Universidad Nacional Autónoma

De México

Facultad De Ingeniería

Materia: Sistemas Operativos

Proyecto Final

Integrantes:

-Hernández Padilla Hugo

-Marín Montaño Josué

- Rufino López María Elena

- Ruiz Bastián Oscar

Semestre 2025-1

Grupo: 02

Profesor: M. A. José Alberto Avalos Velez

Fecha de entrega:20/Noviembre/2024

Introducción

Un **sistema operativo** (SO) es el componente esencial del software que actúa como intermediario entre los usuarios, las aplicaciones y el hardware de una computadora. Su objetivo principal es gestionar de manera eficiente los recursos del sistema, como la memoria, el procesador, los dispositivos de entrada/salida y los archivos. Además, el sistema operativo proporciona una interfaz para que los usuarios y los programas interactúen con el hardware sin preocuparse por los detalles técnicos.

Un **proceso** es una instancia de un programa en ejecución. No se limita solo a las instrucciones del programa, sino que incluye el estado de ejecución actual, los recursos asignados y los datos necesarios para su ejecución.

Un proceso consta de los siguientes elementos principales:

- Código ejecutable: El conjunto de instrucciones que componen el programa.
- Estado del proceso: Indica si el proceso está en ejecución, en espera o terminado.
- Recursos asignados: Incluyen memoria, tiempo de CPU y acceso a dispositivos de entrada/salida.
- Información de control: Contiene detalles como las prioridades, los punteros al programa y los registros actuales.

El sistema operativo administra los procesos a través de:

1. Planificación: Decide el orden y la cantidad de tiempo que cada proceso puede usar el procesador.
2. Gestión de recursos: Asigna memoria y dispositivos a los procesos de manera eficiente.
3. Sincronización y comunicación: Coordina la interacción entre procesos para evitar conflictos y garantizar un flujo de trabajo correcto.

En los sistemas operativos, la **planificación de procesos** es una de las funciones más críticas. Consiste en decidir qué proceso, de entre los que están listos para ejecutarse, utilizará el procesador en un momento dado. Este mecanismo asegura que los recursos del sistema se asignen de manera eficiente y que los procesos se ejecuten de manera justa y ordenada.

La importancia de la planificación de procesos radica en:

- Eficiencia: Garantiza que el procesador esté ocupado ejecutando procesos siempre que sea posible.
- Equidad: Proporciona oportunidades justas a todos los procesos para acceder al CPU.
- Rendimiento: Optimiza el tiempo promedio de espera y el tiempo de retorno de los procesos.
- Prioridad: Permite que procesos más críticos o urgentes sean atendidos antes que los menos importantes.

Los procesos atraviesan diferentes estados en su ciclo de vida:

1. Nuevo: El proceso acaba de crearse.
2. Listo: El proceso está preparado para ejecutarse, pero está esperando turno.
3. Ejecución: El proceso está utilizando el procesador.
4. Bloqueado: El proceso está esperando por un evento externo, como entrada/salida.
5. Terminado: El proceso ha finalizado su ejecución.

Análisis

Planificación de Procesos

La **planificación de procesos** es una de las tareas esenciales de un sistema operativo, encargada de gestionar el uso del procesador por parte de los procesos en ejecución. Para el simulador, se implementarán tres algoritmos fundamentales:

- **FIFO (First In, First Out)**:
    - Los procesos se ejecutan en el orden en que llegan a la cola.
    - Es sencillo y proporciona una visión básica de la planificación secuencial.
- **Round Robin**:
    - Los procesos se ejecutan en intervalos de tiempo fijos (*quantum*), alternando entre ellos.
    - Representa un sistema multitarea justo y eficiente, adecuado para entornos interactivos.
- **Shortest Job First (SJF)**:
    - Prioriza los procesos con menor tiempo de ejecución.
    - Es útil para minimizar tiempos de espera y retorno, aunque requiere conocer el tiempo de ejecución de antemano.

Administración de Memoria

La **administración de memoria** es crucial para asignar y gestionar eficientemente los recursos de memoria de los procesos. En el simulador, se implementará un

esquema de **paginación** o **segmentación** para ilustrar cómo los sistemas operativos manejan la memoria.

- **Paginación**:
    - Divide la memoria física en bloques de tamaño fijo llamados **marcos** y la memoria lógica en **páginas**.
    - Permite asignar páginas de procesos a marcos disponibles en la memoria.
- **Segmentación**:
    - Divide la memoria lógica en segmentos de tamaño variable según las partes del programa, como código, datos y pila.
    - Proporciona flexibilidad y un modelo lógico más cercano al programa.

Algoritmos de reemplazo de páginas:

1. **FIFO**:
    - Reemplaza la página más antigua cuando ocurre un fallo de página.
2. **LRU (Least Recently Used)**:
    - Reemplaza la página menos recientemente utilizada, optimizando el acceso futuro.

Sistema de Archivos

El **sistema de archivos** es el componente encargado de organizar y almacenar los datos en el sistema operativo. En el simulador, se desarrollará un sistema básico que permita operaciones comunes de manipulación de archivos y directorios.

Operaciones implementadas:

1. **mkdir**: Crear un nuevo directorio en el sistema.
2. **touch**: Crear un nuevo archivo, inicialmente vacío.
3. **rm**: Eliminar un archivo o directorio existente.
4. **ls**: Listar el contenido del directorio actual.

Desarrollo

Diagramas UML

Gestión de clases de procesos

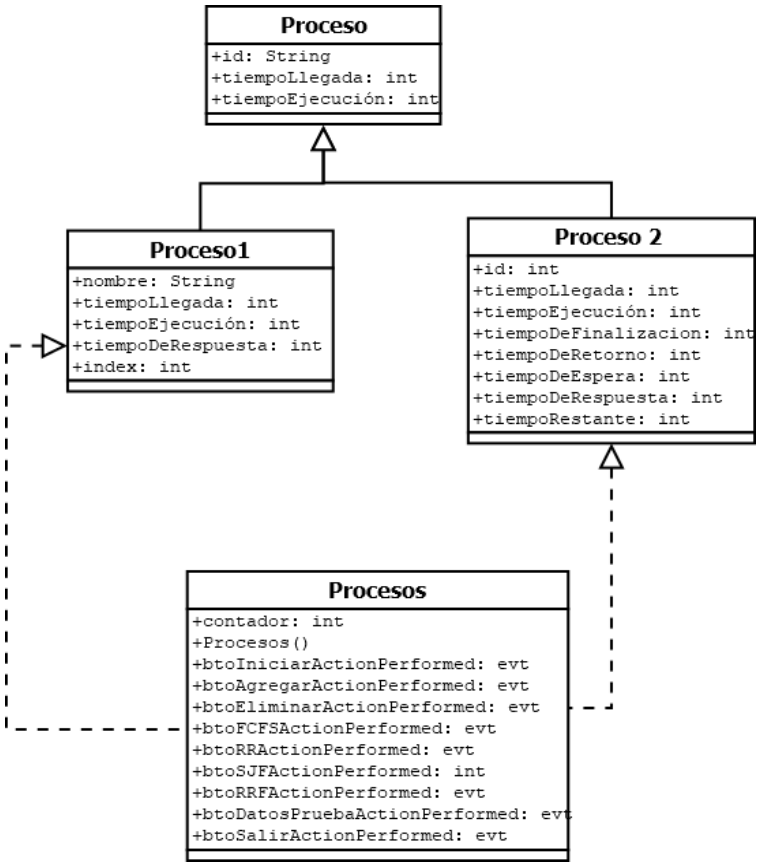


Diagrama de Clases para el Sistema de Archivos

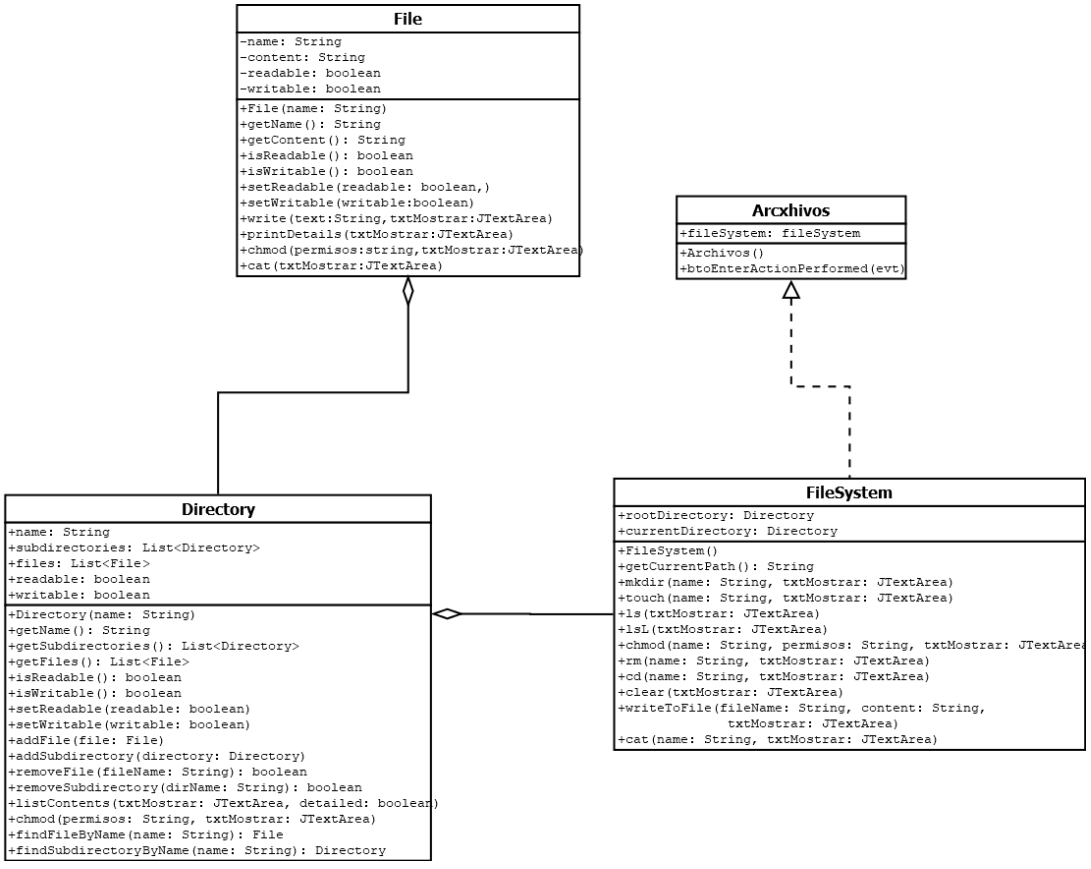


Diagrama de Secuencia para la Creación de un Archivo

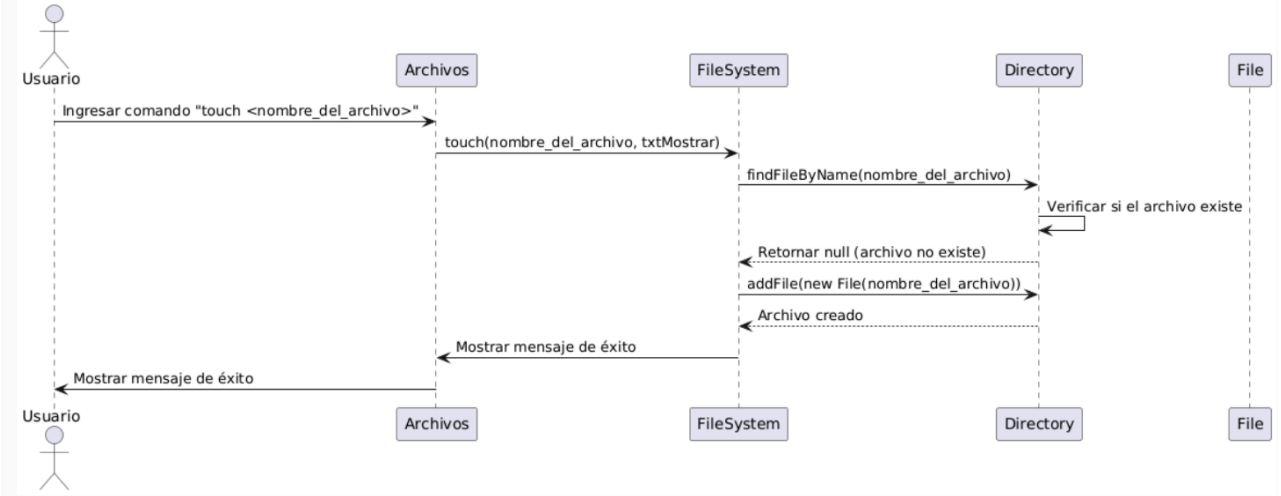
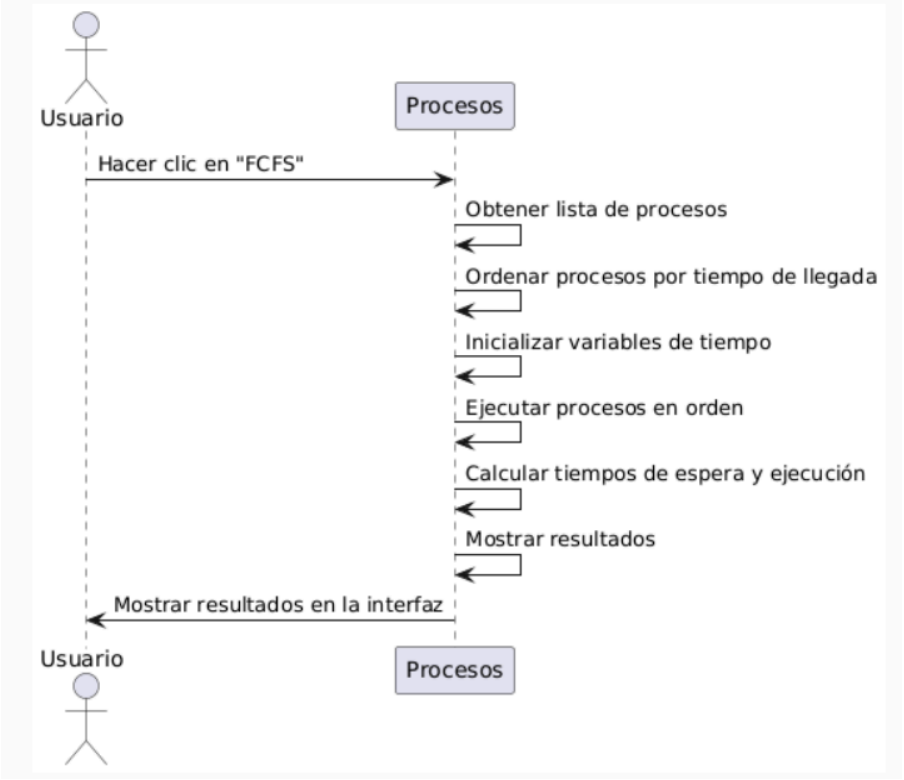


Diagrama de Secuencia para la Ejecución del Algoritmo FCFS



Detalles Técnicos de implementación

1. Principal.java

Este archivo actúa como el controlador principal y punto de entrada del programa. Maneja la inicialización de la interfaz gráfica y permite la navegación entre los módulos del simulador.

- **Funciones Principales:**
  - Inicializa los módulos de gestión de procesos y sistema de archivos.
  - Proporciona botones y eventos para que el usuario pueda navegar fácilmente entre diferentes funcionalidades.
- **Interfaz gráfica**:
- **Tecnología**: Usa **Swing** para crear botones y áreas de interacción.
- **Opciones del Menú**:
  - Navegar hacia el módulo de gestión de procesos.
  - Navegar hacia el sistema de archivos.
  - Salir del simulador.

Fragmento de Código:

```
public static void main(String[] args) {
    java.awt.EventQueue.invokeLater(() -> {
        new Principal().setVisible(true);
    });
}
```

2. Archivos.java

Este archivo implementa las funcionalidades de un sistema de archivos básico, permitiendo operaciones comunes como crear, listar, y eliminar archivos y directorios.

- **Funciones principales**:
  - **mkdir**: Crea un nuevo directorio en la ruta actual.

```
public void mkdir(String name, JTextArea txtMostrar) {
    if (currentDirectory.findSubdirectoryByName(name) != null) {
        txtMostrar.append("Error: Ya existe un directorio con ese nombre.\n");
    } else {
        Directory newDir = new Directory(name);
        currentDirectory.addSubdirectory(newDir);
        txtMostrar.append("Directorio '" + name + "' creado con éxito.\n");
    }
}
```

- **touch**: Crea un nuevo archivo.

```
public void touch(String name, JTextArea txtMostrar) {
    if (currentDirectory.findFileByName(name) != null) {
        txtMostrar.append("Error: Ya existe un archivo con el nombre '" + name + "'.\n");
    } else {
        File newFile = new File(name);
        currentDirectory.addFile(newFile);
        txtMostrar.append("Archivo '" + name + "' creado con éxito.\n");
    }
}
```

- **rm**: Elimina un archivo o directorio existente.

```
public void rm(String name, JTextArea txtMostrar) {
    if (!currentDirectory.removeFile(name)) {
        if (!currentDirectory.removeSubdirectory(name)) {
            txtMostrar.append("No se encontró el archivo o directorio '" + name + "'.\n");
        } else {
            txtMostrar.append("Directorio '" + name + "' eliminado.\n");
        }
    } else {
        txtMostrar.append("Archivo '" + name + "' eliminado.\n");
    }
}
```

- **Clases importantes**:
  - **File**: Representa un archivo con atributos como nombre, contenido, y permisos (readable y writable).

```
private String name;
private String content;
private boolean readable;
private boolean writable;
```

  - **Directory**: Representa un directorio que puede contener archivos y subdirectorios.

```
private List<Directory> subdirectories;
private List<File> files;
```

- **FileSystem**: Encapsula las operaciones del sistema de archivos y gestiona la estructura jerárquica de directorios.

```
private Directory rootDirectory;
private Directory currentDirectory;
```

3. Procesos.java

Este módulo implementa la planificación de procesos y simula la administración de memoria.

- **Gestión de procesos**:
  - Los procesos tienen atributos como nombre, tiempo de llegada, tiempo de ejecución, y tamaño.
  - Permite registrar procesos y actualiza dinámicamente la memoria disponible:

```
int tamannio = Integer.parseInt(txtTamPro.getText());
if (tamannio > memoriaRestante) {
    JOptionPane.showMessageDialog(null, "No hay memoria para otro proceso");
} else {
    modelo.addRow(datos);
    memoriaRestante -= tamannio;
    txtTamMemRes.setText(String.valueOf(memoriaRestante));
}
```

- **Algoritmos de planificación implementados**:
  - **FIFO (First In, First Out)**: Ejecuta los procesos en el orden en que llegan.

```
procesos.sort((p1, p2) -> Integer.compare(p1.tiempoLlegada, p2.tiempoLlegada));
```

  - **SJF (Shortest Job First)**: Prioriza los procesos con menor tiempo de ejecución.

```
procesos.sort(Comparator.comparingInt(p -> p.tiempoEjecucion));
```

  - **Round Robin**: Distribuye los recursos del CPU en intervalos de tiempo fijos (*quantum*).

```
while (!cola.isEmpty()) {
    Proceso2 proceso = cola.poll();
    int tiempoProcesado = Math.min(proceso.tiempoRestante, quantum);
    proceso.tiempoRestante -= tiempoProcesado;
```

```
    // Actualizar tiempos de espera y respuesta...
}
```

Gestión de memoria:

- Se implementa un esquema simple de asignación de memoria basado en bloques, representado por tablas en la interfaz
- Los métodos actualizan dinámicamente la memoria restante y asignada según los procesos cargados.
- Representada mediante tablas gráficas (tablaIniPag y tablaPag), que reflejan los estados de la memoria.

```
Object [] datosRes = {dirTabla, "---", "-------", tamB};
DefaultTableModel modelo = (DefaultTableModel) tablaPag.getModel();
modelo.addRow(datosRes);
```

Estructuras de Datos Utilizadas

1. **ArrayList**:
    - Almacena procesos y directorios en el sistema.
    - Permite ordenar y buscar elementos de manera eficiente.
2. **DefaultTableModel**:
    - Maneja las tablas que representan los estados de memoria y procesos en la interfaz gráfica.
3. **Clases personalizadas**:
    - **File**: Define los atributos y métodos para manejar archivos.
    - **Directory**: Permite gestionar la estructura jerárquica de archivos y carpetas.

Interfaz Gráfica

- **Swing**:
    - Se utiliza para crear ventanas y elementos interactivos como botones, áreas de texto y tablas.
    - Ejemplo:
        - Botones como btoAgregar, btoEliminar, y btoIniciar para gestionar procesos.
        - Área de texto txtComandos para ingresar comandos en el sistema de archivos.

Pruebas

En las siguientes imágenes se muestra la interfaz gráfica de nuestro programa. La interfaz está organizada en siete secciones principales, cada una con un propósito específico. Como se muestra en el manual de usuario.
En este ejemplo podemos ver que ingresamos el tamaño de la memoria en 190000 con 4 procesos, ejecutando los procesos con FCFS. También se muestra la memoria inicial, la ejecución paso a paso y la memoria final

En esta imagen se muestra como funciona de forma correcta el planificador de procesos RR

Y por último un ejemplo del planificador de proceso SIF.

Con esto se muestra que nuestro programa funciona de forma correcta cumpliendo los objetivos planteados al inicio de este proyecto, agregamos funciones adicionales que ayudan a mejorar el entendimiento de los conceptos fundamentales en el diseño y funcionamiento de los sistemas operativos modernos. Al término de este proceso, se pudo concluir que la instalación y configuración de un sistema operativo requiere atención a los detalles en cada paso para garantizar su correcto funcionamiento.

Bibliografía

- *Introducción a los algoritmos de planificación de procesos*. (n.d.). https://introduccion-a-los-algor-of0emlw.gamma.site/
- *Gestión de Procesos en Sistemas Operativos*. (n.d.). https://sistemas-operativos02-3ztay28.gamma.site/
- *Concurrencia*. (n.d.). https://concurrencia-h2r3vny.gamma.site/
- *Introducción a la gestión de procesos e hilos*. (n.d.). https://procesos-kfglgqv.gamma.site/
- *Introducción al planificador de procesos*. (n.d.). https://introduccion-al-planific-8vjxrsy.gamma.site/
- *Gestión de Procesos en Sistemas Operativos*. (n.d.). https://sistemas-operativos02-3ztay28.gamma.site/
- *Sistema de Archivos*. (n.d.). https://sistema-de-archivos-et2dxrc.gamma.site/
- *Directorio Raíz*. (n.d.). https://directorio-raiz-tbef2mi.gamma.site/
- *Sistema de Archivos*. (n.d.). https://sistema-de-archivos-et2dxrc.gamma.site/
- *Administración de la Memoria: Un Vistazo al Interior*. (n.d.). https://administracion-de-la-mem-1ld8tx0.gamma.site/