Por siempre responsables de lo que se ha cultivado

SISTEMAS OPERATIVOS

ACTIVIDAD 3: EJERCICIOS

EQUIPO:4

Profesor

Manuel Triana Vega

Integrantes

Ángel Alejandro Alcalá Aguilar
Romel Arenas Jiménez
ángel Christian Lopez
Adair Espinosa Estrada
Jose Emiliano Jauregui Guzman

06/04/2025

Introducción

El ejercicio 3 se pondrá en práctica los conceptos clave de los sistemas operativos relacionados con la gestión de procesos, la concurrencia, la creación y control de hilos, la identificación de regiones críticas y el uso de semáforos como mecanismo de sincronización.

A lo largo del ejercicio se trabaja con ejemplos en Java y C para observar el comportamiento de programas concurrentes. En primer lugar, se realiza la compilación y ejecución del programa Tuberia.java, analizando cómo se implementa la concurrencia mediante hilos y de qué forma se controla el acceso a los recursos compartidos. Posteriormente, se realizará la implementación del problema del barbero dormilón en lenguaje C, utilizando también hilos y semáforos, con el objetivo de comparar los enfoques y reforzar la comprensión práctica del manejo de procesos concurrentes en diferentes entornos de programación.

1. Con base en el material consultado en la unidad resuelve los ejercicios que se plantean acerca de los siguientes temas:

   - Procesos en los sistemas operativos
   - Concurrencia
   - Hilos
   - Región crítica
   - Semáforos

Parte I

1. Descarga e instala el compilador de Java, a partir de la versión 7.
2. Descarga e instala algún IDE de desarrollo para Java de tu elección, para compilar y correr programas.
3. Realiza la compilación y corrida del programa **Tuberia.java**.
4. Describe el código, haciendo énfasis en las partes en las que se realiza la concurrencia y cómo se maneja.
5. Utiliza las librerías **semaphore.h** y **pthread.h.** Ubícalas en el lugar de las librerías dentro del Sistema Operativo Linux.
6. Revisa el código **programa.c**. Describe el código y realiza al menos 2 corridas del programa.
7. Elabora un breve informe de los resultados, archivos y problemáticas que se desarrollan al correrlo.
8.

Parte 1

Código del supuesto Tuberia.java: El programa de “Tuberia.java” no existe dentro de nuestra plataforma de blackboard, lo atribuyo a un error de esta, un compañero publico en el foro de dudas sobre donde encontrar estos programas específicamente, pero no obtuvo respuesta, en la videollamada se mencionó que seria el problema que aparece en el video de plataforma, así que a continuación se muestra tanto el código recabado de github Camacho, P. (Productor). (20 de noviembre de 2016). Como su ejecución en VSC. Se adjuntan las explicaciones al final del código, después de su respectiva terminal.

```
import java.util.concurrent.Semaphore;
import java.util.PriorityQueue;

class ITV {
    private Semaphore semaforo;
    private PriorityQueue <Integer> listaCoches;
    private Integer tiempoTotal;

    public ITV() {
        semaforo = new Semaphore(1);
        listaCoches = new PriorityQueue <Integer>();
        tiempoTotal = 0;
    }

    public void nuevoCoche(Integer numeroCoche) {
        try {
            semaforo.acquire();
            listaCoches.add(numeroCoche);
            semaforo.release();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    public int terminarCoche(Integer tiempoParcial) {
        int coche=0;
        try {
```

```
            if (isCochesPendientes()) {
                semaforo.acquire();
                coche = listaCoches.poll();
                tiempoTotal += tiempoParcial;
                semaforo.release();
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        return coche;
    }

    public boolean isCochesPendientes() {
        return listaCoches.size() > 0;
    }

    public Integer getTiempoTotal () {
        return tiempoTotal;
    }

}

class Puesto extends Thread {
    private int identif;
    private ITV itv;
    private Integer tiempoPuesto;
```

```
    public Puesto(int identif, ITV itv) {
        this.identif = identif;
        this.itv = itv;
        this.tiempoPuesto=0;
    }

    public void run() {
        int retardo;
        int numeroCoche;
        while (itv.isCochesPendientes()) {
            try {
                retardo = (int) (Math.random() * 90 + 10);
                tiempoPuesto +=retardo;
                numeroCoche=itv.terminarCoche(retardo);
                sleep(retardo);
                System.out.println("El puesto " + identif + " ha revisado el coche " +
numeroCoche + " en un tiempo de " + retardo);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        System.out.println("Fin del puesto " + identif + ", que termina con un tiempo
parcial de " + tiempoPuesto);
    }
}

class Vehiculo extends Thread {
    private int identif;
```

```
    private ITV itv;

    public Vehiculo(int identif, ITV itv) {
        this.identif = identif;
        this.itv = itv;
    }

    public void run() {
        itv.nuevoCoche(identif);
    }
}

public class Principal {

    public static void main(String[] args) {
        int pueRandom = (int) (Math.random() * 4) + 1;
        int vehRandom = (int) (Math.random() * 30) + 20;
        ITV itv = new ITV();
        System.out.println(vehRandom + " Vehículos serán atendidos por " +
pueRandom + " puestos.");
        // Creación de vehículos
        Vehiculo[] v = new Vehiculo[vehRandom];
        for (int i = 0; i < vehRandom; i++) {
            v[i] = new Vehiculo(i + 1, itv);
            v[i].start();
        }
        // Creación de puestos
```

```
        Puesto[] p = new Puesto[pueRandom];
        for (int i = 0; i < pueRandom; i++) {
            p[i] = new Puesto(i + 1, itv);
            p[i].start();
        }

        // Se espera a que terminen todos los puestos
        for (int i = 0; i < pueRandom; i++) {
            try {
                p[i].join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        // Se espera a que terminen todos vehículos
        for (int i = 0; i < vehRandom; i++) {
            try {
                v[i].join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        // Se cierra la itv
        System.out.println("Se cierra la ITV con un tiempo acumulado de " +
itv.getTiempoTotal());
    }
}
```

Principal.java > Principal

```java

import java.util.concurrent.Semaphore;
import java.util.PriorityQueue;

class ITV {
    private Semaphore semaforo;
    private PriorityQueue <Integer> listaCoches;
    private Integer tiempoTotal;

    public ITV() {
        semaforo = new Semaphore(permits:1);
        listaCoches = new PriorityQueue <Integer>();
        tiempoTotal = 0;
    }

    public void nuevoCoche(Integer numeroCoche) {
        try {
            semaforo.acquire();
            listaCoches.add(numeroCoche);
            semaforo.release();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    public int terminarCoche(Integer tiempoParcial) {
        int coche=0;
        try {
            if (isCochesPendientes()) {
                semaforo.acquire();
                coche = listaCoches.poll();
                tiempoTotal += tiempoParcial;
                semaforo.release();
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        return coche;
    }

    public boolean isCochesPendientes() {
        return listaCoches.size() > 0;
    }

    public Integer getTiempoTotal () {
        return tiempoTotal;
    }

}
```

Principal.java > Principal

```java
class Puesto extends Thread {
    private int identif;
    private ITV itv;
    private Integer tiempoPuesto;

    public Puesto(int identif, ITV itv) {
        this.identif = identif;
        this.itv = itv;
        this.tiempoPuesto=0;
    }

    public void run() {
        int retardo;
        int numeroCoche;
        while (itv.isCochesPendientes()) {
            try {
                retardo = (int) (Math.random() * 90 + 10);
                tiempoPuesto +=retardo;
                numeroCoche=itv.terminarCoche(retardo);
                sleep(retardo);
                System.out.println("El puesto " + identif + " ha revisado el coche " + numeroCoche + " en un tiempo de " + retardo);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        System.out.println("Fin del puesto " + identif + ", que termina con un tiempo parcial de " + tiempoPuesto);
    }
}

class Vehiculo extends Thread {
    private int identif;
    private ITV itv;

    public Vehiculo(int identif, ITV itv) {
        this.identif = identif;
        this.itv = itv;
    }

    public void run() {
        itv.nuevoCoche(identif);
    }
}

public class Principal {
```

Principal.java > Principal

```java
class Vehiculo extends Thread {
    public Vehiculo(int identif, ITV itv) {
        this.itv = itv;
    }

    public void run() {
        itv.nuevoCoche(identif);
    }
}

public class Principal {

    Run | Debug
    public static void main(String[] args) {
        int pueRandom = (int) (Math.random() * 4) + 1;
        int vehRandom = (int) (Math.random() * 30) + 20;
        ITV itv = new ITV();
        System.out.println(vehRandom + " Vehículos serán atendidos por " + pueRandom + " puestos.");
        // Creación de vehículos
        Vehiculo[] v = new Vehiculo[vehRandom];
        for (int i = 0; i < vehRandom; i++) {
            v[i] = new Vehiculo(i + 1, itv);
            v[i].start();
        }
        // Creación de puestos
        Puesto[] p = new Puesto[pueRandom];
        for (int i = 0; i < pueRandom; i++) {
            p[i] = new Puesto(i + 1, itv);
            p[i].start();
        }

        // Se espera a que terminen todos los puestos
        for (int i = 0; i < pueRandom; i++) {
            try {
                p[i].join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        // Se espera a que terminen todos vehículos
        for (int i = 0; i < vehRandom; i++) {
            try {
                v[i].join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        // Se cierra la itv
        System.out.println("Se cierra la ITV con un tiempo acumulado de " + itv.getTiempoTotal());
    }
}
```

Evidencia de la primera ejecución en la terminal de Visual Studio:

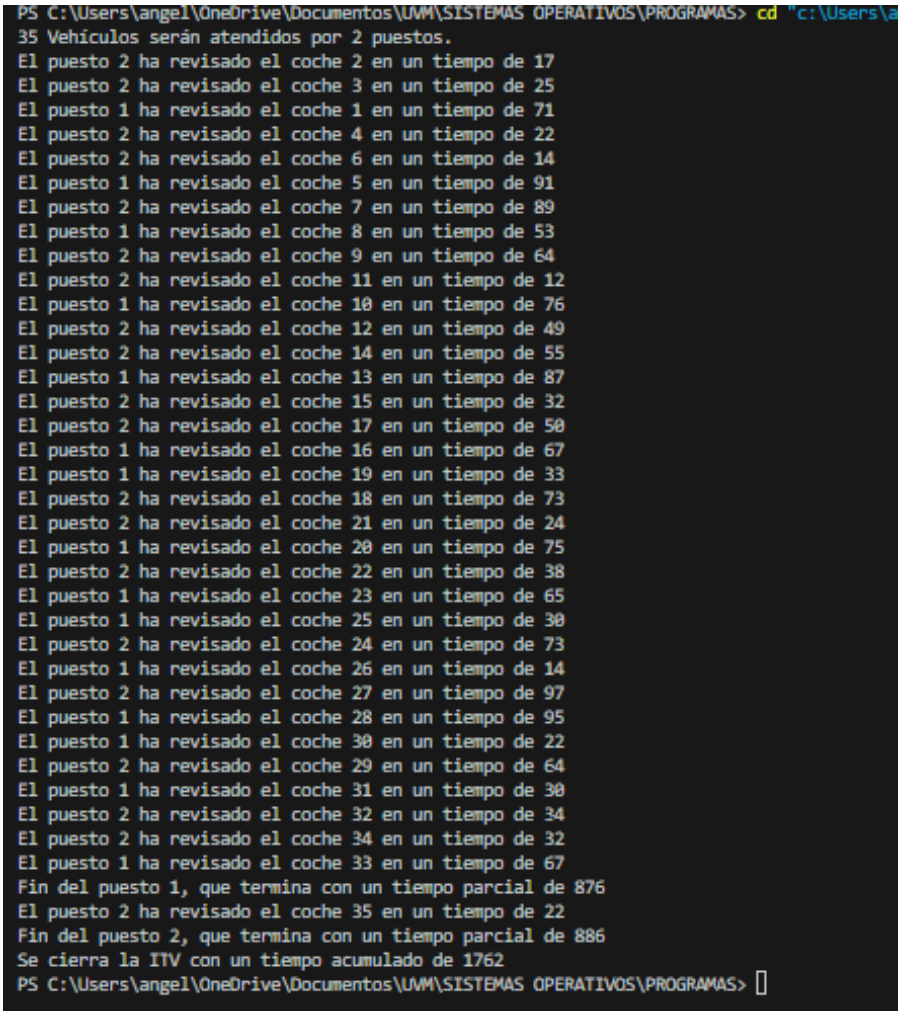
```
49 Vehículos serán atendidos por 1 puestos.
El puesto 1 ha revisado el coche 1 en un tiempo de 15
El puesto 1 ha revisado el coche 2 en un tiempo de 52
El puesto 1 ha revisado el coche 3 en un tiempo de 70
El puesto 1 ha revisado el coche 4 en un tiempo de 37
El puesto 1 ha revisado el coche 5 en un tiempo de 47
El puesto 1 ha revisado el coche 6 en un tiempo de 51
El puesto 1 ha revisado el coche 7 en un tiempo de 43
El puesto 1 ha revisado el coche 8 en un tiempo de 16
El puesto 1 ha revisado el coche 9 en un tiempo de 19
El puesto 1 ha revisado el coche 10 en un tiempo de 15
El puesto 1 ha revisado el coche 11 en un tiempo de 27
El puesto 1 ha revisado el coche 4 en un tiempo de 37
El puesto 1 ha revisado el coche 5 en un tiempo de 47
El puesto 1 ha revisado el coche 6 en un tiempo de 51
El puesto 1 ha revisado el coche 7 en un tiempo de 43
El puesto 1 ha revisado el coche 8 en un tiempo de 16
El puesto 1 ha revisado el coche 9 en un tiempo de 19
El puesto 1 ha revisado el coche 10 en un tiempo de 15
El puesto 1 ha revisado el coche 11 en un tiempo de 27
El puesto 1 ha revisado el coche 12 en un tiempo de 31
El puesto 1 ha revisado el coche 13 en un tiempo de 22
El puesto 1 ha revisado el coche 14 en un tiempo de 57
El puesto 1 ha revisado el coche 5 en un tiempo de 47
El puesto 1 ha revisado el coche 6 en un tiempo de 51
El puesto 1 ha revisado el coche 7 en un tiempo de 43
El puesto 1 ha revisado el coche 8 en un tiempo de 16
El puesto 1 ha revisado el coche 9 en un tiempo de 19
El puesto 1 ha revisado el coche 10 en un tiempo de 15
El puesto 1 ha revisado el coche 11 en un tiempo de 27
El puesto 1 ha revisado el coche 12 en un tiempo de 31
El puesto 1 ha revisado el coche 13 en un tiempo de 22
El puesto 1 ha revisado el coche 14 en un tiempo de 57
El puesto 1 ha revisado el coche 6 en un tiempo de 51
El puesto 1 ha revisado el coche 7 en un tiempo de 43
El puesto 1 ha revisado el coche 8 en un tiempo de 16
El puesto 1 ha revisado el coche 9 en un tiempo de 19
El puesto 1 ha revisado el coche 10 en un tiempo de 15
El puesto 1 ha revisado el coche 11 en un tiempo de 27
El puesto 1 ha revisado el coche 12 en un tiempo de 31
El puesto 1 ha revisado el coche 13 en un tiempo de 22
El puesto 1 ha revisado el coche 14 en un tiempo de 57
El puesto 1 ha revisado el coche 7 en un tiempo de 43
El puesto 1 ha revisado el coche 8 en un tiempo de 16
El puesto 1 ha revisado el coche 9 en un tiempo de 19
El puesto 1 ha revisado el coche 10 en un tiempo de 15
El puesto 1 ha revisado el coche 11 en un tiempo de 27
El puesto 1 ha revisado el coche 12 en un tiempo de 31
El puesto 1 ha revisado el coche 13 en un tiempo de 22
El puesto 1 ha revisado el coche 14 en un tiempo de 57
El puesto 1 ha revisado el coche 8 en un tiempo de 16
El puesto 1 ha revisado el coche 9 en un tiempo de 19
```

```
El puesto 1 ha revisado el coche 33 en un tiempo de 75
El puesto 1 ha revisado el coche 34 en un tiempo de 75
El puesto 1 ha revisado el coche 35 en un tiempo de 91
El puesto 1 ha revisado el coche 36 en un tiempo de 61
El puesto 1 ha revisado el coche 33 en un tiempo de 75
El puesto 1 ha revisado el coche 34 en un tiempo de 75
El puesto 1 ha revisado el coche 35 en un tiempo de 91
El puesto 1 ha revisado el coche 36 en un tiempo de 61
El puesto 1 ha revisado el coche 37 en un tiempo de 33
El puesto 1 ha revisado el coche 35 en un tiempo de 91
El puesto 1 ha revisado el coche 36 en un tiempo de 61
El puesto 1 ha revisado el coche 37 en un tiempo de 33
El puesto 1 ha revisado el coche 36 en un tiempo de 61
El puesto 1 ha revisado el coche 37 en un tiempo de 33
El puesto 1 ha revisado el coche 37 en un tiempo de 33
El puesto 1 ha revisado el coche 38 en un tiempo de 33
El puesto 1 ha revisado el coche 38 en un tiempo de 33
El puesto 1 ha revisado el coche 39 en un tiempo de 58
El puesto 1 ha revisado el coche 40 en un tiempo de 74
El puesto 1 ha revisado el coche 41 en un tiempo de 29
El puesto 1 ha revisado el coche 42 en un tiempo de 24
El puesto 1 ha revisado el coche 43 en un tiempo de 71
El puesto 1 ha revisado el coche 44 en un tiempo de 52
El puesto 1 ha revisado el coche 42 en un tiempo de 24
El puesto 1 ha revisado el coche 43 en un tiempo de 71
El puesto 1 ha revisado el coche 44 en un tiempo de 52
El puesto 1 ha revisado el coche 43 en un tiempo de 71
El puesto 1 ha revisado el coche 44 en un tiempo de 52
El puesto 1 ha revisado el coche 45 en un tiempo de 18
El puesto 1 ha revisado el coche 44 en un tiempo de 52
El puesto 1 ha revisado el coche 45 en un tiempo de 18
El puesto 1 ha revisado el coche 45 en un tiempo de 18
El puesto 1 ha revisado el coche 46 en un tiempo de 24
El puesto 1 ha revisado el coche 48 en un tiempo de 21
El puesto 1 ha revisado el coche 49 en un tiempo de 24
Fin del puesto 1, que termina con un tiempo parcial de 2260
Se cierra la ITV con un tiempo acumulado de 2260
```

Explicación del código:

Después de examinar el código a profundidad y correrlo en múltiples ocasiones me paso que en la primera corrida solo laburo el puesto 1, en esta primera ejecución no hubo concurrencia, si no un proceso completamente lineal en el que todos los autos esperaban a que el único agente de la ITV se desocupara para ser atendidos, esto me extraño bastante porque durante todo el video se habla en múltiples ocasiones sobre la concurrencia, al tener dos puestos de ITV que examinan los vehículos. Tuve que volver a ejecutar el código y en esta ocasión si se ejecutó un código con componentes de concurrencia.

Evidencia de la segunda ejecución en la terminal de Visual Studio:



Segunda explicación del código:

Después de correrlo por segunda vez, ahora si se logra apreciar el factor de la concurrencia, en esta ocasión ambos puestos trabajaron en conjunto para realizar sus labores de verificación de los automóviles, uno de los puntos a destacar sobre la diferencia entre ambas ejecuciones, es el tiempo, en la primera ejecución se observa un tiempo acumulado de 2260 min, mientras que en la segunda ejecución se observa un tiempo bastante menor, de 876 y 886 min para cada puesto, dando un tiempo acumulado menor, debido a que eran menos autos por examinar, pero un tiempo real mucho menor al de la ejecución con un solo puesto, esto resalta enormemente el valor que tiene la concurrencia en nuestros programas, pudiendo ayudar a literalmente reducir a la mitad los tiempos de espera y mejorando enormemente la eficiencia de nuestros códigos, aunque obviamente, siempre teniendo los cuidados adecuado

Parte II

9. Elabora un programa en Java basándote en el programa **Tuberia.java**, para modelar el **"Problema del barbero dormilón",** que se describe a continuación, utilizando HILOS, como en el programa de la tubería:
    - Identifica la región crítica, los hilos, la concurrencia, etc.
    - Considera los resultados de la solución implementada, si cumple con los criterios expresados y cómo se evitan fallos

Problema del Barbero dormilón

En una barbería trabaja un barbero que tiene un único sillón y varias sillas para esperar. Cuando no hay clientes, el barbero se sienta en una silla y se duerme. Cuando llega un nuevo cliente, éste, o bien despierta al barbero o — si el barbero está afeitando a otro cliente— se sienta en una silla (o se va si todas las sillas están ocupadas por clientes esperando). El problema consiste en realizar la actividad del barbero sin que ocurran condiciones de carrera.

10. Redacta tu informe de resultados **(Parte I y II)** considerando lo siguiente:
    - Código desarrollado en Lenguaje C
    - Imágenes con resultados de la ejecución
    - Explicación completa de la implementación haciendo énfasis en la parte de los procesos, su creación, gestión y concurrencia.

10. Redacta una **conclusión** sobre la comunicación entre procesos, utilizando semáforos, su utilidad y los problemas que crees que se puedan resolver utilizando esta técnica.

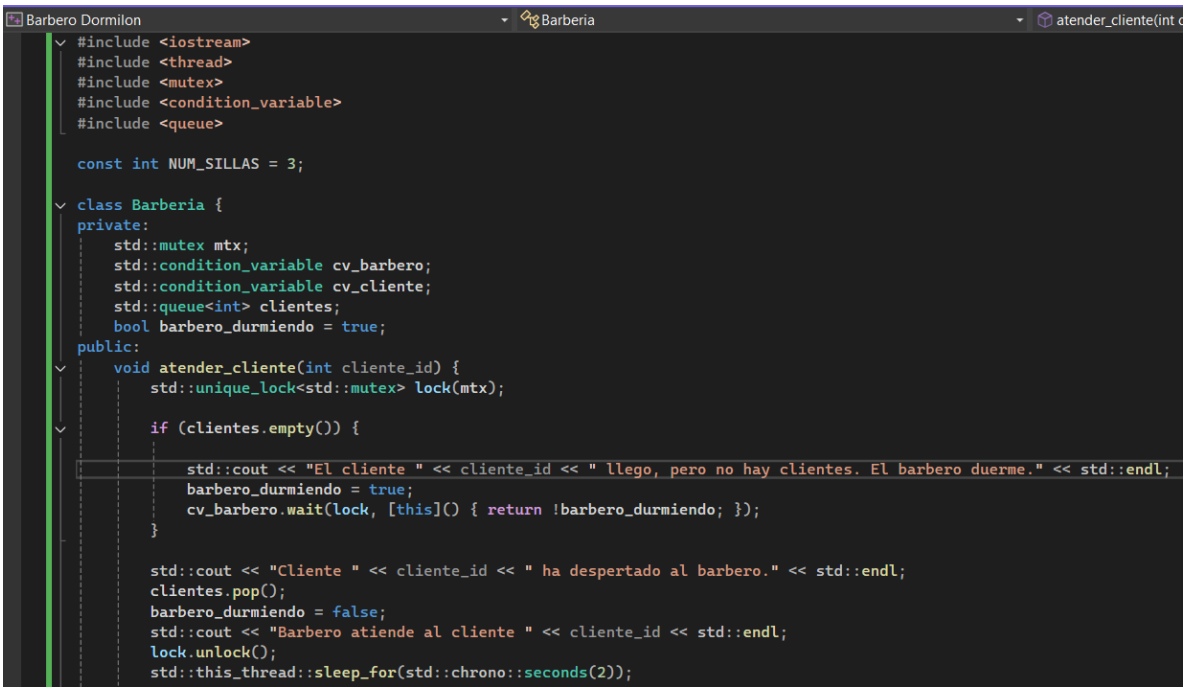
11. Agrega las fuentes consultadas (mínimo 2) referenciadas en estilo APA.

12. Al finalizar, vuelve a la plataforma y sigue los pasos que se indican para enviar tu actividad.

* * *

Parte 2

Código del problema del barbero dormilón C++



Barbero Dormilon — Barberia

```
        std::cout << "Cliente " << cliente_id << " ha terminado y se va." << std::endl;
        lock.lock();
        if (clientes.empty()) {
            barbero_durmiendo = true;
            std::cout << "El barbero se queda dormido." << std::endl;
        }
        cv_cliente.notify_one();
    }
    void cliente_llega(int cliente_id) {
        std::unique_lock<std::mutex> lock(mtx);

        if (clientes.size() < NUM_SILLAS) {
            clientes.push(cliente_id);
            std::cout << "Cliente " << cliente_id << " se sienta en la sala de espera." << std::endl;

            if (barbero_durmiendo) {
                barbero_durmiendo = false;
                cv_barbero.notify_one();
            }
        }
        else {
            std::cout << "Cliente " << cliente_id << " llega, pero no hay espacio. Se va." << std::endl;
        }
    }

    void ejecutar_barbero() {
        while (true) {
            std::unique_lock<std::mutex> lock(mtx);
            if (clientes.empty()) {
                barbero_durmiendo = true;
                std::cout << "El barbero duerme esperando clientes." << std::endl;
                cv_barbero.wait(lock, [this]() { return !barbero_durmiendo; });
```

Barbero Dormilon — Barberia

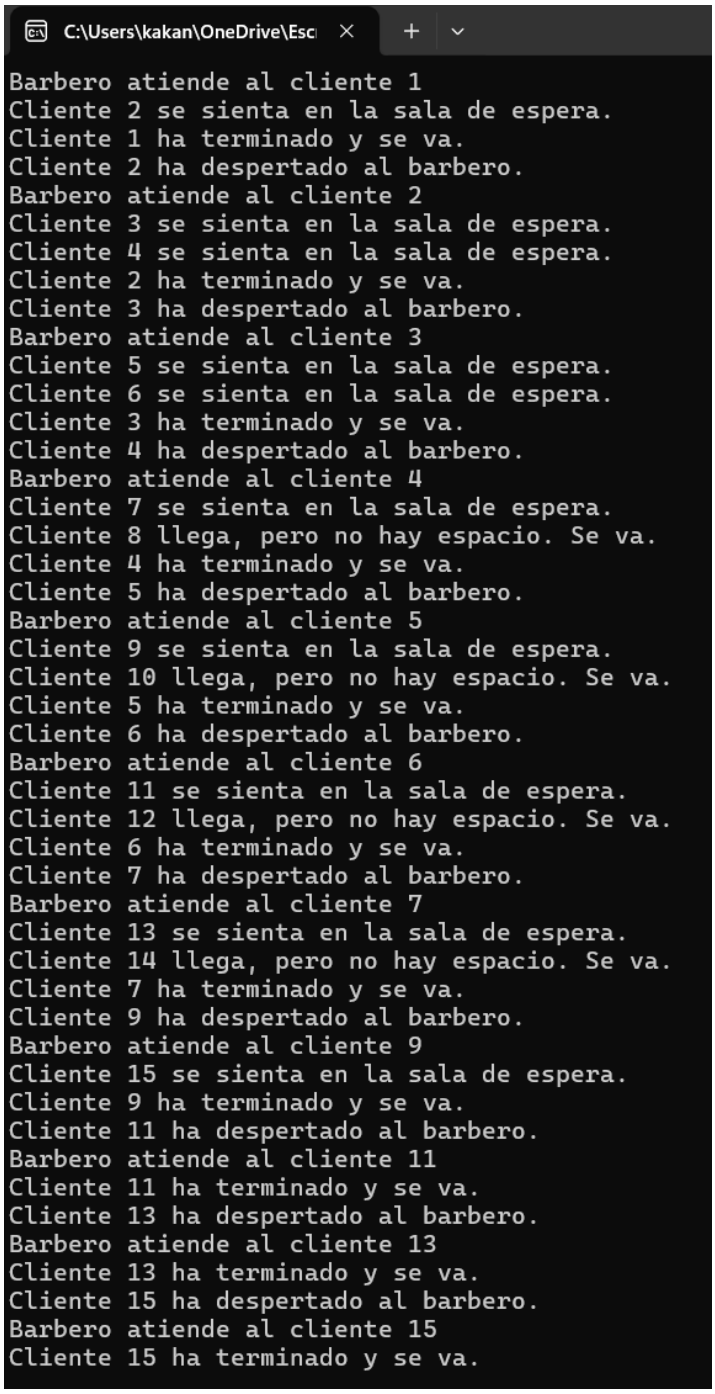
```
            if (!clientes.empty()) {
                int cliente_id = clientes.front();
                lock.unlock();
                atender_cliente(cliente_id);
            }
        }
    }
};

int main() {
    Barberia barberia;

    std::thread barbero_thread(&Barberia::ejecutar_barbero, &barberia);
    std::vector<std::thread> clientes_threads;
    for (int i = 1; i <= 15; ++i) {
        clientes_threads.push_back(std::thread([&barberia, i]() { barberia.cliente_llega(i); }));
        std::this_thread::sleep_for(std::chrono::seconds(1));
    }
    for (auto& t : clientes_threads) {
        t.join();
    }

    barbero_thread.join();

    return 0;
}
```



Región critica

La región critica en este programa es el acceso modificado de la cola de clientes esto es porque tanto el barbero interactúa como los clientes interactúan con la misma estructura, para evitar las condiciones de carrera se utiliza un (mutex)esto asegura que un hilo pueda acceder a la cola de clientes a la vez evitando problemas de concurrencia.

Además, esto sería un comportamiento equivalente a un semáforo: (mutex)

- atender_cliente: accede a la cola cliente y cambia el estado barbero_durmiendo esta región está protegida para que no tos los hilos entren a competir con std:unique_lock<std:mutex> lock(mtx);
- cliente _llega aquí se verifica y modifica la cola cliente y cambia el estado del barbero si lo despierta
- ejecutar_barbero () aquí se accede y evaluar si la cola de clientes está vacía y espera con (cv_barbero.wait)

Exclusión mutua

Se usa el (mutex) llamado (mtx) para asegurar que el acceso a recursos compartidos como la cola de clientes y la del barbero dormilón sea exclusiva.

Hilos y concurrencia

El programa maneja concurrencia mediante el uso de hilos (threads) estos realizan las siguientes funciones.

- Se crean 15 hilos para clientes que llegan 1 a 1 cada segundo con std::this_thread::sleep_for(std::chrono::seconds(1)
- Hilo del barbero ejecuta un bucle infinito en el que espera clientes cuando hay clientes en la sala de espera los atiende uno por uno.
- Hilo de clientes estos son creados como hilos que simulan la llegada de estos a la barbería, cada cliente intenta sentarse en la sala de espera si hay espacio disponible si no hay espacio se va.

La concurrencia se maneja a través de un (mutex) y (condiciones de variables)para sincronizar los hilos.

Uso de (std::mutex) y (std::condition_variable)

- std::mutex este se utiliza para proteger el acceso a variables compartidas como la cola de clientes asegurando que solo un hilo pueda modificar la cola de clientes a la vez.
- **std::condition_variable cv_barbero** esta variable de condición se usa para esperar hasta que haya un cliente disponible para atender. Si no hay clientes el barbero se queda dormido hasta que sea despertado por un cliente.

- **std::condition_variable cv_cliente** se utiliza para notificar al barbero cuando a terminado de atender a un cliente y para coordinar las interacciones entre los hilos del cliente y el barbero

Recursos compartidos

Estos se comparten entre hilos y pueden se modificados por varios hilos al mismo tiempo y se protegen con el mutex.

- La cola del cliente `std::queue<int> clientes`
- `barbero_durmiendo`

Resultados de la solución

- Clientes que esperan estos llegan de uno en uno y si hay espacio en la sala de espera se sientan si no hay espacio se van
- Despertar del barbero: cuando un cliente llega y el barbero esta dormido el cliente lo despierta y el barbero comienza a atender al cliente
- Termino de un cliente después de que el barbero atiende a un cliente verifica que la barbería este vacía, si esta el barbero se duerme hasta que llega otro cliente.

Conclusiones

Romel Arenas Jiménez:

En conclusión, el ejercicio 3 ha sido una experiencia enriquecedora que nos ha permitido aplicar y reforzar nuestros conocimientos sobre la gestión de procesos, concurrencia, creación y control de hilos, y sincronización en sistemas operativos. A través del trabajo en equipo y la colaboración, hemos podido analizar y comprender de manera práctica los conceptos clave de la concurrencia y la sincronización en diferentes entornos de programación, como Java y C. La implementación del problema del barbero dormilón nos ha permitido comparar enfoques y reforzar nuestra comprensión práctica del manejo de procesos concurrentes. La experiencia de trabajar en conjunto con mis compañeros de equipo ha sido fundamental para superar los desafíos y alcanzar nuestros objetivos. A lo largo de este proceso, hemos desarrollado habilidades prácticas y teóricas en la programación y el desarrollo de software concurrente.

La utilización de semáforos como mecanismo de sincronización ha sido un aspecto clave en la implementación de programas concurrentes. En general, esta experiencia ha sido un paso importante en nuestro crecimiento profesional y estamos seguros de que nos beneficiará en el futuro. La comprensión de los conceptos de concurrencia y sincronización nos permitirá desarrollar software más eficiente y escalable.

Adair Espinosa Estrada

Puedo concluir que la comunicación entre procesos mediante el uso de semáforos es una técnica fundamental para lograr la sincronización y el control adecuado del acceso a recursos compartidos en sistemas concurrentes, los semáforos permiten evitar condiciones de carrera, garantizar la exclusión mutua y coordinar la ejecución ordenada de múltiples procesos o hilos, gracias a esta técnica, es posible resolver problemas como la pérdida de datos, bloqueos indeseados o la ejecución desordenada de procesos en tareas críticas como la gestión de memoria, el acceso a archivos o la ejecución paralela de programas. En definitiva, aprender sobre semáforos me ha ayudado a entender mejor cómo funciona la programación concurrente y por qué es esencial aplicar mecanismos de sincronización para asegurar la estabilidad y eficiencia de los sistemas.

Jose Emiliano Jauregui Guzman

Al concluir este ejercicio, puedo afirmar que he adquirido una comprensión más profunda sobre los conceptos fundamentales de los sistemas operativos, especialmente en lo que respecta a la gestión de procesos y la concurrencia. La implementación y ejecución de los programas en Java y C me permitió observar de manera práctica cómo se manejan los hilos y cómo se controla el acceso a los recursos compartidos mediante técnicas de sincronización, como el uso de semáforos.

El análisis del programa Tuberia.java fue útil para entender cómo los hilos pueden trabajar de manera concurrente y la importancia de gestionar correctamente el acceso a los recursos para evitar conflictos. Además, la implementación del problema del barbero dormilón en C me permitió comparar cómo se resuelven problemas similares en diferentes lenguajes de programación y cómo cada uno maneja la sincronización y los procesos concurrentes de manera eficiente.

Angel Alejandro Alcala Aguilar

Para finalizar esta actividad, quiero mencionar que, en lo personal, aunque me pareció una actividad muy divertida de realizar, con un ejemplo practico muy interesante, sobre todo porque me encantan los autos y ver un poco sobre como funciona la ITV me llamo mucho la atención, aunque la ITV sea el proceso español para la revisión de los vehículos.

Ahora, el pero que le pongo a la actividad es a la ausencia de explicación sobre donde encontrar tanto los códigos de Tuberia.java como de programa.c, estos códigos no venían en la plataforma, o al menos ni yo ni los compañeros a los que pregunte pudieron localizarlos, gracias a la videollamada se logró identificar que Tuberia.java era el código que se menciona en el video recurso de plataforma, contenido de Camacho, P. (Productor). (20 de noviembre de 2016), más, sin embargo, programa.c permaneció como una incógnita completa, sin aparecer por ningún lado, revise el foro de dudas y otro compañero tenia la misma duda que yo, aunque no recibió respuesta.

Lamentablemente solo pude revisar los contenidos de plataforma hasta este fin de semana, lo que hizo que me percatara muy tarde sobre el inconveniente, ya no dejando tiempo para hacer preguntas, espero que la próxima semana no me sea tan atareada y pueda realizarla con mas tiempo para poder resolver mis dudas con usted.

Para finalizar, nuevamente me pareció una actividad muy interesante, me gusta mucho programar, no había usado mucho java pero cada vez me familiarizo más, incluí más información de la solicitada sobre el programa de tubería.java para compensar lo demás, espero que no nos perjudique en nuestra calificación, pues de verdad nos esforzamos mucho en la actividad pese a la limitante del tiempo que tenemos como equipo, pues todos trabajamos.

Referencias

Gelpi Fleta, D. & Sierra González, J. M. (2013). Sistemas operativos monopuestos Haga clic para ver más opciones [Archivo PDF]. Recuperado de https://www.macmillaneducation.es/formacion-profesional/grado-medio/sistemas-microinformaticos-y-redes/sistemas-operativos-monopuesto/

Wolf, G., Ruiz, E., Bergero, F. y Meza, E. (2015). Fundamentos de Sistemas Operativos Haga clic para ver más opciones [Versión electrónica]. Recuperado de http://ru.iiec.unam.mx/2718/1/sistemas_operativos.pdf

Henrik, H. (2016). Synchronization Mechanisms Haga clic para ver más opciones [Versión electrónica]. Recuperado de http://www2.compute.dtu.dk/courses/02158/sync.pdf

Camacho, P. (Productor). (20 de Noviembre de 2016). Hilos en Java con semáforos [versión electrónica]. Recuperado de https://www.youtube.com/watch?v=o0Lzvm1OFiA

Miadelets, O. (2024, 7 febrero). Diferencia entre un mutex, un monitor y un semáforo. CodeGym. https://codegym.cc/es/groups/posts/es.220.diferencia-entre-un-mutex-un-monitor-y-un-semaforo