UMAG Universidad de Magallanes | Facultad de Ingeniería

Simulador de sistemas operativos

Mario Calbuyahue[†], **Felipe Paillacar**[†] **y Constanza Araya**[†]

[†] *Universidad de Magallanes*

This manuscript was compile on 29 de octubre de 2024

Resumen

Derrollo de un simulador de sistema operativo, aplicando estructuras de datos fundamentales (arreglos, listas enlazadas, pilas, colas y listas circulares). Este simulador gestionará la memoria y la planificación de procesos, asignando y liberando dinámicamente memoria, administrando el stack de cada proceso y manejando las colas de espera o procesos suspendidos.

Keywords: *Simulación, sistema operativo, estructuras de datos, planificación de procesos, gestión de memoria*

Índice

1. Introduccion

En ste proyecto se desarrolla un simulador de sistema operativo básico, cuyo objetivo es aplicar y profundizar en conceptos fundamentales de estructuras de datos en la gestión de memoria y planificación de procesos. Este simulador emula varios aspectos esenciales de un sistema operativo, como la asignación dinámica y liberación de memoria, la gestión de pilas para cada proceso y la planificación eficiente de su ejecución.

A través de esta simulación, se implementan y exploran estructuras de datos clave, como listas enlazadas, pilas, colas y listas circulares, así como algoritmos de asignación de memoria (First Fit, Best Fit y Worst Fit) y de planificación de procesos (FIFO, Round Robin y SJF). Cada uno de estos componentes se integra para lograr una representación simplificada de los mecanismos de administración de recursos que utiliza un sistema operativo real.

El simulador está diseñado para gestionar de manera dinámica múltiples procesos, asignándoles memoria y orden de ejecución de acuerdo con los algoritmos seleccionados.

2. Objetivo del Proyecto

El objetivo principal de este proyecto es aplicar conceptos fundamentales de estructuras de datos en la simulación de un sistema operativo básico, enfocado en la gestión de memoria y la planificación de procesos. Este simulador busca integrar estructuras de datos como listas enlazadas, pilas, colas y listas circulares para gestionar eficazmente la memoria y los recursos de CPU, implementando distintos algoritmos de asignación y planificación que optimicen el uso de recursos.

3. Descripción del Simulador

Este simulador gestiona de forma dinámica la memoria y la planificación de procesos, imitando el comportamiento de un sistema operativo real. Para ello, asigna y libera memoria de forma dinámica, maneja las pilas de cada proceso, y organiza colas de procesos en distintos estados. También implementa algoritmos de planificación como FIFO, Round Robin y SJF (Shortest Job First) para determinar el orden de ejecución, permitiendo gestionar múltiples procesos de forma eficiente.

4. Estructuras de Datos Utilizadas

4.1. Arreglos

Se utilizan arreglos para representar el espacio de memoria dividido en bloques. Este diseño permite un acceso rápido y directo a los bloques de memoria y facilita el manejo de fragmentación interna y externa.

4.2. Lista enlazada simple

Las listas enlazadas simples se utilizan para manejar la lista de procesos en memoria. Estas listas permiten la flexibilidad de añadir o eliminar procesos de forma dinámica y son fundamentales para la gestión de memoria en la asignación dinámica de bloques.

4.3. Pilas

Se utiliza una pila para manejar el stack de cada proceso, simulando la estructura de llamadas de función y manejo de variables locales para cada proceso. Esta estructura refleja el uso de memoria temporal para cada llamada de función y facilita el cambio de contexto entre procesos.

4.4. Colas

Las colas representan la lista de procesos en espera o suspendidos. Cada cola permite almacenar procesos en diferentes estados, como "Listo.º "Bloqueado", permitiendo al sistema decidir el proceso a ejecutar de acuerdo con las condiciones de disponibilidad de memoria y CPU.

4.5. Listas circulares

Se emplean listas circulares para implementar la rotación de procesos en el algoritmo de planificación tipo Round Robin, permitiendo que los procesos ejecuten en turnos de tiempo fijos y vuelvan al final de la cola de espera tras completar su quantum.

4.6. Estructuras de proceso y estado

Arreglos y listas de estructuras (struct) se utilizan para almacenar la información de cada proceso y su estado actual, tales como "Nuevo", "Listo", .Ejecutando", "Bloqueadoz "Finalizado". Esta estructura permite al sistema gestionar de forma clara y eficiente las transiciones de estado de los procesos según los recursos disponibles.

Las estructuras de datos son fundamentales en la implementación de este simulador de sistema operativo, ya que permiten gestionar eficazmente la memoria y el flujo de los procesos. A continuación, se muestran fragmentos de código que ilustran las estructuras de datos clave utilizadas:

```c
typedef struct BloqueMemoria {
    int tamano;
    int libre;
    struct BloqueMemoria *siguiente;
} BloqueMemoria;
```

Código 1. Estructura de bloque de memoria en lista enlazada

Esta estructura ‘BloqueMemoria‘ permite almacenar bloques de memoria en una lista enlazada, donde cada nodo representa un bloque de memoria del sistema. Esta lista facilita la asignación y liberación de memoria en algoritmos como First Fit, Best Fit y Worst Fit.

```c
typedef struct Proceso {
    char nombre[10];
    int tiempo_ejecucion;
    int memoria_solicitada;
    struct Proceso *siguiente;
} Proceso;
```

Código 2. Estructura de proceso en cola para planificación

La estructura ‘Proceso‘ define cada proceso en la simulación, con datos básicos como el nombre, el tiempo de ejecución y la memoria solicitada. Esta estructura se emplea en la implementación de la cola de procesos en espera de ejecución.

5. Algoritmos de Asignación de Memoria

Los algoritmos de asignación de memoria implementados en el simulador son First Fit, Best Fit y Worst Fit, cada uno con un enfoque diferente en la gestión de fragmentación:
First Fit: asigna el primer bloque libre suficientemente grande para el proceso. Es el más rápido en términos de tiempo de búsqueda, aunque tiende a producir fragmentación interna.
Best Fit: selecciona el bloque más pequeño disponible que sea suficientemente grande para el proceso, minimizando la fragmentación interna. Sin embargo, su eficiencia se ve limitada por la necesidad de recorrer todos los bloques para encontrar el más adecuado.
Worst Fit: asigna el bloque más grande disponible al proceso, con el objetivo de reducir la fragmentación externa. No obstante, este algoritmo puede ser menos eficiente cuando existen pocos bloques grandes.
Cada algoritmo se implementa en la función asignar memoria, que recorre la lista de bloques de memoria y evalúa las condiciones de cada algoritmo antes de asignar el bloque adecuado.
A continuación, se presentan fragmentos de código de cada uno de estos algoritmos:

```
int asignar_memoria_first_fit(int tamano) {
    BloqueMemoria *bloque = memoria;
    while (bloque != NULL) {
        if (bloque->libre && bloque->tamano >= tamano) {
            bloque->libre = 0;  // Marcar bloque como ocupado
            return 1;
        }
        bloque = bloque->siguiente;
    }
    return -1;
}
```

Código 3. Algoritmo de Asignación de Memoria - First Fit

El algoritmo ‘First Fit‘ recorre la lista de bloques y asigna el primer bloque libre que tenga suficiente espacio para el proceso.

```
int asignar_memoria_best_fit(int tamano) {
    BloqueMemoria *mejor_bloque = NULL;
    BloqueMemoria *bloque = memoria;
    while (bloque != NULL) {
        if (bloque->libre && bloque->tamano >= tamano) {
            if (mejor_bloque == NULL || bloque->tamano < mejor_bloque->tamano) {
                mejor_bloque = bloque;
            }
        }
        bloque = bloque->siguiente;
    }
    if (mejor_bloque != NULL) {
        mejor_bloque->libre = 0;  // Marcar bloque como ocupado
        return 1;
    }
    return -1;
}
```

Código 4. Algoritmo de Asignación de Memoria - Best Fit

El ‘Best Fit‘ busca el bloque más pequeño que sea suficientemente grande para el proceso, minimizando la fragmentación interna.

6. Algoritmos de Planificación de Procesos

Para gestionar la ejecución de procesos, se implementan los algoritmos FIFO, Round Robin y SJF:
FIFO (First In, First Out):los procesos se ejecutan en el orden en que llegan. Es sencillo de implementar y garantiza la ejecución en el orden de llegada, aunque es ineficiente en situaciones con procesos largos al inicio de la cola.
Round Robin: asigna un quantum de tiempo fijo a cada proceso. Una vez agotado, el proceso vuelve a la cola de ejecución. Este algoritmo es útil en sistemas interactivos, ya que asegura que todos los procesos reciben tiempo de CPU periódicamente.
SJF (Shortest Job First):selecciona el proceso con el menor tiempo de ejecución primero, minimizando el tiempo de espera promedio. Este algoritmo es eficiente, aunque puede generar inanición en procesos largos si continuamente se agregan procesos cortos.
Cada algoritmo se implementa en la función planificar, que organiza y ejecuta los procesos en función del tipo de algoritmo seleccionado, respetando el orden de llegada o el quantum (en el caso de Round Robin).
A continuación, se presentan los fragmentos de código de los algoritmos FIFO, Round Robin y SJF:

```c
void planificar_fifo() {
    Proceso *proceso = cola_procesos;
    while (proceso != NULL) {
        ejecutar_proceso(proceso);
        proceso = proceso->siguiente;
    }
}
```

Código 5. Algoritmo de Planificación - FIFO

El algoritmo 'FIFO' ejecuta los procesos en el orden en que llegaron, sin interrupciones.

```c
void planificar_round_robin(int quantum) {
    Proceso *proceso = cola_procesos;
    while (proceso != NULL) {
        if (proceso->tiempo_ejecucion > quantum) {
            ejecutar_proceso_parcial(proceso, quantum);
            proceso->tiempo_ejecucion -= quantum;
            agregar_a_cola(proceso);  // Regresa el proceso a la cola
        } else {
            ejecutar_proceso(proceso);
        }
        proceso = proceso->siguiente;
    }
}
```

Código 6. Algoritmo de Planificación - Round Robin

El 'Round Robin' alterna la ejecución de cada proceso en intervalos de tiempo definidos (quantum), regresando el proceso a la cola si no finaliza.

```c
void planificar_sjf() {
    Proceso *proceso = encontrar_proceso_menor_tiempo(cola_procesos);
    while (proceso != NULL) {
        ejecutar_proceso(proceso);
        proceso = encontrar_proceso_menor_tiempo(cola_procesos);
    }
}
```

Código 7. Algoritmo de Planificación - SJF (Shortest Job First)

El 'SJF' selecciona el proceso con el menor tiempo de ejecución, optimizando el tiempo de espera promedio.

7. Desiciones de diseño

Modularidad: la estructura modular del simulador permite una separación clara de las funcionalidades, lo que facilita la comprensión y el mantenimiento del código.
Gestión Dinámica: las listas enlazadas y pilas fueron seleccionadas por su flexibilidad y adaptabilidad en la gestión dinámica de procesos y memoria, mientras que las listas circulares optimizan la rotación en el algoritmo Round Robin.
Eficiencia de Algoritmos: se decidió implementar tanto algoritmos rápidos como First Fit y FIFO para escenarios menos críticos, así como algoritmos más complejos como Best Fit y SJF para optimizar el rendimiento en casos específicos.
Las decisiones de diseño incluyen la estructura modular del sistema, el uso de listas enlazadas para bloques de memoria y colas para procesos. A continuación, se muestra cómo se inicializa la lista de bloques de memoria:

```c
void inicializar_memoria(int *tamano_bloques, int cantidad) {
    for (int i = 0; i < cantidad; i++) {
        BloqueMemoria *nuevo_bloque = (BloqueMemoria*)malloc(sizeof(BloqueMemoria));
        nuevo_bloque->tamano = tamano_bloques[i];
        nuevo_bloque->libre = 1;
        nuevo_bloque->siguiente = memoria;
        memoria = nuevo_bloque;
    }
}
```

Código 8. Inicialización de Bloques de Memoria

Esta función configura la lista enlazada de bloques de memoria al inicio de la simulación, permitiendo que el sistema asigne y libere memoria de forma dinámica.

8. Resultados y Análisis

8.1. Resultados de Asignación de Memoria

El simulador fue probado bajo distintos algoritmos de asignación con resultados variados:
First Fit mostró un rendimiento alto en términos de velocidad, pero generó mayor fragmentación en configuraciones con bloques de memoria pequeños.
Best Fit logró un menor índice de fragmentación, aunque su rendimiento fue inferior en configuraciones con un gran número de procesos.
Worst Fit ayudó a reducir la fragmentación externa en configuraciones con bloques de gran tamaño, pero dejó memoria inutilizable cuando los bloques grandes eran asignados a procesos pequeños.
Los resultados de los algoritmos de asignación de memoria muestran diferencias en la eficiencia y fragmentación. A continuación se muestra un ejemplo de salida donde el algoritmo Best Fit asigna bloques de manera óptima en comparación con First Fit:

```
        Best Fit:
        - Proceso P1 asignado al bloque de 100KB
        - Proceso P2 asignado al bloque de 60KB

        First Fit:
        - Proceso P1 asignado al bloque de 120KB
        - Proceso P2 asignado al bloque de 80KB

```

8.2. Resultados de Planificación de Procesos

El análisis de los algoritmos de planificación reveló:
FIFO presentó tiempos de espera más largos en configuraciones donde el primer proceso tenía un tiempo de ejecución largo.
Round Robin mostró tiempos de respuesta regulares en configuraciones con un quantum pequeño, siendo ideal para sistemas interactivos.
SJF ofreció el menor tiempo de espera promedio, pero en configuraciones con procesos largos, ciertos procesos experimentaron inanición.
Los tiempos de espera y ejecución varían según el algoritmo de planificación. El siguiente ejemplo muestra cómo Round Robin distribuye el tiempo entre procesos con un quantum de 3:

```
        Round Robin (Quantum = 3):
        - Proceso P1 ejecutado por 3 unidades de tiempo
        - Proceso P2 ejecutado por 3 unidades de tiempo

```

8.3. Simulación de Fragmentación de Memoria

La fragmentación interna y externa se evaluó en cada algoritmo de asignación. Se observó que la fragmentación interna era mayor en First Fit y Best Fit, mientras que Worst Fit generaba menos fragmentación externa.

8.4. Simulación de Fragmentación de Memoria

En este simulador, la fragmentación de memoria se evalúa a partir de los algoritmos de asignación implementados (First Fit, Best Fit y Worst Fit). La fragmentación es un aspecto clave en la gestión de memoria, ya que afecta la capacidad del sistema para utilizar eficazmente los bloques disponibles.
El simulador permite simular fragmentación interna y externa a partir del uso dinámico de bloques de memoria, según los requerimientos de cada proceso. En el siguiente fragmento, el algoritmo First Fit asigna el primer bloque de memoria libre suficientemente grande para el proceso, lo cual puede dejar espacios de memoria sin utilizar (fragmentación interna).

```
        int asignar_memoria_first_fit(int tamano) {
            BloqueMemoria *bloque = memoria;
            while (bloque != NULL) {
                if (bloque->libre && bloque->tamano >= tamano) {
                    bloque->libre = 0;  // Marcar bloque como ocupado
                    return 1;
                }
                bloque = bloque->siguiente;
            }
            return -1;
        }

```

Código 9. Asignación de memoria en First Fit y manejo de fragmentación

En este ejemplo, **‘asignar memoria first fit‘** recorre la lista de bloques y asigna el primer bloque libre que tenga suficiente espacio para el proceso. En situaciones donde el bloque asignado es mayor que el tamaño solicitado, la diferencia representa la fragmentación interna para ese proceso. Cada algoritmo tiene un impacto particular en la disponibilidad de memoria y en la eficiencia general, proporcionando una visión de los efectos de la fragmentación en la simulación.

9. Conclusiones

La implementación de este simulador de sistema operativo nos permitió comprender y aplicar una variedad de conceptos esenciales en el ámbito de la gestión de memoria y la planificación de procesos. A través de la construcción de estructuras de datos como listas enlazadas, pilas, colas y listas circulares, logramos simular y gestionar de forma eficiente los recursos, abordando problemas comunes en sistemas operativos reales, tales como la fragmentación de memoria y la asignación de recursos limitados.

El simulador incorpora diferentes algoritmos de asignación de memoria (First Fit, Best Fit, Worst Fit) y planificación de procesos (FIFO, Round Robin, SJF), lo que nos permitió observar cómo cada enfoque afecta la eficiencia del sistema y el tiempo de espera de los procesos. Los resultados mostraron que cada algoritmo tiene fortalezas y debilidades específicas: mientras que First Fit es rápido y sencillo, Best Fit minimiza la fragmentación interna, y Worst Fit reduce la fragmentación externa. Asimismo, en la planificación de procesos, el algoritmo SJF optimiza el tiempo de espera promedio, aunque Round Robin resulta más equitativo en sistemas interactivos.

Más allá de la implementación técnica, este proyecto ha subrayado la importancia de la elección de estructuras de datos adecuadas para optimizar el rendimiento del sistema. La modularidad del código, junto con la clara separación de responsabilidades en los diferentes módulos (memoria, planificación y proceso), ha facilitado el mantenimiento y escalabilidad del sistema, mostrando la relevancia del diseño modular en sistemas complejos.