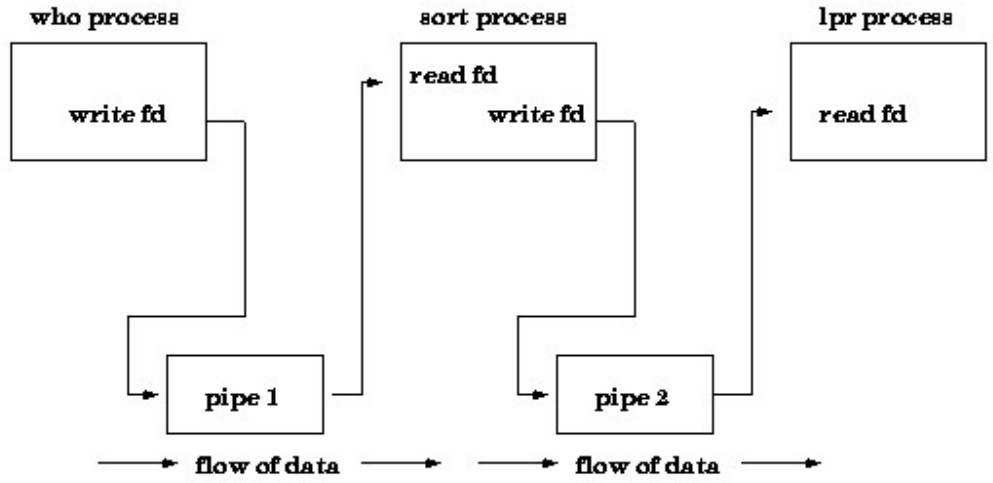
Practical 8 - Study of pipe()

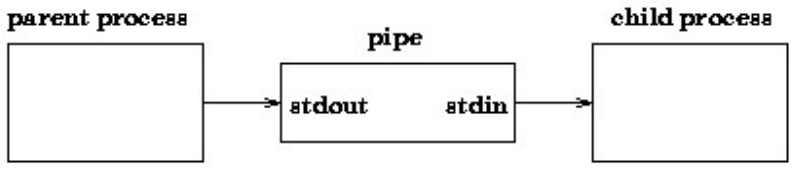
Pipe() system call in Unix

A Unix pipe provides a one-way flow of data.



For example, if a Unix users issues the command

who | sort | lpr



then the Unix shell would create three processes with two pipes between them:

SYNOPSIS

#include <unistd.h>

int pipe(int *filedes***[2]);**

DESCRIPTION

pipe() creates a pair of file descriptors, pointing to a pipe inode, and places them in the array pointed to by *filedes*. filedes[0] is for reading, filedes[1] is for writing.

A pipe can be explicitly created in Unix using the pipe system call. Two file descriptors are returned--fildes[0] and fildes[1], and they are both open for reading and writing. A read

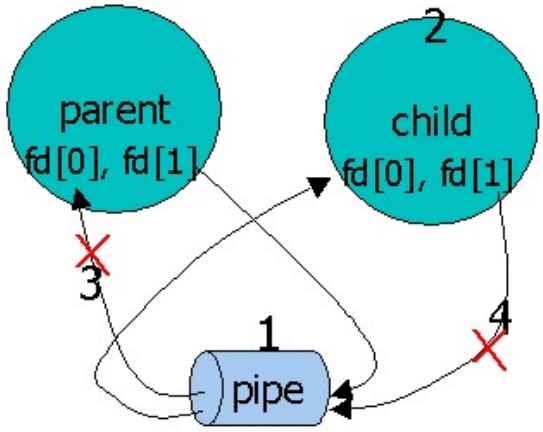
from fildes[0] accesses the data written to fildes[1] on a first-in-first-out (FIFO) basis and a read from fildes[1] accesses the data written to fildes[0] also on a FIFO basis.

When a pipe is used in a Unix command line, the first process is assumed to be writing to stdout and the second is assumed to be reading from stdin. So, it is common practice to assign the pipe write device descriptor to stdout in the first process and assign the pipe read device descriptor to stdin in the second process. This is elaborated below in the discussion of multiple command pipelines.

// program for parent process writing into pipe and child process reading from pipe.

```
int main( void ) {
   int n, fd[2];
   int pid;
   char line[MAXLINE];

   if (pipe(fd) < 0 )      // 1: pipe created
      perror( "pipe error" );
   else if ( (pid = fork( ) ) < 0 )     // 2: child forked
      perror( "fork error" );
   else if ( pid > 0 ) {   // parent
      close( fd[0] );      // 3: parent's fd[0] closed
      write( fd[1], "hello world\n", 12 );
   } else {                // child
      close( fd[1] );      // 4: child's fd[1] closed
      n = read( fd[0], line, MAXLINE );
      write( 1, line, n );
   }
   exit( 0 );
}
```



Program 1

//This program shows use of pipe() system call

```
#include<stdio.h>
#include<unistd.h>

int main()
{
        int n,fd[2];
        char buf[100];

        pipe(fd);
```

```
        printf("Writing to the pipe...\n");
        write(fd[1],"ABC",3);
        printf("Reading from the pipe...\n");
        n=read(fd[0],buf,100);
        write(STDOUT_FILENO,buf,n);
        exit(0);
}
```

Program 2

```
//This program shows use of pipe() system call
//Parent writes to the pipe and child reads from the pipe

#include<unistd.h>
#include<stdio.h>
#include<sys/stat.h>

int main()
{
        int n,fd[2];
        char buf[100];

        pipe(fd);

        switch(fork())
        {
                case -1:
                        printf("Fork error...\n");
                        exit(1);
                case 0:
                        close(fd[1]);
                        n=read(fd[0],buf,100);
                        write(STDOUT_FILENO,buf,n);
                        close(fd[0]);
                        break;
                default:
                        close(fd[0]);
                        write(fd[1],"writing to pipe\n",16);
                        close(fd[1]);
        }
        exit(0);
}
```

Program 3

```
//This program uses pipe for cat f1.txt | wc -l
//Child writes to the file and parent reads from the pipe

#include<stdio.h>
#include<unistd.h>

int main()
{
        int fd[2],n,in,out;

        pipe(fd);

        switch(fork())
        {
                case -1:
                        printf("Fork error\n");
                        exit(1);
                case 0:
                        close(fd[0]);
                        in=dup2(fd[1],STDOUT_FILENO);
                        execlp("cat","cat","f1.txt",(char *)0);
```

//const char *arg : describe a list of one or more pointers to null-terminated strings that //represent the argument list available to the executed program.

```
                        close(fd[1]);

                default:
                        close(fd[1]);
                        out=dup2(fd[0],STDIN_FILENO);
                        execlp("wc","wc","-c","-l",(char *)0);
                        close(fd[0]);
        }

        close(in);
        close(out);
        exit(0);
}
```