Sistemas Operativos

Comunicación via pipes

Departamento de Ingeniería en Sistemas y Computación
Universidad Católica del Norte, Antofagasta.

Sincronización

- El sistema de comunicación por mensajes puede ser sincrónico (bloqueante) o asincrónico (no bloqueante)
- El send bloqueante deja al proceso bloqueado hasta que el mensaje es recibido
- El receive bloqueante deja al receptor bloqueado hasta que el mensaje está disponible

Sincronización

- El sistema de comunicación por mensajes puede ser sincrónico (bloqueante) o asincrónico (no bloqueante)
- El send bloqueante deja al proceso bloqueado hasta que el mensaje es recibido
- El receive bloqueante deja al receptor bloqueado hasta que el mensaje está disponible

Sincronización

- El sistema de comunicación por mensajes puede ser sincrónico (bloqueante) o asincrónico (no bloqueante)
- El send bloqueante deja al proceso bloqueado hasta que el mensaje es recibido
- El receive bloqueante deja al receptor bloqueado hasta que el mensaje está disponible

Sincronización

- En el send no bloqueante, el proceso envía y continúa
- En el receive no bloqueante, el receptor recibe un mensaje válido o nulo

Comunicación via Pipes

- Los Pipes fueron el primer mecanismo de comunicación de los primeros sistemas UNIX
- Normalmente permiten la comunicación de dos procesos en la forma productor-consumidor:
  - El productor escribe en un extremo del pipe, y el consumidor lee desde el otro extremo
  - Pipes corrientes son unidireccionales

Comunicación via Pipes

- Los Pipes fueron el primer mecanismo de comunicación de los primeros sistemas UNIX
- Normalmente permiten la comunicación de dos procesos en la forma productor-consumidor:
  - El productor escribe en un extremo del pipe, y el consumidor lee desde el otro extremo
  - Pipes corrientes son unidireccionales

Comunicación via Pipes

- Los Pipes fueron el primer mecanismo de comunicación de los primeros sistemas UNIX
- Normalmente permiten la comunicación de dos procesos en la forma productor-consumidor:
  - El productor escribe en un extremo del pipe, y el consumidor lee desde el otro extremo
  - Pipes corrientes son unidireccionales

Comunicación via Pipes

- Los Pipes fueron el primer mecanismo de comunicación de los primeros sistemas UNIX
- Normalmente permiten la comunicación de dos procesos en la forma productor-consumidor:
  - El productor escribe en un extremo del pipe, y el consumidor lee desde el otro extremo
  - Pipes corrientes son unidireccionales

Creación de un Pipe

- `pipe(int fd[])`
- El Pipe puede ser accesado usando el descriptor:
  - `fd[0]` para leer el pipe
  - `fd[1]` para escribir en el pipe
- No bloqueante: `pipe2(int fd[], O_NONBLOCK)`

Creación de un Pipe

- `pipe(int fd[])`
- El Pipe puede ser accesado usando el descriptor:
  - `fd[0]` para leer el pipe
  - `fd[1]` para escribir en el pipe
- No bloqueante: `pipe2(int fd[], O_NONBLOCK)`

Creación de un Pipe

- `pipe(int fd[])`
- El Pipe puede ser accesado usando el descriptor:
  - `fd[0]` para leer el pipe
  - `fd[1]` para escribir en el pipe
- No bloqueante: `pipe2(int fd[], O_NONBLOCK)`

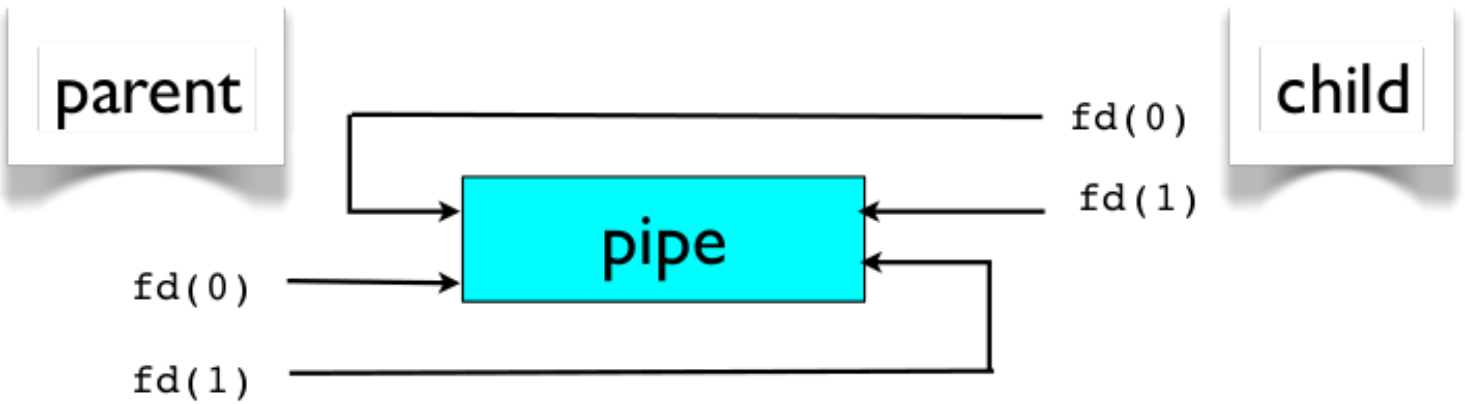
Creación de un Pipe

- `pipe(int fd[])`
- El Pipe puede ser accesado usando el descriptor:
  - `fd[0]` para leer el pipe
  - `fd[1]` para escribir en el pipe
- No bloqueante: `pipe2(int fd[], O_NONBLOCK)`

Creación de un Pipe

- `pipe(int fd[])`
- El Pipe puede ser accesado usando el descriptor:
  - `fd[0]` para leer el pipe
  - `fd[1]` para escribir en el pipe
- No bloqueante: `pipe2(int fd[], O_NONBLOCK)`

Comunicación via Pipes



Ejemplo

```c
#include <sys/types.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

#define BUFF_SIZE 25
#define READ 0
#define WRITE 1

int main(void)
{
        char write_msg[BUFF_SIZE] = "18\n";
        char read_msg[BUFF_SIZE];
        int fd[2];
        pid_t pid;
        /*crear el pipe*/
        pipe(fd);

        /* se crea un hijo */
        pid = fork();
        if (pid > 0) {
                close(fd[READ]); /*no se usa*/
                write(fd[WRITE], write_msg, strlen(write_msg)+1);
                close(fd[WRITE]); /* Ya no se usa*/
        }
        else { /*proceso hijo*/
                close(fd[WRITE]); /*no se usa*/
                read(fd[READ], read_msg, BUFF_SIZE);
                printf("El mensaje dice: %s", read_msg);
                close(fd[READ]);
        }
        return 0;
}
```

Términación de procesos

kill(pid,sigkill);

- pid: número entero con identificador del proceso
- sigkill: parámetro (9 obliga a terminar el proceso de forma inmediata)