



## Multiprocesadores Curso 2017-18



### Práctica 03.- Bloqueando y desbloqueando **MUTEX**

Daniel García Mármol

## Actividad 1

Modificar el programa dado para el uso de variables MUTEX en Pthread, donde la creación de hilos es dinámica y se deberá pensar en la mejor forma de balancear la carga.

```
#include <pthread.h>
#include <stdlib.h>
#include <stdio.h>

#define ARRAYSIZE 1000000 // Definimos el tamaño del array
#define NUM_THREADS 4    // Definimos el número de hilos

pthread_mutex_t llave;    // Creamos una variable global de tipo mutex,
                          // que usaremos para bloquear o activar el
                          // acceso a escritura para una variable para
                          // evitar conflictos entre los distintos hilos

double sum = 0.0;        // Creamos una variable global de tipo double
                          // que será la que guarde la suma entre los
                          // distintos hilos

// Estructura para pasar al hilo

struct parametros {      // Definimos una estructura con los parámetros
    int inicio; ;        // necesarios que necesitaremos pasarle a nuestro
    int fin ;            // hilo. Esto incluye 3 variables tipo int, que
    int nh;              // indicaran, principio, fin, numero de hilo y
    double *vector;      // un puntero de tipo double, que será el array en
};                       // el cual el hilo estará trabajando

int main(int argc, char *argv[]){

    // Declaramos el puntero para un vector
    double *vector;

    // Se reserva dinámicamente memoria para el vector
    vector = malloc(sizeof(double)*ARRAYSIZE);

    // Estructura que rellena las celdas del vector creado anteriormente
    for(int i = 0; i < ARRAYSIZE; i++){
        vector[i]= i* 1.0;
    }
```

```

// Declaramos un puntero de hilos y reservamos dinámicamente memoria para
// los hilos

pthread_t *thread;
thread = (pthread_t*)malloc(sizeof(pthread_t)*NUM_THREADS);

// Inicializamos el mutex para poder bloquear/abrir cuando necesitemos

pthread_mutex_init(&llave, 0);

// Paquete guarda la división entre el número de celdas que tiene nuestro array
// entre el numero de hilos, para saber cuantas celdas le corresponde a cada uno

int paquete = ARRAYSIZE / NUM_THREADS;

// Excedente recoge el resto para balancear la carga

int excedente = ARRAYSIZE % NUM_THREADS;

// Inicializamos las variables fin e inicio, que usaremos para informar donde empieza
// y acaba el trabajo de cada hilo en el vector

int fin = 0;
int inicio = 0;

// Creamos una array de parámetros, de tamaño cuantos hilos tengamos pues estos van
// a ser los datos con los que trabaje cada uno, que se les pasará cuando se inicialicen

struct parametros parametros_array[NUM_THREADS];

// Bucle for que rellena los parametros con inicio del hilo, fin del hilo, numero de hilo
// y crea el hilo, esto se repite en cada iteración con nuevos valores.

for(int i = 0; i<NUM_THREADS; i++){
    parametros_array[i].vector = vector;
    fin = inicio + paquete;
    if(excedente > 0){
        fin++;
        excedente--;
    }
    parametros_array[i].inicio = inicio;
    parametros_array[i].fin = fin;
    parametros_array[i].nh = i;
    pthread_create ( &thread[i], NULL, do_work, (void*)&parametros_array[i]);
    inicio = fin;
}

// Esperamos a que todos los hilos acaben

for(int i = 0; i<NUM_THREADS; i++){
    pthread_join( thread[i], NULL);
}

```

```
// Todos los hilos han llegado, por lo tanto sum almacena todo
// que es donde lo hemos guardado
```

```
printf ( "\nDone. Sum= %lf \n", sum);
```

```
//Comprobacion:
```

```
sum=0.0;
int i = 0;
for (i=0;i<ARRAYSIZE;i++){
    if(i%2 == 0) sum += vector[i];
    else sum -= vector[i];
}
printf("Check Sum= %lf\n",sum);
```

```
// Liberamos memoria reservada y eliminamos el mutex
```

```
free(thread);
pthread_mutex_destroy(&llave);
```

```
*** FIN DEL MAIN ***
```

A partir de aquí, es la función donde va a trabajar cada hilo:

```
void *do_work(void *arg){
    int i;
    double mysum=0.0;                                // Variable local suma

    struct parametros * p;                            // Mediante un casting creamos una
    p = ( struct parametros *) arg ;                  // estructura de tipo parametros,
                                                        // que recibe los valores que le hemos
                                                        // pasado al crear el hilo (inicio, fin,
                                                        // numero de hilo, y el vector)

    // Bucle for que recorre el vector que se ha pasado como parámetro, con su inicio
    // y su fin, por ejemplo 1/4 del vector para el caso del hilo 1 con un número
    // de 4 hilos, y suma si es par o resta si es impar en la variable local, el valor
    // de la posición del vector

    for(int i = (p -> inicio); i < (p -> fin); i++){
        if(i%2==0) mysum += (p -> vector[i]);
        else mysum -= (p -> vector[i]);
    }
}
```

```

pthread_mutex_lock(&llave);           // Cerramos la llave para que otros
                                       // hilos no puedan acceder temporalmente
                                       // a la variable global que vamos a
                                       // modificar

printf("\nLlave está cerrada");       // Imprimimos que la llave está
                                       // cerrada para comprobar por pantalla

sum += mysum;
printf("\nValor de sum = %lf", sum);  // Guardamos la suma local en la suma
                                       // global y la imprimimos

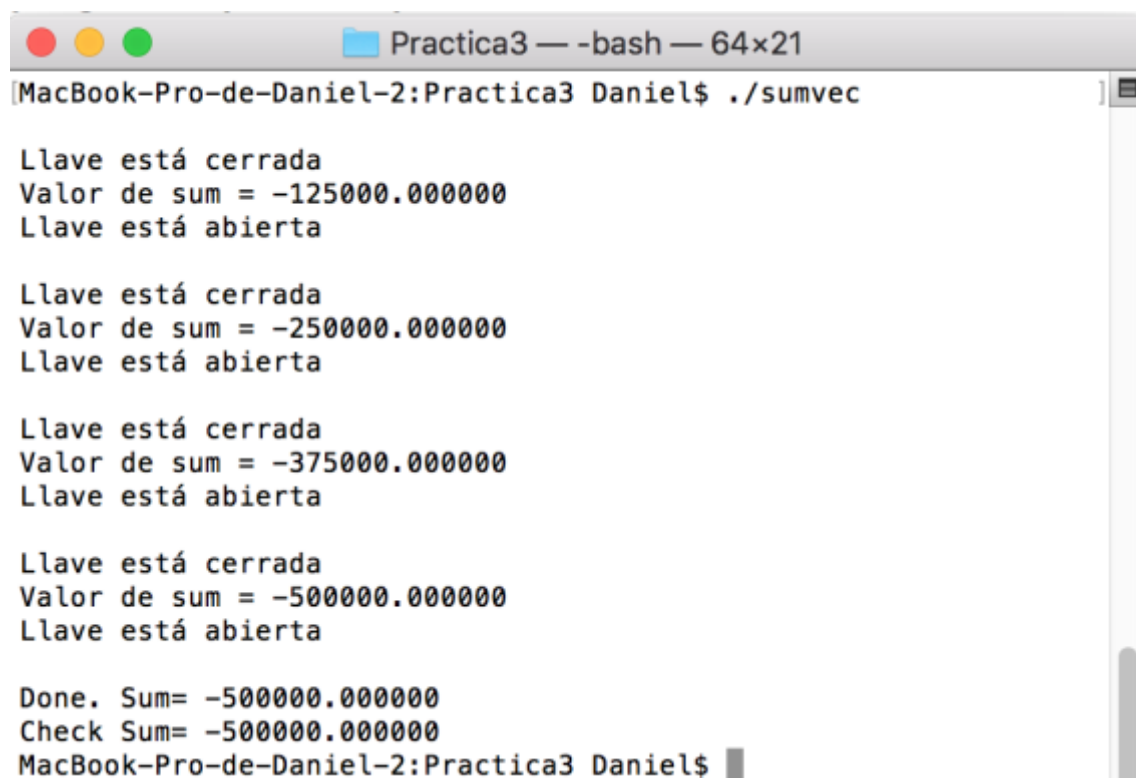
pthread_mutex_unlock(&llave);         // Abrimos la llave para que acceda el
                                       // siguiente hilo

printf("\nLlave está abierta\n");     // Imprimimos que la llave está abierta
                                       // para comprobar por pantalla

return 0;

```

A continuación, vamos a ver la salida del programa que hemos hecho:



```

MacBook-Pro-de-Daniel-2:Practica3 Daniel$ ./sumvec

Llave está cerrada
Valor de sum = -125000.000000
Llave está abierta

Llave está cerrada
Valor de sum = -250000.000000
Llave está abierta

Llave está cerrada
Valor de sum = -375000.000000
Llave está abierta

Llave está cerrada
Valor de sum = -500000.000000
Llave está abierta

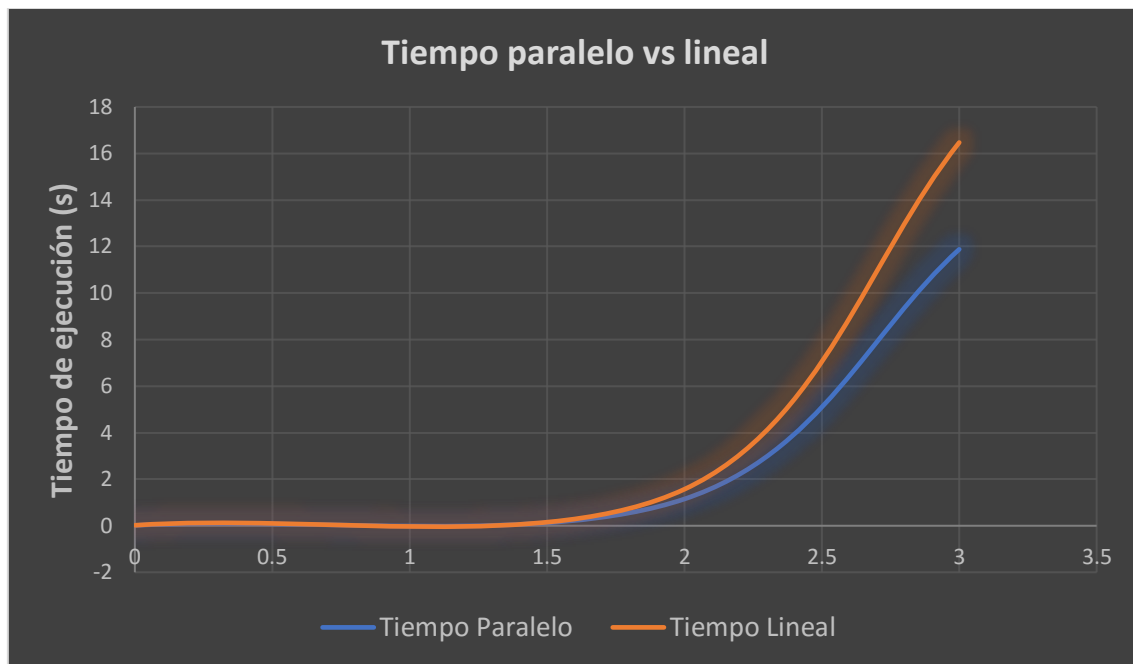
Done. Sum= -500000.000000
Check Sum= -500000.000000
MacBook-Pro-de-Daniel-2:Practica3 Daniel$

```

Como podemos ver, obtenemos el mismo valor.

Aunque no se pide, he realizado una gráfica en la que podamos ver como con valores muy grandes, el tiempo de ejecución de este programa que hemos realizado es más rápido que el programa inicial de partida:

	10 <sup>6</sup>	10 <sup>8</sup>	10 <sup>9</sup>
Paralelo	0.016s	1.15s	11.88s
Lineal	0.021s	1.57s	16.47s



Como podemos observar, a valores pequeños no se nota la diferencia, pero a medida que el tamaño del vector se hace más grande, hay una diferencia de tiempo importante. Se ha mantenido el número de hilos en 4, y se ha ampliado el tamaño del array hasta 10<sup>9</sup>, que es el máximo que deja introducir, pues recordemos que el máximo número de una variable int es 2147483647.