Gerência de Memória

1- Na **alocação contígua** , um processo recebe um bloco único e contínuo de memória. Isso significa que todas as partes do processo são armazenadas em um único espaço sequencial .
Na **alocação não contígua** , um processo pode ser dividido em várias partes e armazenado em diferentes locais da memória.

2- **Fragmentação Externa**

🔹 O **que é?**
A fragmentação externa acontece quando a memória livre está espalhada em pequenos blocos não contíguos. Isso impede que processos maiores sejam alocados, mesmo que haja memória suficiente no total.

🔹 **Causa:**

- Uso de **alocação dinâmica** de memória.
- Os processos são carregados e removidos, deixando pequenos espaços entre blocos alocados.

🔹 **Impacto no Desempenho:**
❌ **Redução da capacidade de alocação** – Processos não podem ser carregados, pois não há um bloco grande o suficiente.
❌ **Maior tempo de processamento** – O sistema pode gastar mais tempo tentando compactar ou reorganizar a memória.

🔹 **Solução:**
✔ **Compactação de memória** – Mover processos para juntar os espaços livres.
✔ **Uso de técnicas como Paginação e Segmentação** – Divida a memória de forma mais flexível.

Fragmentação Interna

🔹 O **que é?**
A fragmentação interna ocorre quando a memória alocada a um processo é maior do que a necessidade real, desperdiçando espaço dentro do próprio bloco alocado.

🔹 **Causa:**

- Uso de **tamanhos fixos de blocos** (ex: um processo precisa de 18 KB, mas recebe um bloco de 20 KB).
- Alocação baseada em **múltiplos de um tamanho fixo** .

🔹 **Impacto no Desempenho:**
❌ **Desperdício de memória** – Pequenas quantidades de RAM ficam inutilizáveis.
❌ **Redução da eficiência** – O sistema pode parecer cheio mesmo com memória livre.

🔹 **Solução:**
✔ **Uso de blocos menores e flexíveis** .
✔ **Paginação** – Divida a memória em páginas menores para reduzir desperdício.

3- **Paginação**

🔹 **Definição:**
A memória é dividida em **blocos de tamanho fixo** , chamados de **páginas** (para os processos) e **quadros (frames)** (na memória física).

🔹 **Funcionamento:**

- Cada processo é dividido em páginas do mesmo tamanho dos quadros da memória física.
- Quando um processo é carregado, suas páginas podem ser armazenadas em qualquer quadro livre.
- O **MMU (Memory Management Unit)** usa uma **tabela de páginas** para mapear as páginas virtuais para os quadros financeiros.

🔹 **Vantagens:**
✅ **Evita fragmentação externa** – Qualquer quadro livre pode ser usado.
✅ **Gerenciamento simples** – Todas as páginas têm o mesmo tamanho.
✅ **Facilita a implementação de memória virtual** – Permite o uso de swap (paginação em disco).

🔹 **Desvantagens:**
❌ **Pode causar fragmentação interna** – Se o processo não usar completamente uma página, o espaço restante será desperdiçado.
❌ **Maior tempo de acesso** – A necessidade de consultar a tabela de páginas adiciona uma sobrecarga.

Segmentação

🔹 **Definição:**
A memória é dividida em **blocos de tamanho variável** , chamados **segmentos** , que se baseiam em partes lógicas de um programa (código, pilha, dados, etc.).

🔹 **Funcionamento:**

- Cada processo é dividido em segmentos de diferentes tamanhos.
- O sistema mantém uma **tabela de segmentos** , onde cada entrada armazena a base (endereço inicial) e o limite (tamanho) do segmento.
- O acesso à memória envolve a conversão de um número de segmentos + deslocamento para um endereço físico.

🔹 **Vantagens:**
✅ **Evita fragmentação interna** – Como os segmentos são do tamanho exato necessário.
✅ **Facilita a modularidade** – Cada segmento pode representar partes distintas do programa (pilha, código, variáveis globais).
✅ **Permite proteção e compartilhamento** – Diferentes segmentos podem ter permissões distintas e serem compartilhados entre processos.

🔹 **Desvantagens:**
❌ **Pode causar fragmentação externa** – Pequenos espaços livres podem se acumular entre segmentos.
❌ **Gerenciamento mais complexo** – Os segmentos têm tamanhos variados, exigindo mais controle.

4- Uma tabela de páginas é um componente crucial em um sistema de paginação virtual (também conhecido como paginação por demanda ou paginação de memória virtual), usada para mapear o espaço de endereçamento lógico (virtual) de um processo para o espaço de endereçamento físico (real) da memória RAM. Sua função principal é traduzir os endereços de memória virtual em endereços físicos.

5- **Segmentação** : Lida com unidades lógicas de memória (como código, dados, pilha), e os segmentos podem ter tamanhos variáveis. Facilita o compartilhamento e a organização mais próxima da estrutura do programa.

Paginação : Divida a memória em blocos fixos de tamanho igual, sem se preocupar com a estrutura lógica do programa. Pode ser mais eficiente em termos de alocação, mas menos flexível e menos intuitivo.

6-

```
from collections import deque



class Processo:

    def __init__(self, id_processo):

        self.id_processo = id_processo

        self.tabela_paginas = [None] * 4  # 4 páginas para cada
processo

        self.pagina_atual = 0  # Controla a próxima página a ser
alocada



    def alocar_pagina(self, pagina):

        if self.pagina_atual < 4:

            self.tabela_paginas[self.pagina_atual] = pagina

            self.pagina_atual += 1

        else:

            print(f"Erro: O processo {self.id_processo} já tem todas as
páginas alocadas.")
```

```
    def __str__(self):
        return f"Processo {self.id_processo} - Tabela de Páginas: {self.tabela_paginas}"

class MemoriaFisica:
    def __init__(self, num_quadros=16, tamanho_pagina=4):
        self.num_quadros = num_quadros
        self.tamanho_pagina = tamanho_pagina
        self.memoria = [None] * self.num_quadros
        self.fila_fifo = deque()  # Fila para armazenar os quadros de memória utilizados (FIFO)
        self.processos = {}

    def alocar_pagina(self, processo, pagina):
        # Verifica se há espaço na memória
        if None not in self.memoria:
            # Se não houver espaço, ocorre uma substituição
            self.substituir_pagina(processo, pagina)
        else:
            # Caso haja espaço livre, aloca a página no primeiro quadro vazio
            for i in range(self.num_quadros):
                if self.memoria[i] is None:
                    self.memoria[i] = (processo.id_processo, pagina)
                    self.fila_fifo.append(i)  # Adiciona o índice do quadro à fila FIFO
                    processo.alocar_pagina(pagina)
```

```
                    print(f"Página {pagina} do processo
{processo.id_processo} alocada no quadro {i}.")

                    break



    def substituir_pagina(self, processo, pagina):

        # Substitui a página mais antiga (FIFO)

        quadro_antigo = self.fila_fifo.popleft()  # Remove o quadro
mais antigo da fila FIFO

        processo_antigo, pagina_antiga = self.memoria[quadro_antigo]



        # Substitui a página no quadro

        self.memoria[quadro_antigo] = (processo.id_processo, pagina)

        self.fila_fifo.append(quadro_antigo)  # Adiciona o quadro
substituído de volta à fila FIFO



        # Aloca a nova página no processo

        processo.alocar_pagina(pagina)



        print(f"Substituindo a página {pagina_antiga} do processo
{processo_antigo} por {pagina} do processo {processo.id_processo} no
quadro {quadro_antigo}.")



    def exibir_estado_memoria(self):

        print("\nEstado Atual da Memória Física:")

        for i, quadro in enumerate(self.memoria):

            if quadro:
```

```
                print(f"Quadro {i}: Processo {quadro[0]} - Página
{quadro[1]}")
            else:
                print(f"Quadro {i}: Vazio")

    def exibir_estado_tabelas(self):
        for processo in self.processos.values():
            print(processo)

def simular():
    memoria = MemoriaFisica()

    # Criando processos
    processo1 = Processo(id_processo=1)
    processo2 = Processo(id_processo=2)
    processo3 = Processo(id_processo=3)

    # Atribuindo processos à memória
    memoria.alocar_pagina(processo1, 1)
    memoria.alocar_pagina(processo1, 2)
    memoria.alocar_pagina(processo2, 1)
    memoria.alocar_pagina(processo2, 2)

    # Exibindo estado após as primeiras alocações
    memoria.exibir_estado_memoria()
```

```
    memoria.exibir_estado_tabelas()



    # Tentando alocar mais páginas

    memoria.alocar_pagina(processo1, 3)  # Isso causará uma
substituição

    memoria.alocar_pagina(processo2, 3)  # Isso causará uma
substituição

    memoria.alocar_pagina(processo3, 1)  # Isso causará uma
substituição



    # Exibindo estado final

    memoria.exibir_estado_memoria()

    memoria.exibir_estado_tabelas()



if __name__ == "__main__":

    simular()
```