int svc_pipe(int fds[2]);

The pipe system call is a svc call.

It creates a pair of file descriptors and places them in the array pointed to by fds. fds[0] is for reading and fds[1] is for writing.

Each pipe will use its own buffer, the writing process will fill the buffer and reading process will fetch characters from buffer and thus giving more room for writing. Reading a buffer will be on a first in first out(FIFO) basis. A circular buffer data structure has been used, i.e. write and read call puts and gets characters from buffer in a circular manner.

Read and write calls are blocking.
If buffer is empty read call will block, and if buffer is full write call will block.

If write end is closed, read will behave as reading end of file (EOF).
If read end is closed, write will return 0 (zero bytes written) and my_errno is set to EMYPIPE (my_errno is the global variable to represent error conditions specific to my implementation).

close(int fd) will close calling process's end of the pipe.

To demonstrate pipe() system call I am spawning multiple processes, each separated by a vertical bar on command line. Process on left side of vertical bar is going to write on pipe and read by process on right side of a bar.

The following commands have been implemented for demonstration of working of pipe.

- echo - prints arguments to standard output
- sort - reads list of numbers from standard input, sorts them and prints them to standard output
- unique - reads list of numbers from standard input (list should be sorted), and prints unique numbers from list to standard output
- print - reads from standard input and sends them to standard output

Usage

echo 5 5 5 5 4 4 4 4 4 3 3 3 2 2 1 1 | sort | unique | print

Design

Each process has its own standard input and output, handle of which is stored in its pcb. When user opens a pipe by calling svc_pipe() it returns a pair of file descriptors fds[2]. fd[0] is open in read mode and fds[1] is open in write mode.. Both descriptors points to the same buffer. Read and write to this buffer is synchronized by enabling and disabling interrupts.

When user enters commands separated by vertical bars, command line is parsed using a string tokenizer function. For each vertical bar, a pipe opened and standard output of process on left side of bar is set to write end of pipe and standard input of process on right side of bar is set to read end of pipe. The standard input/output of processes are set to pipes by passing pipe fds to spawn(). In spawn() these values are set in PCBs. And in getc() and putc(), read and write are made to buffers pointed by std_in and std_out of processes.

Since a pipe is owned by two processes, the pipe file system has a ref count. This ref count is decremented in pipe's close() function and if it reaches 0 then resources are freed.