Implementació de Pipes

Unitat 2.3 - Nel Banqué Torné

---

1. Pipes Sense Nom

Les **pipes sense nom** són canals de comunicació temporals que es poden implementar com a **buffers circulars** dins de la memòria assignada pel sistema operatiu. Aquest tipus de pipes desapareix quan els processos que les utilitzen acaben o quan es tanquen tots els descriptors associats.

Característiques de les Pipes Sense Nom:

- Són **transitòries** i existeixen només mentre hi ha processos actius que les utilitzen.
- Es creen en **memòria** i permeten la **comunicació unidireccional** entre processos.
- Els descriptors associats a la pipe són **heretats** pels processos fills, fet que permet la seva utilització.

---

2. Funcionament d'una Pipe

Quan es crea una pipe, aquesta assigna dos descriptors:

- **Descriptor de lectura**: El primer element de la pipe, obert per permetre la lectura.
- **Descriptor d'escriptura**: El segon element de la pipe, obert per permetre l'escriptura.

Exemple de Codi:

```
void main() {
    int fd[2]; // Descriptors associats a una pipe
    pid_t pidFill;

    // El primer element de la matriu (fd[0]) s'usa per a la lectura
   // El segon element de la matriu (fd[1]) s'usa per a l'escriptura
}
```

Procés Pare i Procés Fill:

- El **procés pare** tanca la lectura o l'escriptura segons la seva necessitat.
- El **procés fill** hereta els descriptors de la pipe i tanca l'escriptura o la lectura segons convingui.

Diagrama de Comunicació Unidireccional:

Procés Pare ---------> [Pipe] ---------> Procés Fill

- El pare tanca l'escriptura després d'enviar dades.
- El fill tanca la lectura després de llegir les dades.

---

3. Pipes Plenes i Blocatge

3.1 Blocatge d'Escriptura:

- Si un procés intenta escriure en una **pipe plena**, es bloqueja fins que la pipe es buidi parcialment i l'escriptura pot continuar.

3.2 Lectura amb Manca de Dades:

- Si un procés intenta llegir més dades de les que estan disponibles a la pipe, només es llegiran les dades disponibles, i es retornarà el nombre de **bytes** llegits.

Exemple d'Escenaris:

- **Pipe ple**: L'escriptura es bloqueja fins que el buffer es buidi.
- **Pipe amb menys dades del sol·licitat**: La lectura retorna el nombre de bytes llegits sense bloquejar el procés.

---

4. Pipe2 i Maneig de Descriptors

El **maneig de descriptors** en pipes és crucial per evitar errors en la comunicació:

- **Procés Pare**: Ha de tancar els descriptors que no utilitza, com l'escriptura a l'entrada de la pipe quan ja no es necessita.
- **Procés Fill**: També ha de tancar els descriptors adequats, per exemple, la lectura
- quan ja no s'espera més dades.

---

5. Simulació del Comportament amb Crides a Sistema

Per utilitzar pipes amb la crida a sistema **exec** i simular el comportament del sistema operatiu en la shell, necessitem redireccionar la sortida i l'entrada de la pipe a descriptors de fitxers predefinits assignats a cada procés.

Passos:

1. Tancar els descriptors estàndard.
2. Duplicar els descriptors de fitxer utilitzant la crida a sistema **dup()** (definida a unistd.h).

Exemple de Codi:

```
int dup(int fdold);
int dup2(int fdold, int fdnew);
```

- **dup**: Utilitza el descriptor de fitxer lliure més petit per duplicar el descriptor de fitxer **fdold**.
- **dup2**: Fa que **fdnew** sigui una còpia de **fdold**, tancant **fdold** si és necessari.

Exemple de Processos:

- **Procés 1**: Executa la comanda `ls`, que normalment imprimeix el llistat de fitxers al **stdout**. Amb una **pipe**, la sortida de `ls` no es mostra a la terminal, sinó que s'envia a la pipe.
- **Procés 2**: Executa la comanda `wc -l`, que compta el nombre de línies rebudes per **stdin**. El procés llegeix el contingut enviat per `ls` a través de la pipe.

---

6. Tancament de Descriptors

El procés pare ha de tancar els descriptors de fitxer que no utilitza. Si no es tanquen correctament, el procés fill que llegeix de la pipe no acaba mai, ja que no tots els descriptors han estat tancats.

Exemple de Codi:

```
// Pare
if (close(fd[0]) == -1)
    perror("close");
if (close(fd[1]) == -1)
    perror("close");
waitpid(pid1, 0, 0);
waitpid(pid2, 0, 0);
```