Comunicación entre Procesos
(IPC)

Departamento de Computación, FCEyN
Universidad de Buenos Aires

22 de Agosto de 2024

Temario

Temario

Funciones Útiles (1)

Importante

¡Complementar todo con el manual!

Procesos

- **int fork()**: Crea un nuevo proceso, copiando el actual. Retorna **0** en el proceso hijo, y el **PID** del proceso creado en el proceso padre.
- **void exit(int status)**: Termina el proceso actual utilizando valor de **status** como valor de retorno.

Funciones Útiles (2)

Pipes

- **int pipe(int descriptores[2])**: Crea **un pipe unidireccional**, el cual tiene un **extremo de escritura**[a], y un **extremo de lectura**[b]. Genera dos descriptores, representando a los extremos de lectura y escritura respectivamente, y los guarda en **descriptores**.
- **int dup2(int oldfd, int newfd)**: si oldfd y newfd son distintos, primero se elimina la referencia al objeto apuntado por newfd, y luego se apunta newfd al mismo objeto que oldfd.

---

[a] Por donde "entra" la información.

[b] Por donde "sale" la información.

Funciones Útiles (3)

Archivos / Descriptores

- **int open(char* path, int flags, ...)**: Abre el archivo indicado por `path`, retornando un *descriptor* que apunta a dicho archivo.
- **int close(int d)**: Cierra *para el proceso actual* el descriptor d pasado por parámetro.
- **int read(int d, void *b, size_t s)**: Lee **s** bytes del archivo apuntado por el descriptor **d**, y los escribe en el buffer **b**.
- **int write(int d, void *b, size_t s)**: Lee **s** bytes del buffer **b**, y los escribe en el archivo apuntado por el descriptor **d**.

Otros

- **printf(char* fmt, ...)**: función **variádica**[a] que toma un string de formato **fmt** y cero o más parámetros adicionales, y escribe el resultado en **stdout**.

[a] Permite un número arbitrario de parámetros.

Funciones Útiles (4)

Ejecutar un archivo

- **int execlp(const char *file, const char *arg, ...)**: Sustituye la imagen de memoria del programa por la del programa ubicado en **file**. El **const char *arg** y la subsiguiente elípsis se pueden pensar como los argumentos **arg0, arg1, ..., argn** del programa a ejecutar. Por convención, **arg0** debería ser el nombre del binario (**file**). La lista de argumentos debe ser terminada con NULL ya que es una función variádica. Ver familia de funciones exec[a].

[a] https://linux.die.net/man/3/execlp

Temario

Ejercicio 1: Los Simonzón

Dados y Doppelgänger...

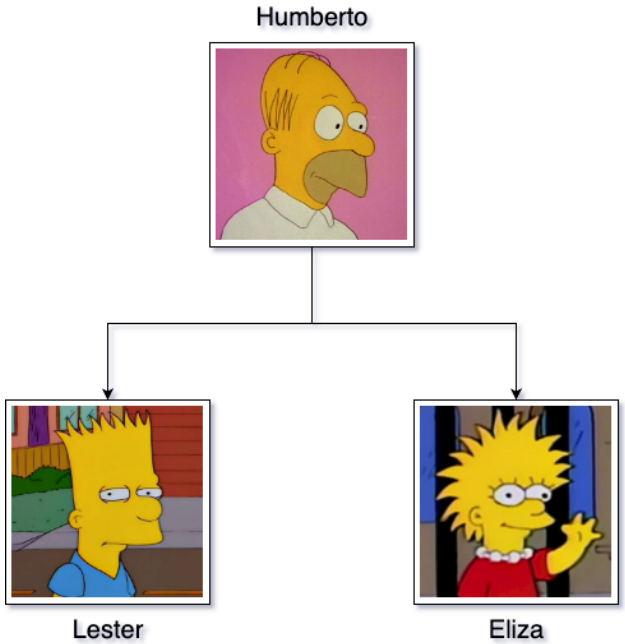
Lester y Eliza están muy aburridos. Para matar el tiempo, su ocurrente padre Humberto les propone un juego:

- Lester y Eliza tiran cada uno un dado, al mismo tiempo.
- Luego le cuentan a Humberto qué valor les dio el dado.
- Finalmente, Humberto se fija cuál es el número más grande y grita bien fuerte el nombre del ganador.

Ejercicio 1: Los Simonzón

Proponer un código en C que represente este juego respetando:

- Cada personaje debe estar representado por un proceso.
- Se debe respetar la genealogía.



- La comunicación entre procesos deberá realizarse con pipes.
- Los anuncios deberán realizarse imprimiendo a `stdout`.

Ejercicio 1: Pseudocódigo

Humberto:

Ejercicio 1: Pseudocódigo

Humberto:

- Crear a Lester.
- Crear a Eliza.

Lester:

Eliza:

Ejercicio 1: Pseudocódigo

Humberto:

- Crear a Lester.
- Crear a Eliza.

Lester:

- Tirar el dado.
- Enviar resultado a Humberto.

Eliza:

Ejercicio 1: Pseudocódigo

Humberto:

- Crear a Lester.
- Crear a Eliza.

Lester:

- Tirar el dado.
- Enviar resultado a Humberto.

Eliza:

- Tirar el dado.
- Enviar resultado a Humberto.

Ejercicio 1: Pseudocódigo

Humberto:

- Crear a Lester.
- Crear a Eliza.
- Recibir número de Lester.
- Recibir número de Eliza.

Lester:

- Tirar el dado.
- Enviar resultado a Humberto.

Eliza:

- Tirar el dado.
- Enviar resultado a Humberto.

Ejercicio 1: Pseudocódigo

Humberto:

- Crear a Lester.
- Crear a Eliza.
- Recibir número de Lester.
- Recibir número de Eliza.
- Anunciar al ganador.

Lester:

- Tirar el dado.
- Enviar resultado a Humberto.

Eliza:

- Tirar el dado.
- Enviar resultado a Humberto.

Ejercicio 1: Resolución

Nos tomamos unos minutos para pensarlo...

...y luego lo resolvemos en pizarrón.

Ejercicio 1: Solución

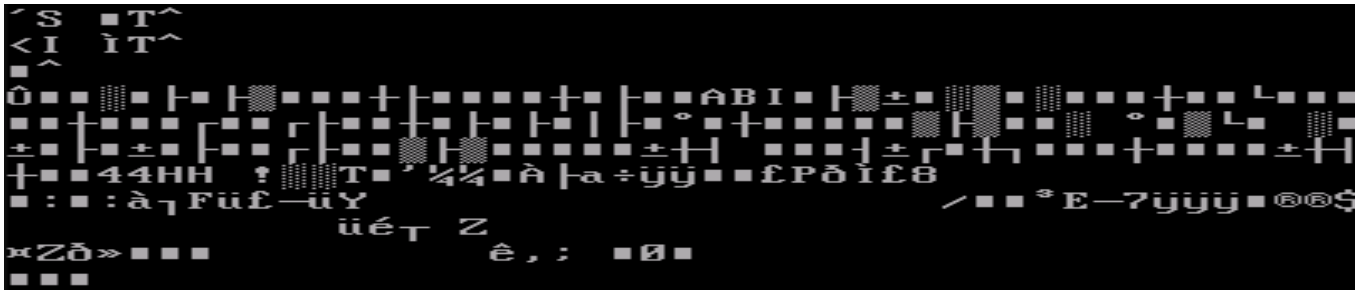
Importante

Todas las soluciones mostradas están simplificadas a efecto de poder incluirlas en la clase. Deberán tener **cuidado** y criterio a la hora de decidir si aplicar estas simplificaciones en sus resoluciones.

Es imprescindible que **lean minuciosamente el manual** para cada función que utilicen.

Temario

Ejercicio 2: Fantasma en el Shell



¡No se que toqué... y ahora no anda más el shell!

Ejercicio 18 de la Práctica 1: "Procesos y APIs del SO"

Escribir el código de un programa que se comporte de la misma manera que la ejecución del comando "**ls -al** | **wc -l**" en un intérprete de comandos.

No está permitido utilizar la función **system()**, y cada uno de los programas involucrados en la ejecución del comando deberá ejecutarse como un subproceso.

Ejercicio 2: Consideraciones preliminares

Primero, veamos qué hacen los comandos:

ls -al	
wc -l	
cmd1 | cmd2	

ls -al | wc -l	

Ejercicio 2: Consideraciones preliminares

Primero, veamos qué hacen los comandos:

ls -al	Lista los archivos del directorio actual.
wc -l	
cmd1 | cmd2	
ls -al | wc -l	

Ejercicio 2: Consideraciones preliminares

Primero, veamos qué hacen los comandos:

ls -al	Lista los archivos del directorio actual.
wc -l	Cuenta la cantidad de líneas del input.
cmd1 | cmd2	

ls -al | wc -l	

Ejercicio 2: Consideraciones preliminares

Primero, veamos qué hacen los comandos:

ls -al	Lista los archivos del directorio actual.
wc -l	Cuenta la cantidad de líneas del input.
cmd1 | cmd2	Ejecuta **cmd1**, y usa su output como input para ejecutar **cmd2**.

ls -al | wc -l	

Ejercicio 2: Consideraciones preliminares

Primero, veamos qué hacen los comandos:

ls -al	Lista los archivos del directorio actual.
wc -l	Cuenta la cantidad de líneas del input.
cmd1 | cmd2	Ejecuta **cmd1**, y usa su output como input para ejecutar **cmd2**.
ls -al | wc -l	El comando completo cuenta la cantidad de archivos en el directorio actual.

Ejercicio 2: Pseudocódigo

Pasos a seguir...

- Creamos el pipe.
- Creamos los subprocesos.
- Queremos que todo lo que el subproceso 1 **escriba a stdout**, el subproceso 2 lo **lea desde stdin**. Para ello, usando dup2:
  - Conectamos **el descriptor de stdout** del subproceso 1 al **extremo de escritura** del pipe.
  - Conectamos **el descriptor de stdin** del subproceso 2 al **extremo de lectura** del pipe.
- Ejecutamos el comando "**ls -al**" en el subproceso 1.
- Ejecutamos el comando "**wc -l**" en el subproceso 2.

Ejercicio 2: Detectando el final del input

Todavía falta algo...

El subproceso 2 tiene que poder **detectar cuando el subproceso 1 terminó de escribir**. ¿Por qué en este caso es necesario esto?

Ejercicio 2: Detectando el final del input

Todavía falta algo...

El subproceso 2 tiene que poder **detectar cuando el subproceso 1 terminó de escribir**. ¿Por qué en este caso es necesario esto?

Y, más importante, ¿cómo lo logramos?

Ejercicio 2: Detectando el final del input

Todavía falta algo...

El subproceso 2 tiene que poder **detectar cuando el subproceso 1 terminó de escribir**. ¿Por qué en este caso es necesario esto?

Y, más importante, ¿cómo lo logramos?

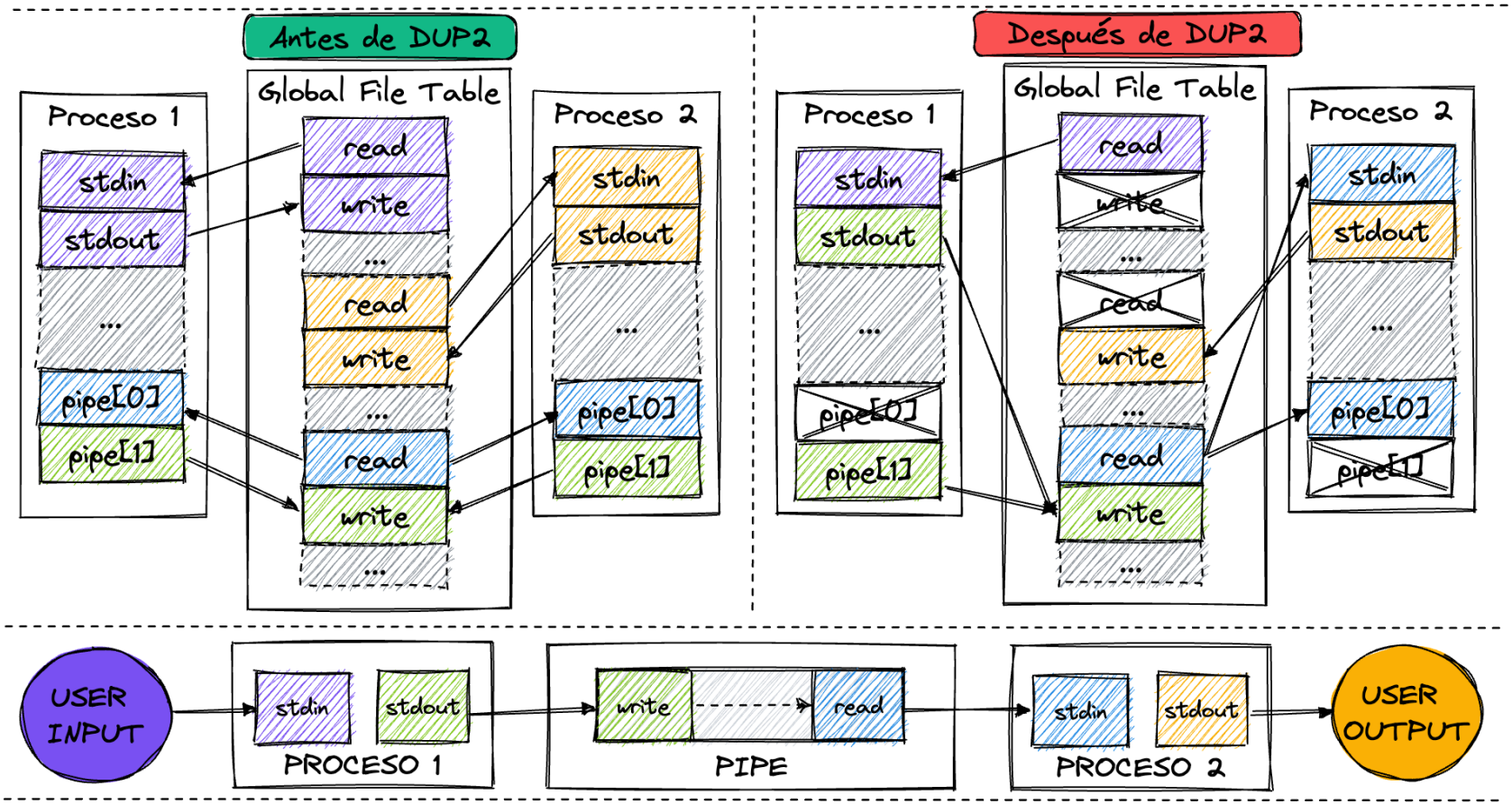
¡Haciendo un buen uso de los pipes!

Debemos cerrar los descriptores no utilizados en cada subproceso.

- El subproceso 1 jamás va a usar el extremo de lectura del pipe.
- El subproceso 2 jamás va a usar el extremo de escritura.

Cerrándolos previamente, nos aseguramos de que cuando el subproceso 1 termine de escribir a *"su stdout"* (que en realidad es el extremo de escritura del pipe), el sistema operativo detecte que **ya no existen más referencias hacia ese descriptor**, retornando un valor de **End-Of-File** cuando el subproceso 2 intente leer *"su stdin"* (que en realidad es el extremo de lectura del pipe).

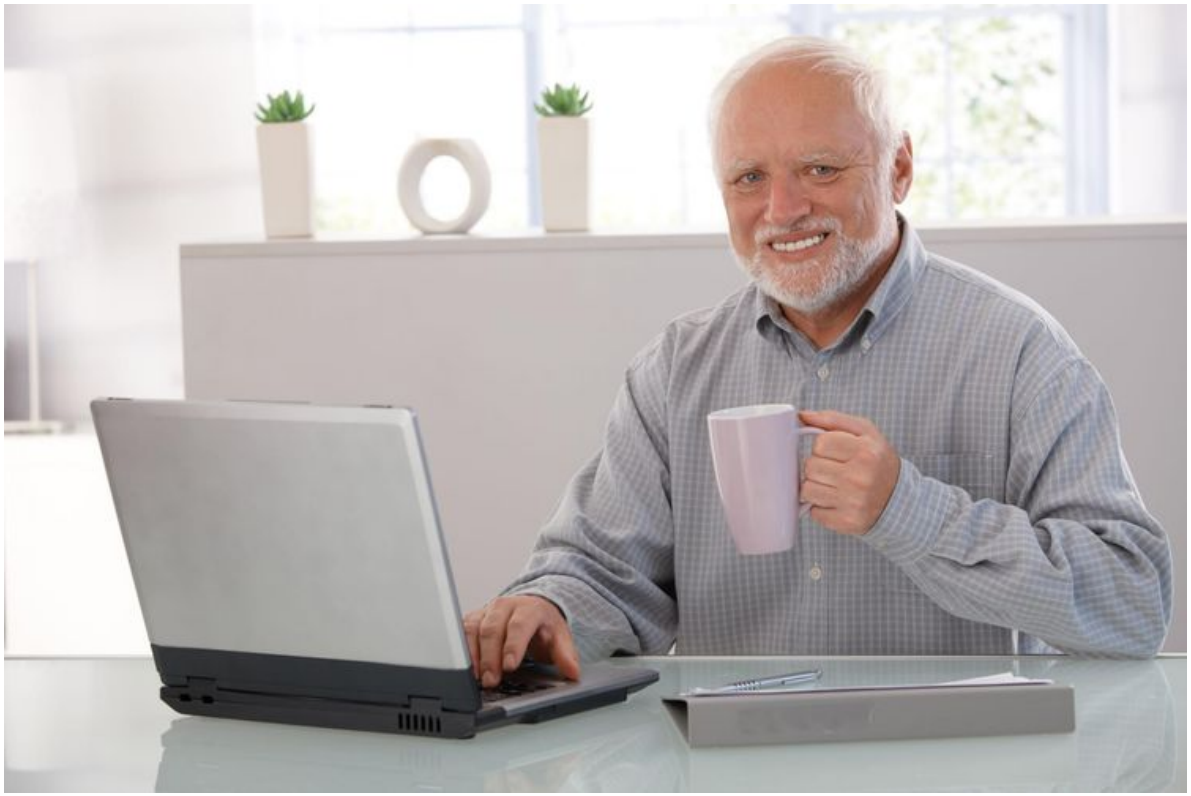
Ejercicio 2: Diagrama dup2



Esquema de comunicación entre los procesos utilizando un pipe y dup2

Ejercicio 2: Resolución

Nos tomamos unos minutos para pensarlo...



...y luego lo resolvemos en pizarrón.

Bonus: Para pensar...

¿Qué tendríamos que **modificar al código** para lograr lo siguiente?

- Encadenar un tercer comando.
  - Por ejemplo: "**ls -al | grep e | wc -l**"
- Que el output se guarde en un archivo.
  - Por ejemplo: "**ls -al | wc -l > file.txt**"

Temario

Ejercicio 3: ¿Cien cabezas piensan mejor que una?

Ejercicio 3: Consigna

Escriba un programa que cuente los números pares en el rango de 2 a mil millones, utilizando para ello una función **bool esPar(long numero)** provista por la cátedra.
El programa dividirá el cálculo de forma equitativa entre varios procesos ejecutándose en paralelo. El número de procesos se especificará como parámetro en la línea de comandos.

Ejercicio 3: Consigna (2)

- El proceso primario creará el número especificado de procesos, y luego dividirá el rango de números en subrangos iguales y asignará un subrango a cada proceso secundario.
- Por ejemplo, si la cantidad de procesos p_i es 10, cada subrango tendrá aproximadamente cien millones de números.
  - p_1 calculará en el rango [2, 100.000.002).
  - p_2 calculará en el rango [100.000.002, 200.000.001).
  - ...
  - p_{10} calculará en el rango [900.000.002, 1.000.000.001).
- El proceso primario informará a cada proceso secundario el subrango que le corresponde usando pipes. Luego, los procesos secundarios deberán enviar sus recuentos al primario, también utilizando pipes.
- Finalmente, el primario deberá sumar todos los recuentos e imprimir el resultado total.

Ejercicio 3: Pseudocódigo

Proceso primario

- Crear los pipes
- Crear a los procesos secundarios
- Calcular e informar los rangos
- Recibir los resultados parciales
- Sumarlos e imprimir el total

Procesos secundarios

- Cerrar pipes
- Recibir los rangos
- Contar los pares
- Informar el resultado

Temario

Recapitulando

Hoy vimos...

- Cómo crear una **topología** de procesos.
- Cómo usar **pipes** para comunicar procesos.
- Cómo usar **dup2** para duplicar un pipe.
- Por qué es importante **usar correctamente** los pipes.
- Para qué sirve la señal de **End-Of-File**.
- Cómo establecer una comunicación bidireccional:
  - Usando un solo pipe por cada par de procesos[a]
  - Usando dos pipes por cada par de procesos.

[a] **¡muy peligroso!**

¿Preguntas?