Taller de IPC

Sistemas Operativos

3 de septiembre de 2024

Segundo cuatrimestre - 2024

1. Ejercicios

En este taller se implementarán mecanismos de comunicación entre procesos mediante el uso de `pipes`.

1.1. Ejercicio 1: Mini Shell

Se pide implementar parte de la funcionalidad de un `shell` minimal. El mismo debe soportar comandos de dos formas:

- Nombre de un programa junto con sus argumentos. Por ejemplo: `ls -al`
- Múltiples programas (con sus argumentos) comunicados por `|`. Por ejemplo: `ps aux | grep root | grep sbin`.

Para esto:

a) Partiendo del esqueleto provisto por la cátedra, implementar el programa `mini-shell` que recibe una cadena de texto que contiene comandos unidos con `pipes` y se comporte de la misma manera que lo haría cualquier otro `shell`[1].
Por ejemplo: `./mini-shell 'du -h /home | head -n 10 | sort -rh'`.
ATENCIÓN: Recordar que se deben colocar comillas para que funcione correctamente.
Nota: No se puede utilizar la función `system` para dicha tarea.
Nota2: En el ejemplo usado, en caso que el directorio **home** sea muy grande, puede ser que el proceso no termine con exito, pero aun asi se imprima igual el resultado.

1.2. Ejercicio 2: *One ring to rule them all*

Se pide implementar un proceso que:

a) Realice un esquema de comunicación en forma de anillo para intercomunicar a sus procesos hijos (al menos tres), y ejecute el protocolo de comunicación detallado a continuación:

[1] Se aconseja partir del Ejercicio 2 de la presentación "Comunicación entre Procesos (IPC)" y extenderlo a tres comandos encadenados.

- Cada uno de los procesos hijos debe comunicarse con exactamente dos procesos: su antecesor y su sucesor (visto desde el orden de creación de los mismos). El proceso actual recibe un mensaje del antecesor y envía un mensaje al sucesor. En este caso, dicha comunicación se realizará a través de `pipes`.
- Para comenzar la comunicación, el padre envía a alguno de los procesos hijo del anillo un número entero como mensaje a comunicar. Este proceso hijo será el encargado de generar un *número secreto* (que solo él conoce) y, además, enviar el mensaje recibido del padre al siguiente proceso en el anillo. Luego de recibir un mensaje, cada proceso debe incrementar el valor recibido en uno y enviarlo al próximo proceso en el anillo. Esto se debe repetir hasta que el proceso inicial reciba de su antecesor un número que sea mayor o igual a su *número secreto*.
- Ejemplo: el hijo 1 recibe el mensaje inicial, lo incrementa, y se lo envía al hijo 2, éste lo vuelve a incrementar y se lo envía al hijo 3, y así sucesivamente hasta llegar al proceso inicial el cual dará pie a una nueva ronda en el anillo si el número recibido no supera su *número secreto*. Caso contrario, no generará nuevos mensajes en el anillo pero sí enviará al proceso padre el último valor obtenido. El padre también deberá mostrar en la salida estándar el resultado final del proceso de comunicación.

Se espera que el programa pueda ejecutarse como `./anillo <n><c><s>`, donde:

- `<n>` es la cantidad de procesos del anillo.
- `<c>` es el valor del mensaje inicial.
- `<s>` es el número de proceso que inicia la comunicación.

El número secreto `p` será generado aleatoriamente únicamente por el proceso distinguido, de manera que `c` $<$ `p`.

Antes de empezar:

- Leer cuidadosamente los manuales de las funciones `read()` y `pipe()`

1.3. Ejercicio 3: *Ahora usemos sockets*

Escriba un programa para un servidor que acepte varias conexiones de clientes, haciendo un `fork` por cada nuevo cliente que reciba.

Cada cliente tiene que poder chatear simultáneamente con el servidor. El cliente enviará al servidor una operación matemática básica (suma, multiplicación, resta o división) para que resuelva, el servidor hará el procesamiento y deberá retornar un resultado.

La conexión no debe terminar ahí, sino que el cliente puede seguir hablando con el servidor. Solamente sale diciendo la palabra *exit* o haciendo `CTRL+C` (SIGINT).

2. Notas útiles para la resolución del taller

2.1. Preliminares

Para la resolución del taller es conveniente repasar la Clase Práctica 2 "Comunicación entre Procesos (IPC)", así como poder entender el código de los programas provistos junto con la misma y haberlos compilado y ejecutado por separado.

2.2. Consideraciones

Para tener en cuenta:

- En un *pipeline* todos los procesos se pueden ejecutar al mismo tiempo, es decir, de manera **concurrente**.
- Es importante no dejar abiertos los *file descriptors* (FD) de los `pipes` que no se utilicen. En este caso, ¿qué sucedería si no se cierran algunos FD de los `pipes`? ¿Por qué?

2.3. Manual (RTFM)

Recuerden que las páginas del manual son de mucha utilidad. Por ejemplo:

- Consultar el manual de una función: `man 2 pipe`, `man 2 read`, `man 2 dup`
- Obtener una visión general de *pipelines*: `man 7 pipe`, y
- Leer el manual del manual: `man man`