

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:

```
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<unistd.h>
4 #include<fcntl.h>
5 #include<dirent.h>
6 #include<string.h>
7
8 int main(int argc, char* argv[]){
9
10     struct dirent *direntp;
11
12     for(int i=2; i < argc; i++){
13
14         DIR *dirp = opendir(argv[i]);
15
16         if(dirp == NULL)
17             return -1;
18
19         while((direntp = readdir(dirp)) != NULL){
20
21
22             if(strcmp(direntp->d_name, ".") == 0 || strcmp(direntp->d_name, "..") == 0)
23                 continue;
24
25             if(strcmp(direntp->d_name, argv[i]) == 0){
26                 printf("File %s Found in %s\n", argv[i], argv[i]);
27             }
28         }
29     }
30     return 0;
31 }
32 }
```

Output:

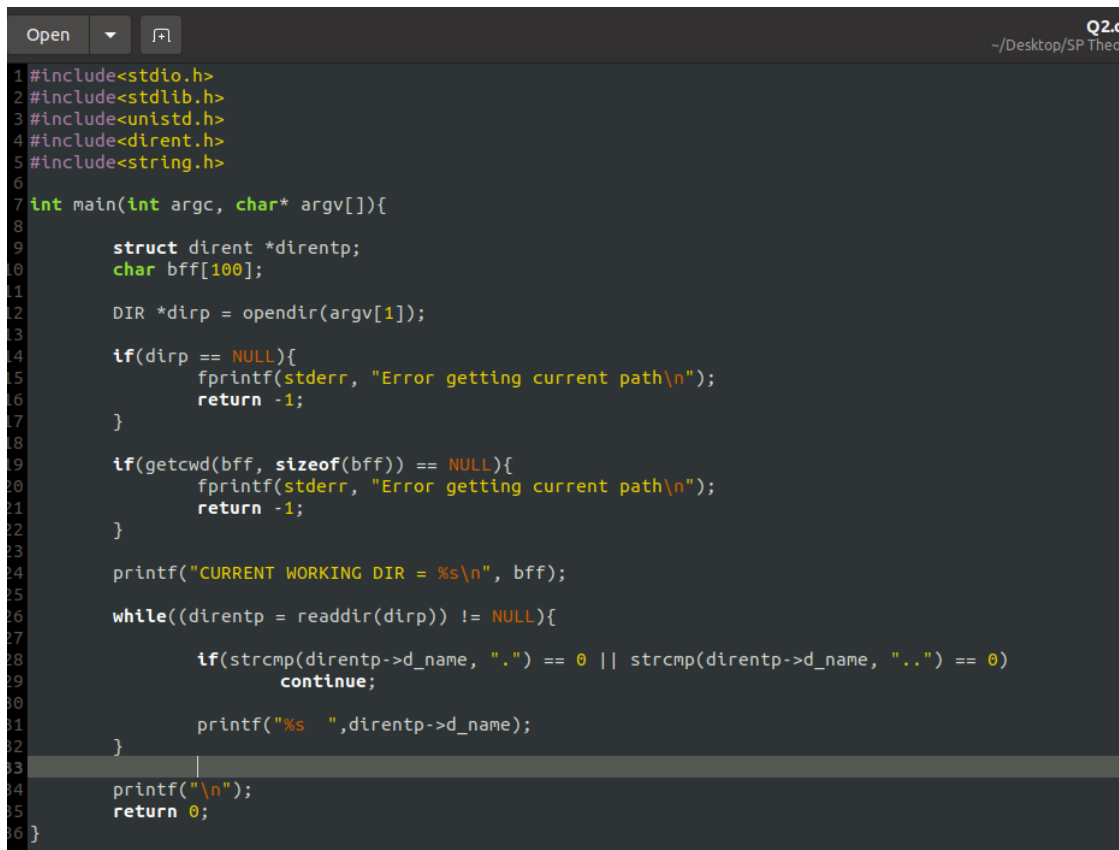
```
ali@Ubuntu:~/Desktop/SP Theory/Assignment$ gcc Q1.c -o Q1.o
ali@Ubuntu:~/Desktop/SP Theory/Assignment$ ./Q1.o Q1.o . .. /home
File Q1.o Found in .
ali@Ubuntu:~/Desktop/SP Theory/Assignment$
```

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:

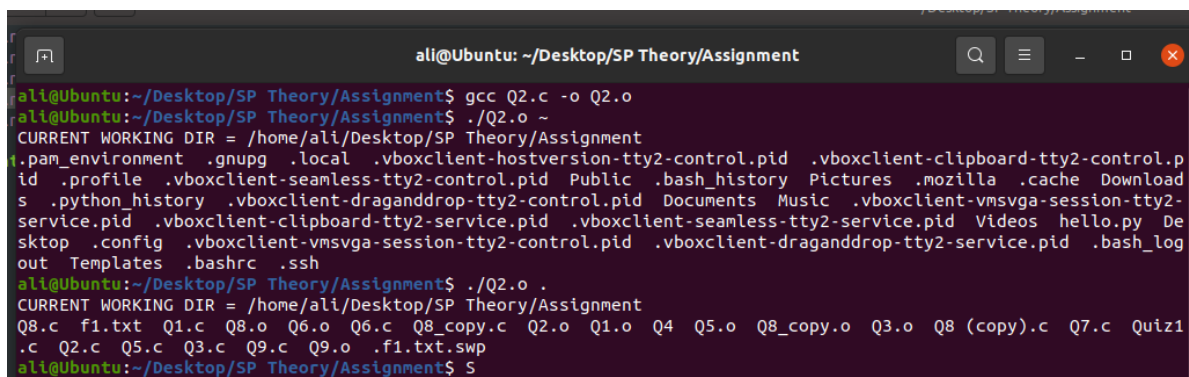
./t1.o SP

Code:



```
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<unistd.h>
4 #include<dirent.h>
5 #include<string.h>
6
7 int main(int argc, char* argv[]){
8
9     struct dirent *direntp;
10    char bff[100];
11
12    DIR *dirp = opendir(argv[1]);
13
14    if(dirp == NULL){
15        fprintf(stderr, "Error getting current path\n");
16        return -1;
17    }
18
19    if(getcwd(bff, sizeof(bff)) == NULL){
20        fprintf(stderr, "Error getting current path\n");
21        return -1;
22    }
23
24    printf("CURRENT WORKING DIR = %s\n", bff);
25
26    while((direntp = readdir(dirp)) != NULL){
27
28        if(strcmp(direntp->d_name, ".") == 0 || strcmp(direntp->d_name, "..") == 0)
29            continue;
30
31        printf("%s ",direntp->d_name);
32    }
33
34    printf("\n");
35    return 0;
36 }
```

Output:



```
ali@Ubuntu: ~/Desktop/SP Theory/Assignment
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-vmvga-session-tty2-
service.pid .vboxclient-clipboard-tty2-service.pid .vboxclient-seamless-tty2-service.pid Videos hello.py De
sktop .config .vboxclient-vmvga-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
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<unistd.h>
4 #include<fcntl.h>
5 #include<dirent.h>
6 #include<string.h>
7
8 int main(int argc, char* argv[]){
9
10     struct dirent *direntp;
11     int flag = 0;
12     DIR *dirp = opendir(argv[1]);
13
14     if(dirp == NULL)
15         return -1;
16
17     while((direntp = readdir(dirp)) != NULL){
18
19         if(strcmp(direntp->d_name, ".") == 0 || strcmp(direntp->d_name, "..") == 0)
20             continue;
21
22         if(strcmp(direntp->d_name, argv[2]) == 0){
23             flag = 1;
24             break;
25         }
26     }
27
28     if(flag)
29         printf("File %s Found in %s\n", argv[2], argv[1]);
30     else
31         printf("File %s Not Found in %s\n", argv[2], argv[1]);
32
33     return -1;
34 }
35
36 }
```

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
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<unistd.h>
4 #include<fcntl.h>
5 #include<dirent.h>
6 #include<string.h>
7 #include<sys/stat.h>
8 #include<errno.h>
9
10 int perm = S_IRWXU;
11 int bw, br, fd;
12
13 void list(char* buff){
14
15     struct dirent *direntp;
16     DIR *dirp = opendir(buff);
17
18     if(dirp == NULL){
19         perror("opendir");
20         return;
21     }
22
23     while((direntp = readdir(dirp)) != NULL){
24         if(strcmp(direntp->d_name, ".") == 0 || strcmp(direntp->d_name, "..") == 0)
25             continue;
26
27         bw = write(fd, direntp->d_name, strlen(direntp->d_name) + 1);
28         if(bw == -1){
29             perror("write");
30             return;
31         }
32         bw = write(fd, " ", 2);
33         if(bw == -1){
34             perror("write");
35             return;
36         }
37     }
38
39     if(closedir(dirp) == -1){
40         perror("closedir");
41         return;
42     }
43 }
```

```
Q4_Client.c  Q4_Server.c
2      bw = write(fd, buf, 2);
3      if(bw == -1){
4          perror("write");
5          return;
6      }
7  }
8
9      if(closedir(dirp) == -1){
10         perror("closedir");
11         return;
12     }
13 }
14
15 int main(int argc, char* argv[]){
16     char buff[100];
17
18     int m = mkfifo("myfifo1", perm);
19     if(m == -1 && errno != EEXIST){
20         perror("mkfifo");
21         return -1;
22     }
23
24     fd = open("myfifo1", O_RDWR);
25     if(fd == -1){
26         perror("open");
27         return -1;
28     }
29
30     br = read(fd, buff, 100);
31     if(br > 0){
32         list(buff);
33         printf("Successfully Written names\n");
34     }
35
36     if(close(fd) == -1){
37         perror("close");
38         return -1;
39     }
40
41     return 0;
42 }
```

Server Code

```

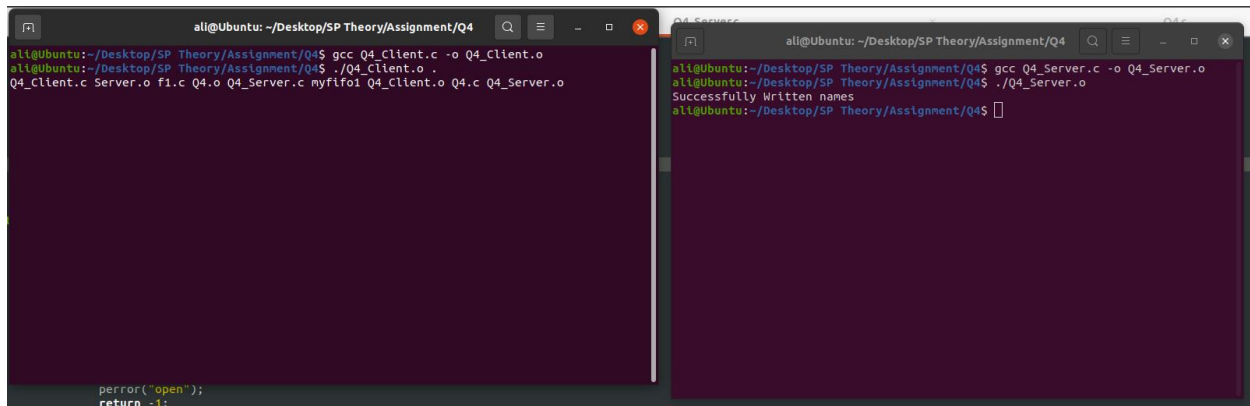
Q4_Client.c
Q4_Server.c

1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<unistd.h>
4 #include<fcntl.h>
5 #include<dirent.h>
6 #include<string.h>
7 #include<errno.h>
8 #include<sys/stat.h>
9
10 int main(int argc, char* argv[]){
11
12     char buff[100];
13     int perm = S_IRWXU;
14     int bw, br;
15     int m = mkfifo("myfifo1", perm);
16
17     if(m < 0 && errno != EEXIST){
18         perror("mkfifo");
19         return -1;
20     }
21
22     int fd = open("myfifo1", O_RDWR);
23
24     if(fd == -1){
25         perror("open");
26         return -1;
27     }
28
29     bw = write(fd, argv[1], strlen(argv[1])+1);
30     if(bw == -1){
31         perror("write");
32         return -1;
33     }
34
35     sleep(1);
36     while((br = read(fd, buff, 100)) != 0){
37         if(br == -1){
38             perror("read");
39             return -1;
40         }
41         bw = write(STDOUT_FILENO, buff, br);
42         if(bw == -1){
43             perror("write");
44             return -1;
45         }
46     }
47
48     if(close(fd) == -1){
49         perror("close");
50         return -1;
51     }
52
53     return 0;
54 }

```

Client Code

Output:



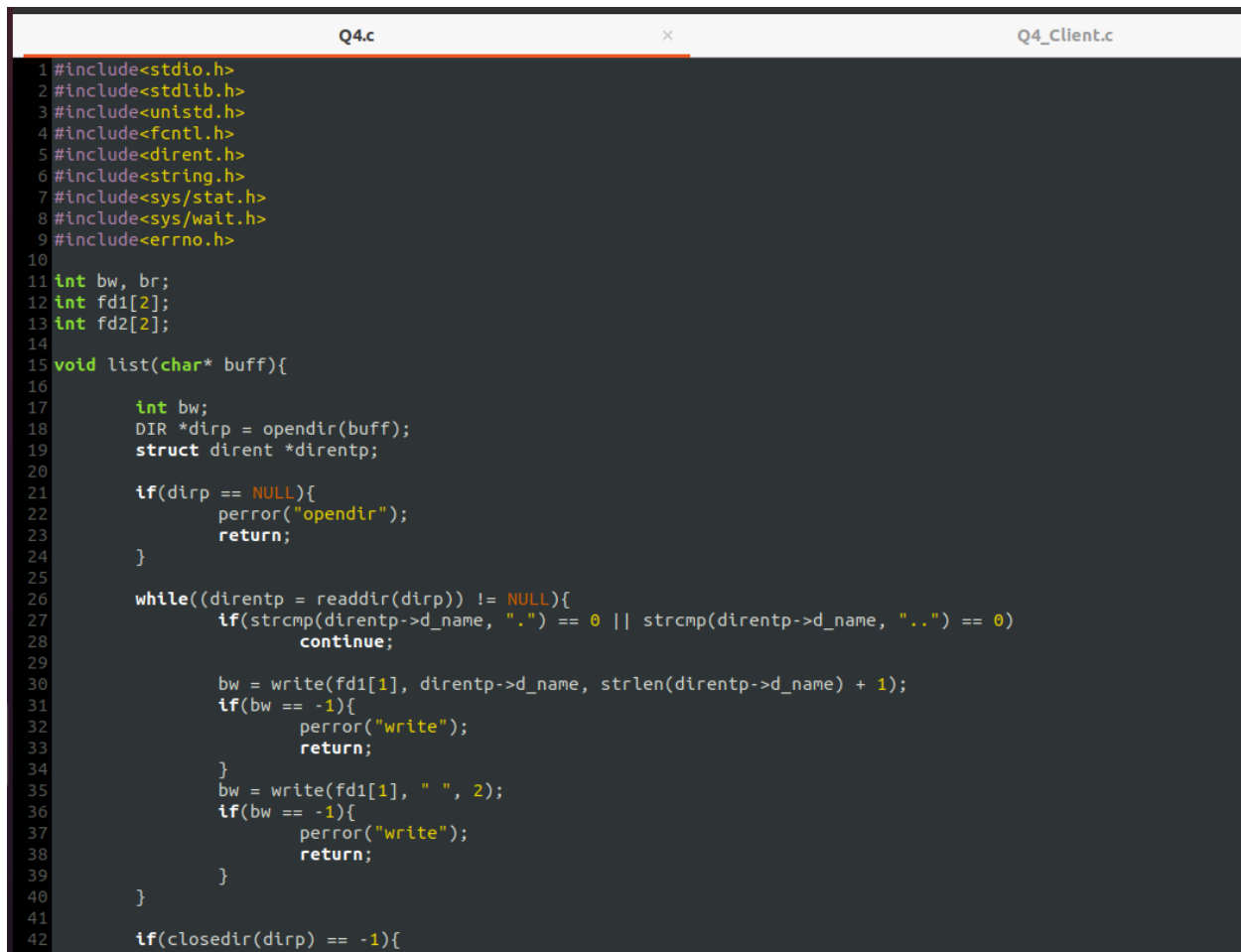
The image shows two terminal windows side-by-side. The left window shows the compilation of Q4_Client.c into Q4_Client.o and then the execution of Q4_Client.o, which outputs 'Successfully Written names'. The right window shows the compilation of Q4_Server.c into Q4_Server.o and then the execution of Q4_Server.o, which outputs 'Successfully Written names'.

```
ali@Ubuntu: ~/Desktop/SP Theory/Assignment/Q4$ gcc Q4_Client.c -o Q4_Client.o
ali@Ubuntu: ~/Desktop/SP Theory/Assignment/Q4$ ./Q4_Client.o
Q4_Client.c Server.o fi.c Q4.o Q4_Server.c myfifo1 Q4_Client.o Q4.c Q4_Server.o
Successfully Written names

ali@Ubuntu: ~/Desktop/SP Theory/Assignment/Q4$ gcc Q4_Server.c -o Q4_Server.o
ali@Ubuntu: ~/Desktop/SP Theory/Assignment/Q4$ ./Q4_Server.o
Successfully Written names
ali@Ubuntu: ~/Desktop/SP Theory/Assignment/Q4$
```

Using Pipes

Code:



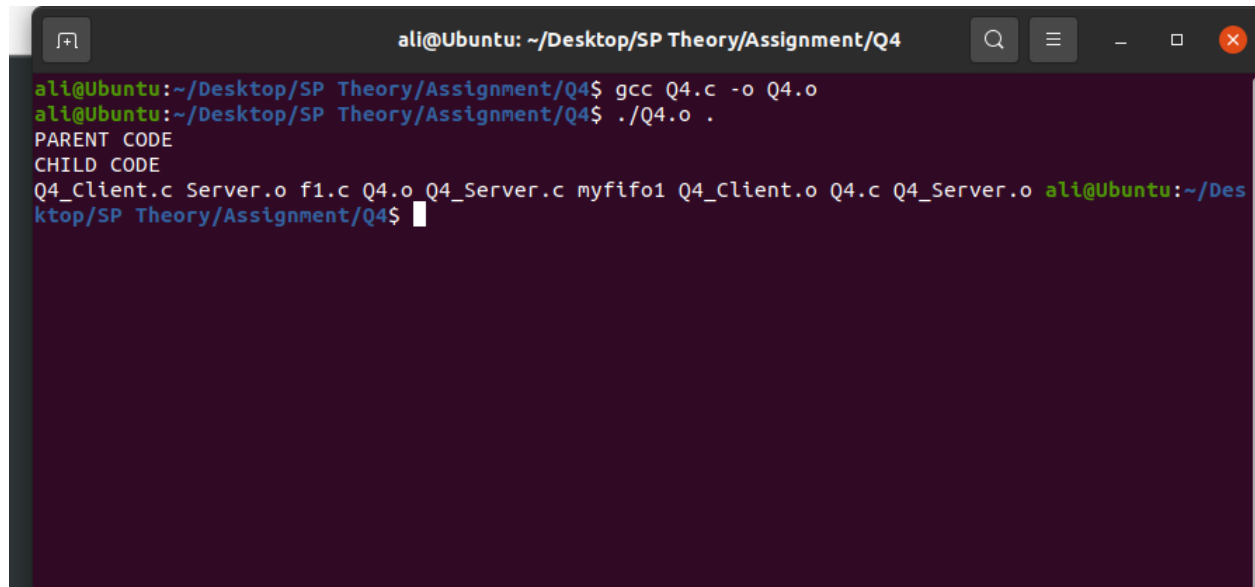
The image shows a code editor with two tabs: Q4.c and Q4_Client.c. The Q4.c tab is active, showing the source code for the Q4 program. The code includes headers for stdio, stdlib, unistd, fcntl, dirent, string, sys/stat, sys/wait, and errno. It defines a list function that takes a buffer and writes the names of files in the directory to the buffer. The Q4_Client.c tab is also visible, showing the client code.

```
Q4.c
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<unistd.h>
4 #include<fcntl.h>
5 #include<dirent.h>
6 #include<string.h>
7 #include<sys/stat.h>
8 #include<sys/wait.h>
9 #include<errno.h>
10
11 int bw, br;
12 int fd1[2];
13 int fd2[2];
14
15 void list(char* buff){
16
17     int bw;
18     DIR *dirp = opendir(buff);
19     struct dirent *direntp;
20
21     if(dirp == NULL){
22         perror("opendir");
23         return;
24     }
25
26     while((direntp = readdir(dirp)) != NULL){
27         if(strcmp(direntp->d_name, ".") == 0 || strcmp(direntp->d_name, "..") == 0)
28             continue;
29
30         bw = write(fd1[1], direntp->d_name, strlen(direntp->d_name) + 1);
31         if(bw == -1){
32             perror("write");
33             return;
34         }
35         bw = write(fd1[1], " ", 2);
36         if(bw == -1){
37             perror("write");
38             return;
39         }
40     }
41
42     if(closedir(dirp) == -1){
43         perror("closedir");
44         return;
45     }
46 }
```



```
40     }
41
42     if(closedir(dirp) == -1){
43         perror("closedir");
44         return;
45     }
46 }
47
48 int main(int argc, char* argv[]){
49     char buff[100];
50
51     strcpy(buff, argv[1]);
52
53     int p1 = pipe(fd1);
54     if(p1 == -1){
55         perror("pipe");
56         return -1;
57     }
58
59     int p2 = pipe(fd2);
60     if(p2 == -1){
61         perror("pipe");
62         return -1;
63     }
64
65     int x = fork();
66
67     if(x == -1){
68         perror("fork");
69         return -1;
70     }
71
72     if(x == 0){
73         printf("CHILD CODE\n");
74         bw = write(fd2[1], "CONTENTS", 9);
75         if(bw == -1){
76             perror("write");
77             return -1;
78         }
79         br = read(fd1[0], buff, 100);
80         if(br == -1){
81             perror("read");
82
83             br = read(fd1[0], buff, 100);
84             if(br == -1){
85                 perror("read");
86                 return -1;
87             }
88             bw = write(STDOUT_FILENO, buff, 100);
89             if(bw == -1){
90                 perror("write");
91                 return -1;
92             }
93         }
94         else{
95             printf("PARENT CODE\n");
96             char pipebuff[100]; // create a new buffer for pipe
97             br = read(fd2[0], pipebuff, 100); // read into the new buffer
98             if(br == -1){
99                 perror("read");
100                 return -1;
101             }
102             if(strcmp(pipebuff, "CONTENTS")==0){
103                 list(buff); // buff still contains the directory name
104             }
105             return 0;
106 }
```

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:~/Desktop/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
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<unistd.h>
4 #include<fcntl.h>
5 #include<dirent.h>
6 #include<string.h>
7 #include<sys/stat.h>
8 #include<errno.h>
9
10 int perm = S_IRWXU;
11 int fd;
12
13 void read_file(char* filename){
14     char buff2[10];
15     int bw, br;
16
17     int fd_file = open(filename, O_RDONLY);
18
19     while((br = read(fd_file, buff2, 10)) != 0){
20         bw = write(fd, buff2, br);
21     }
22 }
23
24
25 int main(int argc, char* argv[]){
26
27     int bw, br;
28     char buff[100];
29
30     int m = mkfifo("myfifo1", perm);
31     if(m == -1 && errno != EEXIST){
32         perror("mkfifo");
33         return -1;
34     }
35
36     fd = open("myfifo1", O_RDWR);
37     if(fd == -1){
38         perror("open");
39         return -1;
40     }
41
42     br = read(fd, buff, 100);
43     if(br > 0){
```

```

41     br = read(fd, buff, 100);
42     if(br > 0){
43         read_file(buff);
44         printf("Successfully Read File\n");
45     }
46
47     if(close(fd) == -1){
48         perror("close");
49         return -1;
50     }
51
52     return 0;
53 }
54 }

```

Server Code

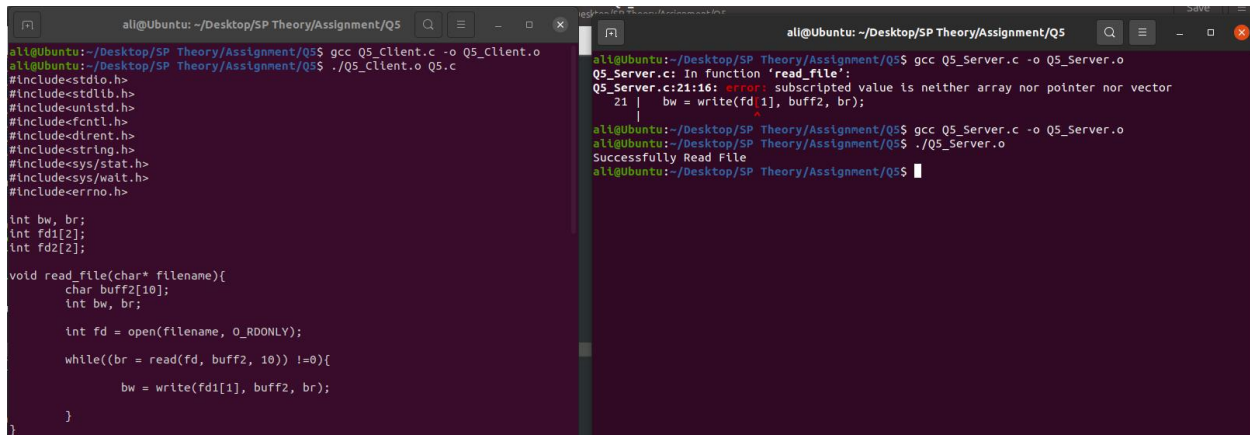
```

Q5_Server.c
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<unistd.h>
4 #include<fcntl.h>
5 #include<dirent.h>
6 #include<string.h>
7 #include<errno.h>
8 #include<sys/stat.h>
9
10 int main(int argc, char* argv[]){
11
12     char buff[100];
13     int perm = S_IRWXU;
14     int bw, br;
15     int m = mkfifo("myfifo1", perm);
16
17     if(m < 0 && errno != EEXIST){
18         perror("mkfifo");
19         return -1;
20     }
21
22     int fd = open("myfifo1", O_RDWR);
23
24     if(fd == -1){
25         perror("open");
26         return -1;
27     }
28
29     bw = write(fd, argv[1], strlen(argv[1])+1);
30     if(bw == -1){
31         perror("write");
32         return -1;
33     }
34
35     sleep(1);
36     while((br = read(fd, buff, 100)) != 0){
37         if(br == -1){
38             perror("read");
39             return -1;
40         }
41         bw = write(STDOUT_FILENO, buff, br);
42         if(bw == -1){
43             perror("write");
44             return -1;
45         }
46     }
47
48     if(close(fd) == -1){
49         perror("close");
50         return -1;
51     }
52
53     return 0;
54 }

```

Client Code

Output:



```
ali@Ubuntu: ~/Desktop/SP Theory/Assignment/Q5$ gcc Q5_Client.c -o Q5_Client.o
ali@Ubuntu: ~/Desktop/SP Theory/Assignment/Q5$ ./Q5_Client.o Q5.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;

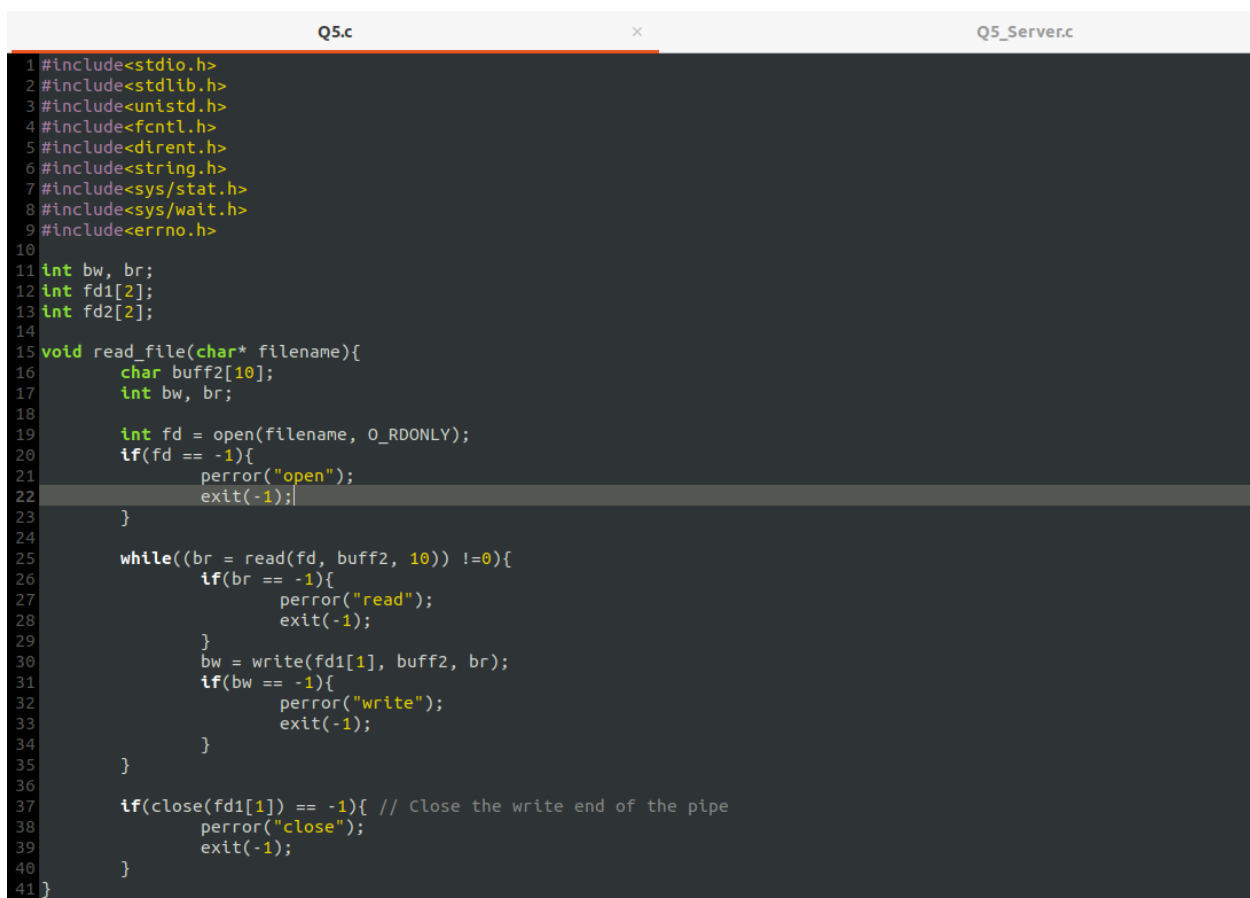
    int fd = open(filename, O_RDONLY);

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

ali@Ubuntu: ~/Desktop/SP Theory/Assignment/Q5$ gcc Q5_Server.c -o Q5_Server.o
Q5_Server.c: In function 'read_file':
Q5_Server.c:21:16: error: subscripted value is neither array nor pointer nor vector
21 |     bw = write(fd_1, buff2, br);
    |                ^
ali@Ubuntu: ~/Desktop/SP Theory/Assignment/Q5$ gcc Q5_Server.c -o Q5_Server.o
ali@Ubuntu: ~/Desktop/SP Theory/Assignment/Q5$ ./Q5_Server.o
Successfully Read File
ali@Ubuntu: ~/Desktop/SP Theory/Assignment/Q5$
```

Using Pipes

Code:



```
Q5.c
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<unistd.h>
4 #include<fcntl.h>
5 #include<dirent.h>
6 #include<string.h>
7 #include<sys/stat.h>
8 #include<sys/wait.h>
9 #include<errno.h>
10
11 int bw, br;
12 int fd1[2];
13 int fd2[2];
14
15 void read_file(char* filename){
16     char buff2[10];
17     int bw, br;
18
19     int fd = open(filename, O_RDONLY);
20     if(fd == -1){
21         perror("open");
22         exit(-1);
23     }
24
25     while((br = read(fd, buff2, 10)) !=0){
26         if(br == -1){
27             perror("read");
28             exit(-1);
29         }
30         bw = write(fd1[1], buff2, br);
31         if(bw == -1){
32             perror("write");
33             exit(-1);
34         }
35     }
36
37     if(close(fd1[1]) == -1){ // Close the write end of the pipe
38         perror("close");
39         exit(-1);
40     }
41 }

Q5_Server.c
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<unistd.h>
4 #include<fcntl.h>
5 #include<dirent.h>
6 #include<string.h>
7 #include<sys/stat.h>
8 #include<sys/wait.h>
9 #include<errno.h>
10
11 int bw, br;
12 int fd1[2];
13 int fd2[2];
14
15 void read_file(char* filename){
16     char buff2[10];
17     int bw, br;
18
19     int fd = open(filename, O_RDONLY);
20     if(fd == -1){
21         perror("open");
22         exit(-1);
23     }
24
25     while((br = read(fd, buff2, 10)) !=0){
26         if(br == -1){
27             perror("read");
28             exit(-1);
29         }
30         bw = write(fd1[1], buff2, br);
31         if(bw == -1){
32             perror("write");
33             exit(-1);
34         }
35     }
36
37     if(close(fd1[1]) == -1){ // Close the write end of the pipe
38         perror("close");
39         exit(-1);
40     }
41 }
```

```
41 }
42
43 int main(int argc, char* argv[]){
44
45     char buff[100];
46
47     int p1 = pipe(fd1);
48     if(p1 == -1){
49         perror("pipe");
50         exit(-1);
51     }
52
53     int p2 = pipe(fd2);
54     if(p2 == -1){
55         perror("pipe");
56         exit(-1);
57     }
58
59     int x = fork();
60     if(x == -1){
61         perror("fork");
62         exit(-1);
63     }
64
65     if(x == 0){
66         if(close(fd1[1]) == -1){ // Close the write end of the pipe in the child process
67             perror("close");
68             exit(-1);
69         }
70         printf("CHILD CODE\n");
71         bw = write(fd2[1], argv[1], strlen(argv[1]) + 1);
72         if(bw == -1){
73             perror("write");
74             exit(-1);
75         }
76
77         sleep(2);
78
79         while((br = read(fd1[0], buff, 100)) != 0){
80             if(br == -1){
81                 perror("read");
82                 exit(-1);
83             }
84         }
85     }
86 }
```

```

81         perror("read");
82         exit(-1);
83     }
84     bw = write(STDOUT_FILENO, buff, br);
85     if(bw == -1){
86         perror("write");
87         exit(-1);
88     }
89 }
90
91 printf("CHILD TERMINATED\n");
92 }
93 else{
94     if(close(fd2[1]) == -1){ // Close the write end of the pipe in the parent process
95         perror("close");
96         exit(-1);
97     }
98     printf("PARENT CODE\n");
99     char pipebuff[100]; // create a new buffer for pipe
100
101     br = read(fd2[0], pipebuff, 100); // read into the new buffer
102     if(br == -1){
103         perror("read");
104         exit(-1);
105     }
106
107     if(strcmp(pipebuff, argv[1]) == 0){
108         read_file(pipebuff); // buff still contains the directory name
109         //printf("INSIDE STRCMP\n");
110     }
111
112     int r = wait(NULL);
113     if(r == -1){
114         perror("wait");
115         exit(-1);
116     }
117     printf("PARENT TERMINATED\n");
118 }
119 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$ ./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:

```
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<unistd.h>
4 #include<string.h>
5 #include<sys/wait.h>
6
7 int main(int argc, char* argv[]){
8
9     int fd1[2], fd2[2];
10    int br, bw, w;
11    char buff[100];
12
13    int p1 = pipe(fd1);
14    int p2 = pipe(fd2);
15
16
17    fd_set myreadset;
18
19
20    int x = fork();
21
22
23    while(1){
```



```

23     while(1){
24         FD_ZERO(&myreadset);
25
26         FD_SET(fd1[0] , &myreadset);
27         FD_SET(fd2[0] , &myreadset);
28         FD_SET(STDIN_FILENO , &myreadset);
29
30         int maxfd;
31         if(fd1 > fd2 && fd1 > STDIN_FILENO)
32             maxfd = fd1[0];
33
34         else if(fd2 > fd1 && fd2 > STDIN_FILENO)
35             maxfd = fd2[0];
36
37         else
38             maxfd = STDIN_FILENO;
39
40
41         if(x == 0){
42

```

```

41         if(x == 0){
42
43
44
45             int nrf = select(maxfd + 1, &myreadset, NULL, NULL, NULL);
46
47             if(FD_ISSET(STDIN_FILENO, &myreadset)){
48
49                 br = read(STDIN_FILENO, buff, 100);
50                 bw = write(fd1[1], buff, br);
51             }
52
53             if(FD_ISSET(fd2[0], &myreadset)){
54
55                 br = read(fd2[0], buff, 100);
56                 if(br > 0)
57                     printf("Child Read Succesfully: \n");
58
59                 write(STDOUT_FILENO, buff, br);
60             }
61         }
62
63         else{
64
65             int nrf = select(maxfd + 1, &myreadset, NULL, NULL, NULL);
66
67             if(FD_ISSET(fd1[0], &myreadset)){
68
69                 br = read(fd1[0], buff, 100);
70                 if(br > 0)
71                     printf("Parent Read Succesfully: \n");
72
73                 write(STDOUT_FILENO, buff, br);
74             }
75
76             if(FD_ISSET(STDIN_FILENO, &myreadset)){
77
78                 br = read(STDIN_FILENO, buff, 100);
79                 write(fd2[1], buff, br);
80             }
81         }
82     }
83     return 0;

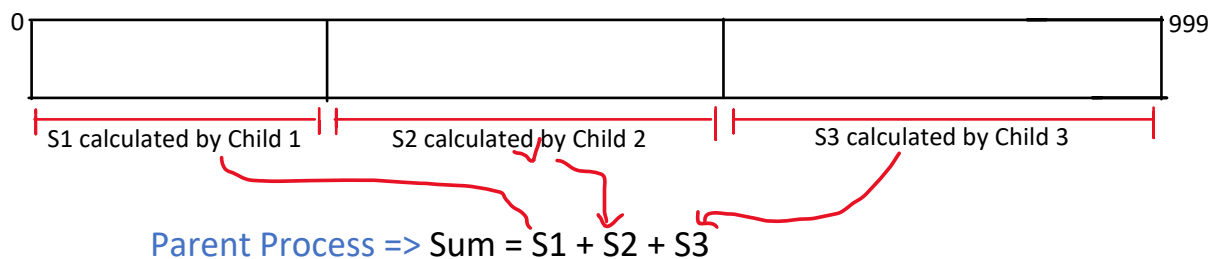
```

Output:

```
ali@Ubuntu: ~/Desktop/SP Theory/Assignment
ali@Ubuntu:~/Desktop/SP Theory/Assignment$ ./Q6.o
Hi
Hello
Child Read Successfully:
Hi
Parent Read Successfully:
Hello
How are you?
Parent Read Successfully:
How are you?
I am good
Child Read Successfully:
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
1 #include<stdio.h>
2 #include<sys/wait.h>
3 #include<unistd.h>
4
5 int main(int argc, char *argv[]){
6     int arr[1000];
7     int status, pid;
8     int res = 0;
9     int sum = 0;
10    int total = 0;
11    int i, fd[2];
12
13    for(i = 0; i<1000; i++){
14        arr[i] = i;
15    }
16
17    int p = pipe(fd);
18
19    for(i = 0; i<3; i++){
20        pid = fork();
21        if(pid < 0){
22            perror("fork");
23            return 1;
24        }
25
26        if(pid == 0){ break; }
27    }
28
29    if(pid == 0 && i == 0){ //First Child's Code
30        for(int j = 0; j<=1000/3 *(1 + i); j++){
31            res += arr[j];
32        }
33        if(write(fd[1], &res, sizeof(sum)) < 0){
34            perror("write");
35            return 1;
36        }
37        return 0;
38    }
39
40    if(pid == 0 && i == 1){ //Second Child's Code
41        for(int j = 334; j<=1000/3 *(1 + i); j++){
42
```

```

Q7.c      Q5.c      Q5_Server.c
40
41     if(pid == 0 && i == 1){//Second Child's Code
42         for(int j = 334; j<=1000/3 *(1 + i); j++){
43             res += arr[j];
44         }
45
46         if(write(fd[1], &res, sizeof(sum)) < 0){
47             perror("write");
48             return 1;
49         }
50         return 0;
51     }
52
53     if(pid == 0 && i == 2){//Third Child's Code
54         for(int j = 667; j<=1000/3 *(1 + i); j++){
55             res += arr[j];
56         }
57
58         if(write(fd[1], &res, sizeof(sum)) < 0){
59             perror("write");
60             return 1;
61         }
62
63         return 0;
64     }
65
66     if(pid > 0){
67         for(i = 0; i<3; i++){
68             int r = wait(&status);
69             if(r < 0){
70                 perror("wait");
71                 return 1;
72             }
73             if(WIFEXITED(status)){
74                 printf("Child %d terminated with return status %d\n", i, WEXITSTATUS(status));
75
76                 if(read(fd[0], &sum, sizeof(sum)) < 0){
77                     perror("read");
78                     return 1;
79                 }
80                 total += sum;
81             }
82         }
83
84         printf("Sum = %d, ", total);
85     }
86     return 0;
87 }
88
79     }
80     total += sum;
81 }
82 }
83
84     printf("Sum = %d, ", total);
85 }
86 return 0;
87 }
88

```

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:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <signal.h>
5 #include <sys/wait.h>
6
7 int main(int argc, char *argv[]) {
8     if (argc != 2) {
9         fprintf(stderr, "Usage: %s <N>\n", argv[0]);
10        exit(EXIT_FAILURE);
11    }
12
13    int N = atoi(argv[1]);
14    sigset_t set;
15    int sig;
16
17    /* Initialize the signal set */
18    sigemptyset(&set);
19
20    /* Add SIGUSR1 to the set */
21    sigaddset(&set, SIGUSR1);
22
23    /* Block SIGUSR1 */
24    if (sigprocmask(SIG_BLOCK, &set, NULL) == -1) {
25        perror("sigprocmask");
26        exit(-1);
27    }
28
29    pid_t pid = fork();
30    if (pid == -1) {
31        perror("fork");
32        exit(EXIT_FAILURE);
33    }
34
35    if (pid == 0) { // Child process
36        for (int i = 0; i < N; i++) {
37            kill(getppid(), SIGUSR1);
38        }
39        exit(0);
40    }
41
42    else { // Parent process
```

```

42     else { // Parent process
43
44         for (int i = 0; i < N; i++) {
45             sigwait(&set, &sig); // Wait for SIGUSR1
46             //count++;
47         }
48         printf("Received %d SIGUSR1 signals\n", N);
49         wait(NULL); // Wait for child process to finish
50     }
51
52     return 0;
53 }
54

```

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
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<unistd.h>
4 #include<signal.h>
5
6 sigset_t myset;
7 int x;
8
9 void my_wait(){
10
11     if(x > 0){
12         sigsuspend(&myset);
13     }
14 }
15
16 void my_handler(int sig_no){
17     printf("\nChild has been terminated..\n");
18 }
19
20 int main(int argc, char* argv){
21
22     struct sigaction my_action;
23
24     my_action.sa_flags = 0;
25     my_action.sa_handler = my_handler;
26     sigemptyset(&my_action.sa_mask);
27

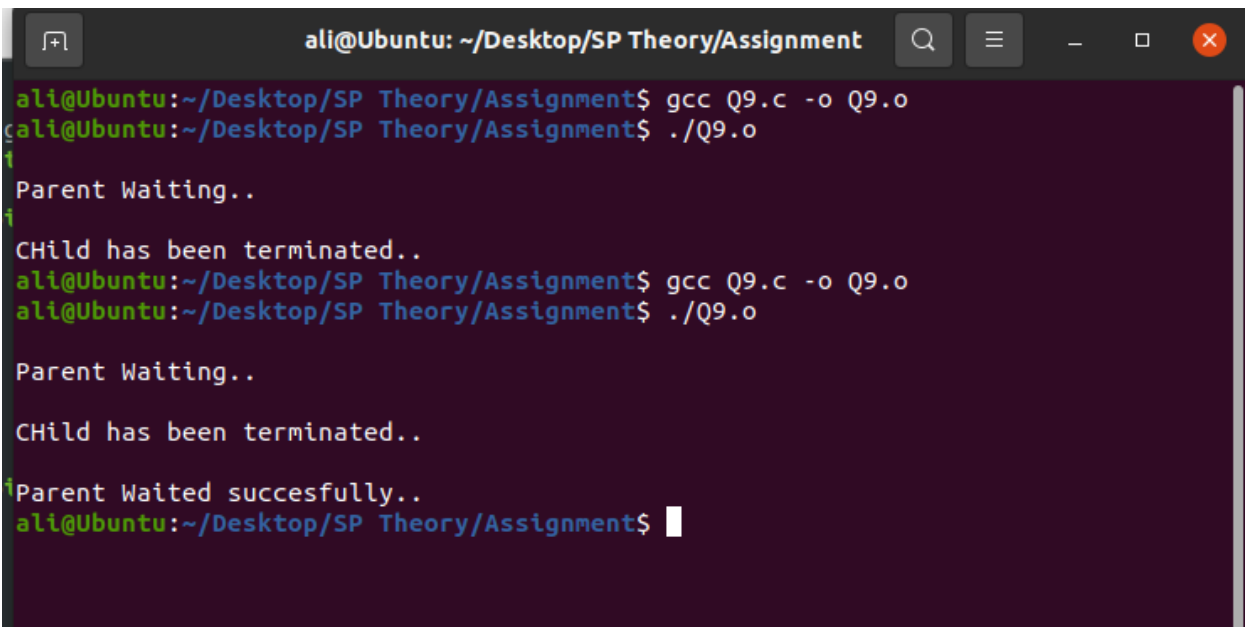
```

```

28     sigemptyset(&my_action.sa_mask);
29
30     sigemptyset(&myset);
31     sigfillset(&myset);
32     sigdelset(&myset, SIGCHLD);
33
34     if(sigaction(SIGCHLD, &my_action, NULL) < 0){
35         perror("sigaction");
36         exit(-1);
37     }
38
39     x = fork();
40
41     if(x < 0){
42         perror("fork");
43         exit(-1);
44     }
45     else if(x > 0){
46         printf("\nParent Waiting..\n");
47         my_wait();
48         printf("\nParent Waited successfully..\n");
49     }
50
51     return 0;
52 }
53

```

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
1
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 successfully..
ali@Ubuntu:~/Desktop/SP Theory/Assignment$

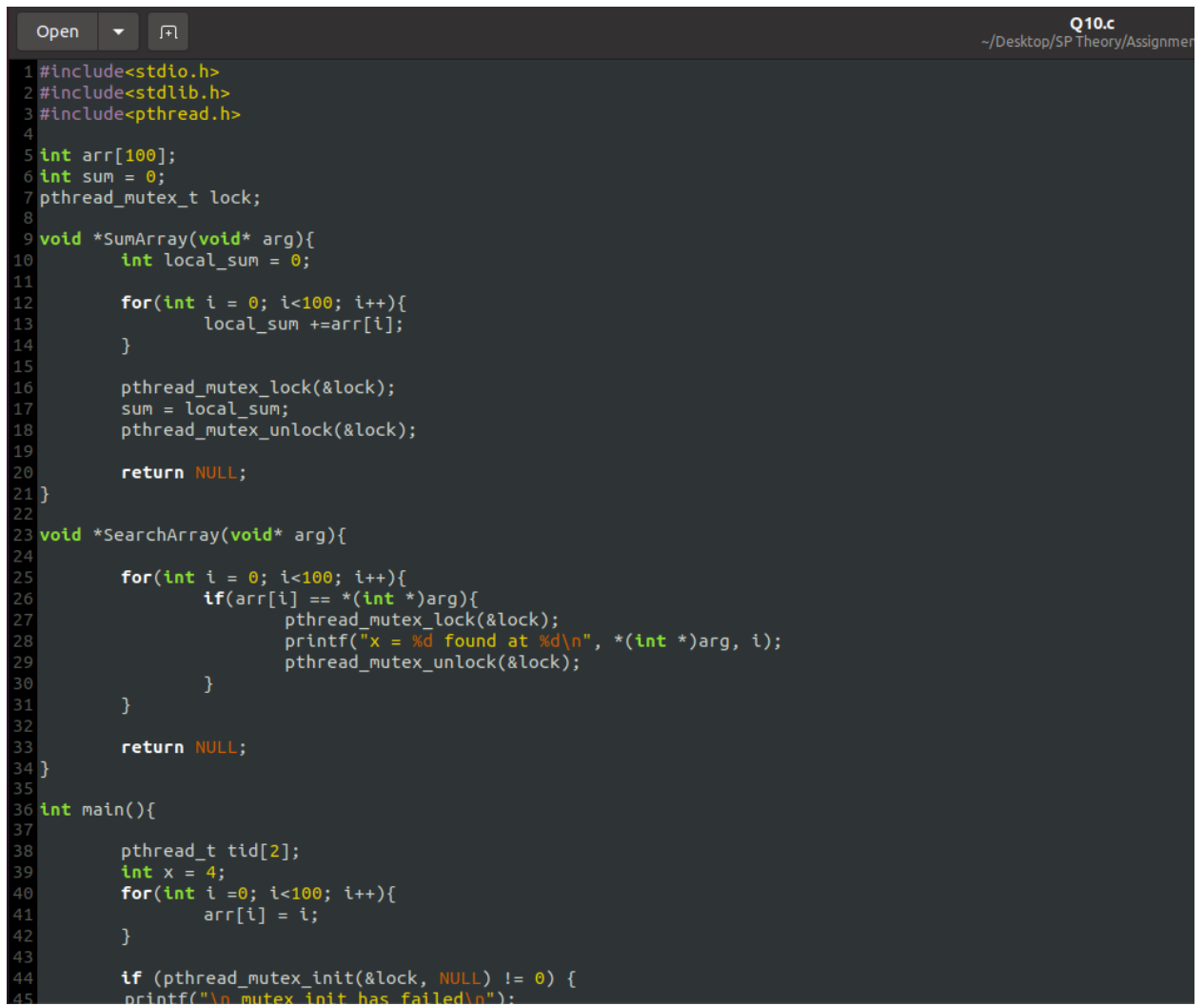
```

Q10: Write a program that creates 2 threads.

Thread 1: Find sum of array elements.

Thread 2: Searches for a key in array.

Code:



```
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<pthread.h>
4
5 int arr[100];
6 int sum = 0;
7 pthread_mutex_t lock;
8
9 void *SumArray(void* arg){
10     int local_sum = 0;
11
12     for(int i = 0; i<100; i++){
13         local_sum +=arr[i];
14     }
15
16     pthread_mutex_lock(&lock);
17     sum = local_sum;
18     pthread_mutex_unlock(&lock);
19
20     return NULL;
21 }
22
23 void *SearchArray(void* arg){
24     for(int i = 0; i<100; i++){
25         if(arr[i] == *(int *)arg){
26             pthread_mutex_lock(&lock);
27             printf("x = %d found at %d\n", *(int *)arg, i);
28             pthread_mutex_unlock(&lock);
29         }
30     }
31
32     return NULL;
33 }
34
35
36 int main(){
37     pthread_t tid[2];
38     int x = 4;
39     for(int i=0; i<100; i++){
40         arr[i] = i;
41     }
42
43     if (pthread_mutex_init(&lock, NULL) != 0) {
44         printf("\n mutex init has failed\n");
45     }
```



```

43
44     if (pthread_mutex_init(&lock, NULL) != 0) {
45         printf("\n mutex init has failed\n");
46         return 1;
47     }
48
49     if(pthread_create(&tid[0], NULL, SumArray, NULL) != 0){
50         printf("\nThread creation failed. Exiting now.");
51         return 1;
52     }
53
54     if(pthread_create(&tid[1], NULL, SearchArray, (void *)&x) != 0){
55         printf("\nThread creation failed. Exiting now.");
56         return 1;
57     }
58
59
60     for(int i =0; i<2; i++){
61         if(pthread_join(tid[i], NULL) != 0){
62             printf("\nThread join failed. Exiting now.");
63             return 1;
64         }
65     }
66
67     printf("Sum = %d\n", sum);
68     pthread_mutex_destroy(&lock);
69     return 0;
70 }

```

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
3 #include<pthread.h>
4 #include<unistd.h>
5 #include<fcntl.h>
6
7 int fd_rd[2];
8 int fd_wr[2];
9
10 char *rd_names[2] = {"f1.txt", "f2.txt"};
11 char *wr_names[2] = {"f1_Copy.txt", "f2_Copy.txt"};
12
13 void *COPY(void* arg){
14     int br, bw;
15     char buff[100];
16
17     while((br = read(fd_rd[(int *)arg], buff, 100)) !=0){
18         bw = write(fd_wr[(int *)arg], buff, br);
19     }
20 }
21
22 int main(){
23     pthread_t tid[2];
24     int perm = S_IRWXU;
25     int args[2] = {0, 1};
26
27     for(int i=0; i<2; i++){
28         fd_rd[i] = open(rd_names[i], O_RDONLY);
29         fd_wr[i] = open(wr_names[i], O_WRONLY | O_CREAT, perm);
30     }
31
32     for(int i=0; i<2; i++){
33         pthread_create(&tid[i], NULL, COPY, (void *)&args[i]);
34     }
35
36     for(int i=0; i<2; i++){
37         pthread_join(tid[i], NULL);
38     }
39
40     printf("Both Files Copied Successfully\n");
41
42     return 0;
43 }
44 }
```

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$
```

