OS LAB ASSIGNMENT - 5

Done By	Roll Number
N Sree Dhyuti	CED19I027

---

(1) Write a program using pipe() and fork() in which the parent process takes one string as input. The same is sent to the child using pipe1 and the child will reverse it. After the reversing is complete, the child process will send it (reversed string) to the parent process using pipe2. Subsequently, the parent process will read the string and display it.

CODE :

```c
// N Sree Dhyuti
// CED19I027
// Lab 5 : Q1

// Inclusion of required libraries
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <signal.h>
#include <sys/wait.h>
#include <string.h>

// Main
int main()
{
    // Create a pid for calling fork() function
    pid_t pid;

    //Create pipe1 and pipe2
    int p1[2], p2[2];
```

```c
// Strings to send and receive messages between parent and child processes
char p1_send[100], p1_recv[100], p2_send[100], p2_recv[100];

if (pipe(p1) == -1 || pipe(p2))     //creating pipe1 and pipe2
{
    printf("\n Pipe1 not created");
    return 1;
}

// Fork function call
pid = fork();

// For Child Process
if (pid == 0)
{
    //close child read and parent write
    close(p2[0]);
    close(p1[1]);

    // Receive string
    read(p1[0], p1_recv, sizeof(p1_recv) + 1);

    // Reverse the string
    int len = strlen(p1_recv);

    for (int i = 0; i < len; i++)
    {
        p2_send[i] = p1_recv[len - i - 1];
    }
    p2_send[len] - '\0';   //delimiter

    // Send the reversed string
    write(p2[1], p2_send, sizeof(p2_send) + 1);

}
// Error
else if (pid < 0)
{
    printf("Failed to execute fork() for a while. \n");
```

```
    }
    // For Parent Process
    else
    {
        //Close parent read (p1[0]) and child write (p2[1])
        close(p2[1]);
        close(p1[0]);

        // User Inputs
        printf("Enter String to be sent : ");
        scanf("%s", p1_send);

        // Send string
        write(p1[1], p1_send, sizeof(p1_send) + 1);

        // Receive reversed string from child
        read(p2[0], p2_recv, sizeof(p2_recv) + 1);  //parent reading from pipe2
        printf("Received reversed string :  %s\n", p2_recv);
    }

    return 0;
}
```

OUTPUT :

OS Lab Assignment - 5

N. Sree Dhyuti
CED19I027

18/9/21

(1) Write a program using fork() and pipe() to reverse a string.

Explanation of code:

(i) Create two pipe arrays P1 & P2 of size 2.
(ii) Create a child process using fork
(ii) In the parent process (pid > 0), take the string input & send it to the child process using write().
(iii) In the child process, receive the string using read() and reverse it and send it back to the parent using write().
(iv) Parent process shall have a receiving read() to receive the reversed string.
(v) Display Outputs.

About fork():

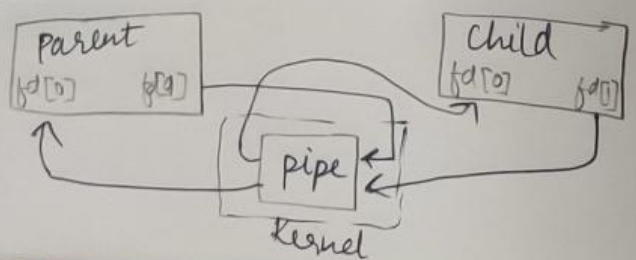
'fork' system call is used to create a new process. It clones the prev program from which it is called excluding the functions call of this one line alone.

pid > 0 ⇒ parent process
pid = 0 ⇒ child process
pid < 0 ⇒ Error

About pipe():

'pipe' is used as a connection between two processes. Although pipe is a one-way communication tool, we can make use of two pipes to allow 2-way communication between processes

pipe between a parent & a child



(2) Write a program using pipe() and fork() in which parent process takes string1 as input. The same is provided to the child process using pipe1. Now, child process will take string2 as input and read string1 from pipe1. Then, child will concatenate string1 with string2. After the concatenation is complete, the child process will send it to the parent process using pipe2. Subsequently, parent will read the concatenated string and displays it.

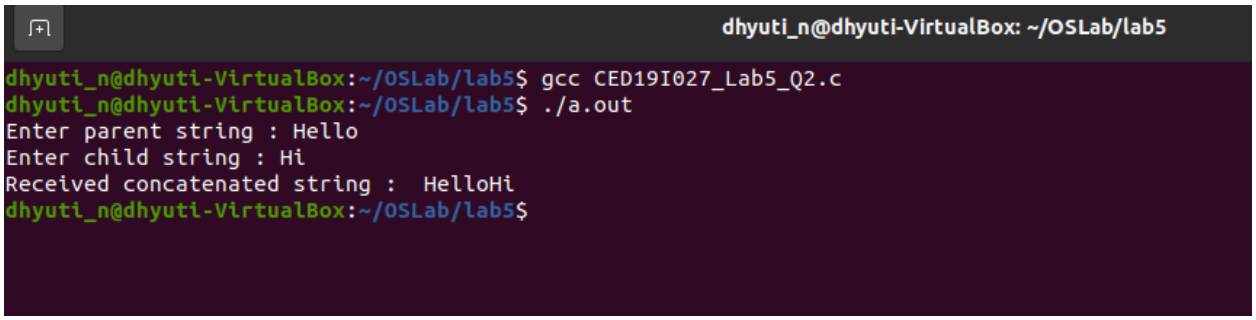
CODE :

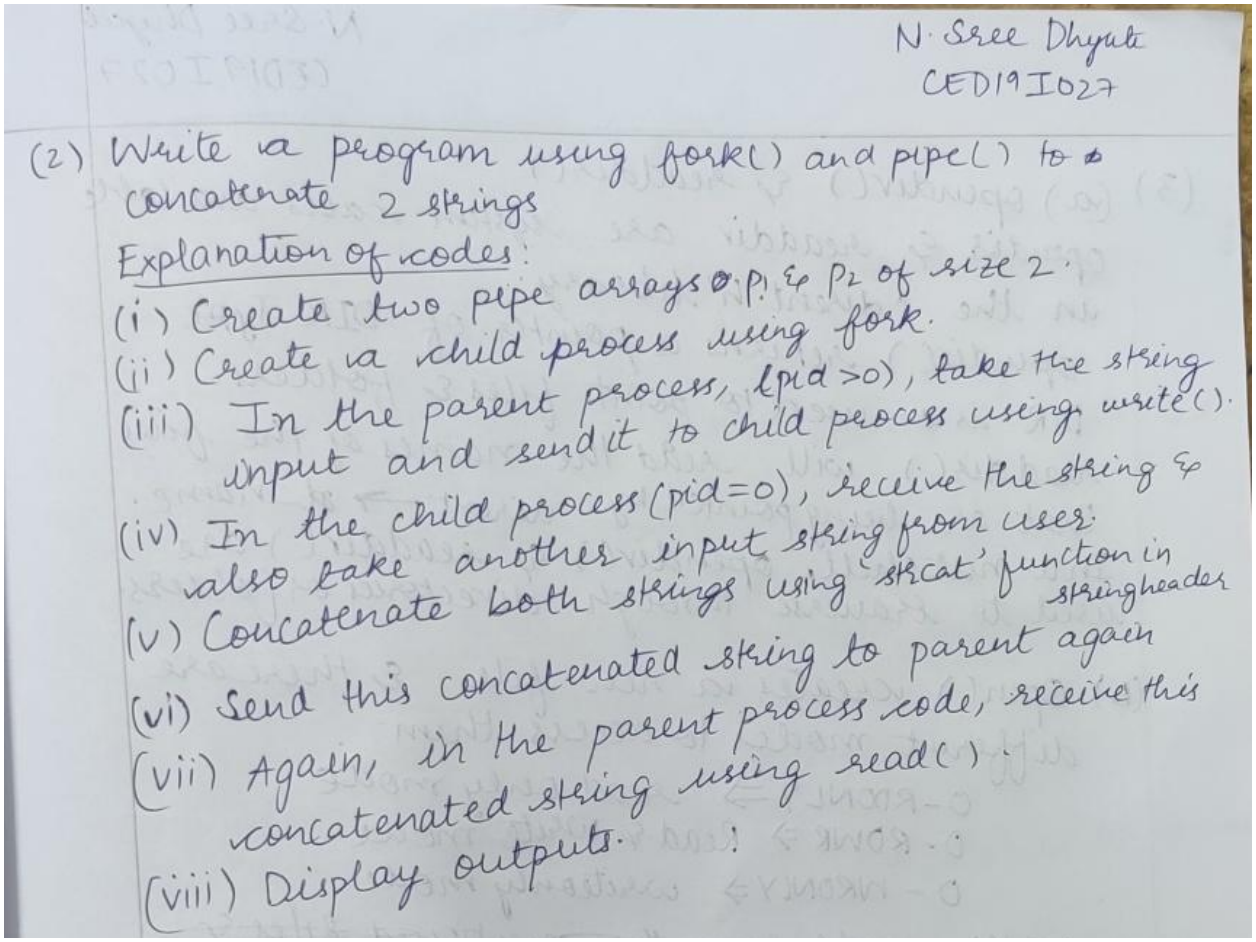
```c
// N Sree Dhyuti
// CED19I027
// Lab 5 : Q2

// Inclusion of required libraries
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <signal.h>
#include <sys/wait.h>
#include <string.h>

// Main
int main()
{
    // Create a pid for calling fork() function
    pid_t pid;

    //Create pipe1 and pipe2
    int p1[2], p2[2];

    // Strings to send and receive messages between parent and child processes
    char p1_send[100], p1_recv[100], p2_send[100], p2_recv[100];

    if(pipe(p1)== -1 || pipe(p2))     //creating pipe1
    {
        printf("\n Pipe1 not created");
        return 1;
```

```
    }

    // Fork function call
    pid = fork();

    // For Child Process
    if (pid == 0)
    {
        //close child read and parent write
        close(p2[0]);
        close(p1[1]);

        // Receive string
        read (p1[0], p1_recv, sizeof(p1_recv) + 1);

        char childstr[100];
        printf("Enter child string : ");
        scanf("%s",childstr);

        // Concatenate
        strcpy(p2_send, p1_recv);
        strcat(p2_send, childstr);

        // Send the reversed string
        write(p2[1], p2_send, sizeof(p2_send) + 1);

    }
    // Error
    else if (pid < 0)
    {
        printf("Failed to execute fork() for a while. \n");
    }
    // For Parent Process
    else
    {
        //Close parent read (p1[0]) and child write (p2[1])
        close(p2[1]);
        close(p1[0]);
```

```
        // User Inputs
        printf("Enter parent string : ");
        scanf("%s",p1_send);

        // Send string
        write(p1[1], p1_send, sizeof(p1_send)+1);

        // Receive reversed string from child
        read(p2[0], p2_recv, sizeof(p2_recv) + 1);  //parent reading from pipe2
        printf("Received concatenated string :  %s\n",p2_recv);
    }

    return 0;
}
```

OUTPUT :





---

(3)Explore the following system calls.

- **a. opendir() and readdir()**
- **b. open() with its ACCESS MODES and close()**
- **c. read() and write()**

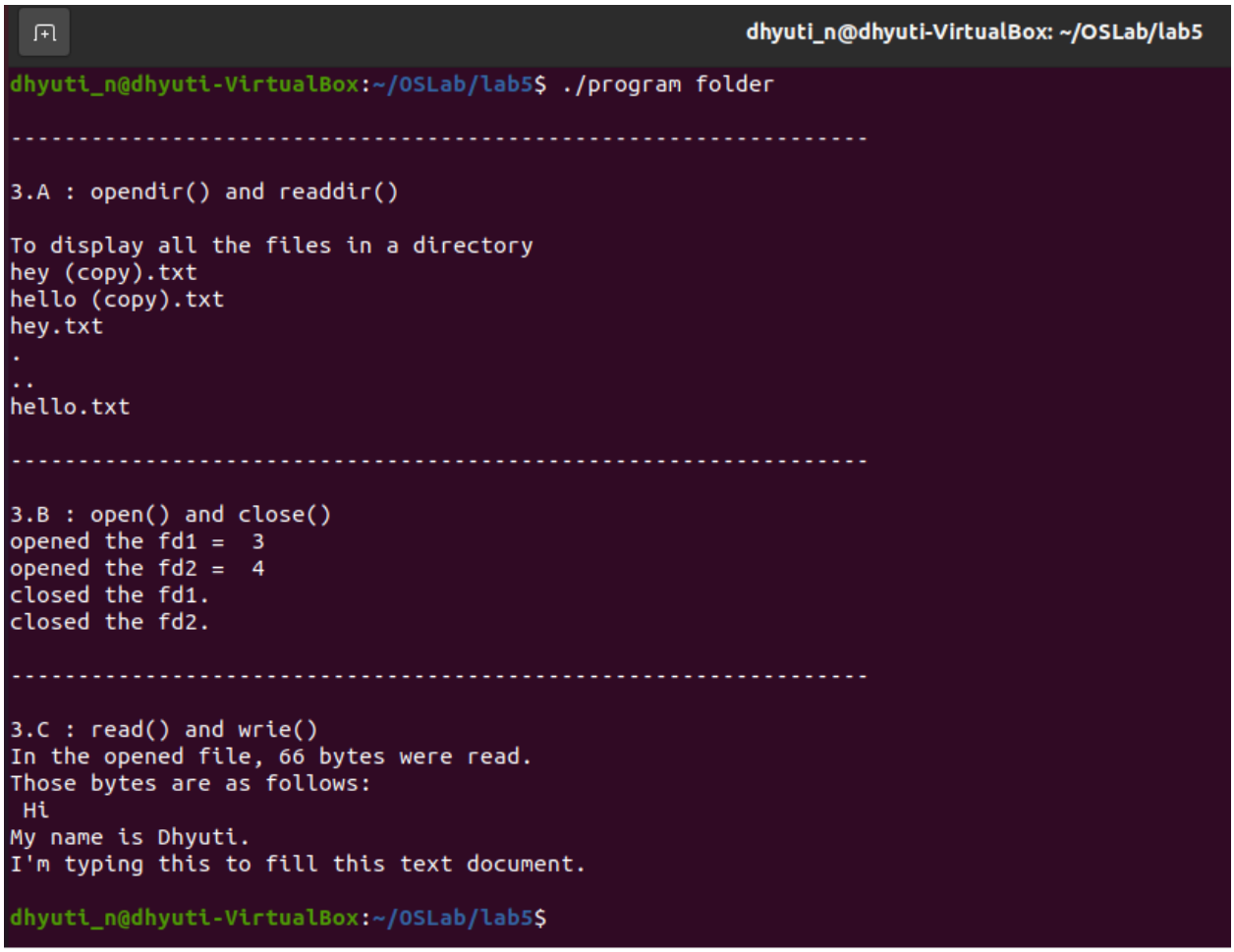
CODE :

```
// N Sree Dhyuti
// CED19I027
// Lab 5 : Q3

// Inclusion of required libraries
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>
#include <errno.h>
```

```c
#include<fcntl.h>


int main(int argc, char *argv[])
{
    // Opendir and readdir to display the names of files in a particular
directory

printf("\n--------------------------------------------------------------------\n");
    printf("\n3.A : opendir() and readdir()\n");
    printf("\nTo display all the files in a directory\n");

    DIR *folder;
    struct dirent *dirp;

    // If no file is mentioned in command line, show error
    if (argc != 2)
    {
        fprintf(stderr, "Wrong command line input by user\n");
        return 0;
    }

    folder = opendir(argv[1]);

    while ((dirp = readdir(folder)) != NULL)
        printf("%s\n", dirp -> d_name);

    // Close directory
    closedir(folder);

    // open () and close ()

printf("\n--------------------------------------------------------------------\n");
    printf("\n3.B : open() and close()\n");
    int fd1 = open("openclose.txt", O_RDONLY);
    int fd2 = open("openclose.txt", O_WRONLY);
    if (fd1 < 0)
    {
        perror("c1");
```

```
        exit(1);
    }
    if (fd2 < 0)
    {
        perror("c1");
        exit(1);
    }
    printf("opened the fd1 = % d\n", fd1);
    printf("opened the fd2 = % d\n", fd2);
    // Using close system Call
    if (close(fd1) < 0)
    {
        perror("c1");
        exit(1);
    }
    printf("closed the fd1.\n");
    if (close(fd2) < 0)
    {
        perror("c1");
        exit(1);
    }
    printf("closed the fd2.\n");


printf("\n----------------------------------------------------------------\n");
    printf("\n3.C : read() and wrie()\n");
    int fd, sz;
    char *c = (char *) calloc(100, sizeof(char));

    fd = open("openclose.txt", O_RDONLY);
    if (fd < 0) { perror("r1"); exit(1); }

    sz = read(fd, c, 200);
    printf("In the opened file, %d bytes were read.\n", sz);
    c[sz] = '\0';
    printf("Those bytes are as follows: \n %s\n", c);

    return 0;
}
```

OUTPUT :


dhyuti_n@dhyuti-VirtualBox: ~/OSLab/lab5
dhyuti_n@dhyuti-VirtualBox:~/OSLab/lab5$ ./program folder

3.A : opendir() and readdir()
To display all the files in a directory
hey (copy).txt
hello (copy).txt
hey.txt
.
..
hello.txt

3.B : open() and close()
opened the fd1 = 3
opened the fd2 = 4
closed the fd1.
closed the fd2.

3.C : read() and wrie()
In the opened file, 66 bytes were read.
Those bytes are as follows:
Hi
My name is Dhyuti.
I'm typing this to fill this text document.
dhyuti_n@dhyuti-VirtualBox:~/OSLab/lab5$

N. Sree Dhyuti
CED19I027

(3) (a) opendir() & readdir()
opendir & readdir are system calls available in the "dirent.h" library.
opendir() returns a pointer of DIR Type.
DIR is used to point files & folders.
readdir() will read the names of the files that are being pointed by dirent → d_name.
In a nutshell, opendir() & readdir() are used to traverse through directories or folders.

(b) Open() creates a new file & there are different modes to access them:
O-RDONLY ⇒ read only mode
O-RDWR ⇒ Read & write mode
O-WRONLY ⇒ write only mode
~~reads) reads from the mentioned files & is used to handle data~~
close() is used to close a file actively used in the program

(c) read() reads from the files used/iterated & write() is used to load or even alter that data

---

(4) Write two different programs (P1 and P2) in C to demonstrate IPC using shared memory. Process P1 will write 100 bytes of data to the shared memory and P2 will read from the shared memory and display it. Then, P2 will write 100 bytes of data to the same shared memory and P1 will read and display it.

SERVER CODE :

```c
// N Sree Dhyuti
// CED19I027
// Lab 5 : Q4 Server

// Inclusion of required libraries
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>
#include <unistd.h>

#define FILLED 0
#define Ready 1
#define NotReady -1

struct memory
{
    //message
    char msg[100];
    //pids of both processes and status for synchronization
    int status, pid1, pid2;
}*ptr;

// Function to print message from client
void handler(int signum)
{
    // if TRUE, then server receives a message from client
    if (signum == SIGUSR1)
    {
        printf("Received message from CLient: ");
        puts(ptr->msg);
    }
}
```

```c
int main()
{
    // pid for server
    int pid = getpid();

    // key value of shared memory
    int key = 14534;

    //while loop condition
    int codn = 0;

    //SM create
    int shmid = shmget(key, sizeof(struct memory), IPC_CREAT | 0666);

    //attach SM
    ptr = (struct memory*)shmat(shmid, NULL, 0);

    //store the process id of server in SM
    ptr->pid1 = pid;
    ptr->status = NotReady;

    // calling the signal function using signal type SIGUSER1
    signal(SIGUSR1, handler);

    while (codn < 1)
    {
        while (ptr->status != Ready)
            continue;
        sleep(1);

        //input from server
        printf("Server: ");
        fgets(ptr->msg, 100, stdin);

        ptr->status = FILLED;

        //sending the message to client using kill function
        kill(ptr->pid2, SIGUSR2);
```

```
        codn++;
    }

    shmdt((void*)ptr);
    shmctl(shmid, IPC_RMID, NULL);
    return 0;
}
```

CLIENT CODE :

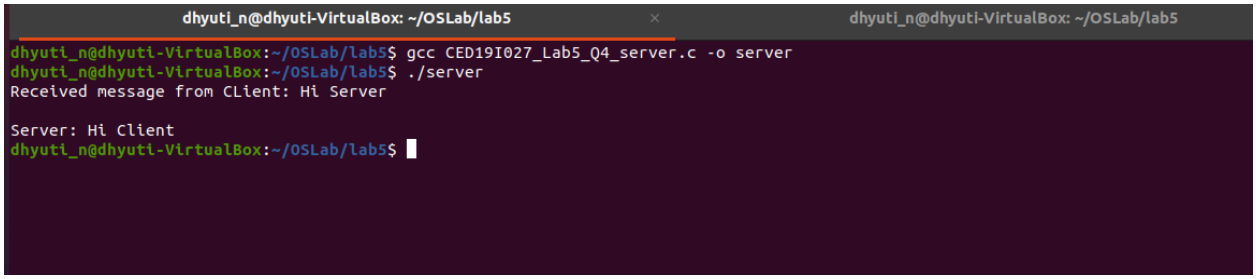
```
// N Sree Dhyuti
// CED19I027
// Lab 5 : Q4 Client

// Inclusion of required libraries
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>
#include <unistd.h>

#define FILLED 0
#define Ready 1
#define NotReady -1

struct memory
{
    char msg[100];
    int status, pid1, pid2;
}*ptr;

// Function to print message received from Server
void handler(int signum)
{
    //if TRUE, then client received message from server
    if (signum == SIGUSR2)
    {
```

```
        printf("Received message from Server: ");
        puts(ptr->msg);
    }
}

int main()
{
    //process id of client
    int pid = getpid();

    //key value of SM
    int key = 14534;

    //while loop break codn
    int codn = 0;

    //create SM
    int shmid = shmget(key, sizeof(struct memory), IPC_CREAT | 0666);

    //attach SM
    ptr = (struct memory*)shmat(shmid, NULL, 0);

    //store the pid of client in SM
    ptr->pid2 = pid;

    ptr->status = NotReady;

    signal(SIGUSR2, handler);

    //runs while 1 time.
    while (codn < 1)
    {
        sleep(1);

        // taking input from client
        printf("CLient: ");
        fgets(ptr->msg, 100, stdin);
        ptr->status = Ready;
```

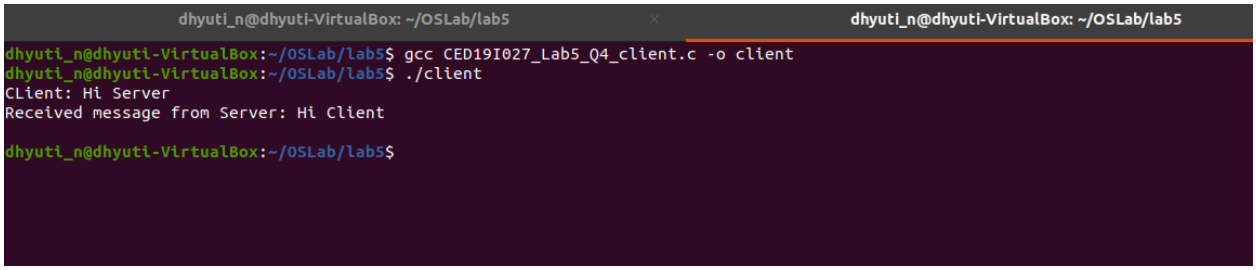
```
        //sending the message to server using kill
        kill(ptr->pid1, SIGUSR1);

        while (ptr->status == Ready)
            continue;
        codn++;
    }

    shmdt((void*)ptr);
    return 0;
}
```

OUTPUT :

Server :



Client :



N. Sree Dhyuti
CED19I027

(4) IPC using shared memory

Explanation of Code

server

(i) Create a key to access the shared memory for both P1 & P2.
(ii) Create a shared memory using shmget();
(iii) Create a pointer 'ptr' that points at the shared memory
(iv) Call the signal function using SIGUSER1
(v) Wait for a message from client.
(vi) Once message is sent, update the status of shared memory and clear the data now.

Client:

(i) Follow steps (i), (ii), (iii) just like server.
(ii) Take input message from user & send it to server using kill().
(iii) Wait for message acknowledgement & return message from server.

THE END