



	<p align="center">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		
Aprobación: 2022/03/01	Código: GUIA-PRLD-001	Página: 1

GUÍA DE LABORATORIO

INFORMACIÓN BÁSICA					
ASIGNATURA:	Laboratorio de Programación de Sistemas				
TÍTULO DE LA PRÁCTICA:	Programación concurrente con threads				
NÚMERO DE PRÁCTICA:	Guía 09	AÑO LECTIVO:	2023	NRO. SEMESTRE:	A
TIPO DE PRÁCTICA:	INDIVIDUAL				
	GRUPAL	X	MÁXIMO DE ESTUDIANTES	5	
FECHA INICIO:	26/07/2023	FECHA FIN:	03/08/2023	DURACIÓN:	4 H
RECURSOS A UTILIZAR: Consideraciones: Utilizar el Sistema Operativo de Linux en cualquiera de sus versiones					
DOCENTE(s): Mg. Edith Giovanna Cano Mamani					

OBJETIVOS/TEMAS Y COMPETENCIAS	
OBJETIVOS: La presente práctica de laboratorio tiene como objetivo utilizar los conceptos de programación concurrente y usar el estandar POSIX Threads .	
TEMAS: Programación concurrente Threads	
COMPETENCIAS	C.c: Diseña responsablemente sistemas componentes o procesos para satisfacer necesidades dentro de restricciones realistas, económicas, medioambientales, sociales, políticas, éticas, de salud, de seguridad, manufacturación y sostenibilidad
	C.m: Construye responsablemente soluciones siguiendo un proceso adecuado llevando a cabo las pruebas ajustadas a los recursos disponibles del cliente.
	C.p: Aplica la forma flexible, técnicas, métodos, principios, normas, estándares y herramientas de ingeniería necesarias para la construcción de software e implementación de sistemas de información.

CONTENIDO DE LA GUÍA

	<p align="center">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		
Aprobación: 2022/03/01	Código: GUIA-PRLD-001	Página: 2

I. MARCO CONCEPTUAL

INTRODUCCIÓN

Una librería o paquete de threads permite escribir programas con varios puntos simultáneos de ejecución, sincronizados a través de memoria compartida. Sin embargo, la programación con hilos introduce nuevas dificultades. La programación concurrente tiene técnicas y problemas que no ocurren en la programación secuencial. Algunos problemas son sencillos (por ejemplo, el inter bloqueo) pero algunos otros aparecen como generalizaciones al rendimiento de la aplicación.

Un thread es un concepto sencillo: un simple flujo de control secuencia. Con un único thread existe en cualquier instante un único punto de ejecución. El programador no necesita aprender nada nuevo para programar un único thread. Sin embargo, cuando se tienen múltiples hilos en un programa significa que en cualquier instante el programa tiene múltiples puntos de ejecución, uno en cada uno de sus threads. El programador decide cuando y donde crear múltiples threads ayudándose de una librería o paquete en tiempo de ejecución.

En un lenguaje de alto nivel, las variables globales son compartidas por todos los threads del programa, esto es, los hilos leen y escriben en las mismas posiciones de memoria. El programador es el responsable de emplear los mecanismos de sincronización de la librería de hilos proporcionada por el lenguaje para garantizar que la memoria compartida se acceda de forma correcta por los hilos. Las facilidades proporcionadas por la librería de hilos que se utilice son conocidas como primitivas ligeras, lo que significa que las primitivas de:

- Creación
- Mantenimiento
- Sincronización y
- Destrucción

Son suficientemente económicas en esfuerzo para las necesidades del programador.

EL ESTANDAR POSIX THREADS

Uno de los problemas que se tenían al utilizar múltiples threads de ejecución es que hasta hace poco no existía un estándar para ello. La extensión POSIX 1c se aprobó en Junio de 1995. Con la adopción de un estándar POSIX para los hilos, se están haciendo más comunes las aplicaciones con threads

El estándar POSIX threads significa técnicamente el Pthreads especificado por el estándar formal internacional POSIX 1003.1c-1995. POSIX significa: Portable Operating System Interface. Todas las fuentes que empleen POSIX 1003.1c, POSIX.1c o simplemente Pthreads, deben incluir el archivo de encabezado pthread.h con la directiva:

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

Ya que pthread es una librería POSIX, se podrán portar los programas hechos con ella a cualquier sistema operativo POSIX que soporte threads. Por tanto, para crear programas que hagan uso de la librería pthread.h necesitamos en primer lugar la librería en sí. Esta viene en la mayoría de las distribuciones de LINUX y si no es así, se puede bajar de la internet.

Una vez que tenemos la librería instalada, deberemos compilar el programa y ligarlo con dicha librería en base al compilador que se utilice. La librería de hilos POSIX.1c debe ser la última librería especificada en la línea de comandos del compilador:

\$cc ... -lpthread

Por ejemplo, la forma más usual de hacer lo anterior, si estamos usando un compilador GNU como gcc, es con el comando:

```
$gcc prog_con_hilos.c -o prog_con_hilos_ejecutable -lpthread
```

En POSIX.1c todos los hilos de un proceso comparten las siguientes características:

- El espacio de direcciones
- El ID del proceso
- El ID DEL PROCESO PADRE
- El ID del proceso líder del grupo
- Los identificadores de usuario
- Los identificadores de grupo
- El directorio de trabajo raíz y actual
- La máscara de creación de archivos
- La tabla de descriptores de archivos
- El timer del proceso

Por otro lado, cada hilo tiene la siguiente información especificar:

- Un identificador de hilo único
- La política de planificación y la prioridad
- Una variante errno por hilo
- Datos específicos por hilo
- Gestores de cancelación por hilo
- Máscara de señales por hilo



Las operaciones llevadas a cabo sobre un hilo son:

- Creación y destrucción
- Sincronización entre hilos
- Posibilidad de disponer para cada thread memoria local propia

LA CLASE THREAD

El thread clase representa un solo hilo de ejecución . Los hilos permiten que múltiples funciones se ejecuten simultáneamente.

Los subprocesos comienzan a ejecutarse inmediatamente después de la construcción del objeto de subproceso asociado (a la espera de cualquier retraso en la programación del sistema operativo), comenzando en la función de nivel superior proporcionada como argumento de constructor . El valor de retorno de la función de nivel superior se ignora y si termina lanzando una excepción, se llama a `std::terminate` . La función de nivel superior puede comunicar su valor de retorno o una excepción a la

	<p style="text-align: center;">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p style="text-align: center;">Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLD-001</p>	<p>Página: 4</p>

persona que llama a través de `std::promise` o modificando variables compartidas (que pueden requerir sincronización, vea `std::mutex` y `std::atomic`).

`std::thread` objetos `std::thread` también pueden estar en el estado que no representa ningún hilo (después de la construcción predeterminada, mover, detach o join), y un hilo de ejecución puede no estar asociado con ningún objeto de thread (después de detach).

No hay dos objetos `std::thread` que puedan representar el mismo hilo de ejecución; `std::thread` no es CopyConstructible o CopyAssignable , aunque es MoveConstructible y MoveAssignable .

Tipos de miembros

<i>Tipo de miembro</i>	<i>Definition</i>
<code>native_handle_type</code>	implementation-defined

Clases de miembros

<i>Tipo de miembro</i>	<i>Definition</i>
<code>id</code>	representa elidde un hilo(clase de miembro público)

Funciones de los miembros


<i>Tipo de miembro</i>	<i>Definition</i>
(constructor)	construye un nuevo objeto de hilo (función de miembro público)
(destructor)	destruye el objeto del hilo,el hilo subyacente debe ser unido o separado(función de miembro público)
<code>operator=</code>	mueve el objeto del hilo (función de miembro público)

Observers

<i>Tipo de miembro</i>	<i>Definition</i>
<code>Joinable</code>	comprueba si el hilo es acoplable,es decir,si puede funcionar en un contexto paralelo(función de miembro público)
<code>get_id</code>	devuelve elidde del hilo (función de miembro público)
<code>native_handle</code>	devuelve el mango del hilo definido por la aplicación subyacente (función de miembro público)
<code>hardware_concurrency</code> [static]	devuelve el número de hilos concurrentes apoyados por la aplicación(función miembro público estático)

Operations

<i>Tipo de miembro</i>	<i>Definition</i>
<code>Join</code>	espera un hilo para terminar su ejecución (función de miembro público)
<code>Detach</code>	permite que el hilo se ejecute independientemente del mango del hilo(función de miembro público)
<code>Swap</code>	intercambia dos objetos de hilo (función de miembro público)

	<p style="text-align: center;">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p style="text-align: center;">Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLD-001</p>	<p>Página: 5</p>

LA CLASE MUTEX

Un mutex es un objeto bloqueable que está diseñado para señalar cuando las secciones críticas del código necesitan acceso exclusivo, evitando que otros subprocesos con la misma protección se ejecuten simultáneamente y accedan a las mismas ubicaciones de memoria.

Los objetos mutex proporcionan propiedad exclusiva y no admiten la recursividad (es decir, un hilo no debe bloquear un mutex que ya posee);

LA CLASE ATOMIC

Una operación atómica tiene dos propiedades clave que ayudan a usar varios subprocesos para manipular correctamente un objeto sin emplear bloqueos de exclusión mutua.

-Dado que una operación atómica es indivisible, una segunda operación atómica en el mismo objeto desde un subproceso diferente puede obtener el estado del objeto solo antes o después de la primera operación atómica.

-Según su argumento `memory_order`, una operación atómica establece requisitos de ordenación para la visibilidad de los efectos de otras operaciones atómicas del mismo subproceso. Por consiguiente, impide las optimizaciones del compilador que infringen los requisitos de ordenación.

En algunas plataformas no sería posible implementar realmente las operaciones atómicas para algunos tipos sin usar bloqueos mutex. Un tipo atómico no tiene bloqueos si ninguna operación atómica sobre ese tipo utiliza bloqueos.

II. EJERCICIO/PROBLEMA RESUELTO POR EL DOCENTE

EJERCICIO 1.

Crear un programa en C++ que cree un thread enviando una función que no realiza nada a través del uso de sus argumentos:

```
//codigo01.cpp
#include <iostream>
#include <thread>
using namespace std;

void func(){
}
int main(){
    thread th(func);
}
```

Para compilar dicho programa realice la siguiente instrucción:

```
$g++ -pthread -o codigo01 codigo01.cpp
```

Para ejecutarlo usar el siguiente comando:

```
$ ./codigo01
```

EJERCICIO 2.

Crear un programa en C++ que cree un thread enviando una función, junto con dos argumentos 1 y 5.7 a través del uso de los argumentos del constructor.

```
//codigo02.cpp
#include <iostream>
#include <thread>

using namespace std;

void func(int n, double m){
}
int main(){
    thread th(func, 1, 5.7);
}
```

Para compilar dicho programa realice la siguiente instrucción:

```
$g++ -pthread -o codigo02 codigo02.cpp
```

Para ejecutarlo usar el siguiente comando:

```
$ ./codigo02
```

EJERCICIO 3.

Crear un programa en C++ que cree un thread enviando una función, junto con dos argumentos 1 y 5.7 a través del uso de los argumentos del constructor, para evitar la aparición de errores utilizar la función joinable sobre el thread creado para utilizar el método join() sobre el thread creado

```
//codigo03.cpp
#include <iostream>
#include <thread>

using namespace std;

void func(int n, double m){
    cout << n << " " << m << endl;
}

int main(){
    thread th(func, 1, 5.7);

    if(th.joinable()) {
        th.join();
    }
}
```

Para compilar dicho programa realice la siguiente instrucción:

```
$g++ -pthread -o codigo03 codigo03.cpp
```

Para ejecutarlo usar el siguiente comando:

```
$ ./codigo03
```

EJERCICIO 4.

Crear un programa en C++ que cree un thread enviando una función, junto con una referencia a través del uso de los argumentos del constructor, para evitar la aparición de errores utilizar la función joinable sobre el thread creado para utilizar el método join() sobre el thread creado, la referencia enviada permitirá que el thread duerma y una vez despierto imprime B, dentro de la función principal también se imprime A.

```
//código04
#include <iostream>
#include <chrono>
#include <thread>

using namespace std;
```

```
void func(bool& empezar){
    while(!empezar){
        this_thread::sleep_for(chrono::milliseconds(1));
    }
    cout << "B" << endl;
}
```

```
int main(){
    bool empezar = false;
    thread th(func, ref(empezar));

    cout << "A" << endl;

    empezar = true;

    if(th.joinable()) {
        th.join();
    }
}
```

Para compilar dicho programa realice la siguiente instrucción:

```
$g++ -pthread -o codigo04 codigo04.cpp
```

Para ejecutarlo usar el siguiente comando:

```
$ ./codigo04
```

EJERCICIO 5.

Crear un programa en C++ que cree un thread e imprima la frase “Hola mundo!”

```
//codigo05.cpp
#include <iostream>
#include <pthread.h>

using namespace std;

#define NUM_THREADS 1

void* say_hello(void* args)
{
    cout << "Hola mundo!" << endl;
}

int main()
```



```
{
pthread_t tids[NUM_THREADS];
for(int i = 0; i < NUM_THREADS; ++i)
{
    int ret = pthread_create(&tids[i], NULL, say_hello, NULL);
    if (ret != 0)
    {
        cout << "pthread_create error: error_code=" << ret << endl;
    }
}
pthread_exit(NULL);
}
```

Para compilar dicho programa realice la siguiente instrucción:

```
$g++ -lpthread -o codigo05 codigo05.cpp
```

Para ejecutarlo usar el siguiente comando:

```
$ ./ codigo05
```

EJERCICIO 6.

Crear un programa en C++ que cree cinco thread e imprima la frase "Hola mundo!" cinco veces también

```
//codigo06.cpp
#include <iostream>
#include <pthread.h>

using namespace std;

#define NUM_THREADS 5

void* say_hello(void* args)
{
    cout << "Hola mundo!\n" << endl;
}

int main()
{
    pthread_t tids[NUM_THREADS];
    for(int i = 0; i < NUM_THREADS; ++i)
    {
```

```
int ret = pthread_create(&tids[i], NULL, say_hello, NULL);
if (ret != 0)
{
    cout << "pthread_create error: error_code=" << ret << endl;
}
}
pthread_exit(NULL);
}
```

Para compilar dicho programa realice la siguiente instrucción:

```
$g++ -lpthread -o codigo06 codigo06.cpp
```

Para ejecutarlo usar el siguiente comando:

```
$ ./codigo06
```

EJERCICIO 7.

Crear un programa en C++ que cree cinco thread e imprima la frase “Hola mundo!” junto con un parámetro ID, este parámetro es el parámetro enviado a través de la función principal, realizar esta acción cinco veces conforme la creación de los hilos

```
//codigo07.cpp
#include <iostream>
#include <cstdlib>
#include <pthread.h>

using namespace std;

#define NUM_THREADS 5

void *PrintHello(void *threadid)
{
    int tid = *((int*)threadid);
    cout << "Hola mundo! parametro ID, " << tid << endl;
    pthread_exit(NULL);
}

int main ()
{
    pthread_t threads[NUM_THREADS];
    int indexes[NUM_THREADS];
    int rc;
    int i;
    for( i=0; i < NUM_THREADS; i++){
```

```
cout << "main() : parametro, " << i << endl;
indexes[i] = i;
rc = pthread_create(&threads[i], NULL,
    PrintHello, (void *)&(indexes[i]));
if (rc){
    cout << "Error: error_code," << rc << endl;
    exit(-1);
}
}
pthread_exit(NULL);
}
```

Para compilar dicho programa realice la siguiente instrucción:

```
$g++ -lpthread -o codigo07 codigo07.cpp
```

Para ejecutarlo usar el siguiente comando:

```
$ ./codigo07
```

EJERCICIO 8.

Crear un programa en C++ que cree cinco thread e imprima la frase “Este es el mensaje” este mensaje es creado utilizando una estructura y esta estructura es enviada como parámetro al constructor del thread, realizar esta acción cinco veces conforme la creación de los hilos

```
//codigo08.cpp
#include <iostream>
#include <cstdlib>
#include <pthread.h>

using namespace std;

#define NUM_THREADS 5

struct thread_data{
    int thread_id;
    char *message;
};

void *PrintHello(void *threadarg)
{
    struct thread_data *my_data;

    my_data = (struct thread_data *) threadarg;
```

```
cout << "Thread ID : " << my_data->thread_id ;
cout << " Mensaje : " << my_data->message << endl;

pthread_exit(NULL);
}

int main ()
{
    pthread_t threads[NUM_THREADS];
    struct thread_data td[NUM_THREADS];
    int rc;
    int i;

    for( i=0; i < NUM_THREADS; i++){
        cout <<"main() : creando un hilo, " << i << endl;
        td[i].thread_id = i;
        td[i].message = "Este es el mensaje";
        rc = pthread_create(&threads[i], NULL,
                           PrintHello, (void *)&td[i]);

        if (rc){
            cout << "Error:no es posible crear el hilo," << rc << endl;
            exit(-1);
        }
    }
    pthread_exit(NULL);
}
```

Para compilar dicho programa realice la siguiente instrucción:

```
$ g++ -Wno-write-strings codigo08.cpp -lpthread -o codigo08
```

Para ejecutarlo usar el siguiente comando:

```
$ ./codigo08
```

EJERCICIO 9.

Crear un programa en C++ que cree cinco thread e imprima el parámetro que se está enviando, haciendo que este espere haciendo que este thread duerma según el orden de creación, finalmente que muestre un mensaje de salida utilizando dentro de la función join()

```
//codigo09.cpp
#include <iostream>
#include <cstdlib>
#include <pthread.h>
#include <unistd.h>
```

```
using namespace std;

#define NUM_THREADS    5

void *wait(void *t)
{
    int i;
    long tid;

    tid = (long)t;

    sleep(1);
    cout << "El thread esta durmiendo" << endl;
    cout << "Thread con id : " << tid << " ...saliendo " << endl;
    pthread_exit(NULL);
}

int main ()
{
    int rc;
    int i;
    pthread_t threads[NUM_THREADS];
    pthread_attr_t attr;
    void *status;

    // joinable
    pthread_attr_init(&attr);
    pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);

    for( i=0; i < NUM_THREADS; i++){
        cout << "main() : creando el thread, " << i << endl;
        rc = pthread_create(&threads[i], NULL, wait, (void *) (intptr_t) i );
        if (rc){
            cout << "Error: inhabilitado para crear un thread," << rc << endl;
            exit(-1);
        }
    }

    //
    pthread_attr_destroy(&attr);
    for( i=0; i < NUM_THREADS; i++){
        rc = pthread_join(threads[i], &status);
        if (rc){
            cout << "Error:inhabilitado para join," << rc << endl;
        }
    }
}
```

```
    exit(-1);
}
cout << "Main: id del thread completado :" << i ;
cout << " saliendo con el estado :" << status << endl;
}

cout << "Main: programas saliendo." << endl;
pthread_exit(NULL);
}
```

Para compilar dicho programa realice la siguiente instrucción:

```
$ g++ -Wno-write-strings codigo09.cpp -lpthread -o codigo09
```

Para ejecutarlo usar el siguiente comando:

```
$ ./codigo09
```

EJERCICIO 10.

Crear un programa en C++ que muestre como es la programación multihilo, para realizar tal acción se creará tres hilos, cada uno de ellos con tres funciones diferentes que les permitirán crear tres threads diferentes una vez creadas se utilizar la función join sobre ellas

```
// codigo10.cpp
#include <iostream>
#include <thread>
using namespace std;

// Una función ficticia
void foo(int Z)
{
    for (int i = 0; i < Z; i++) {
        cout << "Hilo usando función"
              " puntero como objeto invocable\n";
    }
}

// A llamada a objeto
class thread_obj {
public:
    void operator()(int x)
    {
        for (int i = 0; i < x; i++)
            cout << "Hilo usando función"
                  " Objeto como invocable\n";
    }
}
```

```
}  
};  
  
int main()  
{  
    cout << "Hilos 1 , 2 y 3 "  
        "operando independientemente" << endl;  
  
    // Este hilo es iniciado usando  
    // la función puntero como invocable  
    thread th1(foo, 3);  
  
    // Este hilo es iniciado usando  
    // la función objeto como invocable  
    thread th2(thread_obj(), 3);  
  
    // Define la expresión Lambda  
    auto f = [](int x) {  
        for (int i = 0; i < x; i++)  
            cout << "Hilo usando lambda"  
                " expresión como invocable\n";  
    };  
  
    // Este thread es invocable usndo  
    // la expresión lamda como invocable  
    thread th3(f, 3);  
  
    // Espera a que terminen los hilos  
    // Espera a que termine el hilo t1  
    th1.join();  
  
    // Espera a que termine el hilo t2  
    th2.join();  
  
    // Espera a que termine el hilo t3  
    th3.join();  
  
    return 0;  
}  
Para compilar dicho programa realice la siguiente instrucción:  
$g++ -lpthread -o codigo10 codigo10.cpp
```

Para ejecutarlo usar el siguiente comando:

```
$ ./codigo10
```

EJERCICIO 11.

Crear un programa en C++ que muestre como se protege una variable compartida utilizando hilos y mutex, el programa lanza dos hilos los cuales ejecutan la misma función, la cual es realizar 1000 incrementos unitarios a la variable compartida

```
//codigo11.cpp
#include <iostream>
#include <thread>
#include <mutex>

int counter;
std::mutex mtx;

void increment() {
    for(int i=0; i<1000; ++i) {
        mtx.lock();
        counter++;
        mtx.unlock();
    }
}

int main() {
    auto start = std::chrono::high_resolution_clock::now();
    for(int i = 0; i < 10000; ++i) {
        counter = 0;
        std::thread t1 = std::thread(increment);
        std::thread t2 = std::thread(increment);
        t1.join(); t2.join();
        if(counter != 2000) {
            std::cout << i << ": " << counter << std::endl;
        }
    }
    auto stop = std::chrono::high_resolution_clock::now();
    std::cout << "tiempo= " <<
        std::chrono::duration_cast<std::chrono::milliseconds>(stop-start).
        count() << " ms" << std::endl;
    return 0;
}
```

Para compilar dicho programa realice la siguiente instrucción:

```
$g++ -lpthread -o codigo11 codigo11.cpp
```

Para ejecutarlo usar el siguiente comando:

\$./codigo11

EJERCICIO 12.

Crear un programa en C++ que muestre como se protege una variable compartida utilizando hilos y variables atómicas, el programa lanza dos hilos los cuales ejecutan la misma función, la cual es realizar 1000 incrementos unitarios a la variable compartida

```
//codigo12
#include <iostream>
#include <thread>
#include <atomic>

std::atomic<int> counter;

void increment() {
    for(int i=0; i<1000; ++i) {
        counter++;
    }
}



int main() {
    auto start = std::chrono::high_resolution_clock::now();
    for(int i = 0; i < 10000; ++i) {
        counter = 0;
        std::thread t1 = std::thread(increment);
        std::thread t2 = std::thread(increment);
        t1.join(); t2.join();
        if(counter != 2000) {
            std::cout << i << ": " << counter << std::endl;
        }
    }
    auto stop = std::chrono::high_resolution_clock::now();
    std::cout << "tiempo= " <<
        std::chrono::duration_cast<std::chrono::milliseconds>(stop-start).
        count() << " ms" << std::endl;
    return 0;
}
```

Para compilar dicho programa realice la siguiente instrucción:

```
$g++ -lpthread -o codigo12 codigo12.cpp
```

Para ejecutarlo usar el siguiente comando:

```
$ ./codigo12
```

	<p align="center">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		
Aprobación: 2022/03/01	Código: GUIA-PRLD-001	Página: 18

III. EJERCICIOS/PROBLEMAS PROPUESTOS

EJERCICIO 1.

Crear un programa en C++ que permita sumar todos los elementos de un arreglo utilizando hilos y luego imprima ese resultado

EJERCICIO 2.

Crear un programa en C++ utilizando threads que muestre cual es el valor máximo de un arreglo muy grande

EJERCICIO 3.

Crear un programa en C++ utilizando threads que muestre las veces que se repite una cadena de texto

EJERCICIO 4.

Crear un programa en C++ que utilizando threads implemente la búsqueda binaria

IV. CUESTIONARIO

- 1) ¿Existe una llamada abort() cuando se crean algún thread? ¿Cuál es su finalidad
- 2) ¿Describa el problema de la cena de los filósofos dentro de la programación concurrente?

V. REFERENCIAS Y BIBLIOGRAFÍA RECOMENDADAS:

- [1] P.J. Deitel and H.M. Deitel, "Cómo Programar en C++", México, Ed. Pearson Educación, 2009
[2] B. Stroustrup, "El Lenguaje de Programación C++", Madrid, Adisson Pearson Educación, 2002
[3] B. Eckel, "Thinking in C++", Prentice Hall, 2000

TÉCNICAS E INSTRUMENTOS DE EVALUACIÓN

TÉCNICAS:
Redacción escrita

INSTRUMENTOS:
Rúbrica

CRITERIOS DE EVALUACIÓN

Criterio de evaluación / Niveles de expectativa	Primer Criterio: Insatisfecho	Segundo Criterio: En Proceso	Tercer Criterio: Satisfactorio	Cuarto Criterio: Sobresaliente
Informe Puntaje máx: 08	El informe es difícil de leer y no cuenta con la información pedida.	El informe incluye la mayor parte de la información solicitada, pero cuesta	El informe incluye la información solicitada y es comprensible.	El informe está claramente detallado e incluye toda la información solicitada

	Puntaje: 0	comprenderlo. Puntaje: 2	Puntaje: 4	Puntaje: 8
Cantidad de información Puntaje máx: 06	Uno o más de los temas no han sido tratados Puntaje: 0	Todos los temas han sido tratados y la mayor parte de las preguntas han sido contestadas, como mínimo con una frase. Puntaje: 2	Todos los temas han sido tratados y la mayor parte de las preguntas han sido contestadas, como mínimo dos frases cada una. Puntaje: 4	Todos los temas han sido tratados y todas las preguntas han sido contestadas con tres o más frases cada una. Puntaje: 6
Calidad de información Puntaje máx: 06	La información tiene poco que ver con el tema principal. Puntaje: 0	La información está relacionada con el tema principal. Puntaje: 2	La información está claramente relacionada con el tema principal. Puntaje: 3	La información está claramente relacionada con el tema principal y presenta otros ejemplos. Puntaje: 6