Gerenciamento de memória em sistemas operacionais

Projeto para a disciplina Estruturas de Dados e Algoritmos

Alunos:

- Andrew Lucas Guedes de Souza - 16/0023921
- Nivaldo Pereira Lopo Junior - 12/0039460
- Max Henrique Barbosa - 16/0047013

Relatório de Projeto

Índice

1. Introdução

Um gerenciador de memória de sistemas operacionais, é utilizado para gerir a memória, manipulando espaços na memória, alguns com processos, outros livres. Os processos serão executados por um período de tempo gerado pseudo-randomicamente pelo sistema e a inserção de um novo processo na memória será feita utilizando o método *First-Fit*, que utiliza a primeira lacuna que encontrar com tamanho suficiente para alocar o processo, esse método é o que possui melhor performance em comparação com os outros métodos.

2. Objetivo do projeto

O projeto visa o desenvolvimento de um simulador de gerenciamento de memória em sistemas operacionais, utilizando a linguagem C.

3. Requisitos do projeto

- [x] Os processos devem ter como parâmetros o seu tamanho (em Kbytes) e o tempo de execução (em segundos).

- [x] Os processos devem ser alocados na memória de acordo com o seu tamanho e devem ter um identificador (label).
- [x] O processo deve ser colocado na primeira região de memória (a contar do início para o final da memória) onde esse processo couber.
- [x] O programa deve tentar encontrar uma forma de reorganizar os processos na memória de modo a acomodar o novo processo. Se depois dessa tentativa não houver espaço disponível, deve-se imprimir a seguinte mensagem: "Não foi possível alocar memória para o processo: [label do processo]".
- [x] O tempo de execução de um processo é aleatório (não previsível). Portanto, caberá ao aluno definir como ele vai simular esse tempo de execução e o término de um processo. Sugere-se ainda que o programa imprima o registro de entrada e saída de processos na memória.
- [x] O programa deve ser capaz de imprimir, a qualquer momento, quais posições de memória estão ocupadas (P) e quais estão livres (H). Imagine que a memória é um grande vetor onde cada posição armazena um Kbyte.
- [x] Além das opções para inclusão de processos na memória e impressão, o programa deve ter a opção de sair (deixar o programa).
- [x] O programa deve oferecer a opção de gravar os dados em arquivo. E quando o programa for iniciado, deve ser possível ler as informações desse arquivo para continuar a execução da simulação da memória.
- [x] A gerência de espaços de processo (P) e buracos (H) na memória deve ser feito por meio de uma lista circular duplamente encadeada, com cabeçalho. Nesse caso, o aluno deve desenhar o nó considerando pelo menos: tipo de nó (P ou H), posição de memória inicial, tamanho da área de memória livre ou ocupada.

4. Metodologia

4.1 Estruturas de dados

Para o desenvolvimento do simulador, foi utilizada uma lista circular duplamente duplamente encadeada com cabeçalho, para gerenciar a memória.
O cabeçalho é composto por dois apontadores, um apontando para o primeiro elemento no primeiro endereço da memoria, seja um processo ou um buraco. E outro apontando para o útimo elemento da lista, E por fim um inteiro para registrar a soma de todos os espaçoes livres.
Na lista, cada elemento possui:

Nome	Nome na estrutura	Tipo
tipo	type	P - processo ou H - buraco
id	id	int
rótulo	label	char
tamanho	size	int
Endereço	startAt	int
Tempo de início	initialTime	int
Duração	duration	int
Apontador próximo	next	struct MemoryList
Apontador anterior	prev	struct MemoryList

O tipo de estrutura de dados mais recomendado para gerenciamento de memória é uma lista

duplamente encadeada, porém para facilitar o processo de pecorrer a lista, tornamos ela circular.

4.2 Bibliotecas

Biblioteca	Descrição	Utilidade
stdio	Possui definições de subrotinas de entrada/saída de dados.	Utilizada para ler e mostrar informções.
stdlib	biblioteca de propósito geral padrão	Alocação de memória, gerar um tempo aleatório para cada processo e limpar a tela
time	Bibiloteaca dedicada manipulação de datas e horários	Controle de tempo dos processos
locale	Define configurações específicas da localização	Mostrar caracteres especiais.
unistd	Permite acesso a ficheiros e diretórios	Função sleep;
pthread	Biblioteca para manipulação de threads	Utilização de thread para mostrar display e atualização
string	Manipulação de string	Manipular tempo em formato de string

4.3 Funções

void initialize(Memory* memory)

Inicializa a lista encadeada, criando o primeiro elemento, um espaço vazio do tamanho da memória definida. Registra a inicialização do sistema em no log. Caso exista o arquivo de swap, pergunta ao usuário se ele deseja recuperar os processos, caso sim, ele realiza a inicialização dos processos a partir do arquivo.

void shut(Memory * memory)

Encerra o programa. Confirma se o usuário quer realmente sair, caso não retorna ao programa, caso sim, se não houver processos em execução ele registra o encerramento no log, e termina. Caso haja processo em execução ele pergunta se deseja guardar o processo em disco. Se sim ele realiza o swap, se não ele registra e sai do sistema.

int swap(Memory * memory)

Grava processos em execução em disco, chama o garbageCollector para verificar se há processos encerrados na memória, grava em arquivo o id, a label, o tamanho, e a duração com o desconto do tempo já executado de cada processo ativo na memória.

void readSwap(Memory * memory, FILE * fp)

Ler dados processos gravados em disco, e os inicializa na memória.
Obs: o arquivo é aberto na função shut, para verificar sua existência.

void initializeProcess(Memory * memory, int id, char label, int size, int duration, int initTime, int mode)

Inicializa um processo, alocando-o na memória, através do método *First-Fit*. Recebe como parâmetros os dados do processo, e o mode para verificar qual função está chamando a inicialização.

Verifica se há processos encerrados na memória, aloca espaço para o novo processo, define as suas informações. Verifica se há espaço na memória maior ou igual a do processo, se sim verifica se é continuo ou não, se não, chama o procedimento compactMemory, caso não seja necessário compactar a memória, a lista é pecorrida e até encontrar o espaço o qual possa alocar o processo. Ao final da alocação, registra em log o processo criado.
Se não há espaço para o processo, uma mensagem é exibida informando o usuário.

void newProcess(Memory * memory)

Recebe os parâmetros do usuário para inicializar o processo. Gera a duração do processo apartir de um tempo aleatório entre 10s e 180s, e o id apartir de uma variável global(idGeneretor), que é incrementado em um sempre que um processo é executado. Chama a função de inicialização de processo, passando além dos dados do processo, o mode 0

int spaceVerify(Memory * memory, int processSize)

Verifica se o espaço livre na memória é mairo que o tamnho do processo, retornando 1 caso seja verdade e 0 caso não.

void * showMemory()

Executada em uma thread, a função mostra dados da memória na tela. Exibe quais processos em execução, suas informações como ID, label, tamanho, endereço e tempo de execução em segundos. Caso não houver processos em execução, exibe que não há processos em execução.
Mostra a porcentagem de uso da memória, e como os processos estam alocados nela. além de dados como memória livre, ocupada e total. A tela é atualizada a cada segundo, enquanto o usuario não apertar Enter para voltar ao menu.

void * closeThread(void)

Executada em uma thread, espera pela entrada do usuário até que ele aperte a tecla enter.

void callShowMemory(Memory * memory)

Cria a thread em que a função showMemory irá ser executada.

MemorySpace * getProcess(Memory * memory, int id)

Percorre a lista em busca de um processo com o id igual ao do parâmetro, se encontrar, retorna o processo, se não retorna nulo.

void callShutProcess(Memory * memory)

Recebe o id do processo que o usuário deseja forçar o encerramento, e chama a função shutProcess.

void shutProcess(Memory * memory, int id)

Força encerramento do processo. Através do id passado como parâmetro, muda o tipo do processo para buraco, e registra o encerramento no log. Se algum dos espaços vizinhos também for um buraco chama a função que mescla dois buracos em um só.

void freeSpaceCounter(Memory * memory)

Soma todos os tamanhos dos buracos da memória e atribui ao atributo free_space, no Cabecalho da lista.

int findSpace(Memory * memory, int size)

Pecorre a lista, verificando se há algum espaço livre continuo maior ou igual ao tamanho do processo, se sim retorna o endereço do espaço, se não retorna -1.

void compactMemory(Memory * memory, MemorySpace * process)

Realiza a compactação da memória, quando houver espaço para alocar o novo processo, porém não continuo.
A compactação é feita copiando todos processos em execução para um arquivo temporário e depois repassando os processos para a lista seguido da exclusão do arquivo.

void mergeHole(Memory * memory, MemorySpace * p)

Recebe o id do processo que o usuário deseja forçar o encerramento, e chama a função shutProcess.

void garbageCollector(Memory * memory)

Através de um loop do-while analisa todos os processos e verifica se algum deles ja foi finalizado, comparando seu tempo inicial e sua duração. Ao final utiliza a função freeSpaceCounter para atualizar a memoria disponível.

void logRegister(MemorySpace * p, int mode)

Salva em um arquivo texto o log de atividades do gerenciador de memoria, informando o inicio e encerramento do sistema além da criação e encerramento de processos.

void showLog(void);

Imprime na tela o log de atividades do sistema, realizado pela função logRegister

5. Problemas conhecidos do projeto

- A barra de exebição dos processos na memória, eventualmente fica desalinhada da tabela;

6. Melhorias para o projeto

- Exibir o a tabela de processos em execução na mesma tela do menu;
- Desenvolver uma melhor interface gráfica.
- Método mais eficiente para compactação.

7. Imagens do sistema

Menu inicial

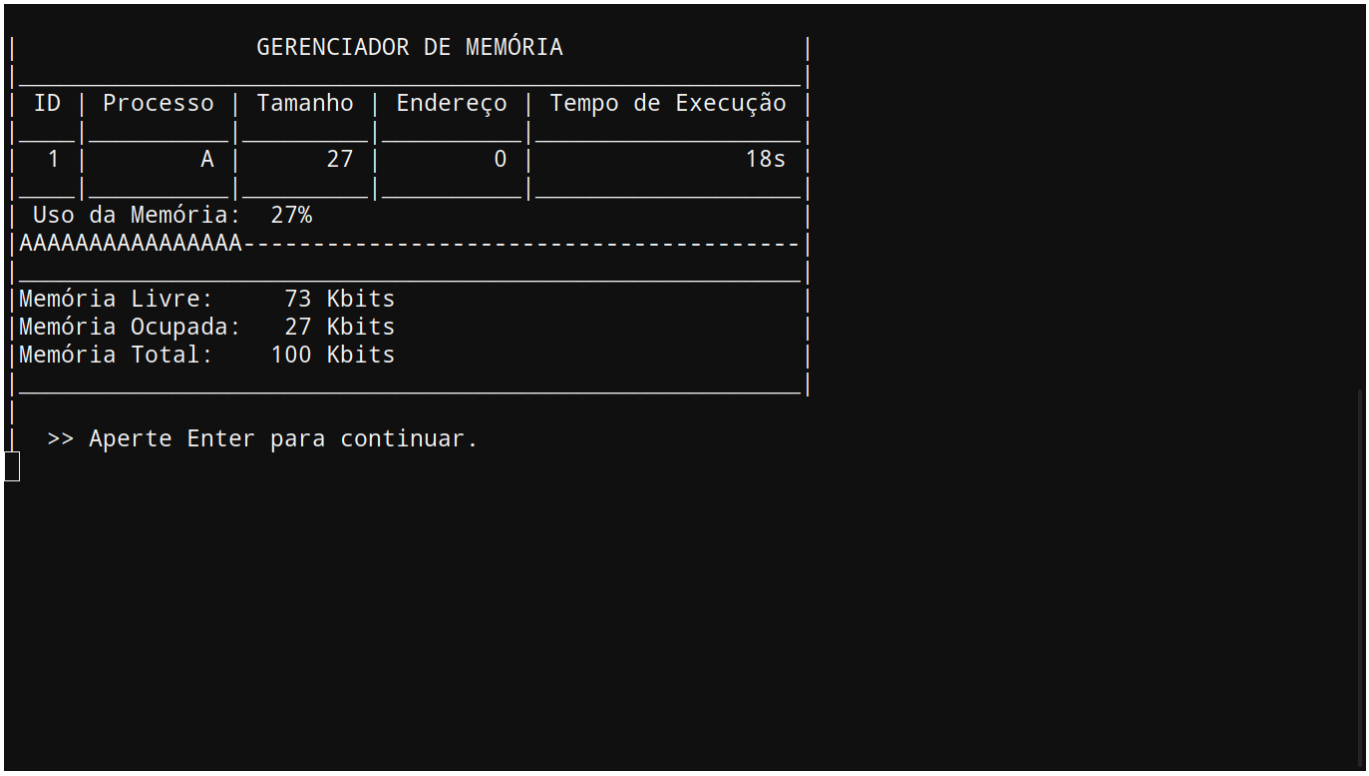
```
|                  GERENCIADOR DE MEMÓRIA                          |
|__________________________________________________________________|
|                                                                  |
|    >>> 1 - Criar novo processo                                   |
|    >>> 2 - Mostrar processos em execução                         |
|    >>> 3 - Encerrar processo em execução                         |
|    >>> 4 - Exibir registros do sistema                           |
|    >>> 5 - Sobre o sitema                                        |
|    >>> 0 - Fechar                                                |
|__________________________________________________________________|

|   >> Escolha uma opção:
|   >>
```

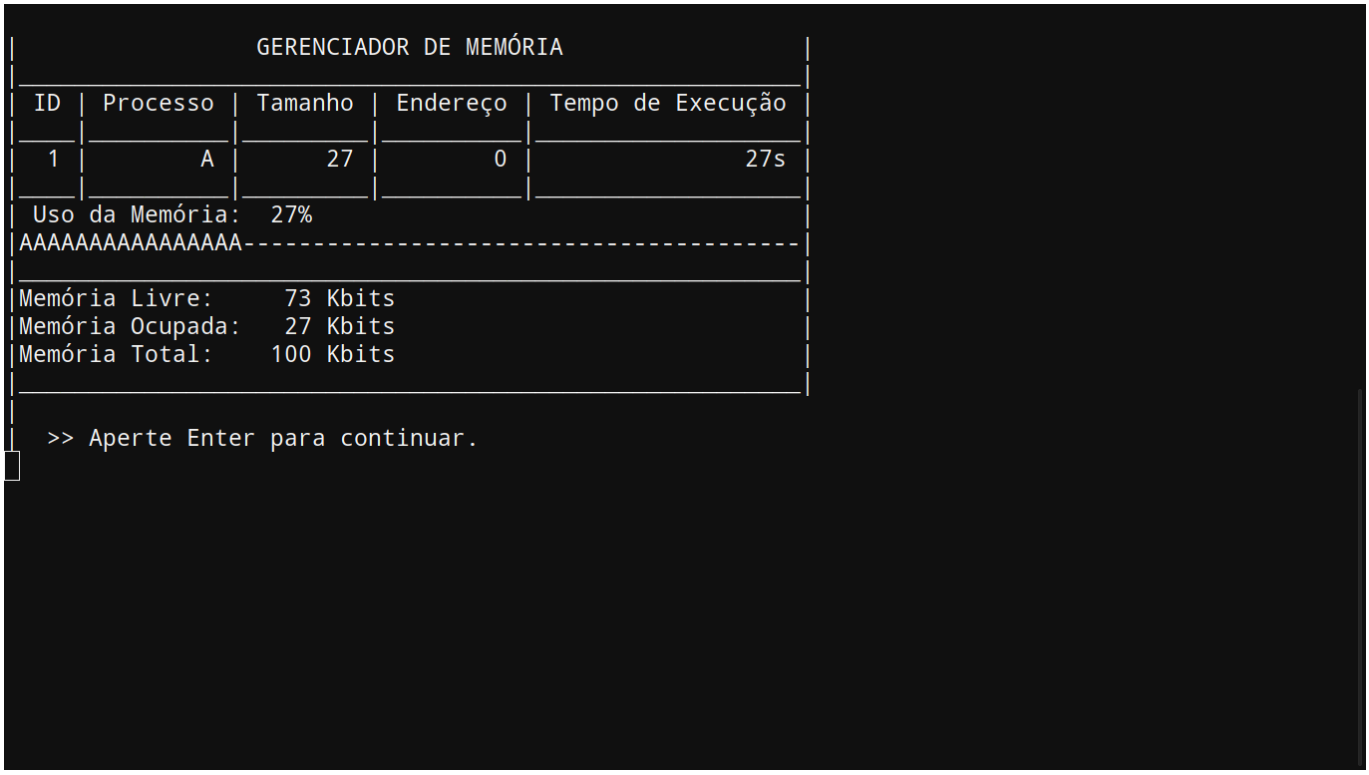
Tela de exibição de processos (não há processos)

```
|                GERENCIADOR DE MEMÓRIA                |
|______________________________________________________|
| ID | Processo | Tamanho | Endereço | Tempo de Execução |
|____|__________|_________|__________|___________________|
|                                                      |
|             NÃO HÁ PROCESSOS EM EXECUÇÃO             |
|                                                      |
|______________________________________________________|
| Uso da Memória:   0%                                 |
|------------------------------------------------------|
|______________________________________________________|
|Memória Livre:    100 Kbits                           |
|Memória Ocupada:    0 Kbits                           |
|Memória Total:    100 Kbits                           |
|______________________________________________________|
|
|   >> Aperte Enter para continuar.
```

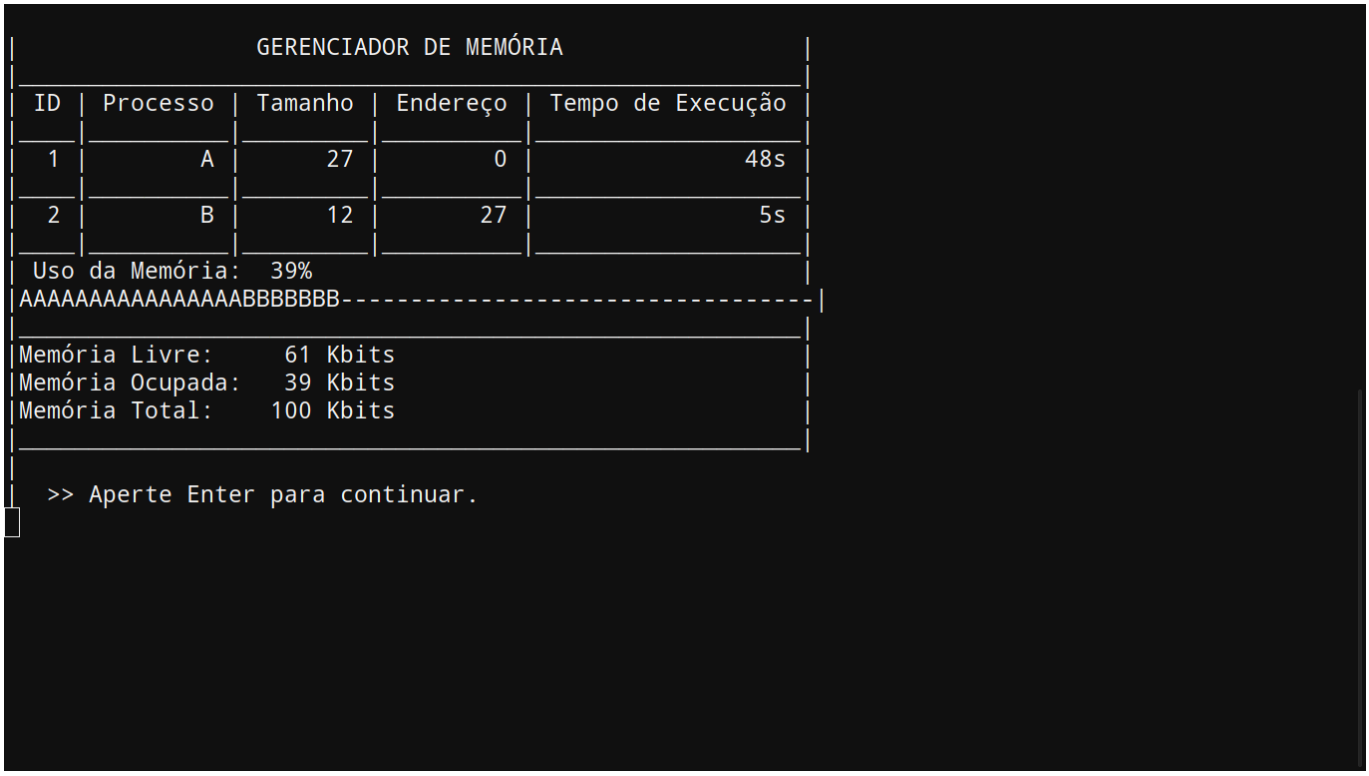
Tela de exibição de processos (processo A criado)



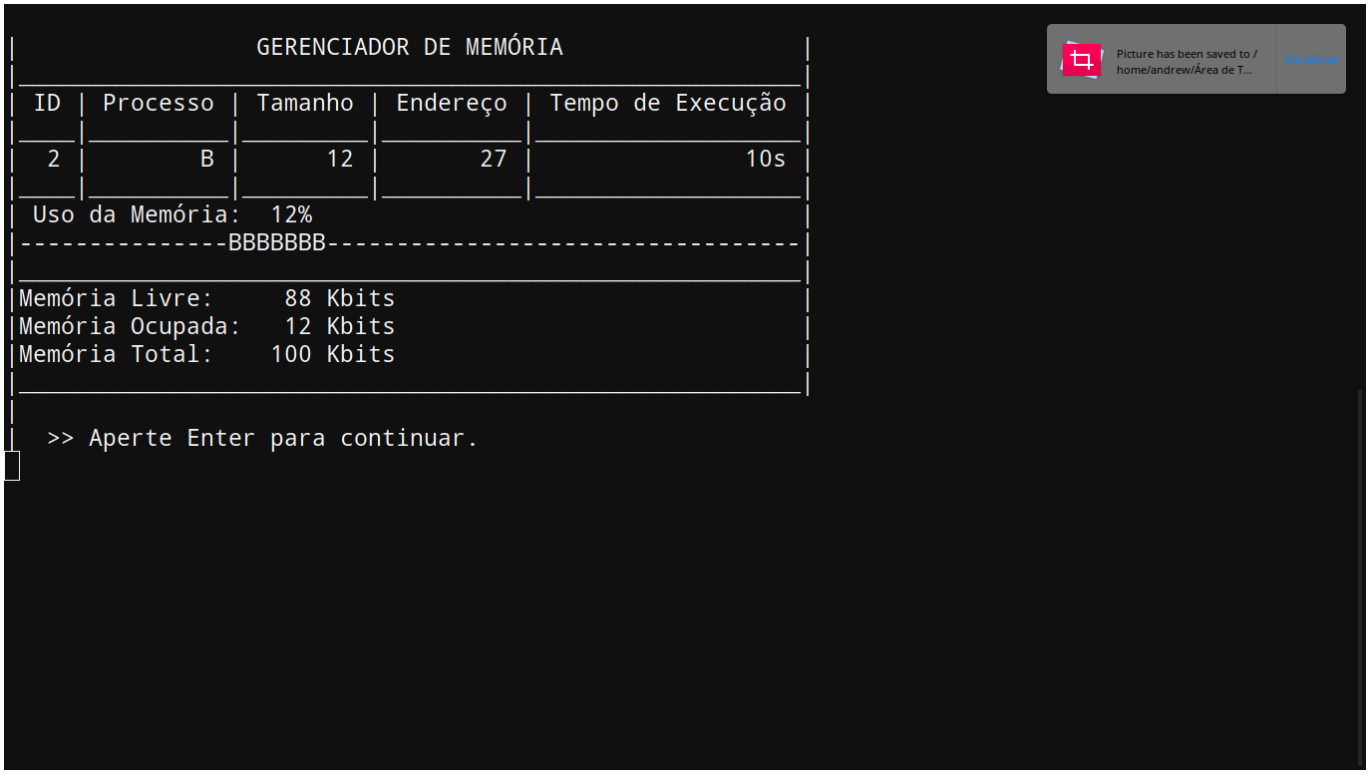
Tela de exibição de processos (processo A após 27s de execução)



Tela de exibição de processos (processo B criado)



Tela de exibição de processos (processo A encerrado)



Relatórios individuais

Andrew Lucas Guedes de Souza 16/0023921

Atividades:

- **Implementação Geral**
  Planejamento de funções e implementação, além de implementar a exibição das telas do sistema.
- **Escrever relatório**
  Descrever funções, objetivos e requisitos do sistema;
- **Organizar estrutura do programa**
  Planejar quais funções necessárias para gerenciamento da memória e qual estrutura utilizar.
- **Gerenciar o projeto**
  Distribuir atividades entre os membros do grupo, e guiar o desenvolvimento do sistema.

Estudos:

- Uso de thread para realizar a verificação e encerramento dos processos, que concluiram o tempo de execução.
- Como utilizar a biblioteca pthread, para criação e gerenciamento de threads em C.
- Utilização da bilioteca time.h e como ela funciona, para implementar a verificação do tempo de execução de um processo.
- Como o gerenciamento de memória é realizado, visando planejar quais funções e procedimentos necessários para realização do sistema simulador de gerenciamento de memória.

Dificuldades:

- Não conseguir implementar um método eficiente de compactação da memória feita com arquivo temporário.
- Consegui identificar onde poderia ser utilizada uma estrutura de pilha, mas não consegui implementar a sua utilização.

As funções mais difíceis de implementar foram:

- **initializeProcess(inicializar e alocar um processo):** Tive dificuldades em implemntar uma função que abrangesse todos os casos de inserção de um novo processo na lista, mas após alguns estudos de como é feita a alocação de um novo processo na memória, consegui entender a lógica, e tornou a implementação mais fácil.
- **compactMemory (compactar memória):** Tentei realizar esse metódo utilizando a estrutura de pilhas, sem sucesso. Mas segui a mesma lógica de pilhas, implementando em arquivo, mesmo sabendo que não é um processo eficiente;
- **showMemory (mostrar informações):**
  No inicio foi fácil, porém depois que decidimos implementar novas funções, essa função se tornou mais complexa, e foi necessário investir mais tempo no estudo de sua implementação e na implementação de fato.

Análise Pessoal do projeto

No início, não pensei que seria muito complicado elaborar esse sistema, e por estar muito ocupado com outras disciplinas, adiei o início da implementação desse projeto. Porém quando comecei a estudar como implementaria, percebi que não seria fácil. E comecei uma maratona de estudo e desenvolvimento do sistema, assim consegui aprender bastante sobre gerenciamento de memória, mais a respeito da linguagem C, como uso de threads, bibliotecas time.h e outras funções que eu

não conhecia.
Com a chegada do prazo de entrega, o projeto percebi que não teria tempo suficiente para termina-lo, então decidimos, atrasar, porém entregar um projeto bem feito para compensar o atraso, implementando novas pequenas funcionalidades e deixando o visual mais bonito.
Por fim, gostei bastante da proposta do projeto, consegui aprender bastante informação nova, e úteis para aplicar em projetos futuros.

Max Henrique Barbosa 16/0047013

Ingressei no grupo dia 10 onde até o primeiro momento havia apenas uma ideia das possíveis funções para implementação do que foi solicitado. Acredito pessoalmente não ter sido de grande auxílio devido a algumas limitações da minha parte, porém tentei auxiliar com alguma coisa que estivesse ao meu alcance. Para a realização do trabalho, procuramos formas de entender como uma memória se comporta para elaborar uma abordagem que fosse possível de sanar o problema. Definimos formas de criar, encerrar e mostrar os processos em execução. Das dificuldades houve uma certa complicação com o tempo e com as possíveis ferramentas que poderíamos utilizar para resolver coisas de alguns requisitos como, a exibição do tempo de cada processo. Além de procurar outras soluções que viessem de encontro com coisas que eram de nosso conhecimento. Houve ao longo do desenvolvimento pesquisas dos membros e solicitação de auxílio com outros alunos que estavam um pouco mais a frente do curso que entendiam melhor do assunto.
Tirando isso conseguimos alcançar grande parte dos requisitos solicitados, desde tempo e parâmetros a uma possibilidade de poder gravar dados em arquivo.

Nivaldo Pereira Lopo Junior 12/0039460

Atividades:

- Implementação
- Escrita do relatório

Estudos:

- Estudo de maneiras de realizar a compactação de processos.
- Utilização da biblioteca time.h, para implementar a geração da semente para geração de numero pseudorrandômico.
- Estudo da função rand(); para geração pseudorrandômica da duração de um novo processo.
- Estudos relacionado a causas de falhas por segmentação.

Dificuldades:

- Diversas tentativas de realizar a compactação da memoria, primeiramente ao tentar utilizar pilhas o programa estava encerrando devido a falha por segmentação, após repetidamente tentar resolver este problema (utilizando outras variáveis, vetores e outras maneiras de lidar com a memoria) não foi encontrada solução com pilha para a compactação. Foi então tentado realizar a compactação ao tentar religar os processos ainda ativos retirando a conexão dos mesmos com buracos, ainda assim não foi possível. Portanto a compactação foi realizada utilizando um arquivo temporário.

Análise Pessoal do projeto:

- A realização desse projeto foi interessante para mim por ser o primeiro projeto de software que realizei em grupo na universidade, uma vez estou realizando essa matéria como optativa na minha grade curricular (curso engenharia aeroespacial). Por estar ocupado com outras matérias e com meu trabalho de conclusão de curso, adiei o inicio deste trabalho e acabei tendo algumas dificuldades, com ajuda de colegas que já cursaram a disciplina e do membro do grupo Andrew consegui entender bem o problema a ser trabalhado e como realizar a implementação das funções que trabalhei no grupo. Em geral, a experiência de realizar esse trabalho e de cursar essa disciplina como um todo tem me ajudado a me profissionalizar e me capacitar mais em desenvolvimento de software, conhecimento que tenho também aplicado em meu TCC, além de um conhecimento mais avançado da linguagem C.