Practical List

**Instructions:**

ISO Practical Format: Aim, Software/Hardware Required, Knowledge Required, Theory, Algorithm/Flow chart, Program, Input and Output, Advantages and Disadvantages, Conclusion, Questions and Answers.

**Practical : 1**

Unix Architecture

Here is a basic block diagram of a Unix system −



The main concept that unites all the versions of Unix is the following four basics −

* **Kernel** − The kernel is the heart of the operating system. It interacts with the hardware and most of the tasks like memory management, task scheduling and file management.
* **Shell** − The shell is the utility that processes your requests. When you type in a command at your terminal, the shell interprets the command and calls the program that you want. The shell uses standard syntax for all commands. C Shell, Bourne Shell and Korn Shell are the most famous shells which are available with most of the Unix variants.
* **Commands and Utilities** − There are various commands and utilities which you can make use of in your day to day activities. **cp**, **mv**, **cat**and **grep**, etc. are few examples of commands and utilities. There are over 250 standard commands plus numerous others provided through 3rd party software. All the commands come along with various options.
* **Files and Directories** − All the data of Unix is organized into files. All files are then organized into directories. These directories are further organized into a tree-like structure called the **filesystem**.

**Types of OS :**

Operating systems are there from the very first computer generation and they keep evolving with time. In this chapter, we will discuss some of the important types of operating systems which are most commonly used.

## Batch operating system

The users of a batch operating system do not interact with the computer directly. Each user prepares his job on an off-line device like punch cards and submits it to the computer operator. To speed up processing, jobs with similar needs are batched together and run as a group. The programmers leave their programs with the operator and the operator then sorts the programs with similar requirements into batches.

The problems with Batch Systems are as follows −

* Lack of interaction between the user and the job.
* CPU is often idle, because the speed of the mechanical I/O devices is slower than the CPU.
* Difficult to provide the desired priority.

## Time-sharing operating systems

Time-sharing is a technique which enables many people, located at various terminals, to use a particular computer system at the same time. Time-sharing or multitasking is a logical extension of multiprogramming. Processor's time which is shared among multiple users simultaneously is termed as time-sharing.

The main difference between Multiprogrammed Batch Systems and Time-Sharing Systems is that in case of Multiprogrammed batch systems, the objective is to maximize processor use, whereas in Time-Sharing Systems, the objective is to minimize response time.

Multiple jobs are executed by the CPU by switching between them, but the switches occur so frequently. Thus, the user can receive an immediate response. For example, in a transaction processing, the processor executes each user program in a short burst or quantum of computation. That is, if nusers are present, then each user can get a time quantum. When the user submits the command, the response time is in few seconds at most.

The operating system uses CPU scheduling and multiprogramming to provide each user with a small portion of a time. Computer systems that were designed primarily as batch systems have been modified to time-sharing systems.

Advantages of Timesharing operating systems are as follows −

* Provides the advantage of quick response.
* Avoids duplication of software.
* Reduces CPU idle time.

Disadvantages of Time-sharing operating systems are as follows −

* Problem of reliability.
* Question of security and integrity of user programs and data.
* Problem of data communication.

## Distributed operating System

Distributed systems use multiple central processors to serve multiple real-time applications and multiple users. Data processing jobs are distributed among the processors accordingly.

The processors communicate with one another through various communication lines (such as high-speed buses or telephone lines). These are referred as **loosely coupled systems** or distributed systems. Processors in a distributed system may vary in size and function. These processors are referred as sites, nodes, computers, and so on.

The advantages of distributed systems are as follows −

* With resource sharing facility, a user at one site may be able to use the resources available at another.
* Speedup the exchange of data with one another via electronic mail.
* If one site fails in a distributed system, the remaining sites can potentially continue operating.
* Better service to the customers.
* Reduction of the load on the host computer.
* Reduction of delays in data processing.

## Network operating System

A Network Operating System runs on a server and provides the server the capability to manage data, users, groups, security, applications, and other networking functions. The primary purpose of the network operating system is to allow shared file and printer access among multiple computers in a network, typically a local area network (LAN), a private network or to other networks.

Examples of network operating systems include Microsoft Windows Server 2003, Microsoft Windows Server 2008, UNIX, Linux, Mac OS X, Novell NetWare, and BSD.

The advantages of network operating systems are as follows −

* Centralized servers are highly stable.
* Security is server managed.
* Upgrades to new technologies and hardware can be easily integrated into the system.
* Remote access to servers is possible from different locations and types of systems.

The disadvantages of network operating systems are as follows −

* High cost of buying and running a server.
* Dependency on a central location for most operations.
* Regular maintenance and updates are required.

## Real Time operating System

A real-time system is defined as a data processing system in which the time interval required to process and respond to inputs is so small that it controls the environment. The time taken by the system to respond to an input and display of required updated information is termed as the **response time**. So in this method, the response time is very less as compared to online processing.

Real-time systems are used when there are rigid time requirements on the operation of a processor or the flow of data and real-time systems can be used as a control device in a dedicated application. A real-time operating system must have well-defined, fixed time constraints, otherwise the system will fail. For example, Scientific experiments, medical imaging systems, industrial control systems, weapon systems, robots, air traffic control systems, etc.

There are two types of real-time operating systems.

### **Hard real-time systems**

Hard real-time systems guarantee that critical tasks complete on time. In hard real-time systems, secondary storage is limited or missing and the data is stored in ROM. In these systems, virtual memory is almost never found.

### **Soft real-time systems**

Soft real-time systems are less restrictive. A critical real-time task gets priority over other tasks and retains the priority until it completes. Soft real-time systems have limited utility than hard real-time systems. For example, multimedia, virtual reality, Advanced Scientific Projects like undersea exploration and planetary rovers, etc.

**Flavors of LINUX :**

[**Debian**](http://beginlinux.com/web/index.php?app=ccp0&ns=prodshow&ref=debian)  
Debian is regarded as the largest Linux distribution to date in terms of users. It is renowned for its stability and quality and its interoperability with other software. It has a very large following and over 28,000 packages you can add to it. It is very stable and newbie friendly. The community is knowledgeable and helpful and most of the questions you may have will have been asked before on their forums. Debian seems somewhat over engineered to me, which isn’t a bad thing. It just means that it is never the latest and greatest, but is very, very stable. Currently on Debian Version 5, Lenny.

[**Ubuntu**](http://beginlinux.com/web/index.php?app=ccp0&ns=prodshow&ref=ubuntu)  
Seemingly the most popular distro out in the world right now, since its release five years ago. Some suggest it has as much as 40% of Linux desktops. It is another good quality, stable release that is very newbie friendly. It even has a migration assistant for Windows refugees. This eases the pain a little by helping you in your first steps. Another distro that works with most hardware straight out of the box. Current version 9.04 Jaunty Jackalope.

[**Fedora**](http://beginlinux.com/web/index.php?app=ccp0&ns=prodshow&ref=fedora)  
This is a distribution with a long history. Once part of Red Hat, Fedora was launched for the ‘Linux Hobbyist’. It is very secure and has a large number of packages available for it. This one seemed to be less newbie friendly than others, with not so many options for desktop users. If you want to run a web server or something then this would be ideal. Current version is Fedora 10, new version shortly

[**OpenSUSE**](http://beginlinux.com/web/index.php?app=ccp0&ns=prodshow&ref=opensuse)  
This is another distro with a rich history, now owned by Novell. It is a comprehensive package with some good documentation. The following is helpful and informative and there are lots of packages available for it. There is however some controversy over a deal it has with Microsoft, something about use of patents. This doesn’t detract from the usability of the product though. It does have graphical tools for an administrator. Current release is 11.1

[**Mandriva**](http://beginlinux.com/web/index.php?app=ccp0&ns=prodshow&ref=mandriva)  
Mandriva used to be Mandrake, and even as a Windows user back then I knew what that was. It is one of the distributions that offer both free and commercial versions of the software. The commercial version is regarded as one of the best distros for newbies. Although I don’t see that it has much that Ubuntu or Debian doesn’t offer and for free. Current version is Mandriva 2009.1.

[**Slackware**](http://beginlinux.com/web/index.php?app=ccp0&ns=prodshow&ref=slackware)  
This is allegedly the oldest Linux distribution still around today. It is supposed to be the cleanest and most bug free edition available, but seems quite hard core. You definitely need skills to get into Slackware. It lags a little behind other distributions and doesn’t have all the cool bits to make your desktop smart. Current version 12.2

[**FreeBSD**](http://beginlinux.com/web/index.php?app=ccp0&ns=prodshow&ref=freebsd)  
Yes I know…it is not Linux but it is worth mentioning. It is primarily directed a server applications and is rated as one of the most stable. It doesn’t have all the bells and whistles that some of the more popular distros have but if you need something to run a server, you can’t go wrong with this. Current version 7.2.

[**Mint**](http://beginlinux.com/web/index.php?app=ccp0&ns=prodshow&ref=linuxmint)  
This is my personal favorite, and not only for its cool name. It is based on Ubuntu, but is developed mainly ‘on demand’. The programmers have forums and feedback loops which they subscribe to. They listen and try to implement the best suggestions into the distribution. It has many specific tools, and to my mind is the most useable out of the box. Current version is Mint 6 Felicia.

[**PCLinuxOS**](http://beginlinux.com/web/index.php?app=ccp0&ns=prodshow&ref=pclinuxos)  
Despite the not so catchy name this distribution is supposed to be very good for the Linux newcomer. Another distro that works out of the box, natively supporting lots of hardware. The documentation and website are pretty good if a little disorganized. This one also seems to be English only, so if you speak something other, then you had best look elsewhere. Current version is PCLinuxOS 2009.1

**Practical 2**

**Study of Unix Architecture and the following Unix commands with option:**

**---------------------------------------------------------------------------------------------------**

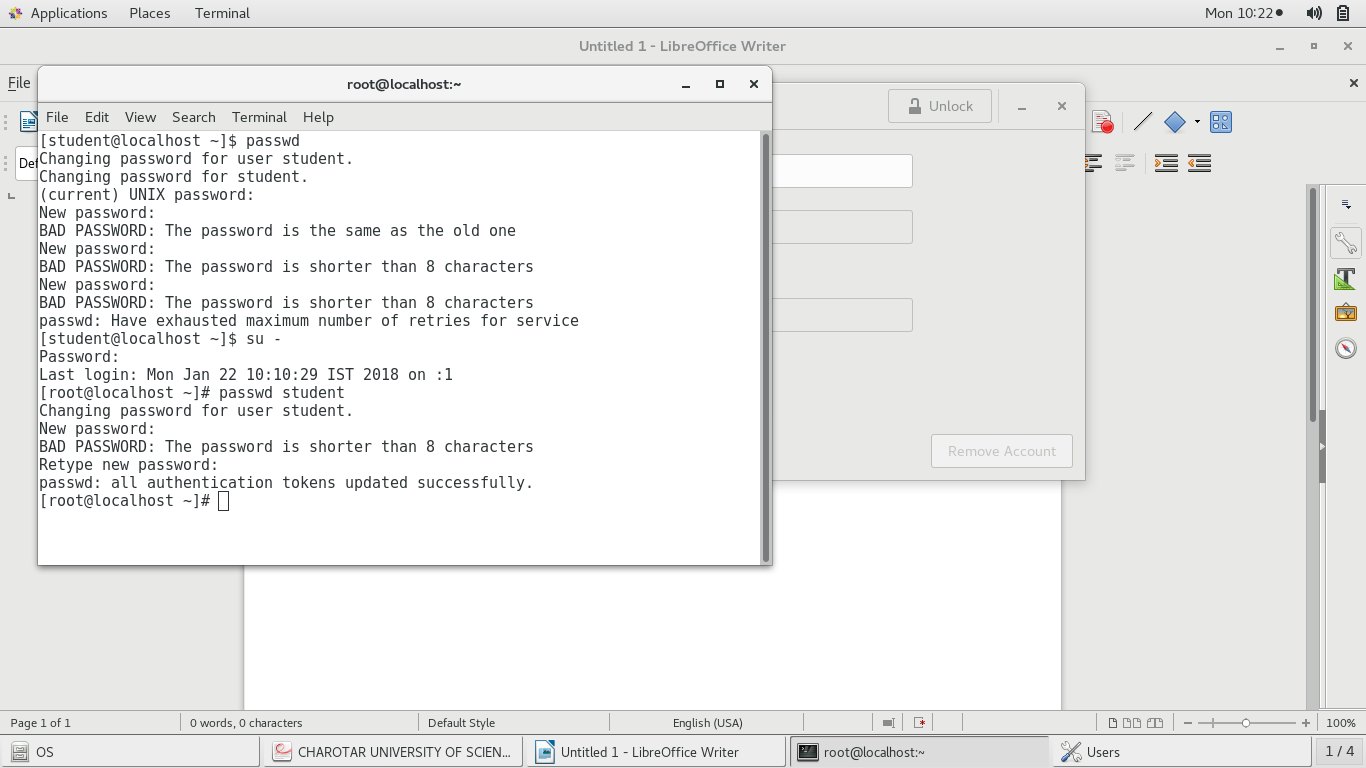
**User Access**

**1).login**

**2).logut**

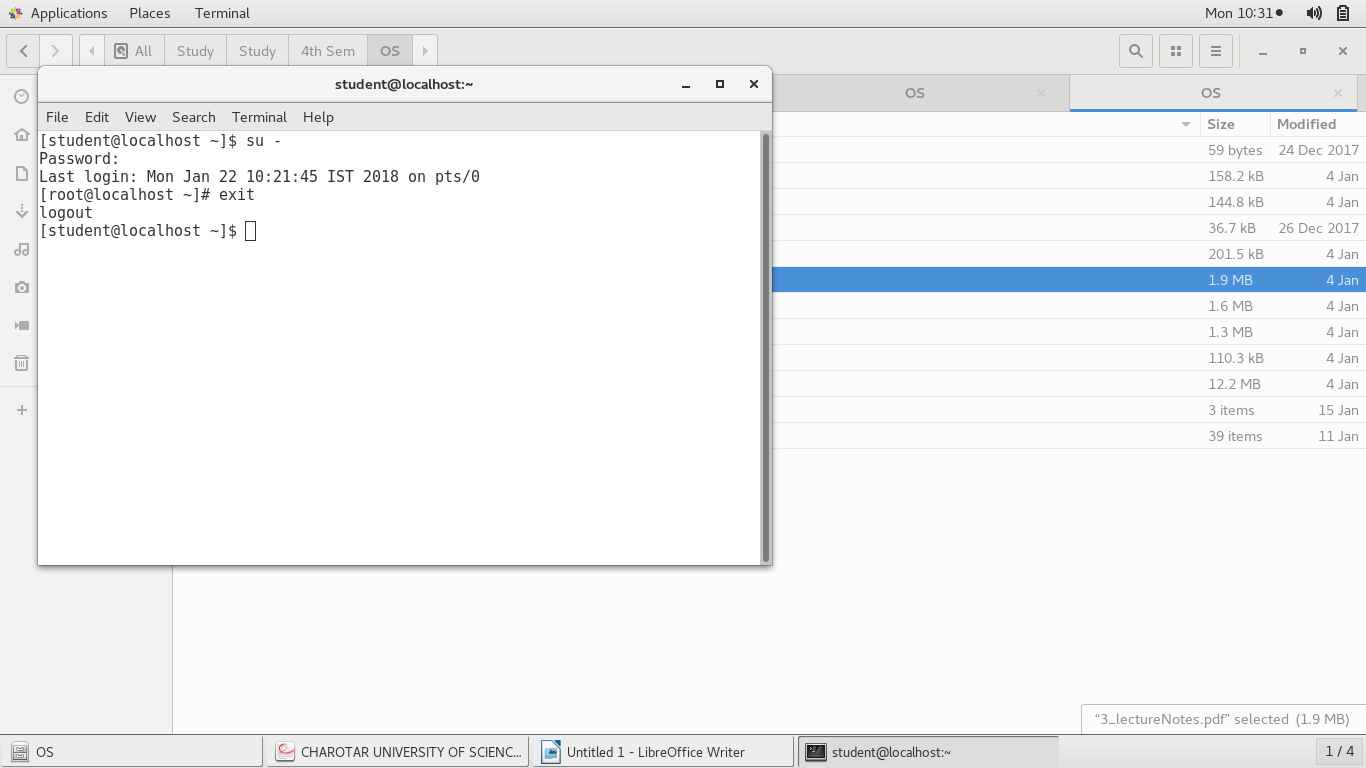
**3). passwd**

changes the password of the current working user and with user name it will change the password for the user.



**4). exit**

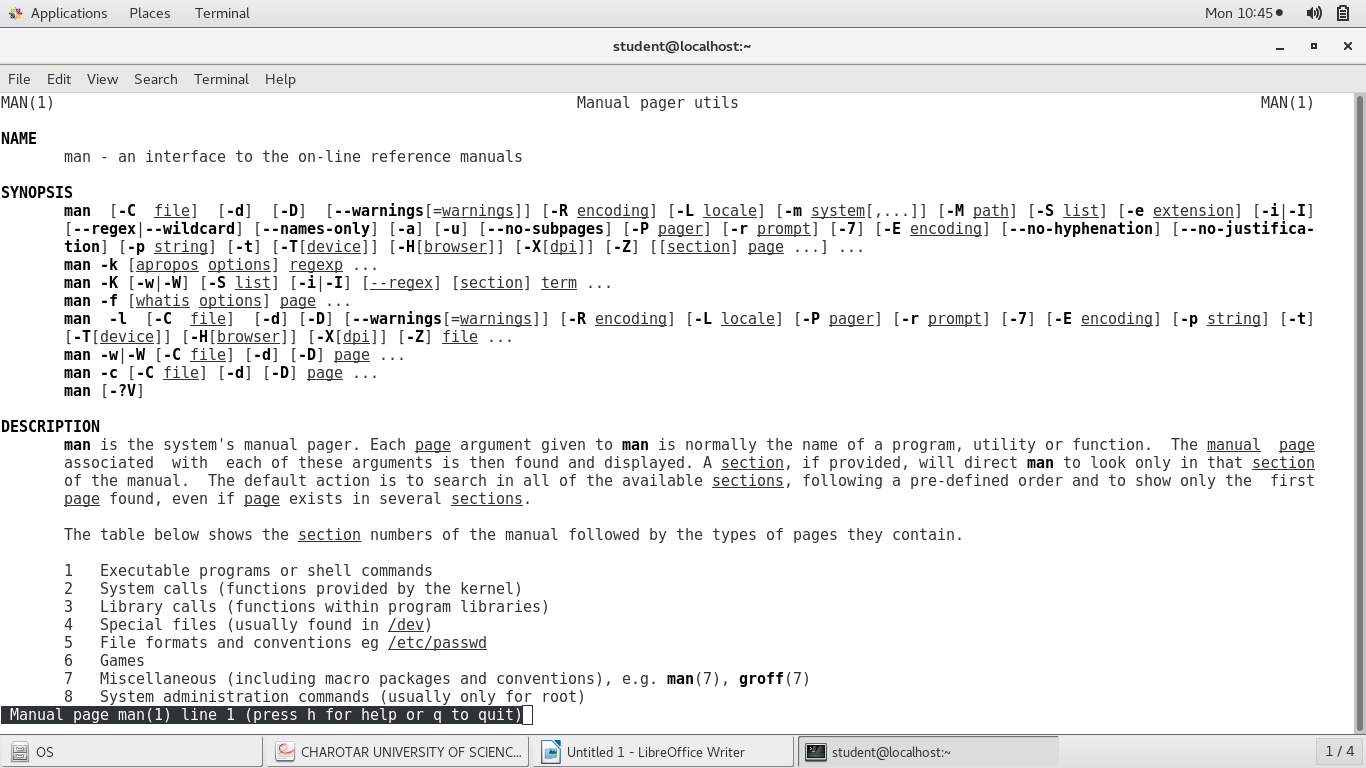
exit from the current session



**Help**

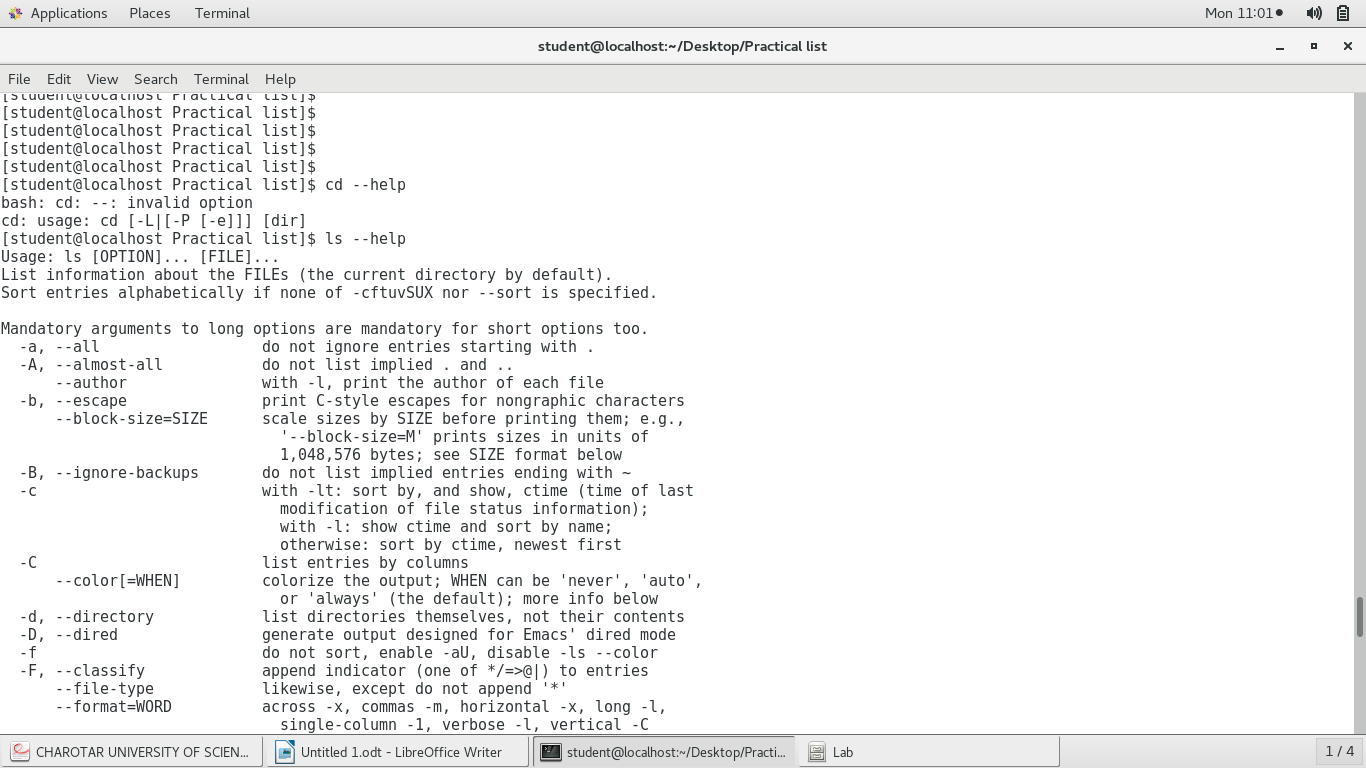
**1). man**

man will show the on-line reference manual for the command which we give in the arguments.



**2).help**

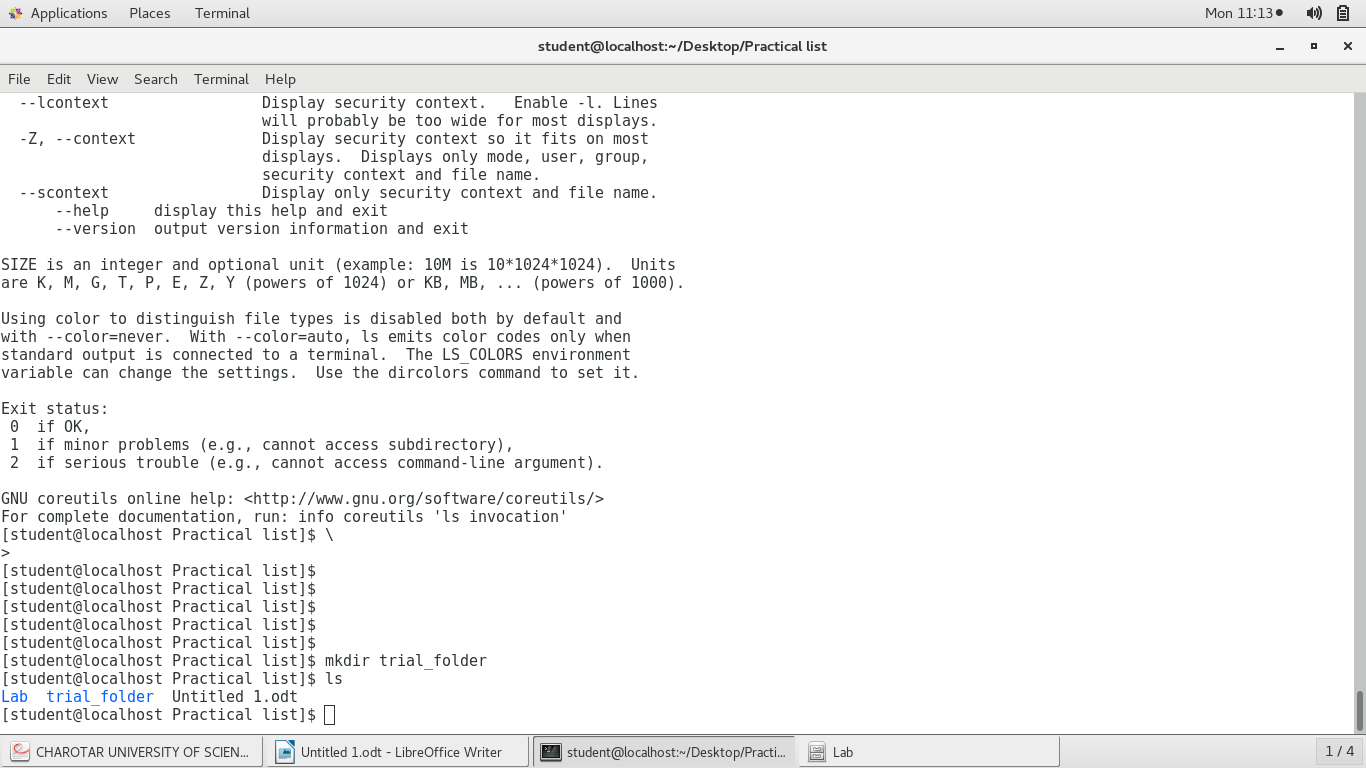
gives the help for the command

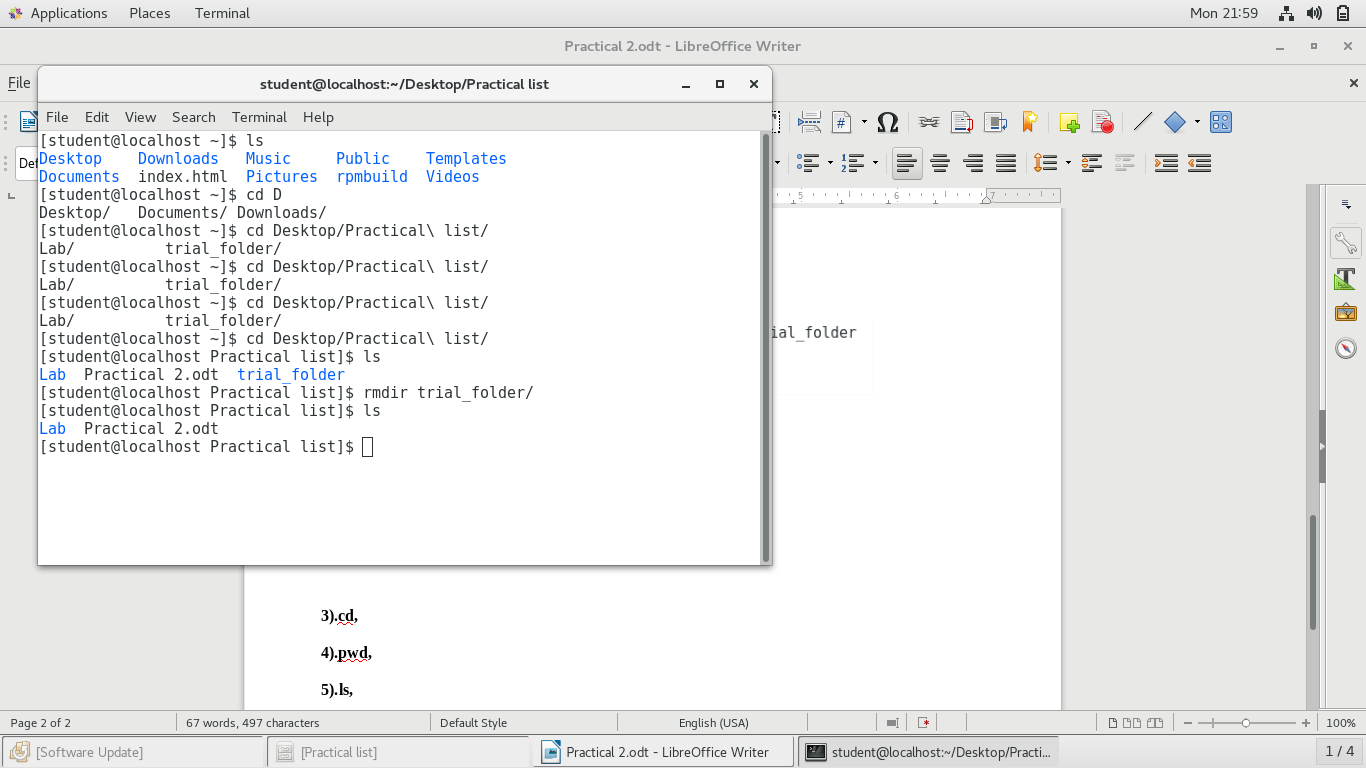


**Directory:**

**1).mkdir,**

Createa new directory with the specified name.

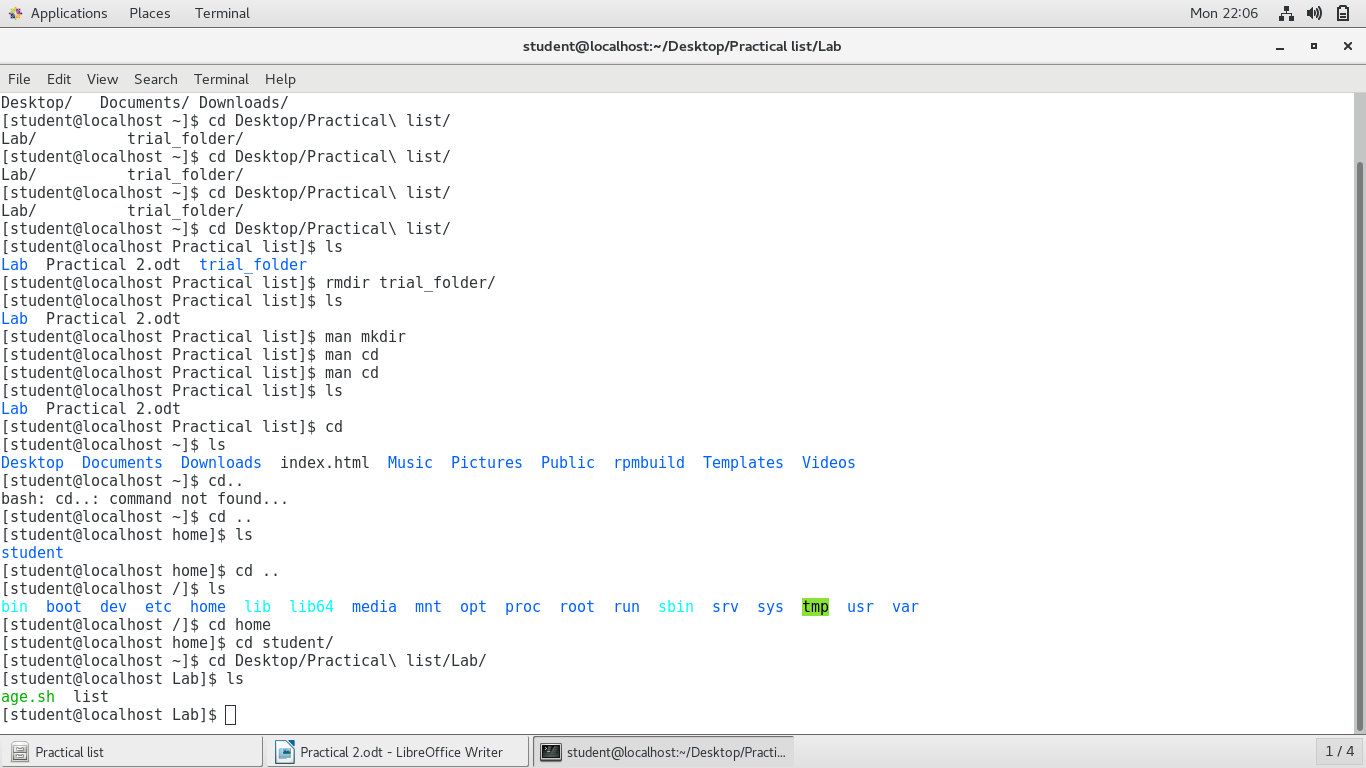


**2).rmdir**

Remove the director if present with the name specified.

**3).cd,**

change the current directory to dir(path specified). The default path is home directory.



**4).pwd,**

specifies in which directory are we working now.

**5).ls** list the content of the present working directory in different way with different arguments.

**6).mv**

move(remane) a file.

**Editor**

**1. vi,**

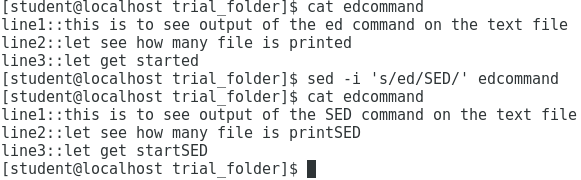
Vim is a text editor that is upwards compatible to Vi. It can be used to edit all kinds of plain text. It is especially useful for editing programs.

**2. gedit,**

gedit is the official text editor of the GNOME desktop environment so gedit opens gnome text editor.

**3. ed,**

line oriented text editor

**4. sed**

**File Handling / Text Processing:**

**1) cp,**

Is used to copy files and directory.

**eg.** cp filename destination

**2) mv**

Is used to move and rename files and directorychmod - change file mode bits

**eg.** mv filename destination

**3) rm**

Is used to remove files and directory

**eg.** rm [filename/foldername]

**4) sort**

Is used for sorting the order for the output of the file.

**eg.** sort –reverse filename

**5) cat**

Displays the text of the file

**eg.** cat filename

**6) pg**

works similarly as more command

**7) lp**

Print the specified file.

**eg.** lp filename

**8) pr**

convert to text file for printing

**eg.** pr filename

**9) file**

Determines the file type of the file.

**eg.** file filename

**10) find**

Search for files in a directory hierarchy

**11) more**

Works similar as cat command but it waits for the user input to scroll down instead of exiting after reaching at the end.

**eg.** more filename

**12) cmp**

Compare the given files byte by byte

**eg.** cmp file1 file2

**13) diff**

Compare the given file line by line

**eg.** diff file1 file2

**14) comm**

Compare two sorted files line by line

**eg.** comm file1 file2

**15) head**

Print first 10 lines of the file by default

**eg.** head filename1 filename2

**16) tail**

print last 10 lines of the file by default

**eg.** tail filename1 filename2

**17) cut**

removes selection from each line of files.

**eg.**

**18) grep**

print the line matching the pattern

**eg.** grep pattern filename

**19) touch**

changes file time-stamp and create a empty file if file does't exist.

**eg.** touch filename

**20) tr**

Translate or delete characters

**eg.** echo “hello there” | tr h T

**21) uniq**

Report or omit constitutive repeated lines

**eg.** cut filename

**Security and Protection**

**1) chmod**

change file mode bits

**2) chown**

changes file owner and group

**3) chgrp**

change group ownership

**4) newgrp**

log into a new group

**Information:**

**1) learn**

**2) man**

an interface to online reference manual.

**3) who**

shows who is logged on.

**4) date**

print or set system date and time.

**5) cal**

shows calendar.

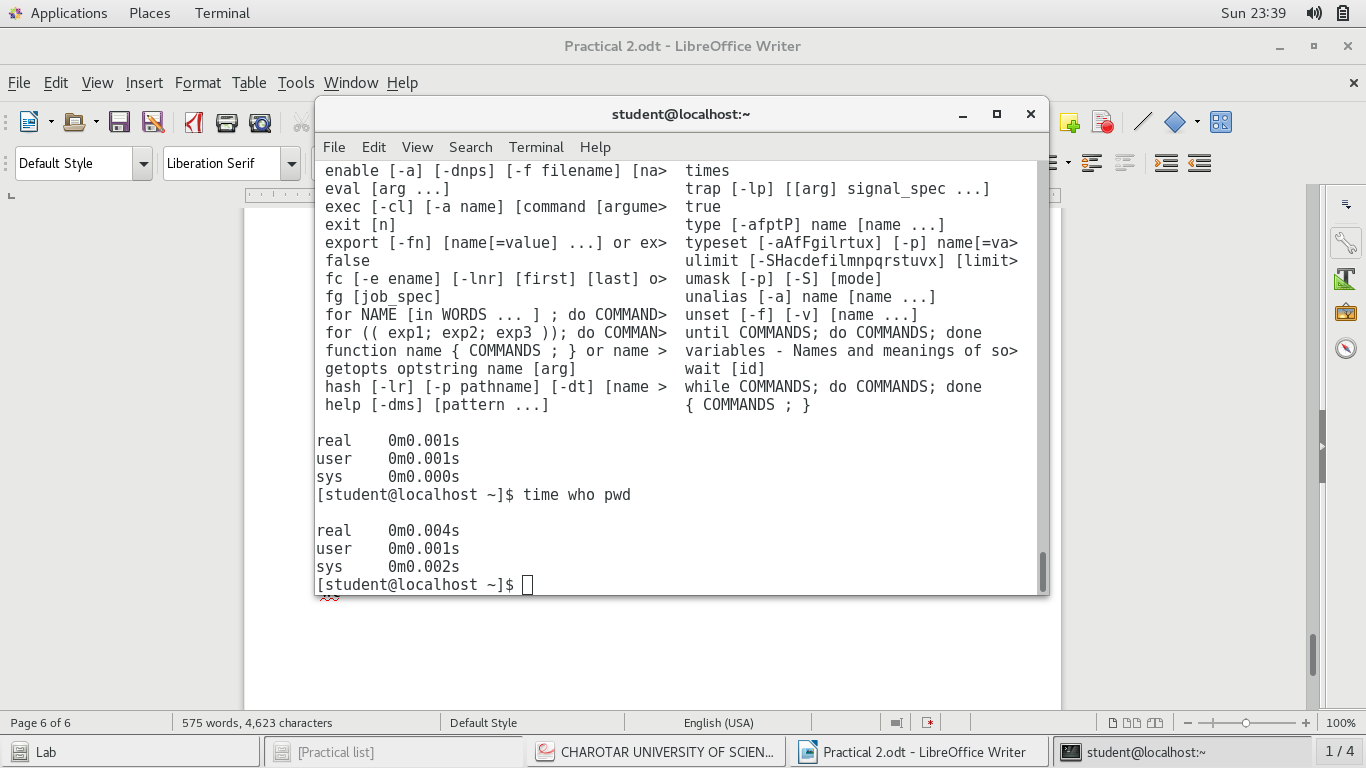
**6) tty**

prints the file name of the terminal connected to the standard input.

**7) calendar**

**8) time**

gives the time taken for the execution of the commad.



**9) bc**

an arbitrary precision calculator to do simple calculations.

**10) whoami**

print effective userid

**11) which**

shows the full path of the shell command.

**12) hostname**

show or sets the system hostname.

**13) history**

show the history of all the executed commands.

**14) wc**

print newline, word, and byte counts for each file

**System Administrator:**

**1) su or root**

login into root account.

**2) date**

same as above date command.

**3) inti 2**

systemd system and service manager

**4) wall**

prints the message on everybody's terminal

**5) shutdown**

shutdown the system.

**6) mkfs**

built a linux file system

**7) mount**

mount a filesystem.

**8) unmount**

**9) dump**

**10) restor**

**11) tar**

GNU `tar' saves many files together into a single tape or disk archive, and can restore individual files from the archive.

**12) adduser**

create a new user or update new user information.

**13) rmuser**

**Terminal**

**1) echo**

displays the line of text.

**2) printf**

formats and print data.

**3) clear**

clears the terminal screen.

**Process**

**1) ps**

report snapshot of current processes.

**2) kill**

terminate a process.

**3) exec-\***

**I/O redirection**

**1) (**<,>,>>**)**

use to store the output in the file with name specified.

**2) Pipe**

used for giving more than one command at a time (command combination).

**3) \***

**4) gcc**

GNU project C and C++ Compiler.

**Practical : 3.1**

echo "enter no."

read n

a=0

b=1

echo "Fibonacci Number : $b"

for((i=1;i<=n-2;i++))

do

fibo=$(( $a+$b ));

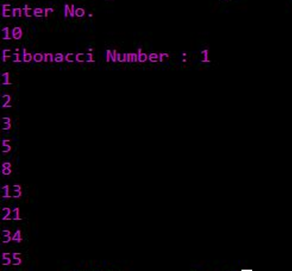
echo $fibo;

a=$b;

b=$fibo

done

**OUTPUT :**

****

**Practical : 3.2**

echo "Enter Value For Factorial :"

read n

fact=1;

for((i=1;i<=$n;i++))

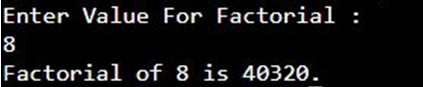
do

fact=$(( $fact\*$i ))

done

echo “Factorial of$n is %fact.”

**OUTPUT :**

****

**Practical : 3.3**

echo "Enter Number n :"

read no

echo "Enter values which you want to Sort :"

for (( i=0 ; i < $no ; i++ ))

do

read n[$i]

done

for (( i=0 ; i<$no ; i++ ))

do

for (( j = $i ; j < $no; j++ ))

do

if [ "${n[$i]}" -gt "${n[$j]}" ]; then

t=${n[$i]}

n[$i]=${n[$j]}

n[$j]=$t

fi

done

done

echo "Sorted Elements are…"

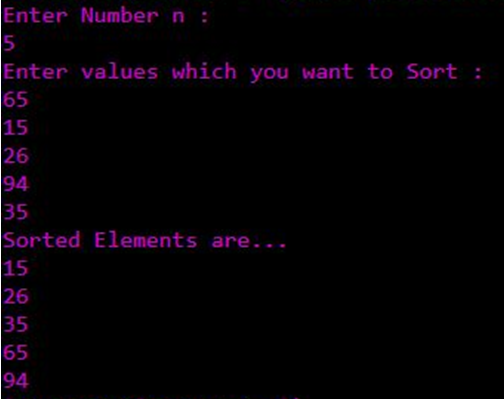
for((i=0; i<=$no; i++))

do

echo ${n[$i]}

done

**OUTPUT :**

****

**Practical 9**

**Aim:** Write a C program in UNIX to implement inter process communication (IPC) using Semaphore.

**Code:**

**Writer.c**

#include <fcntl.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <unistd.h>

int main()

{

intfd;

char \* myfifo = "/tmp/myfifo";

/\* create the FIFO (named pipe) \*/

mkfifo(myfifo, 0666);

/\* write "Hi" to the FIFO \*/

fd = open(myfifo, O\_WRONLY);

write(fd, "Hi", sizeof("Hi"));

close(fd);

/\* remove the FIFO \*/

unlink(myfifo);

return 0;

}

**Reader.c**

#include <fcntl.h>

#include <stdio.h>

#include <sys/stat.h>

#include <unistd.h>

#define MAX\_BUF 1024

int main()

{

intfd;

char \* myfifo = "/tmp/myfifo";

charbuf[MAX\_BUF];

/\* open, read, and display the message from the FIFO \*/

fd = open(myfifo, O\_RDONLY);

read(fd, buf, MAX\_BUF);

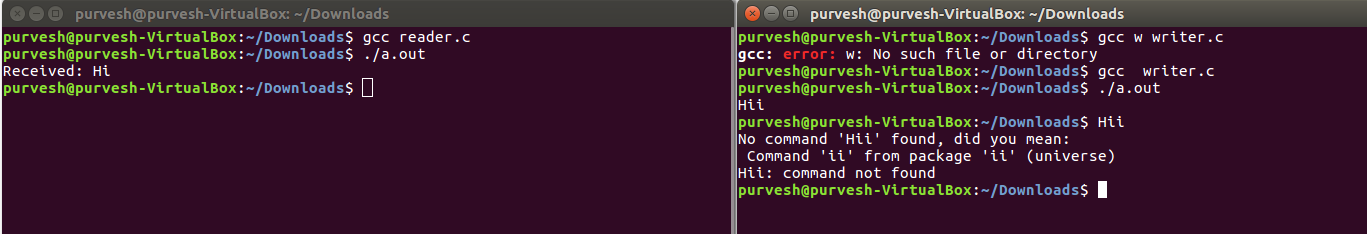
printf("Received: %s\n", buf);

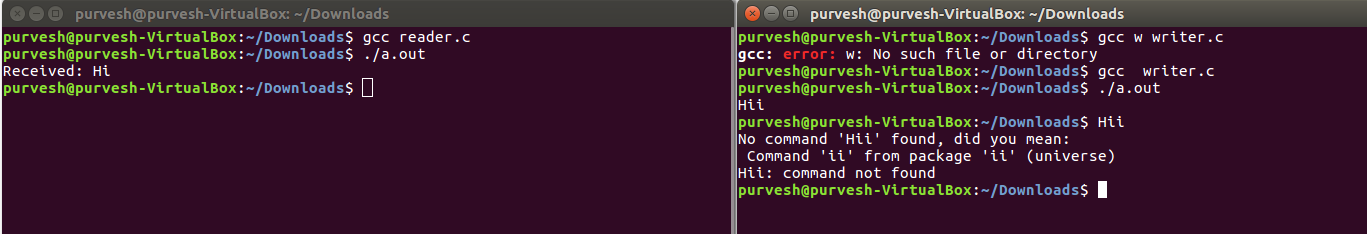
close(fd);

return 0;

}

**Output:**

****

****

**Practical 10**

**Aim:** Thread synchronization using counting semaphores and mutual exclusion using mutex.

**Code:**

#include<stdio.h>

#include<semaphore.h>

#include<sys/types.h>

#include<pthread.h>

#include<unistd.h>

#include<stdlib.h>

#define BUFFER\_SIZE 10

pthread\_mutex\_t mutex;

sem\_t empty,full;

int buffer[BUFFER\_SIZE];

int counter;

pthread\_t tid;

void \*producer();

void \*consumer();

void insert\_item(int);

int remove\_item();

void initilize()

{

pthread\_mutex\_init(&mutex,NULL);

sem\_init(&full,0,0);

sem\_init(&empty,0,BUFFER\_SIZE);

}

void \*producer()

{

int item,wait\_time;

wait\_time=rand()%5;

sleep(wait\_time)%5;

item=rand()%10;

sem\_wait(&empty);

pthread\_mutex\_lock(&mutex);

printf("Producer produce %d\n\n",item);

insert\_item(item);

pthread\_mutex\_unlock(&mutex);

sem\_post(&full);

}

void \*consumer()

{

int item,wait\_time;

wait\_time=rand()%5;

sleep(wait\_time);

sem\_wait(&full);

pthread\_mutex\_lock(&mutex);

item=remove\_item();

printf("Consumer consume %d\n\n",item);

pthread\_mutex\_unlock(&mutex);

sem\_post(&empty);

}

void insert\_item(int item)

{

buffer[counter++]=item;

}

int remove\_item()

{

return buffer[--counter];

}

int main()

{

int n1,n2;

int i;

printf("Enter number of Producers");

scanf("%d",&n1);

printf("Enter number of Consumers");

scanf("%d",&n2);

initilize();

for(i=0;i<n1;i++)

pthread\_create(&tid,NULL,producer,NULL);

for(i=0;i<n2;i++)

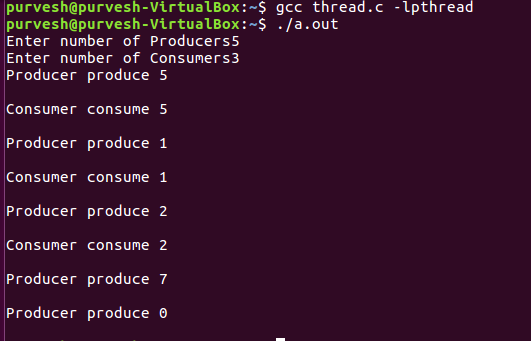
pthread\_create(&tid,NULL,consumer,NULL);

sleep(5);

exit(0);

}

**Output:**

****

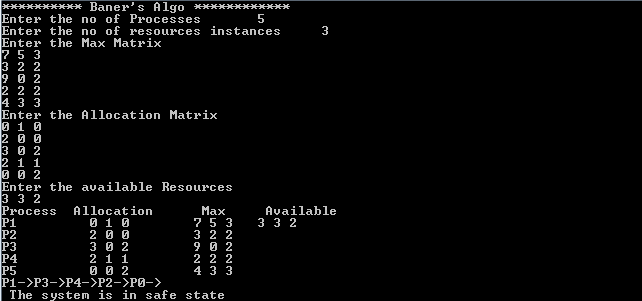
**Practical 11**

**Aim:** Write a C program in UNIXto implement Bankers algorithm for Deadlock Avoidance.

**Code:**

#include<stdio.h>  
#include<conio.h>  
int max[100][100];  
int alloc[100][100];  
int need[100][100];  
int avail[100];  
int n,r;  
void input();  
void show();  
void cal();  
int main()  
{  
 int i,j;  
 printf(“ Baner's Algo");  
 input();  
 show();  
 cal();  
 getch();  
 return 0;  
}  
void input()  
{  
 int i,j;  
 printf("Enter the no of Processes\t");  
 scanf("%d",&n);  
 printf("Enter the no of resources instances\t");  
 scanf("%d",&r);  
 printf("Enter the Max Matrix\n");  
 for(i=0;i<n;i++)  
 {  
 for(j=0;j<r;j++)  
 {  
 scanf("%d",&max[i][j]);  
 }  
 }  
 printf("Enter the Allocation Matrix\n");  
 for(i=0;i<n;i++)  
 {  
 for(j=0;j<r;j++)  
 {  
 scanf("%d",&alloc[i][j]);  
 }  
 }  
 printf("Enter the available Resources\n");  
 for(j=0;j<r;j++)  
 {  
 scanf("%d",&avail[j]);  
 }  
}  
void show()  
{  
 int i,j;  
 printf("Process\t Allocation\t Max\t Available\t");  
 for(i=0;i<n;i++)  
 {  
 printf("\nP%d\t   ",i+1);  
 for(j=0;j<r;j++)  
 {  
 printf("%d ",alloc[i][j]);  
 }  
 printf("\t");  
 for(j=0;j<r;j++)  
 {  
 printf("%d ",max[i][j]);   
 }  
 printf("\t");  
 if(i==0)  
 {  
 for(j=0;j<r;j++)  
 printf("%d ",avail[j]);  
 }  
 }  
}  
void cal()  
{  
 int finish[100],temp,need[100][100],flag=1,k,c1=0;  
 int safe[100];  
 int i,j;  
 for(i=0;i<n;i++)  
 {  
 finish[i]=0;  
 }  
 //find need matrix  
 for(i=0;i<n;i++)  
 {  
 for(j=0;j<r;j++)  
 {  
 need[i][j]=max[i][j]-alloc[i][j];  
 }  
 }  
 printf("\n");  
 while(flag)  
 {  
 flag=0;  
    for(i=0;i<n;i++)  
 {  
 int c=0;  
 for(j=0;j<r;j++)  
 {  
 if((finish[i]==0)&&(need[i][j]<=avail[j]))  
 {  
      c++;  
 if(c==r)  
 {  
 for(k=0;k<r;k++)  
 {  
 avail[k]+=alloc[i][j];  
 finish[i]=1;  
 flag=1;  
       }  
   printf("P%d->",i);  
   if(finish[i]==1)  
 {  
   i=n;  
   }  
        }  
 }  
 }  
   }  
  }  
  for(i=0;i<n;i++)  
  {  
 if(finish[i]==1)  
 {  
 c1++;  
 }  
 else  
 {  
 printf("P%d->",i);  
 }  
 }  
 if(c1==n)  
 {  
 printf("\n The system is in safe state");  
 }  
 else  
 {  
 printf("\n Process are in dead lock");  
 printf("\n System is in unsafe state");  
 }   
}

**Output:**



Practical 12

**Aim: To perform Kernel Space Programming.**

**Problem Statement:**

Implement and add a loadable kernel module to Linux kernel, demonstrate using insmod, lsmod and rmmod commands. A sample kernel space program should print the "Hello World" while loading the kernel module and "Goodbye World" while unloading the kernel module.

**Theory:**

**What is a loadable kernel module?**

To add a new code to a Linux kernel, it is necessary to add some source files to kernel source tree and recompile the kernel. You can also add code to the Linux kernel while it is running. A chunk of code added in such way is called a loadable kernel module.

**Typical modules:**

Device drivers, File system drivers, System calls

**Advantages of modules**

1. There is no necessity to rebuild the kernel, when a new kernel option is added.
2. Modules help find system problems (if system problem caused a module just don't load it).
3. Modules are much faster to maintain and debug.
4. Modules once loaded are in as much fast as kernel.

**Module Implementation**

* Modules are stored in the file system as ELF object files.
* The kernel makes sure that the rest of the kernel can reach the module's global symbols.
* Module must know the addresses of symbols (variables and functions) in the kernel and in other modules (/proc/kallsyms).
* The kernel keeps track of the use of modules, so that no modules is unloaded while another module or kernel is using it (/proc/modules)
* The kernel considers only modules that have been loaded into RAM by the insmod program and for each of them allocates memory area containing: a module object
* null terminated string that represents module's name
* the code that implements the functions of the module

**Module Object**

****

**Programs for linking and unlinking**

1. **insmod**

* inserts the module into the kernel space.
* Reads from the name of the module to be linked
* Locates the file containing the module's object code
* Computes the size of the memory area needed to store the module code, its name, and the module object.
* Invokes the create\_module( ) system call

1. **Insmod**

* Invokes the query\_module( ) system call Using the kernel symbol table, the module symbol tables, and the address returned by the create\_module( ) system call, relocates the object code included in the module's file.
* Allocates a memory area in the User Mode address space and loads with a copy of the module object.
* Invokes the init\_module( ) system call, passing to it the address of the User Mode memory area.
* Releases the User Mode memory area and terminates

1. **lsmod**

* reads /proc/modules and displays on the terminal.

1. **Rmmod**

* reads the name of the module to be unlinked.
* Invokes the query\_module( )
* Invokes the delete\_module( ) system call

1. **modprobe**

* takes care of possible complications due to module dependencies, uses depmod program and /etc/modules.conf file

**Compiling kernel module**

* A kernel module is not an independent executable, but an object file which will be linked into the kernel in runtime and they should be compiled with
* -c flag
* \_KERNEL\_ symbol
* CONFIG\_MODVERSIONS symbol

**Steps for the program compilation, linking, loading and executing**

1. Edit a Hello.c program
2. Edit a makefile
3. The program and makefile should be kept in a single folder.
4. Change directory to this folder
5. Execute “ make” on terminal
6. Execute insmodHello.ko from this folder.
7. Execute dmesg from a terminal to see the kernel buffer
8. contents (reads the kernel log file /var/log/syslog)
9. Execute lsmod from this folder.
10. Execute rmmodHello.ko from this folder when done.

**Files created after building the module**

* Hello.o - Module object file before linking
* Hello.mod.c - Contains module’s information
* Hello.mod.o - After compilation and linking of Hello.mod.c
* Modules.order - The order in which two or three modules get linked..
* Modules.symvers - Symbol versions if any.
* Hello.ko - A module kernel object file after linking Hello.o and
* Hello.mod.o - Hello.mod.c details

**Hello.mod.c details**

Include/linux/module.h , include/linux/vermagic.h, include/linux/kernel.h

Step 1 –

* Find all modules from the files listed in $(MODVERDIR)/
* modpost is then used to create one <module>.mod.c file
  + Create one Module.symvers file with CRC for all exported symbols
  + Compile all <module>.mod.c files
  + Final link of the module to a <module.ko> file

Step 2 - is used to place certain information in the module's ELF

• Version magic (see include/linux/vermagic.h for full details)

• Kernel release

• GCC Version

• Module info

• Module version (MODULE\_VERSION)

• Module license (MODULE\_LICENSE)

Step 3 - is used to allow module versioning in external modules, where the CRC of each module is retrieved from the Module.symvers file.

**Module definition:**

struct module \_\_this\_module

\_\_attribute\_\_((section(".gnu.linkonce.this\_module")

)) = {

.name = KBUILD\_MODNAME,

.init = init\_module,

#ifdef CONFIG\_MODULE\_UNLOAD

.exit = cleanup\_module,

#endif

.arch = MODULE\_ARCH\_INIT

}

**The symbol version definitions:**

(nm – modules in the object file- nm hello.ko)

static const struct modversion\_info \_\_\_\_versions[]

\_\_used

\_\_attribute\_\_((section("\_\_versions"))) =

{

{ 0x4d5503c4, "module\_layout" },

{ 0xb4390f9a, "mcount" },

{ 0x50eedeb8, "printk" },

};

• */usr/src/linux-version-headers/Module.symvers*

• *This file contains the list of symbols that are exported*

*by kernel to loadable* 07/01/15 *kernel modules (LKM).*

**Conclusion:**

Loadable kernel module is implemented and added to Linux kernel using insmod, lsmod and rmmod commands.