OS_Lab8

K. Ram Mohan

COE19B055

Q1)

```
#include<stdio.h>

#include<stdlib.h>

#include<limits.h>

#define SIZE 100


void swap(int *p,int *q)

{

        int temp;

        temp=*p;

        *p=*q;

        *q=temp;

}


//function to sort based on arrival time

void sort_arr(int id[], int arr[], int burst_time[] , int size)

{

        int i, j;

        int go=1;

        while(go)

        {

                for(i=0; i<size-1; i++){

                        go=0;

                        for(j=0; j<size-1-i; j++){

                                if(arr[j]>arr[j+1]){

                                        swap(&arr[j], &arr[j+1]);
```

```
                    swap(&id[j], &id[j+1]);
                    swap(&burst_time[j], &burst_time[j+1]);
                    go=1;
                }
            }
        }
    }
}

int main()
{
    int total_process, i, j;
    int process_id[SIZE], arr_time[SIZE], burst_time[SIZE], completed[SIZE], c_time=0, index;
    float tot_waiting_time=0, tot_trt=0, response_ratio, temp;

    printf("Enter no of process: ");
    scanf("%d", &total_process);
    for(i=0; i<total_process; i++)
    {
        printf("Process_id of process %d: ", i+1);
        scanf("%d", &process_id[i]);
        printf("Arrival time of process %d: ", i+1);
        scanf("%d", &arr_time[i]);
        printf("Burst time of process %d: ", i+1);
        scanf("%d", &burst_time[i]);
        completed[i]=0;
    }

    sort_arr(process_id, arr_time, burst_time, total_process);
    int completion_time[SIZE], waiting_time[SIZE], turn_arnd_time[SIZE];
```

```
for(i=0; i<total_process; i++)
{
        response_ratio = -999;
        c_time= ((i==0 | | c_time<arr_time[i]) ? arr_time[i] : c_time);
        printf("%d-ctime\n", c_time);

        for(j=0; j<total_process; j++)
        {
                if(c_time>=arr_time[j] && completed[j]!=1)
                {
                        temp = (float)((c_time-
                arr_time[j])+burst_time[j])/(float)(burst_time[j]);

                        if(temp>response_ratio)
                        {
                                response_ratio = temp;
                                index=j;
                        }
                }
        }

        c_time=c_time+burst_time[index];
        completed[index]=1;
        completion_time[index]=c_time;
}

for(i=0; i<total_process; i++)
{
        turn_arnd_time[i] = completion_time[i] - arr_time[i];
        tot_trt = tot_trt + turn_arnd_time[i];
}
```

```
        for(i=0; i<total_process; i++)
        {
                waiting_time[i] = turn_arnd_time[i] - burst_time[i];
                tot_waiting_time = tot_waiting_time + waiting_time[i];
        }
        printf("Processes  arrival_time   Burst time  completion time   Turn around time   Waiting
        time\n");
        for(i=0; i<total_process; i++)
        {
                printf("%d", process_id[i]);
                printf("\t\t%d", arr_time[i]);
                printf("\t\t%d", burst_time[i]);
                printf("\t\t%d", completion_time[i]);
                printf("\t\t%d", turn_arnd_time[i]);
                printf("\t\t%d \n", waiting_time[i]);
        }

        float avg_wait = tot_waiting_time/total_process;
        float avg_trt = tot_trt/total_process;
        printf("Average waiting time is : %f\n", avg_wait);
        printf("Average turn around time is: %f\n", avg_trt);
}
```

HRRN

COE18B055
K.Ram Mohan

Q1)

I/P:

PID	AT	BT
1	0	2
2	2	6
3	4	4
4	6	5
5	8	2

O/P:

PID	AT	BT	CT	TAT	WT
1	0	2	2	2	0
2	2	6	8	6	0
3	4	4	12	8	4
4	6	5	19	13	8
5	8	2	14	6	4

GANTT CHART:

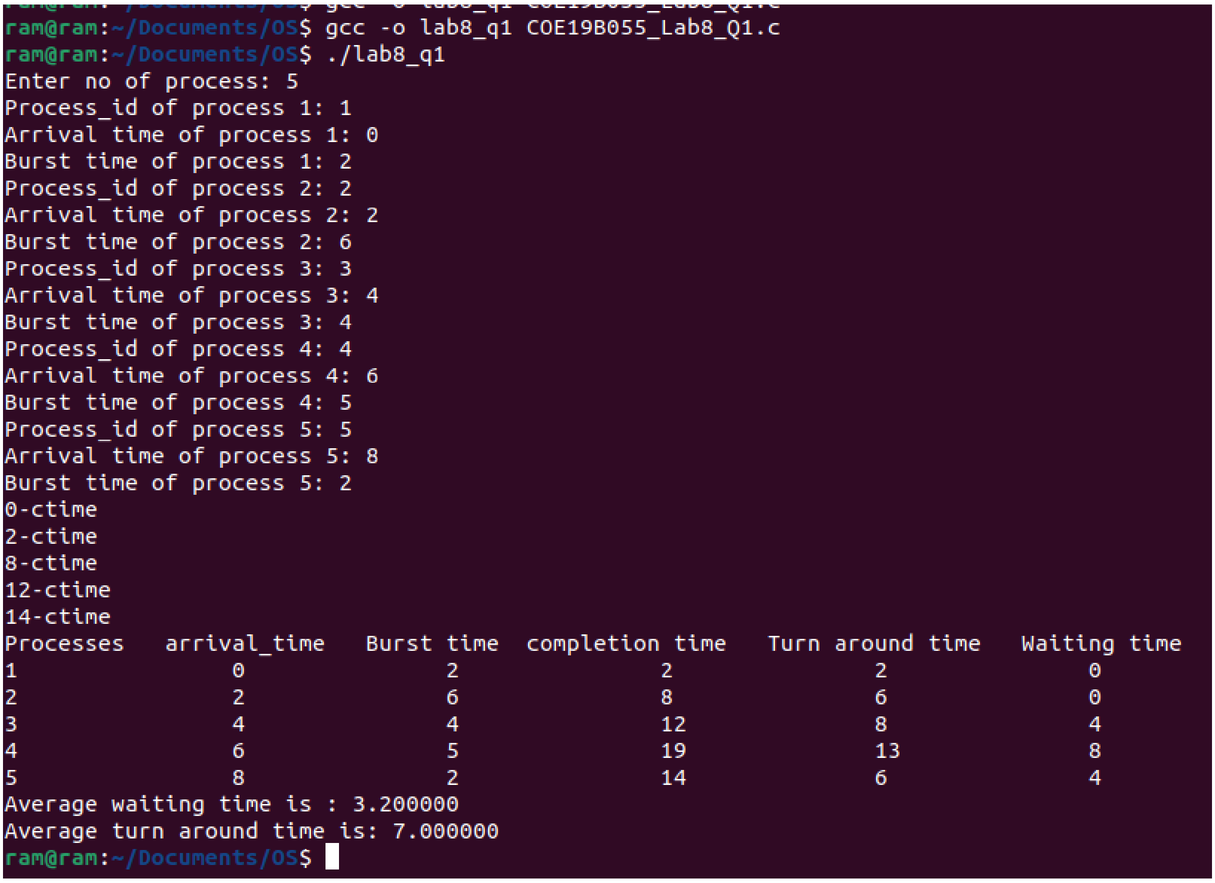
P1	P2	P3	P5	P4

0 2 8 12 14 19

Average waiting time = 3.2

Average turn around time = 7.

Output:

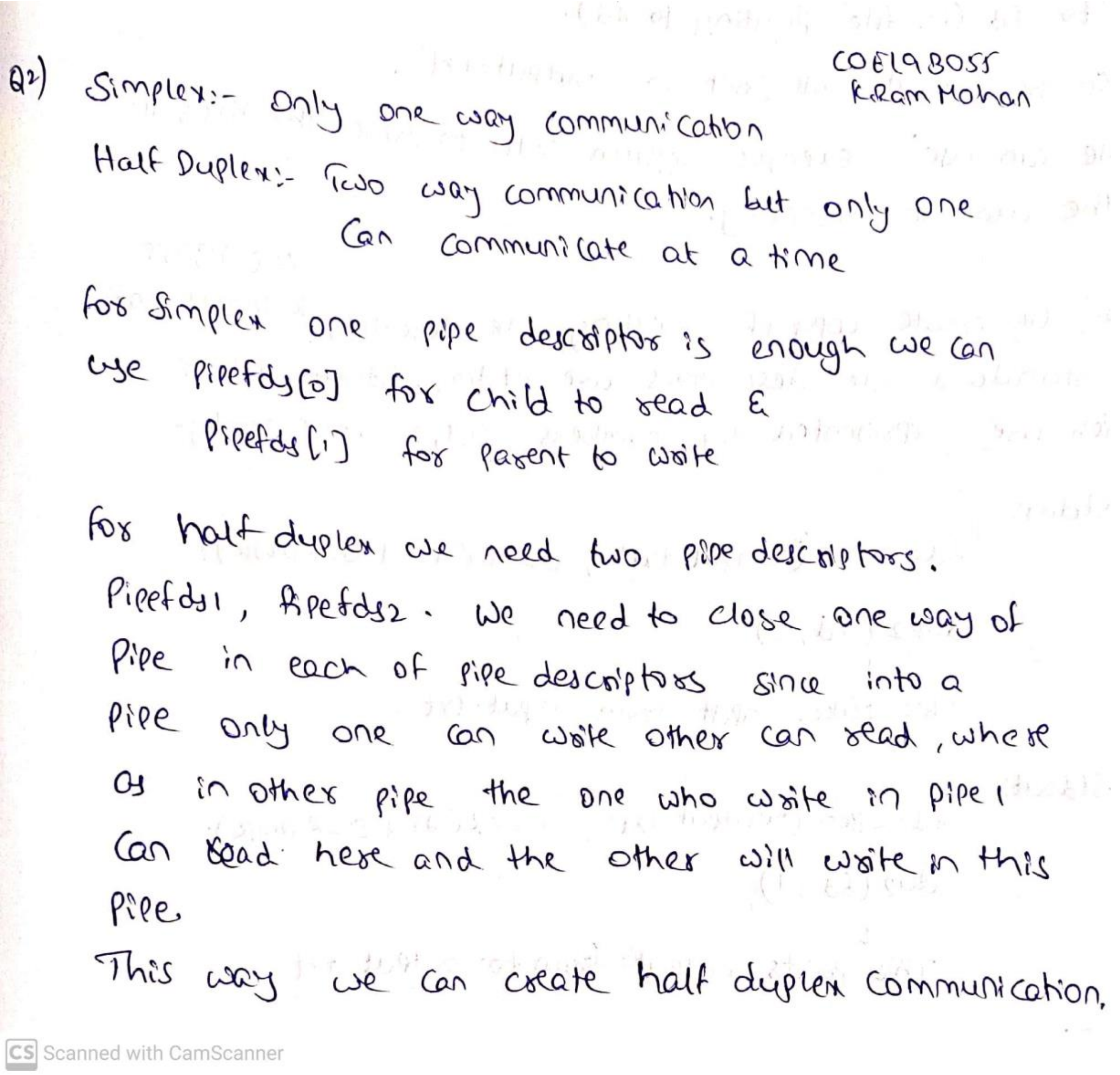


```
ram@ram:~/Documents/OS$ gcc -o lab8_q1 COE19B055_Lab8_Q1.c
ram@ram:~/Documents/OS$ ./lab8_q1
Enter no of process: 5
Process_id of process 1: 1
Arrival time of process 1: 0
Burst time of process 1: 2
Process_id of process 2: 2
Arrival time of process 2: 2
Burst time of process 2: 6
Process_id of process 3: 3
Arrival time of process 3: 4
Burst time of process 3: 4
Process_id of process 4: 4
Arrival time of process 4: 6
Burst time of process 4: 5
Process_id of process 5: 5
Arrival time of process 5: 8
Burst time of process 5: 2
0-ctime
2-ctime
8-ctime
12-ctime
14-ctime
Processes   arrival_time   Burst time   completion time   Turn around time   Waiting time
1                0               2               2               2               0
2                2               6               8               6               0
3                4               4               12              8               4
4                6               5               19              13              8
5                8               2               14              6               4
Average waiting time is : 3.200000
Average turn around time is: 7.000000
ram@ram:~/Documents/OS$ 
```

Q2a)Simplex:

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

int main()
{
        int pipefds[2], returnstatus, pid;
        char writemessage[100], readmessage[100];

        returnstatus = pipe(pipefds);
        if(returnstatus < 0)
        {
                printf("Failed to create pipe\n");
                return 0;
        }

        pid = fork();
        //printf("This is a simplex communication where parent process can write and child process can only read what parent has written\n");
        if(pid==0)
        {
                read(pipefds[0], readmessage, sizeof(readmessage));
                printf("Child process - Reading from pipe - Message is %s\n", readmessage);
        }
```

```
	else
	{
		printf("Paren process-Enter message: ");
		fgets(writemessage, 100, stdin);

		printf("Parent process - Writing to pipe - Message is %s\n", writemessage);
		write(pipefds[1], writemessage, sizeof(writemessage));
	}
}
```



COE19B055
K.Ram Mohan

Q2) Simplex:- Only one way communication

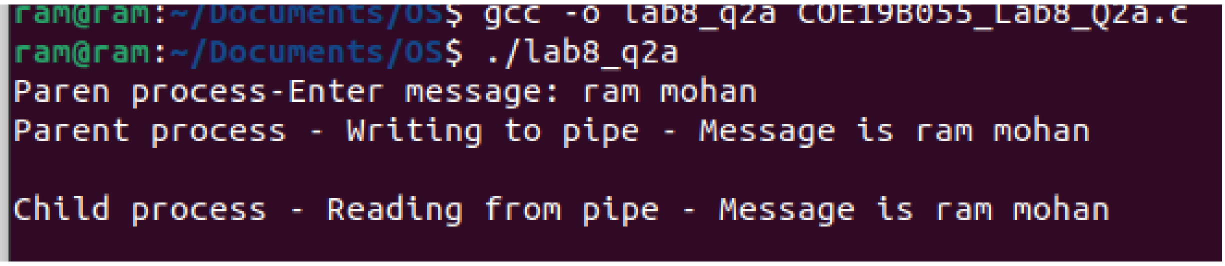
Half Duplex:- Two way communication but only one Can communicate at a time

for Simplex one pipe descriptor is enough we can use Pipefds[0] for Child to read &
Pipefds[1] for Parent to write

for half duplex we need two pipe descriptors. Pipefds1, Pipefds2. We need to close one way of Pipe in each of pipe descriptors since into a Pipe only one can write other can read, where as in other pipe the one who write in pipe 1 Can Read here and the other will write in this Pipe.

This way we can create half duplex communication.

Output:



```
ram@ram:~/Documents/OS$ gcc -o lab8_q2a COE19B055_Lab8_Q2a.c
ram@ram:~/Documents/OS$ ./lab8_q2a
Paren process-Enter message: ram mohan
Parent process - Writing to pipe - Message is ram mohan

Child process - Reading from pipe - Message is ram mohan
```

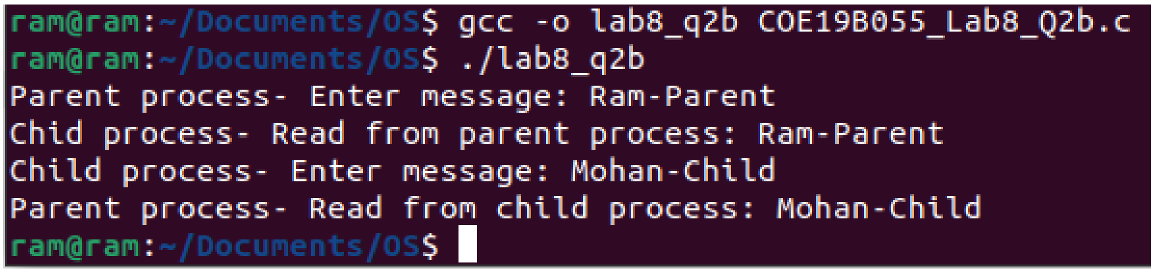
Q2b)Half Duplex:

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/wait.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<string.h>
int main()
{
        int pipefds1[2],pipefds2[2];
        int returnstatus1, returnstatus2;
        int pid;
        char writemessage[100], readmessage[100];

        returnstatus1 = pipe(pipefds1);
        if(returnstatus1 < 0)
        {
                printf("Failed to create pipe1\n");
                return 0;
        }
```

```
returnstatus2 = pipe(pipefds2);
if(returnstatus2 < 0)
{
        printf("Failed to create pipe2\n");
        return 0;
}
pid = fork();
if(pid == 0)
{
        close(pipefds1[1]);
        close(pipefds2[0]);

        read(pipefds1[0], readmessage, sizeof(readmessage));
        printf("Chid process- Read from parent process: %s", readmessage);

        printf("Child process- Enter message: ");
        fgets(writemessage, 100, stdin);
        write(pipefds2[1], writemessage, sizeof(writemessage));
}
Else
{
        close(pipefds1[0]);
        close(pipefds2[1]);

        printf("Parent process- Enter message: ");
        fgets(writemessage, 100, stdin);
        write(pipefds1[1], writemessage, sizeof(writemessage));
```

```
                read(pipefds2[0], readmessage, sizeof(readmessage));

                printf("Parent process- Read from child process: %s", readmessage);

        }

}
```

Output:

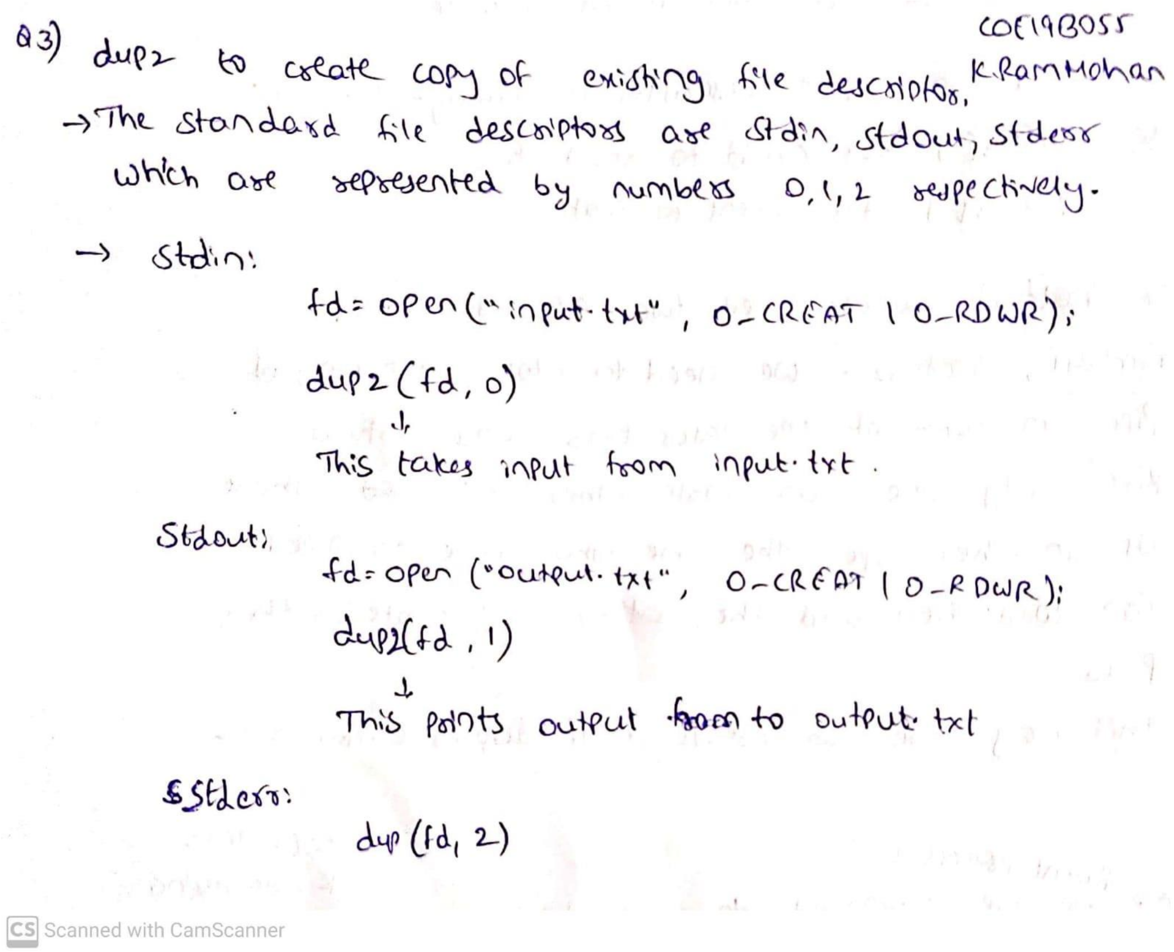


```
ram@ram:~/Documents/OS$ gcc -o lab8_q2b COE19B055_Lab8_Q2b.c
ram@ram:~/Documents/OS$ ./lab8_q2b
Parent process- Enter message: Ram-Parent
Chid process- Read from parent process: Ram-Parent
Child process- Enter message: Mohan-Child
Parent process- Read from child process: Mohan-Child
ram@ram:~/Documents/OS$ 
```

Q3)

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/wait.h>
#include<string.h>
#include<fcntl.h>
int main()
{
	int pipefds[2], returnstatus, pid, fd, fd1, fd2;
	char writemessage[100], readmessage[100];

	returnstatus = pipe(pipefds);
	if(returnstatus < 0)
	{
		printf("Failed to create pipe\n");
		return 0;
	}

	pid = fork();
	if(pid==0)
	{
		fd = open("output1.txt",  O_RDWR | O_CREAT, 0666);
		fd1 = open("error.txt", O_RDWR | O_CREAT, 0666);
		printf("We will print output  in output1.txt\n");
		read(pipefds[0], readmessage, sizeof(readmessage));
```

```
        dup2(fd, STDOUT_FILENO);
        printf("Child process - Reading from pipe - Message is %s", readmessage);
        dup2(fd1, 2);
        //printf("Stderr\n");
        execlp("la", "ls", NULL);
    }
    else
    {
        //printf("Paren process-Enter message: ");
        fd = open("input.txt",  O_RDONLY);
        if(dup2(fd, 0)<0)
        {
            printf("Unable to duplicate file descriptor.");
        }

        fgets(writemessage, 100, stdin);
        printf("Parent process - Writing to pipe(Read input from input.txt) - Message is %s", writemessage);
        write(pipefds[1], writemessage, sizeof(writemessage));
    }
}
```



Q3) dup2 to create copy of existing file descriptor. COE19B055 K.RamMohan

→The standard file descriptors are stdin, stdout, stderr which are represented by numbers 0,1,2 respectively.

→ stdin:

```
fd= open("input.txt", O_CREAT | O_RDWR);
dup2(fd, 0)
   ↓
This takes input from input.txt.
```

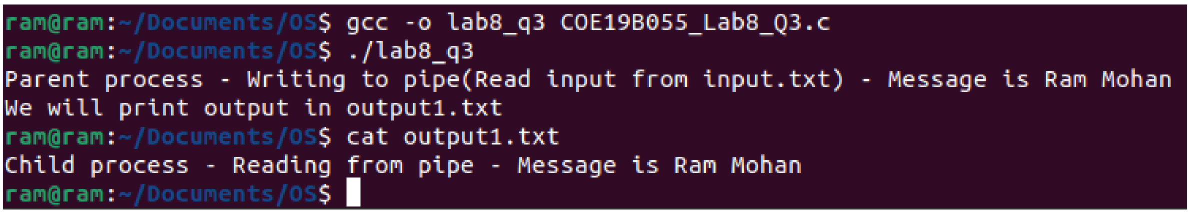
Stdout:

```
fd= open ("output.txt", O_CREAT | O_RDWR);
dup2(fd, 1)
   ↓
This points output ~~from~~ to output.txt
```

~~S~~Stderr:

```
dup (fd, 2)
```

Output:

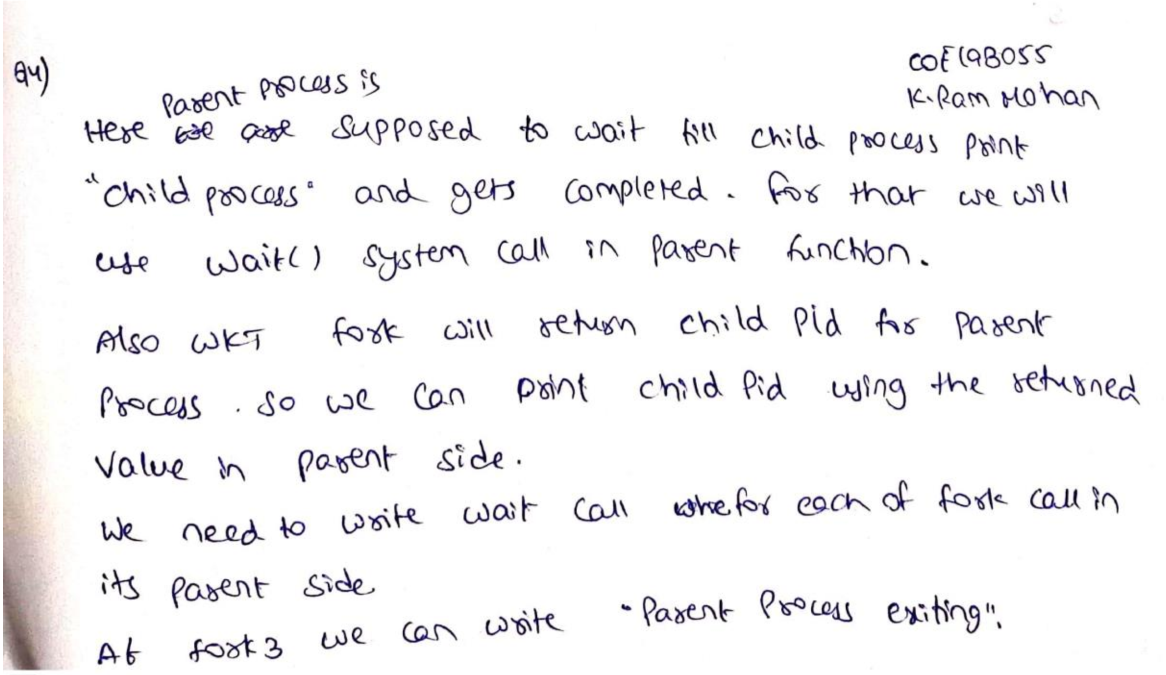


```
ram@ram:~/Documents/OS$ gcc -o lab8_q3 COE19B055_Lab8_Q3.c
ram@ram:~/Documents/OS$ ./lab8_q3
Parent process - Writing to pipe(Read input from input.txt) - Message is Ram Mohan
We will print output in output1.txt
ram@ram:~/Documents/OS$ cat output1.txt
Child process - Reading from pipe - Message is Ram Mohan
ram@ram:~/Documents/OS$ 
```

Q4)

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/wait.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<time.h>

int main()
{
        int pid1, pid2, pid3;
        srand(time(0));

        pid1 = fork();
        if(pid1 == 0)
        {
                printf("Child Process\n");
                sleep(rand()%10);
        }
        else
        {
                wait(NULL);
                printf("Child process with ID=%d terminated\n", pid1);

                pid2=fork();
                if(pid2 == 0)
                {
```

```
            printf("Child process\n");
            sleep(rand()%10);
        }
        Else
        {
            wait(NULL);
            printf("Child process with ID=%d terminated\n", pid2);

            pid3 = fork();
            if(pid3==0)
            {
                printf("Child process\n");
                sleep(rand()%10);
            }
            Else
            {
                wait(NULL);
                printf("Child process with ID=%d terminated\n", pid3);
                printf("Parent process exiting\n");
            }
        }
    }
}
```



Q4)

COE19B055
K.Ram Mohan

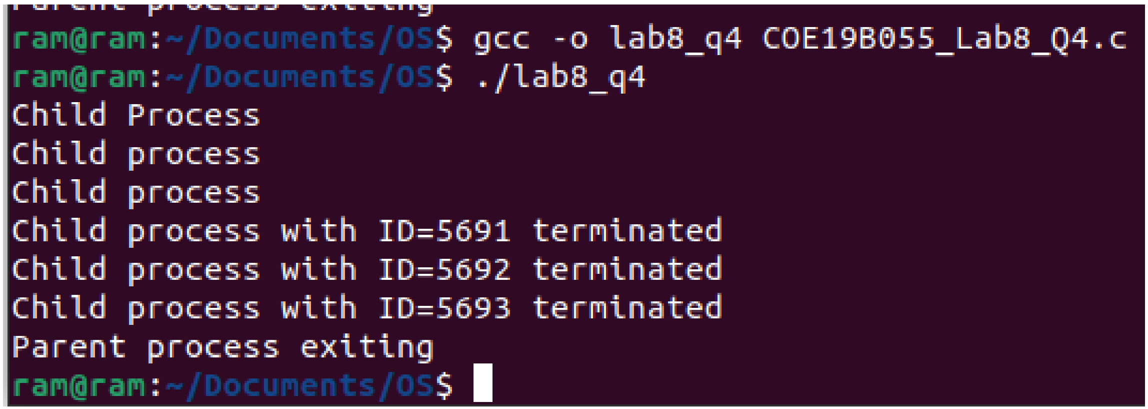
Here ~~we~~ ~~are~~ Parent process is supposed to wait till child process print "child process" and gets completed. for that we will use wait() system call in parent function.

Also WKT fork will return child pid for parent process. so we can print child pid using the returned value in parent side.

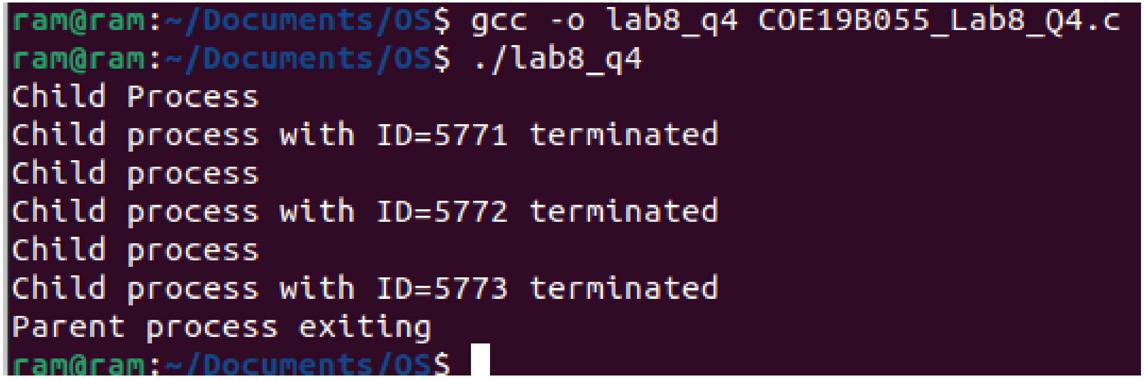
We need to write wait call ~~where~~ for each of fork call in its parent side.

At fork3 we can write "Parent process exiting".

Output:



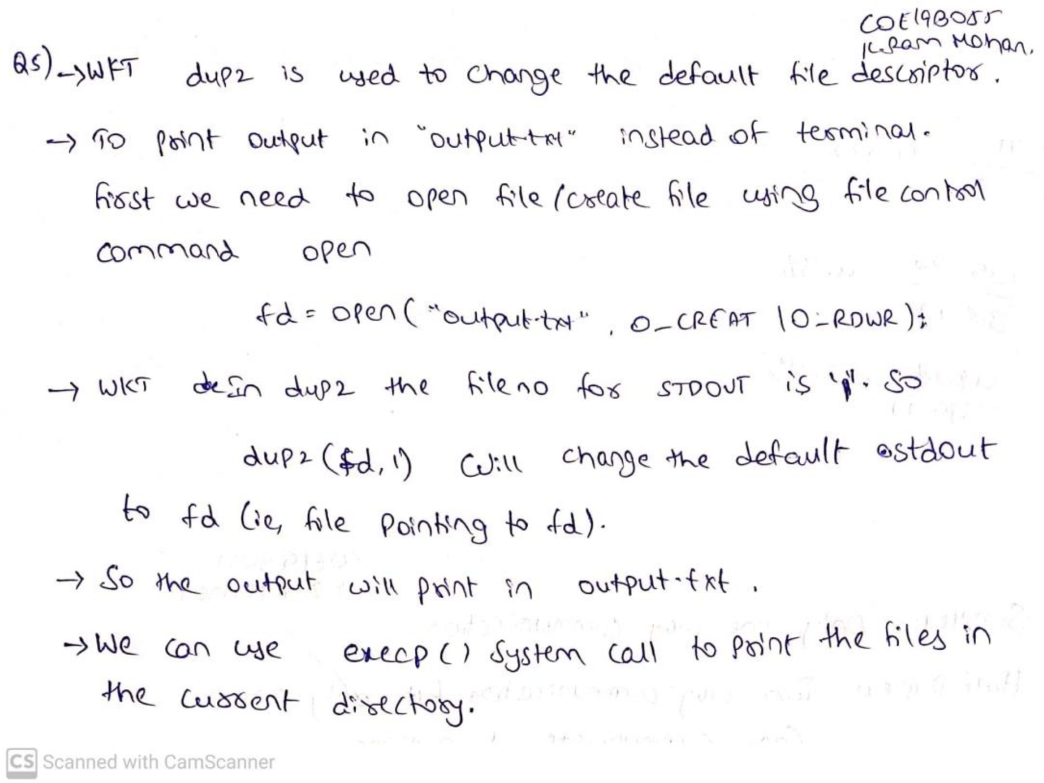
```
Parent process exiting
ram@ram:~/Documents/OS$ gcc -o lab8_q4 COE19B055_Lab8_Q4.c
ram@ram:~/Documents/OS$ ./lab8_q4
Child Process
Child process
Child process
Child process with ID=5691 terminated
Child process with ID=5692 terminated
Child process with ID=5693 terminated
Parent process exiting
ram@ram:~/Documents/OS$
```



```
ram@ram:~/Documents/OS$ gcc -o lab8_q4 COE19B055_Lab8_Q4.c
ram@ram:~/Documents/OS$ ./lab8_q4
Child Process
Child process with ID=5771 terminated
Child process
Child process with ID=5772 terminated
Child process
Child process with ID=5773 terminated
Parent process exiting
ram@ram:~/Documents/OS$
```

Q5)

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

int main()
{
        int pid, fd;

        fd = open("output.txt",  O_CREAT | O_RDWR);
        if(fd<0)
        {
                printf("File unable to create or open\n");
        }
        else
        {
                dup2(fd, 1);
                execlp("ls", "ls", NULL);}
}
```



COE19B055
K.Ram Mohan.

Q5)→WKT dup2 is used to change the default file descriptor.

→ To print output in "output.txt" instead of terminal. first we need to open file/create file using file control command open

fd = open("output.txt", O_CREAT |O_RDWR);

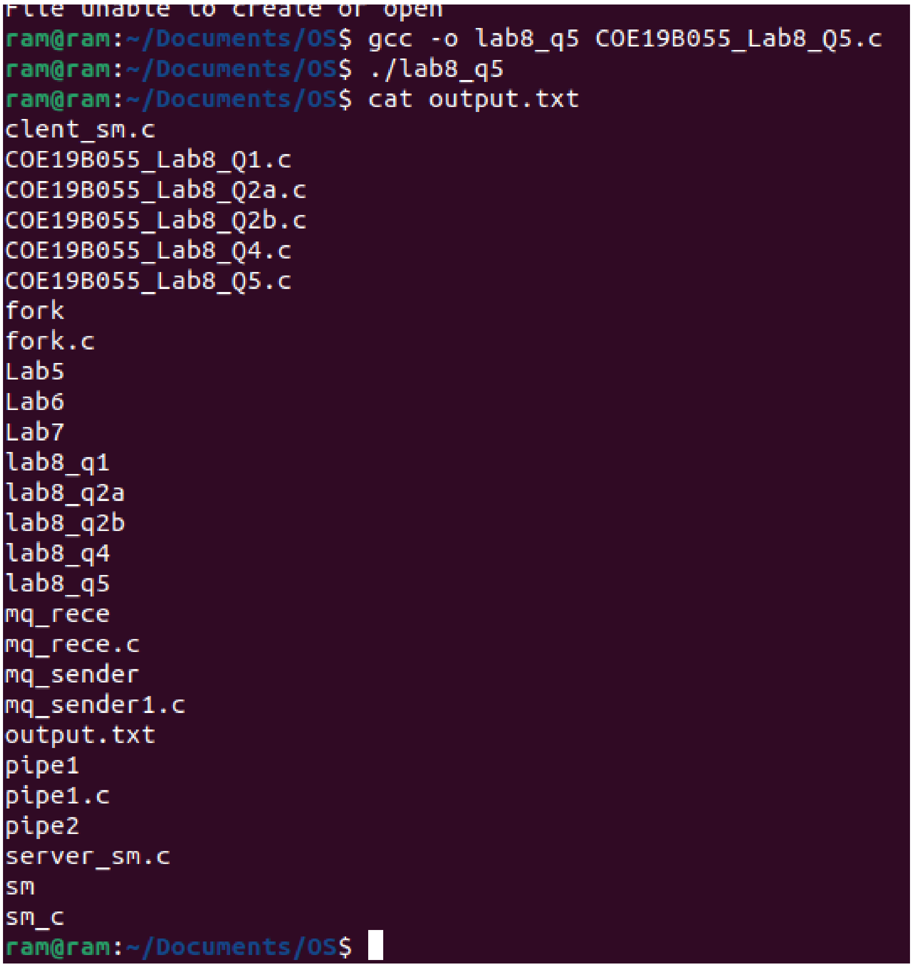
→ WKT ~~de~~ in dup2 the fileno for STDOUT is '1'. So

dup2(fd,1) will change the default stdout to fd (ie, file pointing to fd).

→ So the output will print in output.txt.

→We can use execp() system call to print the files in the current directory.

Output:



```
File unable to create or open
ram@ram:~/Documents/OS$ gcc -o lab8_q5 COE19B055_Lab8_Q5.c
ram@ram:~/Documents/OS$ ./lab8_q5
ram@ram:~/Documents/OS$ cat output.txt
clent_sm.c
COE19B055_Lab8_Q1.c
COE19B055_Lab8_Q2a.c
COE19B055_Lab8_Q2b.c
COE19B055_Lab8_Q4.c
COE19B055_Lab8_Q5.c
fork
fork.c
Lab5
Lab6
Lab7
lab8_q1
lab8_q2a
lab8_q2b
lab8_q4
lab8_q5
mq_rece
mq_rece.c
mq_sender
mq_sender1.c
output.txt
pipe1
pipe1.c
pipe2
server_sm.c
sm
sm_c
ram@ram:~/Documents/OS$
```