



# Práctica 2: Gestión de Threads

**Nicolás Sela Ramos**  
**Arturo Requejo Portilla**



UNIVERSIDAD  
**NEBRIJA**

## Tabla de contenido

<b><i>I. Introducción .....</i></b>	<b><i>3</i></b>
<b><i>II. Enunciados .....</i></b>	<b><i>3</i></b>
A. Ejercicio 1 .....	3
B. Ejercicio 2 .....	4
C. Ejercicio 3 .....	4
<b><i>III. Resultados .....</i></b>	<b><i>4</i></b>
A. Ejercicio 1 .....	5
B. Ejercicio 2 .....	6
C. Ejercicio 3 .....	7
<b><i>IV. Conclusiones.....</i></b>	<b><i>8</i></b>
<b><i>V. Bibliografía .....</i></b>	<b><i>8</i></b>

---

## I. Introducción

En esta práctica vamos a conocer la manera de presentar los threads y cómo lo gestionan los sistemas operativos. Lo que se pretende realizar en esta práctica es cómo planificar los procesos y aligerarlos mediante el uso de los hilos.

## II. Enunciados

### A. Ejercicio 1

Implementar un programa que reciba dos números enteros como parámetros de entrada y calcule sus factoriales de forma concurrente utilizando dos hilos que se ejecutan en paralelo con el hilo principal. El hilo principal deberá esperar a que terminen los otros dos hilos. **Justificar el proceso 3 PUNTOS.**

### B. Ejercicio 2

El programa que se muestra a continuación cuenta el número de veces que el carácter 'a' o 'A' aparece en el fichero indicado como parámetro de entrada. Modificarlo para que ahora se cree un thread y sea éste el que ejecute la función encargada de contar los caracteres. **Justificar el proceso 3 PUNTOS.**

```
1  #include <unistd.h>
2  #include <stdlib.h>
3  #include <stdio.h>
4  #include <sys/types.h>
5  #include <sys/stat.h>
6  #include <fcntl.h>
7
8  #define MAXLON 1000
9
10 void cuenta (char *nombre) {
11     int pos, cont = 0, leidos;
12     char cadena[MAXLON];
13     int fd;
14
15     fd = open (nombre, O_RDONLY);
16     while ((leidos = read (fd, cadena, MAXLON)) != 0){
17         for (pos = 0; pos < leidos; pos++){
18             if ((cadena[pos] == 'a') || (cadena [pos] == 'A')){
19                 cont++;
20             }
21         }
22     }
23
24     printf ("Fichero %s: %d caracteres 'a' o 'A' encontrados\n", nombre, cont);
25     close (fd);
26 }
27
28 int main (int argc, char *argv[]){
29     if (argc != 2){
30         printf ("Indica el nombre de un fichero.\n");
31         exit(0);
32     }
33     cuenta (argv[1]);
34     return 0;
35 }
```

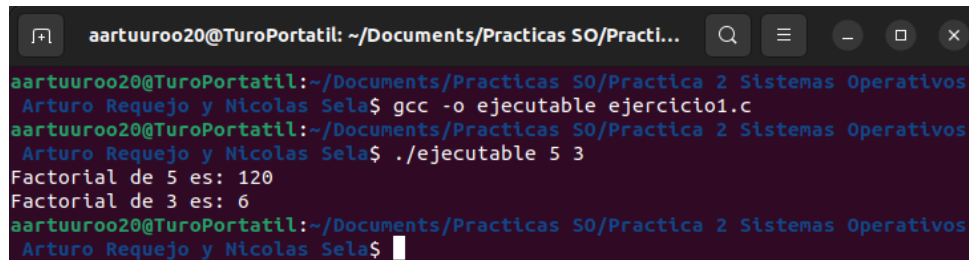
### C. Ejercicio 3

---

Generar un código en C que muestre el funcionamiento de los mutex. Para ello, se creará un programa que realice el cuadrado de un número introducido por consola. El programa debe ser capaz realizar el cálculo del cuadrado de los números impares en el thread1 y el de los pares en el thread2. La salida debe ser sincronizada. **Justificar el proceso 4 PUNTOS.**

### III. Resultados

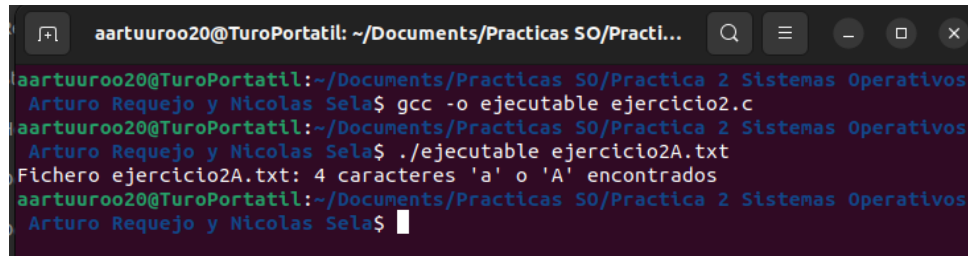
#### A. Ejercicio 1



```
aartuuroo20@TuroPortatil: ~/Documents/Practicas SO/Practi...
aartuuroo20@TuroPortatil:~/Documents/Practicas SO/Practica 2 Sistemas Operativos
Arturo Requejo y Nicolas Sela$ gcc -o ejecutable ejercicio1.c
aartuuroo20@TuroPortatil:~/Documents/Practicas SO/Practica 2 Sistemas Operativos
Arturo Requejo y Nicolas Sela$ ./ejecutable 5 3
Factorial de 5 es: 120
Factorial de 3 es: 6
aartuuroo20@TuroPortatil:~/Documents/Practicas SO/Practica 2 Sistemas Operativos
Arturo Requejo y Nicolas Sela$
```

Creamos una función factorial que reciba como parámetro un puntero a el número del que hay que sacar el factorial. En la función principal se inicializa el thread principal y los parámetros por defecto. Después se crean los threads 1 y 2 que reciben por parámetro la dirección de ambos, el parámetro por defecto, la función a realizar y el dato a emplear. Se invoca al thread1 y se mantiene a la espera el 2 hasta que el primero, termine. Por último, finalizo la ejecución de ambos threads. Mediante un condicional que comprueba la introducción de los números, se gestiona la excepción en caso de que falten datos por introducir.

#### B. Ejercicio 2



```
aartuuroo20@TuroPortatil: ~/Documents/Practicas SO/Practi...
aartuuroo20@TuroPortatil:~/Documents/Practicas SO/Practica 2 Sistemas Operativos
Arturo Requejo y Nicolas Sela$ gcc -o ejecutable ejercicio2.c
aartuuroo20@TuroPortatil:~/Documents/Practicas SO/Practica 2 Sistemas Operativos
Arturo Requejo y Nicolas Sela$ ./ejecutable ejercicio2A.txt
Fichero ejercicio2A.txt: 4 caracteres 'a' o 'A' encontrados
aartuuroo20@TuroPortatil:~/Documents/Practicas SO/Practica 2 Sistemas Operativos
Arturo Requejo y Nicolas Sela$
```

Hemos añadido dos líneas de código, usando los métodos join y create. Mediante el uso de create creamos un hilo que llama a la función de contar los caracteres y envía como parámetro el nombre del archivo. Mediante el uso de Join sincronizamos el subproceso que realiza la ejecución del método que realiza el contado de letras “a” para asegurarse de que hasta que no haya terminado no se realiza ninguna otra operación. Mediante el uso de exit finalizamos la ejecución de los threads.

#### C. Ejercicio 3

```
le ejercicio3.c
aartuuroo20@TuroPortatil:~/Documents/Practicas SO/Practica 2 Sistemas Operativos
Arturo Requejo y Nicolas Sela$ ./ejecutable 10
Hilo 1, que calcula impares:
El cuadrado del numero 1 es: 1
El cuadrado del numero 3 es: 9
El cuadrado del numero 5 es: 25
El cuadrado del numero 7 es: 49
El cuadrado del numero 9 es: 81
Hilo 2, que calcula pares:
El cuadrado del numero 0 es:
El cuadrado del numero 2 es: 4
El cuadrado del numero 4 es: 16
El cuadrado del numero 6 es: 36
El cuadrado del numero 8 es: 64
1
4
9
16
25
36
49
64
81
```

En este ejercicio hemos realizado dos funciones, una que calcula el cuadrado de los números pares hasta el numero introducido por el usuario y la otra realiza lo mismo, solo que con los números impares. Dentro de ambas funciones hemos introducido un `mutex_lock` el cual protege los recursos empleados por la función, para evitar acceder a otras zonas de la memoria, por ejemplo. Posteriormente cuando haya acabado de realizar todas las operaciones correspondientes mediante el `mutex_unlock` se liberan los recursos. Este proceso se realiza en ambas funciones. En la función principal inicializamos una variable de tipo `mutex`, además inicializamos los parámetros de los threads por defecto, también realizamos la creación de threads pasando por parámetros los threads por referencia, las funciones de cálculo de pares e impares y el numero a partir del cual se van a generar los factoriales. Por último hemos realizado un `thread join` sincroniza los procesos, mantiene uno a la espera mientras el otro finaliza.

Hemos realizado el ejercicio de dos formas, la primera forma nos imprime el cuadrado de los números divididos en dos bloques los pares y los impares, luego hemos conseguido juntar y mostrar en orden los cuadrados de ambos números mediante la creación de un array como variable global, que sirva de sincronizador entre los hilos.

## D. Conclusiones

Durante la realización de esta práctica hemos aprendido a implementar tanto la creación de hilos, como la sincronización y finalización de la ejecución de los mismos. También hemos aprendido la importancia que tiene su introducción en el ejercicio 2. Por último, en el ejercicio 3 hemos aprendido a implementar los `mutex` en consonancia con los threads y la importancia de los mismos durante la ejecución del programa.

## E. Bibliografía

---

Thread - Método, Microsoft Learn

<https://learn.microsoft.com/eses/dotnet/api/system.threading.thread.join?view=net-6.0>

IBM – Pthread Create()

<https://www.ibm.com/docs/en/zos/2.3.0?topic=functions-pthread-create-create-thread>

---