**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, 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. **INTRODUÇÃO 1**
2. **LÓGICA 2**

**2.1 Método Alocar2**

**2.2 Método Desalocar2**

**2.3 Método Realocar2**

**2.4 Método Limpar Processos2**

**2.5 Método Gerador Nome Processo3**

**2.6 Método Criar Grade3**

**2.7 Método Cor Bloco3**

1. **REQUISITOS 4**

**3.1 Linguagem Utilizada4**

**3.2 Bibliotecas Utilizadas4**

**3.2.1 tkinter4**

**3.2.2 tkinter.messagebox4**

**3.2.3 tkinter.simpledialog4**

**3.2.4 itertools4**

**3.2.5 random4**

**3.2.6 strings5**

**3.2.7 tkinter.ttk5**

**3.3 Hardware5**

**3.4 Armazenamento5**

**3.5 Memória Utilizada5**

**3.6 Compilador Utilizado6**

**3.6.1 Leitura do Código Fonte6**

**3.6.2 Análise do Código Fonte6**

**3.6.3 Geração do Código de Máquina6**

**3.6.4 Execução do Código de Máquina6**

1. **FUNCIONAMENTOS E TELAS 7**

**4.1 Tela Inicial7**

**4.2 Botão Alocar8**

**4.3 Botão Desalocar9**

**4.4 Botão Realocar10**

**4.5 Botão Limpar Processos11**

1. **FLUXOGRAMA 12**

**5.1 Alocar12**

**5.2 Desalocar12**

**5.3 Limpar Processos12**

**5.4 Realocar12**

**5.5 Ajudar13**

1. **CÓDIGO FONTE 14**
2. **MANUAL DO USUÁRIO22**

**7.1 Como Entrar no Aplicativo22**

**7.2 Alocar22**

**7.3 Desalocar22**

**7.4 Limpar Processos22**

**7.5 Realocar22**

**7.6 Ajuda23**

**7.7 Como Fechar o Aplicativo 23**

1. **CONCLUSÃO24**
2. **REFERÊNCIAS25**
3. **INTRODUÇÃO**

O gerenciamento de memória é uma função essencial em qualquer sistema de computação, responsável por alocar, desalocar e, se necessário, realocar espaços de memória. Este trabalho se concentrará no “Gerenciador de Memória”, uma parte integral do sistema operacional que realiza essas funções.

O Gerenciador de Memória tem a capacidade de alocar e desalocar espaços de memória para diferentes processos. Ele também pode realocar esses espaços conforme necessário, garantindo que cada processo tenha a quantidade adequada de memória para funcionar eficientemente.

Além disso, o Gerenciador de Memória tem a capacidade de limpar todos os processos de uma vez, liberando toda a memória e permitindo que o sistema comece do zero. Isso pode ser útil em situações onde o sistema está sobrecarregado ou quando um novo processo precisa de uma grande quantidade de memória.

Este trabalho explorará essas funções em detalhes, discutindo os vários algoritmos e técnicas usados para gerenciar a memória de maneira eficiente. Também discutiremos os desafios associados ao gerenciamento de memória e as soluções para esses desafios.

Com este estudo, nosso objetivo é aprofundar nosso entendimento sobre o papel crucial do Gerenciador de Memória em sistemas computacionais.

1. **LÓGICA**

**2.1 Método alocar**

Esse método é usado quando o usuário quer alocar memória para um processo. Ele pede ao usuário o tamanho do processo, verifica os blocos de memória livres e tenta encontrar o melhor ajuste para o processo. Se houver espaço suficiente, ele gera um nome e uma cor para o processo e atualiza os blocos de memória para mostrar que estão ocupados pelo novo processo. Se não houver espaço suficiente, ele informa ao usuário com uma mensagem de erro.

**2.2 Método realocar**

Esse método é acionado quando um processo precisa ser realocado. Ele encontra o primeiro processo que pode ser realocado, desaloca esse processo e, em seguida, procura blocos livres para realocar o processo. Este processo continua até que todo o processo tenha sido realocado ou não haja mais blocos livres suficientes. E por fim, o método então retorna a grade atualizada.

**2.3 Método desalocar**

O método é acionado quando o usuário deseja desalocar memória de um processo específico. Ele solicita ao usuário o nome do processo. Em seguida, percorre todos os 100 blocos de memória e, se encontrar um bloco ocupado pelo processo, atualiza o bloco para mostrar que está livre. Isso é feito alterando o texto e a cor de fundo do bloco para branco na interface do usuário e atualizando o status do bloco para 0 (livre).

**2.4 Método limpar processos**

O método é acionado quando o usuário deseja desalocar toda a memória. Ele percorre todos os 100 blocos de memória e, se encontrar um bloco ocupado, atualiza o bloco para mostrar que está livre. Isso é feito alterando o texto e a cor de fundo do bloco para branco na interface do usuário e atualizando o status do bloco para 0 (livre). No final, toda a memória estará desalocada.

**2.5 Método gerador nome processo**

O método é um gerador que produz nomes de processos únicos. Ele começa gerando todas as possíveis combinações de letras do alfabeto inglês de tamanho 1 (ou seja, ‘A’ até ‘Z’). Quando todas as combinações de tamanho 1 são esgotadas, ele passa para combinações de tamanho 2 (ou seja, ‘AA’ até ‘ZZ’), e assim por diante. Isso garante que cada nome de processo gerado seja único.

**2.6 Método criar grade**

O método é responsável por criar uma grade visual de blocos de memória no aplicativo. Ele cria um quadro e, em seguida, adiciona 100 rótulos a esse quadro. Cada rótulo representa um bloco de memória e é inicialmente configurado para ser branco e ter um certo tamanho.

**2.7 Método cor bloco**

Esse método gera uma cor aleatória em formato hexadecimal. Ela cria um número inteiro aleatório dentro do intervalo de cores possíveis em RGB, e então converte esse número para uma string hexadecimal de 6 dígitos. O resultado é uma string que representa uma cor RGB aleatória.

1. **REQUISITOS**

**3.1 Linguagem Utilizada**

A Linguagem utilizada no programa foi python 3.11.5 64bits.

O que é o python?

O Python é uma linguagem de programação para back-end. Desenvolvedores Web podem usar códigos Python para transferir dados de/para servidores, interagir com bancos de dados e administrar a segurança de um site. O Python foi criado com base em linguagem C, a facilidade de uso do Python é o fato de que estamos falando de uma linguagem de programação interpretada, e não compilada.

**3.2 Bibliotecas utilizadas**

**1.** **tkinter:** É uma interface padrão do Python para o kit de ferramentas de GUI Tk. Ele fornece um conjunto robusto e uma plataforma de ferramentas de janela. O pacote tkinter é uma camada orientada a objetos em cima de Tcl/Tk.

**2. tkinter.messagebox:** Este módulo do tk é utilizado para exibir caixas de mensagem na aplicação Python. As caixas de mensagem são uma forma de fornecer feedback ao usuário ou para uma decisão rápida do usuário.

**3.** **tkinter.simpledialog:** Este módulo possui classes e funções de conveniência para criar diálogos modais simples para obter um valor do usuário

**4. itertools:** Este módulo implementa uma série de blocos de construção de iteradores inspirados por construções APL, Haskell e SML. Cada um deles foi refeitos para adequar de melhor forma para o Python.

**5. random:** Este módulo implementa geradores de números pseudo-aleatórios para várias distribuições. Para inteiros, há seleção uniforme de um intervalo. Para sequências, há seleção uniforme de um elemento aleatório.

**6.** **string:** Este módulo implementa operações comuns de string. Ele fornece operações para manipular strings, incluindo funções para case conversion, padding, e busca de prefixos e sufixos.

**7.** **tkinter.ttk:** Este módulo fornece acesso ao conjunto de widgets temáticos Tk.

Para criar aplicativos modernos e com um visual nativo, o ttk é o caminho a seguir. Ele fornece uma série de widgets, incluindo botões, labels e text boxes, com uma aparência melhorada em relação ao tkinter padrão.

**3.3 Hardware**

O programa devido a baixo uso de memória facilmente pode ser utilizados em sistemas Windows 11/10/8, MacOs, Linux, Solaris, Unix e até FreeBSD. Então qualquer computador com 4GB de RAM por exemplo, é capaz de rodar o programa tranquilamente com um ótimo desempenho.

**3.4 Armazenamento**

O programa possui: 10.34 MB

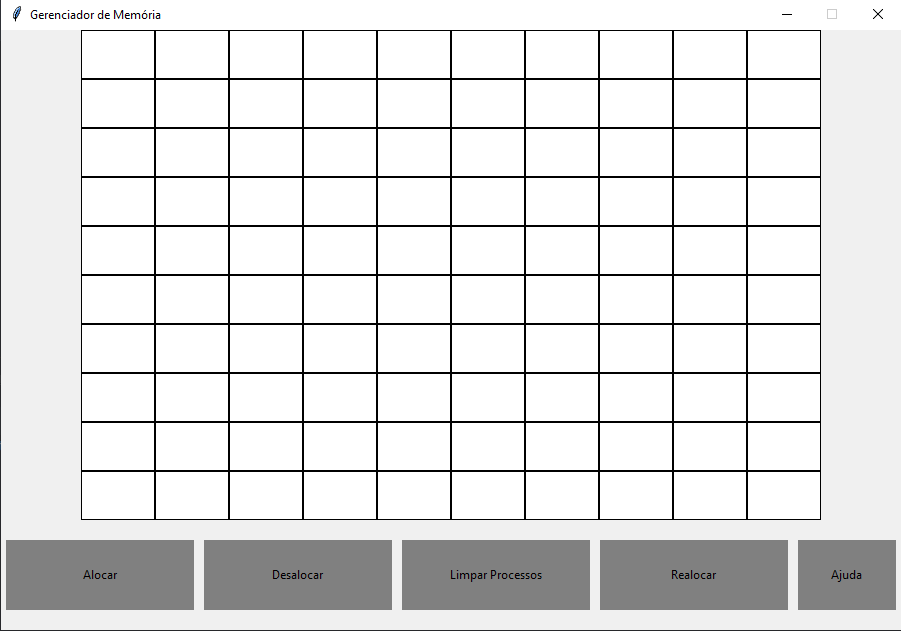
**3.5 Memória utilizada**

O programa utiliza no máximo 11.5MB de memória Física

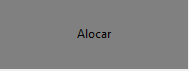
**3.6 Compilador utilizado**

O CPython é um interpretador de Python, o que significa que ele lê e executa o código fonte Python linha por linha. O processo de execução de um programa Python no CPython pode ser dividido em quatro etapas:  
  
**3.6.1 Leitura do código fonte**  
  
O interpretador CPython lê o código fonte Python linha por linha e armazena-o em uma estrutura de dados interna chamada de árvore sintática. A árvore sintática representa a estrutura do programa Python, incluindo os blocos de código, as declarações de variáveis, as atribuições, as chamadas de funções e outros elementos.  
  
**3.6.2 Análise do código fonte**  
O interpretador CPython analisa o código fonte para identificar os elementos básicos da linguagem Python. Os elementos básicos são palavras-chave, símbolos, identificadores e outros elementos que compõem o código fonte.  
  
**3.6.3 Geração de código de máquina**  
O interpretador CPython usa o código fonte analisado para gerar um código de máquina específico para a plataforma em que o programa está sendo executado. O código de máquina é um conjunto de instruções que o computador pode entender e executar.  
  
**3.6.4 Execução do código de máquina**  
  
O computador executa o código de máquina para realizar as tarefas especificadas no programa.

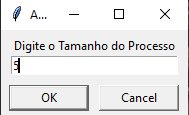
1. **FUNCIONAMENTO E TELAS**

**TELA INICIAL**

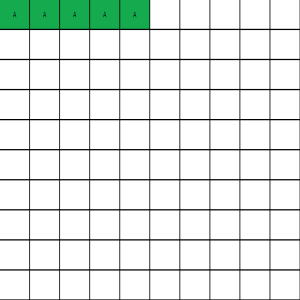
Esse Programa simula uma memória de um computador. Você poderá alocar processos em uma memória com até 100 blocos de memória os processos serão criados em ordem alfabética e cores aleatórias. Você poderá desalocar um processo específico da memória ou limpar todos os processos. Conforme o uso do programa haverá espaços vazios no meio de processos que não podem ser alocados com processos maiores, então o programa possui um meio de realocar os blocos de processos até o início da memória e liberar o mesmo espaço vazio para a parte final da memória

**4.1 Botão Alocar**

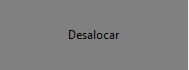
Antes de alocar um processo, a memória estará vazia. Após o clicar no botão Alocar, abrirá uma aba onde é possível adicionar o tamanho de um processo com números inteiros, após confirmar o tamanho, a Memória adicionará um processo com a quantidade de blocos nomeada de forma sequencial do alfabeto e cor aleatória. Exemplo abaixo:

****

Foi adicionado um processo com tamanho 5 que será inserido na memória a partir do momento que o usuário clica em OK.

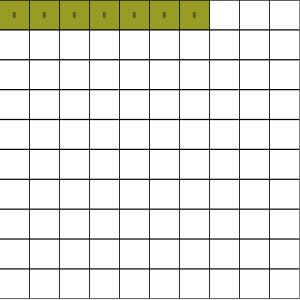
****

**4.2 Botão Desalocar**

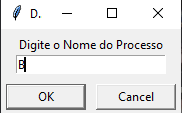


A Memória estando com um processo alocado, é possível desalocar com a letra correspondente ao processo inserido na memória, após confirmar o programa irá remover o processo em específico. Exemplo:

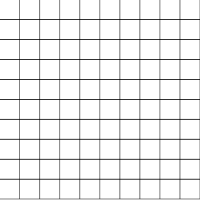
Memória com a Letra B alocada com um tamanho de 7 blocos.



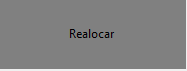
Após isso Digitando a letra B em desalocar:



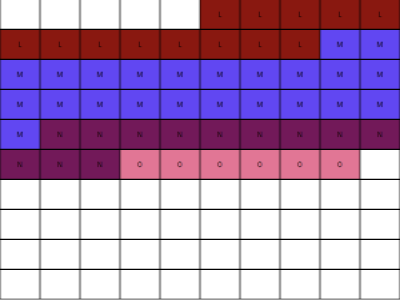
Ele limpará o processo B da Memória, liberando os blocos para inserir outros processos:



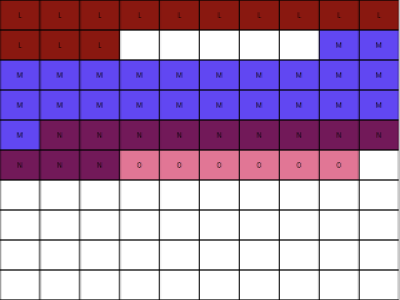
**4.3 Botão Realocar**

****

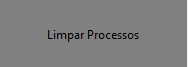
Com espaços vazios entre os processos que podem ser reutilizados para alocar outros processos, a cada clique será realocado o primeiro bloco de processo que tenha espaço vazio antes de seu início.Exemplo abaixo:

****

Após o clique do botão realocar o programa realocará o primeiro bloco encontrado para o início:



**4.4 Botão Limpar Processos**

****

A Memória estando com alguns processos, é possível apagar tudo ao clicar no botão Limpar Processos, deixando todos os blocos disponíveis, Exemplo:

Antes do clique de limpar processos:



Após o clique todos os blocos ficaram vazios.

1. **FLUXOGRAMA**
   1. **Alocar**

Diagrama

Descrição gerada automaticamente

* 1. **Desalocar**

Diagrama

Descrição gerada automaticamente

* 1. **Limpar Processos**

Diagrama

Descrição gerada automaticamente

* 1. **Realocar**

Diagrama

Descrição gerada automaticamente

* 1. **Ajuda**

Diagrama

Descrição gerada automaticamente

1. **CÓDIGO FONTE**

# Importações de bibliotecas

# Bibliotecas de Tkinter para a parte visual

from tkinter import \*

# Biblioteca para caixa de mensagem

from tkinter import messagebox

import tkinter as tk

# Bibioteca para janela com entrada de dados

import tkinter.simpledialog

from tkinter import ttk

# Biblioteca de iteração

import itertools

# Biblioteca para geração random de valores

import random

# Biblioteca com alfabeto

import string

# Gera nomes em ordem alfabética

def gerador\_nome\_processo():

    tamanho = 1

    while True:

        for s in itertools.product(string.ascii\_uppercase, repeat=tamanho):

            yield "".join(s)

        tamanho += 1

# Objeto do Gerenciador de Memório

class GerenciadorMemoria:

    # Atributos do objeto

    def \_\_init\_\_(self, mestre):

        self.mestre = mestre

        self.grade = [None] \* 100

        self.grupos = {}

        self.chars = gerador\_nome\_processo()

        self.status = [0] \* 100

        self.criar\_grade()

    # Métodos do objeto

    # Cria grade de memória de 100 blocos

    def criar\_grade(self):

        quadro = tk.Frame(self.mestre)

        quadro.pack(expand=True)

        for i in range(100):

            rotulo = tk.Label(quadro, bg="white", width=10, height=3, relief="solid", borderwidth=1.2)

            rotulo.grid(row=i//10, column=i%10)

            self.grade[i] = rotulo

    # Aloca processo na memória

    def alocar(self):

        # Solicita ao usuário o tamanho do processo

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

        blocos\_livres = []

        best\_fit = None

        total\_livre = 0

        # Procura um espaço livre de tamanho exato ou maior com a menor quantidade sobrando

        for i in range(100):

            # Verifica todos os espaços livres e adiciona a uma lista e acrescenta no contador a quantidade livre

            if self.status[i] == 0:

                total\_livre += 1

                blocos\_livres.append(i)

            # Ao achar um bloco ocupado, verifica se o tamanho livre adicionado anteriormente é maior ou igual a entrada de tamanho do usuário e salva as posições na variável best\_fit

            else:

                if len(blocos\_livres) >= n:

                    if best\_fit is None or len(best\_fit) > len(blocos\_livres):

                        best\_fit = list(blocos\_livres)

                # Após alocar o processo no menor espaço disponível, limpa a lista dos blocos\_livres que está desatualizada para o próximo processo solicitado pelo usuário

                blocos\_livres = []

        # Atualiza o best\_fit com a lista de blocos\_livres atualizada para próxima solicitação de alocação do usuário

        if len(blocos\_livres) >= n:

            if best\_fit is None or len(best\_fit) > len(blocos\_livres):

                best\_fit = list(blocos\_livres)

        # Se não houver espaço livre total suficiente, uma mensagem de erro é exibida

        if total\_livre < n:

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

            return

        # Se não houver espaço livre sequencial suficiente, uma mensagem de erro é exibida

        if best\_fit is None:

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

            return

        # Chama a função para gerar o nome do processo e gera uma cor aleatória

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

        nome\_bloco = next(self.chars)

        # Aloca na memória usando as posições da variável best\_fit, altera seu status para "1" com o nome em ordem alfabética e cor aleatória

        for k in range(n):

            self.status[best\_fit[k]] = 1

            self.grade[best\_fit[k]]["text"] = nome\_bloco

            self.grade[best\_fit[k]]["background"] = cor\_bloco

    # Desalocar um processo

    def desalocar(self):

        # Solicita ao usuário o nome do processo para desalocar

        d = tkinter.simpledialog.askstring("Desalocar", "Digite o Nome do Processo")

        # Busca o processo pelo nome do processo declarado pelo usuário e altera para "0"

        for i in range(100):

            if self.grade[i]["text"] == d.upper():

                self.status[i] = 0

                # Altera parte visual do Tkinter, removendo o nome do processo e mudando sua cor para branco

                self.grade[i]["text"] = ""

                self.grade[i]['background'] = "white"

    # Busca nas 100 posições todos os processos que não estão livres e muda de "1" para "0" o status e altera para sem texto e cor branca visualmente

    def limpar\_processos(self):

        for i in range(100):

            if self.status[i] != 0:

                self.status[i] = 0

                # Altera parte visual do Tkinter

                self.grade[i]["text"] = ""

                self.grade[i]["background"] = "white"

    # Realoca um processo por vez a cada clique

    def realocar(self):

        tamanho\_processo = 0

        texto\_memoria = None

        cor\_memória = None

        # Começa na posição 1 para evitar o primeiro processo e achar os espaços vazio entre os processos

        for pos in range(1, 100):

            # Verifica se a posição anterior está vazia e se a atual está ocupada, localizando o começo de um processo

            if self.status[pos - 1] == 0 and self.status[pos] == 1:

                # Salva o texto e memória do processo atual para atualizar visualmente no Tkinter

                texto\_memoria = self.grade[pos]["text"]

                cor\_memória = self.grade[pos]["background"]

                # Enquanto o nome do processo for o mesmo, conta seu tamanho e a posição em que está dentro da lista

                while pos < 100 and self.grade[pos]["text"] == texto\_memoria:

                    tamanho\_processo += 1

                    pos += 1

                break

        # Altera os dados da posição atual do processo

        for pos in range(100):

            # Enquanto o nome do processo for o mesmo, altera seu status para "0"

            if texto\_memoria == self.grade[pos]["text"]:

                self.status[pos] = 0

                # Altera parte visual do Tkinter, removendo o nome do processo e mudando sua cor para branco

                self.grade[pos]["text"] = ""

                self.grade[pos]["background"] = "white"

        # Faz a busca na lista até achar o primeiro bloco livre e verifica se a memória é maior que zero, alterando seu status para "1"

        for pos in range(100):

            if self.status[pos] == 0 and tamanho\_processo > 0:

                self.status[pos] = 1

                # Altera parte visual do Tkinter, removendo o nome do processo e mudando sua cor para branco e diminui um contador referente ao tamanho do processo

                self.grade[pos]["text"] = texto\_memoria

                self.grade[pos]["background"] = cor\_memória

                tamanho\_processo -= 1

        # Atualiza a cada realocação de processo visualmente no Tkinter

        return self.grade

    # Botão de Ajuda para o usuário

    def ajuda(self):

        # Define que a janela aparecerá por cima do programa

        janela\_ajuda = tk.Toplevel()

        #Título da janela

        janela\_ajuda.title("Ajuda")

        # Definição do tamanho e posição da janela centralizado conforme a tela do usuário ao fazer SysCall para o Windows

        largura\_ajuda = 600

        altura\_ajuda = 300

        largura\_tela = raiz.winfo\_screenwidth()

        altura\_tela = raiz.winfo\_screenheight()

        x = (largura\_tela/2) - (largura\_ajuda/2)

        y = (altura\_tela/2) - (altura\_ajuda/2)

        janela\_ajuda.geometry('%dx%d+%d+%d' % (largura\_ajuda, altura\_ajuda, x, y))

        # Título dentro da janela de ajuda

        Info = Label(janela\_ajuda, text="Explicação do funcionamento de cada botão")

        Info.pack(padx=10, pady=3)

        nb = ttk.Notebook(janela\_ajuda, width=600, height=400)

        nb.pack(padx=10, pady=10)

        # Aba sobre o Funcionamento do Programa

        funcionamento = Frame(nb)

        nb.add(funcionamento, text="Funcionamento do Programa")

        label\_funcionamento = Label(funcionamento, text="Esse Programa simula uma memória de um computador.\nVocê poderá alocar processos em uma memória com até 100 blocos de memória\n os processos serão criadas em ordem alfabética e cores aleatórias.\nVocê poderá desalocar um processo especifico da memória ou limpar todos os processos.\nConforme o uso do programa haverá espaços vazios no meio de processos que não podem ser alocados\n com processos maiores, então o programa possui um meio de realocar os blocos de processos até o inicio\n da memória e liberar o mesmo espaço vazio para a parte final da memória.")

        label\_funcionamento.pack(pady=20, anchor=CENTER, fill="both")

        # Rodapé com o nome dos criadores do programa

        label\_criadores = Label(funcionamento, text="Desenvolvido por: Guilherme Penso, Matheus Guilherme, Murilo Lustosa\n e Emanoel Andre.")

        label\_criadores.pack(padx=10, pady=0)

        label\_criadores.config(font="Arial 12")

        # Aba sobre o funcionamento da Alocação

        Alocar = Frame(nb)

        label\_info = Label(Alocar, text="Funcionamento do Botão Alocar.")

        label\_info.pack(padx=10, pady=10)

        mensagem\_alocar = Label(Alocar, text="Antes de alocar um processo, a memória estará vazia. Após o clicar no botão Alocar, abrirá uma aba\n onde é possível adicionar o tamanho de um processo com números inteiros, após confirmar\n o tamanho, a Memória adicionará um processo com a quantidade de blocos nomeada de forma sequencial\n do alfabeto e cor aleatória.")

        mensagem\_alocar.pack()

        nb.add(Alocar, text="Alocar")

        # Aba sobre o funcionamento da Desalocação

        Desalocar = Frame(nb)

        label\_info2 = Label(Desalocar, text="Funcionamento do Botão Desalocar.")

        label\_info2.pack(padx=10, pady=10)

        mensagem\_desalocar = Label(Desalocar, text="A Memória estando com pelo menos um processo alocado, é possível desalocar com a letra correspondente\n ao processo, que irá remover o processo em específico.")

        mensagem\_desalocar.pack()

        nb.add(Desalocar, text="Desalocar")

        # Aba sobre o funcionamento do Desalocar Tudo

        Desalocar\_tudo = Frame(nb)

        label\_info3 = Label(Desalocar\_tudo, text="Funcionamento do Botão Limpar Processos.")

        label\_info3.pack(padx=10, pady=10)

        mensagem\_desalocar\_tudo = Label(Desalocar\_tudo, text="A Memória estando com um ou mais processos, é possível apagar tudo ao clicar no botão Limpar Processos.")

        mensagem\_desalocar\_tudo.pack()

        nb.add(Desalocar\_tudo, text="Limpar Processos")

        # Aaba sobre o funcionamento de Realocação

        Realocar = Frame(nb)

        label\_info4 = Label(Realocar, text="Funcionamento do Botão Realocar.")

        label\_info4.pack(padx=10, pady=10)

        mensageRealocar = Label(Realocar, text="Com espaços vazios entre os processos que podem ser reutilizados para alocar outros processos, a \ncada clique será realocado o primeiro bloco de processo que tenha espaço vazio antes de seu início. ")

        mensageRealocar.pack()

        nb.add(Realocar, text="Realocar")

# Cria a tela principal chamada raiz e aciona o Tkinter

raiz = tk.Tk()

# Definição do tamanho e posição da janela centralizado conforme a tela do usuário ao fazer SysCall para o Windows

largura\_menu = 800

altura\_menu = 600

raiz.resizable(0, 0)

raiz.title('Gerenciador de Memória')

largura\_tela = raiz.winfo\_screenwidth()

altura\_tela = raiz.winfo\_screenheight()

x = (largura\_tela/2) - (largura\_menu/2)

y = (altura\_tela/2) - (altura\_menu/2)

raiz.geometry('%dx%d+%d+%d' % (largura\_menu, altura\_menu, x, y))

# gm irá chamar os métodos do objeto GerenciadorMemória

gm = GerenciadorMemoria(raiz)

# Cria um quadro de botões abaixo da grade de posições

quadro\_botao = tk.Frame(raiz)

quadro\_botao.pack(side="top", fill="x", pady=20)

# Botão que aciona a função de Alocar

botao\_alocar = tk.Button(quadro\_botao, text="Alocar", command=gm.alocar, height=5, width=21, bg='gray', borderwidth=0)

botao\_alocar.pack(side="left", padx=5)

# Botão que aciona a função de Desalocar

botao\_desalocar = tk.Button(quadro\_botao, text="Desalocar", command=gm.desalocar, height=5, width=21, bg='gray', borderwidth=0)

botao\_desalocar.pack(side="left", padx=5)

# Botão que aciona a função de Limpar Processos

botao\_limpar\_processos = tk.Button(quadro\_botao, text="Limpar Processos", command=gm.limpar\_processos, height= 5, width=21, bg='gray', borderwidth=0)

botao\_limpar\_processos.pack(side='left', padx=5)

# Botão que aciona a função de Realocar

botao\_realocar = tk.Button(quadro\_botao, text="Realocar", command=gm.realocar, height=5, width=21, bg='gray', borderwidth=0)

botao\_realocar.pack(side="left", padx=5)

# Botão que aciona a função de Ajuda

botao\_ajuda = tk.Button(quadro\_botao,text="Ajuda",command=gm.ajuda, height=5, width=21,bg='gray',borderwidth=0)

botao\_ajuda.pack(side="left", padx=5)

raiz.mainloop()

1. **MANUAL DO USUÁRIO**
   1. **Como entrar no Aplicativo**

Basicamente para você entrar no aplicativo após a instalação você terá que clicar duas vezes em cima do gerenciador de memória.exe assim o programa abrirá normalmente

**7.2 Alocar**

Antes de alocar um processo, a memória estará vazia. Após o clicar no botão Alocar, abrirá uma aba onde é possível adicionar o tamanho de um processo com números inteiros, após confirmar o tamanho, a Memória adicionará um processo com a quantidade de blocos nomeada de forma sequencial do alfabeto e cor aleatória.

**7.3 Desalocar**

A Memória estando com um processo, é possível desalocar com a letra correspondente\n ao processo, que irá remover o processo em específico.

**7.4 Limpar Processos**

A Memória estando com alguns processos, é possível apagar tudo ao clicar.

**7.5 Realocar**

Com espaços vazios entre os processos que podem ser reutilizados para alocar outros processos, a cada clique será realocado o primeiro bloco de processo que tenha espaço vazio antes de seu início.

**7.6 Ajuda**

Ao apertar esse botão você terá uma janela aberta com todos as informações do que o programa faz e, o que cada botão faz, para facilitar o uso do usuário

**7.7 Como fechar o Aplicativo**

Para fechar o app basta apenas clicar no X que fica em cima da janela a direita

1. **CONCLUSÃO**

Neste trabalho, desenvolvemos um aplicativo que simula o gerenciamento de memória em um sistema operacional. O aplicativo permite que o usuário aloque, desaloque e realoque memória para processos. Além disso, o aplicativo fornece uma visualização gráfica da memória disponível, o que ajuda o usuário a entender como o gerenciamento de memória funciona. Com base nos resultados deste estudo, podemos concluir que o gerenciamento de memória é uma tarefa difícil. Os algoritmos e técnicas usados para gerenciar a memória devem ser capazes de atender às necessidades dos processos em execução, ao mesmo tempo em que minimizam o desperdício de memória. O aplicativo desenvolvido neste trabalho fornece uma ferramenta útil e simples para entender os conceitos básicos do gerenciamento de memória.  
  
Em geral, é importante utilizar os recursos de memória disponíveis de forma eficiente ao desenvolver softwares. O gerenciamento de memória é uma tarefa complexa que requer um equilíbrio entre o funcionamento e a flexibilidade do software. É importante garantir que o software não seja muito grande, o que pode dificultar sua execução e alocação de memória.

1. **REFERÊNCIAS**

GEEKS FOR GEEKS. Memory Management in Python. Disponível em: <https://www.geeksforgeeks.org/memory-management-in-python/>.

PYTHON TUTORIAL. Tkinter Tutorial. Disponível em: <https://www.pythontutorial.net/tkinter/>.

MEDIUM. Effective Memory Management and Optimization in Python. Disponível em: <https://medium.com/@bpst.blog/effective-memory-management-and-optimization-in-python-d8a4d1992a45.

QA Stack BR. Python vs CPython. Disponível em: https://qastack.com.br/programming/17130975/python-vs-cpython.