

LAB - 1

Task:

1. Create multiple directories and subdirectories.

Solutions:

```
kabeerahmed@kali-linux: ~/LabWork/Task2
kabeerahmed@kali-linux:~/LabWork$ mkdir {Task1,Task2,Task3}
kabeerahmed@kali-linux:~/LabWork$ ls
Task1 Task2 Task3
kabeerahmed@kali-linux:~/LabWork$ mkdir -p Task1/AddtheFile
kabeerahmed@kali-linux:~/LabWork$ ls Task1
AddtheFile
kabeerahmed@kali-linux:~/LabWork$ cd Task2
kabeerahmed@kali-linux:~/LabWork/Task2$ mkdir SchedulerFile
kabeerahmed@kali-linux:~/LabWork/Task2$ ls
SchedulerFile
kabeerahmed@kali-linux:~/LabWork/Task2$
```

2. Search different options of *rmdir* command using man command.

Code:

```
kabeerahmed@kali-linux: ~/LabWork/Task2
kabeerahmed@kali-linux:~/LabWork/Task2$ man mkdir
kabeerahmed@kali-linux:~/LabWork/Task2$
```

Output:

```
MKDIR(1) User Commands
NAME
  mkdir - make directories

SYNOPSIS
  mkdir [OPTION]... DIRECTORY...

DESCRIPTION
  Create the DIRECTORY(ies), if they do not already exist.

  Mandatory arguments to long options are mandatory for short options too.

  -m, --mode=MODE
        set file mode (as in chmod), not a=rwx - umask
  -p, --parents
        no error if existing, make parent directories as needed
  -v, --verbose
        print a message for each created directory
  -Z
        set SELinux security context of each created directory to the default type
  --context[=CTX]
        like -Z, or if CTX is specified then set the SELinux or SMACK security context to CTX
  --help
        display this help and exit
  --version
        output version information and exit

AUTHOR
  Written by David MacKenzie.

REPORTING BUGS
  GNU coreutils online help: <https://www.gnu.org/software/coreutils/>
  Report mkdir translation bugs to <https://translationproject.org/team/>
  Manual page mkdir(1) line 1 (press h for help or q to quit)
```

LAB – 2

Task:

1. Create multiple files and perform all commands.
2. Remove non empty directory.

```
kabeerahmed@kali-linux:~/LabWork$ ls
Task2 task3.c taskout
kabeerahmed@kali-linux:~/LabWork$ rm -d Task2
kabeerahmed@kali-linux:~/LabWork$ ls
task3.c taskout
kabeerahmed@kali-linux:~/LabWork$
```

3. date (for Current Date).

```
kabeerahmed@kali-linux:~/LabWork$ date
Wed 26 Jan 2022 07:19:46 AM EST
kabeerahmed@kali-linux:~/LabWork$
```

4. ps

```
kabeerahmed@kali-linux:~/LabWork$ ps
  PID TTY          TIME CMD
 1812 pts/0    00:00:00 bash
 1834 pts/0    00:00:00 ps
kabeerahmed@kali-linux:~/LabWork$
```

5. finger

```
kabeerahmed@kali-linux:~/LabWork$ finger
Login      Name      Tty      Idle   Login Time   Office      Office Phone
kabeerahmed Kabeer Ahmed tty2      9      Jan 28 04:41 (tty2)
```

6. cal (For Calendar).

```
kabeerahmed@kali-linux:~/LabWork$ cal
  January 2022
Su Mo Tu We Th Fr Sa
                1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31
kabeerahmed@kali-linux:~/LabWork$
```

7. free

```
kabeerahmed@kali-linux: ~/LabWork
kabeerahmed@kali-linux:~/LabWork$ free
              total        used        free      shared  buff/cache   available
Mem:      2039936        911028        374780         48584         754128         931428
Swap:              0              0              0
```

8. Create a file name <yearfile> which displays the contents of current year.

```
kabeerahmed@kali-linux: ~/LabWork
kabeerahmed@kali-linux:~/LabWork$ date +%Y > yearfile
kabeerahmed@kali-linux:~/LabWork$ cat yearfile
2022
kabeerahmed@kali-linux:~/LabWork$
```

9. Display calendar and date simultaneously.

```
kabeerahmed@kali-linux:~/LabWork$ date && cal
Wed 26 Jan 2022 07:36:03 AM EST
    January 2022
Su Mo Tu We Th Fr Sa
                1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31
kabeerahmed@kali-linux:~/LabWork$
```

10. df ./

```
kabeerahmed@kali-linux: ~/LabWork
kabeerahmed@kali-linux:~/LabWork$ df ./
Filesystem      1K-blocks      Used Available Use% Mounted on
/dev/sda1      80110140 13539696  62458060  18% /
kabeerahmed@kali-linux:~/LabWork$
```

11. df -k ./

```
kabeerahmed@kali-linux: ~/LabWork
kabeerahmed@kali-linux:~/LabWork$ df -k
Filesystem      1K-blocks    Used Available Use% Mounted on
udev              995820         0     995820   0% /dev
tmpfs             203996      6184     197812   4% /run
/dev/sda1        80110140 13539700  62458056  18% /
tmpfs            1019968         0     1019968   0% /dev/shm
tmpfs              5120         0         5120   0% /run/lock
tmpfs            1019968         0     1019968   0% /sys/fs/cgroup
tmpfs             203992      9268     194724   5% /run/user/1000
kabeerahmed@kali-linux:~/LabWork$
```

12. du -s

```
kabeerahmed@kali-linux:~/LabWork$ du -s *
4      task1
4      task3.c
20     taskout
4      yearfile
kabeerahmed@kali-linux:~/LabWork$
```

Assignment

1. Using ls – List the contents of a directory shown, from these contents explain any ten switches with an example.

```
kabeerahmed@kali-linux: ~/Lab
kabeerahmed@kali-linux:~/LabWork$ ls
file01 file02 task1 task3.c taskout yearfile
```

-a do not ignore entries starting with.

```
kabeerahmed@kali-linux:~/LabWork$ ls -a
. .. file01 file02 task1 task3.c taskout yearfile
```

-A do not list implied. and ..

```
kabeerahmed@kali-linux:~/LabWork$ ls -A
file01 file02 task1 task3.c taskout yearfile
```

-f do not sort

```
kabeerahmed@kali-linux:~/LabWork$ ls -f
. task1 .. yearfile taskout file02 task3.c file01
```

-g like -l, but do not list owner

```
kabeerahmed@kali-linux:~/LabWork$ ls -g
total 40
-rw-r--r-- 1 kabeerahmed    11 Jan 26 07:59 file01
-rw-r--r-- 1 kabeerahmed    11 Jan 26 07:59 file02
drwxr-xr-x 2 kabeerahmed 4096 Jan 26 07:46 task1
-rw-r--r-- 1 kabeerahmed   241 Jan 22 13:46 task3.c
-rwxr-xr-x 1 kabeerahmed 16736 Jan 22 13:52 taskout
-rw-r--r-- 1 kabeerahmed     5 Jan 26 07:33 yearfile
```

-i print the index number of each file

```
kabeerahmed@kali-linux:~/LabWork$ ls -i
1582343 file01 1582342 task1 1582317 taskout
1582344 file02 1582340 task3.c 1582341 yearfile
```

-m fill width with a comma separated list of entries

```
kabeerahmed@kali-linux:~/LabWork$ ls -m
file01, file02, task1, task3.c, taskout, yearfile
```

-n like -l, but list numeric user and group IDs

```
kabeerahmed@kali-linux:~/LabWork$ ls -n
total 40
-rw-r--r-- 1 1000 1000    11 Jan 26 07:59 file01
-rw-r--r-- 1 1000 1000    11 Jan 26 07:59 file02
drwxr-xr-x 2 1000 1000 4096 Jan 26 07:46 task1
-rw-r--r-- 1 1000 1000   241 Jan 22 13:46 task3.c
-rwxr-xr-x 1 1000 1000 16736 Jan 22 13:52 taskout
-rw-r--r-- 1 1000 1000     5 Jan 26 07:33 yearfile
```

-p append / indicator to directories

```
kabeerahmed@kali-linux:~/LabWork$ ls -p
file01 file02 task1/ task3.c taskout yearfile
```

-r reverse order while sorting

```
kabeerahmed@kali-linux:~/LabWork$ ls -r
yearfile taskout task3.c task1 file02 file01
```

-l use a long listing format

```
kabeerahmed@kali-linux:~/LabWork$ ls -l
total 40
-rw-r--r-- 1 kabeerahmed kabeerahmed    11 Jan 26 07:59 file01
-rw-r--r-- 1 kabeerahmed kabeerahmed    11 Jan 26 07:59 file02
drwxr-xr-x 2 kabeerahmed kabeerahmed 4096 Jan 26 07:46 task1
-rw-r--r-- 1 kabeerahmed kabeerahmed   241 Jan 22 13:46 task3.c
-rwxr-xr-x 1 kabeerahmed kabeerahmed 16736 Jan 22 13:52 taskout
-rw-r--r-- 1 kabeerahmed kabeerahmed     5 Jan 26 07:33 yearfile
```

2. Write in detail about “w” command and also the difference between *who* and *w* command.

‘w’ command:

w command in Linux is used to show who is logged on and what they are doing. This command shows the information about the users currently on the machine and their processes. The header shows, in this order, the current time, how long the system has been running, how many users are currently logged on, and the system load averages for the past 1, 5, and 15 minutes. The following entries are displayed for each user: login name, the tty name, the remote host, login time, idle time, JCPU, PCPU, and the command line of their current process. The JCPU time is the time used by all processes attached to the tty. It does not include past background jobs but does include currently running background jobs. The PCPU time is the time used by the current process, named in the “what” field.

‘w’ vs ‘who’ commands:

‘who’ command is used to determine when the system has booted last time, a list of logged-in users, and the system's current run level. ‘w’ command displays user information like user id and activities on the system. It also gives the knowledge of the system's running time along with the system load average.

```
kabeerahmed@kali-linux:~/LabWork$ who
kabeerahmed tty2          2022-01-26 07:07 (tty2)
kabeerahmed@kali-linux:~/LabWork$ w
 08:41:23 up  1:37,  1 user,  load average: 0.24, 0.05, 0.02
USER      TTY      FROM            LOGIN@   IDLE   JCPU   PCPU WHAT
kabeerah tty2      tty2            07:07    1:37m  0.04s  0.03s /usr/libexec/gn
```

3. Write the difference between the following: >, <, >>.

The “>” is an output operator that overwrites the existing file, while “>>” is also an output operator that appends the data in an already existing file. Both operators are often used to modify the files in Linux.

Just as the output of a command can be redirected to a file, so can the input of a command be redirected from a file. As the greater-than character > is used for output redirection, the less-than character < is used to redirect the input of a command.

Example of ‘>’ operator

```
kabeerahmed@kali-linux:~/LabWork$ who > user
kabeerahmed@kali-linux:~/LabWork$ cat user
kabeerahmed tty2          2022-01-26 07:07 (tty2)
```

Example of ‘>>’ operator

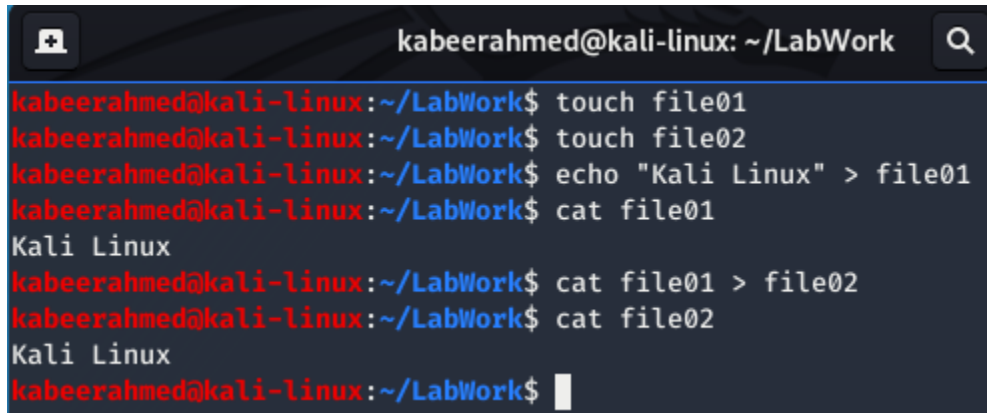
```
kabeerahmed@kali-linux:~/LabWork$ echo "Kali Linux" >>user
kabeerahmed@kali-linux:~/LabWork$ cat user
kabeerahmed tty2          2022-01-26 07:07 (tty2)
Kali Linux
```

Example of ‘<’ operator

```
kabeerahmed@kali-linux:~/LabWork$ wc -l < user
2
```

Assignment

1. Copy text from one file and paste it in another file using *cat* command.



```
kabeerahmed@kali-linux: ~/LabWork
kabeerahmed@kali-linux:~/LabWork$ touch file01
kabeerahmed@kali-linux:~/LabWork$ touch file02
kabeerahmed@kali-linux:~/LabWork$ echo "Kali Linux" > file01
kabeerahmed@kali-linux:~/LabWork$ cat file01
Kali Linux
kabeerahmed@kali-linux:~/LabWork$ cat file01 > file02
kabeerahmed@kali-linux:~/LabWork$ cat file02
Kali Linux
kabeerahmed@kali-linux:~/LabWork$
```

The image shows a terminal window with a dark background. The title bar at the top reads 'kabeerahmed@kali-linux: ~/LabWork'. The terminal content shows a series of commands and their outputs: 'touch file01' and 'touch file02' are executed without output. 'echo "Kali Linux" > file01' is followed by 'cat file01', which outputs 'Kali Linux'. Then 'cat file01 > file02' is executed, followed by 'cat file02', which also outputs 'Kali Linux'. The prompt 'kabeerahmed@kali-linux:~/LabWork\$' is shown at the end with a cursor.

LAB – 3

1. Write few lines about given Program and what is the output of same program?

```
#include<stdio.h>
main(int arc,char*ar[])
{
int pid; char s[100];
pid=fork();
if(pid<0)
printf("error");
else if(pid>0)
{
wait(NULL);
printf("\n Parent Process:\n");
printf("\n\tParent Process id:%d\t\n",getpid());
execlp("cat","cat",ar[1],(char*)0);
error("can't execute cat %s",ar[1]);
}
else
{
printf("\nChild process:");
printf("\n\tChildprocess parent id:\t %d",getppid());
sprintf(s,"\n\tChild process id :%d",getpid());
write(1,s,strlen(s));
printf(" ");
printf(" ");
printf(" ");
execvp(ar[2],&ar[2]);
error("can't execute %s",ar[2]);
}
}
```

Description:

The above program uses the **fork()** system call to create a child process. The program then checks for the return value of fork if its value is **greater than zero** this means the parent is going to execute. Else the child will execute the program respectively. This program also uses the **exec** system call to use **cat** command to print the contents of the specified file.

Output:

```
kabeerahmed@kali-linux:~/LabWork$ ./task2out file01

Child process:

      Child process id :      1753
Parent Process:

      Parent Process id:1752
Kali Linux
```

Assignment

1. Create multiple Childs using the fork() command.

Code:

```
GNU nano 4.5 Task2_1.c M
#include<stdio.h>

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

Output:

```
kabeerahmed@kali-linux:~/LabWork$ ./task2_1out
[child] pid 1816 from [parent] pid 1811
[child] pid 1815 from [parent] pid 1811
[child] pid 1814 from [parent] pid 1811
[child] pid 1813 from [parent] pid 1811
[child] pid 1812 from [parent] pid 1811
```

LAB - 4

TASK:

1. Describe all the versions of exec system call.

```

kabeerahmed@kali-linux: ~
DESCRIPTION
The exec() family of functions replaces the current process image with
a new process image. The functions described in this manual page are
front-ends for execve(2). (See the manual page for execve(2) for fur-
ther details about the replacement of the current process image.)

The initial argument for these functions is the name of a file that is
to be executed.

The functions can be grouped based on the letters following the "exec"
prefix.

l - execl(), execvp(), execle()
The const char *arg and subsequent ellipses can be thought of as arg0,
arg1, ..., argn. Together they describe a list of one or more pointers
to null-terminated strings that represent the argument list available
to the executed program. The first argument, by convention, should
point to the filename associated with the file being executed. The
list of arguments must be terminated by a null pointer, and, since
these are variadic functions, this pointer must be cast (char *) NULL.

By contrast with the 'l' functions, the 'v' functions (below) specify
the command-line arguments of the executed program as a vector.

v - execv(), execvp(), execvpe()
The char *const argv[] argument is an array of pointers to null-termi-
nated strings that represent the argument list available to the new
program. The first argument, by convention, should point to the file-
name associated with the file being executed. The array of pointers
must be terminated by a null pointer.

e - execl(), execvp()
The environment of the caller is specified via the argument envp. The
envp argument is an array of pointers to null-terminated strings and
must be terminated by a null pointer.

All other exec() functions (which do not include 'e' in the suffix)
take the environment for the new process image from the external vari-
able environ in the calling process.

p - execlp(), execvp(), execvpe()
These functions duplicate the actions of the shell in searching for an
executable file if the specified filename does not contain a slash (/)
character. The file is sought in the colon-separated list of directory
pathnames specified in the PATH environment variable. If this variable
isn't defined, the path list defaults to a list that includes the di-
rectories returned by confstr(CS_PATH) (which typically returns the
value "/bin:/usr/bin") and possibly also the current working directory;
see NOTES for further details.

```

2. Write a program that will call another program from child process.

Code:

```
kabeerahmed@kali-linux: ~/LabWork
GNU nano 4.5 task3.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>

void main()
{
    int pt;
    if(pt==0){
        static char *argv[]={ "echo", "Hello World.", NULL };
        execv("/bin/echo", argv);
    }
    else {
        printf("Parent Process");
    }
}
```

Output:

```
kabeerahmed@kali-linux: ~/LabWork
kabeerahmed@kali-linux:~/LabWork$ nano task3.c
kabeerahmed@kali-linux:~/LabWork$ gcc task3.c -o taskout
kabeerahmed@kali-linux:~/LabWork$ ./taskout
Hello World.
kabeerahmed@kali-linux:~/LabWork$
```

Assignment

1. Use *execvp* function in a C Program.

Code:

```
kabeerahmed@kali-linu
GNU nano 4.5 task3_1
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int main()
{
    char *args[]={ "pwd", NULL };
    printf("execvp command:\n");
    execvp(args[0],args);
    printf("This will not print!!!");
    return 0;
}
```

Output:

```
kabeerahmed@kali-linux:~/LabWork$ ./task3_1out
execvp command:
/home/kabeerahmed/LabWork
```

LAB - 5

Task:

1. What did you learn after running Program 3? Write a few lines.

In Program 3, we are first calling `signal()` system call by which we are installing a new signal handler of `SIGALRM` using function `abc()` which just prints out *Time to ring the Sec. Gen. my boy*. Then by calling the `alarm()`, a timer of `3*6` is set after which the signal will occur. Lastly, the `pause()` call will wait or stop the program until the `SIGALRM` signal occurs. So in this program, we learned about how to install a new/custom signal handler, working of `alarm()` and `pause()` system calls.

2. Write a program which creates a child process and then wait for the child to terminate. Send a `SIGCHLD` signal from child to parent and show that parent resumes.

Code:

```

GNU nano 4.5                                     Task5.c
#include <stdio.h>
#include <unistd.h>
#include <wait.h>
#include <string.h>
#include <stdlib.h>
void handler() {
    printf("After Child termination! Signal child handler called\n");
}
int main() {
    int pid;
    pid = fork();
    signal(SIGCHLD, handler);
    if (pid < 0) {
        fprintf(stderr, "Due to some Error fork failed");
        return 1;
    } else if (pid == 0) {
        execlp("/bin/ls", "ls", NULL);
    } else {
        wait(NULL);
        printf("Parent Restart!\n");
    }
    return 0;
}

```

Output:

```

kabeerahmed@kali-linux:~/LabWork$ ./task5output
file01 task1      task2_1out  task2out   task3_1out Task5.c      taskout  yearfile
file02 Task2_1.c Task2.c    task3_1.c  task3.c    task5output user
After Child termination! Signal child handler called
Parent Restart!

```

LAB - 6

Task:

1. Write a few lines about what you learn after running the above Program and show the output.

Description:

The above program demonstrates the usage of threading by using pthread. In the main() function first of all the thread id is declared with type pthread_t. Then the thread attribute is declared using pthread_attr_t. With the line pthread_attr_init(&attr), the thread attr is initialized. Now with pthread_create() the thread is created with the required arguments. Then the pthread_join() function is called which is for threads is the equivalent of wait() for processes. The runner() function calculates the sum upto the number provided as the argument.

So in short the program calls the runner function with the newly created thread and the thread performs the calculation of the runner function and then terminates.

Output:

```
kabeerahmed@kali-linux:~/LabWork$ ./lab6output -2
-2 must be >=0
kabeerahmed@kali-linux:~/LabWork$ ./lab6output
usage :a.out <integer value>
kabeerahmed@kali-linux:~/LabWork$ ./lab6output 5
sum=10
```

Assignment

1. Using more than one thread in a program?

Code:

```
GNU nano 4.5 lab6_1.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
int g = 0;
void *myFunc(void *vargp)
{
    int *myid = (int *)vargp;
    static int s = 0;
    ++s; ++g;
    printf("Thread ID: %d, Static: %d, Global: %d\n", *myid, ++s, ++g);
}
int main()
{
    int i;
    pthread_t tid;
    for (i = 0; i < 3; i++)
        pthread_create(&tid, NULL, myFunc, (void *)&tid);

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

Output:

```
kabeerahmed@kali-linux:~/LabWork$ gcc -pthread lab6_1.c -o lab6_1out
kabeerahmed@kali-linux:~/LabWork$ ./lab6_1out
Thread ID: 1960937216, Static: 2, Global: 2
Thread ID: 1960937216, Static: 4, Global: 4
Thread ID: 1960937216, Static: 6, Global: 6
```

LAB - 7

Open Ended Lab

OBJECTIVE

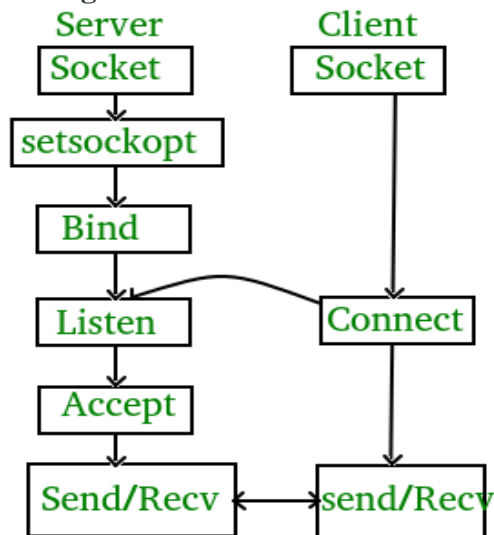
To understand the concept and implementation of socket programming in linux using C.

INTRODUCTION

What is socket programming?

Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection. Server forms the listener socket while client reaches out to the server.

State diagram for server and client model



Stages for server

- **Socket creation:**

```
int sockfd = socket(domain, type, protocol)
```

sockfd: socket descriptor, an integer (like a file-handle)

domain: integer, communication domain e.g., AF_INET (IPv4 protocol) , AF_INET6 (IPv6 protocol)

type: communication type

SOCK_STREAM: TCP(reliable, connection oriented)

SOCK_DGRAM: UDP(unreliable, connectionless)

protocol: Protocol value for Internet Protocol(IP), which is 0. This is the same number which appears on protocol field in the IP header of a packet.(man protocols for more details)

- **Setsockopt:**

```
int setsockopt(int sockfd, int level, int optname, const void *optval, socklen_t optlen);
```

This helps in manipulating options for the socket referred by the file descriptor sockfd. This is completely optional, but it helps in reuse of address and port. Prevents error such as: “address already in use”.

- **Bind:**

```
int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

After creation of the socket, bind function binds the socket to the address and port number specified in addr(custom data structure). In the example code, we bind the server to the localhost, hence we use INADDR_ANY to specify the IP address.

- **Listen:**

```
int listen(int sockfd, int backlog);
```

It puts the server socket in a passive mode, where it waits for the client to approach the server to make a connection. The backlog defines the maximum length to which the queue of pending connections for sockfd may grow. If a connection request arrives when the queue is full, the client may receive an error with an indication of ECONNREFUSED.

- **Accept:**

```
int new_socket= accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

It extracts the first connection request on the queue of pending connections for the listening socket, sockfd, creates a new connected socket, and returns a new file descriptor referring to that socket. At this point, connection is established between client and server, and they are ready to transfer data.

Stages for Client

- **Socket connection:** Exactly same as that of server’s socket creation
- **Connect:**

```
int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

The connect() system call connects the socket referred to by the file descriptor sockfd to the address specified by addr. Server’s address and port is specified in addr.

CONCLUSION

In this lab, we learned about socket programming. Its basic concepts, different terminologies used in the socket programming, the client-server interaction and finally the implementation of socket programming in c where the client says hello to the server and the server receives the message and responds accordingly.

CODE

Two files will be used, one for the server to accept the requests and one for the client to send the request.

client.c

```
GNU nano 4.5
#include <stdio.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>
#define PORT 8080

int main(int argc, char const *argv[])
{
    int sock = 0, valread;
    struct sockaddr_in serv_addr;
    char *hello = "Hey from client";
    char buffer[1024] = {0};
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        printf("\n Socket creation error \n");
        return -1;
    }
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);
    if(inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr)<=0)
    {
        printf("\nInvalid address/ Address not supported \n");
        return -1;
    }

    if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
    {
        printf("\nConnection Failed\n");
        return -1;
    }
    send(sock , hello , strlen(hello) , 0 );
    printf("Hey message sent\n");
    valread = read( sock , buffer, 1024);
    printf("%s\n",buffer );
    return 0;
}
```

server.c

```

GNU nano 4.5
#include <unistd.h>
#include <stdio.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <string.h>
#define PORT 8080
int main(int argc, char const *argv[])
{
    int server_fd, new_socket, valread;
    struct sockaddr_in address;
    int opt = 1;
    int addrlen = sizeof(address);
    char buffer[1024] = {0};
    char *hello = "Hello from server";
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0)
    {
        perror("socket failed");
        exit(EXIT_FAILURE);
    }
    if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT, &opt, sizeof(opt)))
    {
        perror("setsockopt");
        exit(EXIT_FAILURE);
    }
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons( PORT );

    if (bind(server_fd, (struct sockaddr *)&address, sizeof(address))<0)
    {
        perror("bind failed");
        exit(EXIT_FAILURE);
    }
    if (listen(server_fd, 3) < 0)
    {
        perror("listen");
        exit(EXIT_FAILURE);
    }
    if ((new_socket = accept(server_fd, (struct sockaddr *)&address, (socklen_t*)&addrlen))<0)
    {
        perror("accept");
        exit(EXIT_FAILURE);
    }
    valread = read( new_socket , buffer, 1024);
    printf("%s\n",buffer );
    send(new_socket , hello , strlen(hello) , 0 );
    printf("Hello message sent\n");
    return 0;
}

```

Output:

<pre> kabeerahmed@kali-linux:~/LabWork\$./server Hey from client Hello message sent </pre>	<pre> kabeerahmed@kali-linux:~/LabWork\$./client Hey message sent Hello from server </pre>
---	---

LAB – 8

Task:

1. Perform the implementation and show the output of FCFS and SJF(Non-Preemptive) algorithm.

FCFS Implementation:

```
GNU nano 4.5
#include <stdio.h>
int waitingtime(int proc[], int n,
int burst_time[], int wait_time[]) {
    wait_time[0] = 0;
    for (int i = 1; i < n ; i++ )
        wait_time[i] = burst_time[i-1] + wait_time[i-1] ;
    return 0;
}
int turnaroundtime( int proc[], int n,
int burst_time[], int wait_time[], int tat[]) {
    int i;
    for ( i = 0; i < n ; i++)
        tat[i] = burst_time[i] + wait_time[i];
    return 0;
}
int avgtime( int proc[], int n, int burst_time[]) {
    int wait_time[n], tat[n], total_wt = 0, total_tat = 0;
    int i;
    waitingtime(proc, n, burst_time, wait_time);
    turnaroundtime(proc, n, burst_time, wait_time, tat);
    printf("Processes Burst Waiting Turn around \n");
    for ( i=0; i<n; i++) {
        total_wt = total_wt + wait_time[i];
        total_tat = total_tat + tat[i];
        printf(" %d\t %d\t\t %d \t%d\n", i+1, burst_time[i], wait_time[i], tat[i]);
    }
    printf("Average waiting time = %f\n", (float)total_wt / (float)n);
    printf("Average turn around time = %f\n", (float)total_tat / (float)n);
    return 0;
}
int main() {
    int proc[] = { 1, 2, 3};
    int n = sizeof proc / sizeof proc[0];
    int burst_time[] = {5, 8, 12};
    avgtime(proc, n, burst_time);
    return 0;
}
```

Output:

```
kabeerahmed@kali-linux:~/LabWork/lab8$ gcc task1.c -o task1out
kabeerahmed@kali-linux:~/LabWork/lab8$ ./task1out
```

Processes	Burst	Waiting	Turn around
1	5	0	5
2	8	5	13
3	12	13	25

Average waiting time = 6.000000
Average turn around time = 14.333333

SJF Non-Preemptive Implementation:

```
#include<stdio.h>
int main()
{
    int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;
    float avg_wt,avg_tat;
    printf("Enter number of process:");
    scanf("%d",&n);
    printf("\nEnter Burst Time:\n");
    for(i=0;i<n;i++)
    {
        printf("p%d:",i+1);
        scanf("%d",&bt[i]);
        p[i]=i+1;
    }
    for(i=0;i<n;i++)
    {
        pos=i;
        for(j=i+1;j<n;j++)
        {
            if(bt[j]<bt[pos])
                pos=j;
        }
        temp=bt[i];
        bt[i]=bt[pos];
        bt[pos]=temp;
        temp=p[i];
        p[i]=p[pos];
        p[pos]=temp;
    }
    wt[0]=0;
    for(i=1;i<n;i++)
    {
        wt[i]=0;
        for(j=0;j<i;j++)
            wt[i]+=bt[j];
        total+=wt[i];
    }
    avg_wt=(float)total/n;
    total=0;
    printf("\nProcess\t\t Burst Time\t\t \tWaiting Time\tTurnaround Time");
    for(i=0;i<n;i++)
    {
        tat[i]=bt[i]+wt[i];
        total+=tat[i];
        printf("\np%d\t\t %d\t\t\t %d\t\t\t\t %d",p[i],bt[i],wt[i],tat[i]);
    }
    avg_tat=(float)total/n;
    printf("\n\nAverage Waiting Time=%f",avg_wt);
    printf("\nAverage Turnaround Time=%f\n",avg_tat);
}
```

Output:

```
kabeerahmed@kali-linux:~/LabWork/lab8$ gcc task2.c -o task2out
kabeerahmed@kali-linux:~/LabWork/lab8$ ./task2out
Enter number of process:5

Enter Burst Time:
p1:6
p2:2
p3:4
p4:7
p5:3

Process      Burst Time      Waiting Time      Turnaround Time
p2            2                0                 2
p5            3                2                 5
p3            4                5                 9
p1            6                9                15
p4            7               15               22

Average Waiting Time=6.200000
Average Turnaround Time=10.600000
```

Assignment

1. Write the SJF (Preemptive) algorithm and also implement it and show the output.

SJF Preemptive Algorithm:

- 1- Traverse until all process gets completely executed.
 - a) Find process with minimum remaining time at every single time lap.
 - b) Reduce its time by 1.
 - c) Check if its remaining time becomes 0
 - d) Increment the counter of process completion.
 - e) Completion time of current process =
current_time +1;
 - e) Calculate waiting time for each completed process.
 $wt[i] = \text{Completion time} - \text{arrival_time} - \text{burst_time}$
 - f) Increment time lap by one.
- 2- Find turnaround time (waiting_time+burst_time).

SJF Preemptive Implementation:

```

#include <stdio.h>
int main()
{
    int arrival_time[10], burst_time[10], temp[10];
    int i, smallest, count = 0, time, limit;
    double wait_time = 0, turnaround_time = 0, end;
    float average_waiting_time, average_turnaround_time;
    printf("\nEnter Number of Processes:\t");
    scanf("%d", &limit);
    printf("\nEnter Details of %d Processesn", limit);
    for(i = 0; i < limit; i++)
    {
        printf("\nEnter Arrival Time:\t");
        scanf("%d", &arrival_time[i]);
        printf("Enter Burst Time:\t");
        scanf("%d", &burst_time[i]);
        temp[i] = burst_time[i];
    }
    burst_time[9] = 9999;
    for(time = 0; count != limit; time++)
    {
        smallest = 9;
        for(i = 0; i < limit; i++)
        {
            if(arrival_time[i] <= time && burst_time[i] < burst_time[smallest] && burst_time[i] > 0)
            {
                smallest = i;
            }
        }
        burst_time[smallest]--;
        if(burst_time[smallest] == 0)
        {
            count++;
            end = time + 1;
            wait_time = wait_time + end - arrival_time[smallest] - temp[smallest];
            turnaround_time = turnaround_time + end - arrival_time[smallest];
        }
    }
    average_waiting_time = wait_time / limit;
    average_turnaround_time = turnaround_time / limit;
    printf("\n\nAverage Waiting Time:\t%lf\n", average_waiting_time);
    printf("Average Turnaround Time:\t%lf\n", average_turnaround_time);
    return 0;
}

```

Output:

```

kabeerahmed@kali-linux:~/LabWork/lab8$ ./task3out

Enter Number of Processes:      2

Enter Details of 2 Processesn
Enter Arrival Time:      0
Enter Burst Time:      5

Enter Arrival Time:      4
Enter Burst Time:      10

Average Waiting Time:      0.500000
Average Turnaround Time:      8.000000

```

LAB – 9

Task:

1. Perform the implementation and show the output Priority (Non-Preemptive) and Round Robin algorithms.

Priority (Non-Preemptive) Implementation:

```
#include<stdio.h>
struct process
{
    int id,WT,AT,BT,TAT,PR;
};
struct process a[10];
void swap(int *b,int *c)
{
    int tem;
    tem=*c;
    *c=*b;
    *b=tem;
}
int main()
{
    int n,check_ar=0;
    int Cmp_time=0;
    float Total_WT=0,Total_TAT=0,Avg_WT,Avg_TAT;
    printf("Enter the number of process \n");
    scanf("%d",&n);
    printf("Enter the Arrival time , Burst time and priority of the process\n");
    printf("AT BT PR\n");
    for(int i=0;i<n;i++)
    {
        scanf("%d%d%d",&a[i].AT,&a[i].BT,&a[i].PR);
        a[i].id=i+1;
        if(i==0)
            check_ar=a[i].AT;
        if(check_ar!=a[i].AT )
            check_ar=1;
    }
    if(check_ar!=0)
    {
        for(int i=0;i<n;i++)
        {
            for(int j=0;j<n-i-1;j++)
            {
                if(a[j].AT>a[j+1].AT)
                {
                    swap(&a[j].id,&a[j+1].id);
                    swap(&a[j].AT,&a[j+1].AT);
                    swap(&a[j].BT,&a[j+1].BT);
                    swap(&a[j].PR,&a[j+1].PR);
                }
            }
        }
    }
    if(check_ar!=0)
    {
        a[0].WT=a[0].AT;
        a[0].TAT=a[0].BT-a[0].AT;
        Cmp_time=a[0].TAT;
        Total_WT=Total_WT+a[0].WT;
        Total_TAT=Total_TAT+a[0].TAT;
        for(int i=1;i<n;i++)
        {
```



```

        int min=a[i].PR;
        for(int j=i+1;j<n;j++)
        {
            if(min>a[j].PR && a[j].AT<=Cmp_time)
            {
                min=a[j].PR;
                swap(&a[i].id,&a[j].id);
                swap(&a[i].AT,&a[j].AT);
                swap(&a[i].BT,&a[j].BT);
                swap(&a[i].PR,&a[j].PR);
            }
        }
        a[i].WT=Cmp_time-a[i].AT;
        Total_WT=Total_WT+a[i].WT;
        Cmp_time=Cmp_time+a[i].BT;
        a[i].TAT=Cmp_time-a[i].AT;
        Total_TAT=Total_TAT+a[i].TAT;
    }
}
else
{
    for(int i=0;i<n;i++)
    {
        int min=a[i].PR;
        for(int j=i+1;j<n;j++)
        {
            if(min>a[j].PR && a[j].AT<=Cmp_time)
            {
                min=a[j].PR;
                swap(&a[i].id,&a[j].id);
                swap(&a[i].AT,&a[j].AT);
                swap(&a[i].BT,&a[j].BT);
                swap(&a[i].PR,&a[j].PR);
            }
        }
        a[i].WT=Cmp_time-a[i].AT;
        Cmp_time=Cmp_time+a[i].BT;
        a[i].TAT=Cmp_time-a[i].AT;
        Total_WT=Total_WT+a[i].WT;
        Total_TAT=Total_TAT+a[i].TAT;
    }
}
Avg_WT=Total_WT/n;
Avg_TAT=Total_TAT/n;
printf("The process are\n");
printf("ID WT TAT\n");
for(int i=0;i<n;i++)
{
    printf("%d\t%d\t%d\n",a[i].id,a[i].WT,a[i].TAT);
}
printf("Avg waiting time is: %f\n",Avg_WT);
printf("Avg turn around time is: %f",Avg_TAT);
return 0;
}

```

Output:

```

kabeerahmed@kali-linux:~/LabWork/lab9$ ./task1out
Enter the number of process
4
Enter the Arrival time , Burst time and priority of the process
AT BT PR
0 5 2
3 6 3
4 2 1
5 8 4
The process are
ID WT TAT
1      0      5
3      1      3
2      4     10
4      8     16
Avg waiting time is: 3.250000
Avg turn around time is: 8.500000kabeerahmed@kali-linux:~/LabWork/lab9$

```

Round Robin Implementation:

```

#include<stdio.h>
int main()
{
    int count,j,n,time,remain,flag=0,time_quantum;
    int wait_time=0,turnaround_time=0,at[10],bt[10],rt[10];
    printf("Enter Total Process:\t ");
    scanf("%d",&n);
    remain=n;
    for(count=0;count<n;count++)
    {
        printf("Enter Arrival Time and Burst Time for Process Process Number %d :",count+1);
        scanf("%d",&at[count]);
        scanf("%d",&bt[count]);
        rt[count]=bt[count];
    }
    printf("Enter Time Quantum:\t");
    scanf("%d",&time_quantum);
    printf("\n\nProcess\t|Turnaround Time|Waiting Time\n\n");
    for(time=0,count=0;remain!=0;)
    {
        if(rt[count]<=time_quantum && rt[count]>0)
        {
            time+=rt[count];
            rt[count]=0;
            flag=1;
        }
        else if(rt[count]>0)
        {
            rt[count]-=time_quantum;
            time+=time_quantum;
        }
        if(rt[count]==0 && flag==1)
        {
            remain--;
            printf("P[%d]\t|\t%d\t|\t%d\n",count+1,time-at[count],time-at[count]-bt[count]);
            wait_time+=time-at[count]-bt[count];
            turnaround_time+=time-at[count];
            flag=0;
        }
        if(count==n-1)
            count=0;
        else if(at[count+1]<=time)
            count++;
        else
            count=0;
    }
    printf("\nAverage Waiting Time= %f\n",wait_time*1.0/n);
    printf("Avg Turnaround Time = %f",turnaround_time*1.0/n);
    return 0;
}

```

Output:

```
kabeerahmed@kali-linux:~/LabWork/lab9$ ./task2out
Enter Total Process:      2
Enter Arrival Time and Burst Time for Process Process Number 1 :0
50
Enter Arrival Time and Burst Time for Process Process Number 2 :10
28
Enter Time Quantum:      20

Process |Turnaround Time|Waiting Time
P[2]    |      58      |      30
P[1]    |      78      |      28

Average Waiting Time= 29.000000
Avg Turnaround Time = 68.000000kabeerahmed@kali-linux:~/LabWork/lab9$
```

Assignment

1. Write the algorithm of Priority (Preemptive) algorithm and also implement it and show the output.

Priority (Preemptive) Algorithm:

1. First we copy the burst time of the process in a new array temp[] because in the further calculation we will be going to decrease the Burst time of the process but we will have to need the real burst time of the process in the calculation of the waiting time .(If you confused then don't worry you will be able understand after going through code)
2. we initialize the priority of a process with the maximum (you can take any maximum value). and we will use 9th process because we assumed that there will not be more than 10 process but you can use any number.
3. In this code we are going to use a loop which executed until all the processes are completed. for checking how many processes are completed we use **count** .Initially it's value is 0 (i.e no processes are completed yet).
4. In each cycle we will find the process which have highest priority(lowest priority number like 1 have high priority than 2) and arrived at time t and burst time of the process is not equal to zero.
5. After doing this we will decrease the burst time of the process by 1 in each cycle of the time.
6. And if the process will be complete (Burst time =0) then we will increase the value of the count by 1 (i.e one process is completed)
7. For calculating the waiting time we will use a formula (WT= time- arrival-Burst time) let's understand by an example :- Lets say we have any work in the bank for 5 hours . and we go at the 2 pm and we will come at 9 pm from the bank then waiting time in the bank is:-

$$\begin{aligned} &= (\text{time spend in the bank}) - (\text{Total work time}) \\ &= (9-2) - 5 = 2 \end{aligned}$$

So we have to wait for the 2 hours .

8. For calculating turnaround time we simply use $TAT = \text{completion time} - \text{arrival time}$.

Priority (Preemptive) Implementation:

```
#include<stdio.h>
struct process
{
    int WT,AT,BT,TAT,PT;
};

struct process a[10];

int main()
{
    int n,temp[10],t,count=0,short_p;
    float total_WT=0,total_TAT=0,Avg_WT,Avg_TAT;
    printf("Enter the number of the process\n");
    scanf("%d",&n);
    printf("Enter the arrival time , burst time and priority of the process\n");
    printf("AT BT PT\n");
    for(int i=0;i<n;i++)
    {
        scanf("%d%d%d",&a[i].AT,&a[i].BT,&a[i].PT);
        temp[i]=a[i].BT;
    }
    a[9].PT=10000;
    for(t=0;count!=n;t++)
    {
        short_p=9;
        for(int i=0;i<n;i++)
        {
            if(a[short_p].PT>a[i].PT && a[i].AT<=t && a[i].BT>0)
            {
                short_p=i;
            }
        }
        a[short_p].BT=a[short_p].BT-1;
        if(a[short_p].BT==0)
        {
            count++;
            a[short_p].WT=t+1-a[short_p].AT-temp[short_p];
            a[short_p].TAT=t+1-a[short_p].AT;
            total_WT=total_WT+a[short_p].WT;
            total_TAT=total_TAT+a[short_p].TAT;
        }
    }
    Avg_WT=total_WT/n;
    Avg_TAT=total_TAT/n;
    printf("ID WT TAT\n");
    for(int i=0;i<n;i++)
    {
        printf("%d %d\t%d\n",i+1,a[i].WT,a[i].TAT);
    }
    printf("Avg waiting time of the process is %f\n",Avg_WT);
    printf("Avg turn around time of the process is %f\n",Avg_TAT);
    return 0;
}
```

Output:

```
kabeerahmed@kali-linux:~/LabWork/lab9$ ./task3out
Enter the number of the process
4
Enter the arrival time , burst time and priority of the process
AT BT PT
0 5 2
4 6 3
5 3 1
6 4 4
ID WT TAT
1 0 5
2 4 10
3 0 3
4 8 12
Avg waiting time of the process is 3.000000
Avg turn around time of the process is 7.500000
```

LAB – 10

Task:

1.What did you learn after running the above Program 1 and Program 2?

Program 1 works as the reading end of a named pipe while Program 2 is used as the write end of the pipe. In Program 1, a file *testfile* is being used as a pipe. The program opens the *testfile* and runs an infinite loop in which it is reading from the **read()** system call. Now Program 2 opens the same *testfile* and writes the message using the **write()** call. Both Programs show some kind of error on invalid syntax. From these programs, we learned the concept of pipes, reading from the pipe and writing to the pipe.

```
kabeerahmed@kali-linux:~/LabWork/lab10$ ./task1out
message received:hey
message received:Its
message received:me
message received:Kabeer
```

```
kabeerahmed@kali-linux:~/LabWork/lab10$ ./task2out hey Its me Kabeer
```

2.What did you learn after running Program 3?

In Program 3, we are making a pipe named *myfifo*. We created this pipe using **mknod()**. **mknod()** takes pipe name as the first parameter and permission of the file as the second parameter. Now *myfifo* will be used as a pipe for inter-process communication. We opened the pipe as Read/Write then used the **read()** system call to read from the pipe. Now whenever we write to the pipe *myfifo* the Program 3 will receive the message as it is reading from the pipe. So by Program 3, we learned about **mknod()** to create a pipe file and then opened the file and everytime someone writes to the file we get the output in the Program 3.

```
kabeerahmed@kali-linux:~/LabWork/lab10$ ./task3out
message received:hey
```

Assignment:

Code:

```
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
    int fd;
    char * myfifo = "/tmp/myfifo";
    mkfifo(myfifo, 0666);
    char arr1[80], arr2[80];
    while (1)
    {
        fd = open(myfifo, O_WRONLY);
        fgets(arr2, 80, stdin);
        write(fd, arr2, strlen(arr2)+1);
        close(fd);
        fd = open(myfifo, O_RDONLY);
        read(fd, arr1, sizeof(arr1));
        printf("man1: %s\n", arr1);
        close(fd);
    }
    return 0;
}
```

```
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
    int fd1;
    char * myfifo = "/tmp/myfifo";
    mkfifo(myfifo, 0666);
    char str1[80], str2[80];
    while (1)
    {
        fd1 = open(myfifo, O_RDONLY);
        read(fd1, str1, 80);
        printf("man2: %s\n", str1);
        close(fd1);
        fd1 = open(myfifo, O_WRONLY);
        fgets(str2, 80, stdin);
        write(fd1, str2, strlen(str2)+1);
        close(fd1);
    }
    return 0;
}
```

Output:

```
kabeerahmed@kali-linux:~/LabWork/lab10/Ass$ ./task1out
Hey
man1: Hey

how are you?
man1: I am fine and hru?

I am also fine
█
```

```
kabeerahmed@kali-linux:~/LabWork/lab10/Ass$ ./task2out
Hey
man2: Hey

man2: how are you?

I am fine and hru?
man2: I am also fine
```


LAB – 11

Task:

1.What did you learn after running Program 1?

In Program 1, we learned about the pipe() call by which we can create a one-way communication b/w parent and the child. In this program the parent process asks the user for input for the limit of the fibonacci series. The parent then writes through the pipe using pfd[1] (write file descriptor) and then closes the write end. Now the child process reads the fibonacci series limit from the read end of the pipe using pfd[0] (read file descriptor) and then calculates the fibonacci series upto the given limit and returns the output. So in this way the parent process asked for input, provided it to the child using a pipe and then the child calculated on the basis of read input from the pipe.

Assignment

Code:

```
#include <stdio.h>
#include <unistd.h>
#define MSGSIZE 16
char* msg1 = "hello, world #1";
char* msg2 = "hello, world #2";
char* msg3 = "hello, world #3";
int main()
{
    char inbuf[MSGSIZE];
    int p[2], pid, nbytes;
    if (pipe(p) < 0)
        exit(1);
    if ((pid = fork()) > 0) {
        write(p[1], msg1, MSGSIZE);
        write(p[1], msg2, MSGSIZE);
        write(p[1], msg3, MSGSIZE);
        close(p[1]);
        wait(NULL);
    }
    else {
        close(p[1]);
        while ((nbytes = read(p[0], inbuf, MSGSIZE)) > 0)
            printf("%s\n", inbuf);
        if (nbytes != 0)
            exit(2);
        printf("Finished reading\n");
    }
    return 0;
}
```

Output

```
kabeerahmed@kali-linux:~/LabWork/lab11$ ./task1out
hello, world #1
hello, world #2
hello, world #3
Finished reading
```

LAB - 12

Task:

1. Write a few lines about what you learn after running Programs 3 and Program 4.

In Program 3 & 4, we learned about giving a lock to a file using **lockf()** so that no other file can modify the same file while the first file is writing to it. Program 3 runs first and acquires the lock and after its execution completes, then Program 4 gets access to the file and can modify the file *locktest*.

Assignment

Code:

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

int main( )
{
    int fd, i;
    char s[] = "Hey! this will be written to the file by task1 and will be read by Task2!";
    fd = open("testfile",O_APPEND | O_CREAT | O_RDWR,0777);
    write(fd, s, sizeof(s));
}
```

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

int main( )
{
    int fd, i;
    char s[100];
    fd = open("testfile",O_APPEND | O_CREAT | O_RDWR,0777);
    read(fd, &s, sizeof(s));
    printf(s);
}
```

Output:

```
kabeerahmed@kali-linux:~/LabWork/lab12$ ./task1out
```

```
kabeerahmed@kali-linux:~/LabWork/lab12$ ./task2out
Hey! this will be written to the file by task1 and will be read by Task
```

LAB - 13

Assignment:

Code:

```
#!/bin/bash

echo "Multiplication is:"
echo `expr $1 \* $2`

echo "Sum is:"
echo `expr $1 + $2`
```

Output:

```
kabeerahmed@kali-linux:~/LabWork/lab13$ ./task1.sh 5 10
Multiplication is:
50
Sum is:
15
```

LAB – 14

Task:

1. Write a script which displays Good Morning if hour is less than 12.00, Good Afternoon if hour is less than 5.00p.m, and Good Evening if hour is greater than 5.00p.m.

Code:

```
#!/bin/bash

h=`date +%H`

if [ $h -lt 12 ]; then
    echo Good morning
elif [ $h -lt 18 ]; then
    echo Good afternoon
else
    echo Good evening
fi
```

Output:

```
kabeerahmed@kali-linux:~/LabWork/lab14$ ./task1.sh
Good morning
```

2. Write a script that determines whether a number is odd or even.

Code:

```
GNU nano 4.5
echo -n "Enter number : "
read n
rem=$(( $n % 2 ))

if [ $rem -eq 0 ]
then
    echo "$n is even number"
else
    echo "$n is odd number"
fi
```

Output:

```
kabeerahmed@kali-linux:~/LabWork/lab14$ ./task2.sh
Enter number : 5
5 is odd number
kabeerahmed@kali-linux:~/LabWork/lab14$ ./task2.sh
Enter number : 8
8 is even number
```

3. Write a script which takes two numbers as input and asks user choice for multiplication, division, addition, and subtraction. Like (*calculator*), and then calculate the output. Apply suitable checks; if user enters wrong input the program again asks for correct input.

Code:

```
GNU nano 4.5
# !/bin/bash
echo "Enter Two numbers : "
read a
read b

echo "Enter Choice :"
echo "1. Addition"
echo "2. Subtraction"
echo "3. Multiplication"
echo "4. Division"
read ch

case $ch in
1)res=`echo $a + $b | bc`
;;
2)res=`echo $a - $b | bc`
;;
3)res=`echo $a \* $b | bc`
;;
4)res=`echo "scale=2; $a / $b" | bc`
;;
esac
echo "Result : $res"
```

Output:

```
kabeerahmed@kali-linux:~/LabWork/lab14$ ./task3.sh
Enter Two numbers :
5
9
Enter Choice :
1. Addition
2. Subtraction
3. Multiplication
4. Division
3
Result : 45
kabeerahmed@kali-linux:~/LabWork/lab14$ ./task3.sh
Enter Two numbers :
9
5
Enter Choice :
1. Addition
2. Subtraction
3. Multiplication
4. Division
2
Result : 4
```

4. Write a script to print no's as 5, 4, 3, 2, 1 using while loop.

Code:

```
GNU nano 4.5
#!/bin/bash

echo "Input number"
read k
echo "The numbers from $k to 1 in reverse are:"
while test $k ≠ 0
do
    echo "$k"
    k=$(( k - 1 ))
done
```

Output:

```
kabeerahmed@kali-linux:~/LabWork/lab14$ ./task4.sh
Input number
6
The numbers from 6 to 1 in reverse are:
6
5
4
3
2
1
```

5. Write script to print given number in reverse order, for e.g. If no is 123 it must print as 321.

Code:

```
#!/bin/bash

echo "Enter the number to reverse: "
read num
echo "The number in reverse order is: "
while [ $num != 0 ]
do
d=$(( $num % 10 ))
num=$(( $num / 10 ))
rev=$(( echo ${rev}${d} ))
done
echo $rev
```

Output:

```
kabeerahmed@kali-linux:~/LabWork/lab14$ ./task5.sh
Enter the number to reverse:
456
The number in reverse order is:
654
```


6. Write script to print given numbers sum of all digits, for e.g. If no is 123 it's sum of all digits will be $1+2+3 = 6$.

Code:

```
echo "Enter a number: "
read num
sum=0
while [ $num -gt 0 ]
do
    mod=$((num % 10))
    sum=$((sum + mod))
    num=$((num / 10))
done
echo "Sum is: $sum"
```

Output:

```
kabeerahmed@kali-linux:~/LabWork/lab14$ ./task6.sh
Enter a number:
456
Sum is: 15
```

7. Write Script to find out the biggest number from given three no's. No's are supplied as command line arguments. Print error if sufficient arguments are not supplied.

Code:

```
#!/bin/bash

if [ $# -ne 3 ]
then
    echo "$0: number1 number2 number3 are not given" >&2
    exit 1
fi
n1=$1
n2=$2
n3=$3
if [ $n1 -gt $n2 ] && [ $n1 -gt $n3 ]
then
    echo "$n1 is Biggest number"
elif [ $n2 -gt $n1 ] && [ $n2 -gt $n3 ]
then
    echo "$n2 is Biggest number"
elif [ $n3 -gt $n1 ] && [ $n3 -gt $n2 ]
then
    echo "$n3 is Biggest number"
elif [ $1 -eq $2 ] && [ $1 -eq $3 ] && [ $2 -eq $3 ]
then
    echo "All the three numbers are equal"
else
    echo "I can not figure out which number is bigger"
fi
```

Output:

```
kabeerahmed@kali-linux:~/LabWork/lab14$ ./task7.sh
./task7.sh: number1 number2 number3 are not given
kabeerahmed@kali-linux:~/LabWork/lab14$ ./task7.sh 52 500 421
500 is Biggest number
kabeerahmed@kali-linux:~/LabWork/lab14$ ./task7.sh 5 5 5
All the three numbers are equal
kabeerahmed@kali-linux:~/LabWork/lab14$ ./task7.sh 5 5 2
I can not figure out which number is bigger
```

Assignment:

1. List out all the files in the same folder.

Code:

```
#!/bin/bash

for f in *
do
if [ -x $f ]
then
echo $f;
fi
done

echo $3
```

Output:

```
kabeerahmed@kali-linux:~/LabWork/lab14$ ./task8.sh
task1.sh
task2.sh
task3.sh
task4.sh
task5.sh
task6.sh
task7.sh
task8.sh
```