Pedro Victor Rodrigues de Carvalho - 17/0113043
Pedro Augusto Coelho Nunes - 16/0141044
Rafael Mascarenhas Dal Moro - 17/0021041

INTRODUÇÃO

Este documento traz as especificações e etapas de implementação do Trabalho Prático da disciplina Sistemas Operacionais, ministrada no semestre de 2023/1. Serão abordados os seguintes tópicos no decorrer deste relatório: visão geral do projeto, objetivos, organização dos arquivos e execução do código, implementação das funcionalidades, membros participantes e conclusão do projeto.

Sistemas operacionais são softwares fundamentais para diversos contextos da computação. Eles gerenciam recursos e fornecem uma interface entre usuários e hardware, garantindo a execução eficiente de programas, proteção de dados, entre outros.

Um pseudo-sistema operacional é uma simulação simplificada de um sistema operacional real, usado para aprendizado e compreensão dos conceitos básicos deste campo de estudos. A implementação de um pseudo-sistema operacional é um processo que envolve uma série de desafios, como definir a estrutura básica do software, implementar diversos algoritmos de gestão de recursos, e integrar diferentes módulos do sistema. É uma atividade valiosa para o aprendizado dos fundamentos dos sistemas operacionais.

VISÃO GERAL

O objetivo do Trabalho de Implementação foi desenvolver um pseudo-sistema operacional multiprogramado, composto por um Gerenciador de Processos, um Gerenciador de Memória, um Gerenciador de E/S e um Gerenciador de Arquivos.

Dentre os requisitos do projeto, constam os seguintes pontos:

- O gerenciador de processos deve escalonar as tarefas da CPU em 4 níveis de prioridade;
- O gerenciador de memória deve assegurar o controle de acesso dos dados de um processo, impedindo que uma tarefa não acesse a memória de outra;
- O gerenciador de E/S deve gerir a atribuição e liberação de todos os recursos disponíveis, garantindo exclusividade de uso; e
- O gerenciador de arquivos deve viabilizar a criação e exclusão de arquivos de maneira condizente com o modelo de alocação determinado durante a implementação.

OBJETIVOS

1. Estudar a teoria relacionada ao desenvolvimento de SO's;
2. Definir uma abordagem de implementação de pseudo-SO;
3. Implementar a solução proposta; e
4. Apresentar e documentar detalhadamente os resultados do Trabalho.

Pedro Victor Rodrigues de Carvalho - 17/0113043
Pedro Augusto Coelho Nunes - 16/0141044
Rafael Mascarenhas Dal Moro - 17/0021041

ESPECIFICAÇÕES TÉCNICAS

O trabalho foi desenvolvido em ambiente integrado *VsCode* com uso do Subsistema do Windows para Linux (WSL) e implementado na linguagem de programação C++ utilizando o padrão C++11. Os arquivos, organizados em pasta única, foram divididos da seguinte forma:

Tipo	Nome do Arquivo	Objetivo
Pasta	Headers	Armazenar os arquivos de Header do projeto
Arquivo	Main (Main.cpp)	Código principal de execução do projeto
Arquivo	PseudoOS (PseudoOS.cpp)	Implementar classe que engloba todo o sistema
Arquivo	File (File.cpp)	Implementação da classe Gerenciadora de Arquivos
Arquivo	Memory (Memory.cpp)	Implementação da classe Gerenciadora de Memória
Arquivo	Process (Process.cpp)	Implementação da classe Gerenciadora de Processos
Arquivo	Queue (Queue.cpp)	Implementação da classe Gerenciadora de Filas
Arquivo	Resource (Resource.cpp)	Implementação da classe Gerenciadora de Dispositivos
Arquivo	Dispatcher (Dispatcher.cpp)	Implementação da classe Dispatcher

Essa distribuição de arquivos se justifica, em um primeiro momento, por exigência das especificações do trabalho. Contudo, percebe-se que esta abordagem também contribui para a modularização do projeto, concentrando funcionalidades específicas em arquivos setorizados (com cabeçalho e implementação). Dessa forma, o arquivo *Main* reduz-se a executar a função de inicialização do sistema que chama as primeiras rotinas do pseudo-SO.

Para executar este trabalho, faça a descompressão do arquivo em anexo ao envio. Caso não possua o pacote “make” instalado no sistema, instale-o. Num sistema como o Ubuntu, isso é feito entrando as seguintes linhas no terminal:

```
sudo apt update
sudo apt install make
```

Após instalar o make, acessando a pasta descomprimida do Trabalho, execute a seguinte linha de comando em um Painel de Terminal:

```
make
```

Em seguida, digite o próximo comando:

Pedro Victor Rodrigues de Carvalho - 17/0113043
Pedro Augusto Coelho Nunes - 16/0141044
Rafael Mascarenhas Dal Moro - 17/0021041

```
.\pseudoos <arquivo de processos> <arquivo de sistema de arquivos>
```

É recomendado que os arquivos tenham extensão ".txt". Além disso, é possível utilizar os arquivos de teste disponibilizados na pasta da implementação para executar e verificar o funcionamento do software desenvolvido.

IMPLEMENTAÇÃO

main.cpp:

Este arquivo tem três funções principais: abrir os arquivos de teste, iniciar a execução do pseudo sistema operacional e imprimir no terminal eventuais erros nos arquivos de entrada e se o sistema executou corretamente. Essa execução é feita usando uma classe que segue o padrão singleton, que é utilizado em várias outras classes no programa. Esse padrão garante que apenas uma instância da classe exista, de forma que essa instância única seja acessível de qualquer ponto do sistema. Nesse primeiro caso, a classe PseudoOS é instanciada e seu método Run é chamado para começar o processamento.

PseudoOS.hpp e PseudoOS.cpp:

Contém a classe *PseudoOS*. Essa classe, ao ser criada, cria as instâncias dos gerenciadores de arquivos, memória, processo, fila de processos e recursos, todos também no padrão singleton. Ela inicia o SO por meio do método Run. Nesse método, os arquivos de entrada são lidos e seus dados armazenados em estruturas de dados nas classes *ProcessManager* e *FileManager*. Feita a leitura da entrada, o sistema executa um loop para simular o funcionamento de um processador, e executa-o até que todos os processos obtidos da entrada tenham sido executados. Foi projetado que teria-se a relação de 1 loop = 1 quantum.

Process.hpp e Process.hpp:

Neste módulo encontram-se duas classes:

- *Process:* Representação de um processo no sistema operacional;
- *ProcessManager:* Gerenciador de processos.

O gerenciador de processos armazena uma lista de processos obtida do primeiro arquivo de entrada. Esses processos são compartilhados com a classe *QueueManager* para serem armazenados nas filas, onde o escalonamento seria de fato feito.

A classe Process possui membros e um método para que esse processo execute uma única instrução por quantum quando for escalonado para a CPU. Ele tem meios de determinar por quanto tempo executar, e se comunica com a classe *FileManager* para efetuar suas operações de arquivos durante seu tempo de execução.

Pedro Victor Rodrigues de Carvalho - 17/0113043
Pedro Augusto Coelho Nunes - 16/0141044
Rafael Mascarenhas Dal Moro - 17/0021041

File.hpp e File.cpp:

Neste módulo encontram-se três classes:

- *File:* Representação de um arquivo no sistema operacional;
- *FSOperation:* Representação de uma operação no sistema de arquivos;
- *FileManager:* Gerenciador de arquivos e disco.

O gerenciador de arquivos é responsável por cuidar dos arquivos existentes no disco do sistema e por executar operações que os processos requisitem sobre o sistema de arquivos, podendo elas ser de criação ou deleção. Ele inicia o sistema de arquivos com base no segundo arquivo de entrada e cuida da alocação e liberação de espaço em disco seguindo um padrão de alocação contígua com *First-Fit*, além de prover a persistência dos arquivos após o término da execução do sistema.

Dispatcher.hpp e Dispatcher.cpp:

Este módulo contém a classe Dispatcher, que é instanciada dentro do método *PseudoOS::Run*. Ela representa um "processo" do SO que tem função de mostrar em tela o que está sendo feito pelo SO. Durante a execução do sistema, ele mostra os detalhes do processo que foi escalonado para execução, e após o término do sistema, mostra um histórico de operações no sistema de arquivos e o estado atual do disco.

Queue.hpp e Queue.cpp:

Este módulo contém a classe *QueueManager*, que é responsável por conter as filas de escalonamento de processos e armazenar referências aos processos que estão contidos nessas filas. Além disso, também possui uma fila de "criação" de processos, que serve para ajudar a simular o surgimento de um novo processo em um dado instante de tempo e determinar quando o sistema deve terminar sua execução.

Das filas do sistema, somente a fila de tempo real foi efetivamente implementada. As filas de usuário existem como estruturas de dados, mas não há código que as utilize de maneira funcional. Sendo assim, o sistema só funciona para processos de tempo real com um escalonamento não-preemptivo, usando o algoritmo *FIFO*.

Memory.hpp e Memory.cpp:

Este módulo contém a classe *MemoryManager*, que é responsável por cuidar da alocação e liberação de memória. Além disso, este gerenciador provê meios para verificar a existência e fornecer a localização de processos em memória. Isso é possível pelo uso do mapa de ocupação de memória que essa classe possui. Foi implementado um algoritmo de alocação contígua com política *First-Fit*.

Resource.hpp e Resource.cpp:

Este módulo contém a classe *ResourceManager*, o gerenciador de dispositivos (recursos) do sistema. Nele, foram declaradas as filas de execução para cada dispositivo

Pedro Victor Rodrigues de Carvalho - 17/0113043
Pedro Augusto Coelho Nunes - 16/0141044
Rafael Mascarenhas Dal Moro - 17/0021041

de E/S e a função *IORequest, que* organiza cada uma das solicitações de uso de E/S por processo e por dispositivo.

Como somente processos de usuário utilizam tais recursos, e tanto as filas de processos de usuário como a funcionalidade de *time-sharing* (preempção) não foram implementadas, essa classe está incompleta.

EQUIPE

As tarefas foram divididas espontaneamente em momentos síncronos e assíncronos de produtividade. Para facilitar a compreensão, coesão e depuração do código, técnicas como *pair programming* foram amplamente utilizadas durante a produção do projeto. Pedro Victor foi o principal desenvolvedor e contou com a revisão em tempo real de Pedro Augusto e Rafael, além de revisar a elaboração do relatório. Pedro Augusto foi responsável pela revisão dos códigos de Rafael e pela elaboração do relatório. Por fim, Rafael elaborou o relatório e desenvolveu certas funções de código, contando com o apoio de ambos os Pedros.

CONCLUSÃO

O Trabalho cumpriu com o papel de aprimorar o entendimento dos alunos acerca do tema, bem como trouxe conhecimentos práticos acerca da implementação de Sistemas Operacionais. Como trabalhos futuros, destaca-se a implementação do escalonamento de processos de usuário, a restrição de permissão para que processos de usuário possam excluir apenas arquivos criados por eles e, finalmente, o algoritmo que garante exclusividade de uso de dispositivos E/S .

Sistemas Operacionais são um campo de pesquisa fundamental para o desenvolvimento da Computação enquanto Ciência e Engenharia. Portanto, os estudos aplicados sobre um setor que impacta diretamente a qualidade de vida da população têm potencial para solucionar diversos problemas em nossa sociedade

REFERÊNCIAS

- Stallings, W. Operating Systems Internals and Design Principles, Pearson Education, 2009.
- de Araújo, A.P.F. (no date) *Sistemas Operacionais - Especificação do Trabalho de Implementação*, *APRENDER3*. Available at: https://aprender3.unb.br/ (Accessed: 07 July 2023).
- Tanenbaum, A. S., Bos, W. Sistemas Operacionais Modernos. Person, São Paulo, 4° ed., 2016.