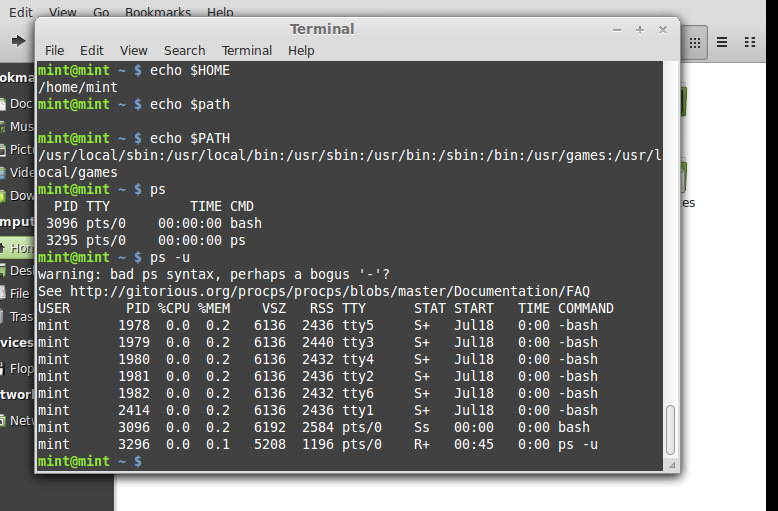
* Using the ‘top’ command, display processor activity of your system and also displays tasks managed by kernel in real-time.
* Using the ‘last’ command, watch activity of ‘mint’ user in the system



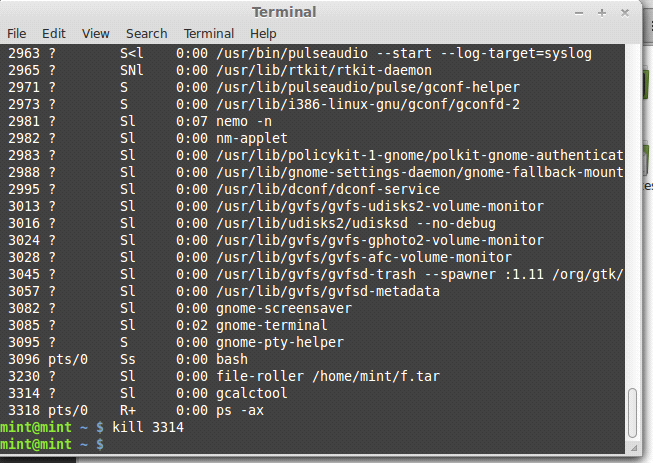
* Using the ‘env’ command, lists all the environment variables of your system. Use ‘echo’ command to print values of $HOME and $PATH



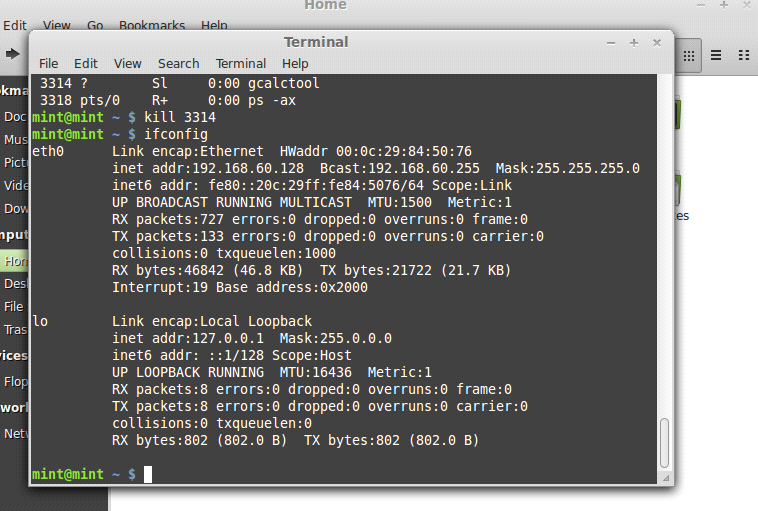
* The ‘ps’ command displays about processes running in the system. Try option –ax, -u.



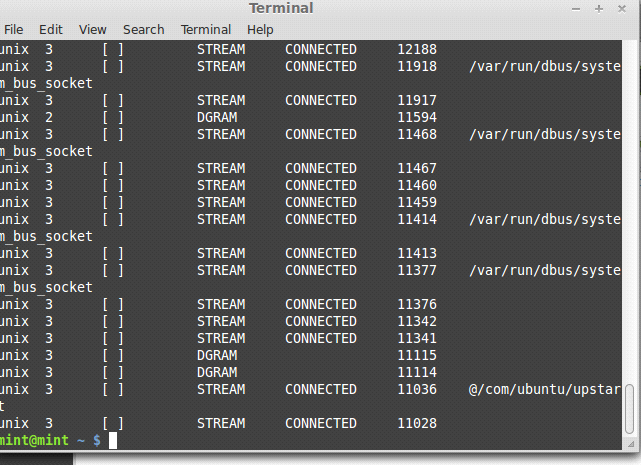
The ‘kill’ command can be used to terminate process. Using this command terminate some processes of your system



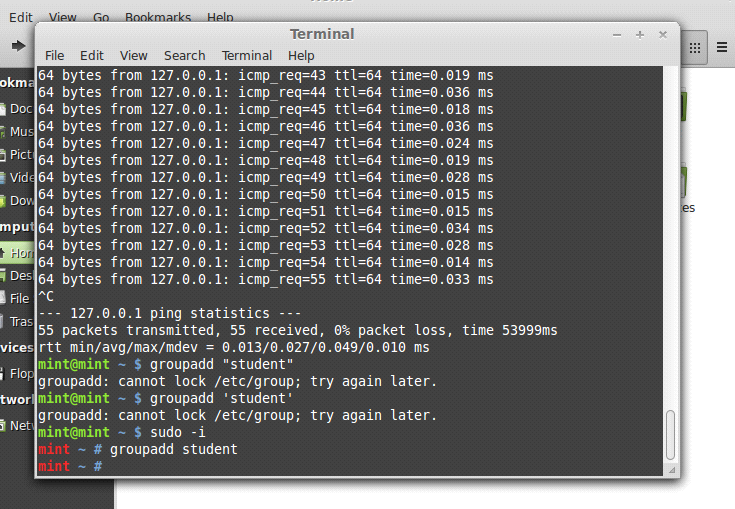
* ‘ifconfig’ command is used to show the configuration of internet on LINUX. Use this command to find IP and MAC address of your computer



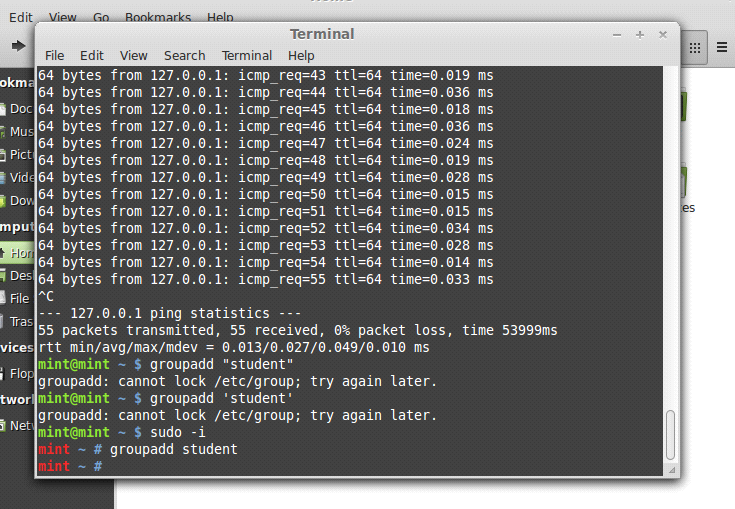
* Using the ‘netstat’ command, show the status of your network



* Using the ping command, to ping your localhost



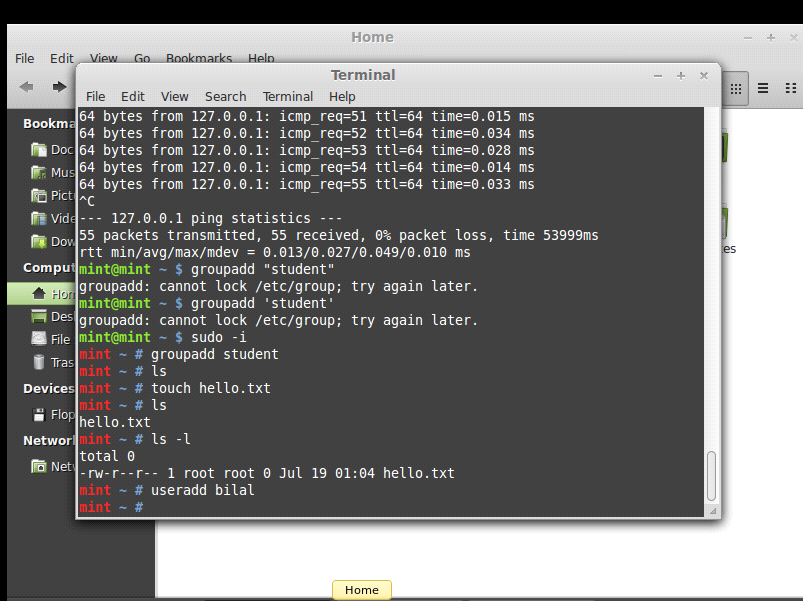
* Create a group named ‘student’ using groupadd



* Create a file named ‘hello.txt’



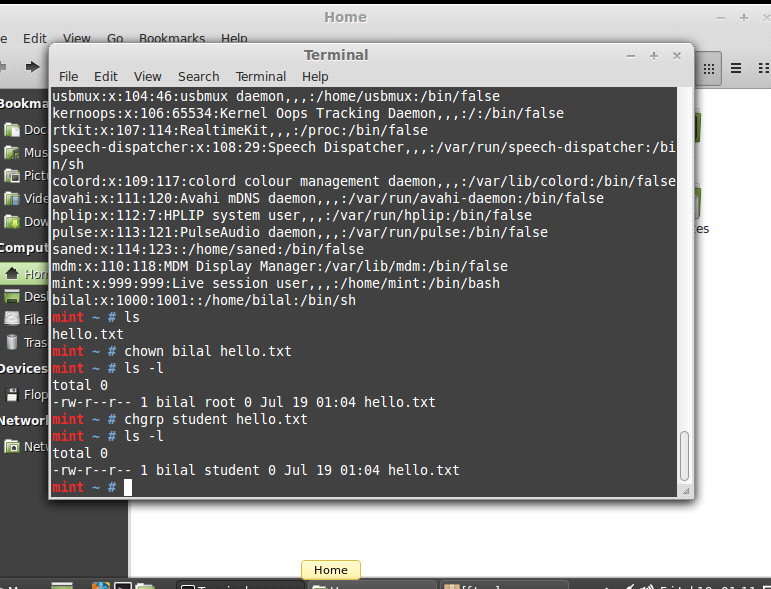
* Using the ‘useradd’ command create a user with your name in the group student



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



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



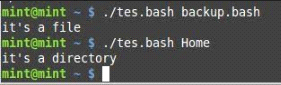
**To study and implement shell programming in Linux**

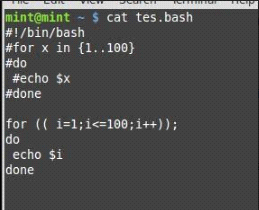
**Instructions:**

* A shell script is a computer program designed to be run by the Unix shell, a command-line interpreter
* 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:**

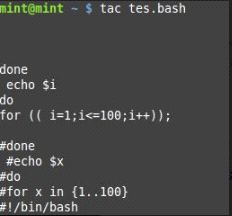
* Write a script that backs itself up, that is, copies itself to a file named backup.sh.   
  Hint: Use the cat command





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

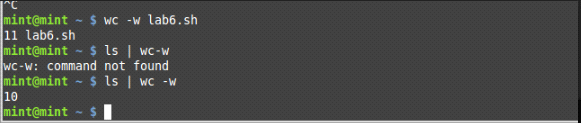
Hint: Use the tac command

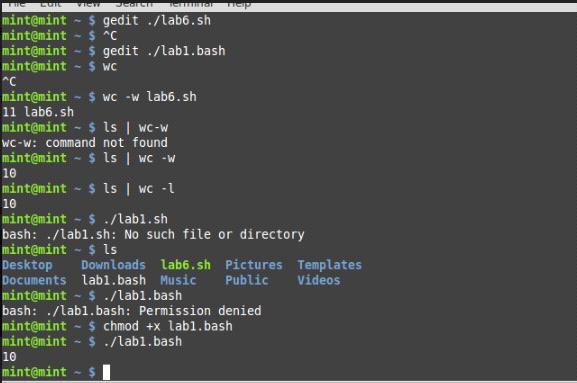


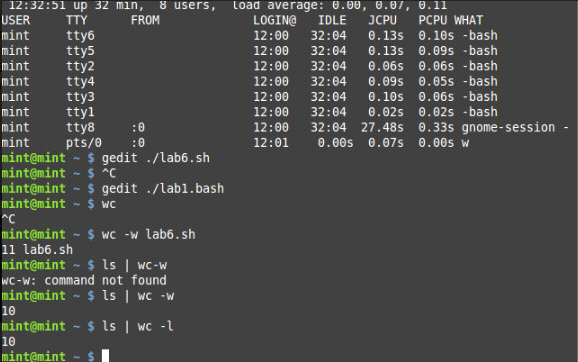
* Perform a recursive directory listing on the user's home directory and save the information to a file.



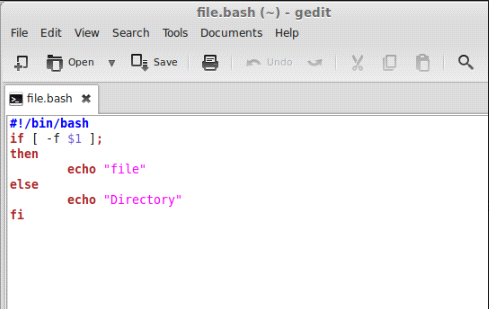
* Write a script that reads each line of a target file, then writes the line back to stdout, but with an extra blank line following. This has the effect of double-spacing the file.



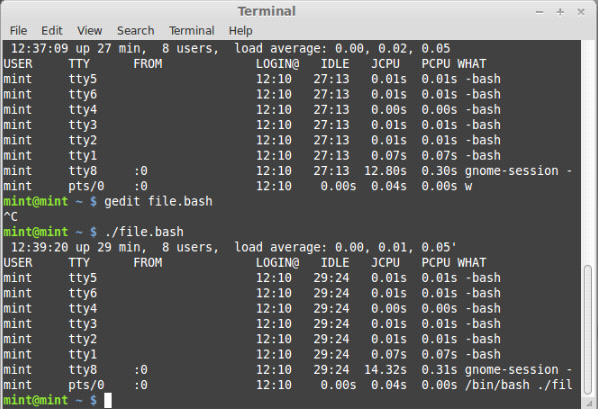




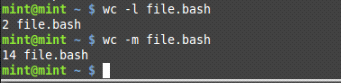
* Write a shell script that takes a command –line argument and reports on whether it is directory, a file



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



* 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.

**Instructions:**

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);

}

}

**Lab Tasks:**

* What output do you see? Explain the output.
* ANSWER:

//unsynchronized class

package unsynchronizedexample;

/\*\*

\*

\* @author 7500

\*/

public class UnsynchronizedExample {

/\*\*

\* @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!");

}

}

//printstring class

package unsynchronizedexample;

/\*\*

\*

\* @author 7500

\*/

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); }

}

//two\_strings class

package unsynchronizedexample;

/\*\*

\*

\* @author 7500

\*/

public class two\_strings {

// This method is not synchronized

static void print(String str1, String str2) {

System.out.print(str1);

try {

Thread.sleep(500);

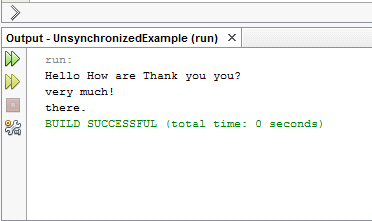
} catch (InterruptedException ie) {

}

System.out.println(str2);

}

}



Now use the synchronized methods to display the desired result.

//unsynchronized class:

package unsynchronizedexample;

/\*\*

\*

\* @author 7500

\*/

public class UnsynchronizedExample {

/\*\*

\* @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!");

}

}

//printstring class

package unsynchronizedexample;

/\*\*

\*

\* @author 7500

\*/

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);

}

}

//two\_string class

package unsynchronizedexample;

/\*\*

\*

\* @author 7500

\*/

public class two\_strings {

// This method is not synchronized

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

System.out.print(str1);

try {

Thread.sleep(500);

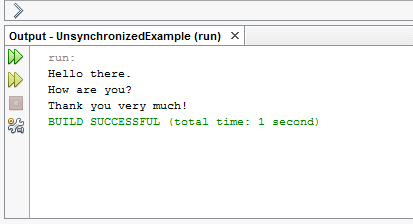
} catch (InterruptedException ie) {

}

System.out.println(str2);

}

}



* Now use the synchronized keyword on an object to synchronize.

ANSWER:

//UNSYNCHRONIZED CLASS

package objectbased;

/\*\*

\*

\* @author 7500

\*/

public class UnsynchronizedExample {

/\*\*

\* @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!");

}

}

//printstring class

package objectbased;

/\*\*

\*

\* @author 7500

\*/

public class printstring implements Runnable{

Thread thread;

String str1, str2;

static two\_strings ts=new two\_strings();

printstring(String str1, String str2) {

this.str1 = str1;

this.str2 = str2;

thread = new Thread(this);

thread.start();

}

@Override

public void run() {

ts.print(str1,str2);

}

}

//two\_strings class

package objectbased;

/\*\*

\*

\* @author 7500

\*/

public class two\_strings {

// This method is not synchronized

void print(String str1, String str2) {

synchronized(this){

System.out.print(str1);

try {

Thread.sleep(500);

} catch (InterruptedException ie) {

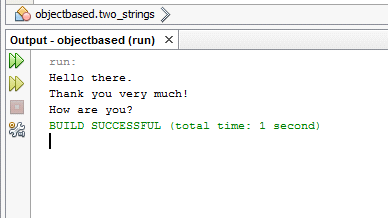
}

System.out.println(str2);

}

}

}



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

**Instructions:**

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

**Lab Tasks**

* **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

* 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

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;

}

}

}

