LABORATORIO DE SISTEMAS OPERATIVOS

PERÍODO ACADÉMICO: 2025 – B **EQUIPO:**

PROFESOR: Ing. Marcela Saavedra

TIPO DE INSTRUMENTO: Guía de Laboratorio

TEMA: CREACIÓN DE HILOS

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

1. OBJETIVOS

1.1. Familiarizar al estudiante con el uso de las funciones **pthread.**
1.2. Realizar varias actividades de creación de hilos.

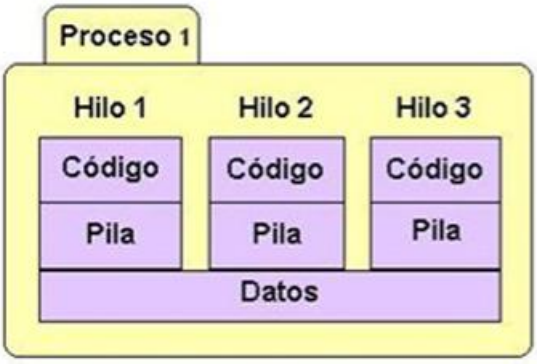
2. MARCO TEÓRICO

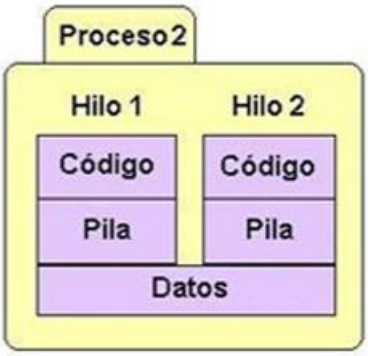
Manejados por el S.O.	Manejados por los procesos
Independientes de otros procesos	Relacionados con otros hilos del mismo proceso
Memoria privada, se necesitan mecanismos de comunicación para compartir información	Memoria compartida con el resto de hilos que forman el proceso

Tabla 1: Procesos e hilos

proceso

hilos





Dentro de un proceso puede haber varios hilos de ejecución. Los hilos dentro de un proceso comparten la misma memoria. Para escribir programas multihilo en C se utiliza la librería ***pthread*** que implementa el standard POSIX (Portable Operating System Interface).

pthread_create() es la función que permite crear un nuevo hilo de ejecución. Admite cuatro parámetros:

- ***pthread_t ** es un puntero a un identificador de thread. La función devolverá este valor, de forma que luego se pueda referenciar al hilo.

- ***pthread_attr_t ** son los atributos de creación del hilo. Hay varios atributos posibles, como por ejemplo la prioridad. Un hilo de mayor prioridad se ejecutará con que otros hilos de menor prioridad. Se puede pasar **NULL**, con lo que el hilo se creará con sus atributos por defecto.
- ***void *(*)(void *)*** es una función que admite un puntero **void *** y que devuelve **void ***. Esta función es la que se ejecutará como un hilo aparte. El hilo terminará cuando la función termine o cuando llame a la función ***pthread_exit()***.
- ***void **** es el parámetro que se le pasará a la función anterior cuando se ejecute en el hilo aparte.

La función ***pthread_create()*** devuelve **0** si todo ha ido bien, o un valor distinto de **0** si ha habido algún problema y no se ha creado el thread.

El código de creación del **hilo** quedaría.

```
void *funcionDelThread (void *);
...
pthread_t idHilo;
...
pthread_create (&idHilo, NULL, funcionDelThread, NULL);
```

En este ejemplo no se le pasará ningún parámetro, es decir, **NULL**.

Ahora el programa principal y la ***funciónDelThread()*** se estarán ejecutando "simultáneamente".

- ***pthread_join()*** permite que un hilo espere por otro**.**
- ***pthread_mutex_lock()*** permite que dos o más hilos accedan sincronizadamente a un recurso común.

Ejemplo:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>

//Primera funcion donde se imprime el valor de a
void *function1 (void *arg)
{
int a = 10;
int *p = &a;
printf ("El valor de p con function1 es %d \n",*p);
}

//Segunda funcion donde se imprime el valor de a
void *function2 (void *arg)
{
int a=20;
int *p = &a;
printf ("El valor de p con function2 es %d \n",*p);
}


int main()
{
/*Declaracion de hilos h1 y h2 de tipo pthread*/

/*Es necesario guardar el identificador ya que una vez que un hilo comienza a funcionar, la única forma de controlarlo es a través de su identificador*/
pthread_t h1 ;
pthread_t h2 ;
/*Se crean los hilos h1 y h2 y se inicia la ejecución de la función que se le pasa como tercer argumento function1 y function2*/
pthread_create (& h1 , NULL , function1, NULL);
/*La funcion pthread_join permite la ejecucion de los hilos*/
pthread_join(h1,NULL);
pthread_create (& h2 , NULL , function2, NULL);
pthread_join(h2,NULL);
}
```

Ejecutar con:
#gcc -pthread código_hilo.c -o código_hilo

3. PROCEDIMIENTO

3.1 Creación de “Hello world” con hilos.

Programa hilo.c que escribe “Hello world” utilizando dos hilos distintos h1 y h2, uno para la palabra Hello y otro para la palabra world.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>


/*Función que imprime Hello*/
void *hello (void *arg)
{
char msg[5] = "Hello";
printf("%s",msg);
}

/*Función que imprime world*/
void *world(void *arg)
{
char msg[5]="world";
printf("%s",msg);
}

/*Funcion main*/
void main()
{
pthread_t h1 ;
pthread_t h2 ;
pthread_create (& h1 , NULL , hello, NULL);
pthread_join(h1,NULL);
printf("\t");
pthread_create (& h2 , NULL , world, NULL);
pthread_join(h2,NULL);
printf ( "\n ");
}
```

Figura 1. Script de creación de un programa que imprime "Hello World" con hilos

3.2 Pasando parámetros a los hilos

El argumento se pasa a la función a través del cuarto parámetro de pthread_create
Si se quiere pasar un parámetro, éste debe ser de tipo puntero a void.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>

void *mensaje(void *arg)
{
        char *msg;
        msg = ( char *) arg;
        printf ("%s",msg) ;
}

void main()
{
        pthread_t h1;
        pthread_t h2;
        char *hello = "Hello";
        char *world = "world";
        pthread_create (& h1 , NULL , mensaje , (void *) hello);
        pthread_join ( h1 , NULL);
        printf("\t");
        pthread_create (& h2 , NULL , mensaje , (void *) world);
        pthread_join ( h2 , NULL);
        printf("\n");
}
```

Figura 2. Script de ejemplo para pasar parámetros a los hilos

3.3 Pasando estructuras a los hilos

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>

struct parametros
{
        int id ;
        float escalar ;
        float matriz [3][3] ;
};

void init ( float m [3][3])
{
        int i;
        int j;
        for ( i = 0 ; i < 3 ; i ++)
        {
        for ( j = 0 ; j < 3 ; j ++)
        {
        m[i ][ j ]= random () *100;
        }
        }
}

void * matrizporescalar ( void * arg )
{
        struct parametros * p;
        int i;
        int j;
        p = ( struct parametros *) arg ;
        for ( i = 0 ; i < 3 ; i ++)
        {
        printf (" Este es el hilo %d , multiplicación de la fila %d \n" , p -> id , i );
        for ( j = 0 ; j < 3 ; j ++)
        {
        p -> matriz [ i ][ j] = p -> matriz [ i ][ j] * p -> escalar ;
        usleep (1000) ;
        }
        }
}

int main ( int argc , char argv [])
{
        pthread_t h1 ;
        pthread_t h2 ;
        struct parametros p1 ;
        struct parametros p2 ;
        p1 . id = 1;
        p1 . escalar = 5.0;
        init ( p1 . matriz);
        p2 . id = 2;
        p2 . escalar = 10.0;
        init ( p2 . matriz );
        pthread_create (& h1 , NULL , matrizporescalar , (void *) & p1);
        pthread_join ( h1, NULL );
        pthread_create (& h2 , NULL , matrizporescalar , (void *) & p2);
        pthread_join ( h2 , NULL );
        printf ( " \n ");
}
```

Figura 3. Script de ejemplo para pasar estructuras en los hilos

Terminación del hilo

La terminación del hilo se produce cuando:

- La función que está ejecutando se termina.
- Ejecuta un return.
- Ejecuta la función pthread_exit.

Tiempo de ejecución del hilo

Se debe incluir la librería # include <time.h >
Se pueden usar las funciones gettimeofday() y settimeofday().
El argumento ***t*** es una struct timeval.

```
struct timeval {
        time_t      tv_sec;     /* seconds */
        suseconds_t tv_usec;    /* microseconds */
};
```

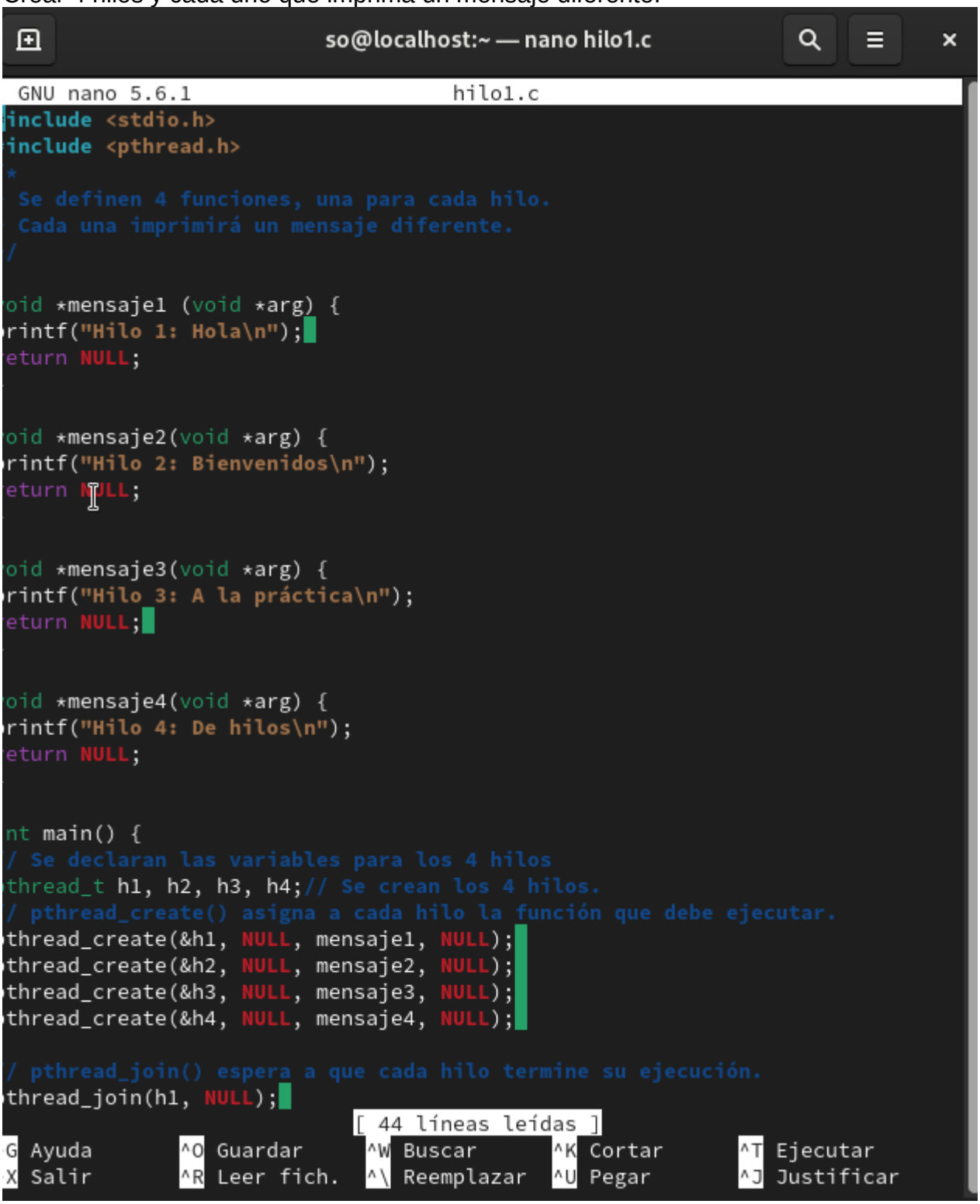
3.4 Cálculo de tiempo de ejecución

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <pthread.h>

struct timeval tinicio , tfin ;
double promedio = 0.0;

void * hilo ( void * arg )
{
        gettimeofday (& tfin , NULL ) ;
        int tfin_seg= tfin.tv_sec;
        int tini_seg = tinicio.tv_sec;
        promedio += ( tfin_seg - tini_seg);
}
int main ()
{
        int i = 0;
        pthread_t h ;
        for ( i = 0 ; i < 100000 ; i ++ )
        {
        gettimeofday (& tinicio , NULL );
        pthread_create (& h, NULL , hilo , NULL) ;
        pthread_join (h, NULL);
        }
printf ("El tiempo de creación de los %d hilos es %f segundos\n", i , promedio);
}
```

Figura 4. Script de ejemplo para calcular el tiempo de ejecución de un programa

Informe

1. Crear 4 hilos y cada uno que imprima un mensaje diferente.



```
[so@localhost ~]$ nano hilo1.c
[so@localhost ~]$ gcc -pthread hilo1.c -o hilo1
[so@localhost ~]$ ./hilo1
Hilo 1: Hola
Hilo 2: Bienvenidos
Hilo 4: De hilos
Hilo 3: A la práctica
```

2. Crear 5 hilos y que cada uno pase un parámetro numérico a una función que devolverá la suma de este número con un valor entero declarado dentro de la función.

so@localhost:~ — nano ejercicio2.c

GNU nano 5.6.1 ejercicio2.c Modificado

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

void *sumaNumero(void *param) {
    int numero = *(int *)param;    // Convertir el puntero a entero
    int valorInterno = 10;         // Valor dentro de la función
    int resultado = numero + valorInterno;

    printf("Hilo con número %d: %d + %d = %d\n", numero, numero, valorInterno>
    pthread_exit(NULL);
}

int main() {
    pthread_t hilos[5];
    int parametros[5] = {1, 2, 3, 4, 5};

    for (int i = 0; i < 5; i++) {
        pthread_create(&hilos[i], NULL, sumaNumero, (void *)&parametros[i]);
    }

    for (int i = 0; i < 5; i++) {
        pthread_join(hilos[i], NULL);
    }

    printf("Todos los hilos han finalizado correctamente.\n");
    return 0;
}
```

```
[so@localhost ~]$ gcc -pthread ejercicio2.c  -o ejercicio2
[so@localhost ~]$ ./ejercicio2
Hilo con número 1: 1 + 10 = 11
Hilo con número 2: 2 + 10 = 12
Hilo con número 3: 3 + 10 = 13
Hilo con número 4: 4 + 10 = 14
Hilo con número 5: 5 + 10 = 15
Todos los hilos han finalizado correctamente.
[so@localhost ~]$
```

3. Utilizando estructuras muestre el resultado del producto de un escalar por valores dentro de una matriz.

so@localhost:~ — nano ejercicio3.c

GNU nano 5.6.1 ejercicio3.c Modificad

```c
include <stdio.h>
include <pthread.h>

define N 3   // Tamaño de la matriz (3x3)

/ Estructura para enviar datos a los hilos
truct Datos {
    int fila[N];   // Fila de la matriz
    int escalar;   // Escalar para multiplicar
    int indice;    // Número de fila (solo para imprimir)
;

/ Función que ejecuta cada hilo
oid *multiplicarFila(void *param) {
    struct Datos *dato = (struct Datos *)param;
    printf("Hilo %d - Resultado: ", dato->indice + 1);

    for (int i = 0; i < N; i++) {
        int resultado = dato->fila[i] * dato->escalar;
        printf("%d ", resultado);
    }

    printf("\n");
    pthread_exit(NULL);


nt main() {
    pthread_t hilos[N];
    struct Datos datosHilo[N];

    int matriz[N][N] = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}
    };

    int escalar = 3;
```

```c
int main() {
    pthread_t hilos[N];
    struct Datos datosHilo[N];

    int matriz[N][N] = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}
    };

    int escalar = 3;

    // Crear hilos, uno por cada fila
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            datosHilo[i].fila[j] = matriz[i][j];
        }
        datosHilo[i].escalar = escalar;
        datosHilo[i].indice = i;

        pthread_create(&hilos[i], NULL, multiplicarFila, (void *)&datosHilo[i]
    }

    // Esperar que todos los hilos terminen
    for (int i = 0; i < N; i++) {
        pthread_join(hilos[i], NULL);
    }

    printf("Multiplicación completada.\n");
    return 0;
}
```

```
[so@localhost ~]$ nano ejercicio3.c
[so@localhost ~]$ gcc -pthread ejercicio3.c  -o ejercicio3
[so@localhost ~]$ ./ejercicio3
Hilo 1 - Resultado: 3 6 9
Hilo 2 - Resultado: 12 15 18
Hilo 3 - Resultado: 21 24 27
Multiplicación completada.
[so@localhost ~]$
```

4. Escribir código para determinar el resultado del tiempo de ejecución de 1 millón, 2 millones, y 3 millones de hilos. Tomar los tiempos en microsegundos.

so@localhost:~ — nano ejercicio4.c

GNU nano 5.6.1 ejercicio4.c Modificado

```c
#include <stdio.h>
#include <pthread.h>
#include <sys/time.h>

#define N 1000000  // número de operaciones simuladas

void *tarea(void *param) {
    long limite = *(long *)param;
    double suma = 0;
    for (long i = 0; i < limite; i++) {
        suma += i * 0.000001;  // operación simple
    }
    pthread_exit(NULL);
}

long medirTiempo(long operaciones) {
    pthread_t hilo;
    struct timeval inicio, fin;

    gettimeofday(&inicio, NULL);
    pthread_create(&hilo, NULL, tarea, (void *)&operaciones);
    pthread_join(hilo, NULL);
    gettimeofday(&fin, NULL);

    long tiempo = (fin.tv_sec - inicio.tv_sec) * 1000000L + (fin.tv_usec - in>
    return tiempo;
}

int main() {
    long tiempos[3];
    long valores[3] = {1000000, 2000000, 3000000};

    printf("Medición del tiempo de ejecución:\n");

    for (int i = 0; i < 3; i++) {
        tiempos[i] = medirTiempo(valores[i]);
        printf("→ %ld operaciones simuladas: %ld microsegundos\n", valores[i]>
    }
```

GNU nano 5.6.1 ejercicio4.c Modificado

```c
void *tarea(void *param) {
    long limite = *(long *)param;
    double suma = 0;
    for (long i = 0; i < limite; i++) {
        suma += i * 0.000001;  // operación simple
    }
    pthread_exit(NULL);
}

long medirTiempo(long operaciones) {
    pthread_t hilo;
    struct timeval inicio, fin;

    gettimeofday(&inicio, NULL);
    pthread_create(&hilo, NULL, tarea, (void *)&operaciones);
    pthread_join(hilo, NULL);
    gettimeofday(&fin, NULL);

    long tiempo = (fin.tv_sec - inicio.tv_sec) * 1000000L + (fin.tv_usec - in>
    return tiempo;
}

int main() {
    long tiempos[3];
    long valores[3] = {1000000, 2000000, 3000000};

    printf("Medición del tiempo de ejecución:\n");

    for (int i = 0; i < 3; i++) {
        tiempos[i] = medirTiempo(valores[i]);
        printf("→ %ld operaciones simuladas: %ld microsegundos\n", valores[i]>
    }

    printf("\nEjecución completada.\n");
    return 0;
}
```

^G Ayuda ^O Guardar ^W Buscar ^K Cortar ^T Ejecutar
^X Salir ^R Leer fich. ^\ Reemplazar ^U Pegar ^J Justificar

```
[so@localhost ~]$ nano ejercicio4.c
[so@localhost ~]$ gcc -pthread ejercicio4.c  -o ejercicio4
[so@localhost ~]$ ./ejercicio4
Medición del tiempo de ejecución:
→ 1000000 operaciones simuladas: 2217 microsegundos
→ 2000000 operaciones simuladas: 3561 microsegundos
→ 3000000 operaciones simuladas: 5462 microsegundos

Ejecución completada.
[so@localhost ~]$
```