IPC through PIPE

Dr. Manmath. N. Sahoo

Dept. Of CSE, NIT Rourkela

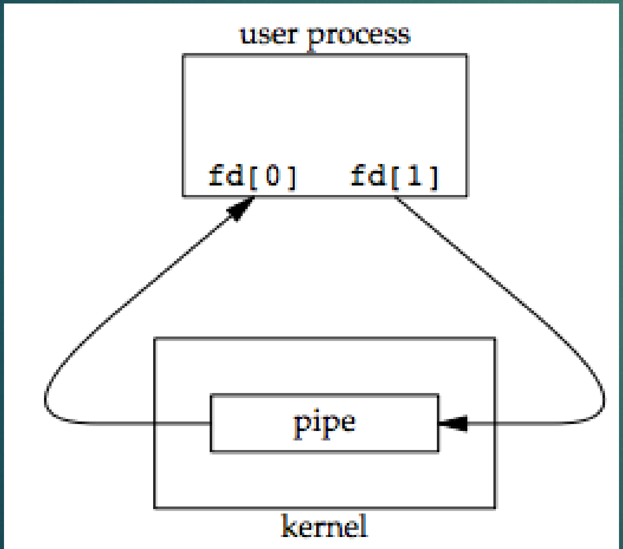
Inter-Process Communication: PIPE

- **PIPE** is a means of inter-process communication.
- Each Unix file has a file descriptor, similar to file pointer (FILE *) in C
- In Unix, all the devices are treated as files.
  - Standard input device is represented as a file with file descriptor ‘0’
  - Standard output device is represented as a file with file descriptor ‘1’
- A PIPE is defined with 02 file descriptors – one for read and other for write.

PIPE

- A PIPE can be created using a system call pipe() that takes one integer array of size 2, as argument, e.g.,

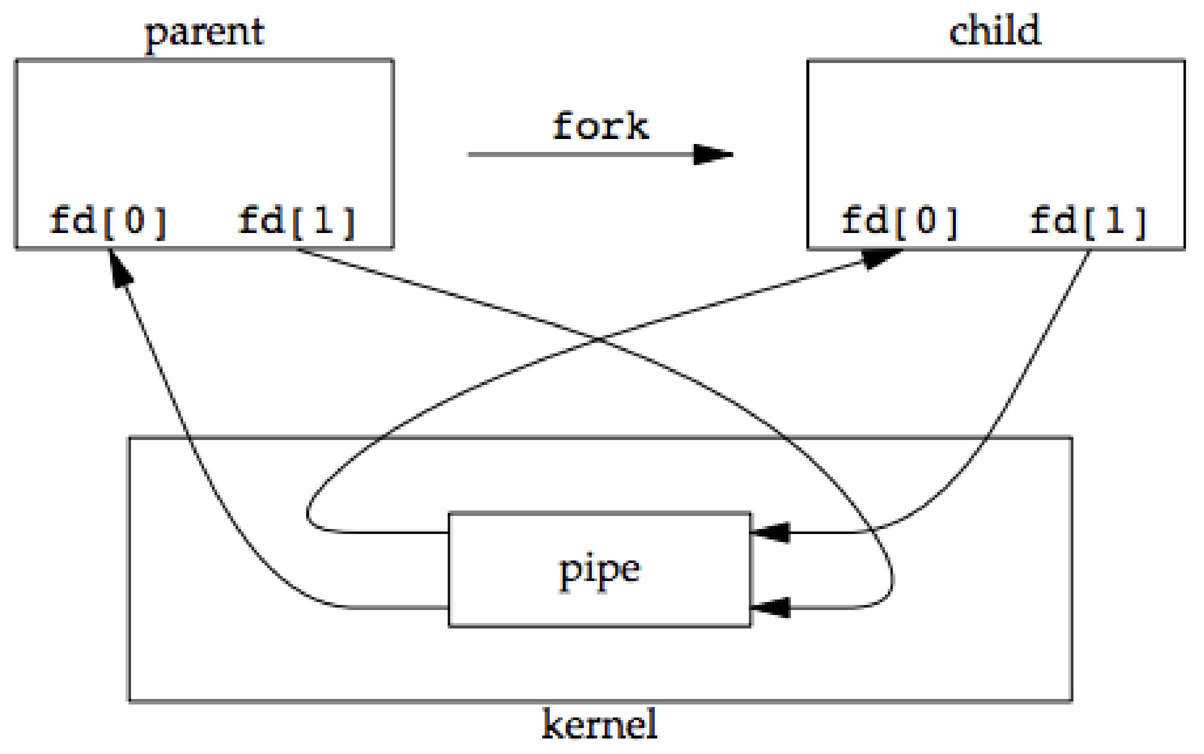
```
#include <unistd.h>
int fd[2]; pipe(fd);
/* Returns: 0 if OK, −1 on error */
```



fd[1] is the write file descriptor
fd[0] is the read file descriptor.

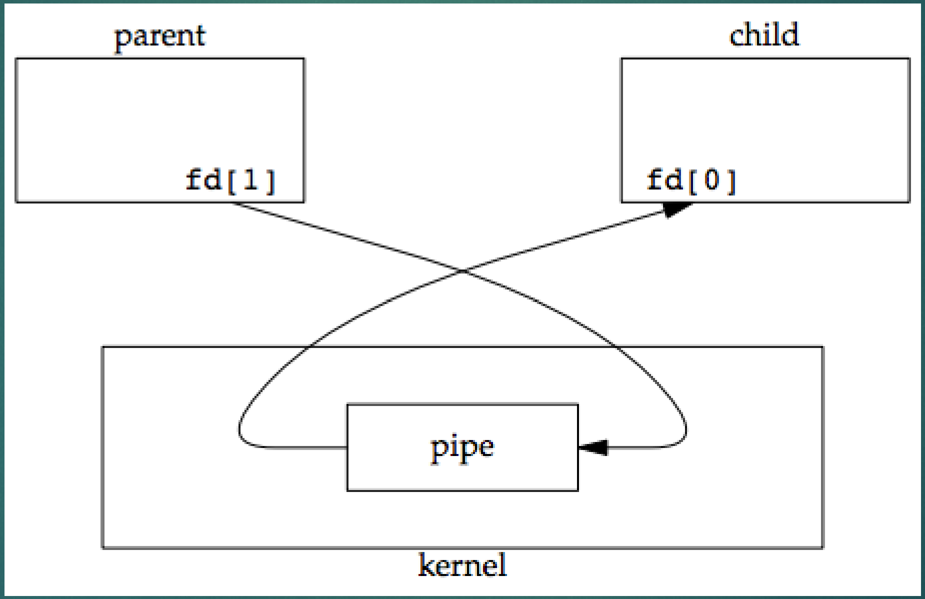
PIPE

- The child inherits all open file descriptors of parent
- However, PIPE supports half-duplex communication



PIPE

- Parent closes its read file descriptor and child closes write file descriptor. // `close(filedescriptor)`



PIPE

- If a process writes to a PIPE without any read fd opened then write returns -1, i.e., write fails.
- If a process reads from a PIPE without any write end opened, read returns 0 to indicate an end of file after all the data has been read from PIPE.
  - Technically, this end of file is not generated until all the write fds are not closed, and the reader waits until some input is available.

- For writing, at least ONE read fd must be opened.
- For reading, if you want end of pipe (or eof) character then all write fds must be closed.
- Example, ls –l | wc –l
  - While executing wc –l, it won’t terminate if it doesn’t find eof character.

```
#include<stdio.h> #include<errno.h> #include<unistd.h> #include<string.h>
int main(void){
    int     n,  fd[2];
    pid_t   pid;
    char    line[20], *msg = “hello world”;
    if (pipe(fd) == -1)        exit(1);
   else       printf(“PIPE created by parent successfully\n”);
    if ((pid = fork()) < 0)    exit(1);
    else if (pid > 0) {                         /* parent */
      close(fd[0]);
      write(fd[1], msg, strlen(msg));   //write(fd[1], &x, sizeof(int)); --for writing integer x
    } else {                                    /* child */
       close(fd[1]);
       n = read(fd[0], line, 20);         //read(fd[0], &y, sizeof(int)); --for reading int
      puts(line)
    }
}
```

Example: IPC through PIPE

exec() system call

- **int execl(const char *path, const char *arg[0], const char * arg[1], … , const char * arg[n], const char * 0);**
- **int execlp(const char *file, const char *arg[0], const char * arg[1], … , const char * arg[n], const char * 0);**
- **int execv(const char *path, char * const argv []);**
- **int execvp(const char *file, char * const argv[]);**

- path – complete path of the executable file
- file – name of the executable file
- arg[0] – command to be executed
- arg[1] … arg[n] – optional arguments for the command
- char * 0 – argument list is always ended with NULL string.

exec() system call

- execl("/bin/ls","ls", "-l", NULL);
- execlp("ls", "ls", "-l", NULL);

- char * const argv[]={"ls", "-l", NULL}
- execv(("/bin/ls",argv);
- execvp(("ls",argv);

- The path of "wc" file is **/user/bin/wc**
- To know the path of a command(file)
  - which wc
  - whereis ls

dup(), dup2()

close(0);

dup(fd[0]);

- duplicates fd[0] as the lowest unused file descriptor.
- this case it is 0

Can be replaced by a single statement

- dup2(fd[0],0);

More on manual page

- $man execlp
- $man dup
- $man dup2