**CENTRO UNIVERSITÁRIO UNIDOMBOSCO**

**ANÁLISE E DESENVOLVIMENTO DE SISTEMAS**

**TRABALHO DE SISTEMAS OPERACIONAIS**

GUILHERME PENSO, MURILO LUSTOSA, EMANOEL ANDRE, MATHEUS GUILHERME

**GERENCIAMENTO DE MEMÓRIA**

**CURITIBA**

**2023**

GUILHERME PENSO

GUILHERME PENSO, MURILO LUSTOSA, EMANOEL ANDRE, MATHEUS GUILHERME

Trabalho do 2º Semestre do Curso Tecnólogo Análise e Desenvolvimento de Sistemas do Centro Universitário UniDomBosco, envolvendo a matéria de Sistemas Operacionais com o objetivo de criar um gerenciador de memória funcional.

Orientador: Edson Pedro Ferlin

CURITIBA

2023

SUMÁRIO

1. **INTRODUÇÃO1**
2. **LÓGICA** **2**

**3. CÓDIGO** ............................................................................................................ **2**

3.1 ALOCAÇÃO**2**

3.2 REALOCAÇÃO ....................................................................................... **4**

**4. TELA** ................................................................................................................. **5**

**5. CONCLUSÃO8**

1. **INTRODUÇÃO**

O gerenciamento de memória é extremamente importante para qualquer sistema operacional hoje em dia, já que sem ele não teríamos como executar softwares como os próprios sistemas operacionais que o gerenciam.

1. LÓGICA

Método allocate():

Esse método é chamado quando o usuário deseja alocar memória para um processo.

Ele solicita ao usuário o tamanho do processo usando uma caixa de diálogo.

Em seguida, ele verifica se há blocos vazios suficientes na grade para alocar o processo.

Se houver espaço suficiente, ele atribui uma cor aleatória ao processo e atualiza a grade visual.

Método reallocate():

Esse método é chamado quando o usuário deseja realocar todos os processos.

Ele salva as informações sobre os blocos de memória ocupados em uma lista temporária.

Em seguida, ele limpa todas as células da grade e realoca os blocos de memória salvos.

1. CÓDIGO
   1. ALOCAÇÃO

def allocate(self):

n = tkinter.simpledialog.askinteger("Alocar", "Digite o Tamanho do Processo")

free\_blocks = []

best\_fit = None

total\_free = 0

***# Pesquisa dentro da grid***

for i in range(10):

for j in range(10):

***# Verificação dos blocos vazios, se verdadeiro aumenta os blocos livres totais e os blocos livres***

if self.grid[i][j]["background"] == "white":

total\_free += 1

free\_blocks.append((i, j))

else:

if len(free\_blocks) >= n:

if best\_fit is None or len(best\_fit) > len(free\_blocks):

best\_fit = list(free\_blocks)

free\_blocks = []

***# Quando terminar de verificar todas as grids, terá outro processo de verificação se o valor digitado é maior que os blocos disponíveis***

if len(free\_blocks) >= n:

if best\_fit is None or len(best\_fit) > len(free\_blocks):

best\_fit = list(free\_blocks)

***# Se o valor digitado for maior que o total livre, uma mensagem de erro aparecerá***

if total\_free < n:

messagebox.showinfo("Erro", "Sem Espaço Total")

return

***# Se o a lista de blocos disponíveis nos free\_blocks for menor, uma mensagem aparecerá que não tem espaço sequencial***

if best\_fit is None:

messagebox.showinfo("Erro", "Sem Espaço Sequencial")

return

***# Gerador de cores e letrar para cada processo***

color = "#{:06x}".format(random.randint(0x0000, 0xFFFFFF))

id\_group = next(self.chars)

***# Separação de blocos de processos com os nomes e cores***

for k in range(n):

self.grid[best\_fit[k][0]][best\_fit[k][1]]["background"] = color

self.grid[best\_fit[k][0]][best\_fit[k][1]]["text"] = id\_group

self.groups[id\_group] = color

3.2 REALOCAÇÃO

def reallocate(self):

memory\_blocks = []

***# Procura dentro da grid dos processos***

for i in range(10):

for j in range(10):

***# Verificação de processos que estão ocupados***

if self.grid[i][j]['background'] != "white":

***# Salva os processos ocupadas no memory\_blocks***

memory\_blocks.append((i, j, self.grid[i][j]["text"], self.grid[i][j]["background"]))

***# Altera os dados atuais para não ocupados visualmente***

self.grid[i][j]['background'] = "white"

self.grid[i][j]['text'] = ""

***# Declaração de variáveis para coordenadas de realocação***

index = 0

x, y = 0, 0

***# Procura dentro da grid dos processos***

for i in range(10):

for j in range(10):

***# Verifica a quantidade restante de processos dos memory\_blocks***

if index < len(memory\_blocks):

if y == 10:

x += 1

y = 0

self.grid[x][y].grid(row=i, column=j)

***# Preenche novamente a partir das primeiras posições os blocos guardado na memory\_blocks***

self.grid[x][y]['background'] = memory\_blocks[index][3]

self.grid[x][y]['text'] = memory\_blocks[index][2]

# Aumenta o index para verificar o if

index += 1

***# Root atualiza a tela***

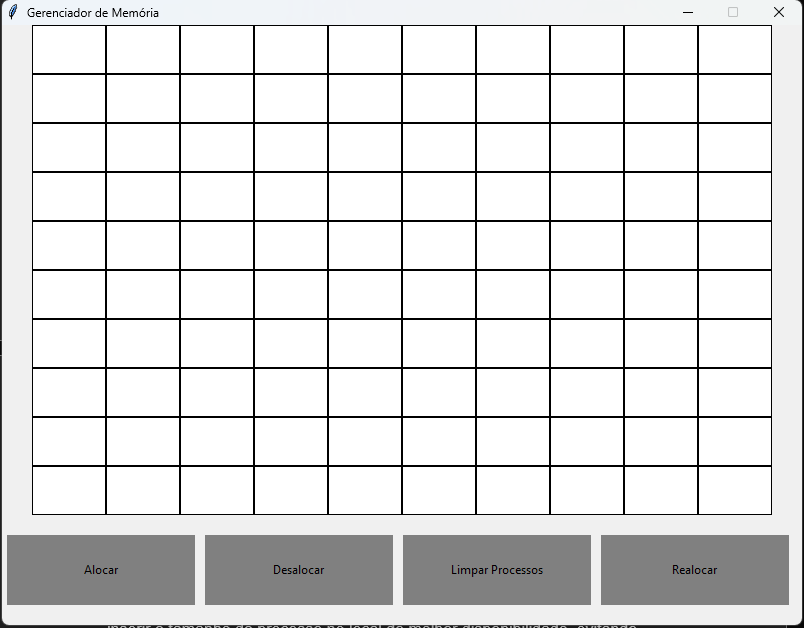
root.update()

***# Timer visual para realocação lenta***

time.sleep(0.05)

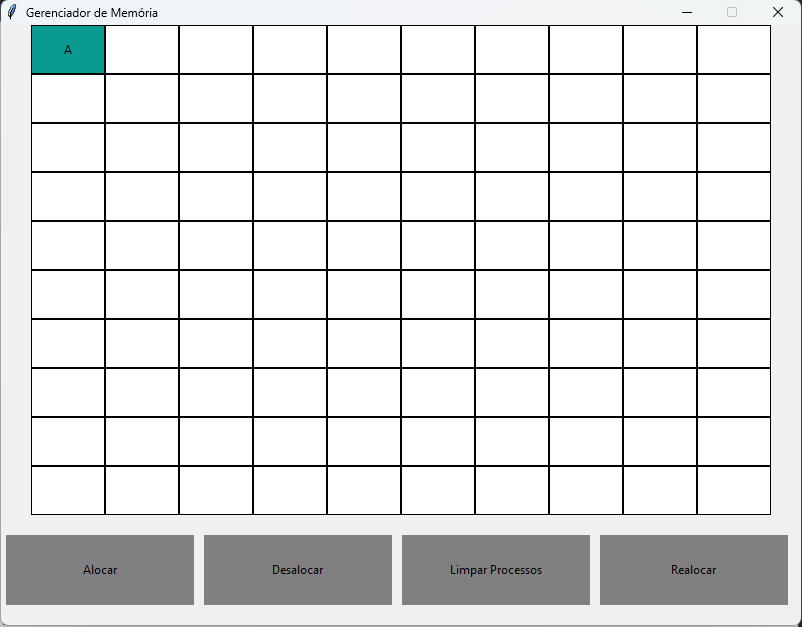
y += 1

1. TELA

 Uma imagem contendo Diagrama

Descrição gerada automaticamente

Tela Vazia Digitar Entrada do Número do bloco de Processos Alocados

 Tabela

Descrição gerada automaticamente

Valor Digitado Alocado Vários Processos Alocados

Tabela

Descrição gerada automaticamente Tabela

Descrição gerada automaticamente

Digitar Entrada Nome Processo Processos Desalocados

Para Desalocação

Calendário

Descrição gerada automaticamente com confiança média Tabela

Descrição gerada automaticamente

Tela Antes da Realocação Tela depois da Realocação

1. CONCLUSÃO

Podemos observar que o processo de alocação e realocação é de extrema importância para o procedimento de gerenciamento de memória, pois com essas funções podemos ajustar os espaços. Com a alocação temos a disponibilidade de inserir o tamanho do processo no local de melhor disponibilidade, evitando desperdício de memória. Já no caso da realocação, podemos reorganizar os processos com a finalidade de liberar mais espaço sequencial.