Trabajo de clase

Descripción general

A continuación se definen los lineamientos de un actividad teórico-práctica para los estudiantes del curso sistemas operativos, en particular, los algoritmos de asignación de memoria.

Objetivos

Durante el desarrollo de de las actividades se logra:

- Identificar la memoria física.
- Aplicar los principios de la memoria virtual.
- Comprender la asignación de memoria que se realiza con los algoritmos First-fit, Best-fit y Worst-fit.
- Aplicar los conocimientos de programación en C++ para implementar estos algoritmos.

Antes de empezar

- Verifique la instalación de docker y de la imagen GCC o PUJGCC

En este trabajo de clase, se espera que se implementen tres algoritmos de asignación de memoria en C++: First-fit, Best-fit y Worst-fit. Estos algoritmos son fundamentales para la gestión de la memoria en sistemas operativos y sistemas embebidos.

INSTRUCCIONES

1. Crea una clase llamada "MemoryManagement" que simulará la gestión de la memoria. Esta clase debe contener al menos las siguientes funciones:

- mem(int size): Esta función inicializa la memoria con un tamaño especificado.
- memAlloc(int processID, int size): Esta función debe implementar el algoritmo de asignación de memoria First-fit. Debe asignar un bloque de memoria al proceso con el ID dado y del tamaño especificado.
- freeMem(int processID): Esta función libera el bloque de memoria asignado al proceso con el ID dado.

2. Implementa las versiones correspondientes de las funciones *memAlloc* y *freeMem* para los algoritmos Best-fit y Worst-fit. Cada algoritmo debe funcionar de acuerdo con sus respectivas reglas.

3. Crea un programa principal que utilice la clase "MemoryManagement" para simular la asignación y liberación de memoria por parte de varios procesos. Debes realizar pruebas con al menos cinco procesos diferentes que soliciten y liberen memoria en momentos distintos.

4. Imprime mensajes informativos en la consola para mostrar cómo se asigna y libera la memoria en cada paso. Puedes utilizar un enfoque simple de impresión para visualizar el estado de la memoria después de cada asignación y liberación.

5. Amplíe la versión anterior, cree la clase Reader que lee desde un TXT uno, dos o más procesos, su necesidad de espacio en memoria y presente si ha sido o no alojado el proceso. Tenga en cuenta la siguiente estructura del archivo de texto plano.

```
# Entrada N-1 # Este es un comentario
#
# Tamaño de la memoria
1024
# Cantidad de segmentos o particiones
4
# Tamanio de las particiones
256, 256, 256, 256
# Algoritmo: 1) First fit, 2) Best-Fit, 3) Worst-Fit
1
# proceso 18 solicita 180
18, 180
# proceso 23 solicita 222
23, 222
# proceso 92 solicita 222
92, 76
```

Archivo: Entrada1.txt

Requisitos Adicionales: Asegúrate de manejar adecuadamente los casos en los que no se pueda asignar memoria debido a la falta de espacio o cuando se intente liberar memoria que no esté asignada. Comenta el código de manera adecuada para explicar los detalles de implementación y los algoritmos utilizados. Cree al menos 2 entradas para cada algoritmo y pruebe que se realiza la asignación correspondiente a la política de asignación de memoria.

6. Aloje el código en un repositorio público de Github y envíe un informe de la implementación y las salidas a través del aula digital.

ANEXO: Entrada14.txt

```
# Entrada N-14
#
1960
5
200, 30, 700, 50, 980
2
J1, 20
J2, 200
J3, 500
J4, 50
```

Archivo: Entrada14.txt

ANEXO: CÓDIGO BASE EN C++

```cpp
#include <iostream>
#include <vector>

class GestorMemoria {
private:
    int tamanoMemoria;
    std::vector<int> memoria;

public:
    GestorMemoria(int tamano) : tamanoMemoria(tamano), memoria(tamano, -1) {
        // Inicializar la memoria con valores negativos (-1) para indicar que está vacía.
    }

    // Implementación del algoritmo First-fit para asignar memoria.
    bool asignarMemoriaFirstFit(int procesoID, int tamano) {

    }

    // Implementación del algoritmo Best-fit para asignar memoria.
    // (Completa esta función de acuerdo al algoritmo Best-fit)

    // Implementación del algoritmo Worst-fit para asignar memoria.
    // (Completa esta función de acuerdo al algoritmo Worst-fit)

    void liberarMemoria(int procesoID) {
        for (int i = 0; i < tamanoMemoria; i++) {
            if (memoria[i] == procesoID) {
                memoria[i] = -1; // Liberar el bloque de memoria
            }
        }
    }

    void imprimirEstadoMemoria() {
        std::cout << "Estado de la memoria: ";
        for (int i = 0; i < tamanoMemoria; i++) {
            std::cout << memoria[i] << " ";
        }
        std::cout << std::endl;
    }
};

int main() {
    int tamanoMemoria = 100; // Tamaño de la memoria
    GestorMemoria gestor(tamanoMemoria);

    // Prueba de asignación y liberación de memoria utilizando First-fit.
    if (gestor.asignarMemoriaFirstFit(1, 20)) {
        std::cout << "Asignado proceso 1 de tamaño 20" << std::endl;
    } else {
        std::cout << "No se pudo asignar memoria para proceso 1" << std::endl;
    }

    gestor.imprimirEstadoMemoria();

    gestor.liberarMemoria(1);
    std::cout << "Proceso 1 liberado" << std::endl;

    gestor.imprimirEstadoMemoria();

    return 0;
}
```

ANEXO: LECTOR SKIP # EN C++

```cpp
#include <iostream>
#include <fstream>
#include <string>

int main() {
    std::string nombreArchivoEntrada = "archivo_entrada.txt";
    std::string nombreArchivoSalida = "archivo_salida.txt";

    std::ifstream archivoEntrada(nombreArchivoEntrada);
    if (!archivoEntrada.is_open()) {
        std::cerr << "No se pudo abrir el archivo de entrada." << std::endl;
        return 1;
    }

    std::ofstream archivoSalida(nombreArchivoSalida);
    if (!archivoSalida.is_open()) {
        std::cerr << "No se pudo crear el archivo de salida." << std::endl;
        return 1;
    }

    std::string linea;
    while (std::getline(archivoEntrada, linea)) {
        // Comprueba si la línea comienza con "#" (comentario)
        if (linea.empty() || linea[0] == '#') {
            continue; // Omite las líneas que comienzan con "#"
        }

        archivoSalida << linea << std::endl;
    }

    archivoEntrada.close();
    archivoSalida.close();

    std::cout << "Proceso completado. Las líneas que comenzaban con '#' se han omitido." << std::endl;

    return 0;
}
```