DOCUMENTACIÓN PRÁCTICA 1

SISTEMAS OPERATIVOS

SEMANA 2:

Esta semana teníamos varios ejercicios los cuales estaban enfocados al aprendizaje y dominio de las funciones fork(), exec() y wait() de control de procesos. El trabajo realizado en esta parte de la práctica nos has servido como toma de contacto para ir familiarizándonos con la ejecución simultanea de código.

Ejercicio 4:

En este ejercicio la parte de código se nos proporcionaba en la documentación de la práctica, a parte modificamos el programa para que cada proceso hijo imprimiese su id y el id de su padre, esto lo hicimos escribiendo las funciones adecuadas en el condicional dedicado exclusivamente a los procesos hijo. El árbol de procesos resultante de este ejercicio es un árbol en el cual el proceso padre lanza 3 hijos, los hijos lanzan 2 nietos y los nietos lanzan un bisnieto.

En cuanto al apartado b de este ejercicio el único cambio es que todos los procesos hacen un wait(). Y por lo tanto todos los procesos acaban siendo recogidos por su respectivo proceso padre y por tanto pueden quedar huérfanos durante un tiempo, pero el wait() asegura que no queden zombis. De todos modos ahora en los Unix modernos cuando un proceso va a quedar zombi es apadrinado por el proceso init (con pid = 1) y este recoge su estado de ejecución. Por tanto, al ejecutar el comando pstree en la terminal tanto en el apartado a como en el b no se aprecia ninguna diferencia en cuanto a procesos huérfanos.

Ejercicio 5:

En este ejercicio hemos realizado algunos cambios, ahora el cuerpo del bucle se ejecuta solamente cuando el iterador es impar. Además, hemos dejado el wait() del final para que como dice el enunciado todos los procesos sean esperados por su padre.

En el apartado b hemos tenido que realiza más cambios, ahora se nos pide que se creen todos los procesos y posteriormente el proceso padre los espere a todos, por ello hemos sacado el wait() fuera del bucle y hacemos un bucle con tantos wait como procesos hijos se han lanzado para asegurarnos de que se espera correctamente a todos los procesos hijos.

Ejercicio 6:

En este ejercicio se nos pide que experimentemos a combinar la reserva de memoria dinámica con la creación de procesos, se nos pide crear memoria dinámica en el proceso padre. Al ser procesos independientes, cado uno con su bloque de control de proceso, entre ellos no tienen ningún tipo de conexión y por tanto a la memoria del proceso hijo no puede acceder el padre y viceversa. En cuanto a la liberación de esta memoria debería hacerse en el proceso padre ya que es este el que ha llamado a la función malloc y por lo tanto debe de ser el mismo que libere dicha memoria.

Ejercicio 8:

En este ejercicio se nos pedía hacer un programa que implementara la ejecución de programas en los procesos hijo mediante el uso de las funciones de la familia de exec.

Estos programas deben ser pasados como argumentos al programa, y el último argumento representa con qué función de la familia de las exec ejecutarlos.

La creación de procesos hijos se hizo de la siguiente forma: el proceso padre crea un hijo. El padre ejecuta el primer programa de los argumentos, y el hijo ejecuta de nuevo el programa principal, con un argumento menos.

Las salidas obtenidas para varios ejemplos son:

```
$ ./ejercicio8 ls df du -l
312     ./html/search
784     ./html
372     ./latex
1480    .
Filesystem            1K-blocks      Used Available Use% Mounted on
none                   2234668    165164   1934388   8% /
tmpfs                 26797712         0  26797712   0% /dev
tmpfs                 26797712         0  26797712   0% /sys/fs/cgroup
/dev/mapper/volg1-lvdata 1238412656 389350236 849046036  32% /mnt
shm                      65536         0     65536   0% /dev/shm
tmpfs                 26797712         0  26797712   0% /sys/firmware
Doxyfile     ejercicio12a.c ejercicio13   ejercicio4a.c ejercicio5a   ejercicio5b.c ejercicio8   ejercicio9.c nombres.txt
Makefile     ejercicio12b   ejercicio13.c ejercicio4b   ejercicio5a.c ejercicio6    ejercicio8.c html
ejercicio12a ejercicio12b.c ejercicio4a   ejercicio4b.c ejercicio5b   ejercicio6.c  ejercicio9   latex
```

```
$ ./ejercicio8 ls df du -lp
312     ./html/search
784     ./html
372     ./latex
1480    .
Filesystem            1K-blocks      Used Available Use% Mounted on
none                   2234668    165164   1934388   8% /
tmpfs                 26797712         0  26797712   0% /dev
tmpfs                 26797712         0  26797712   0% /sys/fs/cgroup
/dev/mapper/volg1-lvdata 1238412656 389350236 849046036  32% /mnt
shm                      65536         0     65536   0% /dev/shm
tmpfs                 26797712         0  26797712   0% /sys/firmware
Doxyfile     ejercicio12a.c ejercicio13   ejercicio4a.c ejercicio5a   ejercicio5b.c ejercicio8   ejercicio9.c nombres.txt
Makefile     ejercicio12b   ejercicio13.c ejercicio4b   ejercicio5a.c ejercicio6    ejercicio8.c html
ejercicio12a ejercicio12b.c ejercicio4a   ejercicio4b.c ejercicio5b   ejercicio6.c  ejercicio9   latex
```

```
$ ./ejercicio8 ls df du -v
312     ./html/search
784     ./html
372     ./latex
1480    .
Filesystem            1K-blocks      Used Available Use% Mounted on
none                   2234668    165164   1934388   8% /
tmpfs                 26797712         0  26797712   0% /dev
tmpfs                 26797712         0  26797712   0% /sys/fs/cgroup
/dev/mapper/volg1-lvdata 1238412656 389350236 849046036  32% /mnt
shm                      65536         0     65536   0% /dev/shm
tmpfs                 26797712         0  26797712   0% /sys/firmware
Doxyfile     ejercicio12a.c  ejercicio13    ejercicio4a.c  ejercicio5a    ejercicio5b.c  ejercicio8    ejercicio9.c  nombres.txt
Makefile     ejercicio12b    ejercicio13.c  ejercicio4b    ejercicio5a.c  ejercicio6     ejercicio8.c  html
ejercicio12a  ejercicio12b.c  ejercicio4a    ejercicio4b.c  ejercicio5b    ejercicio6.c   ejercicio9    latex
```

```
$ ./ejercicio8 ls df du -vp
312     ./html/search
784     ./html
372     ./latex
1480    .
Filesystem            1K-blocks      Used Available Use% Mounted on
none                   2234668    165164   1934388   8% /
tmpfs                 26797712         0  26797712   0% /dev
tmpfs                 26797712         0  26797712   0% /sys/fs/cgroup
/dev/mapper/volg1-lvdata 1238412656 389350236 849046036  32% /mnt
shm                      65536         0     65536   0% /dev/shm
tmpfs                 26797712         0  26797712   0% /sys/firmware
Doxyfile     ejercicio12a.c  ejercicio13    ejercicio4a.c  ejercicio5a    ejercicio5b.c  ejercicio8    ejercicio9.c  nombres.txt
Makefile     ejercicio12b    ejercicio13.c  ejercicio4b    ejercicio5a.c  ejercicio6     ejercicio8.c  html
ejercicio12a  ejercicio12b.c  ejercicio4a    ejercicio4b.c  ejercicio5b    ejercicio6.c   ejercicio9    latex
```

Se puede ver que todas las salidas son exactamente iguales, luego se comprueba que realmente la diferencia entre estas cuatro funciones para llamar a estos programas es cómo se les llama y cómo se les pasa los argumentos, ya que a la hora de ejecución obtenemos lo mismo.

SEMANA 3:

Ejercicio 9:

En este ejercicio debíamos crear un programa que implementara la creación de procesos hijos y comunicación entre ellos con pipes.

Debíamos implementar un programa que creara cuatro procesos hijos, que les enviara a todos un mensaje con dos números enteros, y que esperara un mensaje de retorno de cada hijo con el resultado de una operación aritmética diferente para cada uno.

Este ejercicio nos resultó de los más difíciles de la práctica, puesto que era el primero que implementaba *pipes* y tuvimos bastantes dificultades. Tuvimos problemas al utilizar las mismas variables para todos los procesos, puesto que había procesos hijos que nos devolvían el resultado de otro, al darse una condición de carrera.

Finalmente tuvimos que utilizar variables diferentes para cada hijo, lo que solucionó la mayor parte de los problemas.

Para probar el programa, realizamos varios ejemplos:

```
$ ./ejercicio9 3 2

Datos enviados a través de la tubería por el proceso PID=13489. Operando 1: 3. Operando 2: 2. Potencia: 9

Datos enviados a través de la tubería por el proceso PID=13490. Operando 1: 3. Operando 2: 2. Factorial: 3

Datos enviados a través de la tubería por el proceso PID=13491. Operando 1: 3. Operando 2: 2. Combinatoria: 3

Datos enviados a través de la tubería por el proceso PID=13492. Operando 1: 3. Operando 2: 2. Suma_absolutos: 5
```

```
$ ./ejercicio9 3 -1

Datos enviados a través de la tubería por el proceso PID=13540. Operando 1: 3. Operando 2: -1. Potencia: 0

Datos enviados a través de la tubería por el proceso PID=13541. Operando 1: 3. Operando 2: -1. Factorial: -6


Datos enviados a través de la tubería por el proceso PID=13543. Operando 1: 3. Operando 2: -1. Suma_absolutos:
4
```

Puesto que la combinatoria no está definida para valores negativos, el proceso encargado de ellos no devuelve el valor del cálculo

```
$ ./ejercicio9 5 4

Datos enviados a través de la tubería por el proceso PID=13559. Operando 1: 5. Operando 2: 4. Potencia: 625

Datos enviados a través de la tubería por el proceso PID=13560. Operando 1: 5. Operando 2: 4. Factorial: 30

Datos enviados a través de la tubería por el proceso PID=13561. Operando 1: 5. Operando 2: 4. Combinatoria: 5

Datos enviados a través de la tubería por el proceso PID=13562. Operando 1: 5. Operando 2: 4. Suma_absolutos: 9
```

```
$ ./ejercicio9 3 0
Datos enviados a través de la tubería por el proceso PID=13576. Operando 1: 3. Operando 2: 0. Potencia: 1

Datos enviados a través de la tubería por el proceso PID=13578. Operando 1: 3. Operando 2: 0. Combinatoria: 1
Datos enviados a través de la tubería por el proceso PID=13579. Operando 1: 3. Operando 2: 0. Suma_absolutos: 3
```

En este ejemplo se puede ver que en el hijo que calcula el factorial del operando1 entre el operando2, obtiene una división entre cero, luego no devuelve ningún valor.

SEMANA 4:

Ejercicio 12:

En este ejercicio se nos pedía crear dos programas similares, que realizaran la misma funcionalidad, para comparar la eficiencia de los procesos. Ambos tenían que empezar reservando una estructura que contenía una cadena de 100 caracteres y un entero. Después tenían que crear 100 procesos o 100 hilos, que ejecutaran simultáneamente el cálculo de los N primeros primos.

Para N = 10000 los resultados fueron los siguientes:

- Procesos:

```
$ time ./ejercicio12a 10000
El proceso con PID: 13171 ha calculado 10000 primos
El proceso con PID: 13196 ha calculado 10000 primos
El proceso con PID: 13166 ha calculado 10000 primos
El proceso con PID: 13207 ha calculado 10000 primos
El proceso con PID: 13167 ha calculado 10000 primos
El proceso con PID: 13185 ha calculado 10000 primos
El proceso con PID: 13169 ha calculado 10000 primos
El proceso con PID: 13179 ha calculado 10000 primos
El proceso con PID: 13170 ha calculado 10000 primos
El proceso con PID: 13189 ha calculado 10000 primos
El proceso con PID: 13187 ha calculado 10000 primos
El proceso con PID: 13172 ha calculado 10000 primos
El proceso con PID: 13186 ha calculado 10000 primos
El proceso con PID: 13175 ha calculado 10000 primos
El proceso con PID: 13177 ha calculado 10000 primos
El proceso con PID: 13173 ha calculado 10000 primos
El proceso con PID: 13228 ha calculado 10000 primos
El proceso con PID: 13180 ha calculado 10000 primos
El proceso con PID: 13174 ha calculado 10000 primos
El proceso con PID: 13176 ha calculado 10000 primos
```

```
El proceso con PID: 13238 ha calculado 10000 primos
El proceso con PID: 13258 ha calculado 10000 primos
El proceso con PID: 13259 ha calculado 10000 primos
El proceso con PID: 13256 ha calculado 10000 primos
El proceso con PID: 13205 ha calculado 10000 primos
El proceso con PID: 13202 ha calculado 10000 primos
El proceso con PID: 13198 ha calculado 10000 primos
El proceso con PID: 13168 ha calculado 10000 primos
El proceso con PID: 13261 ha calculado 10000 primos
El proceso con PID: 13239 ha calculado 10000 primos
El proceso con PID: 13241 ha calculado 10000 primos
El proceso con PID: 13206 ha calculado 10000 primos
El proceso con PID: 13240 ha calculado 10000 primos
El proceso con PID: 13200 ha calculado 10000 primos
El proceso con PID: 13197 ha calculado 10000 primos
El proceso con PID: 13209 ha calculado 10000 primos
El proceso con PID: 13199 ha calculado 10000 primos
El proceso con PID: 13204 ha calculado 10000 primos
El proceso con PID: 13203 ha calculado 10000 primos
El proceso con PID: 13194 ha calculado 10000 primos
El proceso con PID: 13245 ha calculado 10000 primos
El proceso con PID: 13246 ha calculado 10000 primos
El proceso con PID: 13234 ha calculado 10000 primos
El proceso con PID: 13201 ha calculado 10000 primos
El proceso con PID: 13211 ha calculado 10000 primos
El proceso con PID: 13219 ha calculado 10000 primos
El proceso con PID: 13212 ha calculado 10000 primos
El proceso con PID: 13192 ha calculado 10000 primos
El proceso con PID: 13208 ha calculado 10000 primos
El proceso con PID: 13193 ha calculado 10000 primos
El proceso con PID: 13190 ha calculado 10000 primos
El proceso con PID: 13188 ha calculado 10000 primos
El proceso con PID: 13191 ha calculado 10000 primos
El proceso con PID: 13195 ha calculado 10000 primos
El programa ha tardado 0.014321 segundos en realizar las
operaciones con N = 10000

real    0m2.003s
user    0m2.400s
sys     0m0.028s
```

Se puede ver que los valores obtenidos con clock() no se corresponden con los obtenidos mediante time. Tras ejecutar tres veces el mismo programa, y obtener valores similares de tiempo, el ejercicio 12a tarda unos 2,4 segundos.

- Hilos:

```
$ ./ejercicio12b 10000

El programa ha tardado 2.633411 segundos en realizar las operaciones con N = 10000
```

Comparando ambos resultados, podemos ver que tardan tiempos similares, pero con procesos es ligeramente más rápido. Opinamos que esto se puede deber, a pesar de que para cada proceso que se crea tiene que volver a reservar la estructura de 100 caracteres y un entero, a la forma que tiene el procesador de ejecutar los hilos

Estos tiempos fueron leídos mediante la función clock() para el de hilos, y con time en el otro, puesto que la creación de procesos hijos en el programa reinicia el reloj de clock().

Ejercicio 13:

En este ejercicio se nos pedía implementar comunicaciones entre dos hilos, en los cuales se multiplicaban dos matrices diferentes simultáneamente, y ver cómo se iban ejecutando simultáneamente.

Este ejercicio no tenía mucha complicación, puesto que lo primero que se hacía era leer los valores por consola, y luego crear dos hilos que ejecutaban una misma función con diferentes parámetros, y luego hacerles “join”.

Tuvimos que implementar una estructura Param para poder pasar más de un parámetro a las funciones, puesto que cuando se crea un hilo solo se puede pasar un puntero a void. Nuestra implementación fue la siguiente:

```
typedef struct param {
   int **matriz;
   int mult;
   int dim;
   int id;
} Param;
```

Esta estructura contiene un array bidimensional de enteros (la matriz a multiplicar), el número por el que multiplicar, la dimensión de las matrices (necesaria porque es variable) y un identificador para diferenciar los mensajes que escribe cada proceso por la terminal.

La función auxiliar que programamos para que ejecutaran los hilos no tiene tampoco mucha complicación, puesto que son dos bucles anidados que multiplican cada elemento de la matriz, y según terminan una fila imprimen el resultado por pantalla.