SISTEMAS OPERATIVOS

TALLER No. 6: IPC y sincronización de procesos en Linux (parte 1)

OBJETIVOS

1. Comprender el mecanismo de tuberías como modelo de comunicación y sincronización de procesos.

CONTENIDO DEL TALLER

1. Tuberías en Linux
2. Uso de tuberías con nombre y sin nombre
3. Productor – consumidor con tuberías
4. Si, el bendito problema de la sumatoria en un modelo de IPC
5. Propuesta de solución al problema de la sumatoria

DESARROLLO DEL TALLER

1. Tuberías en Linux

Una tubería es un mecanismo de comunicación y sincronización de procesos que proporciona el sistema operativo. En Linux hay dos tipos de tuberías: sin nombre y con nombre. Una tubería sin nombre **solo puede ser usada** por procesos que la hereden a través de una llamada a **`fork()`**. Las tuberías con nombre pueden ser usadas por dos o más procesos que se ejecutan en la misma máquina sin necesidad de que la hayan heredado a través de **`fork()`**.

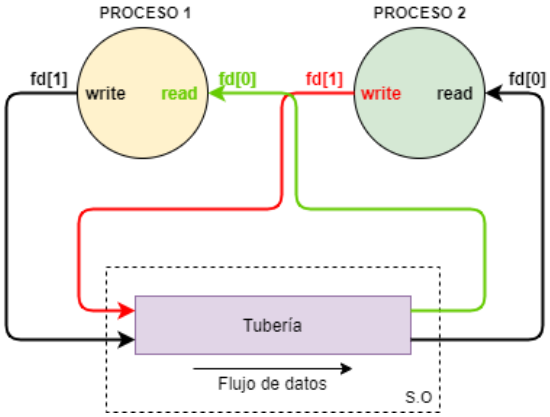
Crear tubería sin nombre: Para crear una tubería sin nombre se debe llamar a la siguiente *system call*.

```
int pipe(int fildes[2]);
```

Esta llamada entrega en el parámetro `fildes[2]` dos descriptores de archivos.

- `fildes[0]`: Descriptor de archivo que se usa para leer de la tubería.
- `fildes[1]`: Descriptor de archivo que se usa para escribir en la tubería.

La llamada a la *system call* devuelve 0 en caso de éxito o -1 en caso de error.



Crear tubería con nombre: Para crear la tubería con nombre se debe llamar a la siguiente *system call*.

```
int mkfifo(const char *pathname, mode_t mode);
```

- `pathname`: Nombre de la tubería.
- `mode`: modo de apertura. Por ejemplo el valor 0666 especifica que la tubería se debe crear con permisos de lectura y escritura para el usuario del proceso, el grupo y otros.

La llamada a la *system call* devuelve 0 en caso de éxito o -1 en caso de error.

Abrir una tubería con nombre: Para realizar operaciones de lectura y escritura en una tubería con nombre, primero se debe abrir con la llamada a la siguiente *system call*.

```
int open(const char *pathname, int flags);
```

- `pathname`: Nombre de la tubería.
- `flags`: Toma alguno de los siguientes valores.
  - `O_RDONLY`: Se abre la tubería solo para operaciones de lectura.
  - `O_WRONLY`: Se abre la tubería solo para operaciones de escritura.
  - `O_RDWR`: Se abre la tubería para lectura y escritura.

La llamada a la *system call* devuelve un descriptor (**`fd`**) de archivo que se puede usar para leer y escribir en la tubería.

Leer: Para leer de una tubería (con nombre o sin nombre) se debe hacer una llamada a la siguiente *system call*.

```
ssize_t read(int fd, void *buf, size_t count);
```

Escribir: Para escribir en una tubería (con nombre o sin nombre) se debe hacer una llamada a la siguiente *system call*.

```
ssize_t write(int fd, const void *buf, size_t count);
```

Cerrar: Para cerrar la tubería (con nombre o sin nombre) se debe hacer una llamada a la siguiente *system call*.

```
int close(int fd);
```

2. Uso de tuberías sin nombre y con nombre.

2.1 Escriba, compile y ejecute el siguiente programa

```c
#include <stdio.h>
#include <unistd.h>

int main(int argc, char *args[]) {
    /* descriptores de lectura y escritura */
    int fildes[2];
    /* se crea tubería sin nombre */
    if (pipe(fildes) < 0) {
        perror("Error al crear la tubería\n");
        return(1);
    }
    /* Se crea proceso hijo */
    if (fork() == 0) {
        /* dato que se pone en la tubería */
        int dato_entra = 100;
        write(fildes[1], &dato_entra, sizeof(int));
    } else { /* proceso padre */
        /* dato que sale de la tubería */
        int dato_sale;
        read(fildes[0], &dato_sale, sizeof(int));
        printf("Dato recibido por tubería: %d\n", dato_sale);
    }
    /* cerrar descriptores */
    close(fildes[0]);
    close(fildes[1]);
    return 0;
}
```

2.2 Escriba, compile y ejecute el siguiente programa

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

int main(int argc, char *args[]) {
    /* nombre y ubicación de la tubería */
    const char mypipe[] = "/tmp/mypipe";

    /* este proceso crea la tubería */
    if (mkfifo(mypipe, 0666) < 0) {
        perror("Error al crear la tubería\n");
        return(1);
    }

    /* se abre la tubería para escritura */
    int fd = open(mypipe, O_WRONLY);

    /* dato que entra a la tubería */
    int dato_entra = 200;

    /* escribir el dato en la tubería */
    write(fd, &dato_entra, sizeof(int));

    close(fd);
    return 0;
}
```

En una segunda terminal ejecute el siguiente programa que lee de la tubería que crea el programa anterior.

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

int main(int argc, char *args[]) {
    /* la tubería que creó el otro proceso */
    const char mypipe[] = "/tmp/mypipe";

    /* abrir la tubería en modo de lectura */
    int fd = open(mypipe, O_RDONLY);

    /* para guardar el dato que sale de la tubería */
    int dato_sale;

    /* leer el dato que se puso en la tubería */
    read(fd, &dato_sale, sizeof(int));
    printf("Dato que sale de tubería: %d\n", dato_sale);

    /* cerrar tuberia */
    close(fd);
    return 0;
}
```

3. Productor – consumidor con tuberías

Teniendo en cuenta los programas anteriores, escriba un programa en lenguaje C usando tuberías que se comporte en un modelo productor – consumidor (revise las diapositivas de clase). Se sugiere que añada retardos de algunos segundos (los que usted estime convenientes) en los ciclos de producción y consumo con la intención de comprobar el comportamiento. Tenga en cuenta que aquí el búfer es ilimitado, es decir, no tiene necesidad de preocuparse por aspectos como el tamaño del búfer, ranuras llenas o ranuras libres. Simplemente el productor pone datos en la tubería y el consumidor los consume de la tubería. Use los dos modelos de tuberías: con nombre y sin nombre.

4. Si, el bendito problema de la sumatoria en un modelo de IPC

Escriba un programa en lenguaje C que defina la siguiente función sumar.

```c
void sumar(int li, int ls) {
    int j;
    int suma = 0;
    for (j = li; j <= ls; j++) {
        suma = suma + j;
    }
    sumatotal = sumatotal + suma;
}
```

La variable **`sumatotal`** es una variable global.

La función **`main`** de este mismo programa debe crear un proceso hijo usando **`fork()`**, que invoque a la función **`sumar`** como se indica a continuación e inmediatamente después debe imprimir el contenido de la variable **`sumatotal`**.

```
sumar(1,50);
```

El proceso padre debe llamar a la función **`sumar`** como se indica a continuación e inmediatamente después debe imprimir el contenido de la variable **`sumatotal`**.

```
sumar(51,100);
```

Ejecute arias veces el programa y verifique que siempre obtenga como resultado **5050**. De no ser así, explique qué es lo que está pasando.

5. Propuesta de solución al problema de la sumatoria

Resuelva el problema del ejercicio anterior usando tuberías y verifique que siempre obtiene como resultado 5050. ¿por qué la tubería resuelve el problema?