**Ejercicio 1: Creación de procesos (1 punto)**

**Preguntas para contestar en el informe de prácticas. Responde a cada pregunta en el informe intentando explicar el porqué de tus respuestas.**

1. **¿Comparten el padre y el hijo el mismo código del programa?** Si, se copia el mismo código. Por lo cual tenemos dos procesos en marcha.

**¿Ejecutan el padre y el hijo las mismas líneas del código?** Si, aunque el hijo parte a partir del fork(), ejecutado por el padre ya que el apuntador partirá de ese momento.

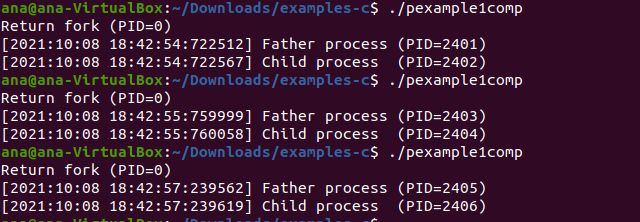
**¿Cómo discriminamos qué partes del código ejecuta cada proceso?** Utilizamos la condición if, ya que el hijo devolverá el valor 0 y el padre un numero mayor a 0. De esta manera en la condición == 0 el hijo pasará y el padre ejecutará la siguiente línea y viceversa.

1. **¿Qué función realiza la función getpid()?** Devuelve el id del proceso

**¿Puede getpid() devolver el mismo resultado a dos procesos distintos?** No, los códigos de los procesos son únicos mientras este se esté ejecutando ya que se pueden reutilizar un vez acabado estos.

**¿Puede devolver cero? Describe qué pruebas has realizado para comprobarlo**.

Devuelve un 0 el valor retornado de la función fork() para identificar si estamos hablando de un padre o de un hijo. El getid() no nos va a devolver un valor 0.



1. **¿Qué sucedería si el hijo llamara a wait()?** La función wait() solo es útil si la llamamos desde el proceso padre ya que espera a que uno de sus procesos hijo finalicé. En el caso de ponerlo por error el SO avisara de que no tiene ningún hijo y seguirá ejecutando (creando así una redundancia de datos innecesarios y alargando el tiempo de ejecución), en el caso de que el hijo fuera padre se realizara el código de manera normal.
2. **¿Es cierto que, dado que el padre crea al hijo con fork(), el padre siempre se ejecuta primero? ¿Cómo lo puedes comprobar?** No, nunca se podrá saber el orden de ejecución de los procesos.

Notas de clase:

Era mas eficiente copiar un bloque entero antes que discriminar, por ello el copiar todo el código con fork, hoy dia casi no se usa fork

Cuando el hijo no obtiene el ok del padre porque el padre se ha muerto anteriormente el proceso hijo se queda en modo zombie.

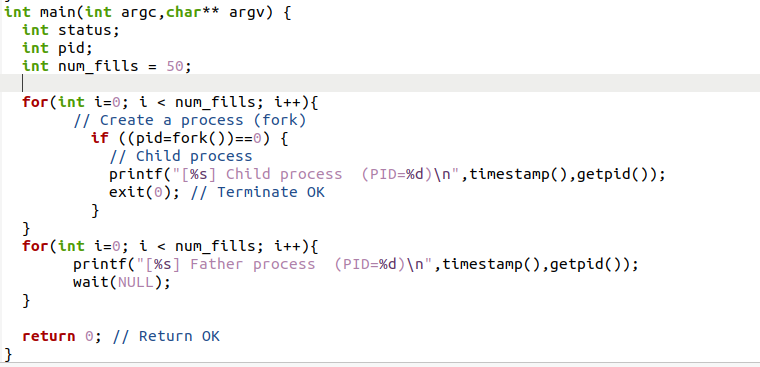
Pipe: tuberies per pasar dades entre pares i fills. Write Pi[1] escriure per pantalla, read p[0] per rebre del teclat. Ls estructura es concretamente write/read (p[algo],circumstancial, nºbytes). Los valores se consumen una vez han sido leidos, tendremos que crear los valores tantas veces como queremos pasarlo.

Si no pasem un valor i el fill esta esperant per llegir un valor, el fill es queda bloquejat esperant

**Código para desarrollar y presentar en el informe con los comentarios necesarios**

**Extender el código anterior para crear 50 procesos hijos. Cada proceso hijo tiene que imprimir su timestamp y terminar. El proceso padre deberá esperar la finalización de todos los procesos hijos y, solo entonces,**

**terminar.**



1. **¿Qué se observa en los resultados al ejecutar múltiples veces el programa?** Podemos observar que no se identifica un orden entre los procesos. Es poco probable que se repita la misma secuencia de ejecución.

f) **¿Genera siempre los mismos resultados?** No, como hemos explicado anteriormente los procesos no tienen un orden lógico establecido por lo cual no generaran siempre los mismos resultados

g) **¿Se ejecutan los procesos siempre en el mismo orden? ¿Cómo lo has comprobado? ¿Por qué pasa esto?** Hemos comprobado el campo timetemp() de cada proceso además de obtener diferentes combinaciones de padre/hijo al ejecutar varias veces el mismo código. Esto pasa porque no podemos establecer un orden entre ellos, ya que el gestor de procesos es el que elije la entrada de estos a la CPU.

**Pregunta 2: Espacio de memoria de procesos y concurrencia (2 puntos)**

**Cuestiones para resolver y describir el resultado obtenido y su justificación en la memoria final.**

**a) ¿Imprimen los dos procesos el mismo número por pantalla? ¿Por qué?** No, porque el valor de la variable “number” se le asigna un valor 400 y al padre un 500.

**b) ¿Qué sucedería si ambas llamadas a printf() se realizaran antes de la asignación a ‘number’? ¿Porqué?** Cambiando los printf() de lugar podemos observar al compilar que la variable “number” devuelve ambos procesos es valor 300 asignado por defecto al inicio de la funcion

**c) ¿Depende el resultado de la ejecución del programa del orden en que los dos procesos asignan valor a la variable ‘number’? ¿Por qué?** Si, ya que como hemos observado nos mostrará el ultimo valor asignado a la variable “number”. En el caso de poner el asignar el valor antes del printf() mostrará el nuevo valor, en caso contrario el valor por defecto.





**d) ¿En qué punto del programa se reserva espacio para cada variable ‘number’?**

En la primera declaración de la variable “number”, al iniciar el proceso.

1. ¿**Es necesario más de una CPU para ejecutar el código anterior? En caso negativo, describe cómo se ejecutarían los dos procesos en un procesador con 1 sola CPU. Por ejemplo, puedes explicar las instrucciones que se van ejecutando de cada proceso.**

No es necesario tener mas de una CPU, ya que los procesos se pueden ir alternando en el procesador.

1. Se declaran las variables (creamos espacio de memoria para estas)
2. Se bifurca el proceso en padre/hijo
3. Se ejecuta un de los dos procesos
4. Se asigna un nuevo valor a la variable “number”
5. Se imprime por pantalla este valor (si es el hijo imprime por pantalla y muere. Si es el padre, espera hasta que acabe el hijo)
6. Fin programa

**Pregunta 3: Comunicación entre procesos. Pipes. (2 puntos)**

**Preguntas:**

**¿Qué sucede si el padre envía el dato antes que el hijo este leyendo desde el otro extremo? ¿Se pierde el dato? ¿Salta un error?** Si ya tenemos un valor para lectura en el pipe el proceso queda bloqueado hasta que exista espacio en el pipe.

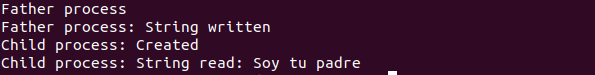
**¿Qué sucede si el hijo quiere leer el dato antes que el padre haya escrito en el otro extremo? ¿El hijo ignorará la llamada a read()?** No, no ignora la llamada read(), sino que el proceso se bloquea hasta que haya algún dato para leer.

¿Qué le sucederá al padre si la pipe está llena? ¿Qué le sucederá al hijo si la pipe está vacía? Se bloqueará el proceso que intenta inserir datos en el pipe. Por otra parte, el otro proceso que intenta leer también porque encontrar una pipe vacía.

¿Qué sucede si intentamos leer de una pipe cuyo otro extremo se ha cerrado? ¿Qué sucede si intentamos escribir en una pipe cuyo otro extremo se ha cerrado? Por políticas se ha de cerrar el extremo/opción que no se utiliza. Si intentamos leer o escribir en un pipe con el otro extremo cerrado el proceso se suspenderá porque en el pipe o no habrá nada a leer o nunca se podrá leer lo escrito.

**Ejercicio:** Extender el código anterior para que el proceso padre le envíe al proceso hijo un buffer de caracteres (string) con el contenido “SOY TU PADRE”.





**Pregunta 4: Sistema de impresión de trabajos (5 puntos)**

**Ejercicio:** Se desea implementar un programa donde 50 procesos ‘Senders’ envíen cada uno 255 trabajos de impresión a un proceso ‘Printer’ a través de una tubería (pipe).

En el informe final de la práctica de este ejercicio debéis explicar y justificar el diseño de:

• Proceso padre y procesos hijos: esquema de creación de los procesos

• Proceso printer y proceso sender: esquema de comunicación

• Funcionalidad de crear trabajos y de recibirlos y procesarlos entre printer y senders

• Esquema de sincronización entre padre e hijos y gestión de la finalización de la ejecución

**ESQUEMA:**Diagrama

Descripción generada automáticamente

Como se puede ver en el esquema, lo primero que haremos para dar solución a este ejercicio será que se creen 51 procesos hijos. 50 de los procesos hijos se encargaran de hacer el trabajo de “Senders” y otro proceso de “Printer”. Estos hijos estarán conectados mediante una Pipe.

Los hijos “Senders” escribirán en el pipe mientras que el hijo “Printer” leerá de este. De esta manera podemos tener un funcionamiento en paralelo de los 51 hijos. Por otro lado, el proceso padre simplemente esperá inicialmente por los 50 hijos “Senders”, después esperará y, previamente, escribirá en el Pipe un -1 para avisar al proceso “Printer” que ha de acabar de leer del pipe.

Para escribir el código de este programa hemos decidido hacer dos funciones. La función “Sender” que se encarga de escribir en el pipe, y la función “Printer” que se encarga de leer del pipe y escribir en pantalla.

**FUNCIÓN SENDER:**

Texto

Descripción generada automáticamente

**FUNCIÓN PRINTER**

Texto

Descripción generada automáticamente

En esta imagen de a continuación podemos ver la función main(), esta será el padre de nuestro programa. Creerá y esperará a los 51 procesos hijos, conjuntamente de detectar errores en la creación de estos. Para este programa hemos visto conveniente parar la ejecución si uno de los hijos da error en su creación. En total podemos ver que se crearán 52 procesos.

**FUNCIÓN MAIN:**

Texto

Descripción generada automáticamente