PARCIAL 2

SEBASTIAN IZQUIERDO SAAVEDRA

SISTEMAS OPERATIVOS

14/10/2025

Informe Parcial II – Asignación de Memoria

Descripción general

Este informe hace parte del segundo parcial del curso de Sistemas Operativos. En esta actividad se trabajó el tema de **asignación de memoria**, buscando entender de una forma práctica cómo los sistemas operativos administran la memoria.

El propósito principal fue **simular en C++ cómo se asigna y libera la memoria** usando distintos algoritmos, y analizar cómo se comporta cada uno frente a la fragmentación y la eficiencia del uso de espacio. La idea era no solo programar, sino también comprender la lógica detrás de cómo un sistema operativo decide dónde colocar cada proceso en memoria.

Desarrollo teórico-práctico

Para esta práctica se desarrolló un programa en **C++** que simula una memoria dividida en bloques. Desde la consola, el usuario puede hacer tres operaciones principales:

- **Asignar memoria (A)** a un proceso.
- **Liberar memoria (L)** de un proceso que ya terminó.
- **Mostrar el estado (M)** actual de la memoria.

Cada vez que se ejecuta una acción, el programa actualiza la estructura interna de la memoria, mostrando cuáles bloques están ocupados y cuáles están libres. Esto facilita ver cómo cambian los espacios disponibles después de cada operación.

El comportamiento del programa depende del algoritmo de asignación que el usuario elija: **First Fit**, **Best Fit** o **Worst Fit**. Cada uno tiene su propia forma de decidir dónde colocar un proceso, lo cual afecta directamente la eficiencia y la cantidad de fragmentación que se produce.

Explicación de los algoritmos

- **First Fit:** Recorre la memoria desde el principio y usa el primer espacio libre que sea lo suficientemente grande. Es el más rápido, pero con el tiempo tiende a dejar huecos en las primeras posiciones de la memoria.
- **Best Fit:** Busca el bloque libre que quede más justo al tamaño que se necesita. Ahorra espacio en cada asignación, pero puede crear muchos huecos pequeños dispersos (fragmentación externa).
- **Worst Fit:** Usa el bloque libre más grande disponible, dejando huecos más grandes que podrían aprovecharse después. Sin embargo, no siempre resulta eficiente y puede terminar desperdiciando espacio.

Representación del estado de la memoria

El programa muestra visualmente cómo queda la memoria después de cada acción, usando un formato como este:

```
[P1: 25][Libre: 10][P2: 8][Libre: 57]
```

Así se puede observar claramente cuáles espacios están ocupados por procesos y cuáles están disponibles, lo que ayuda a entender mejor el efecto de cada algoritmo en la fragmentación.

Fragmentación interna y externa

Durante las pruebas se vio que la memoria puede quedar fragmentada de dos maneras:

- **Fragmentación interna:** cuando un bloque asignado es más grande que lo que el proceso realmente necesita, por lo que parte del bloque queda sin usar.
- **Fragmentación externa:** cuando hay varios espacios libres, pero ninguno es lo suficientemente grande para el proceso que llega, aunque sumados sí alcanzan.

En general, **Best Fit** fue el que más fragmentación externa generó, mientras que **First Fit** y **Worst Fit** tuvieron un comportamiento más equilibrado, dependiendo del orden en que se hacían las operaciones.

Resultados y análisis

```
--- Test5 ---
Ingrese algoritmo de asignacion (First, Best, Worst):
Ingrese la capacidad de cada bloque:
Ingrese tamano total de memoria:
El proceso P2 ya existe.
[P1: 50][P2: 50][Libre: 100][Libre: 100][Libre: 100]
Fragmentacion externa total: 300
Fragmentacion interna total: 100
```

```
--- Test1 ---
Ingrese algoritmo de asignacion (First, Best, Worst):
Ingrese la capacidad de cada bloque:
Ingrese tamano total de memoria:
[P1: 30][P2: 40][Libre: 100][Libre: 100][Libre: 100]
Fragmentacion externa total: 300
Fragmentacion interna total: 130


--- Test2 ---
Ingrese algoritmo de asignacion (First, Best, Worst):
Ingrese la capacidad de cada bloque:
Ingrese tamano total de memoria:
[P1: 20][Libre: 100][P3: 10][Libre: 100][Libre: 100]
Fragmentacion externa total: 300
Fragmentacion interna total: 170

[P1: 20][P4: 25][P3: 10][Libre: 100][Libre: 100]
Fragmentacion externa total: 200
Fragmentacion interna total: 245


--- Test3 ---
Ingrese algoritmo de asignacion (First, Best, Worst):
Ingrese la capacidad de cada bloque:
Ingrese tamano total de memoria:
[P1: 70][P2: 40][Libre: 100][Libre: 100][Libre: 100]
Fragmentacion externa total: 300
Fragmentacion interna total: 90


--- Test4 ---
Ingrese algoritmo de asignacion (First, Best, Worst):
Ingrese la capacidad de cada bloque:
Ingrese tamano total de memoria:
[Libre: 100][B: 20][Libre: 100][Libre: 100][Libre: 100]
Fragmentacion externa total: 400
Fragmentacion interna total: 80
```

Al realizar varios casos de prueba, los tres algoritmos respondieron correctamente a las operaciones de asignar, liberar y mostrar memoria.

- **First Fit** resultó ser el más rápido, pero puede fragmentar la memoria rápidamente.
- **Best Fit** fue más eficiente al principio, pero terminó generando muchos huecos pequeños.
- **Worst Fit** se comportó de manera intermedia, aunque a veces desperdiciaba espacio.

En conclusión, **no hay un algoritmo perfecto**: cada uno funciona mejor en distintos contextos. Si hay muchas operaciones pequeñas, **First Fit** suele rendir mejor; en cambio, si la memoria se usa de forma más controlada, **Best Fit** puede ser más útil.

Conclusiones

Este parcial fue una excelente oportunidad para entender cómo funciona la administración de memoria en los sistemas operativos. Implementar los algoritmos permitió ver de forma práctica cómo el sistema toma decisiones y cómo afectan la fragmentación y el rendimiento general.

Para concluir esta experiencia permitió conectar la teoría con la práctica y comprender mejor por qué la administración de memoria es una de las tareas más importantes y complejas dentro de un sistema operativo.