05/03/2018

SISTEMAS OPERATIVOS

Práctica 1

Lucía Fuentes

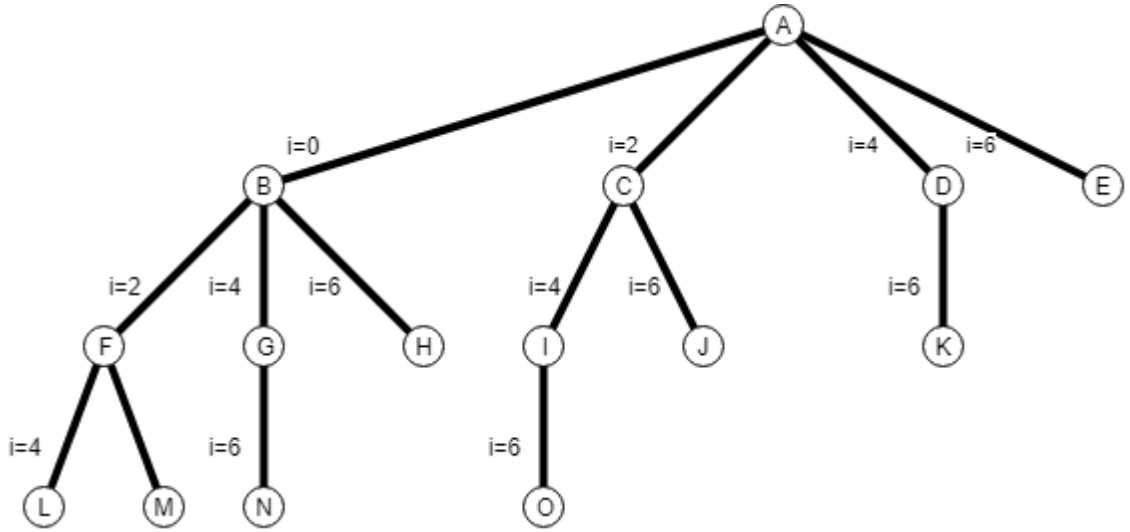
Mihai Blidaru

Pareja 9

Ejercicio 4

a) Analiza el árbol de procesos vinculado al siguiente código:

Analizando el código podemos observar que se genera el siguiente árbol de procesos:



Cada proceso genera tantos hijos como múltiplos de 2 que hay entre la variable interna i de cada uno y 6 (inclusive). Por ejemplo, el proceso padre **A** crea 4 hijos para $i \in \{0, 2, 4, 6\}$. Su primer hijo, **B,** crea 3 hijos para los 3 múltiplos restantes hasta 6 (2, 4, 6). Su segundo hijo, **C**, solo crea 2 hijos $i \in \{4, 6\}$ ya que en el momento de su creación i = 2. Así hasta llegar a tener un total de 16 procesos.

Para confirmar, hemos usado ***pstree*** para imprimir el árbol de procesos generado. Hemos usado la función ***sleep*** para pausar todos los procesos y así poder capturar el árbol con ***pstree***.

```
tmux─┬─bash───bash───ejercicio4a─┬─ejercicio4a─┬─ejercicio4a─┬─ejercicio4a───ejercicio4a
     │                           │             │             └─ejercicio4a
     │                           │             ├─ejercicio4a───ejercicio4a
     │                           │             └─ejercicio4a
     │                           ├─ejercicio4a─┬─ejercicio4a───ejercicio4a
     │                           │             └─ejercicio4a
     │                           ├─ejercicio4a───ejercicio4a
     │                           └─ejercicio4a
     └─bash───bash───pstree
```

b) Explica la diferencia entre el código anterior y el siguiente:

El código del ejercicio4b.c contiene una instrucción ***wait(NULL)*** que hace que el proceso se bloquee hasta que acaba uno de sus hijos. Si el proceso no tiene hijos, no se bloquea y ejecuta la función ***exit***.

¿Existen procesos huérfanos en alguno de los dos programas analizados?

Tanto en la versión ejercicio4a.c como en la versión ejercicio4b.c existe la posibilidad de que haya procesos huérfanos. En el ejercicio4a.c, no hay una llamada a **wait** o **waitpid** por lo que ningún proceso espera a que acaben sus hijos. De esta forma, la ejecución de procesos no es controlada por el programa y no se puede garantizar que no haya huérfanos:

```
mihai314:~/workspace/practica1 (master) $ ./ejercicio4a
Soy padre, mi PID = 99510        PID del ultimo hijo creado = 99511
Soy hijo, mi PID = 99511         PID del padre = 99510
Soy padre, mi PID = 99510        PID del ultimo hijo creado = 99512
Soy padre, mi PID = 99510        PID del ultimo hijo creado = 99514
Soy padre, mi PID = 99511        PID del ultimo hijo creado = 99513
Soy hijo, mi PID = 99512         PID del padre = 99510
Soy padre, mi PID = 99511        PID del ultimo hijo creado = 99515
Soy padre, mi PID = 99510        PID del ultimo hijo creado = 99516
Soy padre, mi PID = 99512        PID del ultimo hijo creado = 99517
Soy padre, mi PID = 99511        PID del ultimo hijo creado = 99518
Soy padre, mi PID = 99512        PID del ultimo hijo creado = 99519
Soy hijo, mi PID = 99513         PID del padre = 99511
Soy padre, mi PID = 99513        PID del ultimo hijo creado = 99520
Soy hijo, mi PID = 99516         PID del padre = 1
Soy padre, mi PID = 99513        PID del ultimo hijo creado = 99521
Soy hijo, mi PID = 99520         PID del padre = 99513
Soy hijo, mi PID = 99521         PID del padre = 99513
Soy hijo, mi PID = 99514         PID del padre = 1
Soy padre, mi PID = 99520        PID del ultimo hijo creado = 99522
Soy padre, mi PID = 99514        PID del ultimo hijo creado = 99523
Soy hijo, mi PID = 99523         PID del padre = 1
Soy hijo, mi PID = 99515         PID del padre = 1
Soy padre, mi PID = 99515        PID del ultimo hijo creado = 99525
Soy hijo, mi PID = 99517         PID del padre = 1
Soy hijo, mi PID = 99525         PID del padre = 99515
Soy hijo, mi PID = 99522         PID del padre = 1
Soy padre, mi PID = 99517        PID del ultimo hijo creado = 99526
Soy hijo, mi PID = 99518         PID del padre = 1
Soy hijo, mi PID = 99526         PID del padre = 1
Soy hijo, mi PID = 99519         PID del padre = 1
```

En el ejercicio4b, aunque hay una llamada a ***wait***, solo espera a que acabe uno de los hijos. Por tanto, también se da la posibilidad de que haya procesos huérfanos:

```
mihai314:~/workspace/practica1 (master) $ ./ejercicio4b
Soy padre, mi PID = 99544        PID del ultimo hijo creado = 99545
Soy padre, mi PID = 99544        PID del ultimo hijo creado = 99546
Soy padre, mi PID = 99544        PID del ultimo hijo creado = 99547
Soy hijo, mi PID = 99545         PID del padre = 99544
Soy padre, mi PID = 99544        PID del ultimo hijo creado = 99548
Soy padre, mi PID = 99545        PID del ultimo hijo creado = 99549
Soy padre, mi PID = 99545        PID del ultimo hijo creado = 99550
Soy padre, mi PID = 99545        PID del ultimo hijo creado = 99551
Soy hijo, mi PID = 99551         PID del padre = 99545
Soy hijo, mi PID = 99550         PID del padre = 99545
Soy padre, mi PID = 99550        PID del ultimo hijo creado = 99552
Soy hijo, mi PID = 99549         PID del padre = 99545
Soy padre, mi PID = 99549        PID del ultimo hijo creado = 99553
Soy hijo, mi PID = 99553         PID del padre = 99549
Soy hijo, mi PID = 99546         PID del padre = 1
Soy padre, mi PID = 99546        PID del ultimo hijo creado = 99554
Soy padre, mi PID = 99553        PID del ultimo hijo creado = 99555
Soy padre, mi PID = 99546        PID del ultimo hijo creado = 99557
Soy padre, mi PID = 99549        PID del ultimo hijo creado = 99556
Soy hijo, mi PID = 99557         PID del padre = 99546
Soy hijo, mi PID = 99547         PID del padre = 1
Soy hijo, mi PID = 99554         PID del padre = 99546
Soy hijo, mi PID = 99555         PID del padre = 99553
Soy padre, mi PID = 99547        PID del ultimo hijo creado = 99558
Soy padre, mi PID = 99554        PID del ultimo hijo creado = 99559
Soy hijo, mi PID = 99552         PID del padre = 99550
Soy hijo, mi PID = 99558         PID del padre = 99547
Soy hijo, mi PID = 99556         PID del padre = 99549
Soy hijo, mi PID = 99559         PID del padre = 99554
Soy hijo, mi PID = 99548         PID del padre = 1
```

Ejercicio 5

a) Introduce el mínimo número de cambios en el código del segundo programa del ejercicio de forma que se generen un conjunto de procesos de modo secuencial para i % 2 != 0 (cada proceso tiene un único hijo y ha de esperar a que concluya la ejecución de su proceso hijo). Todos los cambios introducidos han de explicarse adecuadamente.

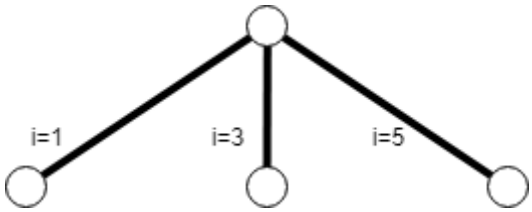
El primer padre crea un hijo, y se queda esperándolo, ese hijo crea a su vez otro y le espera y así hasta que el bucle llega a su fin. Cuando un hijo acaba, el padre ejecuta un **break** para salir del bucle y no crear más hijos.

Si analizamos el árbol de procesos con **pstree** podemos ver que nuestro programa cumple los requisitos del enunciado:

```
tmux─┬─bash───bash───ejercicio5a───ejercicio5a───ejercicio5a───ejercicio5a
     └─bash───bash
```

b) Introduce el mínimo número de cambios en el código del segundo programa del ejercicio anterior de forma que exista un único proceso padre que dar lugar a un conjunto de procesos hijo para i % 2 != 0. El proceso padre ha de esperar a que termine la ejecución de todos sus procesos hijo. Todos los cambios introducidos han de explicarse convenientemente.

En este ejercicio, para hacer que solo el padre cree hijos, dentro del hijo hemos usado una instrucción **break** para que estos a su vez no creen otros hijos y acabe su ejecución. El padre después de crear un hijo usa un **wait(NULL)** para esperarle. Cuando el hijo acaba, vuelve a ejecutar la misma secuencia para crear otro hijo hasta que el padre sale del bucle.



Dada la forma en la que se crean los hijos, en este caso **pstree** no ofrece información relevante sobre el árbol de procesos ya que nunca existen más de 2 procesos a la vez.

Ejercicio 6

El proceso padre no tiene acceso a la memoria del hijo. Inicialmente el hijo y el padre comparten recursos. Cuando el hijo intenta modificar la estructura se crea una copia (Copy-On-Write) a la que solo puede acceder el hijo.

Como los dos procesos no comparten la misma memoria, hay que liberar memoria en los dos procesos, al cómo se puede observar usando **valgrind**:

```
/* FREE SOLO EN EL PADRE */

mihai314:~/workspace/practica1 (master) $ valgrind --leak-check=full --show-
leak-kinds=all --trace-children=yes ./ejercicio6
==2418== Memcheck, a memory error detector
==2418== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==2418== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==2418== Command: ./ej6
==2418==
Inserte un nombre por teclado: qwerty
HIJO: Cadena = qwerty

==2419==
==2419== HEAP SUMMARY:
==2419==     in use at exit: 88 bytes in 1 blocks
==2419==   total heap usage: 1 allocs, 0 frees, 88 bytes allocated
==2419==
==2419== 88 bytes in 1 blocks are still reachable in loss record 1 of 1
==2419==    at 0x4C2CC70: calloc (in /usr/lib/valgrind/vgpreload_memcheck-
amd64-linux.so)
==2419==    by 0x4007C4: main (in /home/ubuntu/workspace/practica1/ej6)
==2419==
==2419== LEAK SUMMARY:
==2419==    definitely lost: 0 bytes in 0 blocks
==2419==    indirectly lost: 0 bytes in 0 blocks
==2419==      possibly lost: 0 bytes in 0 blocks
==2419==    still reachable: 88 bytes in 1 blocks
==2419==         suppressed: 0 bytes in 0 blocks
==2419==
==2419== For counts of detected and suppressed errors, rerun with: -v
==2419== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
PADRE: Cadena = Contenido inicial que deberia ser sobreescrito por el hijo si
comparten memoria
==2418==
==2418== HEAP SUMMARY:
==2418==     in use at exit: 0 bytes in 0 blocks
==2418==   total heap usage: 1 allocs, 1 frees, 88 bytes allocated
==2418==
==2418== All heap blocks were freed -- no leaks are possible
==2418==
==2418== For counts of detected and suppressed errors, rerun with: -v
==2418== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

```
/* FREE SOLO EN EL HIJO */

mihai314:~/workspace/practica1 (master) $ valgrind --leak-check=full --show-
leak-kinds=all --trace-children=yes ./ejercicio6
==2449== Memcheck, a memory error detector
==2449== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==2449== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==2449== Command: ./ej6
==2449==
Inserte un nombre por teclado: QWERTY2
HIJO: Cadena = QWERTY2

==2450==
==2450== HEAP SUMMARY:
==2450==     in use at exit: 0 bytes in 0 blocks
==2450==   total heap usage: 1 allocs, 1 frees, 88 bytes allocated
==2450==
==2450== All heap blocks were freed -- no leaks are possible
==2450==
==2450== For counts of detected and suppressed errors, rerun with: -v
==2450== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
PADRE: Cadena = Contenido inicial que deberia ser sobreescrito por el hijo si
comparten memoria
==2449==
==2449== HEAP SUMMARY:
==2449==     in use at exit: 88 bytes in 1 blocks
==2449==   total heap usage: 1 allocs, 0 frees, 88 bytes allocated
==2449==
==2449== 88 bytes in 1 blocks are still reachable in loss record 1 of 1
==2449==    at 0x4C2CC70: calloc (in /usr/lib/valgrind/vgpreload_memcheck-
amd64-linux.so)
==2449==    by 0x4007C4: main (in /home/ubuntu/workspace/practica1/ej6)
==2449==
==2449== LEAK SUMMARY:
==2449==    definitely lost: 0 bytes in 0 blocks
==2449==    indirectly lost: 0 bytes in 0 blocks
==2449==      possibly lost: 0 bytes in 0 blocks
==2449==    still reachable: 88 bytes in 1 blocks
==2449==         suppressed: 0 bytes in 0 blocks
==2449==
==2449== For counts of detected and suppressed errors, rerun with: -v
==2449== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

```
/*   FREE EN LOS DOS    */

mihai314:~/workspace/practica1 (master) $ valgrind --leak-check=full --show-
leak-kinds=all --trace-children=yes ./ejercicio6
==2477== Memcheck, a memory error detector
==2477== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==2477== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==2477== Command: ./ej6
==2477==
Inserte un nombre por teclado: QWERTY3
HIJO: Cadena = QWERTY3

==2478==
==2478== HEAP SUMMARY:
==2478==     in use at exit: 0 bytes in 0 blocks
==2478==   total heap usage: 1 allocs, 1 frees, 88 bytes allocated
==2478==
==2478== All heap blocks were freed -- no leaks are possible
==2478==
==2478== For counts of detected and suppressed errors, rerun with: -v
==2478== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
PADRE: Cadena = Contenido inicial que deberia ser sobreescrito por el hijo si
comparten memoria
==2477==
==2477== HEAP SUMMARY:
==2477==     in use at exit: 0 bytes in 0 blocks
==2477==   total heap usage: 1 allocs, 1 frees, 88 bytes allocated
==2477==
==2477== All heap blocks were freed -- no leaks are possible
==2477==
==2477== For counts of detected and suppressed errors, rerun with: -v
==2477== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Ejercicio 8

Nuestro programa permite recibir comandos no soportados por argumento. Si se da ese caso, no se crea un proceso hijo y tampoco se ejecuta una función **exec.** Por ejemplo, si se ejecuta de la siguiente forma:

```
./ejercicio8 ls pstree ls -l
```

El programa ejecuta el comando **ls**, en **pstree** imprime un error indicando que no está soportado y ejecuta el segundo **ls**.

En las funciones **exec** que usan el PATH, se les pasa directamente lo que el usuario ha metido como argumento (después de comprobar que está dentro de los comandos soportados), mientras que para las otras se les pasa la ruta completa (funciona solo para los tres ejecutables especificados en el enunciado).

Ejercicio 9

En el ejercicio 9 creamos 2 pipes (una para cada dirección) que usamos con los 4 procesos hijos. Para las operaciones de potencia y suma de valores absolutos no se comprueban los parámetros ya que estas dos funciones matemáticas pueden trabajar con cualquier número real.

En el caso de la operación op1!/op2, no se puede calcular si alguno de estos dos operandos es negativo o si op2 = 0. En el caso de las permutaciones op1 ≥ op2 ≥ 0.

En los dos casos donde pueden surgir errores, estás se comprueban después de intentar hacer los cálculos dentro del código que ejecutan los hijos. En caso de error, el hijo manda al padre un mensaje de error:

```
mihai314:~/workspace/practica1 (master) $ ./ejercicio9 -3 -5
Datos enviados a traves de la tubería por el proceso PID=16511. Operando 1:-3 Operando 2:-5
    1.Potencia:  -0.004115

Datos enviados a traves de la tubería por el proceso PID=16512. Operando 1:-3 Operando 2:-5
    2.Factorial:  No se puede calcular con los operandos -3 -5

Datos enviados a traves de la tubería por el proceso PID=16513. Operando 1:-3 Operando 2:-5
    3.Permutaciones:  No se puede calcular con los operandos -3 -5

Datos enviados a traves de la tubería por el proceso PID=16514. Operando 1:-3 Operando 2:-5
    4.Valor absoluto:  8.000000
```

Pruebas: Hemos probado la correcta funcionalidad del programa usando diferentes entradas:

- **0 y 0**

```
mihai314:~/workspace/practica1 (master) $ ./ejercicio9 0 0
Datos enviados a traves de la tubería por el proceso PID=5895. Operando 1:0 Opera
ndo 2:0 1.Potencia:  1.000000

Datos enviados a traves de la tubería por el proceso PID=5896. Operando 1:0 Opera
ndo 2:0 2.Factorial:  No se puede calcular con los operandos 0 0

Datos enviados a traves de la tubería por el proceso PID=5897. Operando 1:0 Opera
ndo 2:0 3.Permutaciones:  1.000000

Datos enviados a traves de la tubería por el proceso PID=5898. Operando 1:0 Opera
ndo 2:0 4.Valor absoluto:  0.000000
```

- **3 y -6**

```
mihai314:~/workspace/practica1 (master) $ ./ejercicio9 3 -6

Datos enviados a traves de la tubería por el proceso PID=5886. Operando 1:3 Opera
ndo 2:-6     1.Potencia:  0.001372

Datos enviados a traves de la tubería por el proceso PID=5887. Operando 1:3 Opera
ndo 2:-6     2.Factorial:  -1.000000

Datos enviados a traves de la tubería por el proceso PID=5888. Operando 1:3 Opera
ndo 2:-6     3.Permutaciones:  No se puede calcular con los operandos 3 -6

Datos enviados a traves de la tubería por el proceso PID=5889. Operando 1:3 Opera
ndo 2:-6     4.Valor absoluto:  9.000000
```

- **10 y 3**

```
mihai314:~/workspace/practica1 (master) $ ./ejercicio9 10 3
Datos enviados a traves de la tubería por el proceso PID=5842. Operando 1:10 Operando 2:3
 1.Potencia:  1000.000000

Datos enviados a traves de la tubería por el proceso PID=5843. Operando 1:10 Operando 2:3
 2.Factorial:  1209600.000000

Datos enviados a traves de la tubería por el proceso PID=5844. Operando 1:10 Operando 2:3
 3.Permutaciones:  720.000000

Datos enviados a traves de la tubería por el proceso PID=5845. Operando 1:10 Operando 2:3
 4.Valor absoluto:  13.000000
```

Como podemos ver, el programa no ejecuta operaciones con argumentos inválidos y los resultados matemáticos son correctos.

Ejercicio 12

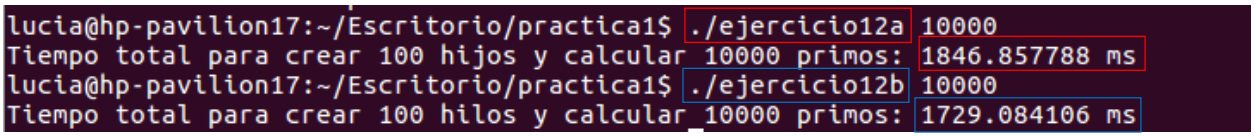
Este ejercicio está dividido en dos apartados.

El primero, ejercicio 12a, se mide el tiempo que tarda un proceso padre en crear 100 procesos hijos, cada uno de esos hijos debe calcular el número de primos que pasemos como argumento desde 1. Para ello guardamos el tiempo antes de iniciar las llamadas a fork. Dentro de cada hijo (cuando fork devuelve 0), llamamos a la función calcular_primos, que utiliza la función esPrimo (Boolean) para cada número de 1 a N hasta que llega al límite. Cuando finaliza la ejecución de todos los hijos, registramos el tiempo de finalización y con ello calculamos el tiempo (en ms) que ha tardado.

El ejercicio12b consiste en lo mismo, pero utilizando hilos. Es un poco diferente al ejercicio anterior porque la función que realizan los hilos se incluye en su rutina de creación, por lo tanto, hemos tenido que modificar la rutina calcular_primos, incluyendo el exit en la propia función, y hemos tenido que modificar el paso de parámetros para poder introducir el número límite de primos. Para ello hemos utilizado la estructura que hemos creado, donde incluimos el número limite al que accedemos desde calcular_primos con un puntero.

El tiempo que tarda el ejercicio12b, el que usa hilos, es ligeramente inferior al del ejercicio12a, el que usa procesos hijo, porque al hacer fork() y wait() para crear hijos se crean y se destruye el espacio de memoria que ocupan los hijos, mientras que los hilos, que comparten el mismo espacio de memoria por cada proceso, no necesitan tiempo extra para la creación de este espacio.

Para medir los tiempos adecuadamente hemos tenido que hacer uso de la función clock_gettime(CLOCK_REALTIME, &tiempo).



Ejercicio 13

En este ejercicio vamos a multiplicar dos números por dos matrices cuadradas que introducimos por teclado. Cada matriz se multiplicará por el número introducido mediante un hilo de ejecución, con un ligero retardo para que se pueda apreciar que los hilos se ejecutan concurrentemente. Como los hilos incluyen la función que van a utilizar en su proceso de creación, hemos creado la rutina MultiplicarMatriz donde se multiplica cada fila de la matriz por el multiplicador, se imprime por pantalla y se introduce un retardo por cada iteración. Hemos creado una estructura, llamada ThreadArgs, donde se recoge la matriz, el multiplicador, el número de hilo y el estado de los dos hilos, en definitiva, todo lo que necesita la rutina MultiplicarMatriz para funcionar.

Respondiendo a la pregunta de si se puede compartir información entre hilos, como los hilos comparten el mismo espacio de memoria sí que es posible, incluyendo en la estructura mencionada anteriormente un puntero que guarde el progreso de cada hilo. Cuando un hilo multiplica una fila, aumenta en 1.

```
lucia@hp-pavilion17:~/Escritorio/practica1$ ./ejercicio13
Introduce la dimension de la matriz cuadrada: 3

Introduzca multiplicador 1: 3

Introduzca multiplicador 2: 6
Introduzca matriz 1:
3 5 4 6 2 2 1 2 3
Introduzca matriz 2:
3 5 5 3 2 2 2 2 3
Hilo 1 multiplicando fila 0 resultado  9  15  12  - el Hilo 2 va por la fila 0
Hilo 2 multiplicando fila 0 resultado  18  30  30  - el Hilo 1 va por la fila 1
Hilo 1 multiplicando fila 1 resultado  18  6  6  - el Hilo 2 va por la fila 1
Hilo 2 multiplicando fila 1 resultado  18  12  12  - el Hilo 1 va por la fila 2
Hilo 1 multiplicando fila 2 resultado  3  6  9  - el Hilo 2 va por la fila 2
Hilo 2 multiplicando fila 2 resultado  12  12  18  - el Hilo 1 ha terminado
```