

BANGALORE UNIVERSITY
UNIVERSITY VISVESVARAYA COLLEGE OF
ENGINEERING
K R Circle Bengaluru-560001



UNIX SYSTEM PROGRAMMING
LAB MANUAL

Department of Computer Science and Engineering
YEAR 2020-2021

TABLE OF CONTENT

Sl No	Name of the Experiment	Page No
1	Check POSIX runtime limits	8
2	a)Copy file using a system calls b)Output the contents of the environment list	12 15
3	a)Emulate the UNIX ln command b>Create child from parent using fork()	18 22
4	Two process communicating using shared memory	25
5	Producer and consumer problem using semaphores	28
6	Round Robin Scheduling Algorithm	32
7	Priority Based Scheduling algorithm	36
8	Sender sends data in queue and receiver reads data in queue	40
9	Parent writes message to pipe and child reads message from pipe	44
10	Demonstrate setting up a simple Web Server and Host Website on Your Own Linux Computer.	48
11	a)Create two threads using pthread, where both thread counts until 100 and joins later. b) Create two threads using pthreads.Here main thread creates 5 and other threads for 5 times and each new thread print "Hello World" message with its thread number.	53 56
12	Using Socket APIs establish communication between remote and local processes	59

UNIX SYSTEM

Introduction to UNIX System

Unix is an Operating System that is truly the base of all Operating Systems like Ubuntu, Solaris, POSIX, etc. It was developed in the 1970s by Ken Thompson, Dennis Ritchie, and others in the AT&T Laboratories. It was originally meant for programmers developing software rather than non-programmers.

Unix and the C were found by AT&T and distributed to government and academic institutions, which led to both being ported to a wider variety of machine families than any other operating system. The main focus that was brought by the developers in this operating system was the Kernel. Unix was considered to be the heart of the operating System. System Structure of Unix OS are as follows:

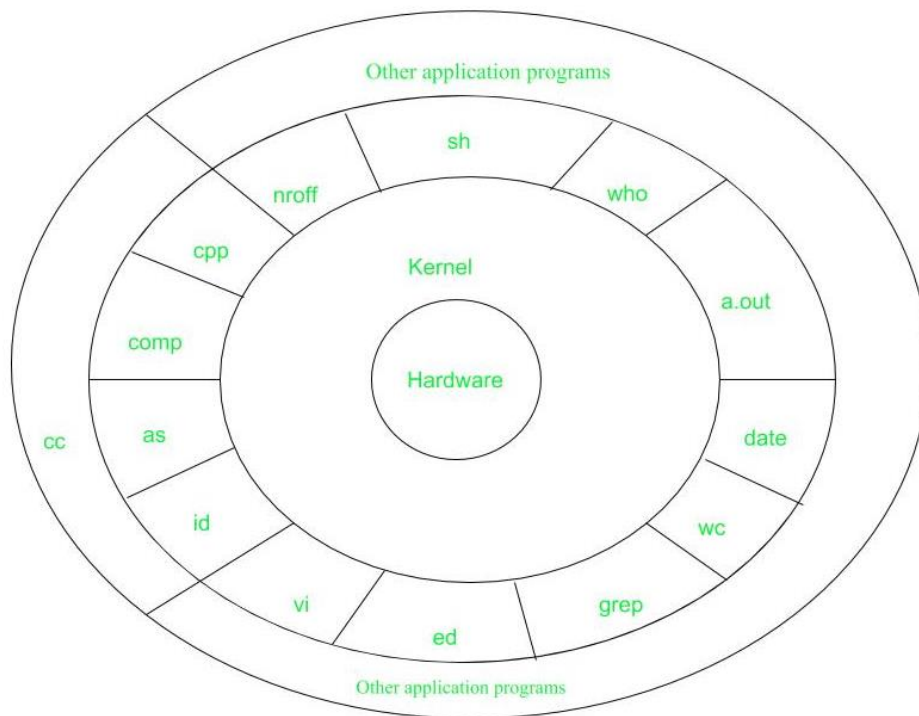


Figure – system structure

Layer-1: Hardware –

It consists of all hardware related information.

Layer-2: Kernel –

It interacts with hardware and most of the tasks like memory management, task scheduling, and management are done by the kernel.

Layer-3: Shell commands –

Shell is the utility that processes your requests. When you type in a command at the terminal, the shell interprets the command and calls the program that you want.

There are various commands like cp, mv, cat, grep, id, wc, nroff, a.out and more.

Layer-4: Application Layer –

It is the outermost layer that executes the given external applications.

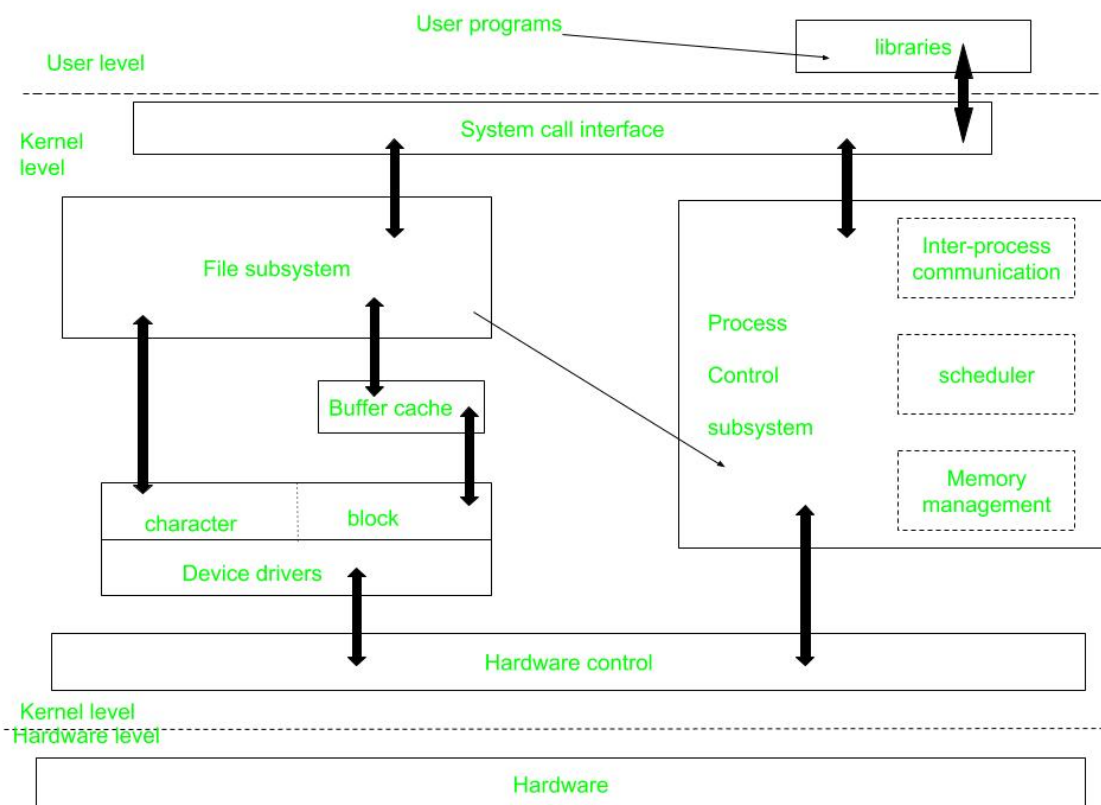


Figure – kernel and its block diagram

This diagram shows three levels: user, kernel, and hardware.

The system call and library interface represent the border between user programs and the kernel. System calls look like ordinary function calls in C programs. Assembly language programs may invoke system calls directly without a system call library. The libraries are linked with the programs at compile time.

The set of system calls into those that interact with the file subsystem and some system calls interact with the process control subsystem. The file subsystem manages files, allocating file space, administering free space, controlling access to files, and retrieving data for users.

Processes interact with the file subsystem via a specific set of system calls, such as open (to open a file for reading or writing), close, read, write, stat (query the attributes of a file), chown (change the record of who owns the file), and chmod (change the access permissions of a file).

The file subsystem accesses file data using a buffering mechanism that regulates data flow between the kernel and secondary storage devices. The buffering mechanism interacts with block I/O device drivers to initiate data transfer to and from the kernel.

Device drivers are the kernel modules that control the operation of peripheral devices. The file subsystem also interacts directly with “raw” I/O device drivers without the intervention of the buffering mechanism. Finally, the hardware control is responsible for handling interrupts and for communicating with the machine. Devices such as disks or terminals may interrupt the CPU while a process is executing. If so, the kernel may resume execution of the interrupted process after servicing the interrupt.

Interrupts are not serviced by special processes but by special functions in the kernel, called in the context of the currently running process.

Unix Commands:

<code>man {command}</code>	- Type man rm to read the manual for the rm command.
<code>whatis{command}</code>	- Give short description of command.
<code>ls{path}</code>	- Gets a long listing of all files
<code>ls -l{path}</code>	- Long listing, with date, size and permissions
<code>cd{directory}</code>	- Go to directory
<code>cd~</code>	- Go back to home directory
<code>cd ..</code>	- Go back one directory
<code>mkdir</code>	- Creates new directory
<code>rmdir</code>	- Removes a directory
<code>pwd</code>	- Shows present working directory
<code>cp</code>	- Copy one file to other file
<code>rm</code>	- Remove a file
<code>gedit</code>	- open a file in text editor
<code>chmod</code>	- Give read, write and execute permission to a particular file
<code>sudo</code>	- Super user operation
<code>poweroff</code>	- Shuts down the system

What is gedit in Linux?

gedit is the default text editor of the GNOME desktop environment and part of the GNOME Core Applications. Designed as a general-purpose text editor, gedit emphasizes simplicity and ease of use, with a clean and simple GUI, according to the philosophy of the GNOME project

To install gedit:

Select gedit in Synaptic (System → Administration → Synaptic Package Manager)

From a terminal or ALT-F2:

```
sudo apt-get install gedit.
```

How to save and exit gedit?

You can save files in the following ways:

To save changes to the current file, choose File->Save or click Save on the toolbar. ...

To save a new file or to save an existing file under a new filename, choose File->Save As. ...

To save all of the files that are currently open in gedit, choose File->Save All

Command Line Tips

Opening gedit via the command line allows the user to take advantage of several options unavailable from the GUI menu. If a path is not included in the startup command, gedit will look for the file in the current directory. If the file is not found, gedit will open a blank file with the file name entered on the command line:

1. **To open a specific file:** *gedit filename*
2. **To open multiple files:** *gedit file1 file2*
3. **To edit system files** such as `sources.list` and `fstab`, open it with administrative privileges. It is NOT recommended to manually run graphical applications with administrative privileges, but in case you insist to do it, be sure to use *gksudo* rather than *sudo*. *gksudo gedit*
4. **To open at a specific line number**, useful when an error message includes the line number, include "+<line number>". (*gksudo* is used in this example since the file is a system file owned by root): *gksudo gedit +21 /etc/apt/sources.list*

Environmental Variables:

Environment variables or **ENVs** basically define behavior of the environment. They can affect the processes ongoing or the programs that are executed in the environment.

Scope of an environment variable

Scope of any variable is the region from which it can be accessed or over which it is defined. An environment variable in Linux can have **global** or **local** scope.

Global

A globally scoped ENV that is defined in a terminal can be accessed from anywhere in that particular environment which exists in the terminal. That means it can be used in all kind of scripts, programs or processes running in the environment bound by that terminal.

Local

A locally scoped ENV that is defined in a terminal cannot be accessed by any program or process running in the terminal. It can only be accessed by the terminal(in which it was defined) itself.

Some commonly used ENVs in Linux

\$USER: Gives current user's name.

\$PATH: Gives search path for commands.

\$PWD: Gives the path of present working directory.

\$HOME: Gives path of home directory.

\$HOSTNAME: Gives name of the host.

\$LANG: Gives the default system language.

\$EDITOR: Gives default file editor.

\$UID: Gives user ID of current user.

\$SHELL: Gives location of current user's shell program.

PROGRAM-1

Check POSIX runtime limits

Write a C/C++ POSIX compliant program to check the following limits:

1. Number of clock ticks
2. Max number of child processes
3. Max path length
4. Max number of characters in a file name
5. Max number of open files/processes

Commands used in the program:

1. `_POSIX_SOURCE`

If you define this macro, then the functionality from the POSIX.1 standard (IEEE Standard 1003.1) is available, as well as all of the ISO C facilities.

The state of `_POSIX_SOURCE` is irrelevant if you define the macro `_POSIX_C_SOURCE` to a positive integer.

2. `_POSIX_C_SOURCE 199309L`

If you define this macro to a value greater than or equal to 199309L, then the functionality from the 1993 edition of the POSIX.1b standard (IEEE Standard 1003.1b-1993) is made available.

Header files used:

<stdio.h>: The C programming language provides many standard library functions for file input and output. These functions make up the bulk of the C standard library header `<stdio.h>`.

<unistd.h>: This header defines miscellaneous symbolic constants and types, and declares miscellaneous functions. The contents of this header are shown below.

The following symbolic constants are defined:

`_POSIX_VERSION`

`_POSIX2_VERSION`

`_POSIX2_C_VERSION`

`_XOPEN_VERSION`

`_POSIX_VERSION` is defined in the ISO POSIX-1 standard. It changes with each new version of the ISO POSIX-1 standard.

`_POSIX2_VERSION` is defined to have the value of the ISO POSIX-2 standard's `_POSIX2_VERSION` limit. It changes with each new version of the ISO POSIX-2 standard.

`_XOPEN_VERSION` is defined as an integer value equal to 500.

`_XOPEN_XCU_VERSION`

<limits.h>: The `limits.h` header determines various properties of the various variable types. The macros defined in this header, limits the values of various variable types like `char`, `int` and `long`.

These limits specify that a variable cannot store any value beyond these limits, for example an unsigned character can store up to a maximum value of 255.

Program:

```
#include<stdio.h>
```

```
#include<unistd.h>
```

```
#include<limits.h>
```

```
##define _POSIX_SOURCE
```

```
##define _POSIX_C_SOURCE 199309L
```

```

int main()

{

printf("Runtime values\n");

printf("The max number of clock ticks : %ld\n",sysconf(_SC_CLK_TCK));

printf("The max runtime child processes : %ld\n",sysconf(_SC_CHILD_MAX);

printf("The max runtime path length :%ld\n",pathconf("usp1.cpp",_PC_PATH_MAX));

printf("The max characters in a file name :%ld\n",pathconf("usp1.cpp",_PC_NAME_MAX));

printf("The max number of opened files : %ld\n",sysconf(_SC_OPEN_MAX));

return 0;


/* Many system expects argument of type long int (%ld) instead of int (%d) <-- Can use int
type also runtime child processes : %ld\n",sysconf(_SC_CHILD_MAX)); printf("The max
runtime path */

```

Output:

```

cc prog1.c -o prog1

$ ./prog1

Runtime values

The max number of clock ticks : 100

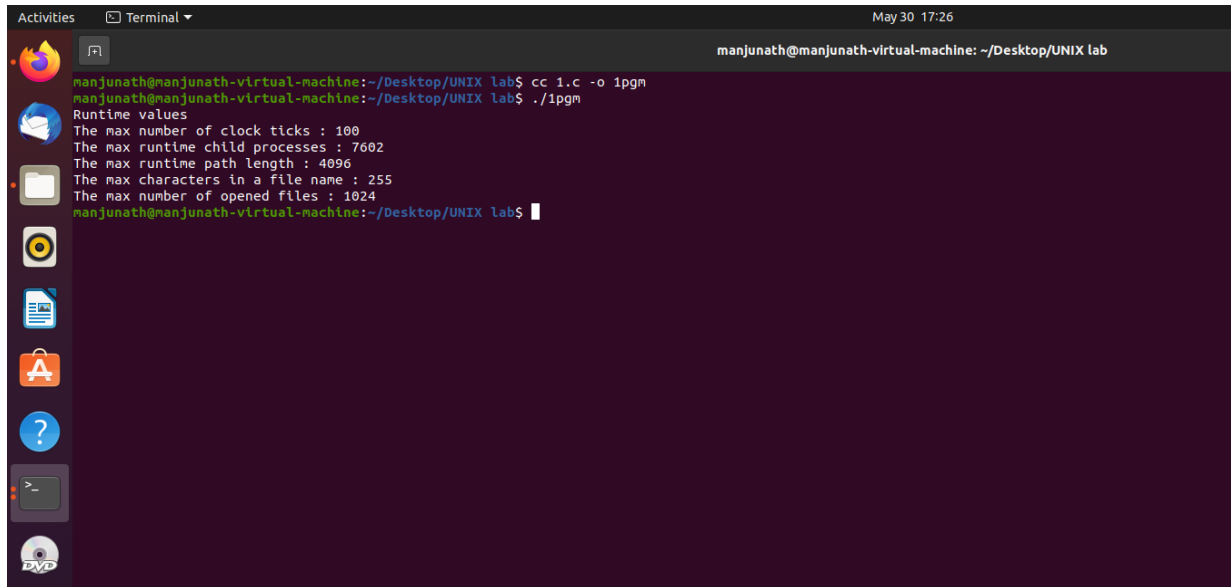
The max runtime child processes : 7602

The max runtime path length : 4096

The max characters in a file name : 255

The max number of opened files : 102

```



The screenshot shows a terminal window titled "Terminal" with a date and time of "May 30 17:26". The user is logged in as "manjunath" on a "manjunath-virtual-machine". The current directory is "~/Desktop/UNIX lab". The terminal shows the following commands and output:

```
manjunath@manjunath-virtual-machine:~/Desktop/UNIX lab$ cc 1.c -o 1pgm
manjunath@manjunath-virtual-machine:~/Desktop/UNIX lab$ ./1pgm
Runtime values
The max number of clock ticks : 100
The max runtime child processes : 7602
The max runtime path length : 4096
The max characters in a file name : 255
The max number of opened files : 1024
manjunath@manjunath-virtual-machine:~/Desktop/UNIX lab$
```

The terminal window has a dark purple background. On the left side, there is a vertical dock with several application icons: a red and orange flame icon (Firefox), a blue and white icon (LibreOffice Writer), a blue and white icon (LibreOffice Calc), an orange icon (LibreOffice Impress), a blue icon with a question mark, a green icon with a terminal symbol, and a DVD icon. The top of the window has a title bar with "Activities" and "Terminal" tabs.

PROGRAM-2

2a. Write C program that makes copy of a file using standard I/O and System calls.

Commands used in the program:

1. define BUFSIZE 1024: defines a macro named BUFFER_SIZE as an abbreviation for the token 1024. If somewhere after this #define directive there comes a C statement of the form

foo = (char *) malloc (BUFFER_SIZE); then the C pre-processor will recognize and expand the macro BUFFER_SIZE.

Header files used:

<syscall.h>: syscall() is a small library function that invokes the system call whose assembly language interface has the specified number with the specified arguments. Employing syscall() is useful, for example, when invoking a system call that has no wrapper function in the C library.

syscall() saves CPU registers before making the system call, restores the registers upon return from the system call, and stores any error returned by the system call in errno(3).

<stdlib.h>: The stdlib.h header defines four variable types, several macros, and various functions for performing general functions.

Following are the macros defined in the header stdlib.h -NULL, EXIT_FAILURE, EXIT_SUCCESS, RAND_MAX, MB_CUR_MAX.

Program:

```
#include<syscall.h>

#include<stdio.h>

#include<stdlib.h>

#define BUFSIZE 1024

char buf[BUFSIZE];

int main(int argc, char** argv) {

int src, dst, amount;
```

```

if (argc!=3) {
printf("Usage: cp <src> <dst>\n");
return 1;
}

src = open(argv[1]);
if (src==-1) {
printf("Unable to open %s\n", argv[1]);
return 1;
}
creat(argv[2]);
dst = open(argv[2]); if (dst==-1) {
printf("Unable to create %s\n", argv[2]);
return 1;
}
while ((amount = read(src, buf, BUFSIZE))>0)
{ write(dst, buf, amount);
}
close(src);
close(dst);
return 0;
}

```

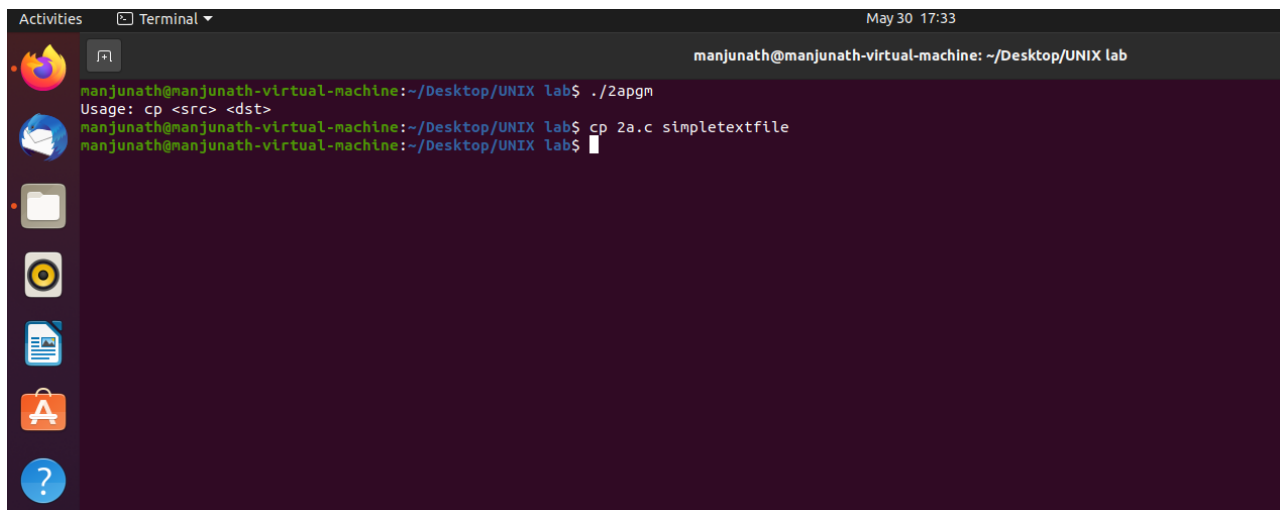
Output :

```
cc 2a.c -o prog2a
```

```
$ ./prog2a
```

```
Usage : cp <src> <dst>
```

```
$ cp 2a.c simpletextfile
```



The screenshot shows a terminal window titled "Terminal" with a dark background. The prompt is "manjunath@manjunath-virtual-machine: ~/Desktop/UNIX lab". The user enters the command `./2apgm`, which outputs `Usage: cp <src> <dst>`. Then, the user enters `cp 2a.c simpletextfile`, and the prompt returns. The left sidebar of the window shows various application icons like Firefox, Files, and the Dash icon.

```
manjunath@manjunath-virtual-machine:~/Desktop/UNIX lab$ ./2apgm
Usage: cp <src> <dst>
manjunath@manjunath-virtual-machine:~/Desktop/UNIX lab$ cp 2a.c simpletextfile
manjunath@manjunath-virtual-machine:~/Desktop/UNIX lab$
```

2b. Output the contents of its environment list

Commands used: extern char **environ: Inclusion of the <unistd.h> header file provides for definitions of environ, the environ variable is an array of character strings containing environment variables and their values in the form name=value (environment variable names cannot contain the character '='). Programs should obtain the definition of environ by the inclusion of <unistd.h>. The practice of defining environ in a program as extern char **environ;

Program:

```
#include<stdio.h>

int main(int argc, char* argv[ ])
{
    int i;
    char **ptr;
    extern char **environ;
    for( ptr = environ; *ptr != 0; ptr++ ) /*echo all env strings*/
        printf("%s\n", *ptr);
    return 0;
}
```

To Run the Program:

```
[root@localhost /]# cc environ1.c
[root@localhost /]# ./a.out
```

Output:

```
SSH_AGENT_PID=3207
HOSTNAME=localhost.localdomain
DESKTOP_STARTUP_ID=
SHELL=/bin/bash
TERM=xterm
HISTSIZE=1000
KDE_NO_IPV6=1
GTK_RC_FILES=/etc/gtk/gtkrc:/root/.gtkrc-1.2-gnome2
```


WINDOWID=44040273
OLDPWD=/root/tan
QTDIR=/usr/lib/qt-3.3
QTINC=/usr/lib/qt-3.3/include
USER=root
LS_COLORS=no=00:fi=00:di=00;34:l
GNOME_KEYRING_SOCKET=/tmp/keyring-vsDBVL/socket
SSH_AUTH_SOCK=/tmp/ssh-SEwJHJ3149/agent.3149
KDEDIR=/usr
SESSION_MANAGER=local/localhost.localdomain:/tmp/.ICE-unix/3149
MAIL=/var/spool/mail/root
DESKTOP_SESSION=default
PATH=/usr/lib/qt3.3/bin:/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
GDM_XSERVER_LOCATION=local
INPUTRC=/etc/inputrc
PWD=/root/tan/uspl
XMODIFIERS=@im=none
KDE_IS_PRELINKED=1
LANG=en_US.UTF-8
GDMSESSION=default
SSH_ASKPASS=/usr/libexec/openssh/gnome-ssh-askpass
HOME=/root
SHLVL=2
GNOME_DESKTOP_SESSION_ID=Default
LOGNAME=root
QTLIB=/usr/lib/qt-3.3/lib
CVS_RSH=ssh
DBUS_SESSION_BUS_ADDRESS=unix:abstract=/tmp/dbusi0dVljt8MQ,
guid=f47759511fe6adb91b249b482809fa00 LESSOPEN=|/usr/bin/lesspipe.sh %s

```
Activities Terminal May 30 17:34
manjunath@manjunath-virtual-machine: ~/Desktop/UNIX lab

DBUS_STARTER_BUS_TYPE=session
PWD=/home/manjunath/Desktop/UNIX lab
LOGNAME=manjunath
XDG_SESSION_DESKTOP=ubuntu
XDG_SESSION_TYPE=x11
GPG_AGENT_INFO=/run/user/1000/gnupg/S.gpg-agent:0:1
XAUTHORITY=/run/user/1000/gdm/Xauthority
WINDOWPATH=2
HOME=/home/manjunath
USERNAME=manjunath
IM_CONFIG_PHASE=1
LANG=en_IN
LS_COLORS=rs=0:dl=01;34:ln=01;36:rh=00;pl=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31;01:ml=00:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;3
1:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;31:*.lzh=01;31:*.lzm=01;31:*.tlz=01;31:*.txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31:*.z=01;31:*.dz=01;31:*.
.gz=01;31:*.lrz=01;31:*.lz=01;31:*.lzo=01;31:*.xz=01;31:*.zst=01;31:*.taz=01;31:*.bz2=01;31:*.bz=01;31:*.tbz2=01;31:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war
=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;31:*.alz=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.wlm=01;31:*.swm=01;31:*.dwm=01;31:*.esd=01;31:*.jpg=0
1;35:*.jpeg=01;35:*.mjpg=01;35:*.mjpeg=01;35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=01;35:*.s
vg=01;35:*.svgz=01;35:*.nngv=01;35:*.pcx=01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.n2v=01;35:*.nkv=01;35:*.webm=01;35:*.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:
*.qt=01;35:*.nuv=01;35:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;35:*.fli=01;35:*.flv=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;35:*.yuv=01;35:*.cg
n=01;35:*.enf=01;35:*.ogv=01;35:*.ogx=01;35:*.aac=00;36:*.au=00;36:*.flac=00;36:*.m4a=00;36:*.mld=00;36:*.mldl=00;36:*.mka=00;36:*.mp3=00;36:*.mpc=00;36:*.ogg=00;36:*.ra=00;36:*.wav
=00;36:*.oga=00;36:*.opus=00;36:*.spx=00;36:*.xspf=00;36:
XDG_CURRENT_DESKTOP=ubuntu:GNOME
VTE_VERSION=6003
GNOME_TERMINAL_SCREEN=/org/gnome/Terminal/screen/f5abd3b1_908b_4328_be9e_2a1de9798bdb
INVOCATION_ID=372fe23209e84292beed4f258d17b3c7
MANAGERPID=1335
LESSCLOSE=/usr/bin/lesspipe %s %s
XDG_SESSION_CLASS=user
TERM=xterm-256color
LESSOPEN=| /usr/bin/lesspipe %s
USER=manjunath
GNOME_TERMINAL_SERVICE=:1.117
DISPLAY=:0
SHLVL=1
QT_IM_MODULE=ibus
DBUS_STARTER_ADDRESS=unix:path=/run/user/1000/bus,guid=d560def250c9fe67e9292a9560b372a3
XDG_RUNTIME_DIR=/run/user/1000
JOURNAL_STREAM=8:47223
XDG_DATA_DIRS=/usr/share/ubuntu:/usr/local/share/:/usr/share/:/var/lib/snapd/desktop
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
GDMSESSION=ubuntu
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus,guid=d560def250c9fe67e9292a9560b372a3
_=/usr/bin/gdm
manjunath@manjunath-virtual-machine:~/Desktop/UNIX lab$
manjunath@manjunath-virtual-machine:~/Desktop/UNIX lab$
```

PROGRAM-3

3a. Write a C / C++ program to emulate the unix ln command

Commands used:

Symlink: A symlink (also called a symbolic link) is a type of file in Linux that points to another file or a folder on your computer. Symlinks are similar to shortcuts in Windows. Some people call symlinks "soft links" – a type of link in Linux/UNIX systems – as opposed to "hard links."

Header files used:

<sys/types.h>: The <sys/types.h> header contains a number of basic derived types that should be used whenever appropriate. In particular, the following are of special interest:

clock_t

The type clock_t represents the system times in clock ticks.

dev_t

The type dev_t is used for device numbers.

off_t

The type off_t is used for file sizes and offsets.

ptrdiff_t

The type ptrdiff_t is the signed integral type for the result of subtracting two pointers.

size_t

The type size_t is for the size, in bytes, of objects in memory.

ssize_t

The signed size type ssize_t is used by functions that return a count of bytes or an error indication.

time_t

The type time_t is used for time in seconds.

Program:

```
#include<stdio.h>

#include<sys/types.h>

#include<unistd.h>

#include<string.h>

int main(int argc, char * argv[])

{

if(argc < 3 || argc > 4 || (argc == 4 && strcmp(argv[1],"-s")))

{

printf("Usage: ./a.out [-s] <org_file> <new_link>\n");

return 1;

}

if(argc == 4)

{

if((symlink(argv[2], argv[3])) == -1)

printf("Cannot create symbolic link\n") ;

elseprintf("Symbolic link created\n") ;

}

else

{

if((link(argv[1], argv[2])) == -1)

printf("Cannot create hard link\n") ;

else

printf("Hard link created\n") ;

}

return 0;

}
```

Output:

To Run the Program:

```
[root@localhost /]# cc link.c
```

```
[root@localhost /]# ./a.out
```

Sample Output:

```
Usage: ./a.out [-s] <org_file> <new_link>
```

```
[root@localhost uspl]# ./a.out 1 2 3 4
```

```
Usage: ./a.out [-s] <org_file> <new_link>
```

```
[root@localhost uspl]# ./a.out 1.c z
```

Hard link created

```
[root@localhost uspl]# ls -l
```

```
-rw-r--r-- 2 root root 657 Mar 27 16:44 1a.c
```

```
-rw-r--r-- 2 root root 657 Mar 27 16:44 z (Bolded columns are hardlink count & inode number respectively)
```

```
[root@localhost uspl]# ./a.out 1a.c z
```

Cannot create hard link (Because z already exists)

```
[root@localhost uspl]# ./a.out -s 1a.c zz
```

Symbolic link created

```
[root@localhost uspl]# ls -l
```

```
-rw-r--r-- 2 root root 657 Mar 27 16:44 1a.c
```

```
lrwxrwxrwx 1 root root 4 Apr 1 18:32 zz ->
```

1a.c

```
[root@localhost uspl]# readlink zz
```

```
Activities Terminal May 30 17:37
manjunath@manjunath-virtual-machine: ~/Desktop/UNIX lab
manjunath@manjunath-virtual-machine:~/Desktop/UNIX lab$ cc 3a.c -o 3apgm
manjunath@manjunath-virtual-machine:~/Desktop/UNIX lab$ ./3apgm
Usage: ./a.out[-s] <org_file> <new_link>
manjunath@manjunath-virtual-machine:~/Desktop/UNIX lab$ ./3apgm 1.c z
Hard link created
manjunath@manjunath-virtual-machine:~/Desktop/UNIX lab$ ./3apgm -s 2a.c zz
Symbolic link created
manjunath@manjunath-virtual-machine:~/Desktop/UNIX lab$
```

```
Activities Terminal May 30 17:38
manjunath@manjunath-virtual-machine: ~/Desktop/UNIX lab
manjunath@manjunath-virtual-machine:~/Desktop/UNIX lab$ cc 3a.c -o 3apgm
manjunath@manjunath-virtual-machine:~/Desktop/UNIX lab$ ./3apgm
Usage: ./a.out[-s] <org_file> <new_link>
manjunath@manjunath-virtual-machine:~/Desktop/UNIX lab$ ./3apgm 1.c z
Hard link created
manjunath@manjunath-virtual-machine:~/Desktop/UNIX lab$ ./3apgm -s 2a.c zz
Symbolic link created
manjunath@manjunath-virtual-machine:~/Desktop/UNIX lab$ ls -l
total 144
-rw-rw-r-- 1 manjunath manjunath 470 May 30 17:22 11.c
-rw-rw-r-- 1 manjunath manjunath 569 May 30 17:20 1.c
-rwxrwxr-x 1 manjunath manjunath 16824 May 30 17:24 1pgm
-rw-rw-r-- 1 manjunath manjunath 549 May 30 17:22 2a.c
-rwxrwxr-x 1 manjunath manjunath 16976 May 30 17:30 2apgm
-rw-rw-r-- 1 manjunath manjunath 204 May 30 17:22 2b.c
-rwxrwxr-x 1 manjunath manjunath 16768 May 30 17:33 2bpgm
-rw-rw-r-- 1 manjunath manjunath 583 May 30 17:22 3a.c
-rwxrwxr-x 1 manjunath manjunath 16808 May 30 17:35 3apgm
-rw-rw-r-- 1 manjunath manjunath 340 May 30 17:22 3b.c
-rw-rw-r-- 1 manjunath manjunath 879 May 30 17:22 4.c
-rw-rw-r-- 1 manjunath manjunath 2247 May 30 17:22 6.c
-rw-rw-r-- 1 manjunath manjunath 1719 May 30 17:22 7.c
-rw-rw-r-- 1 manjunath manjunath 729 May 30 17:22 8receiver.c
-rw-rw-r-- 1 manjunath manjunath 748 May 30 17:22 8sender.c
-rw-rw-r-- 1 manjunath manjunath 440 May 30 17:22 9consumer.c
-rw-rw-r-- 1 manjunath manjunath 617 May 30 17:22 9producer.c
-rw-rw-r-- 1 manjunath manjunath 549 May 30 17:31 sampletextfile
-rw-rw-r-- 1 manjunath manjunath 549 May 30 17:32 sampletextfile
-rw-rw-r-- 2 manjunath manjunath 569 May 30 17:20 z
lrwxrwxrwx 1 manjunath manjunath 4 May 30 17:37 zz -> 2a.c
manjunath@manjunath-virtual-machine:~/Desktop/UNIX lab$
```

3b. Create a child from parent process using fork() and counter counts till 5 in both processes and display.

Commands used:

1.wait(NULL):

A call to wait() blocks the calling process until one of its child processes exits or a signal is received. After child process terminates, parent continues its execution after wait system call instruction. If any process has more than one child processes, then after calling wait(), parent process has to be in wait state if no child terminates. If only one child process is terminated, then return a wait() returns process ID of the terminated child process. If more than one child processes are terminated then wait() reap any arbitrarily child and return a process ID of that child process. When wait() returns they also define exit status (which tells our, a process why terminated) via pointer, If status are not NULL. If any process has no child process then wait() returns immediately "-1"

2. (fork() == 0):

Fork system call is used for creating a new process, which is called child process, which runs concurrently with the process that makes the fork() call (parent process). After a new child process is created, both processes will execute the next instruction following the fork() system call. A child process uses the same pc(program counter), same CPU registers, same open files which use in the parent process.

Negative Value: creation of a child process was unsuccessful.

Zero: Returned to the newly created child process.

Positive value: Returned to parent or caller. The value contains process ID of newly created child process.

3. pid:

A PID is an acronym for the process identification number. PID is automatically assigned to each process when it is created on a Linux operating system. Each process is confirmed with a unique PID

4.getpid:

getpid() returns the process ID (PID) of the calling process. (This is often used by routines that generate unique temporary filenames. getppid() returns the process ID of the parent of the calling process. This will be either the ID of the process that created this process using fork(), or, if that process has already terminated, the ID of the process to which this process has been reparented (either init(1) or a "subreaper" process defined via the prctl(2) PR_SET_CHILD_SUBREAPER operation).

Program:

```
#include<stdio.h>

int main()
{
for(int i=0;i<5;i++) // loop will run n times (n=5)
{
if(fork() == 0)
{
printf("[son] pid %d from [parent] pid %d\n",getpid(),getppid());
exit(0);
}
}
for(int i=0;i<5;i++) // loop will run n times (n=5)
wait(NULL);
}
```

Output :

```
cc 3b.c -o 3bprog
```

```
./3bprog
```

```
[son] pid 9149 from [parent] pid 9147
```

```
[son] pid 9152 from [parent] pid 9147
```

```
[son] pid 9151 from [parent] pid 9147
```

```
[son] pid 9150 from [parent] pid 9147
```

```
[son] pid 9148 from [parent] pid 914
```


Activities Terminal May 30 17:39

manjunath@manjunath-virtual-machine: ~/Desktop/UNIX lab

```
manjunath@manjunath-virtual-machine:~/Desktop/UNIX lab$ ./3bpgm
[son] pid 9149 from [parent] pid 9147
[son] pid 9152 from [parent] pid 9147
[son] pid 9151 from [parent] pid 9147
[son] pid 9150 from [parent] pid 9147
[son] pid 9148 from [parent] pid 9147
manjunath@manjunath-virtual-machine:~/Desktop/UNIX lab$
```

PROGRAM-4

4. Write a C program that illustrates 2 processes communicating using shared memory.

Header files used:

<sys/ipc.h>:The <sys/ipc.h> header is used by three mechanisms for interprocess communication (IPC): messages, semaphores and shared memory. All use a common structure type, ipc_perm to pass information used in determining permission to perform an IPC operation.

The structure ipc_perm contains the following members:

uid_t uid owner's user ID

gid_t gid owner's group ID

uid_t cuid creator's user ID

gid_t cgid creator's group ID

mode_t mode read/write permission

<sys/shm.h>:The <sys/shm.h> header shall define the following symbolic constants:

SHM_RDONLY:Attach read-only (else read-write).

SHM_RND:Round attach address to SHMLBA.

The <sys/shm.h> header shall define the following symbolic value:

SHMLBA: Segment low boundary address multiple.

<errno.h>:The <errno.h> header file defines the integer variable errno, which is set by system calls and some library functions in the event of an error to indicate what went wrong.

Program:

```
#include <sys/types.h>
```

```
#include <sys/ipc.h>
```

```
#include <sys/shm.h>
```

```
#include <unistd.h>
```

```
#include <string.h>
```

```

#include <errno.h>

int main(void) {
    pid_t pid;
    int *shared; /* pointer to the shm */
    int shmid;

    shmid = shmget(IPC_PRIVATE, sizeof(int), IPC_CREAT | 0666);
    printf("Shared Memory ID=%u",shmid);

    if (fork() == 0) { /* Child */
        /* Attach to shared memory and print the pointer */
        shared = shmat(shmid, (void *) 0, 0);
        printf("Child pointer %u\n", shared);
        *shared=1;
        printf("Child value=%d\n", *shared);
        sleep(2);
        printf("Child value=%d\n", *shared);
    } else { /* Parent */
        /* Attach to shared memory and print the pointer */
        shared = shmat(shmid, (void *) 0, 0);
        printf("Parent pointer %u\n", shared);
        printf("Parent value=%d\n", *shared);
        sleep(1);
        *shared=42;
        printf("Parent value=%d\n", *shared);
        sleep(5);
        shmctl(shmid, IPC_RMID, 0);
    }
}

```

Output:

```
cc 4.c -o 4prog
```

```
./4prog
```

```
Shared Memory ID=32820Parent pointer 22526900
```

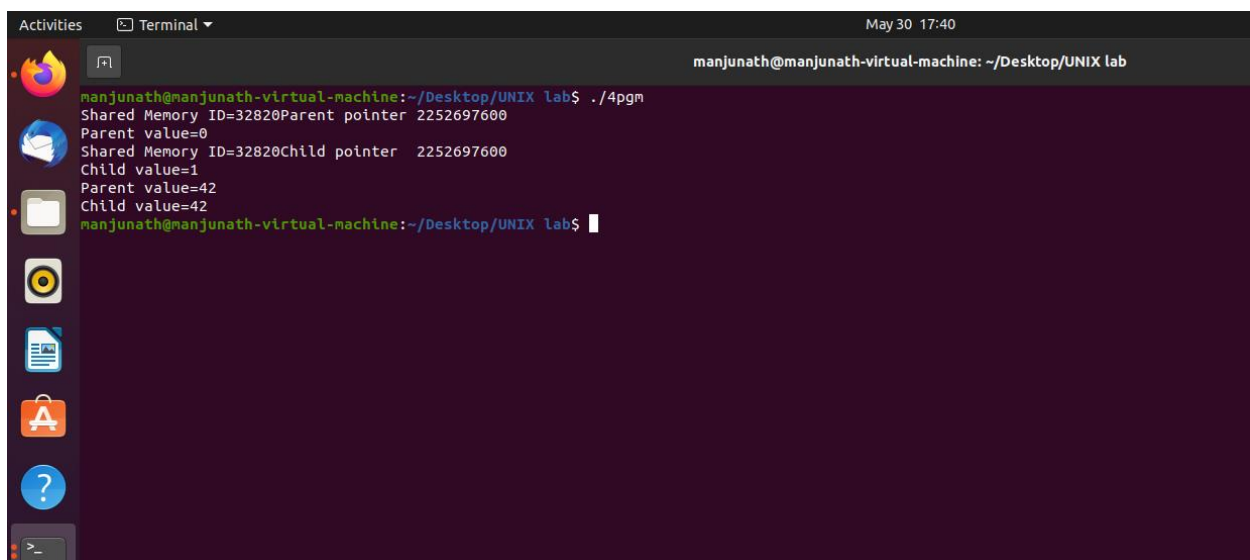
```
Parent value=0
```

```
Shared Memory ID=32820Child pointer 22526900
```

```
Child value=1
```

```
Parent value=42
```

```
Child value=42
```

A screenshot of a Linux terminal window. The window title is "Terminal" and the date/time is "May 30 17:40". The user is "manjunath" on a "manjunath-virtual-machine" at the directory "~/Desktop/UNIX lab". The terminal shows the execution of a program named "4pgm". The output of the program is displayed as follows:

```
manjunath@manjunath-virtual-machine:~/Desktop/UNIX lab$ ./4pgm
Shared Memory ID=32820Parent pointer 2252697600
Parent value=0
Shared Memory ID=32820Child pointer 2252697600
Child value=1
Parent value=42
Child value=42
manjunath@manjunath-virtual-machine:~/Desktop/UNIX lab$
```

PROGRAM-5

5. Write a C program that implements producer –consumer system with two processes using semaphores.

Header files used:

<fcntl.h>:The <fcntl.h> header shall define the following symbolic constants for the cmd argument used by fcntl(). The values shall be unique and shall be suitable for use in #if preprocessing directives.

F_DUPFD Duplicate file descriptor.

F_DUPFD_CLOEXEC Duplicate file descriptor with the close-on-exec flag FD_CLOEXEC set.

F_GETFD Get file descriptor flags

F_SETFD Set file descriptor flags.

F_GETFL Get file status flags and file access modes.

F_SETFL Set file status flags.

F_GETLK Get record locking information.

F_SETLK Set record locking information.

F_SETLKW Set record locking information; wait if blocked.

F_GETOWN Get process or process group ID to receive SIGURG signals.

F_SETOWN Set process or process group ID to receive SIGURG signals.

Program:

Producer :

```
#include<stdio.h>
```

```
#include<unistd.h>
```

```
#include<fcntl.h>
```

```
#define MAXSIZE 10
```

```
#define FIFO_NAME "myfifo"
```

```
int main()
```

```

{
int fifoid; int fd, n; char
*w; int open_mode;
system("clear");
w=(char
*)malloc(sizeof(char)*MAXSIZE);
open_mode=O_WRONLY;
fifoid=mkfifo(FIFO_NAME, 0755);
if(fifoid==-1)
{ printf("\nError: Named pipe cannot be Created\n"); exit(0); }
if( (fd=open(FIFO_NAME, open_mode)) < 0 )
{
printf("\nError: Named pipe cannot be opened\n"); exit(0);
}
while(1)
{
printf("\nProducer :"); fflush(stdin);
read(0, w, MAXSIZE);
n=write(fd, w, MAXSIZE);
if(n > 0)
printf("\nProducer sent: %s", w);
}

```

Consumer:

```

#include<stdio.h>
#include<unistd.h>
#include<fcntl.h>
#define MAXSIZE 10

```

```

#define FIFO_NAME

"myfifo

" int main()

{
int fifoid; int fd, n; char *r;
system("clear");
r=(char *)malloc(sizeof(char)*MAXSIZE); int
open_mode=O_RDONLY; if( (fd=open(FIFO_NAME, open_mode))
< 0 )
{
printf("\nError: Named pipe cannot be opened\n"); exit(0);
}
while(1)
{ n=read(fd, r, MAXSIZE); if(n > 0)
    printf("\nConsumer read: %s", r);
}
}

```

Output:

\$ cc producer.c -o producer	#first window
\$cc consumer.c -o consumer	# second window
\$./producer	#first window
\$./consumer	# second window
Producer:	
Producer sent: KGF	#first window
Consumer read: KGF	# second window
Producer sent: Rocky	#first window
Consumer read: Rocky	# second window

```
manjunath@manjunath-virtual-machine: ~/Desktop/UNIX lab
KGF
Producer :
Producer sent: KGF

Rocky
Producer :
Producer sent: Rocky
```

```
manjunath@manjunath-virtual-machine: ~/Desktop/UNIX lab
Consumer read: KGF
Consumer read: Rocky
█
```


PROGRAM 6

Demonstrate round robin scheduling algorithm and calculate average waiting time and average turn around time

Program:

```
#include<stdio.h>

int main()
{
    int i, limit, total = 0, x, counter = 0, time_quantum;
    int wait_time = 0, turnaround_time = 0, arrival_time[10], burst_time[10], temp[10];
    float average_wait_time, average_turnaround_time;
    printf("\nEnter Total Number of Processes:t=");
    scanf("%d", &limit);
    x = limit;
    for(i = 0; i < limit; i++)
    {
        printf("\nEnter Details of Process[%d]\n", i + 1);
        printf("Arrival Time:t=");
        scanf("%d", &arrival_time[i]);printf("\nBurst Time:t=");
        scanf("%d", &burst_time[i]);
        temp[i] = burst_time[i];
    }
    printf("\nEnter Time Quantum:t=");
    scanf("%d", &time_quantum);
    printf("\nProcess ID\t\tBurst Time\tTurnaround Time\tWaiting Time\n");
    for(total = 0, i = 0; x != 0;)
    {
        if(temp[i] <= time_quantum && temp[i] > 0)
```

```

{
total = total + temp[i];
temp[i] = 0;
counter = 1;
}
else if(temp[i] > 0)
{
temp[i] = temp[i] - time_quantum;
total = total + time_quantum;
}
if(temp[i] == 0 && counter == 1)
{
x--;
printf("\nProcess[%d]\t\t%d\t\t %d\t\t %d", i + 1, burst_time[i], total - arrival_time[i],
total - arrival_time[i] - burst_time[i]);
wait_time = wait_time + total - arrival_time[i] - burst_time[i];
turnaround_time = turnaround_time + total - arrival_time[i];
counter = 0;
}
if(i == limit - 1)
{
i = 0;
}
else if(arrival_time[i + 1] <= total)
{
i++;
}
else{

```

```

i = 0;
}
}
average_wait_time = wait_time * 1.0 / limit;
average_turnaround_time = turnaround_time * 1.0 / limit;
printf("\n\nAverage Waiting Time:t=%f", average_wait_time);
printf("\n\nAvg Turnaround Time:t=%f\n", average_turnaround_time);return 0;
}

```

Output :

Cc 6.c -o 6prog

./6prog

Enter Total Number of Processes: 4

Enter Details of Process[1]nArrival Time: 1

Burst Time: 3

Enter Details of Process[2]nArrival Time: 2

Burst Time: 5

Enter Details of Process[3]nArrival Time: 3

Burst Time: 6

Enter Details of Process[4]nArrival Time: 4

Burst Time: 8

Enter Time Quantum: 10

Process ID	Burst Time	Turnaround Time	Waiting Time
Process[1]	3	2	-1
Process[2]	5	6	1
Process[3]	6	11	5
Process[4]	8	18	10

Average Waiting Time: 3.750000

Average Turnaround Time: 9.250000

```
Activities Terminal May 30 17:42
manjunath@manjunath-virtual-machine: ~/Desktop/UNIX lab
manjunath@manjunath-virtual-machine:~/Desktop/UNIX lab$ cc 6.c -o 6pgm
manjunath@manjunath-virtual-machine:~/Desktop/UNIX lab$ ./6pgm
Enter Total Number of Processes: 4
Enter Details of Process[1]nArrival Time: 1
Burst Time: 3
Enter Details of Process[2]nArrival Time: 2
Burst Time: 5
Enter Details of Process[3]nArrival Time: 3
Burst Time: 6
Enter Details of Process[4]nArrival Time: 4
Burst Time: 8
Enter Time Quantum: 10
Process IDtBurst Timet Turnaround Timet Waiting Timen
Process[1] 3 2 -1
Process[2] 5 6 1
Process[3] 6 11 5
Process[4] 8 18 10
Average Waiting Time: 3.750000
Avg Turnaround Time: 9.250000manjunath@manjunath-virtual-machine:~/Desktop/UNIX lab$
```

PROGRAM 7

Implement Priority based scheduling algorithm and calculate average waiting time and turn around time

```
#include<stdio.h>

int main()
{
    int bt[20],p[20],wt[20],tat[20],pr[20],i,j,n,total=0,pos,temp,avg_wt,avg_tat;

    printf("Enter Total Number of Process:");
    scanf("%d",&n);

    printf("\nEnter Burst Time and Priority\n");
    for(i=0;i<n;i++)
    {
        printf("\nP[%d]\n",i+1);
        printf("Burst Time:");
        scanf("%d",&bt[i]);
        printf("Priority:");
        scanf("%d",&pr[i]);
        p[i]=i+1;
        //contains process number
    }

    //sorting burst time, priority and process number in ascending order using selection sort
    for(i=0;i<n;i++)
    {
        pos=i;
        for(j=i+1;j<n;j++)
        {
            if(pr[j]<pr[pos])
```

```

pos=j;
}temp=pr[i];
pr[i]=pr[pos];
pr[pos]=temp;
temp=bt[i];
bt[i]=bt[pos];
bt[pos]=temp;
temp=p[i];
p[i]=p[pos];
p[pos]=temp;
}
wt[0]=0;
//waiting time for first process is zero
//calculate waiting time
for(i=1;i<n;i++)
{
wt[i]=0;
for(j=0;j<i;j++)
wt[i]+=bt[j];
total+=wt[i];
}
avg_wt=total/n;
total=0;
//average waiting time
printf("\nProcess\tBurst Time \tWaiting Time\tTurnaround Time");
for(i=0;i<n;i++)
{ tat[i]=bt[i]+wt[i]; //calculate turnaround time
total+=tat[i];

```

```

printf("\nP[%d]\t\t %d\t\t %d\t\t %d",p[i],bt[i],wt[i],tat[i]);
}
avg_tat=total/n; //average turnaround time
printf("\n\nAverage Waiting Time=%d",avg_wt);
printf("\n\nAverage Waiting Time=%d",avg_wt);
printf("\n\nAverage Turnaround Time=%d\n",avg_tat);
return 0;
}

```

Output:

```
cc 7.c -o 7prog
```

```
./7prog
```

Enter Total Number of Process: 3

Enter Burst Time and Priority

P[1]

Burst Time:5

Priority:1

P[2]

Burst Time:3

Priority:2

P[3]

Burst Time:7

Priority:3

Process	Burst Time	Waiting Time	Turnaround Time
P[1]	5	0	5
P[2]	3	5	8
P[3]	7	8	15

Average Waiting Time=4

Average Turnaround Time=9

```
Activities Terminal May 30 17:58
manjunath@manjunath-virtual-machine: ~/Desktop/UNIX lab
manjunath@manjunath-virtual-machine:~/Desktop/UNIX lab$ ./7pgm
Enter Total Number of Process:3
Enter Burst Time and Priority
P[1]
Burst Time:5
Priority:1
P[2]
Burst Time:3
Priority:2
P[3]
Burst Time:7
Priority:3
Process    Burst Time    Waiting Time    Turnaround Time
P[1]       5             0              5
P[2]       3             5              8
P[3]       7             8              15
Average Waiting Time=4
Average Turnaround Time=9
manjunath@manjunath-virtual-machine:~/Desktop/UNIX lab$
```


PROGRAM 8

Act as sender to send data in message queues and receiver that reads data from message queue.

Header files:

sys/ipc.h : Interprocess communication access structure.

sys/msg.h : Message queue structures.

Commands:

ftok():The **ftok()** function uses the identity of the file named by the given pathname to generate a key_t type System V IPC key suitable for use with ,msgget(),semget().

msgget(): The msgget() system call returns the System V message queue identifier associated with the value of the key argument.

msgsnd():The *msgsnd()* function shall send a message to the queue associated with the message queue identifier specified by *msqid*.

msgrcv():The *msgrcv()* function reads a message from the queue associated with the message queue identifier specified by *msqid* and places it in the user-defined buffer pointed to by *msgp*.

msgctl():The **msgctl()** function allows the caller to control the message queue specified by the msqid parameter. A message queue is controlled by setting the cmd parameter to one of the following values: IPC_RMID (0x00000000) Remove the message queue identifier msqid from the system and destroy any messages on the message queue.

Program:

Sender:

```
#include <stdio.h>
```

```
#include <sys/ipc.h>
```

```
#include <sys/msg.h>
```

```
#define MAX 10
```

```

// structure for message queue
struct mesg_buffer {
    long mesg_type;
    char mesg_text[100];
} message;
int main()
{
    key_t key;
    int msgid;
    // ftok to generate unique key
    key = ftok("progfile", 65);
    // msgget creates a message queue and returns identifier
    msgid = msgget(key, 0666 | IPC_CREAT);
    message.mesg_type = 1;
    printf("Write Data : ");
    fgets(message.mesg_text, MAX, stdin);
    // msgsnd to send message
    msgsnd(msgid, &message, sizeof(message), 0);
    // display the message
    printf("Data send is : %s \n", message.mesg_text);
    return 0;
}

```

Receiver:

```

#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>

```

```

// structure for message queue
struct mesg_buffer {

    long mesg_type;
    char mesg_text[100];
} message;

int main()
{
    key_t key;
    int msgid;

    // ftok to generate unique key
    key = ftok("progfile", 65);

    // msgget creates a message queue
    // and returns identifier
    msgid = msgget(key, 0666 | IPC_CREAT);

    // msgrcv to receive message
    msgrcv(msgid, &message, sizeof(message), 1, 0);

    // display the message
    printf("Data Received is : %s \n", message.mesg_text);

    // to destroy the message queue
    msgctl(msgid, IPC_RMID, NULL);

    return 0;
}

```

Output:

```
$ cc sender.c -o sender          #first window
$cc receiver.c -o receiver      # second window
$./sender                       #first window
$./receiver                     # second window
```

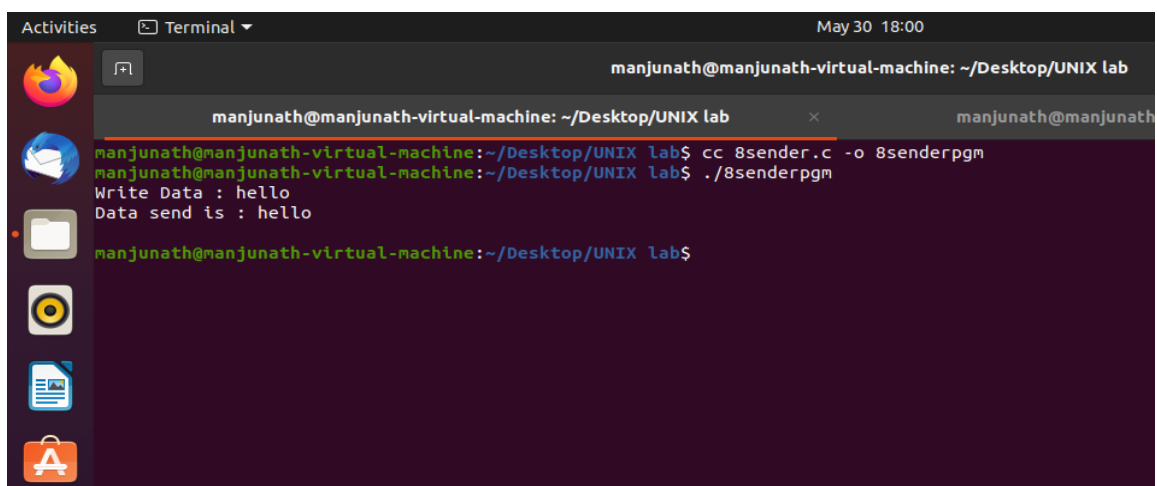
Sender: #first window

Write data: hello

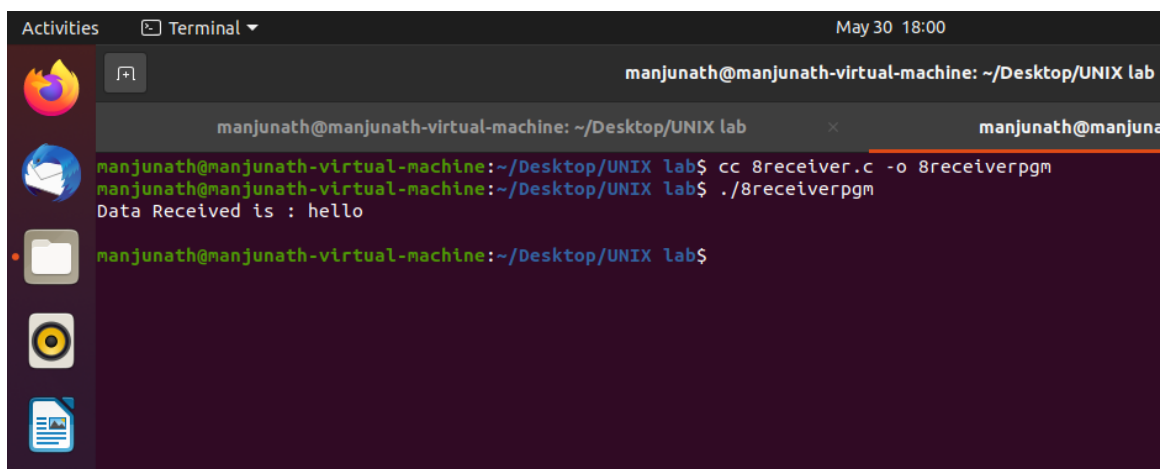
Data send is: hello

Receiver: # second window

Data Received is: hello



```
manjunath@manjunath-virtual-machine: ~/Desktop/UNIX lab
manjunath@manjunath-virtual-machine:~/Desktop/UNIX lab$ cc 8sender.c -o 8senderpgm
manjunath@manjunath-virtual-machine:~/Desktop/UNIX lab$ ./8senderpgm
Write Data : hello
Data send is : hello
manjunath@manjunath-virtual-machine:~/Desktop/UNIX lab$
```



```
manjunath@manjunath-virtual-machine: ~/Desktop/UNIX lab
manjunath@manjunath-virtual-machine:~/Desktop/UNIX lab$ cc 8receiver.c -o 8receiverpgm
manjunath@manjunath-virtual-machine:~/Desktop/UNIX lab$ ./8receiverpgm
Data Received is : hello
manjunath@manjunath-virtual-machine:~/Desktop/UNIX lab$
```

PROGRAM 9

Write a program where parent writes message to a pipe and child reads message from the pipe

Header files :

unistd.h: **unistd.h** is the name of the header file that provides access to the POSIX operating system API. On Unix-like systems, the interface defined by `unistd.h` is typically made up largely of system call wrapper functions such as fork, pipe and I/O primitives (read, write, close, etc.).

fcntl.h: File control options. The `<fcntl.h>` header defines the following requests and arguments for use by the functions [`fcntl\(\)`](#) and [`open\(\)`](#).

Commands:

mkfifo(): Make a FIFO special file. The `mkfifo()` function shall create a new FIFO special file named by the pathname pointed to by *path*. Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned, no FIFO shall be created, and *errno* shall be set to indicate the error.

Program:

Parent:

```
#include<stdio.h>

#include<unistd.h>

#include<fcntl.h>

#define MAXSIZE 10

#define FIFO_NAME "myfifo"

main()

{

int fifoid;

int fd, n;

char *w;

system("clear");
```

```

w=(char *)malloc(sizeof(char)*MAXSIZE);
int open_mode=O_WRONLY;
fifo=mkfifo(FIFO_NAME, 0755);
if(fifo==-1)
{
printf("\nError: Named pipe cannot be Created\n");
exit(0);
}
if( (fd=open(FIFO_NAME, open_mode)) < 0 )
{
printf("\nError: Named pipe cannot be opened\n");
exit(0);
}
while(1)
{
printf("\nProducer :");
fflush(stdin);
read(0, w, MAXSIZE);
n=write(fd, w, MAXSIZE);
if(n > 0)
printf("\nProducer sent: %s", w);
}
}

```

Child:

```

#include<stdio.h>
#include<unistd.h>
#include<fcntl.h>
#define MAXSIZE 10

```

```

#define FIFO_NAME "myfifo"

main()
{
    int fifoid;
    int fd, n;
    char *r;
    system("clear");
    r=(char *)malloc(sizeof(char)*MAXSIZE);
    int open_mode=O_RDONLY;
    if( (fd=open(FIFO_NAME, open_mode)) < 0 )
    {
        printf("\nError: Named pipe cannot be opened\n");
        exit(0);
    }
    while(1)
    {
        n=read(fd, r, MAXSIZE);
        if(n > 0)
            printf("\nConsumer read: %s", r);
    }
}

```

Output:

```

cc parent.c -o parent
./parent                #opens Parent Window

cc child.c -o child
./child                 #opens Child Window

```

Parent Window:

KGF

Rocky

Producer:

Producer sent:KGF

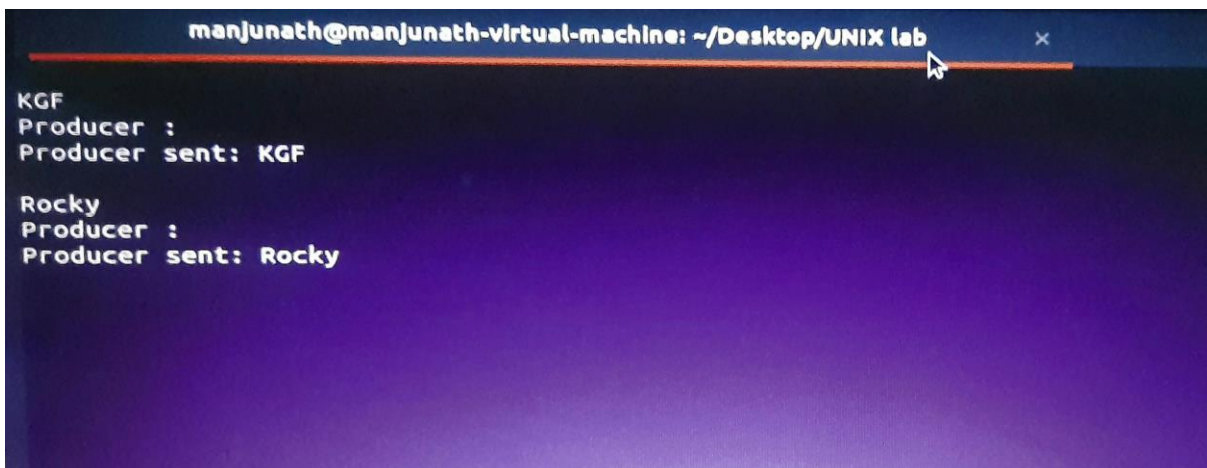
Producer:

Producer sent: Rocky

Child Window:

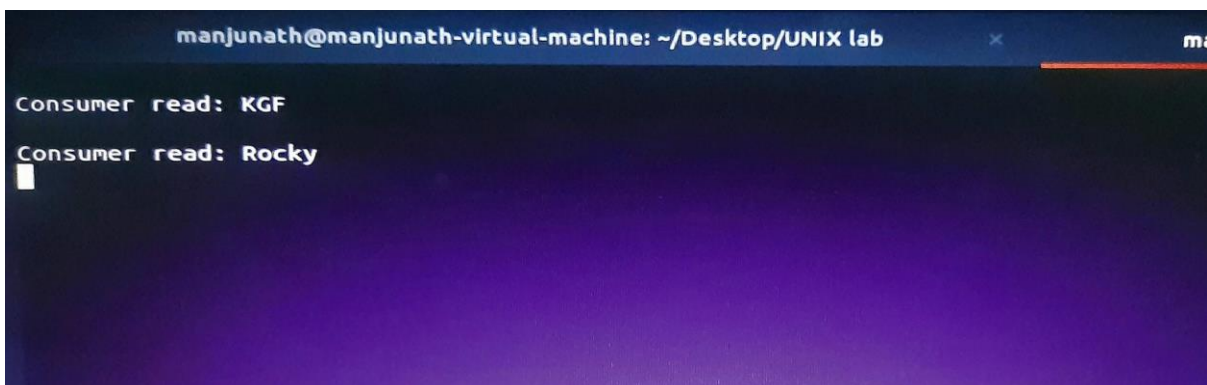
Consumer read:KGF

Consumer read: Rocky



```
manjunath@manjunath-virtual-machine: ~/Desktop/UNIX lab
KGF
Producer :
Producer sent: KGF

Rocky
Producer :
Producer sent: Rocky
```



```
manjunath@manjunath-virtual-machine: ~/Desktop/UNIX lab
Consumer read: KGF
Consumer read: Rocky
```


Program-10

Demonstrate setting up a simple Web Server and Host Website on Your Own Linux Computer.

- Prerequisites to Setup Web Server
- How to Setup A Web Server
- Install Apache2
- Install MySQL
- Install PHP

Prerequisites to Setup Web Server

To setup web server on your own Linux computer, we'll require the following three components to be installed –

- **Apache2:** apache2 is open-source HTTP server. It is still the most popular webserver used worldwide today.
- **php and php SQLite component:** PHP is a server-side scripting language. PHP and its component will help you to interact with a backend MySQL database for your website.
- **MySQL:** MySQL is a database solution in which you shall be storing your data in the table.

If you have installed the above components, you can skip this part and move to the next part.

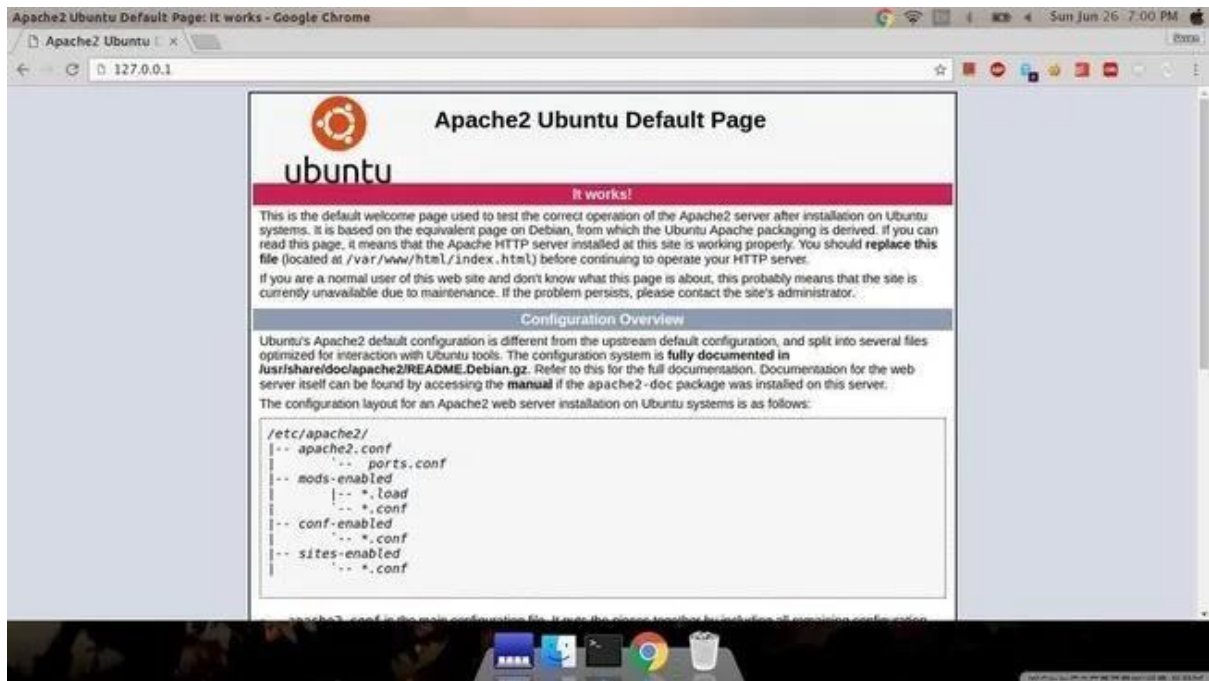
How to Setup A Web Server

Install Apache2

Apache is open source web-server software that powers much of the web today. It is maintained by apache-http-project. Explore more here: <https://httpd.apache.org/>

Open your terminal and type in commands –

```
sudo apt-get updatesudo apt-get install apache2
```

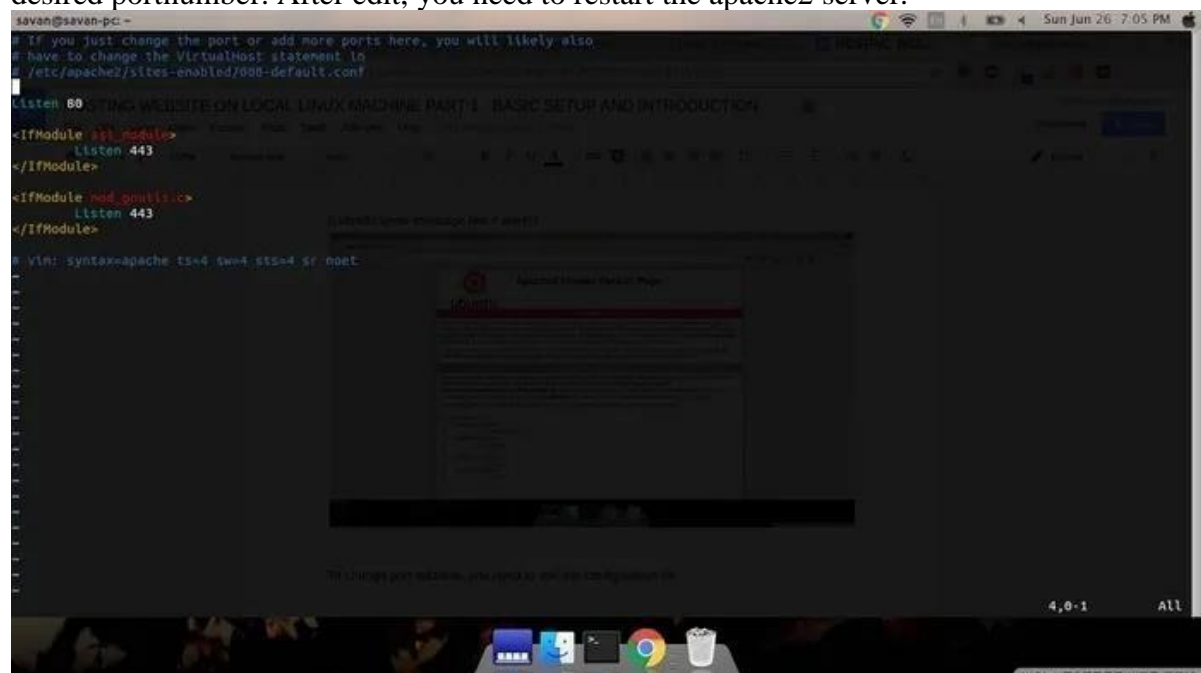


To check if apache2 is installed properly –
 sudo service apache2 restart

Open your web browser and open the link using ip-address of your server. If you are practicing locally, you can type in localhost or 127.0.0.1. By default, Apache runs on port 80 and hence you need not provide the port number in your browser.

127.0.0.1 or ip-address of your server. For example 198.162.12.52

It should show a message as it works! To change port address, you need to edit the configuration file at **/etc/apache2/ports.conf** and change the Listen 80 to your desired port number. After edit, you need to restart the apache2 server.



To restart apache2 web server -
sudo service apache2 restart

Install MySQL

MySQL is the database management solution that helps you to store and retrieve data in tables.

Install php5-mysql component.

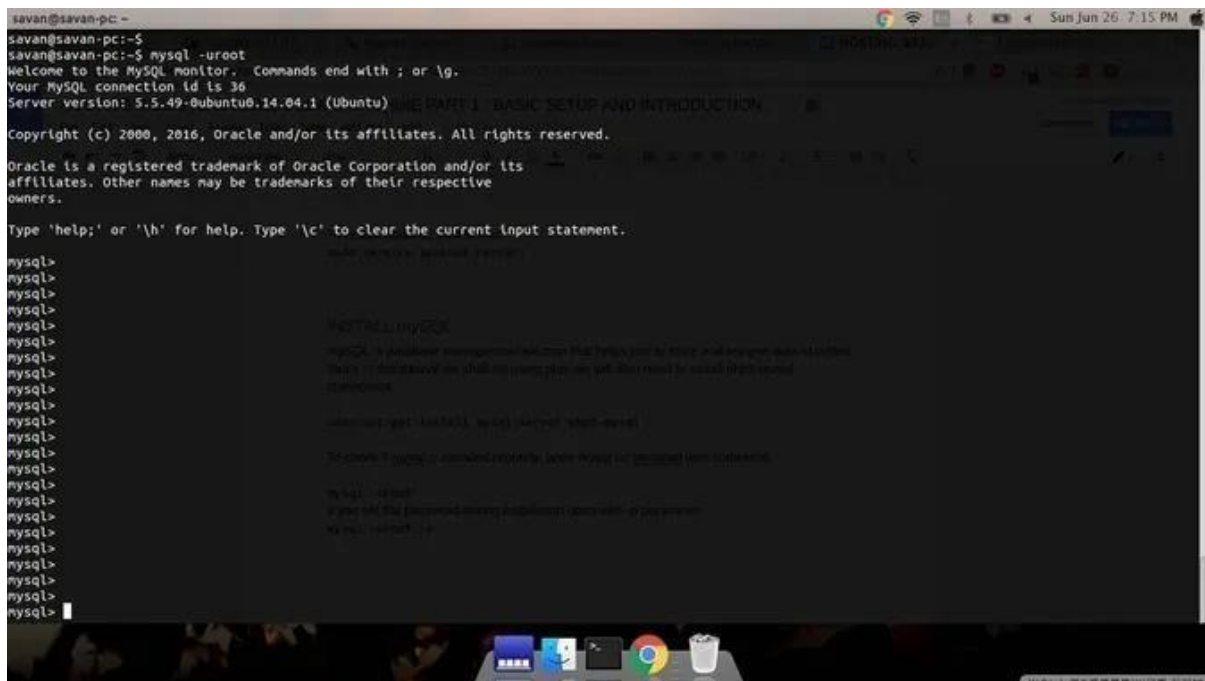
sudo apt-get install mysql-server php5-mysql

To check if MySQL is installed properly, open MySQL on terminal with command –

mysql -uroot

If you set the password during installation open with -p parameter –

mysql -uroot -p



Install PHP

PHP is an open-source web server scripting language. It is a back-end scripting language that will help you to interact with the MySQL database. For example, if you want to show the tabular employee list stored in your MySQL database in your website, with the help of PHP you can interact with MySQL, retrieve the employee list and render in html page. php5-mysql library helps you in this regard. PHP provides multiple auxiliary libraries for different needs. Php5-mysql is one among them.

To search the libraries that are available.

apt-cache search php5-

```
savan@savan-pc:~$ apt-cache search php5-
php5-cgi - server-side, HTML-embedded scripting language (CGI binary)
php5-cli - command-line interpreter for the php5 scripting language
php5-common - Common files for packages built from the php5 source
php5-curl - CURL module for php5
php5-dbg - Debug symbols for php5
php5-dev - Files for PHP5 module development
php5-gd - GD module for php5
php5-gmp - GMP module for php5
php5-json - JSON module for php5
php5-ldap - LDAP module for php5
php5-mysql - MySQL module for php5
php5-odbc - ODBC module for php5
php5-pgsql - PostgreSQL module for php5
php5-pspell - pspell module for php5
php5-readline - Readline module for php5
php5-recode - recode module for php5
php5-snmp - SNMP module for php5
php5-sqlite - SQLite module for php5
php5-tidy - tidy module for php5
php5-xmlrpc - XML-RPC module for php5
php5-xsl - XSL module for php5
libphp5-embed - HTML-embedded scripting language (Embedded SAPI library)
php5-adapter - Extension optimising the ADOdb database abstraction library
php5-apcu - APC User Cache for PHP 5
php5-enchant - Enchant module for php5
php5-exactimage - Fast image manipulation library (PHP bindings)
php5-fpm - server-side, HTML-embedded scripting language (FPM-CGI binary)
php5-gdcm - Grassroots DICOM PHP5 bindings
php5-gearman - PHP wrapper to libgearman
php5-geoip - GeoIP module for php5
php5-gnupg - wrapper around the gpgme library
php5-imagick - ImageMagick module for php5
```

To install PHP and php5-mysql

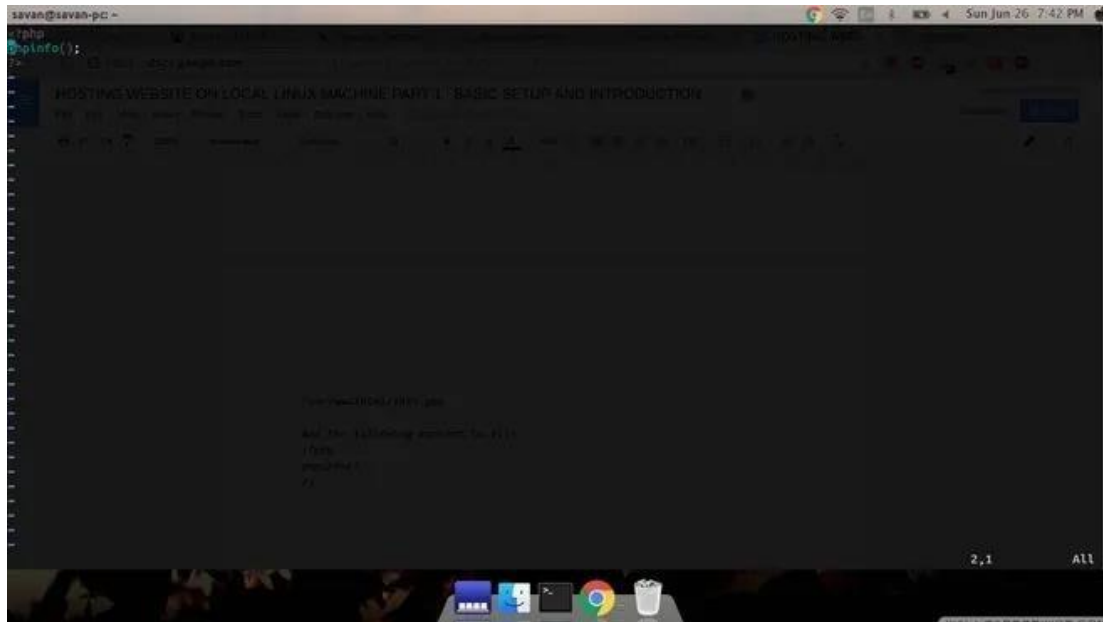
sudo apt-get install php5 libapache2-mod-php5 php5-mcrypt

sudo apt-get install php5-sqlite

```
savan@savan-pc:~$ sudo apt-get install php5 libapache2-mod-php5 php5-mcrypt
[sudo] password for savan:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  bbswitch-dkms dkms lib32gcc1 libauparse0 libcupyfreq0 libcudat1-352 libvdpaui
  linux-lts-wily-tools-4.2.0-30 linux-tools-4.2.0-30-generic nvidia-prime
  nvidia-settings psensor-common screen-resolution-extra
Use 'apt-get autoremove' to remove them.
The following extra packages will be installed:
  libmcrypt4 php5-cli php5-readline
Suggested packages:
  php-pear libmcrypt-dev mcrypt
The following NEW packages will be installed:
  libapache2-mod-php5 libmcrypt4 php5 php5-cli php5-mcrypt php5-readline
0 upgraded, 6 newly installed, 0 to remove and 3 not upgraded.
Need to get 4,465 kB of archives.
After this operation, 19.5 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://ln.archive.ubuntu.com/ubuntu/trusty-updates/main php5-cli amd64 5.5.9+dfsg-1ubuntu4.17 [2,162 kB]
12% [1 php5-cli 543 kB/2,162 kB 25%]
```

To check if PHP is installed correctly, make file /var/www/html/info.php and add the following content to this file -

```
<?php
phpinfo();
?>
```

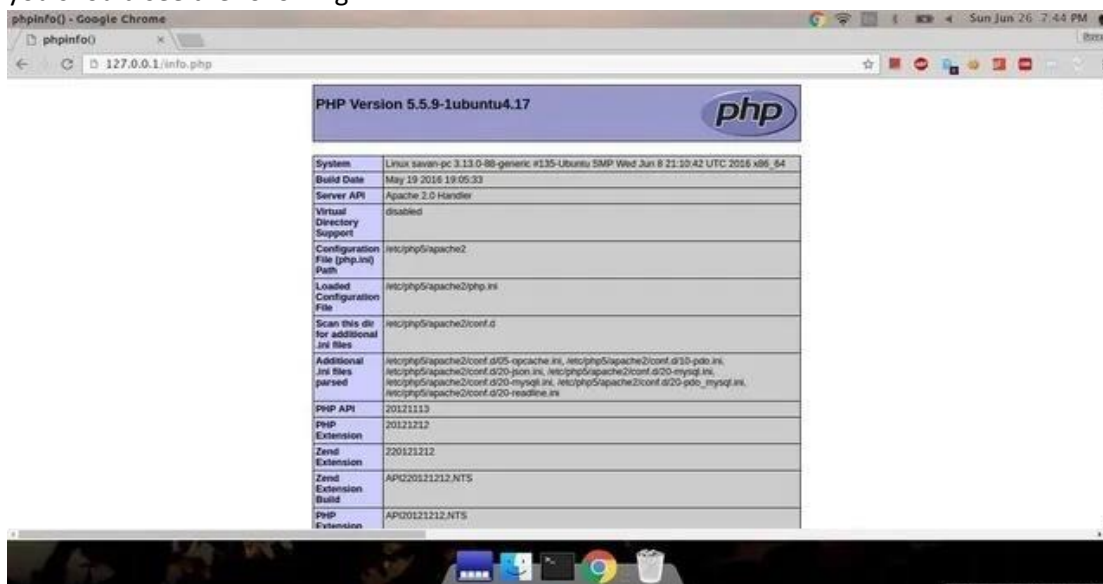


Restart apache2

sudo service apache2 restart

Open web browser and navigate to

127.0.0.1/info.php If you are using remote server replace ip with server's ip address. Upon success, you should see the following



Program-11

11(a). Create two threads using pthread, where both thread counts until 100 and joins later.

What is a Thread?

A thread is a single sequence stream within in a process. Because threads have some of the properties of processes, they are sometimes called lightweight processes.

What are the differences between process and thread?

Threads are not independent of one other like processes as a result threads shares with other threads their code section, data section and OS resources like open files and signals. But, like process, a thread has its own program counter (PC), a register set, and a stack space.

Why Multithreading?

Threads are popular way to improve application through parallelism. For example, in a browser, multiple tabs can be different threads. MS word uses multiple threads, one thread to format the text, other thread to process inputs, etc.

Threads operate faster than processes due to following reasons:

- 1) Thread creation is much faster.**
- 2) Context switching between threads is much faster.**
- 3) Threads can be terminated easily**
- 4) Communication between threads is faster.**

Can we write multithreading programs in C?

Unlike Java, multithreading is not supported by the language standard. POSIX Threads (or Pthreads) is a POSIX standard for threads. Implementation of pthread is available with gcc compiler.

Program:

```
#include<stdio.h>
```

```
#include<unistd.h>
```

```
#include<pthread.h>
```

```
#include<stdlib.h>
```

```
void* myturn(void *arg)
```

```
{
```

```

for(int i=1;i<=10;i++)
{
    sleep(1);
    printf("process 1: i=%d\n",i);
}
return NULL;
}

void yourturn()
{
    for(int i=1;i<=10;i++)
    {
        sleep(2);
        printf("process 2: j=%d\n",i);
    }
}

int main()
{
    pthread_t newthread;
    pthread_create(&newthread,NULL,myturn,NULL);
    yourturn();
    pthread_join(newthread,NULL);
    return 0;
}

```

Output:

```
manjunath@manjunath-virtual-machine: ~/Desktop/U
manjunath@manjunath-virtual-machine:~/Desktop/UNIX lab$ cc 11a.c -o 11apgm -pthread
manjunath@manjunath-virtual-machine:~/Desktop/UNIX lab$ ./11apgm
process 1: i=1
process 2: j=1
process 1: i=2
process 1: i=3
process 2: j=2
process 1: i=4
process 1: i=5
process 2: j=3
process 1: i=6
process 1: i=7
process 2: j=4
process 1: i=8
process 1: i=9
process 2: j=5
process 1: i=10
process 2: j=6
process 2: j=7
process 2: j=8
process 2: j=9
process 2: j=10
manjunath@manjunath-virtual-machine:~/Desktop/UNIX lab$
```


11(b). Create two threads using pthreads. Here main thread creates 5 and other threads for 5 times and each new thread print "Hello World" message with its thread number.

Thread Identification

Just as a process is identified through a process ID, a thread is identified by a thread ID. But interestingly, the similarity between the two ends here.

A process ID is unique across the system where as a thread ID is unique only in context of a single process.

A process ID is an integer value but the thread ID is not necessarily an integer value. It could well be a structure

A process ID can be printed very easily while a thread ID is not easy to print.

Thread Creation

Normally when a program starts up and becomes a process, it starts with a default thread. So we can say that every process has at least one thread of control. A process can create extra threads using the following function :

```
#include <pthread.h>
```

```
int pthread_create(pthread_t *restrict tidp, const pthread_attr_t *restrict attr, void  
*(*start_rtn)(void), void *restrict arg)
```

Program:

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <unistd.h>
```

```
#include <pthread.h>
```

```
// The function to be executed by all threads
```

```
void *myNewThread(void *vargp){
```

```
    printf("Hello world\n");
```

```
}
```

```

void *myThreadFun(void *vargp)
{

    // Store the value argument passed to this thread
    int *myid = (int *)vargp;
    printf("%ld %ld ", pthread_self(), *myid);
    int maint = pthread_self();
    if(maint == (*myid) )
    {
        int i,j;
        printf("main thread encountered\n");
        pthread_t nid;
        for (i = 0; i < 5; i++)
            for(j =0;j<5;j++)
                pthread_create(&nid, NULL, myNewThread, (void *)&nid);

    }
}

```

```

int main()
{
    int i;
    pthread_t tid;

    // Let us create three threads
    for (i = 0; i < 2; i++)
        pthread_create(&tid, NULL, myThreadFun, (void *)&tid);
}

```

```
pthread_exit(NULL);  
return 0;  
}
```

Output:

```
.cc 11b.c -o 11bpgm -pthread
```

```
./11bpgm
```

```
1396266756408 1527351040 139626619176704 1527351040 main thread encountered
```

```
Hello world
```

```
Hello world
```

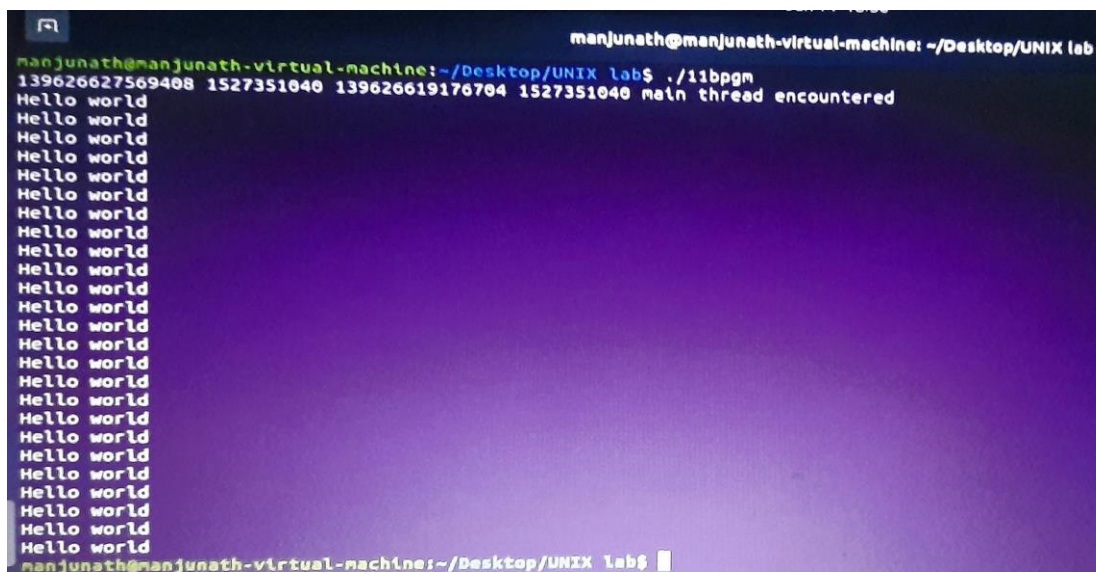
```
Hello world
```

```
Hello world
```

```
.
```

```
.
```

```
Hello world
```



```
manjunath@manjunath-virtual-machine: ~/Desktop/UNIX lab$  
manjunath@manjunath-virtual-machine:~/Desktop/UNIX lab$ ./11bpgm  
139626627569408 1527351040 139626619176704 1527351040 main thread encountered  
Hello world  
Hello world  
Hello world  
Hello world  
Hello world  
Hello world  
Hello world  
Hello world  
Hello world  
Hello world  
Hello world  
Hello world  
Hello world  
Hello world  
Hello world  
Hello world  
Hello world  
Hello world  
Hello world  
Hello world  
Hello world  
Hello world  
Hello world  
manjunath@manjunath-virtual-machine:~/Desktop/UNIX lab$
```

Program-12

Using Socket APIs establish communication between remote and local processes.

A socket is a communications connection point (endpoint) that you can name and address in a network. Socket programming shows how to use socket APIs to establish communication links between remote and local processes.

The processes that use a socket can reside on the same system or different systems on different networks. Sockets are useful for both stand-alone and network applications. Sockets allow you to exchange information between processes on the same machine or across a network, distribute work to the most efficient machine, and they easily allow access to centralized data. Socket application program interfaces (APIs) are the network standard for TCP/IP. A wide range of operating systems support socket APIs.

The server program above performs the classic four-step to ready itself for client requests and then to accept individual requests. Each step is named after a system function that the server calls:

socket(...): get a file descriptor for the socket connection

bind(...): bind the socket to an address on the server's host

listen(...): listen for client requests

accept(...): accept a particular client request

Program :

Socket Server Example

```
#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
```

```

#include<errno.h>

#include<string.h>

#include<sys/types.h>

#include<time.h>

int main(int argc, char *argv[])
{ int listenfd = 0, connfd = 0;

struct sockaddr_in serv_addr; char sendBuff[1025];

time_t ticks;

listenfd = socket(AF_INET, SOCK_STREAM, 0);

memset(&serv_addr, '0', sizeof(serv_addr));

memset(sendBuff, '0', sizeof(sendBuff));

serv_addr.sin_family = AF_INET;

serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);

serv_addr.sin_port = htons(5000);

bind(listenfd, (struct sockaddr*)&serv_addr, sizeof(serv_addr));

listen(listenfd, 10);

while(1)
{
connfd = accept(listenfd, (struct sockaddr*)NULL, NULL);

ticks = time(NULL); snprintf(sendBuff, sizeof(sendBuff), "%.24s\r\n", ctime(&ticks));

write(connfd, sendBuff, strlen(sendBuff));

close(connfd);

sleep(1);
}
}

```

Socket Client Example

```
#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<errno.h>
#include<string.h>
#include<sys/types.h>
#include<time.h>
int main(int argc, char *argv[])
{
    int sockfd = 0, n = 0;
    char recvBuff[1024];
    struct sockaddr_in serv_addr;
    if(argc != 2)
    {
        printf("\n Usage: %s \n",argv[0]);
        return 1;
    }
    memset(recvBuff, '0',sizeof(recvBuff));
    if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        printf("\n Error : Could not create socket \n");
        return 1;
    }
    memset(&serv_addr, '0', sizeof(serv_addr));
```

```

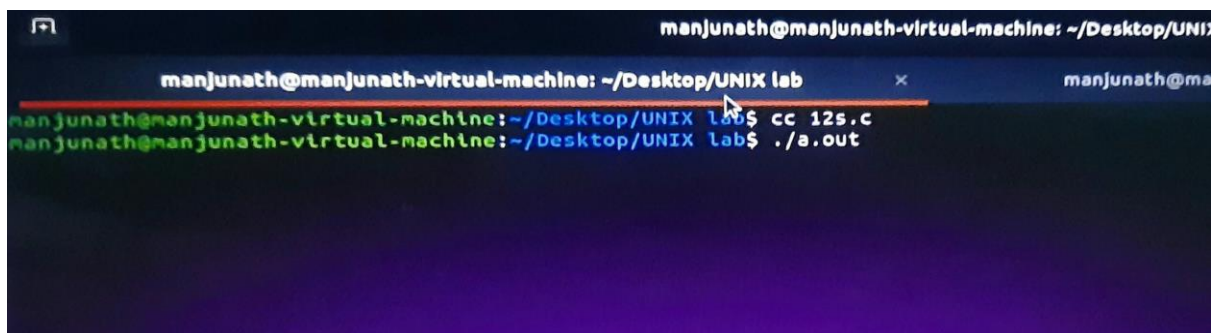
serv_addr.sin_family = AF_INET;
serv_addr.sin_port = htons(5000);
if(inet_pton(AF_INET, argv[1], &serv_addr.sin_addr)<=0)
{
printf("\n inet_pton error occurred\n");
return 1;
}
if( connect(sockfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
{
printf("\n Error : Connect Failed \n");
return 1;
}
while ( (n = read(sockfd, recvBuff, sizeof(recvBuff)-1)) > 0)
{
recvBuff[n] = 0;
if(fputs(recvBuff, stdout) == EOF)
{
printf("\n Error : Fputs error\n");
}
}
if(n < 0)
{
printf("\n Read error \n");
}
return 0;
}

```

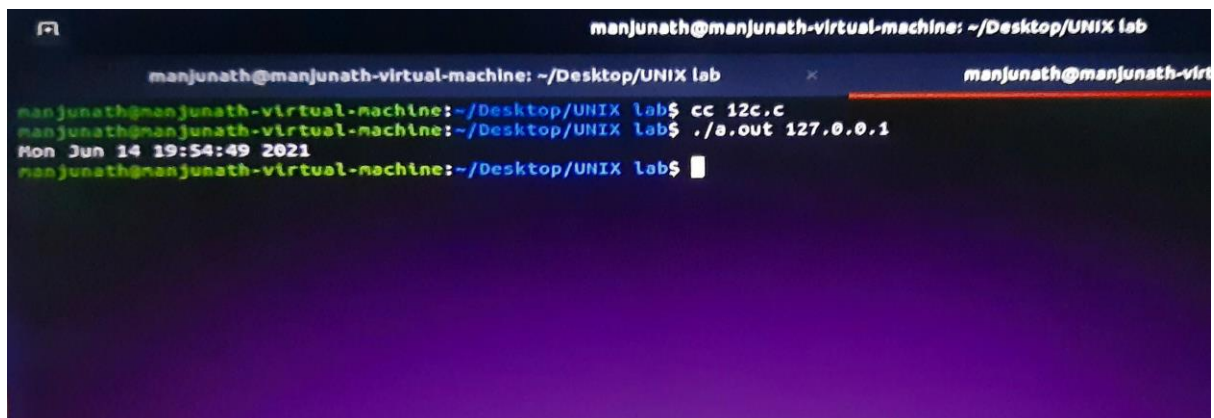
OUTPUT :

\$./newsy 127.0.0.1

Thu May 27 22:22:14 2021



```
manjunath@manjunath-virtual-machine: ~/Desktop/UNIX lab
manjunath@manjunath-virtual-machine:~/Desktop/UNIX lab$ cc 12s.c
manjunath@manjunath-virtual-machine:~/Desktop/UNIX lab$ ./a.out
```



```
manjunath@manjunath-virtual-machine: ~/Desktop/UNIX lab
manjunath@manjunath-virtual-machine:~/Desktop/UNIX lab$ cc 12c.c
manjunath@manjunath-virtual-machine:~/Desktop/UNIX lab$ ./a.out 127.0.0.1
Mon Jun 14 19:54:49 2021
manjunath@manjunath-virtual-machine:~/Desktop/UNIX lab$
```