**Pipes**

* Conceptually, a pipe is a connection between two processes, such that the standard output from one process becomes the standard input of the other process
* It is possible to have a series of processes arranged in a pipeline, with a pipe between each pair of processes in the series.
* Implementation: A pipe can be implemented as a 10k buffer in main memory with 2 pointers, one for the FROM process and one for TO process
* One process cannot read from the buffer until another has written to it
* The UNIX command-line interpreter (e.g., csh) provides a pipe facility.

**% prog | more**

* This command runs the prog1 program and sends its output to the more program.

**Pipe System Call**

* pipe() is a system call that facilitates inter-process communication. It opens a **pipe**, which is an area of main memory that is treated as a "virtual file". The pipe can be used by the creating process, as well as all its child processes, for reading and writing.
* One process can write to this "virtual file" or pipe and another related process can read from it.
* If a process tries to read before something is written to the pipe, the process is suspended until something is written.
* The pipe system call finds the first two available positions in the process's open file table and allocates them for the read and write ends of the pipe. Recall that the open system call allocates only one position in the open file table.

Syntax in a C program:

#include <unistd.h>

int pip[2];

(void) pipe(pip);

With error checking:

#include <unistd.h>

int pip[2];

int result;

result = pipe(pip);

if (result == -1)

{

perror("pipe");

exit(1);

}

**Exercise 1:** Write a program in which a parent writes a string to a pipe and the child reads the string.

#include <stdio.h>

#include <string.h>

#include <unistd.h>

#include <sys/types.h>

#define BUFSIZE 10

int main(void) {

char bufin[BUFSIZE] = "empty";

char bufout[] = "hello";

int bytesin;

pid\_t childpid;

int fd[2];

if (pipe(fd) == -1) {

perror("Failed to create the pipe");

return 1;

}

bytesin = strlen(bufin);

childpid = fork();

if (childpid == -1) {

perror("Failed to fork");

return 1;

}

if (childpid) /\* parent code \*/

write(fd[1], bufout, strlen(bufout)+1);

else /\* child code \*/

bytesin = read(fd[0], bufin, BUFSIZE);

fprintf(stderr, "[%ld]:my bufin is {%s}, my bufout is {%s}\n",

(long)getpid(),bufin, bufout);

return 0;

}