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 mecanismos de segmentación y paginación que son utilizados para la administración de memoria.

Objetivos

Durante el desarrollo de de las actividades se logra:

- Identificar la memoria física.
- Aplicar los principios de la memoria virtual.
- Comprender los mecanismos de administración de memoria
- 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 los dos esquemas de administración de memoria en C++: Segmentación y Paginación. Estos algoritmos son fundamentales para la gestión de la memoria en sistemas operativos.

INSTRUCCIONES: SIMULADOR CON SEGMENTACIÓN

1. Diseña una estructura de datos que represente la memoria segmentada. Debe ser capaz de mantener un seguimiento de los segmentos de memoria y su estado (ocupado o libre).

2. Implementa funciones para asignar y liberar memoria a segmentos. La asignación debe verificar si hay suficiente espacio en un segmento (First-Fit) y, si es así, marcarlo como ocupado. La liberación debe marcar el segmento como libre.

3. Diseña una estructura o clase que represente un proceso. Cada proceso debe tener un identificador único y un tamaño.

4. Escribe el código principal para simular la asignación y liberación de memoria para cinco (5) procesos.

INSTRUCCIONES: SIMULADOR CON PAGINACIÓN

1. Diseña una estructura de datos que represente la memoria paginada. Debe ser capaz de mantener un seguimiento de las páginas de memoria y su estado (ocupadas o libres).

2. Implementa funciones para asignar y liberar memoria a páginas. La asignación debe verificar si hay suficiente espacio en una página y, si es así, marcarla como ocupada. La liberación debe marcar la página como libre.

3. Diseña una estructura o clase que represente un proceso. Cada proceso debe tener un identificador único y un tamaño.

4. Escribe el código principal para simular la asignación y liberación de memoria para cinco (5) procesos. Puedes crear los procesos y observar cómo se asigna y libera la memoria en función de sus solicitudes.

ENTREGA

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

ANEXOS

Versión incompleta del simulador de segmentación

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>

// Estructura para representar una página de memoria
struct Pagina {
 // Completar
};

// Estructura para representar un proceso
struct Proceso {
 // Completar
};

class SimuladorPaginación {
private:
    int tamañoMemoria;
    int tamañoPagina;
    memoriaPaginada;
    procesos;

public:
    SimuladorPaginación(int tamaño, int tamañoPag) : tamañoMemoria(tamaño), tamañoPagina(tamañoPag) {
        // Inicializar la memoria con páginas vacías
    }

    // Función para asignar espacio a un proceso
    bool asignarMemoria(int procesoID, int tamaño) {
        return false; // No se pudo asignar memoria
    }

    // Función para liberar espacio ocupado por un proceso
    void liberarMemoria(int procesoID) {

    }

```

```
38.     // Función para imprimir el estado de la memoria paginada
39.     void imprimirEstadoMemoria() {
40.         std::cout << "Estado de la memoria paginada:" << std::endl;
41.         for (int i = 0; i < memoriaPaginada.size(); i++) {
42.
43.         }
44.         std::cout << std::endl;
45.
46.         std::cout << "Procesos en memoria:" << std::endl;
47.         for (const Proceso &p : procesos) {
48.             std::cout << "Proceso ID: " << p.id << ", Tamaño: " << p.tamaño << ", Páginas: ";
49.             for (int página : p.paginasAsignadas) {
50.                 std::cout << página << " ";
51.             }
52.             std::cout << std::endl;
53.         }
54.     }
55. };
56.
57. int main() {
58.     int tamañoMemoria = 100; // Tamaño de la memoria paginada
59.     int tamañoPagina = 20;  // Tamaño de cada página
60.     SimuladorPaginación simulador(tamañoMemoria, tamañoPagina);
61.
62.     // Asignar espacio a procesos
63.     simulador.asignarMemoria(1, 30);
64.     simulador.asignarMemoria(2, 40);
65.     simulador.asignarMemoria(3, 15);
66.
67.     // Liberar espacio de procesos
68.     simulador.liberarMemoria(1);
69.
70.     // Imprimir estado de la memoria
71.     simulador.imprimirEstadoMemoria();
72.
73.     return 0;
74. }
```

Versión incompleta del simulador de paginación

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

// Estructura para representar una página de memoria
struct Pagina {
    int idProceso; // ID del proceso al que pertenece la página (-1 si está vacía)
};

// Estructura para representar un proceso
struct Proceso {
    int id;
    int tamaño;
    std::vector<int> paginasAsignadas; // Lista de páginas asignadas al proceso
};

class SimuladorPaginación {
private:
    int tamañoMemoria;
    int tamañoPagina;
    std::vector<Pagina> memoriaPaginada;
    std::vector<Proceso> procesos;

public:
    SimuladorPaginación(int tamaño, int tamañoPag) : tamañoMemoria(tamaño), tamañoPagina(tamañoPag) {
        // Inicializar la memoria con páginas vacías
        int numPaginas = tamañoMemoria / tamañoPagina;
        memoriaPaginada.resize(numPaginas, {-1});
    }

    // TODO: Implementar funciones para asignar y liberar memoria a páginas.

    // TODO: Implementar una función para imprimir el estado de la memoria paginada.

    // TODO: Agregar cualquier otra funcionalidad necesaria según las instrucciones.

};

int main() {
    int tamañoMemoria = 100; // Tamaño de la memoria paginada
    int tamañoPagina = 20;  // Tamaño de cada página
    SimuladorPaginación simulador(tamañoMemoria, tamañoPagina);

    // TODO: Realizar la simulación de asignación y liberación de memoria aquí.

    // TODO: Imprimir el estado de la memoria paginada.

    return 0;
}
```