# LAB 7 ASSIGNMENT

**Name:** Birva Babaria

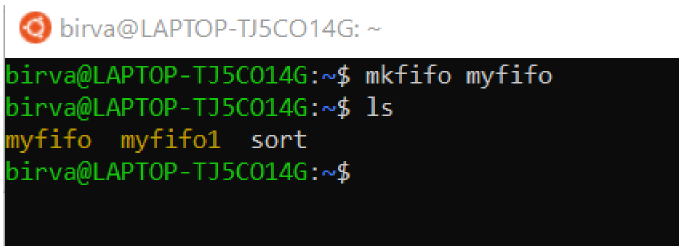
**Roll no.:** CE010

**ID:** 19CEUON064

**Aim:** Study of Dup2 system call and Execl function.

**Description:**

‘mkfifo’ is used to create named pipes.



```
birva@LAPTOP-TJ5CO14G:~$ mkfifo myfifo
birva@LAPTOP-TJ5CO14G:~$ ls
myfifo  myfifo1  sort
birva@LAPTOP-TJ5CO14G:~$
```

## 1). ‘dup2’ system call

**Library:** #include<unistd.h>

**Syntax:** int dup2(int oldfd, int newfd);

**Description:** dup2 system call makes newfd be the copy of oldfd. It closes newfd if necessary. If oldfd is a valid file descriptor and newfd has the same value as oldfd then dup2 just returns newfd. If oldfd is not valid descriptor then call fails and newfd is not closed.

On success, the old and new file descriptor can be used interchangeably.

**Example:** `dup2(fd,1);`

## 2). ‘exec’ system call

**Library:** #include<unistd.h>

**Syntax:** int execl(const char *pathname, const char *arg, .../* (char *) NULL */);

**Description:** The exec() family of functions replaces the current process image with new process image. First argument of the function is the name of the file to be executed. The const char *arg together describes a list of one or more pointers to null-terminated strings that represent the argument list available to the executed program. The list of the argument must be terminated by the null pointer and it must be cast to (char *)NULL.

On error, it returns -1 and errorno is set appropriately. Exec() function returns only if there is an error.

**Example:** `execl(pathname, "ls", (char *)NULL);`

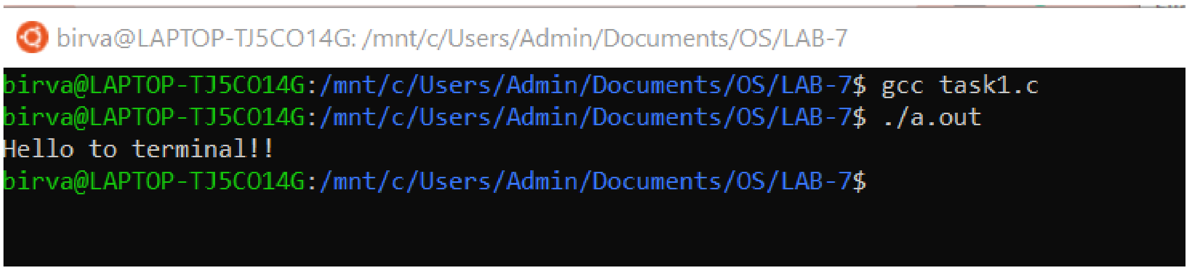
## 1). Create a blank file: "test.txt". Write a program to achieve following:

- **Print "Hello" message on stdout.**
- **Use dup2 in such a way that file behaves as stdout.**
- **Print "Hello" again to ensure that this time the message goes to file, not to the stdout.**

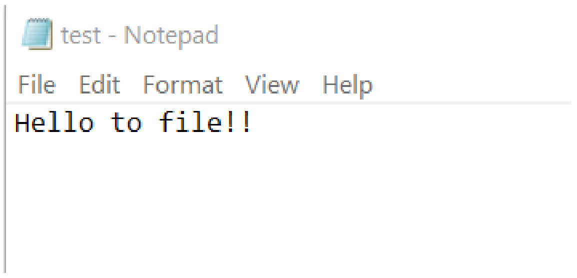
**CODE:**

```
#include<stdio.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include<stdlib.h>

#include<unistd.h>

void main()

{

      char buff[100] = "Hello to terminal!!\n";

      int fd = open("test.txt", O_WRONLY);

      int n = read(fd, buff, sizeof(buff));

      write(1,buff,sizeof(buff));

      char buff1[100] = "Hello to file!!\n";

      dup2(fd,1);

      write(1,buff1,sizeof(buff1));

}
```

**OUTPUT:**


birva@LAPTOP-TJ5CO14G: /mnt/c/Users/Admin/Documents/OS/LAB-7

```
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-7$ gcc task1.c
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-7$ ./a.out
Hello to terminal!!
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-7$
```



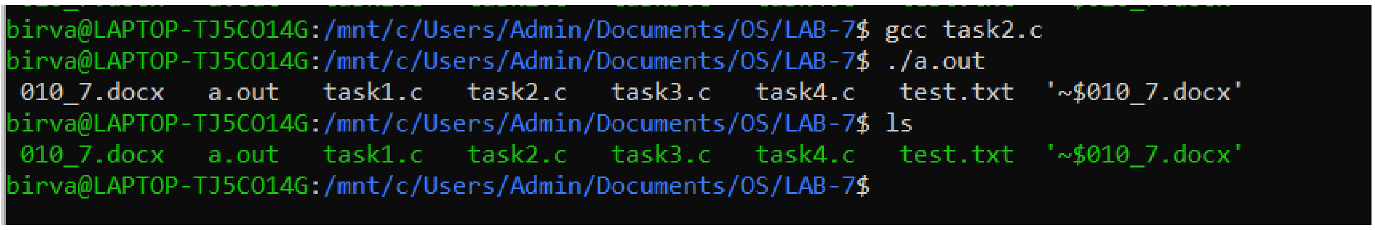
```
Hello to file!!
```

## 2). Write a program to execute ls command using execl.

## CODE:

```
#include<stdio.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include<stdlib.h>

#include<unistd.h>

void main()

{

      char *pathname = "/bin/ls";

      execl(pathname, "ls", (char *)NULL);

      //printf does not work after execl

      //because current process image is overwritten

      printf("hello");

}
```

## OUTPUT:



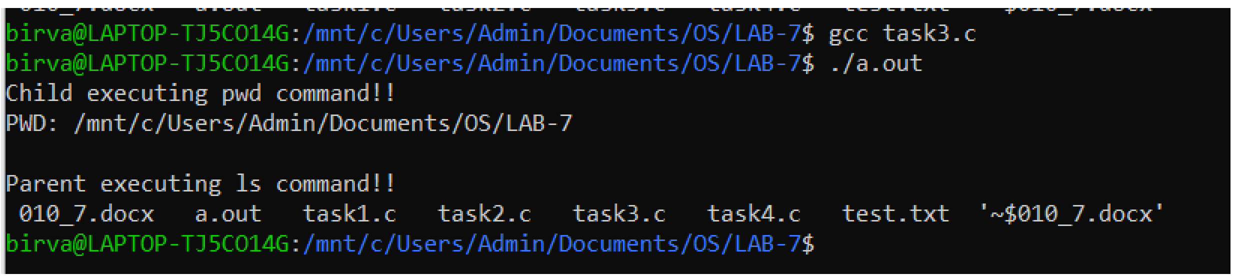
```
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-7$ gcc task2.c
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-7$ ./a.out
 010_7.docx   a.out   task1.c   task2.c   task3.c   task4.c   test.txt  '~$010_7.docx'
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-7$ ls
 010_7.docx   a.out   task1.c   task2.c   task3.c   task4.c   test.txt  '~$010_7.docx'
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-7$
```

**3). Write a program to create a child process that should run pwd command and the parent process should run ls command.**

**CODE:**

```
#include<stdio.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <sys/wait.h>

#include <fcntl.h>

#include<stdlib.h>

#include<unistd.h>

void main()

{

      int pid = fork();

      if(pid == 0)

      {

            printf("Child executing pwd command!!\n");

            char buff[500],*path;

            path = getcwd(buff,sizeof(buff));

            printf("PWD: %s\n\n",path);

      }

      else

      {

            wait(NULL);

            printf("Parent executing ls command!!\n");

            char *binaryPath = "/bin/ls";

            execl(binaryPath, "ls", (char *)NULL);

      }

}
```

**OUTPUT:**



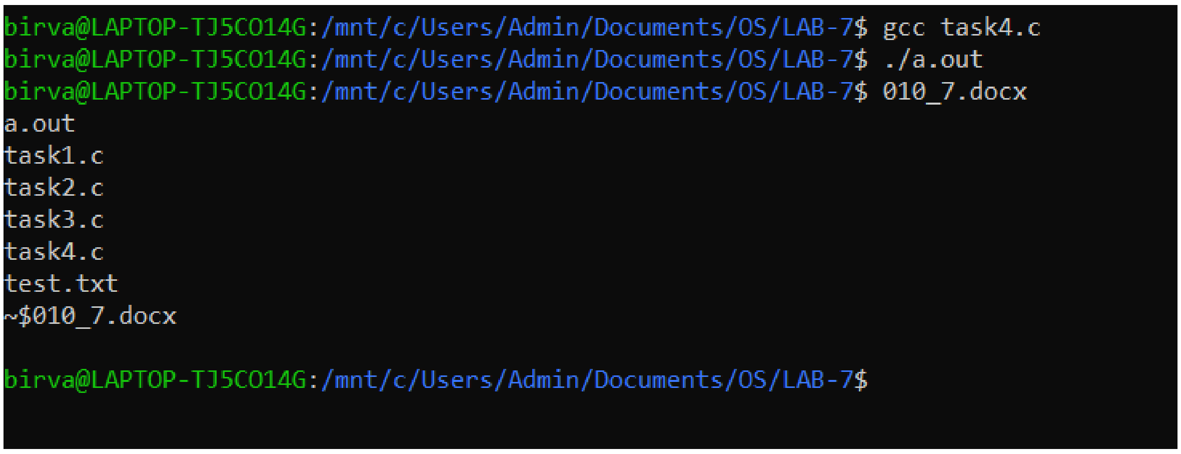
```
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-7$ gcc task3.c
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-7$ ./a.out
Child executing pwd command!!
PWD: /mnt/c/Users/Admin/Documents/OS/LAB-7

Parent executing ls command!!
 010_7.docx   a.out   task1.c   task2.c   task3.c   task4.c   test.txt  '~$010_7.docx'
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-7$
```

**4). Write a program to implement ls | sort functionality using the system calls and functions covered in the lab.**

**CODE:**

```
#include<stdio.h>

#include<sys/types.h>

#include<sys/stat.h>

#include<sys/wait.h>

#include<fcntl.h>

#include<stdlib.h>

#include<unistd.h>

void main()

{

	int pipefd[2];

	int status = pipe(pipefd);

	if(status == -1)

	{

		printf("ERROR CREATING PIPE\n");

	}

	int pid = fork();

	if(pid == 0)

	{

		close(pipefd[1]);

		dup2(pipefd[0], 0);

		char *binaryPath = "/bin/sort";

		execl(binaryPath, "sort", (char *)NULL);

	}

	else

	{

		close(pipefd[0]);

		dup2(pipefd[1], 1);

		char *binaryPath = "/bin/ls";

		execl(binaryPath, "ls", (char *)NULL);

	}
}
```

## OUTPUT:



```
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-7$ gcc task4.c
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-7$ ./a.out
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-7$ 010_7.docx
a.out
task1.c
task2.c
task3.c
task4.c
test.txt
~$010_7.docx

birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-7$
```

## 5). Write a program to achieve following:

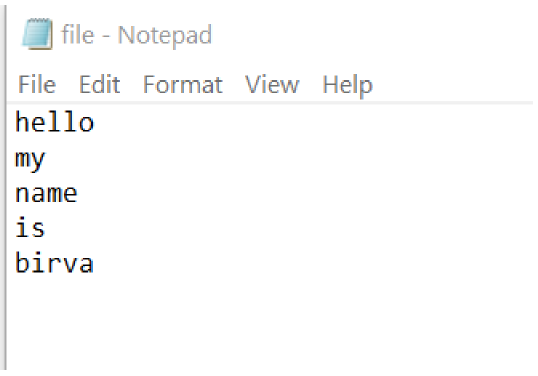
- **Child process should open a file with the contents to be sorted, pass the contents to parent process.**
- **Parent process should sort the contents of the file and display.**

## CODE:

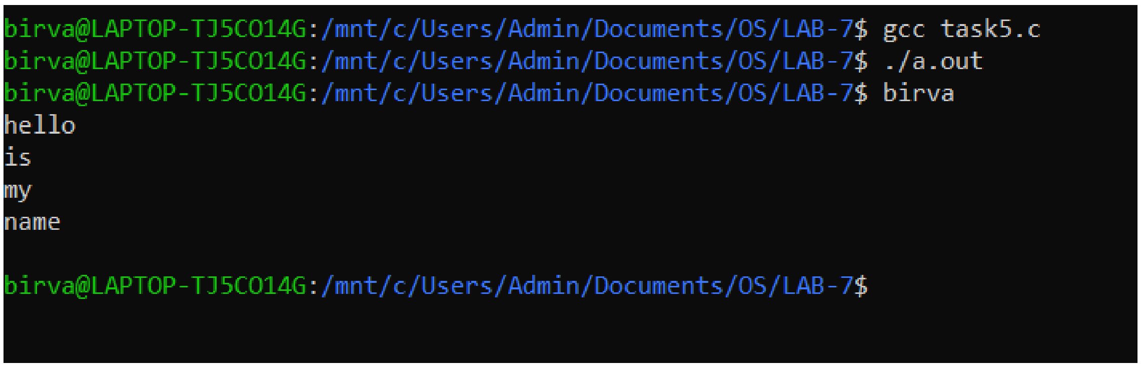
```
#include<stdio.h>

#include<sys/types.h>

#include<sys/stat.h>

#include<sys/wait.h>

#include<fcntl.h>

#include<stdlib.h>

#include<unistd.h>

void main()

{

	int pipefd[2];

	int status = pipe(pipefd);

	if(status == -1)

	{

		printf("ERROR CREATING PIPE\n");

	}

	int pid = fork();

	if(pid == 0)
```

```
    {
        close(pipefd[1]);
        dup2(pipefd[0], 0);
        char *binaryPath = "/bin/sort";
        execl(binaryPath, "sort", (char *)NULL);
    }
    else
    {
        close(pipefd[0]);
        char buff[1000];
        int fd = open("file.txt", O_RDONLY);
        int n = read(fd, buff, sizeof(buff));
        buff[n] = '\0';
        write(pipefd[1], buff, n);
    }
}
```

## OUTPUT:



```
hello
my
name
is
birva
```



```
birva@LAPTOP-TJ5C014G:/mnt/c/Users/Admin/Documents/OS/LAB-7$ gcc task5.c
birva@LAPTOP-TJ5C014G:/mnt/c/Users/Admin/Documents/OS/LAB-7$ ./a.out
birva@LAPTOP-TJ5C014G:/mnt/c/Users/Admin/Documents/OS/LAB-7$ birva
hello
is
my
name

birva@LAPTOP-TJ5C014G:/mnt/c/Users/Admin/Documents/OS/LAB-7$
```