Lab 2: IPC using UnNamed Pipes:

Processes frequently need to communicate with other processes. For example, in a shell pipeline, the output of the first process must be passed to the second process, and so on down the line. Thus there is a need for communication between processes, preferably in a well-structured way not using interrupts. In the following sections we will look at some of the issues related to this Inter Process Communication, or IPC.
Several methods for IPC are:

1. Pipes (Unnamed and Named)
2. Shared Memory
3. Message Passing

Unnamed Pipes:

Piping is a process where the output of one process is made the input of another. We have seen examples of this from the UNIX command line using C. Creating 'pipelines' with the C programming language can be a bit more involved than our simple shell example. To create a simple pipe with C, we make use of the pipe() system call. It takes a single argument, which is an array of two integers, and if successful, the array will contain two new file descriptors to be used for the pipeline. After creating a pipe, the process typically spawns a new process (remember the child inherits open file descriptors).

```
SYSTEM CALL: pipe();
PROTOTYPE: int pipe( int fd[2] );
RETURNS: 0 on success
         -1 on error: errno = EMFILE (no free descriptors)
        EMFILE (system file table is full) EFAULT (fd array is not valid)
        NOTES: fd[0] is set up for reading, fd[1] is set up for writing
```

The first integer in the array (element 0) is set up and opened for reading, while the second integer (element 1) is set up and opened for writing. Visually speaking, the output of fd1 becomes the input for fd0. Once again all data traveling through the pipe moves through the kernel.
As mentioned previously, once the pipeline has been established, the file descriptors may be treated like descriptors to normal files.

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
int main(void)
{
    int fd[2], nbytes;
    pid_t   childpid;
```

```
        char   string[] = "Hello, world!\n";
        char   readbuffer[80];

        pipe(fd);

        if((childpid = fork()) == -1)
        {
            perror("fork");
            exit(1);
        }

        if(childpid == 0)
        {
            /* Child process closes up input side of pipe */
            close(fd[0]);
            /* Send "string" through the output side of pipe */
            write(fd[1], string, (strlen(string)+1));

            exit(0);
        }

        else
        {
            /* Parent process closes up output side of pipe */ close(fd[1]);
            /* Read in a string from the pipe */
            nbytes = read(fd[0], readbuffer, sizeof(readbuffer));
            printf("Received string: %s", readbuffer);
        }
        return(0);
    }
```

Assignments:

1. White a program which will implement a two-way pipe for full duplex communication.
2. Write a program which will take one text file as argument and will send line by line through pipe. The receiving process will save the strings in another text file.