UNIVERSITAT DE BARCELONA
Grado de Ingeniería Informàtica
Facultad de Matemáticas e Informática

Sistemas Operativos I

Práctica 3

Comunicación entre procesos

Grupo A05 :

- **Alejandro Cantero López**
- **William David Sotto Jaime**

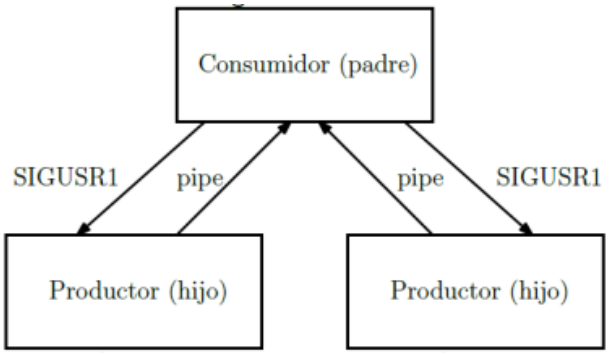
Introducción

En esta práctica se desarrollará un esquema de productor-consumidor, sabiendo sobre los métodos que nos ofrece el sistema operativo para comunicar procesos entre sí. Usaremos lectura de archivos, tuberías y señales. Los productores generarán información y la escribirán en la tubería, entregándosela a los consumidores, los cuales leerán esta respuesta y realizarán respectivos trabajos a partir de aquí.

En este informe se planteará el esquema de consumidor-productor en el juego de Pierda-Papel-Tijeras. En nuestro caso, el consumidor será el árbitro que recibirá la respuesta de nuestros dos productores (jugadores) para saber quién ha ganado el turno.

Estructura Padre-Hijo.

Para seguir este proceso de padre-hijo en el juego de Piedra, papel o tijera, debimos seguir el siguiente esquema:



Lo que hicimos fue crear un hijo al inicio de nuestro programa y sabiendo que el hijo ejecutaría también la parte del código que seguía, preguntamos al proceso si era el padre o el hijo. Si era el padre creaba otro hijo, y así, podíamos tener dos productores y un solo consumidor.

Viendo el esquema se podría intuir que requerimos de dos tuberías, sin embargo al final lo hicimos con una sola. Esto lo pudimos hacer de esta manera porque hacemos una conversación en série de padre a hijo, es decir, el padre primero habla con un hijo y después con el otro, por lo que la tubería estará vacía para cuando el siguiente hijo quiera escribir.

Una señal en sistemas operativos es una interrupción asincrónica que se envía a un proceso para notificarle de un evento o solicitarle que realice una acción determinada. Se podrían usar estas señales para:

- Finalización de procesos: Sigkill (forzar la terminación inmediata de un proceso), Sigterm, se utiliza para solicitar que un proceso se detenga de forma ordenada.
- Gestión de errores.
- Temporización o Control de Procesos.
- O para la comunicación de Procesos: SigUsr1, SigUsr2.

El que utilizamos en nuestro trabajo es una señal de comunicación de procesos, el SIGUSR1. Tanto los productores como el consumidor utilizan esta señal.
Para manejarnos utilizando el SIGUSR1 a cada productor y al consumidor le definimos el método al que debe llamar en su llamada.

```
signal( SIGUSR1,  siguser1  );
```

El esquema que sigue nuestro código se basa en que los dos hijos queden esperando a que su proceso padre envíe la señal SIGUSR1, el padre llama al primer hijo para que llene el buffer con una elección entre piedra, papel o tijera. El padre lee el buffer con la respuesta y le pedirá al otro hijo que haga lo mismo. Esto se repite las veces que se hayan declarado por el usuario como parámetro de entrada.
Finalmente, el padre dictará el resultado…

En nuestro código existen 3 métodos importantes:

- ☐ **logicaProductor**
- ☐ **logicaConsumidor**
- ☐ **siguser1**.

Siguser1: Es el método que se ejecutará cuando se dé la señal siguser1 a partir del comando **'kill'**. La función de este método es la de cambiar el valor de la variable global siguser1 a 1, de tal manera que el padre dejará de esperar la respuesta del hijo y leerá el buffer (pipe) obteniendo la respuesta de este.

LogicaProductor: Es el encargado de generar una respuesta (piedra, papel o tijera) y escribirla en el buffer.

LogicaConsumidor: es el padre de los procesos. Es el encargado de enviar las señales a los hijos para que escriban su respuesta, y de leer el buffer que han llenado los hijos. Además procesa la respuesta y da el ganador, o dicta el empate.

El output por consola se vería así.

```
El Jugador 1 ha elegido Papel
El Jugador 2 ha elegido Piedra
Este turno lo ha ganado el Jugador 1!...
Solución del Encuentro: envuelve

El Jugador 1 ha elegido Papel
El Jugador 2 ha elegido Papel
Solución del Encuentro: Ha sido un empate

El Jugador 1 ha elegido Tijeras
El Jugador 2 ha elegido Tijeras
Solución del Encuentro: Ha sido un empate

Wins P1: 1, Wins P2: 0
El ganador del Juego es el Jugador 1 !!
```

```
El Jugador 1 ha elegido Piedra
El Jugador 2 ha elegido Papel
Este turno lo ha ganado el Jugador 2!...
Solución del Encuentro: envuelve

El Jugador 1 ha elegido Tijeras
El Jugador 2 ha elegido Tijeras
Solución del Encuentro: Ha sido un empate

El Jugador 1 ha elegido Tijeras
El Jugador 2 ha elegido Papel
Este turno lo ha ganado el Jugador 1!...
Solución del Encuentro: cortan

Wins P1: 1, Wins P2: 1
Suerte para la próxima empatados...
```

```
El Jugador 1 ha elegido Piedra
El Jugador 2 ha elegido Papel
Este turno lo ha ganado el Jugador 2!...
Solución del Encuentro: envuelve

El Jugador 1 ha elegido Tijeras
El Jugador 2 ha elegido Tijeras
Solución del Encuentro: Ha sido un empate

El Jugador 1 ha elegido Papel
El Jugador 2 ha elegido Papel
Solución del Encuentro: Ha sido un empate

El Jugador 1 ha elegido Tijeras
El Jugador 2 ha elegido Papel
Este turno lo ha ganado el Jugador 1!...
Solución del Encuentro: cortan

Wins P1: 2, Wins P2: 1, Empates: 7
El ganador del Juego es el Jugador 1 !!
```

Para una buena lectura también especificamos los empates que hubieron durante la partida, así si el total de turnos es muy grande, se puede apreciar si se jugaron todos los turnos…

Caso de turnos=10

Comentario: La idea del encuentro entre los productores con arbitraje del consumidor es que estos productores deben estar esperando al consumidor para poder iniciar su elección, sin embargo, hay casos en los que el padre envía la señal SIGUSR1 antes de que los hijos esten esperando al padre. Para arreglar este pequeño porcentaje de fallos, podemos colocar un sleep() antes de ejecutar el método de nuestro consumidor, de esta forma se evita por completo estos casos.

```
    sleep(0.2);
    logicaConsumidor(N, o_array,r_array, fd);
```

El código es totalmente funcional sin el sleep, pero aún así corre el riesgo de algún fallo.