	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor: *Edgar Tista García*

Asignatura: *Sistemas Operativos*

Grupo: *2*

No de Práctica(s): *3*

Integrante(s): *Flores Chávez Marcos Gabriel*
Salgado Miranda Jorge

No. de Lista o Brigada:

Semestre: *2023-2*

Fecha de entrega: *24 de septiembre 2024*

Observaciones:

CALIFICACIÓN: _____

Practica 3: Procesos Parte 1

Objetivo.

- Revisar el problema del productor-consumidor como un ejemplo clásico de comunicación entre procesos
- Poner en práctica los conceptos básicos de programación de Hilos y ver las diferencias entre hilos en POSIX e hilos en java Introducción Un proceso se puede definir de diferentes formas ya que es una de las unidades fundamentalesde trabajo de un sistema operativo.

Un proceso se puede definir de diferentes formas ya que es una de las unidades fundamentales de trabajo de un sistema operativo.

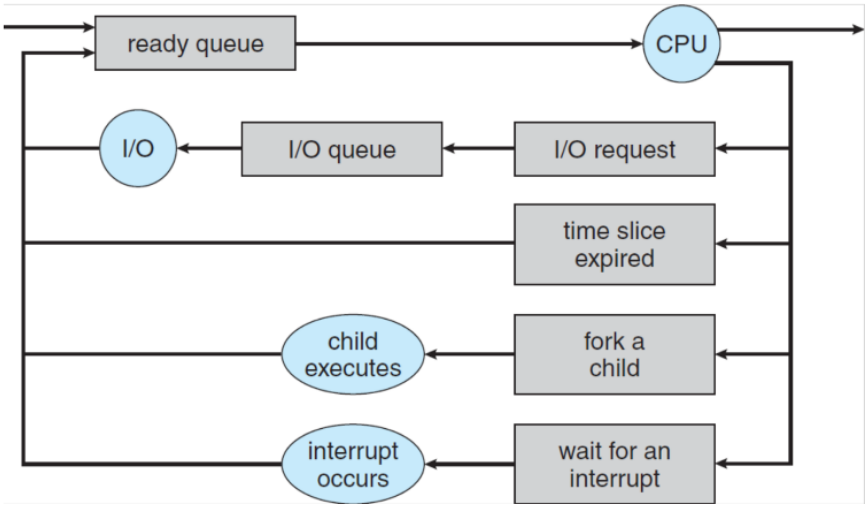
La estructura de un proceso cuenta con 4 elementos básicos:

- ❑ Sección de texto
- ❑ Sección de datos
- ❑ Pila
- ❑ Heap

El estado de un proceso depende de la actividad que esté realizando

- ❑ Nuevo: El proceso está siendo creado
- ❑ En ejecución: El cpu ejecuta las instrucciones
- ❑ Espera: Se encuentra en un estado que depende de un suceso externo
- ❑ Preparado: Listo para que se le asigne un procesador
- ❑ Terminado: Ha concluido la ejecución Los procesos activos se encuentran en la queue principal.

A medida que se crean, ingresan en esta estructura. (Su almacenamiento se implementa en forma de lista ligada Durante su ejecución los procesos se desplazan entre una queue y otra dependiendo de la operación que deban realizar El sistema operativo planifica y distribuye los procesos que se ejecutan



Desarrollo

1.- SIMULADOR DE PROCESOS

Se deberá realizar un programa que sea capaz de simular la ejecución de procesos en un sistema operativo.

El usuario final tomará el rol del sistema operativo y podrá interactuar con el simulador realizando una serie de actividades relacionadas con la creación y ejecución de procesos.

Se podrá seleccionar alguna de las opciones que proporciona el programa con diferentes acciones relacionadas con la creación y ejecución de procesos.

Al iniciar la ejecución del programa, habrá tabla global de "memoria disponible" (2048 localidades) Esta será la memoria disponible para crear "procesos".

En la ejecución, el usuario contará con un menú de opciones donde podrá realizar las siguientes acciones:

- Crear Proceso nuevo
    - El usuario podrá escribir el nombre del nuevo proceso.
    - El programa asignará un número identificador único para el proceso nuevo
    - El programa asignará un número aleatorio de instrucciones de ese proceso (10-30)
    - El programa asignará un espacio que ocupará el proceso (esta cantidad será de 64, 128, 256 o 512 localidades)
    - Después de crear el proceso ingresará a la cola general del sistema (cola de procesos preparados)

- Ver proceso activo. - EL proceso que se encuentra hasta el inicio de la cola será el proceso activo que simulará estar en ejecución, al seleccionar esta opción se mostrará:
    - o Nombre del proceso
    - o Id único o Instrucciones totales o Instrucciones ejecutadas

- Ejecutar proceso actual o Implica realizar 5 instrucciones del proceso activo
    - La ejecución implica "restar" 5 instrucciones totales del proceso
    - Al ejecutarse se mostrará un mensaje indicando que el proceso x se ha ejecutado e indicar el número de instrucciones restantes

- Pasar al proceso siguiente
    - Al seleccionar esta opción, de manera aleatoria podrá ocurrir alguna de las siguientes acciones con el proceso activo
        - Colocarse al final de la cola de procesos
        - Ir a la cola de entrada y salida
        - Solicitar una interrupción

- Ejecutar entrada y salida
    - Esta opción sacará al primer proceso que se encuentre en la lista de E/S y lo moverá a la lista de preparados

- Ejecutar interrupción
    - Esta opción sacará al primer proceso que se encuentre en la lista de interrupciones y lo moverá a la lista de preparados

- Matar proceso actual
    - Simplemente deberá liberar la memoria que utilizaba ese proceso y guardarlo en el registro de procesos eliminados o Informar cuantas instrucciones pendientes tenía el proceso

- Imprimir lista de procesos preparados(detalle)
    - Esta opción mostrará la lista de los procesos preparados para su ejecución.
    - De cada proceso se debe mostrar su nombre, su id, y las instrucciones pendientes por ejecutar
    - El proceso que se encuentre al frente de la cola será "el proceso activo"
    - Direcciones de memoria asignadas

- Imprimir lista de e/s
    - Esta opción mostrará la lista de los procesos que requieren operación de E/S

- Imprimir procesos pendientes de E/S
    - Esta opción mostrará la lista de los procesos pendientes de ejecutar interrupción

- Ver estado actual del sistema
    - o Indicar número de procesos en cola de preparados
    - Indicar número de procesos en cola de e/s o Mostrar lista de procesos pendientes de ejecutar interrupción
    - o Lista con el nombre de los procesos finalizados exitosamente
    - o Lista de procesos finalizados antes de tiempo (eliminados)

- Salir del programa
    - En caso de haber procesos en la cola de preparados se deberá informar al usuario que, si decide salir, x número de procesos no se concluirán exitosamente y preguntar si desea continuar.

Análisis y desarrollo del código

Este código fue implementado en Linux con el lenguaje de programación Java ya que justamente vimos que para posteriores trabajos de procesos será más fácil de modificar, ya que a la hora de querer implementar posteriormente algoritmos de planificación de procesos con las colas de prioridad de Java se nos hará más fácil de llevar a cabo. Este código se complementa de tres clases las cuales estarán conectadas entre si para poder llevar la ejecución y creación de procesos.

Como primera clase tenemos una clase la cual es la Main, esta será la encargada de mostrar nuestro menú de otra de nuestras clases llamada simulador, en si lo que contiene únicamente es un bucle while el cual será true hasta que el usuario decida salir del programa de la clase de simulador.

```
public class Main {
    public static void main(String[] args) {
        Simulador simulador = new Simulador();

        while(true){
            simulador.mostrarMenu();
        }
    }
}
```

Posterior a ello nos pasaremos a la siguiente clase antes de pasar a la más importante, de la cual hablaremos es de la clase procesos, está será la encargada de contener todos nuestros getters necesarios para el manejo de nuestros datos de los procesos, tales como su nombre, id, instrucciones a realizar, instrucciones ejecutadas, memoria asignada, entre otros más. De iugal forma nos apoyamos de la librería random para poder escoger el numero de instrucciones aleatorias y también creamos un arreglo de las posibles memorias para usar en los procesos.

```
public class Proceso {
    private final String nombre;
    private final int id;
    private final int instruccionesTotales;
    private int instruccionesEjecutadas = 0;
    private final int memoriaAsignada;
```

```
public Proceso(String nombre) {
    this.nombre = nombre;
    this.id = new Random().nextInt(10000);  // Un ID único basado en un número aleatorio.
    this.instruccionesTotales = new Random().nextInt(21) + 10; // Un número aleatorio entre 10-30.
    int[] posiblesMemorias = {64, 128, 256, 512};
    this.memoriaAsignada = posiblesMemorias[new Random().nextInt(4)];
}
```

```
public String getNombre() {
    return nombre;
}


public int getId() {
    return id;
}


public int getInstruccionesTotales() {
    return instruccionesTotales;
}
```

Todo esto anterior será utilizado ahora si en nuestra clase mas importante la cual es la de Simualdor, en esta es donde llevaremos a cabo la ejecución de los procesos como si fuera nuestro sistema operativo. Para empezar lo que se realizó fue crear todas nuestras colas para cada uno de los puntos solicitados de nuestra presente práctica, como la cola de procesos, los de E/S, de Interrupciones, procesos eliminados y procesos finalizados. De igual forma nos apoyamos de una variable entera para poder adminstrar lo que es la memoria usada por el proceso.

```java
public class Simulador {
    private final LinkedList<Proceso> colaDeProcesos = new LinkedList<>();
    private final LinkedList<Proceso> colaDeES = new LinkedList<>();
    private final LinkedList<Proceso> colaDeInterrupciones = new LinkedList<>();
    private final LinkedList<Proceso> procesosEliminados = new LinkedList<>();
    private final LinkedList<Proceso> procesosFinalizados = new LinkedList<>();
    private int memoriaUsada = 0;
    private final Scanner scanner = new Scanner(System.in);
```

Posterior a ello lo que realizamos es crear nuestro menú para que el usuario pueda acceder a las distintas funcionalidades de la creación del proceso requeridas y mencionadas anteriormente, de igual forma esto se hace con ayuda de un try catch el cual se encargara de manejar la excepción en el caso de que el usuario meta una respuesta u opción no válida para el programa.

```java
try {
    int seleccion = scanner.nextInt();
    scanner.nextLine(); // consume newline
    switch (seleccion) {
        case 1 -> crearProceso();
        case 2 -> verProcesoActivo();
        case 3 -> ejecutarProcesoActual();
        case 4 -> pasarAlProcesoSiguiente();
        case 5 -> ejecutarEntradaYSalida();
        case 6 -> ejecutarInterrupcion();
        case 7 -> matarProcesoActual();
        case 8 -> imprimirListaProcesosPreparados();
        case 9 -> imprimirListaES();
        case 10 -> imprimirProcesosPendientesInterrupcion();
        case 11 -> verEstadoActualSistema();
        case 12 -> {
            salirDelPrograma();
            continuar = false;
```

Pasemos con nuestro primer case este lo que hicimos fue insanciar a nuestra clase de procesos y le pasamos como parámetros el nombre del proceso a escoger al usuario, posterior a ello evaluamos si la memoria usada mas la memoria del nuevo proceso es menor a la memoria total del sistema lo añadimos a nuestra cola de procesos y añadimos en forma de contador a nuestra variable de memoria usada para simular como es que los procesos consumen memoria. Ya posterior a ello de igual forma le mostramos al usuario el nombre de su proceso como su id y la memoria a ocupar.

```java
// Checar si hay memoria suficiente para el nuevo proceso.
int memoriaTotal = 2048;
if ((memoriaUsada + nuevoProceso.getMemoriaAsignada()) <= memoriaTotal) {
    colaDeProcesos.addLast(nuevoProceso);  // Agregar el proceso a la cola de procesos.
    memoriaUsada += nuevoProceso.getMemoriaAsignada();  // Incrementar la memoria usada.

    System.out.println("Proceso \"" + nombre + "\" creado con ID: " + nuevoProceso.getId());
    System.out.println("Instrucciones: " + nuevoProceso.getInstruccionesTotales());
    System.out.println("Memoria asignada: " + nuevoProceso.getMemoriaAsignada() + " localidades");
} else {
    System.out.println("No hay suficiente memoria para crear el proceso.");
```

```
Escribe el nombre del nuevo proceso: word
Proceso "word" creado con ID: 1600
Instrucciones: 26
Memoria asignada: 512 localidades
```

Ahora pasemos al siguiente case de ver proceso activo, en este lo que hicimos fue con ayuda de un condicional if evaluar si hay algún proceso en la cola de procesos, si no hay alguno le lanzaremos un mensaje al usuario de que no existe ningún proceso activo, en el caso contrario le mostraremos una lista en la cual venga toda la información que se esta llevando a cabo con ayuda de un método peekFirst(), que lo que hará es acceder al primer proceso de la cola sin eliminarlo para que de esa forma podamos mostrar toda su información.

```java
Proceso procesoActivo = colaDeProcesos.peekFirst(); // Accedemos al primer proceso de la cola, sin eliminarlo.

//Se dejo asi por que solo se muestra un proceso, el que es el activo
System.out.println("Información del proceso activo:");
System.out.println("Nombre del proceso: " + procesoActivo.getNombre());
System.out.println("ID único: " + procesoActivo.getId());
System.out.println("Instrucciones totales: " + procesoActivo.getInstruccionesTotales());
System.out.println("Instrucciones ejecutadas: " + procesoActivo.getInstruccionesEjecutadas());
```

```
Información del proceso activo:
Nombre del proceso: word
ID único: 1600
Instrucciones totales: 26
Instrucciones ejecutadas: 0
```

Después pasamos a la opción de ejecutar proceso actual, en está lo que hicimos fue acceder con ayuda del método anterior al primer elemento de nuestra cola de procesos, para que de esta forma ajustemos las instrucciones a realizar por nuestro proceso, si hay menos de 5 instrucciones por ejecutar se ajustaran dichas instrucciones, si quedan menos de y procesos

después del ajuste se ejecutaran dichos procesos para posteriormente instanciemos el método de procesos activos de la clase procesos y modifiquemos con ayuda del set las instrucciones ejecutadas más las instrucciones a realizar, de igual forma mostrara un mensaje a nuestro usuario de las instrucciones restantes mas el nombre del proceso que justamente se está ejecutando.

```
int instruccionesARealizar = Math.min(procesoActivo.getInstruccionesTotales() - procesoActivo.getInstruccionesEjecutadas(), 5);

// Restar las instrucciones ejecutadas del total.
procesoActivo.setInstruccionesEjecutadas(procesoActivo.getInstruccionesEjecutadas() + instruccionesARealizar);

// Mostrar mensaje de ejecución.
System.out.println("El proceso " + procesoActivo.getNombre() + " se ha ejecutado.");
System.out.println("Instrucciones restantes: " + (procesoActivo.getInstruccionesTotales() - procesoActivo.getInstruccionesEjecutadas()));
```

Por último, se evalúa con un if las instrucciones restantes, si las instrucciones totales son iguales a instrucciones ejecutadas le mostraremos a nuestro usuario el mensaje de que se logró concluir toda la ejecución de su proceso y movemos dicho proceso a la cola de procesos finalizados con ayuda ahora del método pollFirst(), el cual moverá el proceso a la lista correctamente.

```
// Si las instrucciones restantes llegan a 0, el proceso ha terminado.
if (procesoActivo.getInstruccionesTotales() == procesoActivo.getInstruccionesEjecutadas()) {
    System.out.println("El proceso " + procesoActivo.getNombre() + " ha finalizado.");
    procesosFinalizados.addLast(colaDeProcesos.pollFirst()); // Mover el proceso a la lista de procesos finalizados.
}
```

```
El proceso word se ha ejecutado.
Instrucciones restantes: 21
```

Ahora pasemos al siguiente case que es para pasar al proceso siguiente, en este lo que hicimos fue generar tres casos para mover el proceso de una cola a otra, en este caso son tres case para la cola de E/S, la cola para solicitar una interrupción y al final de la cola de procesos, cualquiera de estos tres casos se le mostrara al usuario el proceso el cual ha sido movido o solicito una interrupción.

```
El proceso word ha sido movido al final de la cola de procesos.
Seleccione una opción:
1. Crear Proceso nuevo
2. Ver proceso activo
3. Ejecutar proceso actual
4. Pasar al proceso siguiente
```

Todo esto anterior se ve vinculado con los siguientes cases, el primero de ellos es el case ejecutar entrada y salida, este lo que hará es verificar si la cola se encuentra vacía, si es el caso contrario se accederá a la cola de entradas y salidas con ayuda de instanciar la clase de procesos para que de esa forma accedamos y removamos el primer proceso de la cola de E/S y después agreguemos ese proceso al final de la cola de procesos, de igual forma se le mostrara un mensaje al usuario de cual proceso fue movido a la cola de E/S para su ejecución.

```
public void ejecutarEntradaYSalida() {
    if (colaDeES.isEmpty()) {
        System.out.println("No hay procesos en la cola de entrada y salida.");
        return;
    }

    Proceso procesoDeES = colaDeES.poll(); // Accedemos y removemos el primer proceso de la cola de E/S.
    colaDeProcesos.addLast(procesoDeES);   // Añadimos el proceso al final de la cola de procesos preparados.
```

```
El proceso Powerpoint ha sido movido a la cola de entrada y salida.
Seleccione una opción:
```

El siguiente case que viene en conjunto es el de ejecutar interrupción, esta hará lo mismo que el case anterior, pero con la diferencia de que la cola a la que se accedera es a la cola de interrupciones para poder remover dicho proceso y moverlo a la cola de procesos preparados.

```
public void ejecutarInterrupcion() {
    if (colaDeInterrupciones.isEmpty()) {
        System.out.println("No hay procesos en la cola de interrupciones.");
        return;
    }

    Proceso procesoInterrumpido = colaDeInterrupciones.poll(); // Accedemos y removemos el primer proceso de la cola de interrupciones.
    colaDeProcesos.addLast(procesoInterrumpido);   // Añadimos el proceso al final de la cola de procesos preparados.
```

```
El proceso word ha sido movido de la cola de interrupciones a la cola de procesos preparados.
Seleccione una opción:
```

El siguiente case es uno de los mas interesantes que se realizaron el cual es el de matar el proceso actual, en este lo que hicimos fue acceder a nuestro primer elemento de la cola de procesos y de igual forma restamos a nuestra variable de memoria la memoria asignada al proceso que se removió, para que de esa forma se libere la memoria para la entrada de nuevos procesos, posterior a ello lo que hacemos es mandar ese proceso a nuestra cola de procesos eliminados el proceso activo de ese momento y le mostramos al usuario cual fue el proceso y las instrucciones faltantes a realizar.

```
Proceso procesoActivo = colaDeProcesos.poll(); // Accedemos y removemos el primer proceso de la cola de procesos.
memoriaUsada -= procesoActivo.getMemoriaAsignada(); // Liberamos la memoria que estaba utilizando el proceso.
procesosEliminados.add(procesoActivo); // Añadimos el proceso al registro de procesos eliminados.

System.out.println("El proceso " + procesoActivo.getNombre() + " ha sido eliminado.");
```

```
El proceso word ha sido eliminado.
Instrucciones pendientes del proceso: 11
Seleccione una opción:
```

Estos fueron los case mas importantes ya que justamente son los que definen el funcionamiento de nuestro programa, ahora pasemos con los case mas comprensibles ya que justo son los que nos muestran la información de los procesos, por ejemplo, el caes de imprimir la lista de procesos preparados. Éste lo que hará es recorrer nuestra cola de procesos preparados con ayuda de un for each, mientras lo hace le mostrara al usuario en pantalla información del proceso como su nombre, su id, instrucciones pendientes y su memoria asignada, esto con ayuda del método get.

```
for (Proceso proceso : colaDeProcesos) {
    System.out.println("-----------------------------------");
    if (index == 0) {
        System.out.println(">> PROCESO ACTIVO <<");
    }

    //Se dejo asi y no se uso la otra funcion por que se muestra el proceso activo
    System.out.println("Nombre del proceso: " + proceso.getNombre());
    System.out.println("ID del proceso: " + proceso.getId());
    System.out.println("Instrucciones pendientes: " + (proceso.getInstruccionesTotales() - proceso.getInstruccionesEjecutadas()));
    System.out.println("Dirección de memoria asignada: " + proceso.getMemoriaAsignada());
    index++;
```

```
Lista de procesos preparados:
-----------------------------------
>> PROCESO ACTIVO <<
Nombre del proceso: Minecraft
ID del proceso: 8872
Instrucciones pendientes: 14
Dirección de memoria asignada: 128
-----------------------------------
```

Esto mismo se realizará para los siguientes case que son de imprimir la cola de E/S y la cola de procesos pendientes, pero con la diferencia de que se hará la instancia de las respectivas colas de cada uno de los tipos de procesos que haya junto con el método de datos del proceso el cual es el que ayuda a mostrar dicha información.

```java
//Metodo que muestra los datos de los procesos
public void DatosProceso(LinkedList<Proceso> colaProcesos) {
    for (Proceso proceso : colaProcesos) {
        System.out.println("----------------------------------");
        System.out.println("Nombre del proceso: " + proceso.getNombre());
        System.out.println("ID del proceso: " + proceso.getId());
        System.out.println("Instrucciones pendientes: " + (proceso.getInstruccionesTotales() - proceso.getInstruccionesEjecutadas()));
        System.out.println("Dirección de memoria asignada: " + proceso.getMemoriaAsignada());
    }
    System.out.println("----------------------------------");
}
```

```
Lista de procesos en cola de E/S:
----------------------------------
Nombre del proceso: Powerpoint
ID del proceso: 5340
Instrucciones pendientes: 13
Dirección de memoria asignada: 512
----------------------------------
```

```
Lista de procesos pendientes de interrupción:
----------------------------------
Nombre del proceso: Excel
ID del proceso: 6202
Instrucciones pendientes: 25
Dirección de memoria asignada: 64
----------------------------------
```

Por último, pasamos al case el cual es ver el estado actual del sistema, este como su mismo nombre lo dice le mostrara al usuario el estado actual del sistema o mejor dicho del simulador del procesos, se mostraran datos tales como número de procesos de cola preparados, en la cola de E/S, los procesos pendientes (En este nos apoyamos de un método el cual es imprimir procesos pendientes, el cual recorre la lista de procesos pendientes de interrupción y muestra los datos como en los anteriores case de imprimir lista), los procesos finalizados exitosamente como los no finalizados exitosamente, al igual y por ultimo los procesos los cuales fueron finalizados antes de tiempo (eliminados), todo eso se hace evaluando si las listas de esos procesos están vacías y accediendo a ellas con los métodos size y get para obtener los datos de los nombres de los procesos y el número de instrucciones que tengan.

```java
public void verEstadoActualSistema() {
    // Número de procesos en cola de preparados
    System.out.println("Número de procesos en cola de preparados: " + colaDeProcesos.size());

    // Número de procesos en cola de E/S
    System.out.println("Número de procesos en cola de E/S: " + colaDeES.size());

    // Mostrar lista de procesos pendientes de ejecutar interrupción
    imprimirProcesosPendientesInterrupcion();
```

```java
// Lista de procesos finalizados antes de tiempo (eliminados)
System.out.println("Procesos finalizados antes de tiempo (eliminados):");
if (procesosEliminados.isEmpty()) {
    System.out.println("No hay procesos eliminados.");
} else {
    for (Proceso proceso : procesosEliminados) {
        System.out.println("- " + proceso.getNombre());
    }
}
```

```
Número de procesos en cola de preparados: 1
Número de procesos en cola de E/S: 1
Lista de procesos pendientes de interrupción:
---------------------------------
Nombre del proceso: Excel
ID del proceso: 6202
Instrucciones pendientes: 25
Dirección de memoria asignada: 64
---------------------------------
Procesos finalizados exitosamente:
No hay procesos finalizados exitosamente.
Procesos finalizados antes de tiempo (eliminados):
- word
```

Como extra, si el usuario desea salir del programa se le mostrara una advertencia de que la cantidad de procesos que no se concluirán exotistamente, eso se muestra en el caso de que la cola de procesos no este vacía. De igual forma el usuario puede regresar al programa si desea concluir todo exitosamente.

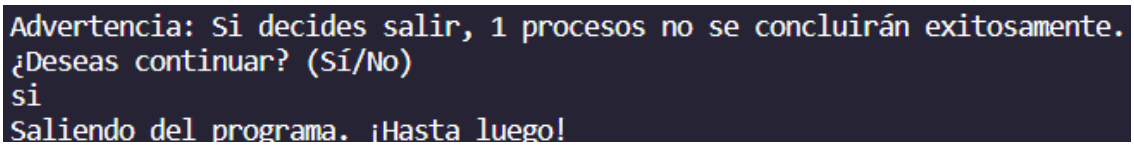
```java
public void salirDelPrograma() {
    Scanner scanner = new Scanner(System.in);

    // Verificar si hay procesos en la cola de preparados
    if (!colaDeProcesos.isEmpty()) {
        System.out.println("Advertencia: Si decides salir, " + colaDeProcesos.size() + " procesos no se concluirán exitosamente.");

        while (true) {
            System.out.println("¿Deseas continuar? (Sí/No)");

            String respuesta = scanner.nextLine().trim().toLowerCase();
```

```
Advertencia: Si decides salir, 1 procesos no se concluirán exitosamente.
¿Deseas continuar? (Sí/No)
no
Regresando al menú principal...
```



Conclusiones

Salgado Miranda Jorge

La realización de esta práctica resultó ser fundamental para comprender a profundidad los mecanismos intrínsecos del funcionamiento de los sistemas operativos y la gestión de procesos y memoria. A través de la simulación, pudimos experimentar en primera persona las decisiones y desafíos que enfrenta un sistema operativo en tiempo real. Este tipo de prácticas nos permite no solo entender la teoría detrás de los conceptos, sino también experimentar su aplicación práctica, consolidando así el aprendizaje. La creación del simulador y el enfrentamiento a sus desafíos técnicos y logísticos reafirmó la relevancia de aplicar conocimientos teóricos en escenarios prácticos. Además, el proceso nos enseñó la importancia de la adaptabilidad y la gestión eficiente del tiempo, especialmente cuando se enfrentan imprevistos. En definitiva, llevar a cabo esta práctica no solo fortaleció nuestra comprensión técnica, sino que también nos brindó habilidades valiosas para la vida profesional y académica.

Flores Chávez Marcos Gabriel

Una vez concluida esta práctica logré comprender de una mejor manera el como es que funcionan los procesos en nuestro sistema operativo y como es que esto nos puede dar hincapié a poder generar lo que son algoritmos de planificación de procesos los cuales nos servirían bastante para poder eficientar el trabajo de los procesos y por su puesto nuestro sistema operativo o en este caso nuestro simulador. También durante la realización de esta practica presentamos algunas dificultades y problemas técnicos sobre todo a la hora de organizar bien los trabajos de los procesos ya que en algunas ocasiones nos salían excepciones, pero esto justo nos sirvió para poder entender de una mejor manera todo lo que conlleva la ejecución de un proceso, al final logramos cumplir con todos los objetivos de esta práctica y fortalecer los conocimientos vistos en la clase de teoría.