INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

UNIDAD DE APRENDIZAJE

SISTEMAS DISTRIBUIDOS

PRACTICA No. 1

PROCESOS E HILOS

ALUMNO

GÓMEZ GALVAN DIEGO YAEL

GRUPO

7CM1

PROFESORA

CARRETO ARELLANO CHADWICK

FECHA DE ENTREGA

24 DE FEBRERO DE 2025

INDICE

ANTESCEDENTES

En la computación moderna, la ejecución eficiente de programas requiere el uso de técnicas que permitan realizar múltiples tareas de manera simultánea. Para lograr esto, los sistemas operativos emplean dos conceptos fundamentales que abordaremos a continuación:

PROCESOS

Un proceso es una instancia en ejecución de un programa. Cada proceso tiene su propio espacio de memoria, recursos asignados y una o más unidades de ejecución llamadas hilos. Los procesos se gestionan de manera independiente en el sistema operativo, lo que significa que no comparten memoria ni recursos directamente con otros procesos. Debido a esta independencia, la comunicación entre procesos suele requerir mecanismos especiales como tuberías, memoria compartida o sockets.

Las principales características de los procesos son las siguientes:

- ✓ Poseen su propio espacio de memoria y recursos
- ✓ Se ejecutan de forma independiente, lo que los hace mas seguros pero menos eficientes en términos de comunicación con otros procesos.
- ✓ Son gestionados por el sistema operativo, lo que implica que cambiar entre procesos es una tarea costosa en términos de rendimiento.

Un ejemplo común de procesos es la ejecución simultanea de diferentes aplicaciones en un sistema operativo, como un navegador web y un editor de texto.

HILOS

Un hilo es la unidad mas pequeña de ejecución dentro de un proceso. Todos los hilos de un proceso comparten el mismo espacio de memoria y pueden acceder a los mismos recursos. Esto permite una ejecución más rápida y eficiente en comparación con los procesos, pero también introduce problemas de concurrencia, como condiciones de carrera y accesos no sincronizados a recursos compartidos.

Las principales características de los hilos son las siguientes:

- ✓ Comparten el mismo espacio de memoria dentro de un proceso
- ✓ Permite ejecutar múltiples tareas de manera paralela de una misma aplicación
- ✓ Son mas ligeros y eficientes en comparación con los procesos
- ✓ Su administración es menos costosa en términos de recursos del sistema

Un ejemplo de hilos es la ejecución de múltiples tareas dentro de una misma aplicación, como un reproductor de música que reproduce una canción mientras carga la siguiente en segundo plano.

DIFERENCIAS ENTRE PROCESOS E HILOS

Los procesos y los hilos presentan diferencias significativas en la forma en que se ejecutan, administran recursos y comunican entre sí:

- ✓ Espacio de memoria: Los procesos poseen su propio espacio de memoria, mientras que los hilos dentro de un mismo proceso comparten la misma memoria. Esto hace que los procesos sean mas seguros, pero menos eficientes en términos de comunicación, ya que requieren mecanismos adicionales para intercambiar datos.
- ✓ Comunicación y sincronización: Debido a su independencia, los procesos deben utilizar técnicas como tuberías, memoria compartida o sockets para comunicarse entre sí, lo que implica mayor complejidad. En cambio, los hilos pueden compartir datos de manera directa, aunque esto conlleva riesgos de concurrencia si no se gestionan correctamente.
- ✓ Independencia: Un proceso es independiente de otros procesos, de modo que si uno falla, no afecta a los demás. Sin embargo, los hilos de un mismo proceso están interconectados, por lo que un fallo en un hilo puede afectar a todo el sistema.
- ✓ Consumo de recursos: Los procesos requieren más recursos del sistema, ya que cada uno necesita su propia asignación de memoria y estructuras de control. Con los hilos, son más ligeros y eficientes, ya que comparten recursos y su administración es menos costosa.

PLANTEAMIENTO DEL PROBLEMA

En la actualidad para el desarrollo de software es necesaria la ejecución simultanea de múltiples tareas con el fin de mejorar el rendimiento y la eficiencia de los sistemas. Uno de los métodos mas utilizados para poder realizar esto, es la programación concurrente, donde se emplean hilos para ejecutar diversas operaciones al mismo tiempo dentro de un mismo programa. No obstante, comprender el uso de hilos y diferenciarlos de los procesos puede resultar complejo sin una implementación practica que permita visualizar su funcionamiento.

Bajo este contexto, surgió la necesidad de desarrollar un programa que permita ilustrar como los hilos pueden ser utilizados para gestionar múltiples interacciones en un sistema. Un ejemplo practico es un cajero automático (ATM), en donde los usuarios pueden realizar operaciones bancarias sin necesidad de esperar a que otros procesos independientes finalicen.

Sin embargo, el uso de hilos también presenta desafíos importantes, ya que la ejecución concurrente de tareas puede generar problemas como accesos inconsistentes a los datos, condiciones de carrera y falta de sincronización en el manejo de recursos compartidos. Estos problemas pueden afectar la seguridad y confiabilidad de un sistema, lo que resalta la importancia de estudiar cómo los hilos impactan en la ejecución de aplicaciones críticas.

Además, en sistemas donde múltiples usuarios requieren acceso simultáneo, es importante entender cuándo es más conveniente usar hilos en lugar de procesos, y qué técnicas pueden aplicarse para evitar errores en entornos concurrentes. Por ello, es fundamental explorar las diferencias entre ambos modelos de ejecución y analizar los problemas que pueden surgir en el manejo de múltiples tareas dentro de una misma aplicación.

PROPUESTA DE SOLUCIÓN

Para abordar la problemática de la ejecución concurrente y la diferencia entre procesos e hilos, se propone el desarrollo de un simulador de cajero automático (ATM) que permita ejemplificar el uso de hilos en la ejecución de múltiples interacciones dentro de un sistema.

El simulador deberá permitir que un usuario interactúa con su cuenta bancaria a través de un menú donde podrá realizar operaciones como consultar saldo, depositar y retirar dinero. La aplicación se diseñará para ejecutar cada sesión de usuario en un hilo independiente, permitiendo así observar el comportamiento de la ejecución concurrente.

Esta solución permitirá visualizar como los hilos pueden utilizarse para manejar tareas simultaneas dentro de un mismo proceso, demostrando tanto sus ventajas como los posibles desafíos que pueden surgir en su implementación.

MATERIALES Y METODOS

Para la implementación del simulador, se utilizaron los siguientes recursos:

- ✓ Lenguaje de programación: Java
- ✓ Entorno de desarrollo: NetBeans IDE 21
- ✓ Compilador: Java Compiler (javac) incluido en el JDK
- ✓ Sistema operativo: Windows 11
- ✓ Librerías utilizadas:
  - Java.lang.Thread: Librería implementada para manejar la ejecución de hilos
  - Java.lang.Runnable: Librería implementada para definir la funcionalidad de los hilos
  - Java.util.Scanner: Librería utilizada para capturar la entrada del usuario desde la consola

DESARROLLO DE SOLUCIÓN

La solución se estructuro en tres clases principales:

Clase CuentaBancaria

Esta clase representa la cuenta del usuario y almacena información relevante como el número de cuenta, el titular, el saldo y el NIP de acceso. También proporciona métodos para la manipulación de los fondos de la cuenta. Los métodos implementados en dicha clase son los siguientes:

```
public CuentaBancaria(String numeroCuenta, String titular, double saldo, int nip){
    this.numeroCuenta = numeroCuenta;
    this.titular = titular;
    this.saldo = saldo;
    this.nip = nip;
}
```

Este método es el constructor que se encarga de inicializar los datos de la cuenta

```
public String getTitular(){
    return titular;
}
```

Este método se encarga de devolver el nombre del titular de la cuenta

```
public double consultarSaldo(){
    return saldo;
}
```

Este método retorna el saldo actual de la cuenta

```
public void depositar(double monto){
    if(monto > 0){
        saldo += monto;
        System.out.println("Deposito exitoso. Nuevo saldo $"+saldo);
    }
}
```

Este método aumenta el saldo de la cuenta si el monto ingresado es valido

```java
public void retirar(double monto){
    if(monto < saldo){
        saldo -= monto;
        System.out.println("Retiro exitoso. Nuevo saldo $"+saldo);
    }else{
        System.out.println("La cantidad debe ser menor al saldo total de la cuenta.");
    }
}
```

Este método disminuye el saldo de la cuenta si hay suficientes fondos disponibles

```java
public boolean autenticar(int claveIngresada) {
    return nip==claveIngresada;
}
```

Este método valida que el NIP ingresado sea el correcto para la cuenta

Clase ATM

Esta clase maneja la interacción del usuario con el cajero automático y permite realizar operaciones bancarias. Se implementa la interfaz Runnable para que cada instancia del cajero pueda ejecutarse en un hilo independiente. Los métodos implementados son los siguientes:

```java
public ATM(CuentaBancaria cuenta, int nip){
    this.cuenta = cuenta;
    this.nip = nip;
}
```

Este método es el constructor que recibe la cuenta y el NIP del usuario

```java
public void run(){
    if(!cuenta.autenticar(nip)){
        System.out.println("Nip incorrecto. Acceso denegado.");
        return;
    }
    Scanner scanner = new Scanner(System.in);
    int opc;
    do{
        System.out.println("\n Bienvenido a su cajero automatico, " + cuenta.getTitular());
        System.out.println("1. Consultar Saldo\n2. Depositar Dinero\n3. Retirar Dinero\n4. Salir");
        System.out.print("Seleccione una opcion: ");
        opc = scanner.nextInt();
        switch (opc) {
            case 1:
                System.out.println("Saldo actual: " + cuenta.consultarSaldo());
                break;
            case 2:
                System.out.print("Ingrese monto a depositar: ");
                double montoDeposito = scanner.nextDouble();
                cuenta.depositar(montoDeposito);
                break;
            case 3:
                System.out.print("Ingrese monto a retirar: ");
                double montoRetiro = scanner.nextDouble();
                cuenta.retirar(montoRetiro);
                break;
            case 4:
                System.out.println("Sesión finalizada.");
                break;
            default:
                System.out.println("Opción no válida.");
        }
    } while (opc != 4);
}
```

Este es el método principal de la clase que se ejecuta cuando el hilo inicia. Este realiza las funciones de verificación del NIP, presenta el menú con opciones para consultar saldo, depositar y retirar dinero, captura la entrada del usuario y ejecuta la acción correspondiente y permite salir del sistema y finalizar la sesión del usuario.

Clase SimuladorCajero

Esta clase principal inicia la simulación creando una cuenta de prueba y gestionando la interacción del usuario. Posee el siguiente único método main:

```
public static void main(String[] args) {
    // Crear una cuenta bancaria de ejemplo
    CuentaBancaria cuenta = new CuentaBancaria("123456", "Juan Perez", 5000, 1234);

    Scanner scanner = new Scanner(System.in);

    System.out.print("Ingrese su clave (4 digitos): ");
    int claveIngresada = scanner.nextInt();

    // Crear e iniciar un único hilo para manejar la sesión del cliente
    Thread cliente = new Thread(new ATM(cuenta, claveIngresada), "UsuarioATM");
    cliente.start();
}
```

Este método es el punto de entrada del programa que crea una cuenta bancaria con un saldo inicial, solicita al usuario el NIP y crea un hilo que ejecuta una instancia de ATM, iniciando la interacción con el usuario

RESULTADOS

```
Ingrese su clave (4 digitos): 1234
```

Al ejecutar el programa, este solicita la clave de 4 dígitos (NIP) para autenticar al usuario, Una vez que este se ingresa, se valida que sea correcta y permite el ingreso al sistema

```
 Bienvenido a su cajero automatico, Juan Perez
1. Consultar Saldo
2. Depositar Dinero
3. Retirar Dinero
4. Salir
Seleccione una opcion: 1
Saldo actual: 5000.0
```

Una vez dentro del sistema, nos despliegue el menú de los posibles movimientos y solicitudes que puedes realizar. En este caso se ejecuto inicialmente la consulta de saldo con la opción 1, donde este nos proporciono el saldo que se encontraba en la cuenta

```
 Bienvenido a su cajero automatico, Juan Perez
1. Consultar Saldo
2. Depositar Dinero
3. Retirar Dinero
4. Salir
Seleccione una opcion: 2
Ingrese monto a depositar: 100
Deposito exitoso. Nuevo saldo $5100.0
```

Aquí se realizó la operación de depositar dinero a través de la opción 2, donde se solicito el monto a depositar y se actualizo el saldo de la cuenta

```
 Bienvenido a su cajero automatico, Juan Perez
1. Consultar Saldo
2. Depositar Dinero
3. Retirar Dinero
4. Salir
Seleccione una opcion: 3
Ingrese monto a retirar: 200
Retiro exitoso. Nuevo saldo $4900.0
```

Aquí se seleccionó la opción de retirar dinero con el numero 3, por ello el sistema solicito el monto a retirar y posteriormente desplego el nuevo balance en la cuenta

```
 Bienvenido a su cajero automatico, Juan Perez
1. Consultar Saldo
2. Depositar Dinero
3. Retirar Dinero
4. Salir
Seleccione una opcion: 4
Sesi□n finalizada.
```

Por último, para cerrar la sesión generada, se ingresó la opción 4, donde se cerró el sistema.

CONCLUSIONES

A lo largo de esta práctica, pude entender la importancia de la programación concurrente y las diferencias clave entre los procesos e hilos. Se vio que los procesos son unidades de ejecución independientes que tienen su propio espacio de memoria, lo que los hace mas seguros pero menos eficientes en términos de comunicación y rendimiento. En cambio, los hilos compartes memoria dentro de un mismo proceso, lo que los hace más rápido y ligeros, pero también pueden causar problemas de concurrencia si no se gestionan correctamente.

El desarrollo del simulador de cajero automático, considero que fue una forma practica de aplicar estos conceptos. Al usar hilos, se pudo ver como un sistema puede generar múltiples sesiones sin necesidad de crear procesos separados. Esto permitió observar que cada usuario tenia su propio hilo de ejecución, interactuando con la cuenta memoria sin bloquear a otros posibles usuarios. Aunque en esta implementación solo se manejo un usuario a la vez, quedo claro que el uso de hilos facilita la ejecución simultanea de múltiples tareas dentro de una misma aplicación.

Además, se identificaron algunos desafíos del uso de hilos, como la necesidad de sincronización cuando hay acceso concurrente a recursos compartidos. En este caso, no hubo conflictos porque cada usuario trabajaba con su propia instancia del ATM, pero en un sistema real con múltiples clientes accediendo a una misma cuenta, sería necesario implementar mecanismos de control como bloqueos o semáforos.

En conclusión, esta práctica me ayudo a reforzar la diferencia entre procesos e hilos, mostrando que los hilos son una herramienta útil para la ejecución concurrente dentro de un mismo programa. También permitió ver que, aunque son más eficientes en ciertos casos, requieren un manejo cuidadoso para evitar problemas de acceso a la memoria compartida.