```
/*
                       COMMUNICATING VIA A PIPE

    Consider an application that requires one process to write a set of
    values, that are to be read by the other. Pipe is a system
    construct which facilitates such communication. A pipe is also a
    shared buffer. When a process tries to read from an empty pipe, it
    waits until someone has written something into the pipe. When the
    pipe is full, any process attempting to write into the pipe is made
    to wait.
*/

#include <stdio.h>
#include <unistd.h>    /* Include this file to use pipes */

#define BUFSIZE 80     /* We will write lines of 80 chars into the pipe.
                          The pipe has a large capacity and can accomodate
                          many such lines.
                       */

main()
{
        int fd[2];
        int n=0;
        int i;
        char line[BUFSIZE];

        /*  A pipe is treated as a file by the system. You must have used
            fopen() to open a file. fopen() returns a "file pointer" which
            is used in fprintf(), fscanf(), fclose() etc. However, when we
            wish to perform reads and writes in blocks from a file, we use
            the system call "open" to open a file. Internally files are
            always opened using the "open" call. For each process the system
            maintains a "file descriptor table" (FDT) containing an entry
            for each file opened by that process. When a new file is opened,
            a new entry is created in the FDT, and the entry number is
            returned as an integer called "file descriptor".

            Unlike in a file, we may want to both read and write from a
            pipe at the same time. Hence when a pipe is created, two file
            descriptors are created -- one for reading the pipe and one for
            writing into the pipe. The pipe() system call requires an
            array of two integers as parameter. The system returns the file
            descriptors through this array.
        */
        pipe(fd);  /* fd[0] is for reading,
                      fd[1] is for writing
                    */

        /*  To illustrate the working of the pipe, we will make the child
            process write the integer n into the pipe and make the parent
            to read from the pipe. We put sleep in the writer process
            (in this case the child) to show that the reader process waits
            for the writer to write into the pipe.

            To write a block of bytes into a pipe (or more generally into
            a file) the write() system call is used. Similarly read() is
```

```
	    used to read a block of bytes from a file. Refer to the online
	    man pages for these calls.
	*/

	if (fork() == 0) {

		close(fd[0]); /* The child will not read and
				    hence we close fd[0]
			       */

		for (i=0; i < 10; i++) {

			sprintf(line,"%d",n); /* Since write() accepts only
						      arrays of bytes, we
						      first write the integer n
						  into the char array "line"
						 */
			write(fd[1], line, BUFSIZE);
			printf("Child writes: %d\n",n);
			n++;
			sleep(2);
		}
	}
	else {

		close(fd[1]); /* The parent will not write and
				    hence we close fd[1]
			       */

		for (i=0; i < 10; i++) {

			printf("\t\t\t Parent trying to read pipe\n");
			read(fd[0], line, BUFSIZE);
			sscanf(line,"%d",&n); /* Read the integer from the
						      line of characters read
						      from the pipe
						 */
			printf("\t\t\t Parent reads: %d\n",n);
		}
	}
}
```