

Informe de Laboratorio 06

Tema: Threads en C++

		Nota
Estudiante	Escuela	Asignatura
Escobedo Ocaña Jorge Luis	Escuela Profesional de Ingeniería de Sistemas	Tecnología de Objetos Semestre: VI
Laboratorio	Tema	Duración
06	Threads en C++	02 horas
Semestre académico	Fecha de inicio	Fecha de entrega
2024 - B	31/10/2024	07/10/2024

1. Tarea

- Desarrollar los ejercicios presentados en la guía de laboratorio 06.
- Utilizar Git para evidenciar su trabajo.
- Enviar trabajo al profesor en un repositorio GitHub Privado, dándole permisos como colaborador.

2. Equipos, materiales y temas utilizados

- Sistema Operativo: Windows.
- Compilador: MinGW (GCC-6.3.0-1) o MSVC.
- Entorno de Desarrollo Integrado (IDE): Qt Creator.
- Versiones de C++: C++17 y C++20.
- Bibliotecas: Bibliotecas estándar de C++ (STL), Qt Widgets, Qt Core, Qt Network, Qt Multimedia.
- Documentación y Recursos: Documentación de Qt, Tutoriales en línea.
- Repositorio en GITHUB:
<https://github.com/Iuerges28/TO-LAB/tree/main/Lab6/>

3. Ejercicios Resueltos

a. Actividad 1

Aplicación en C++ que simula la implementación de procesos paralelos por hilos, utilizando POSIX

C/C++

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

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

```
struct timespec tm = {1, 0};

while (true) {
    qDebug() << "proceso";
    nanosleep(&tm, NULL);
}

return nullptr;
}

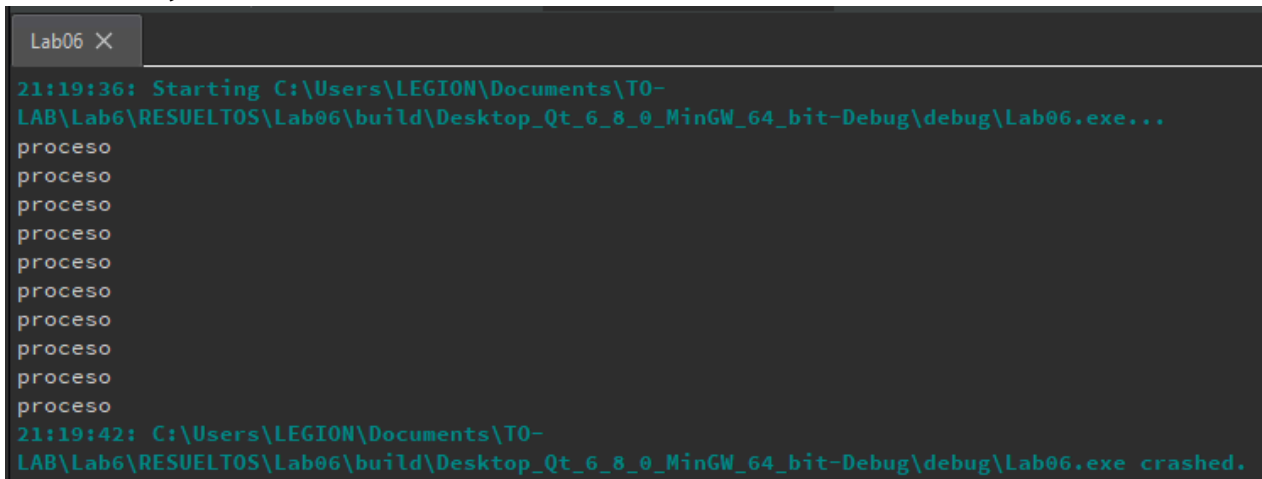
int main() {
    pthread_t proceso1;
    pthread_t proceso2;

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

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

    return 0;
}
```

Ejecución:



```
Lab06 X
21:19:36: Starting C:\Users\LEGION\Documents\T0-
LAB\Lab6\RESUELTOS\Lab06\build\Desktop_Qt_6_8_0_MinGW_64_bit-Debug\debug\Lab06.exe...
proceso
proceso
proceso
proceso
proceso
proceso
proceso
proceso
proceso
proceso
proceso
21:19:42: C:\Users\LEGION\Documents\T0-
LAB\Lab6\RESUELTOS\Lab06\build\Desktop_Qt_6_8_0_MinGW_64_bit-Debug\debug\Lab06.exe crashed.
```

b. Actividad 2

Aplicación en C++ que simula la implementación de procesos paralelos por hilos, utilizando.

C/C++

```
#include <thread>
#include <iostream>
#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);

    tm myLocalTime = *localtime(&sleepTime);

    std::cout << "Thread " << id << " Sleep Time: " << std::ctime(&sleepTime);
    std::cout << "Month: " << (myLocalTime.tm_mon) + 1 << "\n";
    std::cout << "Day: " << myLocalTime.tm_mday << "\n";
    std::cout << "Year: " << myLocalTime.tm_year + 1900 << "\n";
    std::cout << "Hours: " << myLocalTime.tm_hour << "\n";
    std::cout << "Minutes: " << myLocalTime.tm_min << "\n";
    std::cout << "Seconds: " << myLocalTime.tm_sec << "\n";

    std::this_thread::sleep_for(std::chrono::seconds(3));

    nowTime = std::chrono::system_clock::now();
    sleepTime = std::chrono::system_clock::to_time_t(nowTime);

    std::cout << "Thread " << id << " Awake Time: " << std::ctime(&sleepTime) <<
    "\n";
}

int main() {
    std::thread th1(ExecuteThread, 1);
    th1.join();

    std::thread th2(ExecuteThread, 2);
    th2.join();

    return 0;
}
```

Ejecución:

```

Lab6-e2 X
21:26:16: Starting C:\Users\LEGION\Documents\T0-LAB\Lab6\RESUELTOS\Lab6-
e2\build\Desktop_Qt_6_8_0_MinGW_64_bit-Debug\debug\Lab6-e2.exe...
Thread 1 Sleep Time: Thu Nov 07 21:26:16 2024
Month: 11
Day: 7
Year: 2024
Hours: 21
Minutes: 26
Seconds: 16
Thread 1 Awake Time: Thu Nov 07 21:26:19 2024

Thread 2 Sleep Time: Thu Nov 07 21:26:19 2024
Month: 11
Day: 7
Year: 2024
Hours: 21
Minutes: 26
Seconds: 19
Thread 2 Awake Time: Thu Nov 07 21:26:22 2024

21:26:22: C:\Users\LEGION\Documents\T0-LAB\Lab6\RESUELTOS\Lab6-
e2\build\Desktop_Qt_6_8_0_MinGW_64_bit-Debug\debug\Lab6-e2.exe exited with code 0
    
```

c. Actividad 3

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)

C/C++

```

#include <thread>
#include <sstream>
#include <chrono>
#include <ctime>
#include <mutex>
#include <vector>
#include <iostream>
#include <QCoreApplication>
#include <QtDebug>

void FindPrimes1(unsigned int start, unsigned int end, std::vector<unsigned
int>& vect) {
    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);
            }
        }
    }
}
    
```

```
    }  
    }  
}  
  
std::mutex vectLock;  
std::vector<unsigned int> primeVect;  
  
void FindPrimes(unsigned int start, unsigned int end) {  
    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) {  
                vectLock.lock();  
                primeVect.push_back(x);  
                vectLock.unlock();  
            }  
        }  
    }  
}  
  
void FindPrimesWithThreads(unsigned int start, unsigned int end, unsigned int  
numThreads) {  
    std::vector<std::thread> threadVect;  
    unsigned int threadSpread = end / numThreads;  
    unsigned int newEnd = start + threadSpread - 1;  
  
    for (unsigned int x = 0; x < numThreads; x++) {  
        threadVect.emplace_back(FindPrimes, start, newEnd);  
        start += threadSpread;  
        newEnd += threadSpread;  
    }  
  
    for (auto& t : threadVect) {  
        t.join();  
    }  
}  
  
int main(int argc, char *argv[]) {  
    QCoreApplication a(argc, argv);  
  
    std::vector<unsigned int> primeVect;  
    int startTime = clock();  
  
    FindPrimes1(1, 1000000, primeVect);  
    int endTime = clock();  
  
    std::cout << "Execution Time: " << (endTime - startTime) /  
double(CLOCKS_PER_SEC) << " seconds" << std::endl;
```

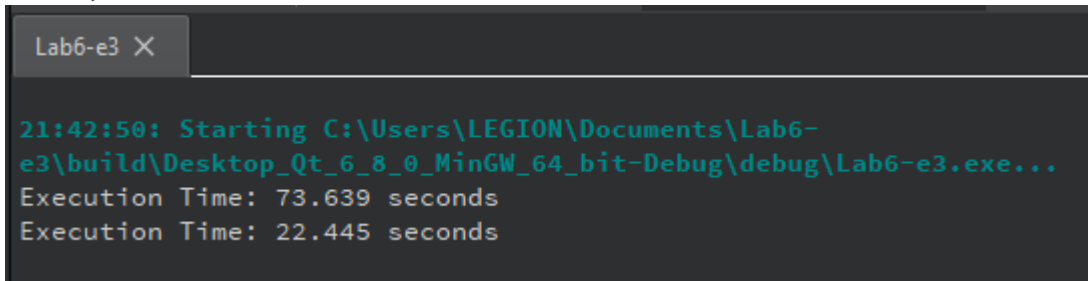
```

startTime = clock();
FindPrimesWithThreads(1, 1000000, 8);
endTime = clock();

std::cout << "Execution Time: " << (endTime - startTime) /
double(CLOCKS_PER_SEC) << " seconds" << std::endl;

return a.exec();
}
    
```

Ejecución



```

Lab6-e3 X
21:42:50: Starting C:\Users\LEGION\Documents\Lab6-
e3\build\Desktop_Qt_6_8_0_MinGW_64_bit-Debug\debug\Lab6-e3.exe...
Execution Time: 73.639 seconds
Execution Time: 22.445 seconds
    
```

4. Ejercicios Propuestos

a. Actividad 1

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

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:
 - o 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").
 - o 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").
 - o El Thread 3 irá consultando valores de forma aleatoria, de encontrarlos los muestra en pantalla "Buscado: N"; caso contrario mostrará: "No encontrado:

N".

o 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".

```
C/C++
#include <iostream>
#include <thread>
#include <mutex>
#include <memory>
#include <cstdlib>
#include <ctime>

using namespace std;

mutex listMutex;

struct Node {
    int value;
    shared_ptr<Node> next;
    Node(int val) : value(val), next(nullptr) {}
};

class LinkedList {
public:
    LinkedList() : head(nullptr) {}

    void insert(int value) {
        lock_guard<mutex> lock(listMutex);
        auto newNode = make_shared<Node>(value);
        newNode->next = head;
        head = newNode;
        cout << "Insertando: " << value << endl;
    }

    bool remove(int value) {
        lock_guard<mutex> lock(listMutex);
        auto current = head;
        shared_ptr<Node> previous = nullptr;
        while (current) {
            if (current->value == value) {
                if (previous) {
                    previous->next = current->next;
                } else {
                    head = current->next;
                }
                cout << "Eliminando: " << value << endl;
                return true;
            }
            previous = current;
            current = current->next;
        }
    }
};
```

```
        previous = current;
        current = current->next;
    }
    cout << "No se eliminó: " << value << endl;
    return false;
}

bool find(int value) {
    lock_guard<mutex> lock(listMutex);
    auto current = head;
    while (current) {
        if (current->value == value) {
            cout << "Buscado: " << value << endl;
            return true;
        }
        current = current->next;
    }
    cout << "No encontrado: " << value << endl;
    return false;
}

bool modify(int oldValue, int increment) {
    lock_guard<mutex> lock(listMutex);
    auto current = head;
    while (current) {
        if (current->value == oldValue) {
            int newValue = oldValue + increment;
            cout << "Modificando " << oldValue << " a " << newValue << endl;
            current->value = newValue;
            return true;
        }
        current = current->next;
    }
    cout << "No se encontró " << oldValue << endl;
    return false;
}

private:
    shared_ptr<Node> head;
};

void threadRemove(LinkedList& list) {
    for (int i = 0; i < 10; ++i) {
        int num = rand() % 100;
        list.remove(num);
    }
}

void threadInsert(LinkedList& list) {
    for (int i = 0; i < 10; ++i) {
        int num = rand() % 100;
```



```
        list.insert(num);
    }
}

void threadFind(LinkedList& list) {
    for (int i = 0; i < 10; ++i) {
        int num = rand() % 100;
        list.find(num);
    }
}

void threadModify(LinkedList& list) {
    for (int i = 0; i < 10; ++i) {
        int oldValue = rand() % 100;
        int increment = rand() % 10 + 1;
        list.modify(oldValue, increment);
    }
}

int main() {
    srand(static_cast<unsigned>(time(0)));
    LinkedList list;

    thread t1(threadRemove, ref(list));
    thread t2(threadInsert, ref(list));
    thread t3(threadFind, ref(list));
    thread t4(threadModify, ref(list));

    t1.join();
    t2.join();
    t3.join();
    t4.join();

    return 0;
}
```

Ejecución:

```
✓ ↗ 📄 ⚙️ 👤  
No se eliminó: 34  
No se eliminó: 71  
No se eliminó: 18  
No se eliminó: 72  
No se eliminó: 1  
No se eliminó: 16  
No se encontró 10  
No se encontró 94  
No se encontró 84  
No se encontró 78  
No se encontró 2  
No se encontró 39  
No se encontró 7  
No se encontró 10  
No se encontró 61  
No se encontró 67  
No encontrado: 15  
No encontrado: 99  
No encontrado: 49  
No encontrado: 19  
No encontrado: 22  
No encontrado: 16  
No encontrado: 91  
No se eliminó: 97  
Insertando: 70  
Insertando: 44  
Insertando: 9  
Insertando: 51  
Insertando: 50  
Insertando: 6  
Insertando: 46  
Insertando: 26  
Insertando: 90  
Insertando: 57  
No se eliminó: 89  
No se eliminó: 4  
No se eliminó: 69  
No encontrado: 40  
No encontrado: 59  
No encontrado: 85  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

5. Cuestionario

a. Revisar la procedencia de las librerías “thread” y “pthread.h”. Ventajas, desventajas, uso.

La biblioteca <thread> de C++ forma parte del estándar desde C++11 y facilita la programación de hilos de forma segura y portable, integrándose bien con el lenguaje y aprovechando técnicas como RAII para gestionar recursos automáticamente. Su principal ventaja es la simplicidad y la facilidad para implementar concurrencia en C++ sin depender de configuraciones específicas del sistema operativo.

Por otro lado, pthread.h (POSIX threads) es una biblioteca de bajo nivel en C, común en sistemas Linux y macOS, que ofrece un control más detallado sobre los hilos. Aunque permite funcionalidades avanzadas y mayor personalización, requiere un manejo manual de los recursos y no es portable a sistemas Windows. En proyectos modernos de C++, se prefiere <thread> por su sencillez y portabilidad, mientras que pthread.h es útil para aplicaciones de bajo nivel que requieren un control detallado de la concurrencia.

b. Investigar sobre <condition_variable>, que es similar a <mutex> para evita los bloqueos de acceso

La biblioteca <condition_variable> en C++ forma parte de la biblioteca estándar desde la versión C++11 y proporciona una forma de sincronización más avanzada que el simple uso de <mutex>. Mientras que un mutex se utiliza para bloquear y desbloquear recursos compartidos para evitar accesos simultáneos y posibles condiciones de carrera, las condition_variable son útiles cuando un hilo necesita esperar una condición específica antes de continuar su ejecución. Esta condición puede ser una señal enviada por otro hilo, lo que permite una sincronización más eficiente en ciertos escenarios, como la espera de eventos o el procesamiento de datos en múltiples hilos.

Una condition_variable trabaja de manera conjunta con un mutex. El hilo que desea esperar a que una condición sea verdadera utiliza un método como wait() en la condición variable. Este método libera temporalmente el mutex para permitir que otros hilos accedan a los recursos compartidos. Una vez que la condición se cumple (lo cual es comunicado por otro hilo), el hilo que estaba esperando se desbloquea y puede reintentar su tarea. Esto evita que el hilo espere de manera innecesaria, lo cual mejora la eficiencia, ya que los hilos no se bloquean de manera continua, sino que esperan activamente hasta que se cumpla la condición.

El uso de condition_variable ayuda a mejorar el rendimiento y la eficiencia de programas multihilo al permitir que los hilos se sincronicen de manera más dinámica y reactiva. Sin embargo, también introduce complejidad adicional en comparación con los mutex, ya que es necesario manejar adecuadamente las condiciones de espera y la notificación entre hilos. Los métodos típicos como notify_one() o notify_all() se usan para notificar a uno o

más hilos que la condición ha cambiado, permitiendo que continúen su ejecución. Aunque el uso de `condition_variable` es fundamental en ciertos patrones de programación, su manejo requiere precaución para evitar problemas como el "spurious wakeup" (despertar falso), lo cual hace que sea más adecuado en situaciones donde el control fino de las condiciones de espera sea necesario.

6. Conclusiones:

El laboratorio permitió aprender sobre la programación concurrente en C++ usando `pthread.h` y `thread`. Se destacó que `pthread.h` es útil en sistemas POSIX, pero `thread` es más simple y portátil para proyectos modernos. La sincronización de hilos con `mutex` y `lock_guard` es esencial para evitar condiciones de carrera al acceder a datos compartidos. El uso de `condition_variable` optimiza la espera de condiciones específicas sin bloquear innecesariamente los hilos. La comparación entre procesamiento secuencial y paralelo mostró mejoras en tiempo de ejecución, aunque la eficiencia depende de la tarea y el número de hilos utilizados.

7. Referencias:

- [1] J. A. Meyer and M. M. Koss, "C++ Templates: The Complete Guide," 2nd ed. Upper Saddle River, NJ, USA: Addison-Wesley, 2006.
- [2] B. Stroustrup, "The C++ Programming Language," 4th ed. Boston, MA, USA: Addison-Wesley, 2013.
- [3] J. W. McCormack, "Polymorphism in C++: An Overview," *Journal of Computer Science and Technology*, vol. 29, no. 3, pp. 471-482, 2014.
- [4] Qt Documentation, "Qt Core Module," [Online]. Available: <https://doc.qt.io/qt-5/qtcore-index.html> [Accessed: 28-Oct-2024].
- [5] Qt Documentation, "Containers and Data Structures," [Online]. Available: <https://doc.qt.io/qt-5/qtcore-containers.html> [Accessed: 28-Oct-2024].
- [6] Qt Documentation, "QVector Class," [Online]. Available: <https://doc.qt.io/qt-5/qvector.html> [Accessed: 27-Oct-2024].
- [7] Qt Documentation, "QList Class," [Online]. Available: <https://doc.qt.io/qt-5/qlist.html> [Accessed: 29-Oct-2024].
- [8] C++ reference, online: <https://en.cppreference.com/w/>
- [9] Qt, [Online]. Available: <https://www.qt.io/>
- [10] QT Creator Manual.
- [11] Ceballos, F.J., "Programación Orientada a Objetos con C++," 2da ed., 1997.
- [12] Robert Lafore, "Object-Oriented Programming in C++," 4th edition.