|  |
| --- |
|  |
| **OPERATING SYSTEM** |
| Lab Manual [Fall 2019] |
|  |
|  |
|  |

|  |
| --- |
|  |

**SUBMITTED BY: Hafiz M.Hammad Khan**

**STUDENT ID:12437**

**SUBMITTED TO: DR. NOMAN ISLAM**

LIST OF EXPERIMENTS

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S. No** | **Date** | **Experiment** | |  |
| **1** | 21/10/19 | To study and implement file I/O in Java | |  |
| **2** | 28/10/19 | To study and implement socket programming in Java | |  |
| **3** | 4/11/19 | To study and implement multi-threading in Java | |  |
| **4** | 11/11/19 | To study and implement concurrency control techniques in Java | |  |
| **5** | 18/11/19 | To study and execute basic Linux commands on a terminal | |  |
| **6** | 25/11/19 | To study and execute system administration commands on a terminal | |  |
| **7** | 2/12/19 | To study and implement shell programming in Linux | |  |
| **8** | 9/12/19 | To study and implement containers and dockers | |  |
| **9** | 16/12/19 | To study and setup a kubernetes cluster | |  |
| **10** | 16/12/19 | To study and implement process scheduling algorithms in Java | |  |
|  |  |  | |  |
|  |  |  | |  |
|  |  |  | |  |
|  |  |  | |  |
|  |  |  | |  |
|  |  |  |
|  |  |  |
|  |  |  |

**WEEK # 01**

**OPERATING SYSTEM (LAB FILE)**

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

**Instructions:**

Type 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();

}

}

**Lab Tasks**

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
2. 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?
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

1. 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?

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.

1. 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.

**Pseudo Code In Java:**

package FileTest;

import java.io.\*;

import static java.lang.System.in;

import java.util.\*;

public class FileTest {

public void fileWrite()

{

try{

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

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

out.println("DANIYAL 25 3.9");

out.println("BASIM 19 3.8");

out.println("FAZEEL 22 3.5");

out.println("AHMAD 35 1.1");

out.println("ALI 16 4.0");

out.close();

}

catch (Exception ex){

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

}

public void fileRead() throws IOException

{

try{

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

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

String text=null;

System.out.println(":: Display Record ::");

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());

System.out.println("\n");

}}

catch(Exception s){

System.out.println(s);

}

in.close();

}

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

// TODO code application logic here

FileTest fileTest = new FileTest( );

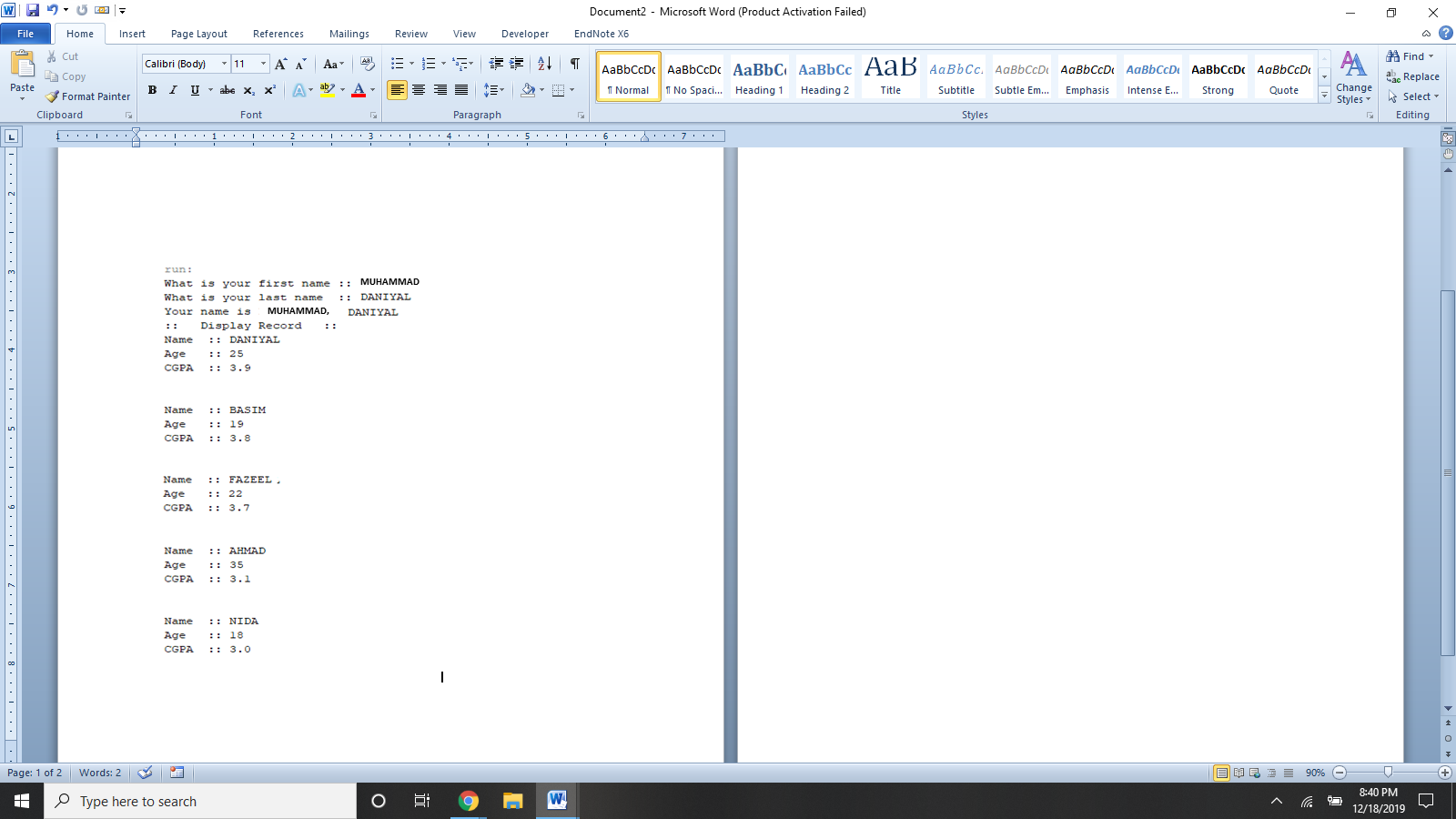
fileTest.fileWrite();

fileTest.consoleRead();

fileTest.fileRead();

}}

**Output & Display:**



**WEEK # 02**

**OPERATING SYSTEM (LAB FILE)**

**Lab 2**

**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:

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

**Lab Tasks:**

1. Find the IP address of a local host using java program. Use the Inet Address class.

**Pseudo Code:**

package ip.address;

import java.net.\*;

/\*\*

@author daniyal

\*/

public class IPADDRESS {

public static void main(String[] args) {

try{

InetAddress IP = InetAddress.getLocalHost();

System.out.println("IP of my system is :: "+IP.getHostAddress());

}catch(Exception ex){

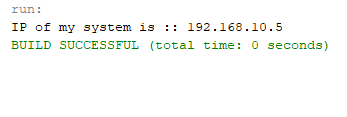
ex.printStackTrace();

}

}

}

**Output & Display:**



1. 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

**Pseudo Code:**

package server;

import java.io.\*;

import java.net.\*;

/\*\*

\*

\* @author daniyal

\*/

public class Server {

/\*\*

\* @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){

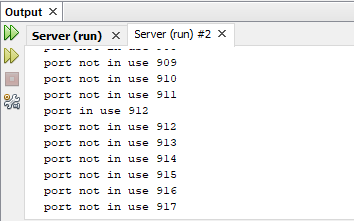
}

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

}

}

}

**Output & Display:**

1. Write a small server that accepts socket connection on port 2020. Develop a client application that connects to the server.
   1. Using BufferedOutputStream, write to the server “Hello”
   2. The server should respond with the word Hello
2. Modify the Task 3 to develop an echo server

**Pseudo Code:**

package server;

/\*\*

\* @author daniyal

// Server

\*/

import java.net.\*;

import java.io.\*;

public class Server {

public static void main(String[] args){

// TODO code application logic here

try{

ServerSocket Seve=new ServerSocket(5000);

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

Socket sp= Seve.accept();

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

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

System.out.println("Client Arrives ::");

// FristPart

// System.out.println("Read ::"+dis.readUTF());

// dos.writeUTF("Wa Alaikum Assalam");

//Second part

String data=dis.readUTF();

System.out.println("Read ::"+data);

dos.writeUTF(data);

Seve.close();

}catch(Exception ex){

System.out.println(""+ex);

ex.printStackTrace();

} } }

//Cleint

package server;

import java.io.DataInputStream;

import java.io.\*;

import java.net.\*;

/\*\*

\*

\* @author daniyal

\*/

public class Client {

public static void main(String[] args) {

try{

Socket sp=new Socket("localhost",5000);

System.out.println("Client Arrives ::");

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

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

dos.writeUTF("Assalam-O-Alaikum");

System.out.println("Read ::"+dis.readUTF());

//System.out.println(""+dos.writeUTF(dis.readUTF()));

sp.close();

}catch(Exception ex){

System.out.println(""+ex);

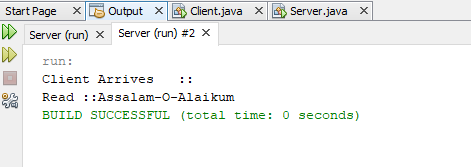
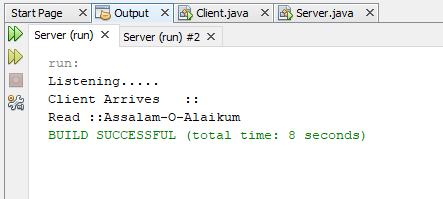
ex.printStackTrace();

}

}

}

**Output & Display:**



Client

**WEEK # 03**

**OPERATING SYSTEM (LAB FILE)**

**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() {

}

}

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.

**Pseudo Code:**

package thread;

/\*\*

\* @author daniyal

// Thread

\*/

public class MyThread implements Runnable{

private String name;

@Override

public void run() {

System.out.println(name);

} public MyThread(String name) { this.name = name ; }

public static void main(String[] args) {

try{

for(int i=1;i<=5;i++){

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

Thread t = new Thread(m);

t.start();

}

}catch(Exception ex){

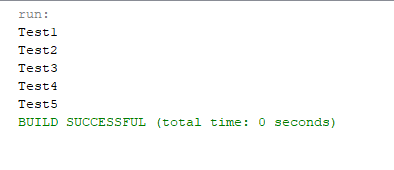
System.out.println("");

}

}

}

**Output & Display:**



**Pseudo Code:**

Package thread;

import java.io.\*;

import java.net.\*;

import java.nio.CharBuffer;

public class Myrunable implements Runnable{

private Socket s;

public Myrunable(Socket s) {

this.s = s;

}

@Override

public void run() {

try {

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

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

dos.writeUTF(dis.readUTF());

s.close();

}

catch (Exception e)

{

e.printStackTrace();

}

}

public static void main(String[] args) {

try{

ServerSocket ss = new ServerSocket(4000);

System.out.println("Server Running....::");

while(true)

{

Socket s = ss.accept();

Myrunable st = new Myrunable(s);

Thread t = new Thread (st);

t.start();

}

}catch(Exception ex){

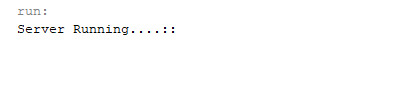
System.out.println(""+ex);

}

}

}

**Output & Display:**



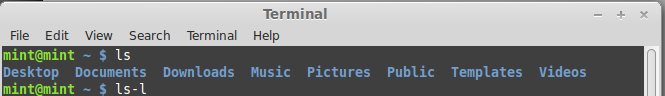
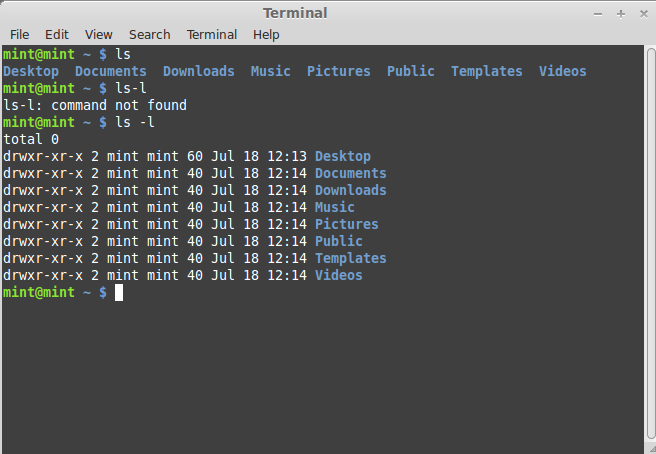
**WEEK # 04**

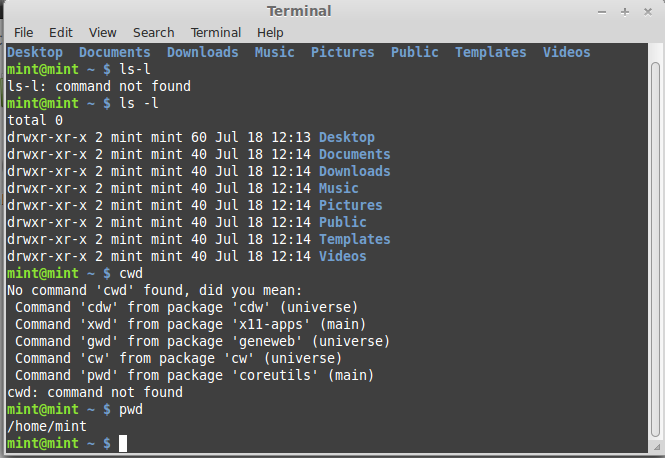
**OPERATING SYSTEM (LAB FILE)**

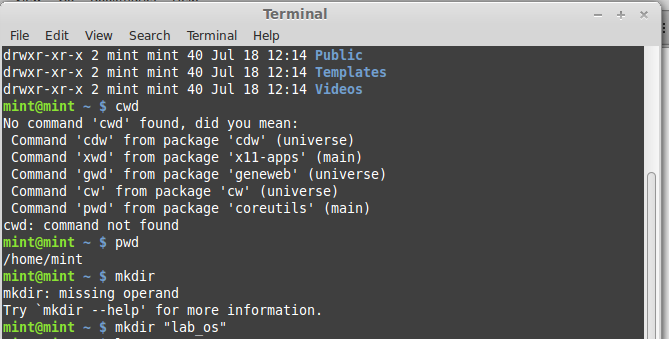
**To study and execute basic Linux commands on a terminal**

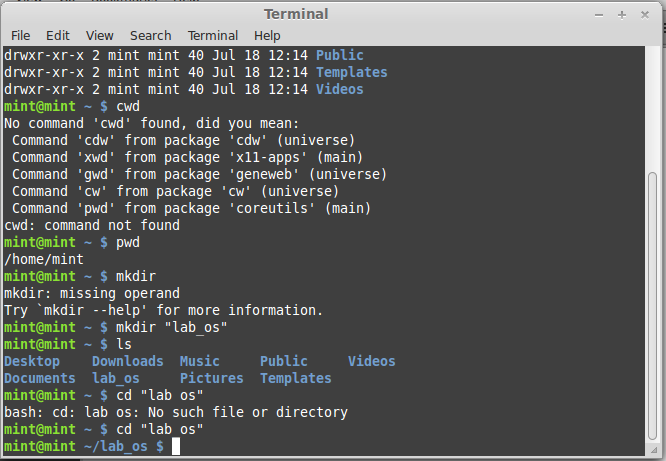
Linux is a Unix-like and mostly POSIX-compliant computer operating system (OS) assembled under the model of free and open-source software development and distribution. In this lab, we will work on Ubuntu, one of the flavors of Linux. For this purpose, we will use virtualization environment.

**Lab Tasks:**

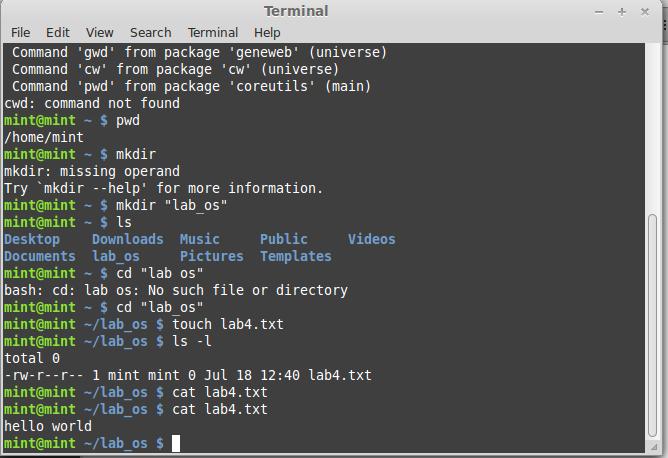
1. Using ls command find out the contents of current directory
2. What are the permissions for normal user, group and world for each file
3. Find out the name of current working directory
4. Create a new folder named “lab os” using the mkdir command
5. Switch to the directory “lab os”
6. Create a file in the directory named “lab4.txt” using touch command
7. List down the contents of file using cat command. Try using “more” and “less” option
8. Find out the space consumed by directory using “du” command.
9. Copy the file to parent directory using cp command
10. Remove the file using rm command
11. Remove the directory using rmdir command
12. Check the free space on disk using df command
13. Change the password of the user using passwd command
14. Switch to super user, using the command “su
15. Using the history command, list down the commands run on the terminal window
16. Using ls command find out the contents of current directory
17. What are the permissions for normal user, group and world for each file
18. Find out the name of current working directory



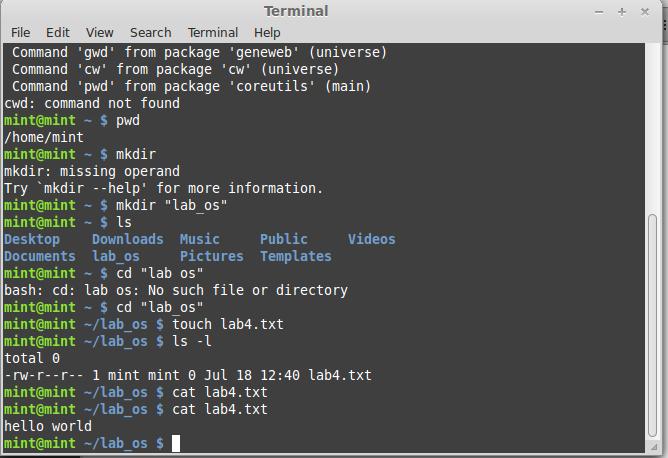
1. Create a new folder named “lab os” using the mkdir command
2. Switch to the directory “lab os”

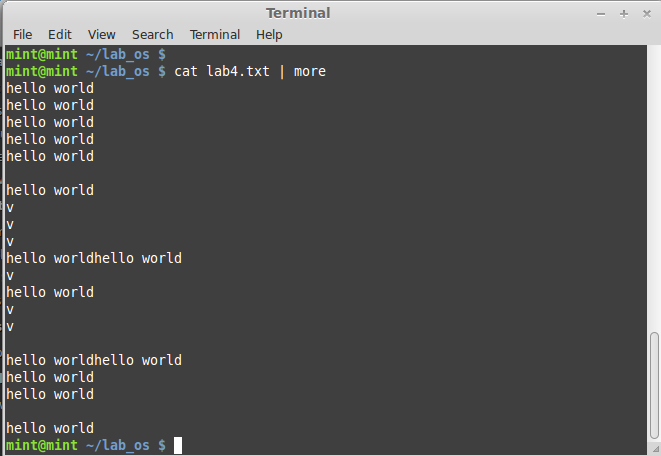


1. Create a file in the directory named “lab4.txt” using touch command

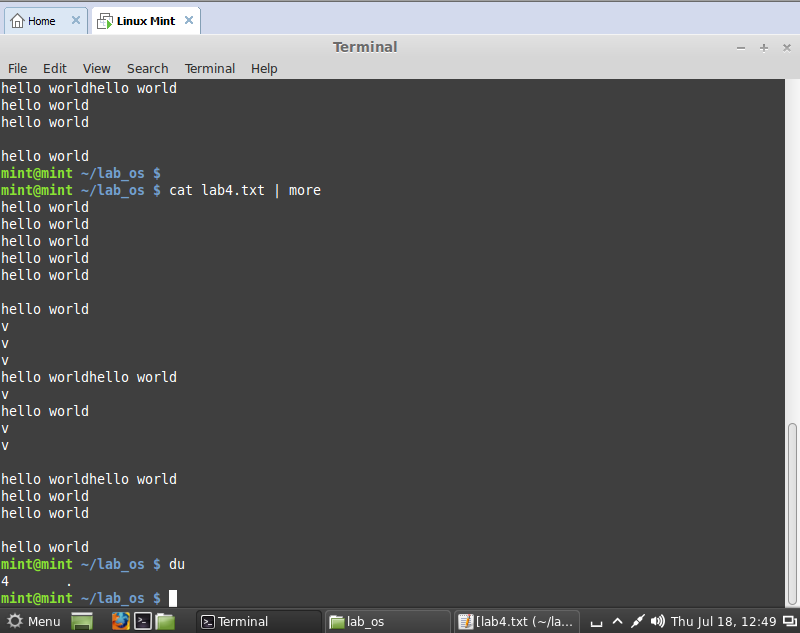


1. List down the contents of file using cat command. Try using “more” and “less” option.

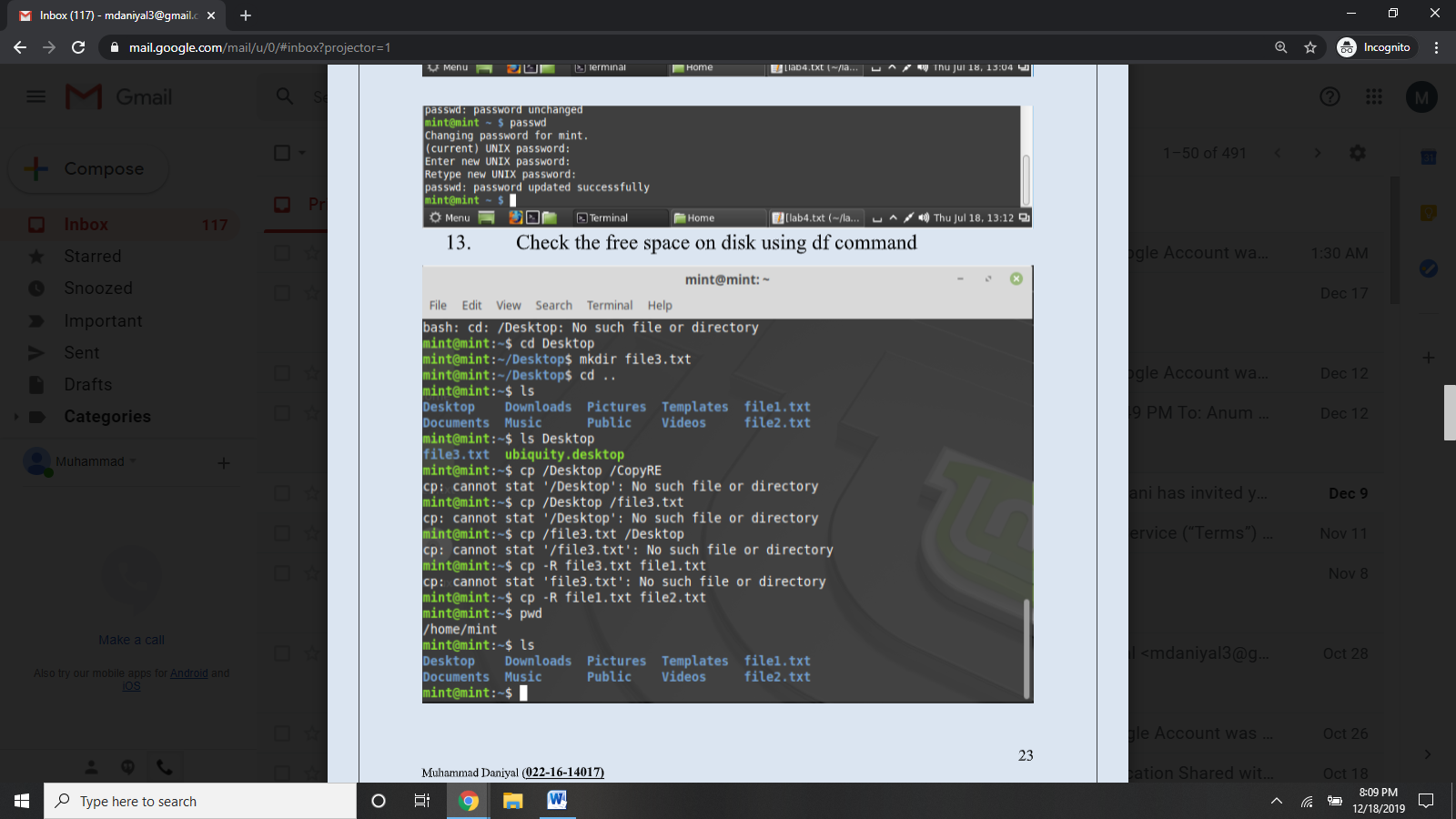




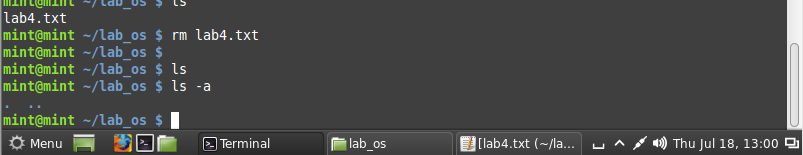
1. Find out the space consumed by directory using “du” command



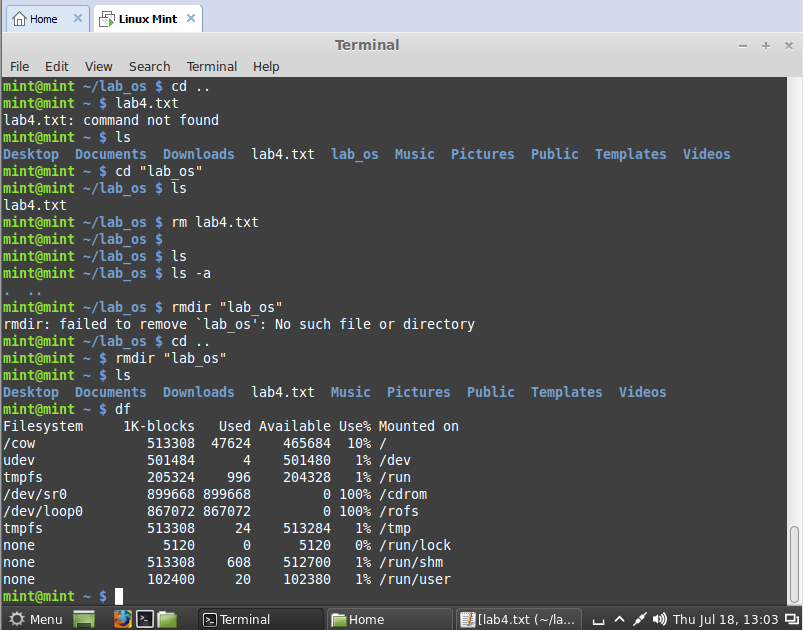
1. Copy the file to parent directory using cp command

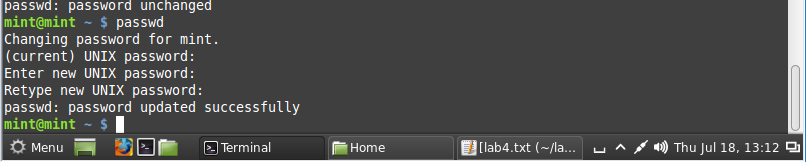
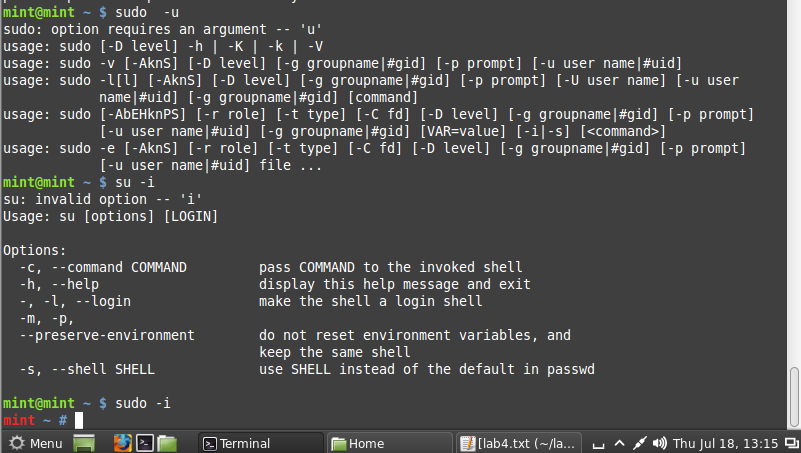


1. Remove the file using rm command.



1. Remove the directory using rmdir command



1. Check the free space on disk using df command
2. Change the password of the user using password command.
3. Switch to super user, using the command “su”

**WEEK # 05**

**OPERATING SYSTEM (LAB FILE)**

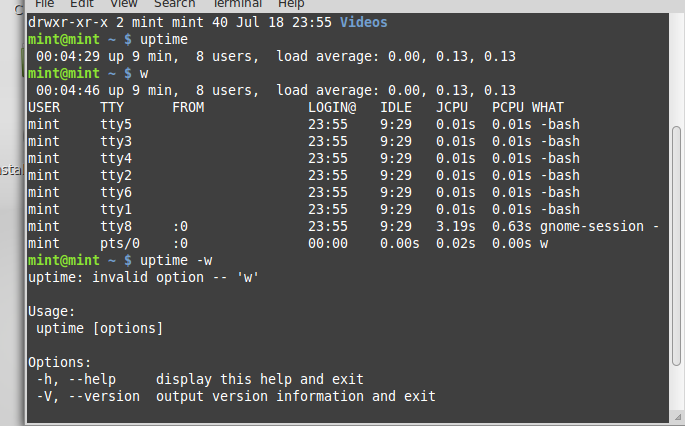
**To study and execute system administration commands on a terminal**

**Instructions:**

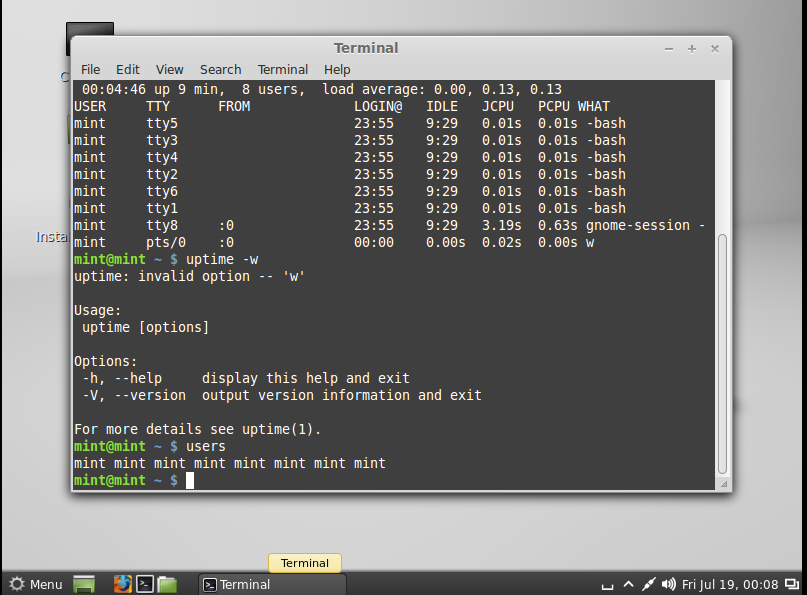
Linux comprises a set of commands for basic system administration. In this lab, we will study these commands.

**Lab Tasks:**

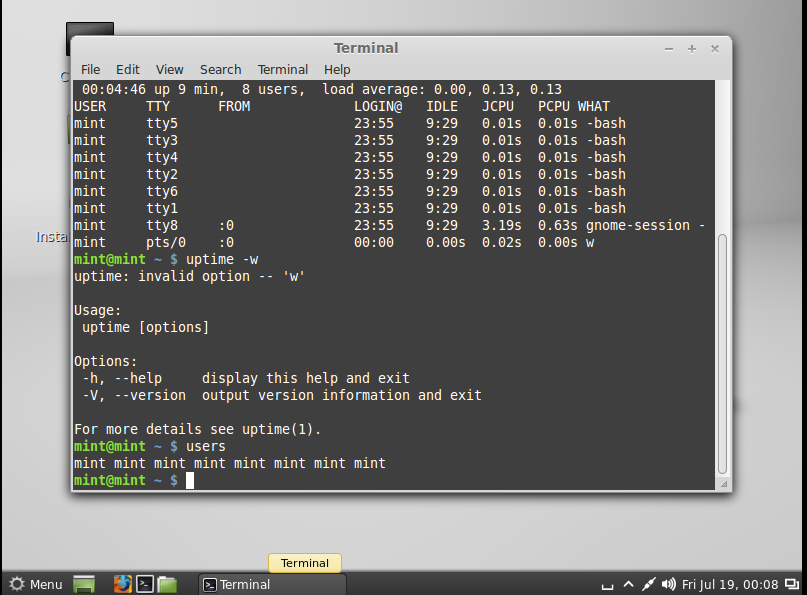
1. Using the ‘uptime' command, since how long your system is running and the number of users that are currently logged in.



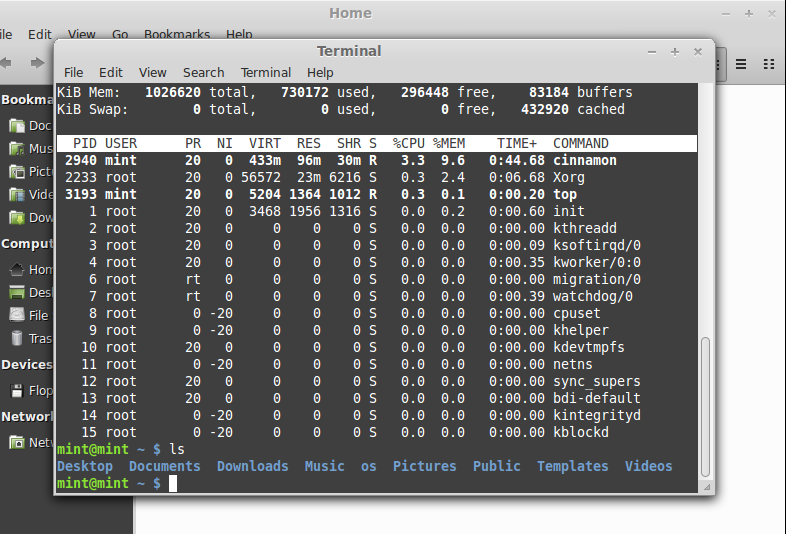
1. Using the ‘w’, display the users currently logged in and their process along-with load averages.



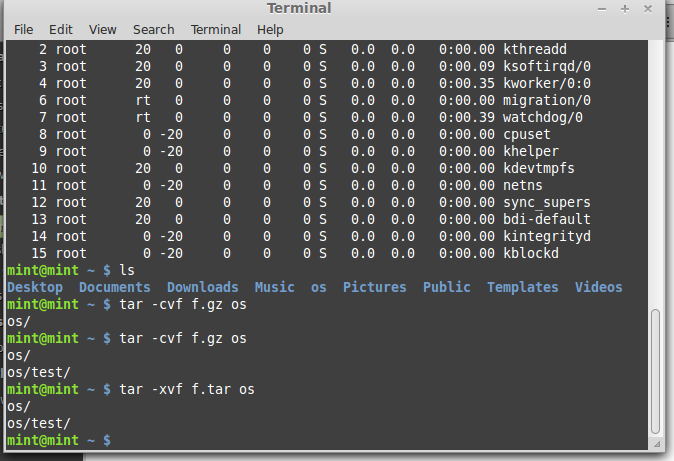
1. Using the ‘users’ command, display the currently logged in users.



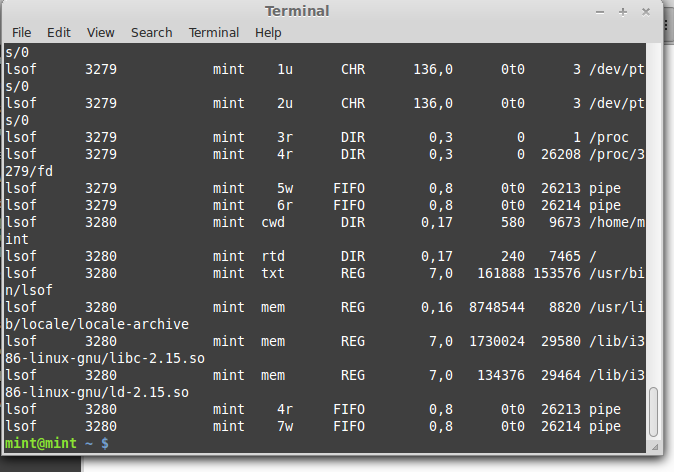
1. Using the ‘top’ command, display processor activity of your system and also displays tasks managed by kernel in real-time.



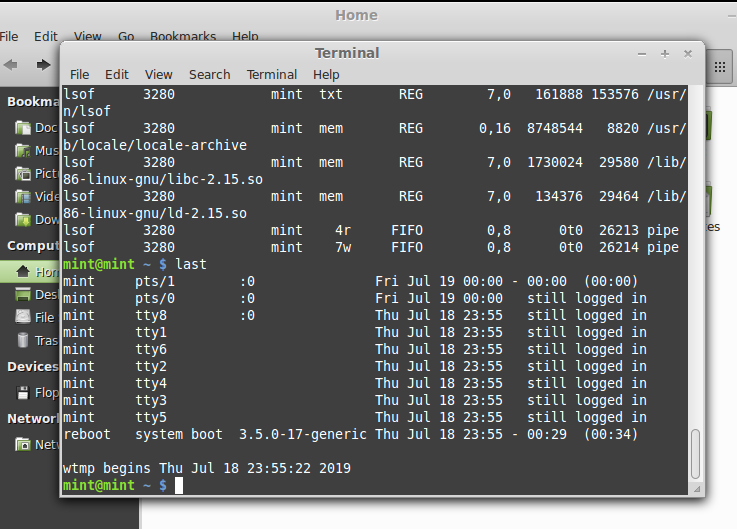
1. Using ‘tar’ command, compress your home directory in Linux.



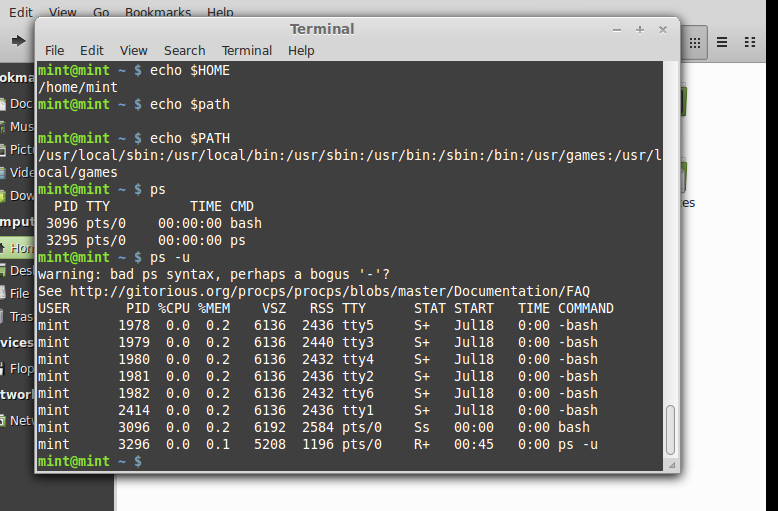
1. ‘lsof’ command to list all open files



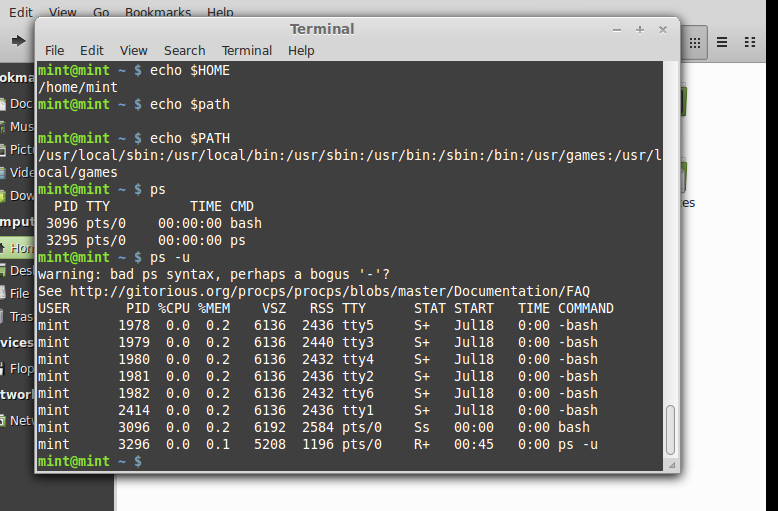
1. Using the ‘last’ command, watch activity of ‘mint’ user in the system



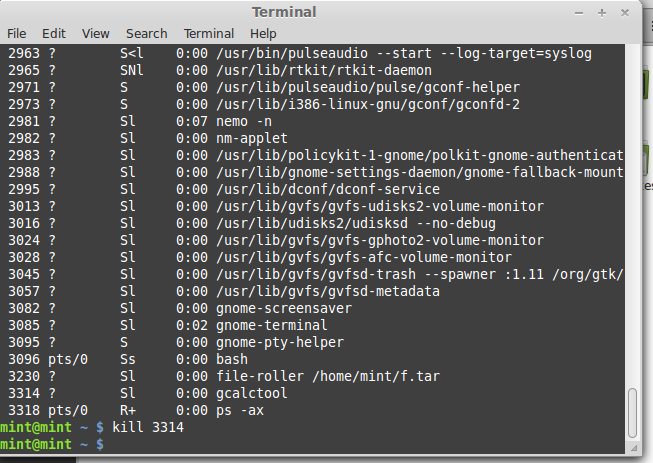
1. Using the ‘env’ command, lists all the environment variables of your system. Use ‘echo’ command to print values of $HOME and $PATH



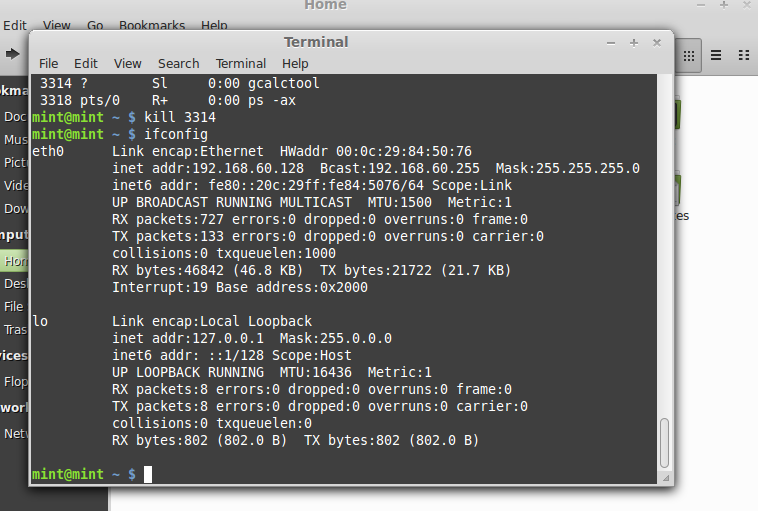
1. The ‘ps’ command displays about processes running in the system. Try option –ax, -u.



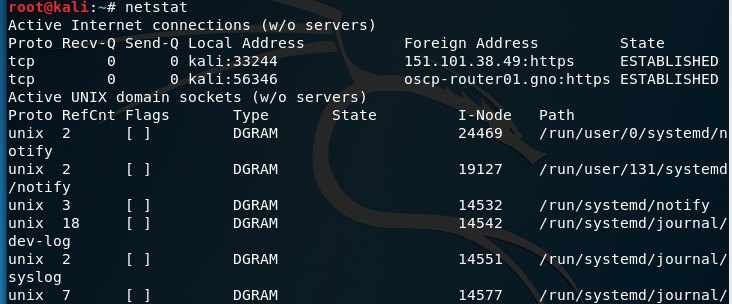
1. The ‘kill’ command can be used to terminate process. Using this command terminate some processes of your system



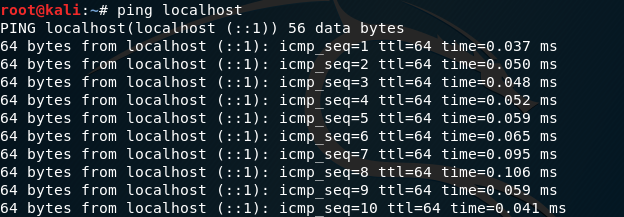
1. ‘ifconfig’ command is used to show the configuration of internet on LINUX. Use this command to find IP and MAC address of your computer



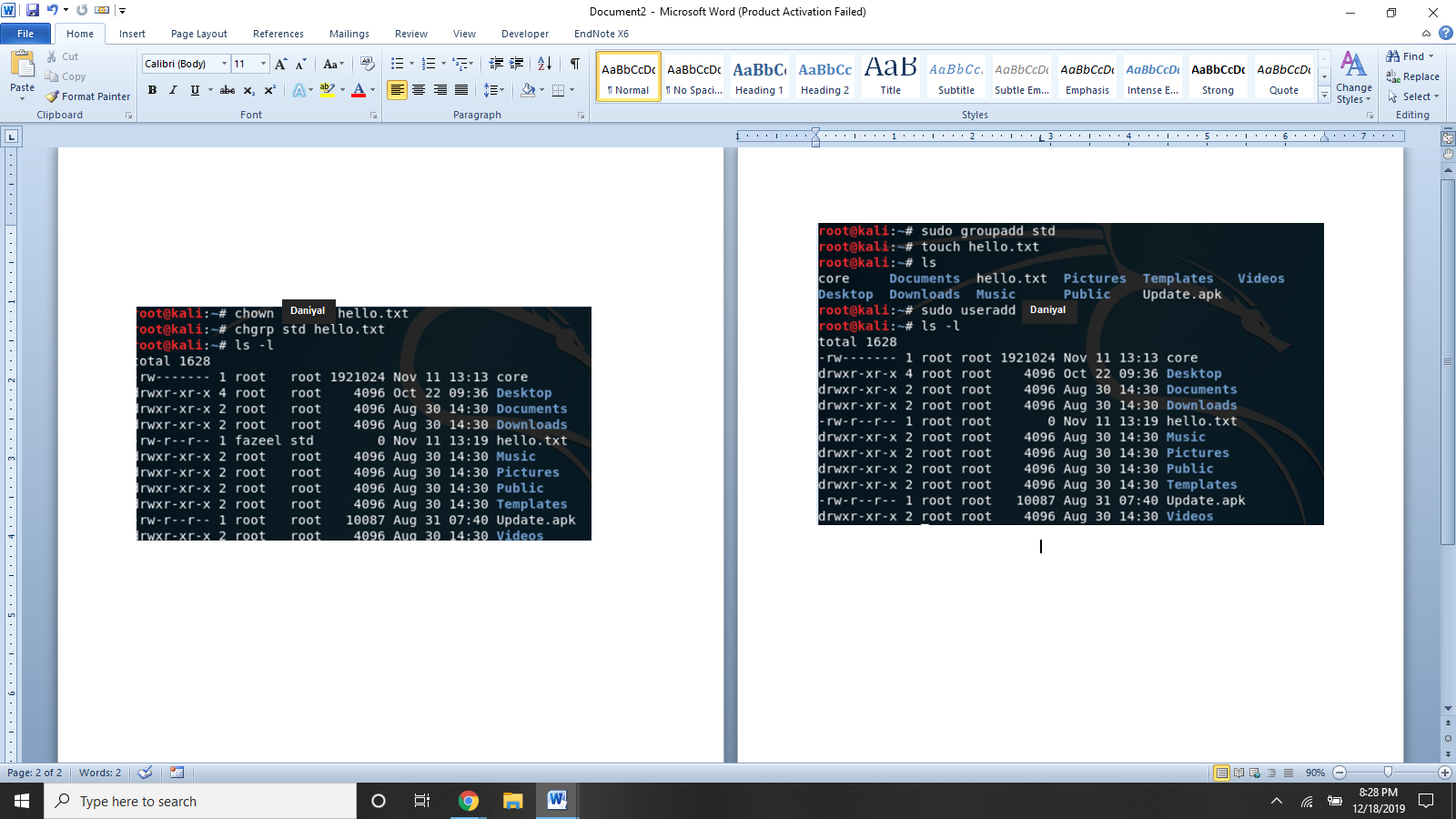
1. Using the ‘netstat’ command, show the status of your network



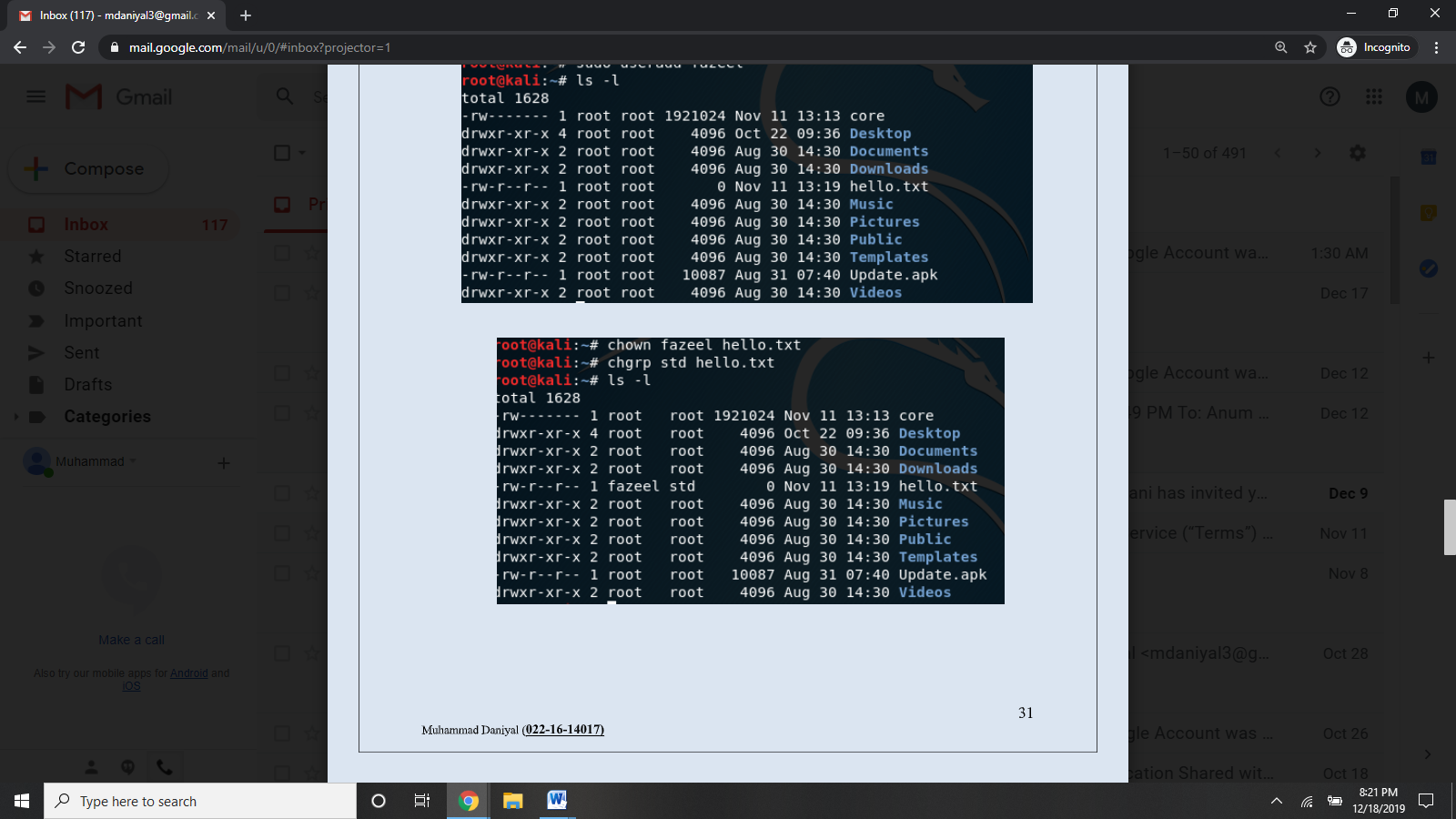
1. Using the ping command, to ping your localhost



1. Create a group named ‘student’ using groupadd



1. Create a file named ‘hello.txt’



**WEEK # 06**

**OPERATING SYSTEM (LAB FILE)**

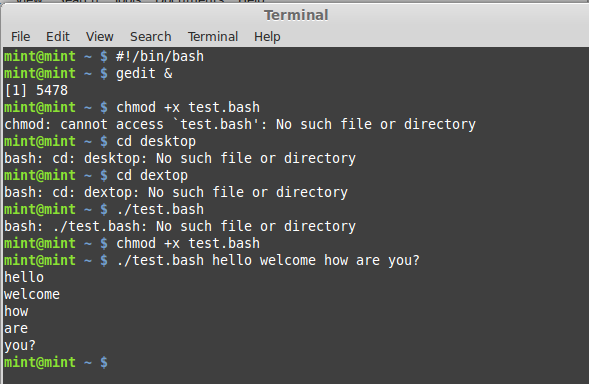
**To study and implement shell programming in Linux**

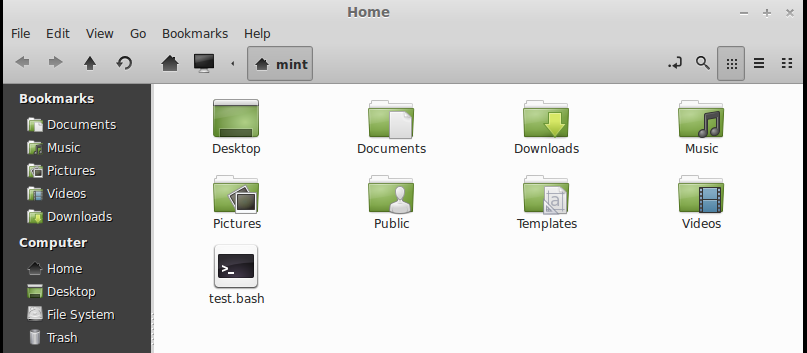
**Instructions:**

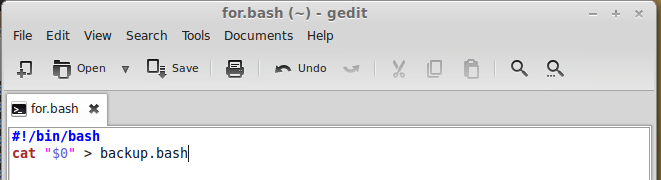
1. A shell script is a computer program designed to be run by the Unix shell, a command-line interpreter
2. The various dialects of shell scripts are considered to be scripting languages.
3. Typical operations performed by shell scripts include file manipulation, program execution, and printing text.

**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

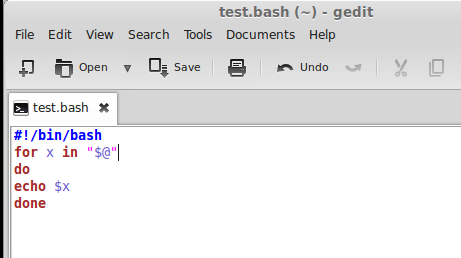


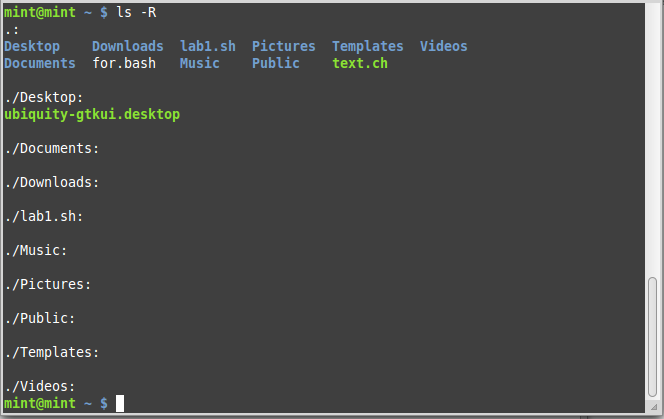


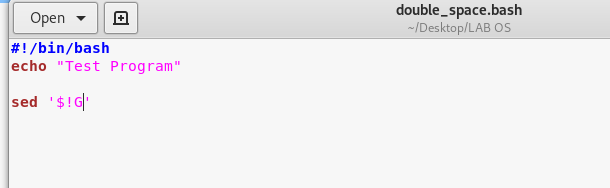
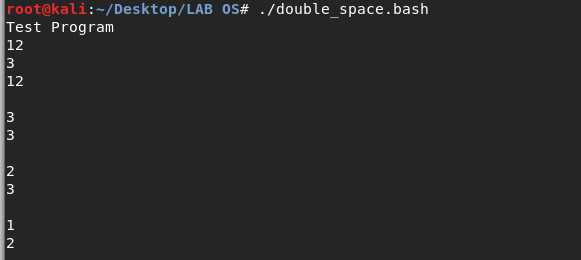


1. Write a script that echoes itself to stdout, but backwards.

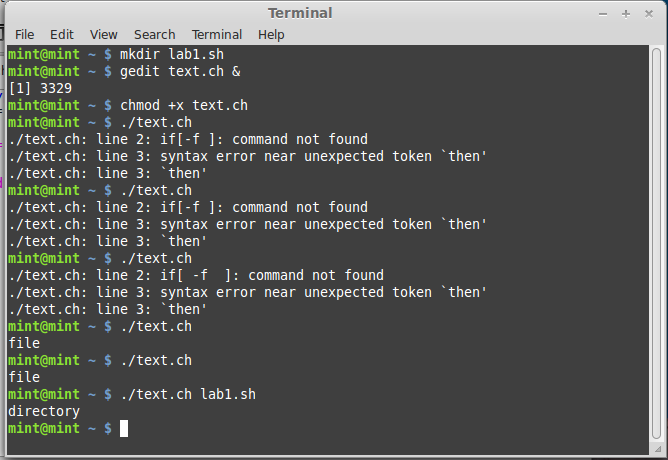
Hint: Use the tac command

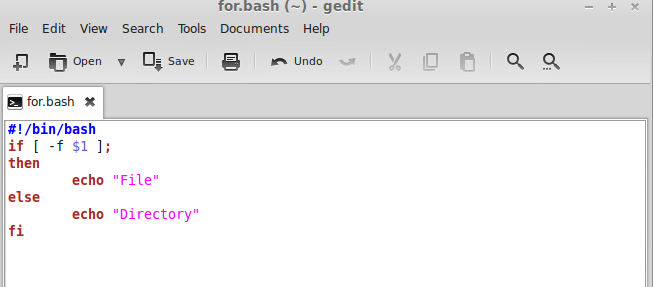


1. Perform a recursive directory listing on the user's home directory and save the information to a file.
2. Write a script that reads each line of a target file, then writes the line back to stdout, but with an extra blank line following. This has the effect of double-spacing the file.

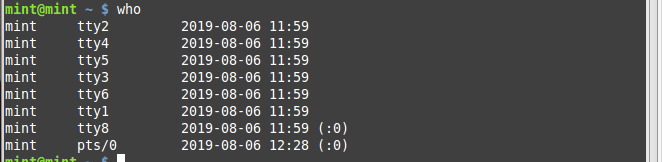


1. Write a shell script that takes a command –line argument and reports on whether it is directory, a file

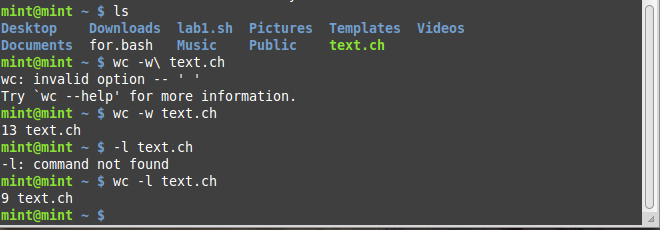


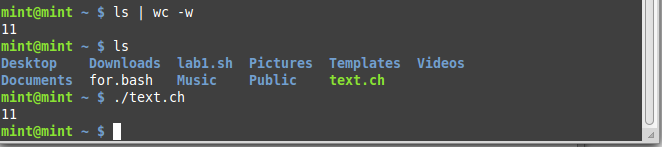


1. Write a shell script program to display list of user currently logged in.



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





**WEEK # 07**

**OPERATING SYSTEM (LAB FILE)**

**To study and implement information security techniques in Linux**

In this lab, we will explore the basic information security tools available in Linux. There are a number of tools available in Linux. This lab only covers nmap, whois and wireshark tool

**Lab Tasks**

1. Download and install the three tools nmap, whois and wireshark tool on Linux. What command did you use to install?
2. Now run the nmap tool on <http://iqra.edu.pk>. Capture the output.
3. Provide a commentary on the output
4. Run the whois tool on <http://iqra.edu.pk>. Capture the output
5. Provide a commentary on the output
6. With the wireshark tool capturing the interface data, browse http://iqra .edu.pk
7. Capture the HTTP protocol message
8. Provide the commentary on the above captured message

**WEEK # 08**

**OPERATING SYSTEM (LAB FILE)**

**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) {

}

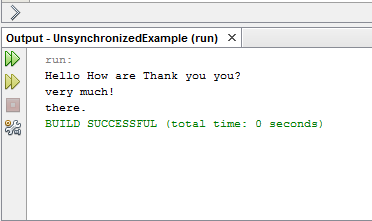
System.out.println(str2);

}

}

**Lab Tasks:**

1. What output do you see? Explain the output.
2. Now use the synchronized methods to display the desired result.
3. Now use the synchronized keyword on an object to synchronize.

**Output & Display:**

**Task 2:**

**Code:**

public class TwoStrings {

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

System.out.print(str1);

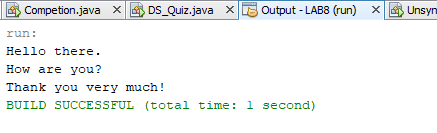
try {

Thread.sleep(500);

} catch (InterruptedException ie) { }

System.out.println(str2);}

}

**Output & Display:**

**Task #3**

**Pseudo Code in Java:**

public class PrintStringsThread implements Runnable {

Thread thread;

String str1, str2;

static TwoStrings Printer=new TwoStrings();

PrintStringsThread (String str1, String str2) {

this.str1 = str1;

this.str2 = str2;

thread = new Thread(this);

thread.start();

}

@Override

public void run() {

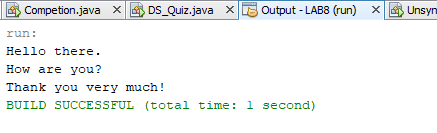
synchronized(Printer){

Printer.print(str1, str2);

}

}

}

**Output & Display:**

**WEEK # 09**

**OPERATING SYSTEM (LAB FILE)**

**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.

**Pseudo Code in Java:**

package shortest.job;

import java.util.\*;

/\*\*

\*

\* @author daniyal

\*/

public class ShortestJob {

/\*\*

\* @param args the command line arguments

\*/

static Vector v = new Vector();

public static void main(String[] args) {

// TODO code application logic here

System.out.println("Enter number of processes: ");

Scanner s = new Scanner(System.in);

int n = s.nextInt();

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

System.out.println("Process "+ i + " Burst Time: ");

process p = new process();

p.burst\_time = s.nextInt();

v.add(p); }

v.sort(new Comparator() {

public int compare(Object a, Object b) {

return ((process)a).burst\_time - ((process)b).burst\_time;

}

}); int c = 0 ;

for(int i=0;i<v.size();i++) {

((process)v.get(i)).wait\_time = c;

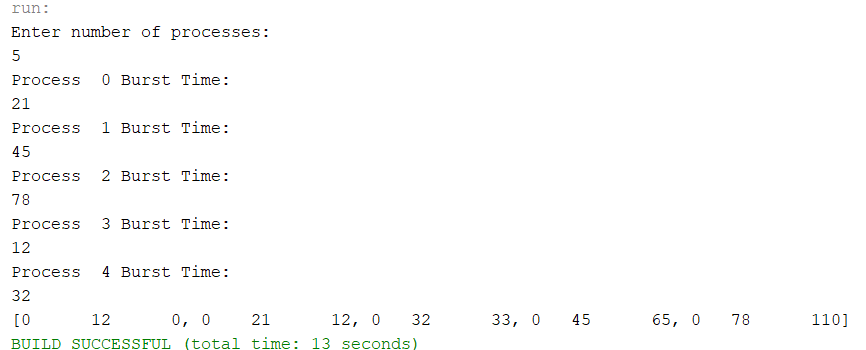
c+=((process)v.get(i)).burst\_time;

}

System.out.println(v);

}

}

**Output & Display:**

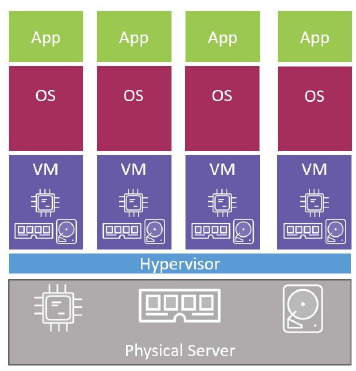
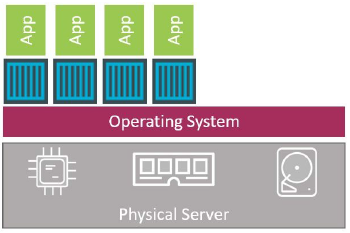
**WEEK # 10**

**OPERATING SYSTEM (LAB FILE)**

**To study and implement containers and dockers.**

**Instructions:**

A container image is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, settings.

The VM Approach The container approach

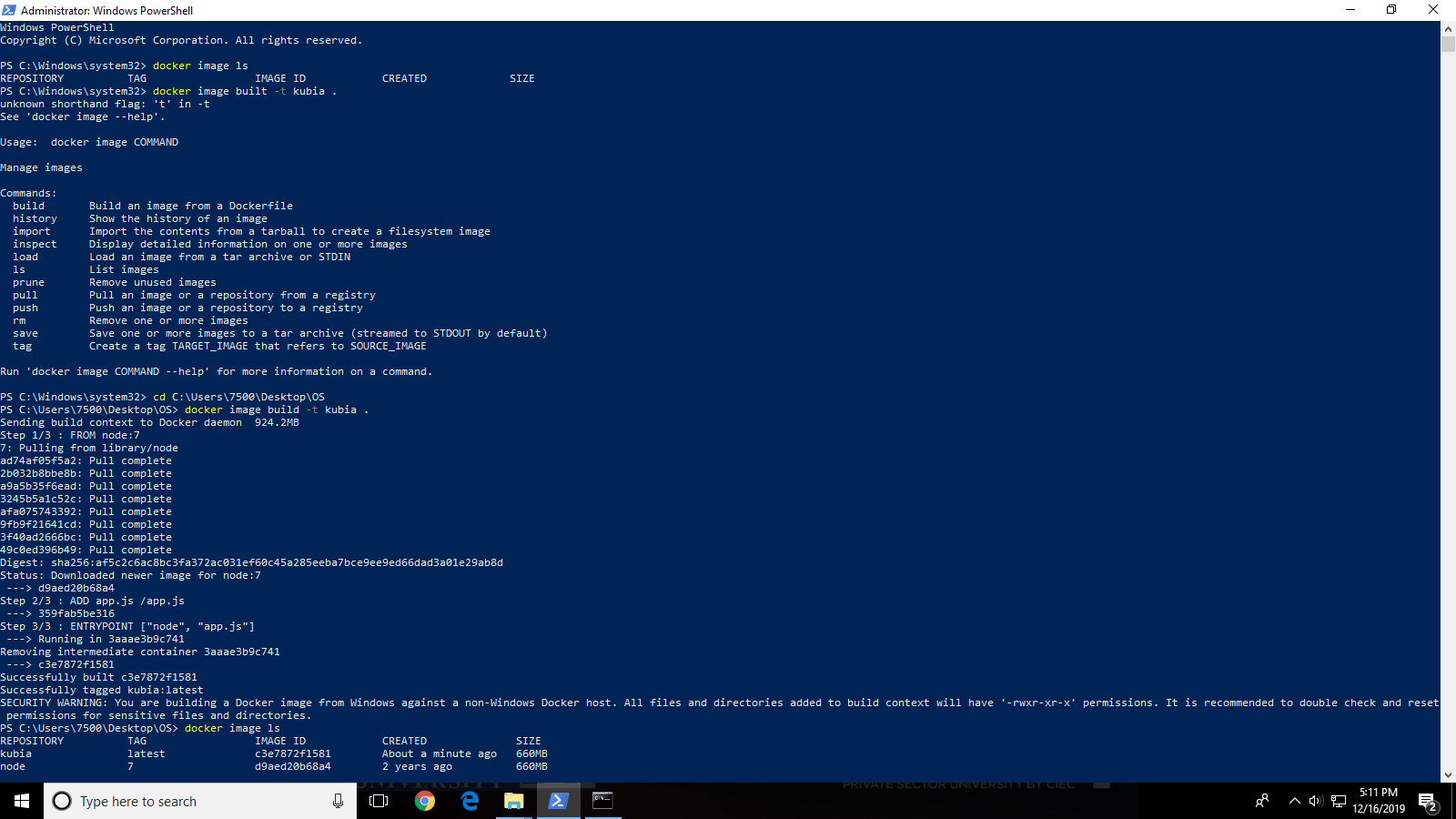
Figure 10.1: The difference between container and VM approach

**Lab Tasks**

***Docker commands***

1. Download and install the VMware toolbox for windows
2. List the images available on your system with command “docker image ls”.
3. Goto dockerhub.com and browse for repositories of alpine, python, tensorflow
4. Pull the repository of alpine using docker pull alpine:latet
5. Now list down the images again
6. Start a container using command docker container run -it alpine:latest /bin/bash
7. Run a ps command from inside of the container to list all running processes
8. Press Ctrl-PQ to exit the container without terminating it
9. You can see all running containers on your system using the docker container ls
10. Attach to the running container again with command “docker container exec -it vigilant\_borg bash” where vigilant\_borg is the name of yhour cotinaer
11. Press Ctl-PQ again to exit the container
12. Stop the running container with command “docker container stop vigilant\_borg”.
13. Remove the container using command docker container rm vigilant\_borg

***Building docker images***

1. Clone a repository using command “git clone <https://github.com/nigelpoulton/psweb.git>”
2. Change your directory using “cd psweb”
3. List the contents of the Dockerfile using “cat Dockerfile”
4. Build the docker image using “docker image build –t os:latest .”
5. Check to make sure that the new os:latest image exists on your host
6. Now run a container with the newly create image “docker container run -d --name web1 -p 8080:8080 os:latest”
7. Open a web browser and navigate to the DNS name or IP address of the host that you are running the container from and point it to port 8080Well done. You’ve taken an application and containerized it (built a Docker image from it).

**WEEK # 11**

**OPERATING SYSTEM (LAB FILE)**

**Lab 11**

**To study and setup a Kubernetes cluster**

Kubernetes abstracts away the hardware infrastructure and exposes your whole datacenter as a single enormous computational resource. It allows you to deploy and run your software components without having to know about the actual servers underneath. When deploying a multi-component application through Kubernetes, it selects a server for each component, deploys it, and enables it to easily find and communicate with all the other components of your application.

**Lab Tasks:**

Setup

1. Install docker desktop from <http://www.dockerhub.com>
2. Next, download the kubectl from <https://kubernetes.io/>

Build an image:

1. Create a sample nodes.js server file (app.js)

const http = require('http');

const os = require('os');

console.log("Kubia server starting...");

var handler = function(request, response) { console.log("Received request from " + request.connection.remoteAddress);

response.writeHead(200);

response.end("You've hit " + os.hostname() + "\n"); };

var www = http.createServer(handler);

www.listen(8080);

1. Now crate a docker file:

FROM node:7

ADD app.js /app.js

ENTRYPOINT ["node", "app.js"]

1. Now run the following command to build an image:

docker build -t kubia .

1. Enable kubernetes on your system (it is available in System tray)
2. Switch the context of kubectl by running:

kubectl config use-context docker-for-desktop

Getting details about nodes and cluster

1. Run the following command to check if your cluster is working: kubectl cluster-info
2. List all the nodes in your cluster: kubectl get nodes
3. To see more detailed information about an object, you can use the kubectl describe command

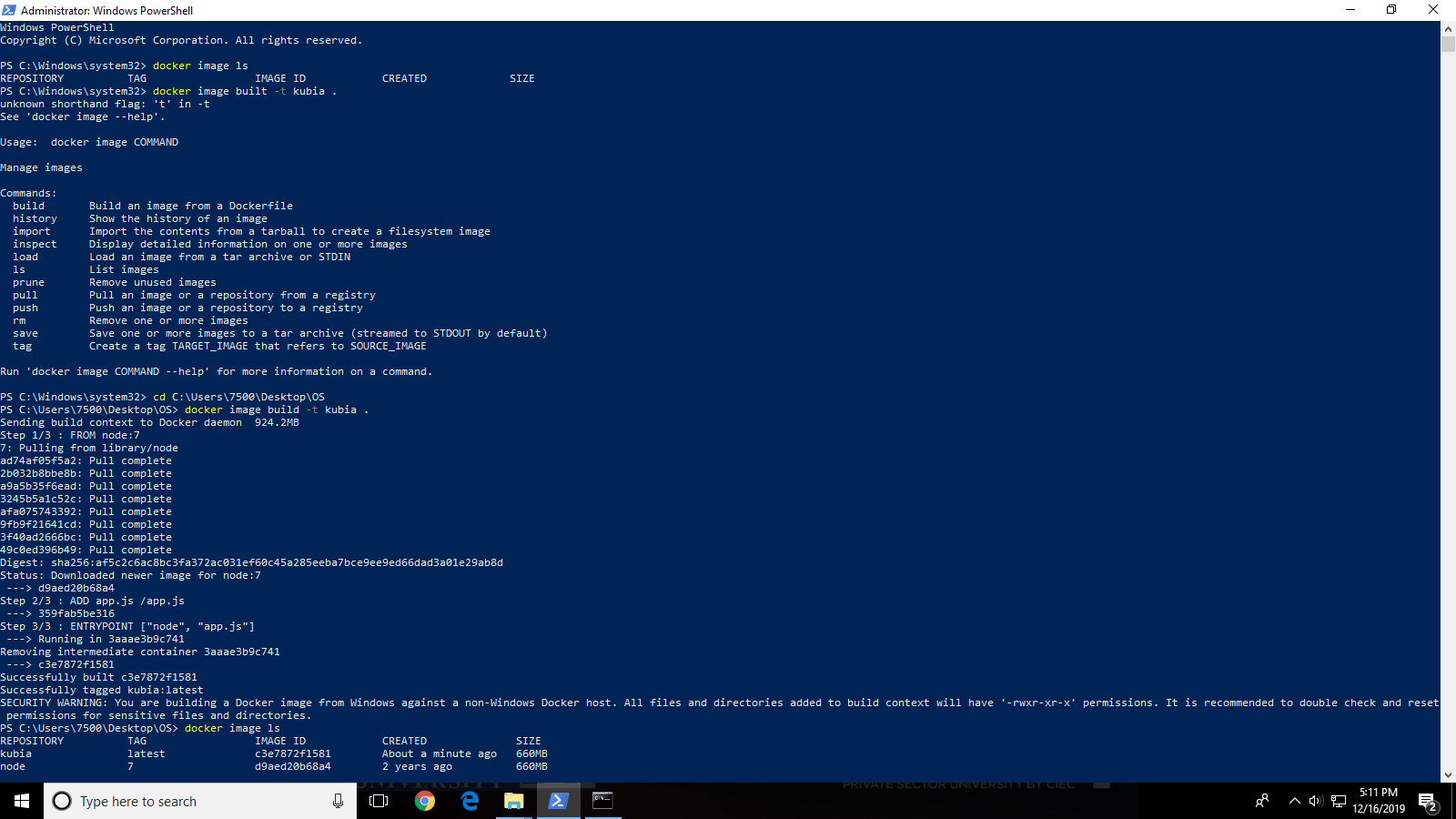
Deploying an application

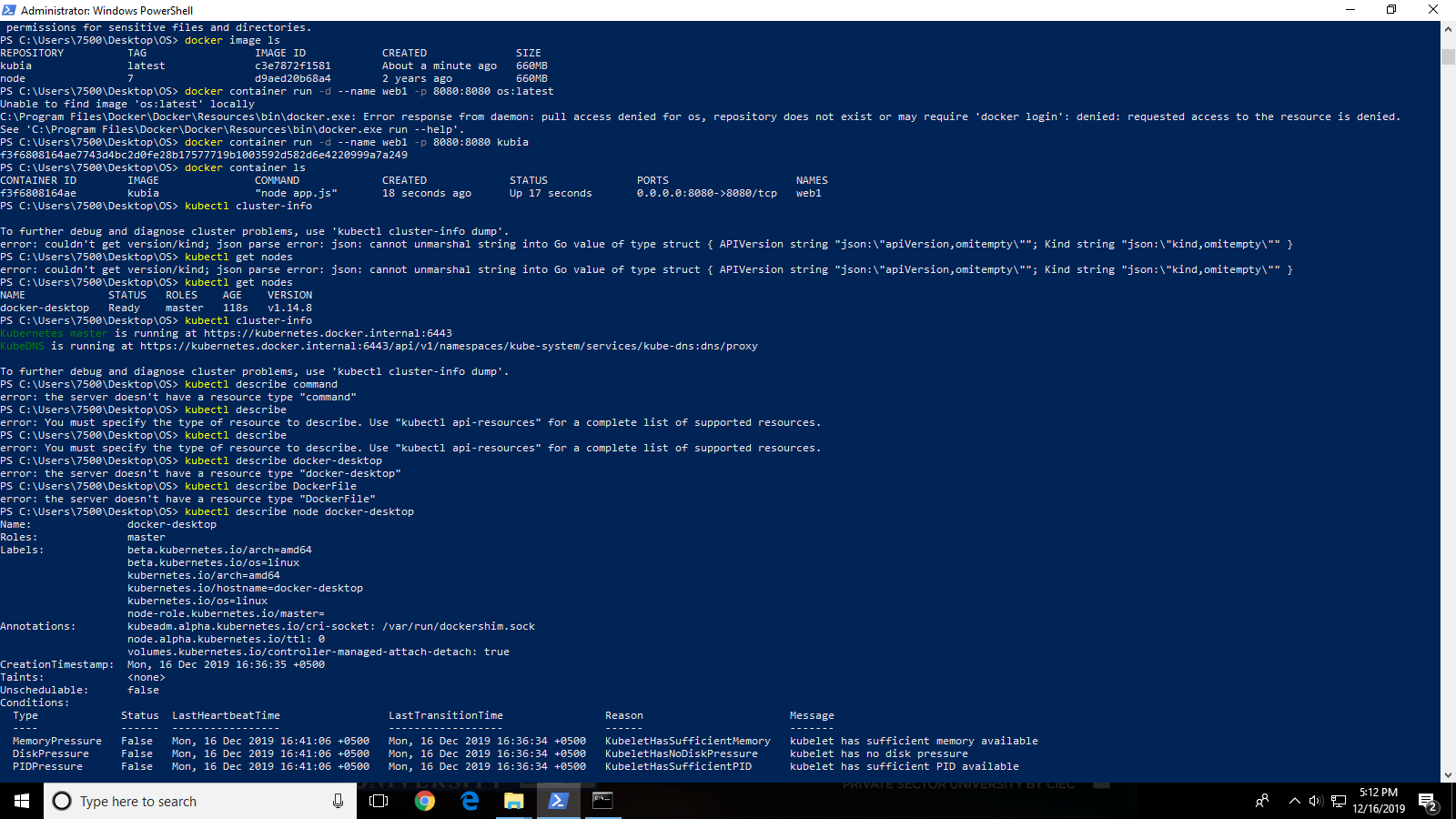
1. Deploy your application using the following command: kubectl run kubia --image=luksa/kubia --port=8080 --generator=run/v1
2. List all the pods using following command: kubectl get pods
3. Expose your replication controller using following command: kubectl expose rc kubia --type=LoadBalancer --name kubia-http
4. Now, list the services using: kubectl get services
5. Access your application from the listed URL.

Replication controllers

1. Your pod is managed by a Replication Controller. See the replication controller with the following command: kubectl get replication controllers
2. Scale up the number of replicas of your pod with following command: kubectl scale rc kubia --replicas=3
3. Now run the following command: kubectl get rc
4. Run the following command to see number of pods: kubectl get pods

For more details, run: kubectl get pods -o wide

****Output & Display:**

**

