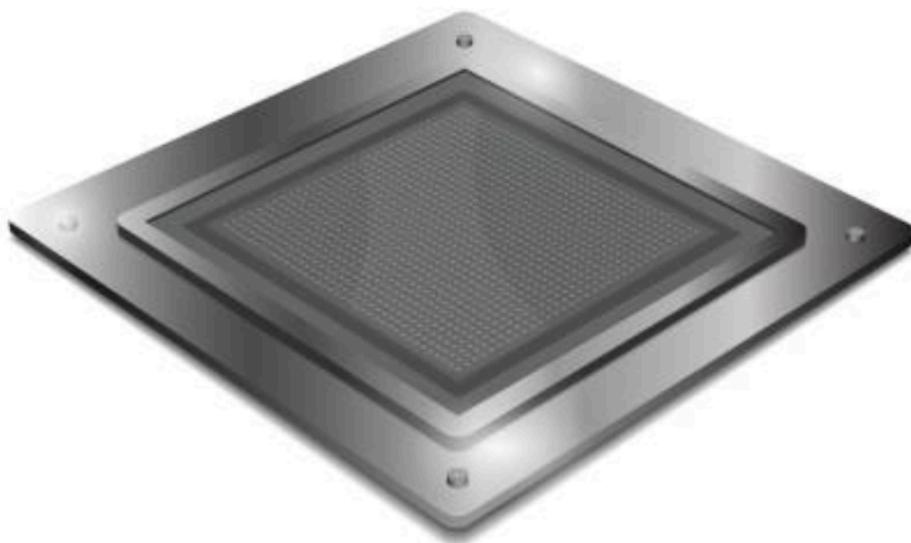


Práctica 2: Introducción a la programación usando POSIX Threads



Realizado por:

Daniel García Márquez



1.- Entorno de trabajo

El entorno de trabajo en el cual realizaremos esta práctica contaremos con un equipo con sistema operativo **macOS High Sierra**, con el compilador **GCC** y el depurador **GDB** instalados previamente y **Gedit** como editor de texto.

Actividad 1: Identificar las funciones Posix-Threads y explicar brevemente que realiza cada una y que parámetros y de qué tipo aceptan. ¿Qué realiza el programa?

Nos dan el siguiente código el cual debemos indentificar las funciones de la librería Posix-Threads:

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>
4
5 void *print_message_function( void *ptr );
6
7 main(){
8
9     pthread_t thread1, thread2;
10    char *message1 = "Thread 1";
11    char *message2 = "Thread 2";
12    int iret1, iret2;
13    /* Create independent threads each of which will execute function */
14    iret1 = pthread_create( &thread1, NULL, print_message_function, (void*)
15    message1);
16    iret2 = pthread_create( &thread2, NULL, print_message_function, (void*)
17    message2);
18    /* Wait till threads are complete before main continues. Unless we */
19    /* wait we run the risk of executing an exit which will terminate */
20    /* the process and all threads before the threads have completed. */
21    pthread_join( thread1, NULL);
22    pthread_join( thread2, NULL);
23    printf("Thread 1 returns: %d\n",iret1);
24    printf("Thread 2 returns: %d\n",iret2);
25    exit(0);
26 }
27
28 void *print_message_function( void *ptr ){
29     char *message;
30     message = (char *) ptr;
31     printf("%s \n", message);
32 }
```

Figura – 1 : Código dado en la práctica para analizarlo

Empezemos por orden:

El tipo de variable **pthread_t** es un medio para hacer referencia a los hilos. Es necesario que exista una variable **pthread_t** para cada subprocesso creado. Algo así como **pthread_t thread1**, como podemos observar en la línea 9. Es lo que solemos hacer por ejemplo cuando queremos crear variables de tipo **int**, **char**, **double**, ect.

Una vez que se ha definido la variable **pthread_t**, podemos crear el hilo usando **pthread_create**. Este método toma cuatro argumentos: un puntero a la variable **pthread_t**, cualquier atributo adicional (no se preocupe por esto por ahora, simplemente ajústelo a **NULL**), un puntero a la función a llamar (es decir, el nombre del punto de entrada a nuestro método) y el puntero pasado como argumento a la función.



Cuando el hilo recién creado ha terminado de hacerlo, necesitamos unir todo. Esto se hace mediante la función `pthread_join` que toma dos parámetros: la variable `pthread_t` utilizada cuando se llamó `pthread_create` (no es un puntero esta vez) y un puntero al puntero de valor de retorno (no se preocupe por ahora, simplemente configúrelo en `NULL`).

Por último, para que nuestro hilo funcione debe apuntarlo a una función para que comience la ejecución. La función debe devolver `void *` y tomar un único argumento `void *`. Por ejemplo, si desea que la función tome un argumento entero, necesitará pasar la dirección del entero y desreferenciarlo más tarde.

Este programa lo que hace es crear 2 hilos, los cuales imprimen por pantalla los mensajes pasados como argumento a la función `create`, pues la función a llamar lo único que hace es coger el valor pasado e imprimirla, y luego, para finalizar imprime que ha acabado.

Actividad 2: Modificar el programa para que acepte por línea de comandos el número de threads ("hilos") que se quieren ejecutar.

A continuación se expondrá el código resultante de la modificación del programa dado:

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>
4
5 void *print_message_function( void *ptr );
6
7 int main(){
8
9     int i;
10
11    // Pedir por pantalla el número de Thread
12    int nThread;
13    printf("Inserte el número de Thread que se quieren crear:\n");
14    scanf("%d", &nThread);
15
16    // Inicializar Thread
17
18    char** mensajes = malloc(sizeof(char*)*nThread);
19
20    pthread_t *thread;
21    thread = (pthread_t*)malloc(sizeof(pthread_t)*nThread);
22    for(i = 1; i <= nThread; i++){
23        mensajes[i] = malloc(sizeof(char)*20);
24        sprintf(mensajes[i], "Thread %d", i);
25        pthread_create ( &thread[i], NULL, print_message_function, (void*) mensajes[i]);
26    }
27
28    for(i = 1; i <= nThread; i++){
29        pthread_join( thread[i], NULL);
30    }
31
32    for(i = 1; i <= nThread; i++){
33        printf("Thread %d returns\n", i);
34    }
35
36    free(thread);
37    for(i = 1; i <= nThread; i++){
38        free(mensajes[i]);
39    }
40    free(mensajes);
41
42
43    exit(0);
44}
45
46 void *print_message_function( void *ptr ){
47
48    //char *message;
49    //message = (char *) ptr;
50    //printf("%s \n", message);
51    printf("%s\n", (char*) ptr);
52    return 0;
53 }
```

Figura – 2 : Programa realizado para la actividad 2



Procedamos por partes, pues lo primero que nos encontramos dentro del main es la parte en la cual usando la función **scanf** guardamos el valor que se inserta por pantalla, como se nos pide en el enunciado que posteriormente utilizaremos para la creación de tantos hilos como le pasemos de parámetro entero. Lo siguiente que vemos, es que reservamos espacio dinámicamente tanto para los mensajes, que será una matriz, como para los hilos, que será un vector. Seguidamente usamos la función **sprintf** para pasarle a mensajes matriz como queremos guardar en esa posición, que usaremos en la siguiente línea para pasarle como dato en la creación del hilo. El siguiente bucle **for** lo que hace es unir todos los hilos. Seguidamente, imprimimos por pantalla que los hilos han sido terminados y para acabar, liberamos la memoria reservada dinámicamente para los punteros **thread**, y **mensajes**. A continuación, una muestra de la salida:

```
Inserte el número de Thread que se quieren crear:  
6  
Thread 1  
Thread 2  
Thread 3  
Thread 4  
Thread 5  
Thread 6  
Thread 1 returns  
Thread 2 returns  
Thread 3 returns  
Thread 4 returns  
Thread 5 returns  
Thread 6 returns
```



Actividad 3: Construir una función que realice la suma de un vector, de tipo double y de una longitud determinada pasada por parámetro.

A continuación se expondrá el código resultante de la modificación del programa dado:

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>
4
5 //pthread_mutex_t llave;
6
7 int longitud;
8
9 void *suma_vector_function( void *longitud );
10
11
12 int main(){
13     int i;
14
15
16     printf("Inserte la longitud de su vector:\n");
17     scanf("%d", &longitud);
18
19     double *vector;
20     vector = malloc(sizeof(double)*longitud);
21     double contador;
22     for(int i = 0; i < longitud; i++ ){
23         contador+=1.0;
24         vector[i]= contador;
25     }
26
27     int nThread = 1;
28
29     pthread_t *thread;
30     thread = (pthread_t*)malloc(sizeof(pthread_t)*nThread);
31     for(i = 1; i <= nThread; i++){
32         pthread_create( &thread[i], NULL, suma_vector_function, (void*)vector);
33     }
34
35
36     for(i = 1; i <= nThread; i++){
37         pthread_join( thread[i], NULL);
38     }
39
40     printf("\nEl programa ha finalizado\n");
41     return 0;
42 }
43
44 void *suma_vector_function( void *array ){
45
46     double *vector = (double *) array;
47     double suma;
48
49     for(int i = 0; i < longitud ; i++){
50
51         suma += vector[i];
52     }
53     printf("\nEl resultado de la suma del vector es %f\n", suma);
54     return 0;
55 }
56 }
```

Figura – 4 : Programa realizado para la actividad 3

La estructura del main es muy parecida al ejercicio anterior pues nos hemos centrado más en cambiar la función, que es lo que se nos pide en el ejercicio. Lo que hemos hecho para que sea de tipo double, es que como el parámetro siempre que le pasamos a la función pthread_create es de tipo void, le hemos hecho un casting tanto a la entrada, como después al recibir este parámetro en nuestra función que hemos denominado suma_vector_funcion. Hemos creado una variable, también de tipo double, para ir



guardando el resultado de la suma iterativa que se realizará en el siguiente paso. Creamos un bucle for, que ayudándonos de haber declarado la longitud como un parámetro int global, accesible a todos pues no forma parte del main, nos ahorraremos crear una estructura de momento. Pues este for recorrerá el vector y guardará la suma en la variable suma. Por último, imprimirá por pantalla el valor de esta. A continuación, una muestra de la salida:

```
[MacBook-Pro-de-Daniel-2:pRACTICA2 Daniel$ ./ej3
Inserte la longitud de su vector:
6

El resultado de la suma del vector es 21.000000

El programa ha finalizado
```

Figura – 5 : Le pasamos por parámetro el valor 6 y nos devuelve 21.



Actividad 4: Construir un programa que incorpore la función de la actividad anterior para que ejecute de manera concurrente 4 threads de dicha función y realizando cada uno de los threads la suma parcial de 1/4 de la longitud total del vector. Al final el programa tendrá que obtener la suma total de los elementos del vector.

A continuación se expondrá el código resultante de la modificación del programa dado:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>
4
5
6 // Estructura para pasar al hilo
7
8 struct parametros {
9     int inicio ;
10    int fin ;
11    int nh;
12    double *vector;
13 };
14
15
16 void *suma_vector_function( void *arg );
17
18 int main(){
19
20
21     // Pedir por pantalla
22     int i;
23     int longitud;
24     printf("Inserte la longitud de su vector:\n");
25     scanf("%d", &longitud);
26
27     //Vector declarado dinamicamente
28
29     double *vector;
30     vector = malloc(sizeof(double*)*longitud);
31
32     // Estructura que rellena el vector
33
34     double contador;
35     for(int i = 0; i < longitud; i++ ){
36         contador+=1.0;
37         vector[i]= contador;
38     }
39
40
41     int nThread = 4; // Nos pide el ejercicio que sea igual a 4
42     pthread_t *thread;
43     thread = (pthread_t*)malloc(sizeof(pthread_t)*nThread);
44
45
46
47     // Ejecución de hilos para determinado sector
48
49
50
51     int elementos = longitud;
52     int nHilos = nThread;
53
54     int paquete = elementos / nHilos;
55     int excedente = elementos % nHilos;
56
57     int fin = 0;
58     int inicio = 0;
59
60
61
62     struct parametros p1 ;
63     struct parametros p2 ;
```



```
61     struct parametros p1 ;
62     struct parametros p2 ;
63     struct parametros p3 ;
64     struct parametros p4 ;
65
66
67     p1.vector = vector;
68     p2.vector = vector;
69     p3.vector = vector;
70     p4.vector = vector;
71
72     for(int i = 0; i<nHilos; i++){
73         fin = inicio + paquete;
74         if(excedente > 0){
75             fin++;
76             excedente--;
77         }
78
79         p&i.inicio = inicio;
80         p&i.fin = fin;
81         p&i.nh = i;
82         pthread_create( &thread[i], NULL, suma_vector_function, (void*)&p&i);
83         pthread_join( thread[i], NULL);
84         inicio = fin;
85     }
86
87
88 // Limpiamos
89
90 free(vector);
91 free(thread);
92 return 0;
93 }
94 }
95
96
97
98
99
100 void *suma_vector_function( void *arg ){
101
102     struct parametros * p;
103     p = ( struct parametros *) arg ;
104
105     double suma;
106
107     for(int i = (p -> inicio); i < (p -> fin); i++){
108         suma += (p -> vector[i]);
109     }
110     printf("\nHilo %d -> Inicio: %d ; Fin: %d, resultado de la suma = %f\n", (p -> nh), (p -> inicio), (p -> fin), suma);
111
112     return 0;
113 }
114 }
```

Figura – 6 : Programa realizado para la actividad 4

Algo debe de haber mal, pues el tiempo que tarda es igual que el de la actividad 3, pero no encuentro el error.