Universidad Autónoma de Madrid

Departamento de Informática

# Sistemas Operativos

# Memorias Práctica - 1

Hecho por: Diego Rodríguez Ortiz

y Alejandro García Hernando

# Introducción

En esta práctica, hemos creado un programa que dada una función hash g(x) y 2 enteros ‘x’ y 'r', calcula ‘y’ tal que (g^r)(y)=x.

Para encontrar dicho ‘y’, hay que empezar buscando g(y1)=x, posteriormente g(y2)=y1 porque (g^2)(y2)=g(y1)=x, después g(y3)=y2 porque (g^3)(y3)=(g^2)(y2)=g(y1)=x… Tras repetir este procedimiento ‘r’ veces, llegaremos a (g^r)(yr)=(g^r)(y)=...=(g^2)(y2)=g(y1)=x.

Como la función hash no es invertible, la única manera de calcular los ‘ya’ (con 1<=a<=r) anteriores, es mediante la prueba y error. Se deben probar todos los valores entre 1 e INT_MAX hasta encontrar uno que cumpla la condición que buscamos. Para hacer este trabajo más rápidamente se especifican un número de hilos de ejecución, que se reparten el rango de búsqueda, reduciendo significativamente el tiempo empleado para encontrar cada valor. Cuando un hilo acaba, avisa al resto cambiando el valor de una variable global y el resto finaliza inmediatamente.

Además de hacer uso de los hilos para optimizar el programa, también hemos utilizado la función fork para dividir el trabajo en tres procesos: Principal (controla el retorno del proceso Minero), Minero (en el encargado de buscar las soluciones con el proceso descrito anteriormente) y Monitor (comprueba que las soluciones obtenidas son correctas y las imprime por pantalla). De este modo se pueden hacer las labores de minado paralelamente a las de impresión y comprobación, sin necesidad de esperarse mutuamente.

# Estructura de código

El proyecto contiene 6 ficheros:

-Makefile: contiene una serie de comandos para la compilación y ejecución del código.

-mrush.c: la función principal se encuentra aquí. Esta función recoge los parámetros introducidos por el usuario y hace el primer fork, que tiene como padre al proceso Principal y como hijo a Minero. Principal se encarga de imprimir por pantalla el código de salida de Minero, mientras que este último llama a la función minero de miner.c.

-pow.c y pow.h: son archivos facilitados en moodle y contienen la función pow_hash con la que se trabajará y la macro POW_LIMIT que indica el valor máximo de su imagen.

-miner.c y miner.h: contienen todas las funciones se encargan de la minería:

-func_minero: función ejecutada por cada uno de los hilos. Busca linealmente el resultado indicado en el rango recibido. El código de salida es la solución encontrada o -1

en caso de acabar sin encontrarla (bien porque un hilo ya ha encontrado la solución o porque hayan comprobado todos los valores del rango).

-monitor: función ejecutada por el proceso Monitor. Gestiona la comunicación con el proceso Minero y comprueba las respuestas que éste le facilita. Las imprime por pantalla y le indica a Minero si son correctas. Si todo funciona correctamente finaliza con código de salida EXIT_SUCCESS o EXIT_FAILURE en caso de que haya sucedido algún error.

-minar: función ejecutada por el proceso Minero, gestión los hilos. En primer lugar crean los hilos y divide el rango de búsqueda de forma equitativa entre todos ellos. Posteriormente recoge todos sus códigos de salida y los analiza hasta encontrar un resultado que escribe en la tubería. Espera la respuesta del Monitor y en caso de que sea positiva vuelve a comenzar con la nueva ronda (en caso de error sale con EXIT_FAILURE).

-minero: es la función principal. Crea el proceso Monitor y las tuberías mediante las que se comunicará con dicho proceso. Posteriormente cada uno de los procesos llama a su función característica.

# Pruebas

## Detención de hilos utilizando una variable global.

Tras familiarizarnos con la noción de hilo, desarrollamos una prueba para cerciorarnos que realizamos correctamente la creación de hilos, su detención y su final recogida.

Para ello simplificamos el programa a esas tres funcionalidades. La prueba crea 100 hilos los manda a ejecutar una simple función de búsqueda lineal que recorre un intervalo hasta encontrar un número pasado por parámetro. Para poder evaluar la detención de hilos a través de una variable global, cada hilo recorre el intervalo de 0 a INT_MAX en busca de INT_MAX, salvo el hilo 73 que buscará el 17. Cada hilo devuelve -1 si encuentra el valor o el número por el número en el que se ha detenido. Tras recoger los retornos de los N hilos los imprimimos por pantalla. De los resultados mostrados somos capaces de deducir que los primeros hilos alcanzan números mayores, ninguno INT_MAX, ya que la creación de hilos sí es secuencial. Sin embargo según vamos avanzando en los índices vemos como aparecen ceros, hilos que al comenzar su ejecución ya se había encontrado el 17. Como se puede apreciar los números devueltos por cada hilo son aparentemente arbitrarios ya que es difícil prever como el planificador les repartirá tiempo de procesador.

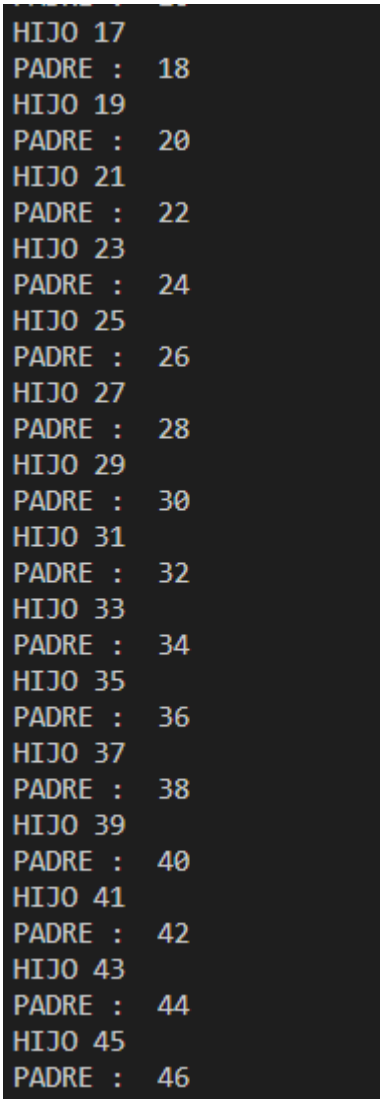
```
53 7536182
54 7562230
55 0
56 0
57 7528003
58 7611650
59 0
60 0
61 7668190
62 7600260
63 0
64 0
65 1880549
66 7743767
67 0
68 0
69 1878783
70 7639860
71 0
72 0
73 -1
74 7254760
75 0
76 0
77 0
78 7261906
79 0
80 0
81 0
82 7566499
83 0
84 0
85 0
86 7566235
87 0
88 0
```

## Creación de pipes , comunicación y sincronización de procesos a través de ellas

De nuevo, para asegurarnos de que la implementación de las pipes era correcta, simplificamos el problema eliminando toda la minería. El programa resultante crea dos pipes para que tras el fork los procesos sean capaces de comunicarse entre ellos. Cada proceso realiza la simple tarea de leer lo que escribe el otro, sumarle uno y reescribirlo. De esta forma, el que comienza escribiendo el 0 sumará de un impar al siguiente par y el que comienza leyendo sumará de par a impar. Además para ser capaces de monitorizar el orden de ejecución realizamos printf tras cada suma.

Como se aprecia en la terminal, al ser read y write rutinas bloqueantes, sincronizan a los procesos y somos capaces de prever el orden de ejecución (aproximadamente, solo el orden

de las sentencias de read y write de cada uno). Gracias a esta prueba nos aseguramos de que ni el monitor recibe una nueva búsqueda antes de haber impreso el resultado de la anterior, saltándose una, ni que el monitor imprima varias veces la misma búsqueda ya que el minero no le ha actualizado con la siguiente.




## Conclusión

En esta práctica al tratarse de la realización de una tarea muy fácilmente divisible, los hilos suponen una gran ventaja en el rendimiento. Al ser capaces de dividir el trabajo entre distintos hilos que se ejecutan concurrentemente somos capaces de dividir el coste en aproximadamente el número de hilos capaces de ejecutarse concurrentemente. Además para minimizar el número de bloqueos del proceso que realiza la búsqueda se crea otro proceso dedicado a las operaciones de E/S. Este sencillo ejemplo nos da una idea de cómo diversas aplicaciones pueden repartir las tareas en especial aquellas relacionadas con E/S para reducir bloqueos y aumentar su eficiencia.