# Assignment # 01

**Fall 2023**

**CSE-302 Systems Programming**


Submitted by: **Ali Asghar**

Registration No.: **21PWCSE2059**

Class Section: **C**


Submitted to:

**Dr. Madiha Sher**

Date:

**25th January 2024**


**Department of Computer Systems Engineering**

**University of Engineering and Technology, Peshawar**

# SP Assignment # 1

**Q1:** Write a program that searches for a file passed to it as a command line argument in all the provided paths. Take paths as CLA.

**Sample Run:**

./find . .. ~/Desktop

**Code:**

```c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<fcntl.h>
#include<dirent.h>
#include<string.h>

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

        struct dirent *direntp;

        for(int i=2; i < argc; i++){

                DIR *dirp = opendir(argv[i]);

                if(dirp == NULL)
                        return -1;

                while((direntp = readdir(dirp)) != NULL){


                        if(strcmp(direntp->d_name, ".") == 0 || strcmp(direntp->d_name, "..") == 0)
                                continue;

                        if(strcmp(direntp->d_name, argv[1]) == 0){
                                printf("File %s Found in %s\n", argv[1], argv[i]);
                        }
                }

        }
        return 0;
}
```

**Output:**

```
ali@Ubuntu:~/Desktop/SP Theory/Assignment$ gcc Q1.c -o Q1.o
ali@Ubuntu:~/Desktop/SP Theory/Assignment$ ./Q1.o Q1.o . .. /home
File Q1.o Found in .
ali@Ubuntu:~/Desktop/SP Theory/Assignment$
```

**Q2:** Write a program to implement ls command. Take the name of the directory to be listed from command line. Also print the path of CWD.

**Sample Run:**

./t1.o SP

**Code:**

```c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<dirent.h>
#include<string.h>

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

        struct dirent *direntp;
        char bff[100];

        DIR *dirp = opendir(argv[1]);

        if(dirp == NULL){
                fprintf(stderr, "Error getting current path\n");
                return -1;
        }

        if(getcwd(bff, sizeof(bff)) == NULL){
                fprintf(stderr, "Error getting current path\n");
                return -1;
        }

        printf("CURRENT WORKING DIR = %s\n", bff);

        while((direntp = readdir(dirp)) != NULL){

                if(strcmp(direntp->d_name, ".") == 0 || strcmp(direntp->d_name, "..") == 0)
                        continue;

                printf("%s  ",direntp->d_name);
        }

        printf("\n");
        return 0;
}
```

**Output:**

```
ali@Ubuntu:~/Desktop/SP Theory/Assignment$ gcc Q2.c -o Q2.o
ali@Ubuntu:~/Desktop/SP Theory/Assignment$ ./Q2.o ~
CURRENT WORKING DIR = /home/ali/Desktop/SP Theory/Assignment
.pam_environment  .gnupg  .local  .vboxclient-hostversion-tty2-control.pid  .vboxclient-clipboard-tty2-control.p
id  .profile  .vboxclient-seamless-tty2-control.pid  Public  .bash_history  Pictures  .mozilla  .cache  Download
s  .python_history  .vboxclient-draganddrop-tty2-control.pid  Documents  Music  .vboxclient-vmsvga-session-tty2-
service.pid  .vboxclient-clipboard-tty2-service.pid  .vboxclient-seamless-tty2-service.pid  Videos  hello.py  De
sktop  .config  .vboxclient-vmsvga-session-tty2-control.pid  .vboxclient-draganddrop-tty2-service.pid  .bash_log
out  Templates  .bashrc  .ssh
ali@Ubuntu:~/Desktop/SP Theory/Assignment$ ./Q2.o .
CURRENT WORKING DIR = /home/ali/Desktop/SP Theory/Assignment
Q8.c  f1.txt  Q1.c  Q8.o  Q6.o  Q6.c  Q8_copy.c  Q2.o  Q1.o  Q4  Q5.o  Q8_copy.o  Q3.o  Q8 (copy).c  Q7.c  Quiz1
.c  Q2.c  Q5.c  Q3.c  Q9.c  Q9.o  .f1.txt.swp
ali@Ubuntu:~/Desktop/SP Theory/Assignment$ S
```

**Q3:** Write a program that finds a file in a directory. Program shall receive the name of the file & directory from command line.

**Sample Run:**

./find.o SP task1.c

**Code:**

Q2.c

```c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<fcntl.h>
#include<dirent.h>
#include<string.h>

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


        struct dirent *direntp;
        int flag = 0;
        DIR *dirp = opendir(argv[1]);

        if(dirp == NULL)
                return -1;

        while((direntp = readdir(dirp)) != NULL){


                if(strcmp(direntp->d_name, ".") == 0 || strcmp(direntp->d_name, "..") == 0)
                        continue;

                if(strcmp(direntp->d_name, argv[2]) == 0){
                        flag = 1;
                        break;
                }
        }

        if(flag)
                printf("File %s Found in %s\n", argv[2], argv[1]);
        else
                printf("File %s Not Found in %s\n", argv[2], argv[1]);

        return -1;
}
```

**Output:**

ali@Ubuntu: ~/Desktop/SP Theory/Assignment

```
ali@Ubuntu:~/Desktop/SP Theory/Assignment$ gcc Q3.c -o Q3.o
ali@Ubuntu:~/Desktop/SP Theory/Assignment$ ./Q3.o . Q2.c
File Q2.c Found in .
ali@Ubuntu:~/Desktop/SP Theory/Assignment$ gcc Q3.c -o Q3.o
ali@Ubuntu:~/Desktop/SP Theory/Assignment$ ./Q3.o . Q2.c
File Q2.c Found in .
ali@Ubuntu:~/Desktop/SP Theory/Assignment$ ./Q3.o ~ Q2.c
File Q2.c Not Found in /home/ali
ali@Ubuntu:~/Desktop/SP Theory/Assignment$ S
```

**Q4:** Write a program that implements FTP Server. Client requests for the contents of a specific directory. Server responds with the list of files/directories.

## Using FIFOs

## Code:

Q4_Client.c | **Q4_Server.c**

```c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<fcntl.h>
#include<dirent.h>
#include<string.h>
#include<sys/stat.h>
#include<errno.h>

int perm = S_IRWXU;
int bw, br, fd;

void list(char* buff){

	struct dirent *direntp;
	DIR *dirp = opendir(buff);

	if(dirp == NULL){
		perror("opendir");
		return;
	}

	while((direntp = readdir(dirp)) != NULL){
		if(strcmp(direntp->d_name, ".") == 0 || strcmp(direntp->d_name, "..") == 0)
			continue;

		bw = write(fd, direntp->d_name, strlen(direntp->d_name) + 1);
		if(bw == -1){
			perror("write");
			return;
		}
		bw = write(fd, " ", 2);
		if(bw == -1){
			perror("write");
			return;
		}
	}

	if(closedir(dirp) == -1){
		perror("closedir");
		return;
	}
```

Q4_Client.c | Q4_Server.c

```c
                bw = write(fd,     , 2);
                if(bw == -1){
                        perror("write");
                        return;
                }
        }

        if(closedir(dirp) == -1){
                perror("closedir");
                return;
        }
}

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

        char buff[100];

        int m = mkfifo("myfifo1", perm);
        if(m == -1 && errno != EEXIST){
                perror("mkfifo");
                return -1;
        }

        fd = open("myfifo1", O_RDWR);
        if(fd == -1){
                perror("open");
                return -1;
        }

        br = read(fd, buff, 100);
        if(br > 0){
                list(buff);
                printf("Successfully Written names\n");
        }

        if(close(fd) == -1){
                perror("close");
                return -1;
        }

        return 0;
}
```

Server Code

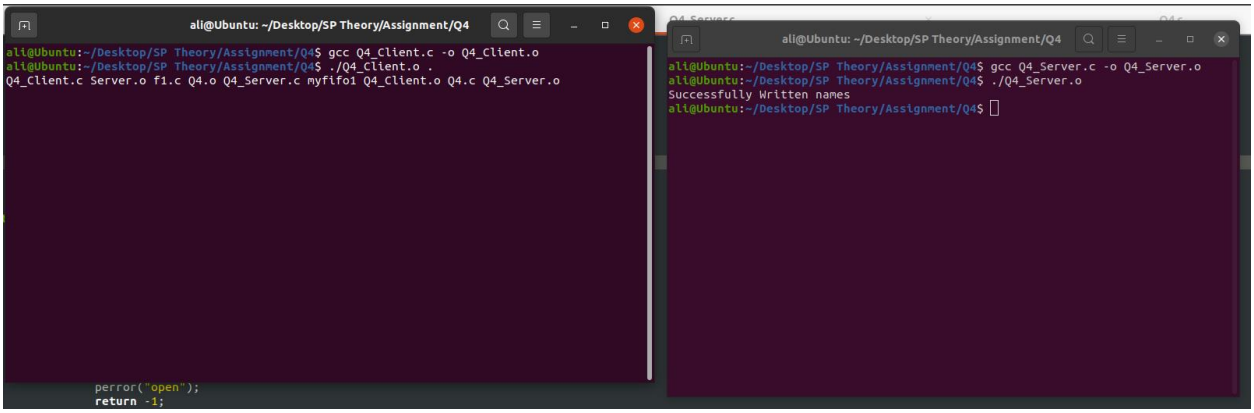
Q4_Client.c  Q4_Server.c

```c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<fcntl.h>
#include<dirent.h>
#include<string.h>
#include<errno.h>
#include<sys/stat.h>

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

        char buff[100];
        int perm = S_IRWXU;
        int bw, br;
        int m = mkfifo("myfifo1", perm);

        if(m < 0 && errno != EEXIST){
                perror("mkfifo");
                return -1;
        }

        int fd = open("myfifo1", O_RDWR);

        if(fd == -1){
                perror("open");
                return -1;
        }

        bw = write(fd, argv[1], strlen(argv[1])+1);
        if(bw == -1){
                perror("write");
                return -1;
        }

        sleep(1);
        while((br = read(fd, buff, 100)) != 0){
                if(br == -1){
                        perror("read");
                        return -1;
                }
                bw = write(STDOUT_FILENO, buff, br);
                if(bw == -1){
```

```c
                if(br == -1){
                        perror("read");
                        return -1;
                }
                bw = write(STDOUT_FILENO, buff, br);
                if(bw == -1){
                        perror("write");
                        return -1;
                }
        }

        if(close(fd) == -1){
                perror("close");
                return -1;
        }

        return 0;
}
```

Client Code

## Output:




## Using Pipes

## Code:

Q4.c | Q4_Client.c

```c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<fcntl.h>
#include<dirent.h>
#include<string.h>
#include<sys/stat.h>
#include<sys/wait.h>
#include<errno.h>

int bw, br;
int fd1[2];
int fd2[2];

void list(char* buff){

        int bw;
        DIR *dirp = opendir(buff);
        struct dirent *direntp;

        if(dirp == NULL){
                perror("opendir");
                return;
        }

        while((direntp = readdir(dirp)) != NULL){
                if(strcmp(direntp->d_name, ".") == 0 || strcmp(direntp->d_name, "..") == 0)
                        continue;

                bw = write(fd1[1], direntp->d_name, strlen(direntp->d_name) + 1);
                if(bw == -1){
                        perror("write");
                        return;
                }
                bw = write(fd1[1], " ", 2);
                if(bw == -1){
                        perror("write");
                        return;
                }
        }

        if(closedir(dirp) == -1){
```

Q4.c × Q4_Client.c

```c
        }

        if(closedir(dirp) == -1){
                perror("closedir");
                return;
        }
}

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

        char buff[100];

        strcpy(buff, argv[1]);

        int p1 = pipe(fd1);
        if(p1 == -1){
                perror("pipe");
                return -1;
        }

        int p2 = pipe(fd2);
        if(p2 == -1){
                perror("pipe");
                return -1;
        }

        int x = fork();

        if(x == -1){
                perror("fork");
                return -1;
        }

        if(x == 0){
                printf("CHILD CODE\n");
                bw = write(fd2[1], "CONTENTS", 9);
                if(bw == -1){
                        perror("write");
                        return -1;
                }
                br = read(fd1[0], buff, 100);
                if(br == -1){
                        perror("read");
```

```c
                br = read(fd1[0], buff, 100);
                if(br == -1){
                        perror("read");
                        return -1;
                }
                bw = write(STDOUT_FILENO, buff, 100);
                if(bw == -1){
                        perror("write");
                        return -1;
                }
        }
        else{
                printf("PARENT CODE\n");
                char pipebuff[100]; // create a new buffer for pipe
                br = read(fd2[0], pipebuff, 100); // read into the new buffer
                if(br == -1){
                        perror("read");
                        return -1;
                }

                if(strcmp(pipebuff, "CONTENTS")==0){
                        list(buff); // buff still contains the directory name
                }
        }
        return 0;
}
```

**Output:**

```
ali@Ubuntu: ~/Desktop/SP Theory/Assignment/Q4
ali@Ubuntu:~/Desktop/SP Theory/Assignment/Q4$ gcc Q4.c -o Q4.o
ali@Ubuntu:~/Desktop/SP Theory/Assignment/Q4$ ./Q4.o .
PARENT CODE
CHILD CODE
Q4_Client.c Server.o f1.c Q4.o Q4_Server.c myfifo1 Q4_Client.o Q4.c Q4_Server.o ali@Ubuntu:~/Des
ktop/SP Theory/Assignment/Q4$
```

**Q5:** Write a program that implements a simple FTP Server. Client requests for a file and server responds with the contents of the file. Client shall receive the contents and display on STD_OUT.

**Code:**

**Using FIFOs**

Q5_Server.c

```c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<fcntl.h>
#include<dirent.h>
#include<string.h>
#include<sys/stat.h>
#include<errno.h>

int perm = S_IRWXU;
int fd;

void read_file(char* filename){
        char buff2[10];
        int bw, br;

        int fd_file = open(filename, O_RDONLY);

        while((br = read(fd_file, buff2, 10)) !=0){
                bw = write(fd, buff2, br);
        }
}


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

        int bw, br;
        char buff[100];

        int m = mkfifo("myfifo1", perm);
        if(m == -1 && errno != EEXIST){
                perror("mkfifo");
                return -1;
        }

        fd = open("myfifo1", O_RDWR);
        if(fd == -1){
                perror("open");
                return -1;
        }

        br = read(fd, buff, 100);
        if(br > 0){
```

```c
        br = read(fd, buff, 100);
        if(br > 0){
                read_file(buff);
                printf("Successfully Read File\n");
        }

        if(close(fd) == -1){
                perror("close");
                return -1;
        }

        return 0;
}
```

Server Code

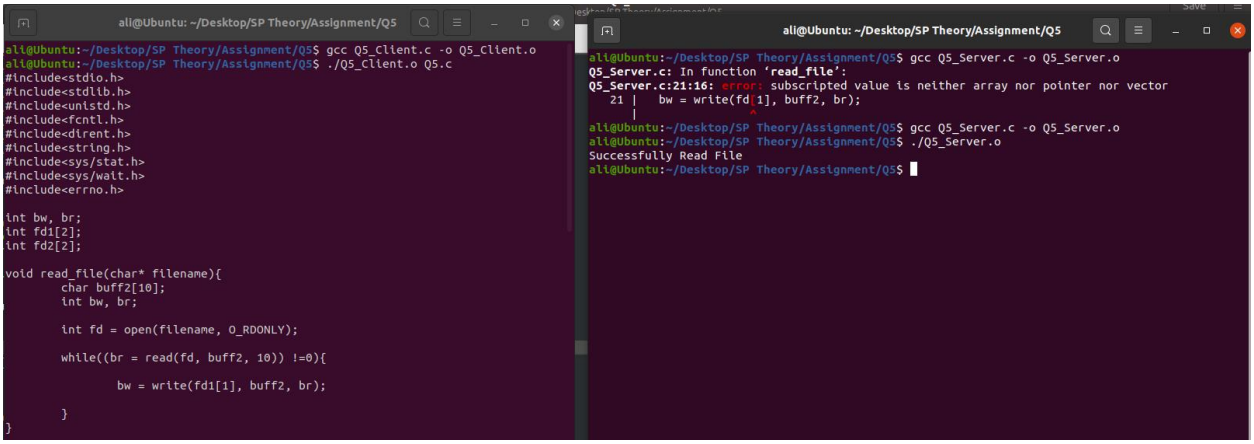
Q5_Server.c | Q5_Client.c

```c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<fcntl.h>
#include<dirent.h>
#include<string.h>
#include<errno.h>
#include<sys/stat.h>

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

        char buff[100];
        int perm = S_IRWXU;
        int bw, br;
        int m = mkfifo("myfifo1", perm);

        if(m < 0 && errno != EEXIST){
                perror("mkfifo");
                return -1;
        }

        int fd = open("myfifo1", O_RDWR);

        if(fd == -1){
                perror("open");
                return -1;
        }

        bw = write(fd, argv[1], strlen(argv[1])+1);
        if(bw == -1){
                perror("write");
                return -1;
        }

        sleep(1);
        while((br = read(fd, buff, 100)) != 0){
                if(br == -1){
                        perror("read");
                        return -1;
                }
                bw = write(STDOUT_FILENO, buff, br);
                if(bw == -1){
```

```c
                }
                bw = write(STDOUT_FILENO, buff, br);
                if(bw == -1){
                        perror("write");
                        return -1;
                }
        }

        if(close(fd) == -1){
                perror("close");
                return -1;
        }

        return 0;
}
```

Client Code

## Output:




## Using Pipes

## Code:

Q5.c | Q5_Server.c

```c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<fcntl.h>
#include<dirent.h>
#include<string.h>
#include<sys/stat.h>
#include<sys/wait.h>
#include<errno.h>

int bw, br;
int fd1[2];
int fd2[2];

void read_file(char* filename){
        char buff2[10];
        int bw, br;

        int fd = open(filename, O_RDONLY);
        if(fd == -1){
                perror("open");
                exit(-1);
        }

        while((br = read(fd, buff2, 10)) !=0){
                if(br == -1){
                        perror("read");
                        exit(-1);
                }
                bw = write(fd1[1], buff2, br);
                if(bw == -1){
                        perror("write");
                        exit(-1);
                }
        }

        if(close(fd1[1]) == -1){ // Close the write end of the pipe
                perror("close");
                exit(-1);
        }
}
```

Q5.c | Q5_Server.c

```c
}

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

        char buff[100];

        int p1 = pipe(fd1);
        if(p1 == -1){
                perror("pipe");
                exit(-1);
        }

        int p2 = pipe(fd2);
        if(p2 == -1){
                perror("pipe");
                exit(-1);
        }

        int x = fork();
        if(x == -1){
                perror("fork");
                exit(-1);
        }

        if(x == 0){
                if(close(fd1[1]) == -1){ // Close the write end of the pipe in the child process
                        perror("close");
                        exit(-1);
                }
                printf("CHILD CODE\n");
                bw = write(fd2[1], argv[1], strlen(argv[1]) + 1);
                if(bw == -1){
                        perror("write");
                        exit(-1);
                }

                sleep(2);

                while((br = read(fd1[0], buff, 100)) != 0){
                        if(br == -1){
                                perror("read");
                                exit(-1);
                        }
```

```c
                        perror("read");
                        exit(-1);
                }
                bw = write(STDOUT_FILENO, buff, br);
                if(bw == -1){
                        perror("write");
                        exit(-1);
                }
        }

        printf("CHILD TERMINATED\n");
}
else{
        if(close(fd2[1]) == -1){ // Close the write end of the pipe in the parent process
                perror("close");
                exit(-1);
        }
        printf("PARENT CODE\n");
        char pipebuff[100]; // create a new buffer for pipe

        br = read(fd2[0], pipebuff, 100); // read into the new buffer
        if(br == -1){
                perror("read");
                exit(-1);
        }

        if(strcmp(pipebuff, argv[1]) == 0){
                read_file(pipebuff); // buff still contains the directory name
                //printf("INSIDE STRCMP\n");
        }

        int r = wait(NULL);
        if(r == -1){
                perror("wait");
                exit(-1);
        }
        printf("PARENT TERMINATED\n");
}
return 0;
```

## Output:

```
ali@Ubuntu:~/Desktop/SP Theory/Assignment/Q5$ gedit f1.txt
ali@Ubuntu:~/Desktop/SP Theory/Assignment/Q5$ gcc Q5.c -o Q5.o
ali@Ubuntu:~/Desktop/SP Theory/Assignment/Q5$ ./Q5.o f1.txt
PARENT CODE
CHILD CODE
This is my f1.
CHILD TERMINATED
PARENT TERMINATED
ali@Ubuntu:~/Desktop/SP Theory/Assignment/Q5$ S
```

```
ali@Ubuntu: ~/Desktop/SP Theory/Assignment/Q5
ali@Ubuntu:~/Desktop/SP Theory/Assignment/Q5$ ./Q5.o Q5.c
PARENT CODE
CHILD CODE
INSIDE STRCMP
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<fcntl.h>
#include<dirent.h>
#include<string.h>
#include<sys/stat.h>
#include<sys/wait.h>
#include<errno.h>

int bw, br;
int fd1[2];
int fd2[2];

void read_file(char* filename){
        char buff2[10];
        int bw, br;

        int fd = open(filename, O_RDONLY);
```

**Q6:** Write a program for continuous communication (2-Way) between parent & child process using pipes.

**Code:**

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<string.h>
#include<sys/wait.h>

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

        int fd1[2], fd2[2];
        int br, bw, w;
        char buff[100];

        int p1 = pipe(fd1);
        int p2 = pipe(fd2);


        fd_set myreadset;


        int x = fork();


        while(1){
```

```c
        while(1){
                FD_ZERO(&myreadset);

                FD_SET(fd1[0] , &myreadset);
                FD_SET(fd2[0] , &myreadset);
                FD_SET(STDIN_FILENO , &myreadset);

                int maxfd;
                if(fd1 > fd2 && fd1 > STDIN_FILENO)
                        maxfd = fd1[0];

                else if(fd2 > fd1 && fd2 > STDIN_FILENO)
                        maxfd = fd2[0];

                else
                        maxfd = STDIN_FILENO;



                if(x == 0){
```

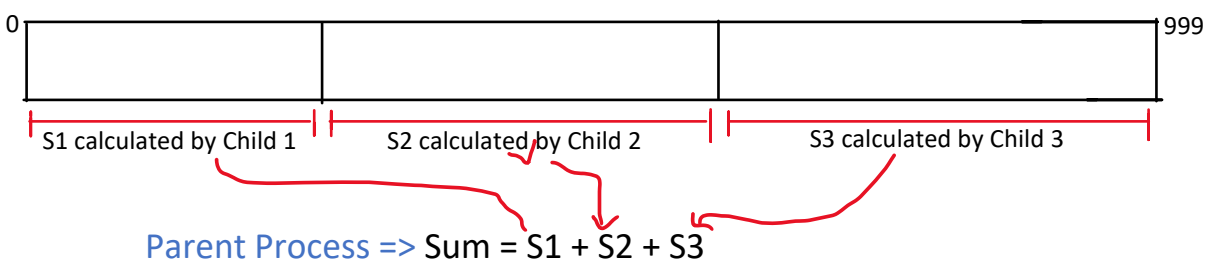
```c

                if(x == 0){


                        int nrf = select(maxfd + 1, &myreadset, NULL, NULL, NULL);

                        if(FD_ISSET(STDIN_FILENO, &myreadset)){

                                br = read(STDIN_FILENO, buff, 100);
                                bw = write(fd1[1], buff, br);
                        }

                        if(FD_ISSET(fd2[0], &myreadset)){

                                br = read(fd2[0], buff, 100);
                                if(br > 0)
                                        printf("Child Read Succesfully: \n");

                                write(STDOUT_FILENO, buff, br);
                        }
                }

                else{

                        int nrf = select(maxfd + 1, &myreadset, NULL, NULL, NULL);

                        if(FD_ISSET(fd1[0], &myreadset)){

                                br = read(fd1[0], buff, 100);
                                if(br > 0)
                                        printf("Parent Read Succesfully: \n");

                                write(STDOUT_FILENO, buff, br);
                        }

                        if(FD_ISSET(STDIN_FILENO, &myreadset)){

                                br = read(STDIN_FILENO, buff, 100);
                                write(fd2[1], buff, br);
                        }
                }
        }
        return 0;
```

**Output:**

```
ali@Ubuntu: ~/Desktop/SP Theory/Assignment
ali@Ubuntu:~/Desktop/SP Theory/Assignment$ ./Q6.o
Hi
Hello
Child Read Succesfully:
Hi
Parent Read Succesfully:
Hello
How are you?
Parent Read Succesfully:
How are you?
I am good
Child Read Succesfully:
I am good
```

**Q7:** Write a program for parallel array addition. The program must create 3 child processes and each child should calculate the sum of the one-third (1/3) of array elements. Parent process shall receive the sum calculated by each child, add them to get final sum and then display it. Make sure there are no orphan child processes.

You can use pipes, fifos or return value of child processes for Inter Process Communication.



## Code:

Q7.c | Q5.c | Q5_Server.c

```c
#include<stdio.h>
#include<sys/wait.h>
#include<unistd.h>

int main(int argc, char *argv[]){
        int arr[1000];
        int status, pid;
        int res = 0;
        int sum = 0;
        int total = 0;
        int i, fd[2];

        for(i = 0; i<1000; i++){
                arr[i] = i;
        }

        int p = pipe(fd);

        for(i = 0; i<3; i++){
                pid = fork();
                if(pid < 0){
                        perror("fork");
                        return 1;
                }

                if(pid == 0){ break; }

        }

        if(pid == 0 && i == 0){//First Child's Code
                for(int j = 0; j<=1000/3 *(1 + i); j++){
                        res += arr[j];
                }
                if(write(fd[1], &res, sizeof(sum)) < 0){
                        perror("write");
                        return 1;
                }
                return 0;
        }

        if(pid == 0 && i ==1){//Second Child's Code
                for(int j = 334; j<=1000/3 *(1 + i); j++){
```

Q7.c | Q5.c | Q5_Server.c

```c

        if(pid == 0 && i ==1){//Second Child's Code
                for(int j = 334; j<=1000/3 *(1 + i); j++){
                        res += arr[j];
                }

                if(write(fd[1], &res, sizeof(sum)) < 0){
                        perror("write");
                        return 1;
                }
                return 0;
        }

        if(pid == 0 && i ==2){//Third Child's Code
                for(int j = 667; j<=1000/3 *(1 + i); j++){
                        res += arr[j];
                }

                if(write(fd[1], &res, sizeof(sum)) < 0){
                        perror("write");
                        return 1;
                }

                return 0;
        }

        if(pid > 0){
                for(i = 0; i<3; i++){
                        int r = wait(&status);
                        if(r < 0){
                                perror("wait");
                                return 1;
                        }
                        if(WIFEXITED(status)){
                                printf("Child %d terminated with return status %d\n", i, WEXITSTATUS(status));

                                if(read(fd[0], &sum, sizeof(sum)) < 0){
                                        perror("read");
                                        return 1;
                                }
                                total += sum;
                        }
```

```c
                                }
                                total += sum;
                        }
                }

                printf("Sum = %d, ", total);
        }
        return 0;
}
```

## Output:

ali@Ubuntu: ~/Desktop/SP Theory/Assignment

```
ali@Ubuntu:~/Desktop/SP Theory/Assignment$ gcc Q7.c -o Q7.o
ali@Ubuntu:~/Desktop/SP Theory/Assignment$ ./Q7.o
Child 0 terminated with return status 0
Child 1 terminated with return status 0
Child 2 terminated with return status 0
Sum = 499500, ali@Ubuntu:~/Desktop/SP Theory/Assignment$
```

**Q8:** Write a program that creates a child process. Child process shall send “N” SIGUSR1 or SIGUSR2 to parent process. Parent process shall count the number of SIGUSR2 received.

**Code:**

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <sys/wait.h>

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <N>\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    int N = atoi(argv[1]);
    sigset_t set;
    int sig;

    /* Initialize the signal set */
    sigemptyset(&set);

    /* Add SIGUSR1 to the set */
    sigaddset(&set, SIGUSR1);

    /* Block SIGUSR1 */
    if (sigprocmask(SIG_BLOCK, &set, NULL) == -1) {
        perror("sigprocmask");
        exit(-1);
    }

    pid_t pid = fork();
    if (pid == -1) {
        perror("fork");
        exit(EXIT_FAILURE);
    }

    if (pid == 0) {  // Child process
        for (int i = 0; i < N; i++) {
            kill(getppid(), SIGUSR1);
        }
        exit(0);

    }
    else {  // Parent process
```

```c
    else {  // Parent process

        for (int i = 0; i < N; i++) {
                sigwait(&set, &sig);  // Wait for SIGUSR1
                //count++;
        }
        printf("Received %d SIGUSR1 signals\n", N);
        wait(NULL);  // Wait for child process to finish
    }

    return 0;
}
```

**Output:**

```
ali@Ubuntu:~/Desktop/SP Theory/Assignment$ gcc Q8_copy.c -o Q8_copy.o
ali@Ubuntu:~/Desktop/SP Theory/Assignment$ ./Q8_copy.o 3
Received 3 SIGUSR1 signals
ali@Ubuntu:~/Desktop/SP Theory/Assignment$
```

**Q9:** Write a program that creates a child process & waits for the child process to terminate using **pause/sigsuspend/sigwait.**

**Code:**

Q9.c  Q10.c

```c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<signal.h>

sigset_t myset;
int x;

void my_wait(){

        if(x > 0){
                sigsuspend(&myset);
        }

}

void my_handler(int sig_no){

        printf("\nChild has been terminated..\n");
}

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

        struct sigaction my_action;

        my_action.sa_flags = 0;
        my_action.sa_handler = my_handler;
        sigemptyset(&my_action.sa_mask);
```

```c
        sigemptyset(&my_action.sa_mask);

        sigemptyset(&myset);
        sigfillset(&myset);
        sigdelset(&myset, SIGCHLD);

        if(sigaction(SIGCHLD, &my_action, NULL) < 0){
                perror("sigaction");
                exit(-1);
        }

        x = fork();

        if(x < 0){
                perror("fork");
                exit(-1);
        }
        else if(x > 0){
                printf("\nParent Waiting..\n");
                my_wait();
                printf("\nParent Waited successfully..\n");
        }

        return 0;
}
```

**Output:**

```
ali@Ubuntu: ~/Desktop/SP Theory/Assignment
ali@Ubuntu:~/Desktop/SP Theory/Assignment$ gcc Q9.c -o Q9.o
ali@Ubuntu:~/Desktop/SP Theory/Assignment$ ./Q9.o

Parent Waiting..

CHild has been terminated..
ali@Ubuntu:~/Desktop/SP Theory/Assignment$ gcc Q9.c -o Q9.o
ali@Ubuntu:~/Desktop/SP Theory/Assignment$ ./Q9.o

Parent Waiting..

CHild has been terminated..

Parent Waited succesfully..
ali@Ubuntu:~/Desktop/SP Theory/Assignment$
```

**Q10:** Write a program that creates 2 threads.

**Thread 1:** Find sum of array elements.

**Thread 2:** Searches for a key in array.

## Code:

Open

Q10.c
~/Desktop/SP Theory/Assignmen

```c
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>

int arr[100];
int sum = 0;
pthread_mutex_t lock;

void *SumArray(void* arg){
        int local_sum = 0;

        for(int i = 0; i<100; i++){
                local_sum +=arr[i];
        }

        pthread_mutex_lock(&lock);
        sum = local_sum;
        pthread_mutex_unlock(&lock);

        return NULL;
}

void *SearchArray(void* arg){

        for(int i = 0; i<100; i++){
                if(arr[i] == *(int *)arg){
                        pthread_mutex_lock(&lock);
                        printf("x = %d found at %d\n", *(int *)arg, i);
                        pthread_mutex_unlock(&lock);
                }
        }

        return NULL;
}

int main(){

        pthread_t tid[2];
        int x = 4;
        for(int i =0; i<100; i++){
                arr[i] = i;
        }

        if (pthread_mutex_init(&lock, NULL) != 0) {
        printf("\n mutex init has failed\n");
```

```c

        if (pthread_mutex_init(&lock, NULL) != 0) {
        printf("\n mutex init has failed\n");
        return 1;
    }

        if(pthread_create(&tid[0], NULL, SumArray, NULL) != 0){
                printf("\nThread creation failed. Exiting now.");
                return 1;
        }

        if(pthread_create(&tid[1], NULL, SearchArray, (void *)&x) != 0){
                printf("\nThread creation failed. Exiting now.");
                return 1;
        }


        for(int i =0; i<2; i++){
                if(pthread_join(tid[i], NULL) != 0){
                        printf("\nThread join failed. Exiting now.");
                        return 1;
                }
        }

        printf("Sum = %d\n", sum);
        pthread_mutex_destroy(&lock);
        return 0;
}
```

## Output:

```
ali@Ubuntu:~/Desktop/SP Theory/Assignment$ gcc Q10.c -o Q10.o -lpthread
ali@Ubuntu:~/Desktop/SP Theory/Assignment$ ./Q10.o
x = 4 found at 4
Sum = 4950
ali@Ubuntu:~/Desktop/SP Theory/Assignment$
```

**Q11:** Write a multithreaded program for parallel file copying. Open both source files in master thread before creating threads.

| Thread 1 | Thread 2 |
|---|---|
| S1 → D1 | S2 → D2 |

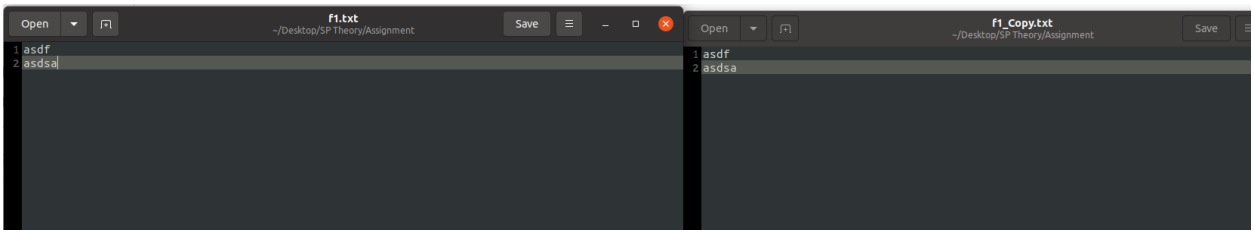
**Code:**

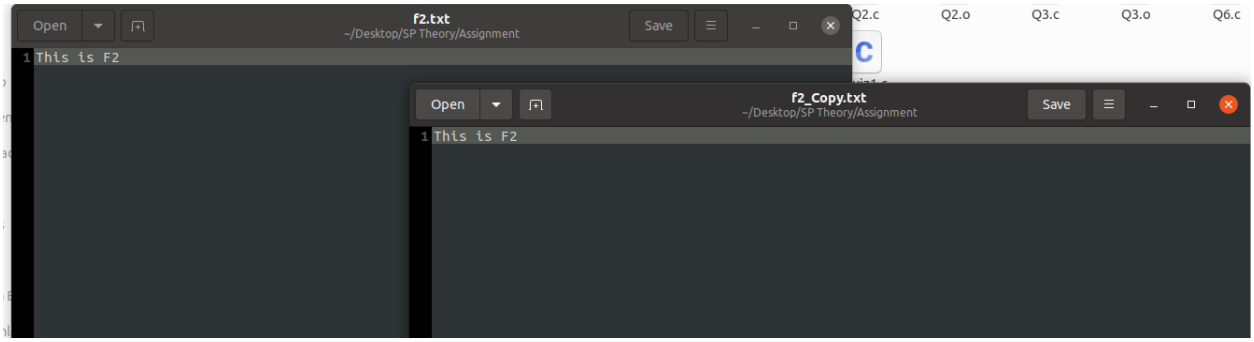
Q11.c  Q1.c

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

int fd_rd[2];
int fd_wr[2];

char *rd_names[2] = {"f1.txt","f2.txt"};
char *wr_names[2] = {"f1_Copy.txt","f2_Copy.txt"};

void *COPY(void* arg){
        int br, bw;
        char buff[100];

        while((br = read(fd_rd[*(int *)arg], buff, 100)) !=0){
                bw = write(fd_wr[*(int *)arg], buff, br);
        }
}

int main(){
        pthread_t tid[2];
        int perm = S_IRWXU;
        int args[2] ={0, 1};


        for(int i=0; i<2; i++){
                fd_rd[i] = open(rd_names[i], O_RDONLY);
                fd_wr[i] = open(wr_names[i], O_WRONLY | O_CREAT, perm);
        }

        for(int i =0; i<2; i++){
                pthread_create(&tid[i], NULL, COPY, (void *)&args[i]);
        }

        for(int i =0; i<2; i++){
                pthread_join(tid[i], NULL);
        }

        printf("Both Files Copied Successfully\n");

        return 0;
}
```

**Output:**

```
ali@Ubuntu:~/Desktop/SP Theory/Assignment$ gcc Q11.c -o Q11.o -lpthread
ali@Ubuntu:~/Desktop/SP Theory/Assignment$ ./Q11.o
Both Files Copied Successfully
ali@Ubuntu:~/Desktop/SP Theory/Assignment$
```


Open
f1.txt
~/Desktop/SP Theory/Assignment
Save
asdf
asdsa
Open
f1_Copy.txt
~/Desktop/SP Theory/Assignment
Save
asdf
asdsa


Open
f2.txt
~/Desktop/SP Theory/Assignment
Save
This is F2
Q2.c
Q2.o
Q3.c
Q3.o
Q6.c
Open
f2_Copy.txt
~/Desktop/SP Theory/Assignment
Save
This is F2