LAB 6 ASSIGNMENT

Name: Birva Babaria

Roll no.: CE010

ID: 19CEUON064

Aim: Inter Process Communication. (Use of pipe system call)

Description:

1). ‘pipe’

Library: #include<unistd.h>

Syntax: int pipe(int pipefd[2]);

Description: pipe() system call creates a unidirectional data channel that can be used for inter process communication. It returns two file descriptors in array referring to the ends of the pipe. 0th index refers to the read end of the pipe. 1st index refers to the write end of the pipe. Data written to the write end of the pipe is buffered by the kernel until it is read from the read end of the pipe.

On success, it returns 0. On error, it returns -1 and errorno is set appropriately.

Example:
```
int pipefd[2];
status = pipe(pipefd);
```

2). ‘close’

Library: #include<unistd.h>

Syntax: int close(int fd);

Description: close() system call is used to close the file descriptor, so that it no longer refers to any file and can be reused. Any record lock held on the file it was associated with or owned by any process is released.

On success, it returns 0. On error, it returns -1 and errorno is set appropriately.

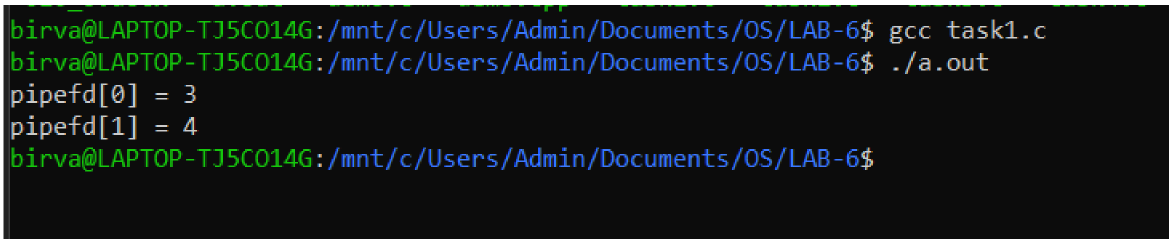
Example:
```
close(pipefd[1]);
close(pipefd[0]);
```

1). Write a program to create a pipe and print the values of pipe file descriptors.

CODE:

```
//create a pipe and print its descriptor values of that pipe.

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

//0,1,2 are reserved for stdin,stdout,stderr respectively.

void main()

{

	int pipefd[2];

	int status = pipe(pipefd);

	if(status != -1)

	{

		printf("pipefd[0] = %d\n",pipefd[0]);

		printf("pipefd[1] = %d\n",pipefd[1]);

		close(pipefd[0]);

		close(pipefd[1]);

	}

	else

	{

		printf("ERROR OCCURED\n");

	}

}
```

OUTPUT:



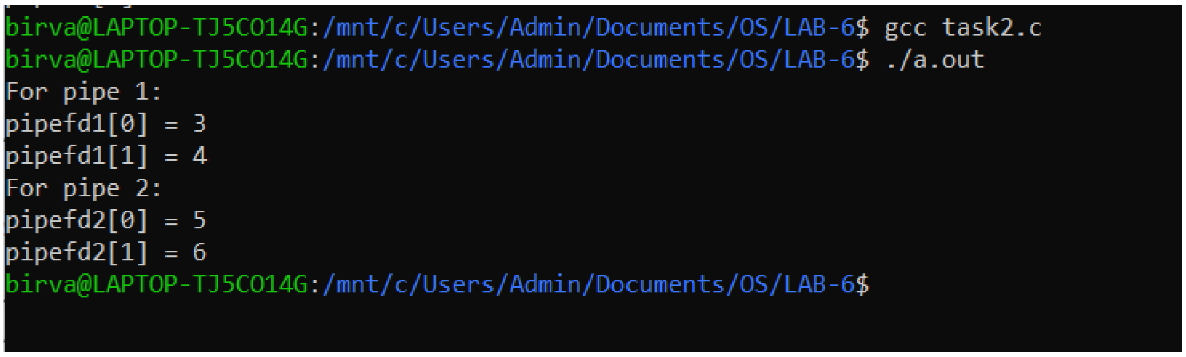
```
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-6$ gcc task1.c
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-6$ ./a.out
pipefd[0] = 3
pipefd[1] = 4
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-6$
```

2). Write a program to create two pipes and print the values of their file descriptors.

CODE:

```
//create two pipe and print descriptor values of both.
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
void main()
{
	int pipefd1[2];
	int pipefd2[2];
	int status1 = pipe(pipefd1);
	int status2 = pipe(pipefd2);
	if(status1 != -1)
	{
		printf("For pipe 1:\n");
		printf("pipefd1[0] = %d\n",pipefd1[0]);
		printf("pipefd1[1] = %d\n",pipefd1[1]);
		close(pipefd1[0]);
		close(pipefd1[1]);
	}
	else
	{
		printf("ERROR OCCURED\n");
	}
	if(status2 != -1)
	{
		printf("For pipe 2:\n");
		printf("pipefd2[0] = %d\n",pipefd2[0]);
		printf("pipefd2[1] = %d\n",pipefd2[1]);
		close(pipefd2[0]);
		close(pipefd2[1]);
	}
	else
	{
		printf("ERROR OCCURED\n");
	}
}
```

OUTPUT:



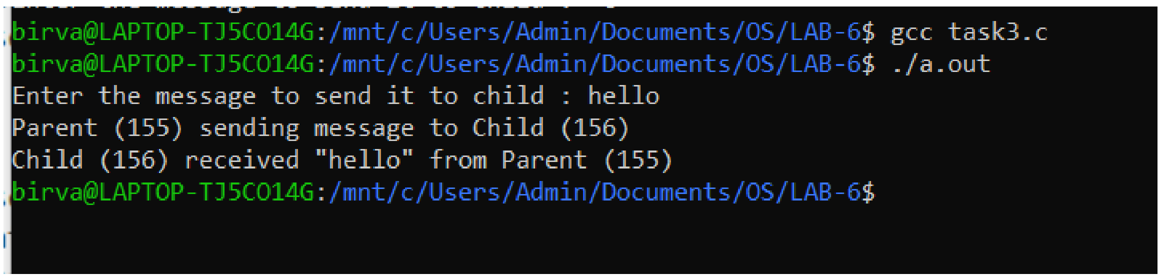
```
birva@LAPTOP-TJ5C014G:/mnt/c/Users/Admin/Documents/OS/LAB-6$ gcc task2.c
birva@LAPTOP-TJ5C014G:/mnt/c/Users/Admin/Documents/OS/LAB-6$ ./a.out
For pipe 1:
pipefd1[0] = 3
pipefd1[1] = 4
For pipe 2:
pipefd2[0] = 5
pipefd2[1] = 6
birva@LAPTOP-TJ5C014G:/mnt/c/Users/Admin/Documents/OS/LAB-6$
```

3). Write a program to pass a message from parent process to child process through a pipe.

CODE:

```
//parent should read the string from terminal
//parent should pass that string in the pipe
//child process should read from pipe and print on terminal
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include <sys/types.h>
#include<sys/wait.h>
void main()
{
	char buff[100];
	int pipefd[2];
	int status = pipe(pipefd);
	int pid = fork();
	if(status==-1 || pid==-1)
	{
		printf("ERROR OCCURED\n");
	}
	else if(pid == 0)
	{
		close(pipefd[1]);
		read(pipefd[0], buff, sizeof(buff));
		printf("Child (%d) received \"%s\" from Parent 
(%d)\n",getpid(),buff,getppid());
		close(pipefd[0]);
	}
	else
	{
		close(pipefd[0]);
		printf("Enter the message to send it to child : ");
		scanf("%s",buff);
		write(pipefd[1], buff, sizeof(buff));
		printf("Parent (%d) sending message to Child (%d)\n",getpid(),pid);
		close(pipefd[1]);
	}
}
```

OUTPUT:



```
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-6$ gcc task3.c
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-6$ ./a.out
Enter the message to send it to child : hello
Parent (155) sending message to Child (156)
Child (156) received "hello" from Parent (155)
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-6$
```

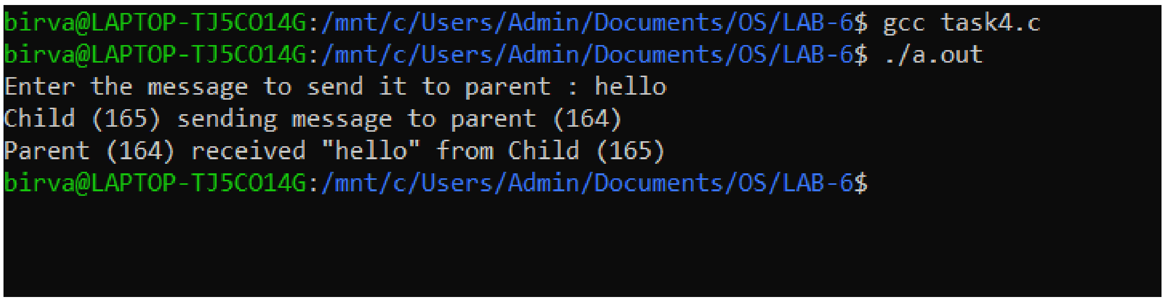
4). Write a program to pass a message from child process to parent process through a pipe.

CODE:

```
//child should read the string from terminal

//child should pass that string in the pipe

//parent process should read from pipe and print on terminal

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include <sys/types.h>

#include<sys/wait.h>

void main()

{

	char buff[100];

	int pipefd[2];

	int status = pipe(pipefd);

	int pid = fork();

	if(status==-1 || pid==-1)

	{

		printf("ERROR OCCURED\n");

	}

	else if(pid == 0)

	{

		close(pipefd[0]);

		printf("Enter the message to send it to parent : ");

		scanf("%s",buff);

		write(pipefd[1], buff, sizeof(buff));

		printf("Child (%d) sending message to parent 
(%d)\n",getpid(),getppid());

		close(pipefd[1]);

	}

	else

	{

		close(pipefd[1]);
```

```
          read(pipefd[0], buff, 100);

          printf("Parent (%d) received \"%s\" from Child
(%d)\n",getpid(),buff,pid);

          close(pipefd[0]);

     }

}
```

OUTPUT:



```
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-6$ gcc task4.c
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-6$ ./a.out
Enter the message to send it to parent : hello
Child (165) sending message to parent (164)
Parent (164) received "hello" from Child (165)
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-6$
```

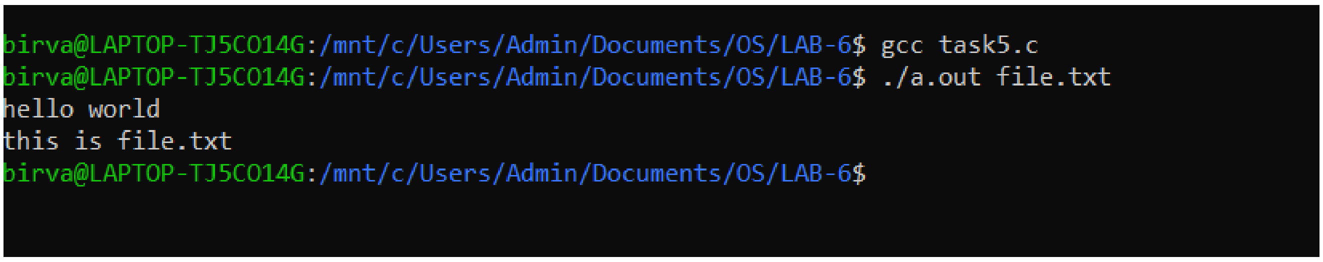
5). Write a program to pass file name from parent process to child process through a pipe, child process should read file name, open the file and print file contents.

CODE:

```
//Parent process should read file name

//using command line arguments

//Parent should pass file name through pipe

//Child process should read file name, open the file, print the contents

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <string.h>

#include <sys/stat.h>

#include <fcntl.h>

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

{

     if (argc == 1)

     {
```

```
		printf("FILENAME NOT PROVIDED!\n");
		return;
	}
	int pipefd[2];
	int status = pipe(pipefd);
	if (status == -1)
	{
		printf("ERROR IN CREATING PIPE\n");
		return;
	}
	pid_t pid = fork();
	if (pid == -1)
	{
		printf("ERROR IN CREATING PROCESS\n");
	}
	else if (pid)
	{
		close(pipefd[0]);
		char buff[1024];
		write(pipefd[1], argv[1], strlen(argv[1]));
	}
	else
	{
		close(pipefd[1]);
		char buff[500];
		int n = read(pipefd[0], buff, sizeof(buff));
		buff[n] = '\0';
		char buff1[1000];
		int fd = open(buff, O_RDONLY);
		n = read(fd, buff1, sizeof(buff1));
		buff1[n] = '\0';
		write(1, buff1, n);
	}
}
```

OUTPUT:



```
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-6$ gcc task5.c
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-6$ ./a.out file.txt
hello world
this is file.txt
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-6$
```

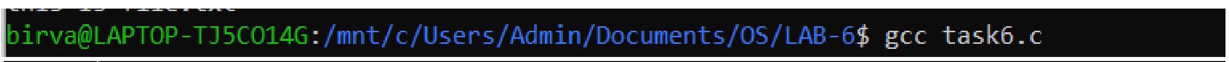
6). Write a program to pass file name from parent process to child process through a pipe, child process should read file name, open the file and print file contents.

CODE:

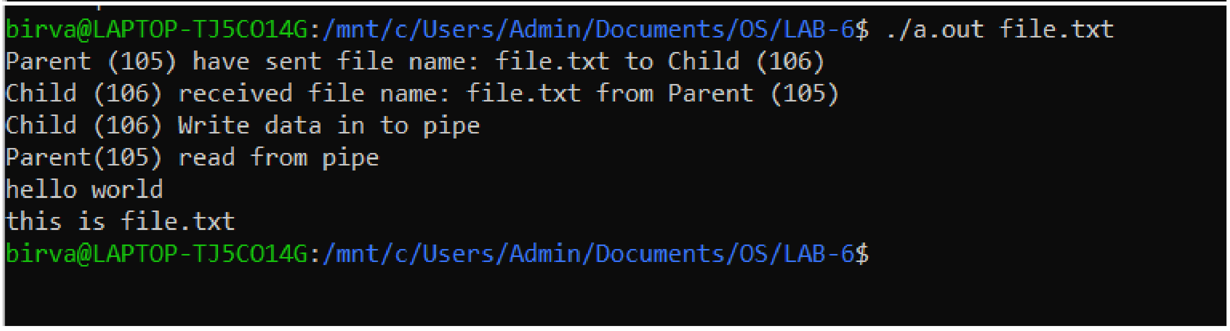
```
//Parent writes file name in pipe

//child reads file content

//Child should write contents in pipe , parent reads file contents and prints

#include <stdio.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <stdlib.h>

#include <fcntl.h>

#include <wait.h>

#include <string.h>

int main(int args, char *argv[])

{

	if (args == 1)

	{

		printf("FILENAME NOT PROVIDED!\n");

	}

	else

	{

		int pipefd1[2], pipefd2[2];

		if (pipe(pipefd1) == -1 || pipe(pipefd2) == -1)

		{

			printf("ERROR IN CREATING PIPE\n");

			exit(0);
```

```
        }
        else
        {
                int pid = fork();
                if (pid == -1)
                {
                        printf("ERROR IN CREATING PROCESS\n");
                        exit(0);
                }
                if (pid)
                {
                        char buff[1000];
                        close(pipefd1[0]);
                        close(pipefd2[1]);
                        write(pipefd1[1], argv[1], strlen(argv[1]));
                        printf("Parent (%d) have sent file name: %s to Child
(%d)\n",getpid(),argv[1],pid);
                        close(pipefd1[1]);
                        wait(NULL);
                        printf("Parent(%d) read from pipe\n", getpid());
                        int n = read(pipefd2[0], buff, sizeof(buff));
                        buff[n] = '\0';
                        write(1, buff, n);
                        close(pipefd2[0]);
                }
                else
                {
                        wait(NULL);
                        close(pipefd1[1]);
                        close(pipefd2[0]);
                        char buff[500];
                        int n = read(pipefd1[0], buff, sizeof(buff));
                        buff[n] = '\0';
                        printf("Child (%d) received file name: %s from Parent (%d)\n",getpid(),buff,getppid());
                        close(pipefd1[0]);
                        int fd = open(buff, O_RDONLY);
                        char buff1[1000];
                        if (fd == -1)
                        {
                              printf("FILE NOT FOUND\n");
                              exit(0);
                        }
                        int m = read(fd, buff1, sizeof(buff1));
                        buff1[m] = '\0';
                        printf("Child (%d) Write data in to pipe\n",getpid());
                        write(pipefd2[1], buff1, m);
                        close(pipefd2[1]);
                  }
            }
      }
return 0;
}
```

OUTPUT:



```
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-6$ gcc task6.c
```



```
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-6$ ./a.out file.txt
Parent (105) have sent file name: file.txt to Child (106)
Child (106) received file name: file.txt from Parent (105)
Child (106) Write data in to pipe
Parent(105) read from pipe
hello world
this is file.txt
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-6$
```