PRÁCTICA 1 – SOPER
Pareja 1
Alba Ramos
Andrea Salcedo

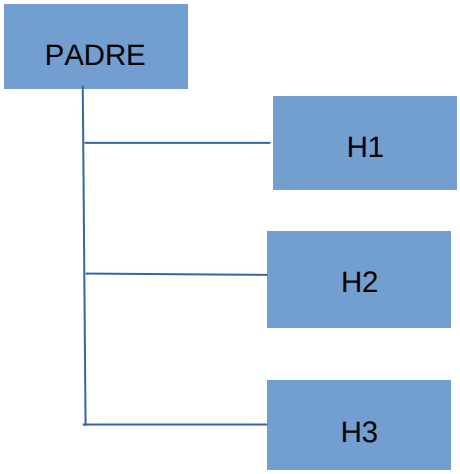
Todos los ejercicios que hay que entregar se compilan haciendo make nombre_archivo (este nombre se indica al final de cada apartado). También podemos compilar todos a la vez haciendo make.
Para ejecutarlos, se escribe ./nombre_archivo.

EJERCICIO 3

a) No se puede saber, porque al momento de tener dos procesos, el S.O. se encarga de decidir cual ejecutar primero, ya sea por un criterio de prioridad o por otras razones. Unas veces será el padre y otras el hijo.

b) Se ha sustituido la sentencia printf("HIJO %d \n", i); por dos printf(): uno en el que se imprime getpid(), que es el id del hijo, y otro en el que se hace una llamada a getppid() para obtener el id del padre. También hemos impreso en el padre su id mediante getpid(), para comprobar. (***p1_exercise_3.c***).

c)



El padre es el que va repitiendo la ejecución del bucle, ya que cuando los hijos van entrando a sus if, acaban su ejecución con exit.

EJERCICIO 4

a) Ocurre porque no hay tantos wait para que el padre espere a que termine la ejecución de todos los hijos en el caso de que el padre se esté ejecutando primero. Entonces podría darse el caso de que el padre termine primero el bucle, y los hijos después, y como fuera del bucle hay un solo wait y un exit, el padre sólo esperaría por un hijo, y terminaría, quedando el resto de los hijos huérfanos.

b) Al finalizar el bucle en el que se realizan las llamadas a fork, hemos añadido otro bucle que realice wait NUM_PROC veces, ya que el padre genera NUM_PROC hijos y tiene que esperarlos a todos. Estos cambios están ya realizados en el fichero ***p1_exercise_3.c***, porque pensábamos que había que modificar el ejercicio 3.

c) En el if() de los padres, estos esperan con wait a la señal de terminación de su hijo y después terminan su ejecución con exit. De esta forma, es el hijo el que sigue repitiendo la ejecución del bucle y creando otro hijo, repitiendo el proceso anterior (convirtiéndose este ahora en el padre). (***ejercicio4.c***)

EJERCICIO 5

a) No es correcto porque solo se imprime “Padre:”. Esto ocurre porque cuando llegamos al fork, el padre crea un hijo que recibe una copia de las variables del padre, pero no son las mismas (es por esto por lo que podemos saber que el pid del hijo es 0, y que el pid del padre es otro). Entonces, el hijo está escribiendo “hola” en su variable sentence, no en la del padre.

b) Se debe liberar la memoria tanto en el hijo como en el padre, ya que, al crear al hijo, se crea una copia de las variables del padre, por tanto si solo liberásemos en uno, quedaría todavía la otra copia.
Hemos comentado la línea strcpy(sentence, "hola"); en el if() del padre, porque al pasar valgrind nos daba un error y era porque esta variable no se había inicializado en el padre, pero no nos especifican que tengamos que cambiar esto en el enunciado.
(***p1_exercise_5.c***)

EJERCICIO 7

En primer lugar, creamos un proceso hijo usando fork y en él hacemos la llamada a execv, mientras el padre espera a que el hijo acabe su ejecución. A esta función le pasamos como primer argumento el path del comando cat, que sabemos que se encuentra en el directorio /bin/. Como segundo argumento, recibe el vector que almacena los nombres de los ficheros introducidos por el usuario.
Finalmente, implementamos la función showAllFiles. En ella, creamos un nuevo proceso hijo utilizando fork. Este hijo realiza la llamada a la función execlp, y el padre espera a que termine. Esta función recibe una serie de parámetros. El primero es el nombre del comando a ejecutar: “ls”. A continuación, los parámetros consisten en los argumentos del comando (pero el primero de ellos deben coincidir con el nombre del comando). Finalmente, un NULL casteado a char* nos indica el final de estos argumentos. (***p1_exercise_7.c***)

EJERCICIO 9

En este programa, el proceso padre genera dos hijos. Uno de ellos genera un numero aleatorio utilizando rand, lo envia a través del pipe y lo imprime. El padre lo lee a través del pipe, y se lo envía al otro hijo utilizando otro pipe. Este hijo lo lee y lo imprime.
En nuestro programa, lo que leemos a través del pipe lo almacenamos en una variable de tipo entero, porque sabemos que va a ser un entero, y que lo que ocupa es sizeof(int). Es

por esto por lo que no usamos un buffer de lectura, como se hacía en el ejemplo de uso.
El programa tiene en cuenta control de errores al crear hijos y pipes, el cierre de los respectivos canales del descriptor de ficheros tanto para escribir como para leer y la espera del padre por sus hijos, para no dejar ninguno huérfano. (***p1_exercise_9.c***)

EJERCICIO 12

El programa crea e inicializa con valores de 0 a N-1 el campo "entrada" de los elementos de un array de tipo Params. Params es una estructura que hemos creado para que la función llamada al crear los hilos pueda recibir varios argumentos. La estructura consta de los campos “entrada” y “salida”, ambos de tipo double. También crea N hilos utilizando la función pthread_create. Estos hilos llaman a la funcion calculoPotencia pasando como parametro cada elemento del array de Params. Este parámetro hay que castearlo a void*.
La funcion calculoPotencia realiza el cálculo de 2 elevado a "entrada", y el resultado lo guarda en el campo "salida" del elemento de tipo Params.
El programa principal espera por todos los hilos al final utilizando un bucle que se realiza N veces llamando a la función pthread_join, e imprime las potencias calculadas.
(***ejercicio12.c***)