OS_Lab5

K. Ram Mohan

COE19B055

Q1)

```
#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];
        //two pipe descriptors
        int returnstatus1, returnstatus2;
        //return status after creating pipes
        int pid;
        char writemessage[20], readmessage[20], buffer[50];

        returnstatus1 = pipe(pipefds1);

        if(returnstatus1 == -1)
        {
        printf("Unable to create pipe1\n");
        return 0;
        }
```

```
returnstatus2 = pipe(pipefds2);

if(returnstatus2==-1)
{
printf("Unable to create pipe2\n");
return 0;
}

pid = fork();

if(pid==0)
{
        close(pipefds1[1]);
        //closing write for child in pipe1
        close(pipefds2[0]);
        //closing read for child in pipe2

        read(pipefds1[0], readmessage, sizeof(readmessage));
        //printf("Child process - read from pipe1: %s", readmessage);

        int n = strlen(readmessage);

        int j=0;
        for(int i=1; i<n; i++)
        {
                writemessage[j] = readmessage[n-1-i];
                j++;
        }
```

```
            write(pipefds2[1], writemessage, sizeof(writemessage));
        }
        else
        {
            close(pipefds1[0]);
            //closing read for parent in pipe1
            close(pipefds2[1]);
            //closing write for parent in pipe2

            printf("Parent process - Write to pipe1: ");
            fgets(buffer, 50, stdin);

            strcpy(writemessage, buffer);
            write(pipefds1[1], writemessage, sizeof(writemessage));

            read(pipefds2[0], readmessage, sizeof(readmessage));
            printf("Patent process - Read from pipe2: %s\n", readmessage);
        }
        return 0;
}
```

COE19B055

Q1) Since there is two way Communication between child & Parent we will use two pipes for it.

And we will close the unwanted file descriptors to avoid wrong actions ie, in a pipe only one can read and other can write, in the other pipe it will be converse.

We will close pipe using close() function.

Do the required operations

Output:

```
ram@ram:~/Documents/OS$ gcc -o pipe2 COE19B055_Lab5_Q1.c
ram@ram:~/Documents/OS$ ./pipe2
Parent process - Write to pipe1: Ram Mohan
Patent process - Read from pipe2: nahoM maR
ram@ram:~/Documents/OS$
```

Q2)

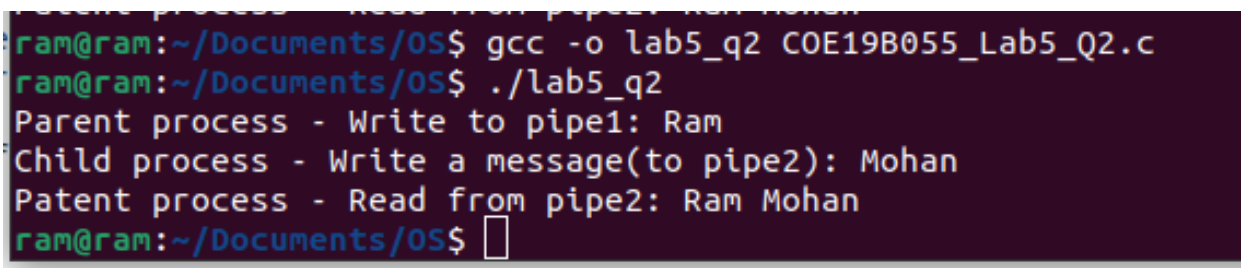
```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/wait.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<string.h>

int main()
{
	int pipefds1[2], pipefds2[2];
	//two pipe descriptors
	int returnstatus1, returnstatus2;
	//return status after creating pipes
	int pid;
	char writemessage[20], readmessage[20], buffer[50];

	returnstatus1 = pipe(pipefds1);
	if(returnstatus1 == -1)
	{
		printf("Unable to create pipe1\n");
		return 0;
	}

	returnstatus2 = pipe(pipefds2);
	if(returnstatus2==-1)
	{
		printf("Unable to create pipe2\n");
```

```
        return 0;
}

pid = fork();
if(pid==0)
{
        close(pipefds1[1]);
        //closing write for child in pipe1
        close(pipefds2[0]);
        //closing read for child in pipe2

        read(pipefds1[0], readmessage, sizeof(readmessage));

        printf("Child process - Write a message(to pipe2): ");
        fgets(buffer, 20, stdin);

        int n = strlen(readmessage);
        int i, j=0;

        for(i=0; i<n-1; i++)
        {
                writemessage[j] = readmessage[i];
                j++;
        }
        writemessage[j] = ' ';
        j++;
        for(i=0; i<strlen(buffer)-1; i++)
        {
                writemessage[j] = buffer[i];
```

```
                j++;
        }
        write(pipefds2[1], writemessage, sizeof(writemessage));
    }
    else
    {
        close(pipefds1[0]);
        //closing read for parent in pipe1
        close(pipefds2[1]);
        //closing write for parent in pipe2

        printf("Parent process - Write to pipe1: ");
        fgets(buffer, 50, stdin);

        strcpy(writemessage, buffer);

        write(pipefds1[1], writemessage, sizeof(writemessage));
        read(pipefds2[0], readmessage, sizeof(readmessage));
        printf("Patent process - Read from pipe2: %s\n", readmessage);
    }
    return 0;
}
```

Q2) Same as Q1

Output:



Q3)

A)

```
#include<stdio.h>
#include<stdlib.h>
#include<dirent.h>
//This header has opendir and readdir and closedir as well

int main(int argc, char *argv[])
{
	DIR *dp;
	struct dirent *dirp;
	//to acces files throught that directory
	char buffer[50];

	if (argc != 2) {
		printf("Enter directory name in format: ./Directory name\n");
		return 0;
	}

	dp = opendir(argv[1]);
	//Returns a DIR pointer to directory if exists else return NULL

	if(dp==NULL)
	{
		printf("NO such directory exists\n");
		return 0;
	}
	while((dirp=readdir(dp))!=NULL)
```

```
    {
        printf("%s\n", dirp->d_name);
    }
    closedir(dp);
    return 0;
}
```

Q3) i) opendir and closedir, readdir :-

CodingBoss

- Through "opendir" we can access a folder in our system by giving the path according to our current code location.

  It returns a DIR pointer to directory

  Syntax:- opendir("location/Path");

- Through "readdir" we can get the file names in the folders.

  Syntax:- readdir(dp)

  ↓ Pointer returned by opendir.

  The names of files will be returned under ->d_name.

Output:

```
ram@ram:~/Documents/OS$ gcc -o lab5_q3_1 COE19B055_Lab5_Q3_1.c
ram@ram:~/Documents/OS$ ./lab5_q3_1
Enter directory name in format: ./Directory name
ram@ram:~/Documents/OS$ gcc -o lab5_q3_1 COE19B055_Lab5_Q3_1.c
ram@ram:~/Documents/OS$ ./lab5_q3_1 ../O
NO such directory exists
ram@ram:~/Documents/OS$ gcc -o lab5_q3_1 COE19B055_Lab5_Q3_1.c
ram@ram:~/Documents/OS$ ./lab5_q3_1 ../OS
COE19B055_Lab5_Q2.c
mq_rece.c
..
COE19B055_Lab5_Q1.c
COE19B055_Lab5_Q3_2.c
clent_sm.c
sm
mq_sender
COE19B055_Lab5_Q3_1.c
fork.c
pipe2
mq_rece
lab5_q2
COE19B055_Lab5_Q4_2.c
COE19B055_Lab5_Q4_1.c
lab5_q4_2
lab5_q4_1
pipe1.c
sm_c
.
pipe1
server_sm.c
mq_sender1.c
fork
lab5_q3_1
ram@ram:~/Documents/OS$
```

Q3)

B)

```
#include<stdio.h>
#include<fcntl.h>
//This header has open and close function

int main()
{
	int fd;
	//Return <0 if failed else >0 if success
	fd = open("sample.txt", O_RDWR);
	//O_RDONLY - Read only
	//O_WRONLY - Write only
	//O_RDWR   - Read and Write
	//O_CREATE - create a file if doesnot  exist
	//O_EXCl   - Prevent creation if already exist

	if(fd<0)
	{
		printf("Failed to open file\n");
		return 0;
	}

	printf("Opened fd=%d\n", fd);
	close(fd);
	//close corresponding fd
}
```

COE19B055

Q3) ii) open and close :-

These are used to open and close files in the same directory.

Syntax:- open("file_name", ACCESS MODES).

ACCESS MODES:-

O_RDONLY	- Read only
O_WRONLY	- Write only
O_RDWR	- Read & write
O_CREATE	- Create a file
O_EXCL	- Stop create if already exists

It returns a number which we can use to close the file.

Output:

```
ram@ram:~/Documents/OS$ gcc -o lab5_q3_2 COE19B055_Lab5_Q3_2.c
COE19B055_Lab5_Q3_2.c: In function 'main':
COE19B055_Lab5_Q3_2.c:24:2: warning: implicit declaration of function 'close'
; did you mean 'pclose'? [-Wimplicit-function-declaration]
   24 |  close(fd);   //close corresponding fd
      |  ^~~~~
      |  pclose
ram@ram:~/Documents/OS$ ./lab5_q3_2
Opened fd=3
ram@ram:~/Documents/OS$
```

Q3)

C)

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

int main()
{
	int pipefds[2], returnstatus, pid;
	char readmessage[50], writemessage[50], buffer[50];

	returnstatus = pipe(pipefds);
	if(returnstatus == -1)
	{
		printf("Unable to create pipe\n");
		return 0;
	}

	pid = fork();

	if(pid<0){
		printf("Unable to execute fork\n");
		return 0;
	}else if(pid==0){
		printf("Child process - Enter message: ");
		fgets(writemessage, 50, stdin);
```

```
        write(pipefds[1], writemessage, sizeof(writemessage));
    }else{
        read(pipefds[0], readmessage, sizeof(readmessage));

        printf("Parent process - Message read is: %s", readmessage);
    }
}
```

Q3) iii) read and write:- COE19B055

We can use these in IPC using pipes.

They both take 3 arguments a) file descriptor
b) message
c) size of message

Output:

```
ram@ram:~/Documents/OS$ gcc -o lab5_q3_3 COE19B055_Lab5_Q3_3.c
ram@ram:~/Documents/OS$ ./lab5_q3_3
Child process - Enter message: Ram Mohan
Parent process - Message read is: Ram Mohan
ram@ram:~/Documents/OS$
```

Q4)

Side1:

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

#define SIZE 100

struct memory{
        char data[SIZE];
        int status;
        //status=0 process read data, status=2 process wrote something
};

int main()
{
        int shmid;
        char buffer[100];
        struct memory *shm;
        key_t key = ftok("shmfile", 65);

        shmid = shmget(key, SIZE, IPC_CREAT|0666);

        if(shmid<0)
```

```
{
        printf("Failed to create shared mameroy\n");
        return 0;
}

shm = shmat(shmid, NULL, 0);

printf("Enter end to stop process\n");
while(1)
{
        printf("Enter message: ");
        fgets(buffer, 100, stdin);

        strcpy(shm->data, buffer);
        shm->status = 1;

        if(strncmp(shm->data, "end", 3)==0)
        {
                break;
        }

        while(shm->status==1)
                sleep(1);

        while(shm->status==0)
                sleep(1);

        printf("Read from shared memory: %s", shm->data);
        shm->status=0;
```

```
                if(strncmp(shm->data, "end", 3)==0)
                {
                        break;
                }
        }
        shmdt(shm);
        shmctl(shmid, IPC_RMID, NULL);
}
```

Side2:

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

#define SIZE 100

struct memory{
        char data[SIZE];
        int status;
        //status=0 process read data, status=2 process wrote something
};

int main()
{
```

```
int shmid;
char buffer[100];
struct memory *shm;
key_t  key = ftok("shmfile", 65);

shmid = shmget(key, SIZE, IPC_CREAT|0666);

if(shmid<0)
{
        printf("Error in creating shared memory\n");
        return 0;
}

shm = shmat(shmid, NULL, 0);
printf("Enter end to stop process\n");

while(1)
{
        while(shm->status!=1)
                sleep(1);

        printf("Read from shared memory: %s", shm->data);
        shm->status = 0;

        if(strncmp(shm->data, "end", 3)==0)
        {
                break;
        }
```

```
            printf("Enter message: ");
            fgets(buffer, 100, stdin);

            strcpy(shm->data, buffer);
            shm->status=1;

            while(shm->status==1)
                  sleep(1);

            if(strncmp(shm->data, "end", 3)==0)
            {
                  break;
            }
      }
      shmdt(shm);
      shmctl(shmid, IPC_RMID, NULL);
}
```

Q4) IPC using shared memory:- COE19B055

Since SM will be in userspace we do not need any system calls to read or write. We just need to create a SM.

Shmget - we can get unique id for SM

Shmat - we can get the SM pointer

↓

It returns a pointer through which we can access it.

Shmdt - to detach SM

Shmctl - to destroy SM

Output:

```
ram@ram:~/Documents/OS$ gcc -o lab5_q4_1 COE1
ram@ram:~/Documents/OS$ ./lab5_q4_1
Enter end to stop process
Enter message: Hi
Read from shared memory: Hello
Enter message: I am Ram
Read from shared memory: This is mohan
Enter message: end
ram@ram:~/Documents/OS$
```

```
ram@ram:~/Documents/OS$ gcc -o lab5_q4_2 COE19B055_Lab5_Q4_2.c
ram@ram:~/Documents/OS$ ./lab5_q4_2
Enter end to stop process
Read from shared memory: Hi
Enter message: Hello
Read from shared memory: I am Ram
Enter message: This is mohan
Read from shared memory: end
ram@ram:~/Documents/OS$
```