


	<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:	TECNOLOGIA DE OBJETOS				
TÍTULO DE LA PRÁCTICA:	Threads en C++				
NÚMERO DE PRÁCTICA:	06	AÑO LECTIVO:	2023-B	NRO. SEMESTRE:	
TIPO DE PRÁCTICA:	INDIVIDUAL				
	GRUPAL	X	MÁXIMO DE ESTUDIANTES	2	
FECHA INICIO:	20/11/2023	FECHA FIN:	29/07/2023	DURACIÓN:	50 min
RECURSOS A UTILIZAR: <i>Código C++, laboratorio PC, framework Qt open-source versión 5 o superior, y/o visual studio, presentaciones, explicación por casuística.</i>					
DOCENTE(s): <i>Mg. Karen Melissa Quispe Vergaray</i>					

OBJETIVOS/TEMAS Y COMPETENCIAS	
OBJETIVOS: <ul style="list-style-type: none"> Conocer el uso de Threads (hilos). 	
TEMAS: <ul style="list-style-type: none"> Threads en C++. Mutex en C++. 	
COMPETENCIAS	<ul style="list-style-type: none"> Diseña responsablemente sistemas, componentes o procesos para satisfacer necesidades dentro de restricciones realistas: económicas, medio ambientales, sociales, políticas, éticas, de salud, de seguridad, manufacturación y sostenibilidad. Aplica de forma flexible técnicas, métodos, principios, normas, estándares y herramientas de ingeniería necesarias para la construcción de software e implementación de sistemas de información.

	<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: 2</p>

--	--

CONTENIDO DE LA GUÍA

I. MARCO CONCEPTUAL

En la actualidad, los principales proveedores de CPU venden procesadores de varios núcleos. El uso de los CPU va desde servidores hasta computadoras de consumo, e incluso hoy en día en teléfonos inteligentes.

Si queremos aprovechar la potencia de todos los núcleos, debemos pensar en utilizar y/o escribir código para multiproceso (hilos/threads).

En el lenguaje C++ se pueden crear subprocesos individuales (hilos) utilizando una de las clases de encabezado:

- `<thread>` → librería creada desde C++11, tiene bases en el subprocesamiento pthreads.
- `<pthread.h>` → librería que implementa subprocesamiento al estilo UNIX, es ideal para entornos Linux, Unix, similares.

En el código orientado a multiproceso, se debe tener cuidado de que varios subprocesos no lean y escriban en el mismo segmento de datos al mismo tiempo, ocasionaría corrupción en la integridad; para manejar bloqueos, se puede usar:



- `<atomic>`, que otorga acceso atómico y seguro para el hilo en una transacción de datos.
- `<condition_variable>` o `<mutex>`, implementan un mecanismo de sincronización de hilos y restringe acceso a una transacción, el más utilizado es ***mutex***.

II. EJERCICIO/PROBLEMA RESUELTO POR EL DOCENTE

- a) Aplicación en C++ que simula la implementación de procesos paralelos por hilos, utilizando POSIX `<pthread.h>`

Necesarias las cabeceras:

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

	<p>UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p>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: 3</p>

```
void* procesoHilo(void *dato) {

    struct timespec tm = { 1,0 };

    while (1) {

        qDebug() << "proceso";
        pthread_delay_np(&tm);
    }
}
```

Bucle infinito, implementado en una función que apunta a una dirección de memoria. *timespec* especifica segundos y nanosegundos.

```
pthread_t proceso1;
pthread_t proceso2;
pthread_create(&proceso1, NULL, &procesoHilo, NULL);
pthread_create(&proceso2, NULL, &procesoHilo, NULL);
```

Se declaran los procesos (proceso1 y proceso2) ambos son punteros, y son creados con la instrucción “pthread_create”, su sintaxis:


`pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine) (void *), void *arg)`



```
pthread_join(proceso1, NULL);
pthread_join(proceso2, NULL);
```

“pthread_join” lanza hacia el CPU los procesos 1 y 2 paralelos, ambos con la ejecución de “procesoHilo”

```
19:05:31: Starting D:\n_UNSA\Tecnologia de Objetos\Pro
proceso
proceso
proceso
proceso
proceso
proceso
proceso
proceso
```

procesos1 y
procesos2



	<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>

b) Aplicación en C++ que simula la implementación de procesos paralelos por hilos, utilizando <thread>

Necesarios las cabeceras:

```
#include <thread>
#include <sstream>
#include <chrono>
#include <ctime>

void ExecuteThread(int id) {
    auto nowTime = std::chrono::system_clock::now();
    std::time_t sleepTime = std::chrono::system_clock::to_time_t(nowTime);
    // calcular la zona horaria local
    tm myLocalTime = *localtime(&sleepTime);

    qDebug() << "Thread: " << id << "sleep Time: " << std::ctime(&sleepTime); // << "\n";
    qDebug() << "Month: " << (myLocalTime.tm_mon)+1; // << "\n";
    qDebug() << "Day: " << myLocalTime.tm_mday; // << "\n";
    qDebug() << "Year: " << myLocalTime.tm_year; // << "\n";
    qDebug() << "Hours: " << myLocalTime.tm_hour; // << "\n";
    qDebug() << "Minutes: " << myLocalTime.tm_min; // << "\n";
    qDebug() << "Seconds: " << myLocalTime.tm_sec; // << "\n";

    // 3 segundos de espera antes de imprimir en consola
    std::this_thread::sleep_for(std::chrono::seconds(GetRandom(3)));
    nowTime = std::chrono::system_clock::now();
    sleepTime = std::chrono::system_clock::to_time_t(nowTime);
    qDebug() << "Thread " << id << " Awake Time : " << std::ctime(&sleepTime) << "\n";
}
```

La función “*ExecuteThread*” recibe un “id” como identificador de proceso y se imprime.

Desde la función principal:

```
std::thread th1 (ExecuteThread, 1);

th1.join();


std::thread th2 (ExecuteThread, 2);

th2.join();
```

“join()” lanza hacia el CPU los procesos th1 y th2 paralelos, ambos con la ejecución de “*ExecuteThread*”.

```
Thread: 1 sleep Time: Thu Nov 11 19:19:31 2021
Month: 11
Day: 11
Year: 121
Hours: 19
Minutes: 19
Seconds: 31
Thread 1 Awake Time : Thu Nov 11 19:19:34 2021

Thread: 2 sleep Time: Thu Nov 11 19:19:34 2021
Month: 11
Day: 11
Year: 121
Hours: 19
Minutes: 19
Seconds: 34
Thread 2 Awake Time : Thu Nov 11 19:19:34 2021
```



- c) Aplicación C++ de comparación de tiempo de procesamiento para encontrar números primos en un rango de 100000 (cien mil) números enteros, algoritmo con procesamiento secuencial y procesamiento paralelo (hilos).

Necesarias las cabeceras:

```
#include <thread>
#include <sstream>
#include <chrono>
#include <ctime>
#include <mutex>
```

```
//Función de procesamiento secuencial
void FindPrimes1(unsigned int start, unsigned int end,
std::vector<unsigned int>& vect){

    // bucle recorre sin tomar los numeros pares
    for(unsigned int x = start; x <= end; x += 2){
        for(unsigned int y = 2; y < x; y++){
            if((x % y) == 0){
                break;
            } else if((y + 1) == x){
                vect.push_back(x);
            }
        }
    }
}
```

//Funciones para procesamiento paralelo

```
std::mutex vectLock;
std::vector<unsigned int> primeVect;
```

```
void FindPrimes(unsigned int start,unsigned int end){
```

```
    // recorre los numeros sin tomar en cuenta los numeros pares
    for(unsigned int x = start; x <= end; x += 2){
```

```
        // If a modulus is 0 we know it isn't prime
```

```
        for(unsigned int y = 2; y < x; y++){
```

```
            if((x % y) == 0){
```

```
                break;
```

```
            } else if((y + 1) == x){
```

```
                vectLock.lock();
```

```
                primeVect.push_back(x);
```

```
                vectLock.unlock();
```

```
            }
```

```
        }
```

```
    }
```

```
}
```

bloquea la transacción para guardar el numero primo en el vector.
desbloquea la transacción.

```
void FindPrimesWithThreads(unsigned int start, unsigned int end,unsigned int numThreads){
```

```
    std::vector<std::thread> threadVect;
```

```
    //divide la cantidad de numeros a procesar para cada hilo.
```

```
    unsigned int threadSpread = end / numThreads;
```

```
    unsigned int newEnd = start + threadSpread - 1;
```

```
    // Create prime list for each thread
```

```
    for(unsigned int x = 0; x < numThreads; x++){
```

```
        threadVect.emplace_back(FindPrimes, start, newEnd);
```

```
        start += threadSpread;
```

```
        newEnd += threadSpread;
```

```
    }
```

```
    for(auto& t : threadVect){
```

```
        t.join();
```

```
    }
```

```
}
```

cantidad a procesar
(100000)

numero de hilos

distribuye para cada hilo el rango de numeros a encontrar numeros primos

guarda los numeros primos correspondientes

lanza los hilos a procesar al CPU

```
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);

    // SIN HILOS
    std::vector<unsigned int> primeVect;

    // capturar la fecha inicial completa
    int startTime = clock();

    // encontrar numeros primos sin hilos
    FindPrimesWithoutThreads(1, 1000000, primeVect);

    // capturar la fecha final de finalización
    int endTime = clock();

    // imprimir el tiempo transcurrido promedio
    qDebug() << "Execution Time : " << (endTime -
    startTime)/double(CLOCKS_PER_SEC) << endl;

    // CON HILOS
    // capturar la fecha inicial completa
    startTime = clock();

    FindPrimesWithThreads(1, 1000000, 8);

    // Get time after execution
    endTime = clock();

    // imprimir el tiempo transcurrido promedio
    qDebug() << "Execution Time : " << (endTime -
    startTime)/double(CLOCKS_PER_SEC) << endl;



}
```

Execution Time : 0.943 ← sin hilos

Execution Time : 0.313 ← con hilos

I. EJERCICIOS/PROBLEMAS PROPUESTOS

1. Manejo de múltiples threads en C++, usar mutex.

	<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: 8</p>

Como parte del proceso de aprendizaje del manejo de múltiples Threads en C++. El alumno debe implementar un programa que ejecute por lo menos 04 procesos (Threads) de forma concurrente sobre el siguiente escenario:

- Se deberá implementar una lista simple enlazada.
- La estructura de datos debe de soportar las operaciones de inserción, eliminación, búsqueda y modificación de valores (se debe de controlar las secciones críticas).
- Cada uno de los procesos que serán lanzados de forma paralela, deberán de realizar 10 operaciones de forma aleatoria, por lo que a cada uno de los Threads deberá ser asignado una tarea específica. Por ejemplo:
 - El *Thread 1* se encarga de eliminar solamente valores (10 números generados de forma aleatoria, mostrar el texto “eliminando: N” y el número que pudo eliminar, caso contrario mostrar: “No se eliminó N”).
 - El *Thread 2* se encarga de ir insertando elementos a la lista de forma aleatoria (genera 10 enteros y llama a la función insertar de la lista, mostrando “Insertando: N”).
 - El *Thread 3* irá consultando valores de forma aleatoria, de encontrarlos los muestra en pantalla “Buscado: N”; caso contrario mostrará: “No encontrado: N”.
 - El último *Thread 4* irá modificando los valores de ciertos elementos (tendrá que buscar un valor aleatorio y sumarle una cantidad), mostrar “Modificando N a M”, caso contrario mostrar “No se encontró”.

CUESTIONARIO

1. Revisar la procedencia de las librerías “thread” y “pthread.h”. Ventajas, desventajas, uso.
2. Investigar sobre <condition_variable>, que es similar a <mutex> para evita los bloqueos de acceso.

IV. REFERENCIAS Y BIBLIOGRAFÍA RECOMENDADAS:

- [1] Ceballos, F.J., programación orientada a Objetos con C++, 2da ed.,1997.
- [2] Robert Lafore, Object-Oriented programming in C++, fourth edition.
- [3] <http://www.w3big.com/es/cplusplus/cpp-multithreading.html>
- [4] https://www.cplusplus.com/reference/vector/vector/emplace_back/
- [5] C++ reference, online: <https://en.cppreference.com/w/>

TÉCNICAS E INSTRUMENTOS DE EVALUACIÓN

TÉCNICAS:

Observación y retroalimentación in-situ.

INSTRUMENTOS:

*Organización y resultado del trabajo en equipo.
 Desarrollo y sustentación de las actividades propuestas –grupal e individual.*

CRITERIOS DE EVALUACIÓN

- Revisión de librería STL para resolver problemas comunes.
- Revisión y aplicación de problemas de hilos usando C++.
- Diseñar y producir código en C++ para la solución de problemas relacionado al buen desempeño de recursos.