Relatório de Implementação de pseudo SO

Artur Padovesi Piratelli - 211038208
Bernardo Manciola Lobo - 211026486
Paulo Henrique Borges Martins - 211068790

Introdução

Este trabalho busca implementar um pseudo-SO, contendo gerenciador e escalonador de processos, gerenciador de memória, gerenciador de I/O e um sistema de arquivos. Para iniciar o programa, dois arquivos txt devem ser fornecidos, um que passa informações dos processos e o outro passando informações sobre o estado do sistema de arquivos e operações a serem realizadas em cima dele.

Descrição das ferramentas/linguagens utilizadas

Foi utilizada a linguagem Python 3.11 com bibliotecas instaladas por pip, garantindo compatibilidade com qualquer ambiente UNIX. As bibliotecas utilizadas foram pytest para testes unitários conforme necessário, e ruff para análise estática de código. A arquitetura monolítica segue a seguinte estrutura:

```
simple-os-sim/
├── main.py                    [ENTRADA PRINCIPAL]
├── tests/                     (Testes unitários feitos conforme necessidade)
└── src/simple_os/
    ├── cpu.py                 [CPU - Executor de Instruções]
    ├── simulation_utils.py    [Utilitários de Simulação]
    ├── process/               [MÓDULO DE PROCESSOS]
    │   ├── pcb.py             (Process Control Block)
    │   ├── process_manager.py (Gerenciador de Processos)
    │   └── scheduler.py       [MÓDULO DE FILAS]
    ├── memory/                [MÓDULO DE MEMÓRIA]
    │   ├── memory.py          (Representação da Memória)
    │   └── memory_manager.py  (Gerenciador de Memória)
    ├── files/                 [MÓDULO DE SISTEMA DE ARQUIVOS]
    │   ├── disk.py            (Representação do Disco)
    │   ├── file.py            (Entidade de Arquivo)
    │   ├── input_reader.py    (Parser de Entrada)
    │   ├── system.py          (Sistema de Arquivos)
    │   └── manager.py         (Gerenciador de Arquivos)
    └── resource/              [MÓDULO DE RECURSOS]
        └── resource_manager.py (Gerenciador de Recursos | I/O)
```

Descrição da solução

Para simplificar, e seguindo orientações, operações realizadas no sistema de arquivos são executadas em ordem, sem consideração para o tempo de chegada ou de execução dos processos. Sendo assim, dois fluxos de execução podem ser encontrados na simulação, um para processos e um para o sistema de arquivos, como pode ser visto na Figura 2 (TODO: modificar e adicionar o outro fluxo de execução). O diagrama de comunicação entre módulos (TODO: perguntar pro chat gpt o nome disso e se realmente é feito assim) pode ser visto na Figura 1.

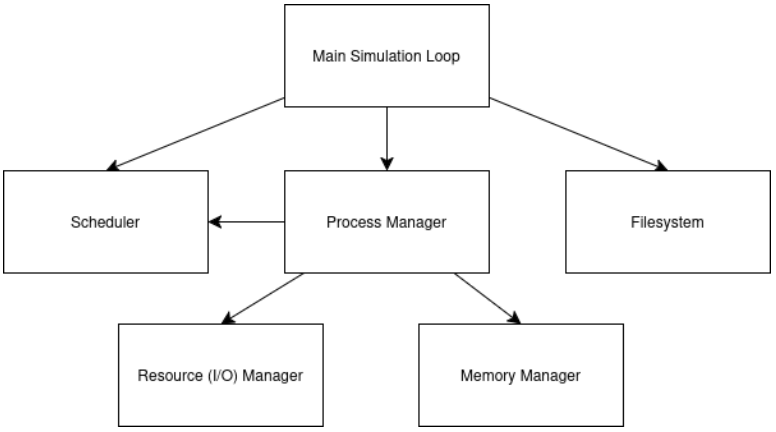

Figura 1 - TODO

O *entrypoint* do programa lê as entradas, inicia o fluxo de execução dos processos e por fim executa as operações no sistema de arquivos. Os módulos de sistema operacional utilizam algoritmos tradicionais, mas também possuem responsabilidade adicional de imprir saídas de acordo com o pedido para o projeto. Por exemplo a CPU que imprime o que o processo está executando. A maneira de simular comunicação entre os módulos foi com a utilização de orientação a objeto e singletons, sendo semelhante à troca de mensagens síncronas a partir de chamadas tradicionais de métodos de instância.

A implementação da simulação do sistema de arquivos ocorre após a execução do fluxo de execução dos processos. O simulador lê o arquivo de entrada, inicializando o sistema com o número de discos indicado, além de definir os arquivos presentes no preenchimento inicial do disco e as operações de criação e exclusão de arquivos que serão executadas. O sistema de arquivos recebe do sistema de processos a lista de processos criados, e organiza-os na lista de processos e na lista de processos de tempo real para gerenciar as permissões das operações requisitadas. Com isso, o sistema executa cada operação pendente, indicando sucesso ou falha na execução, e exibe o estado final do disco após todas as operações.

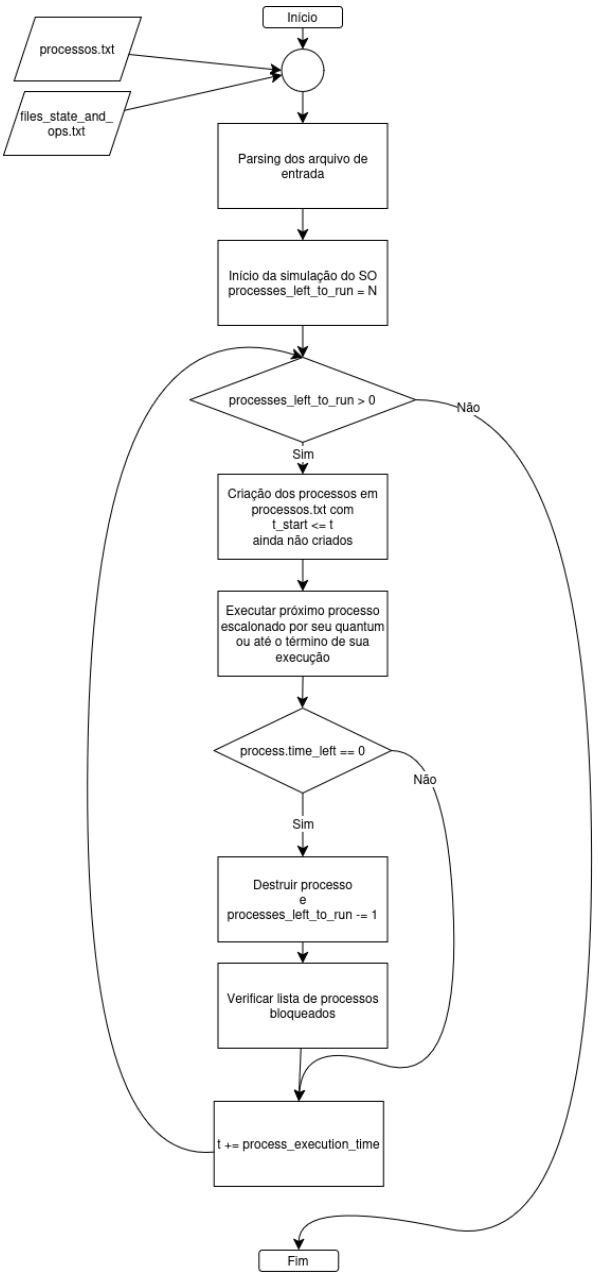

Figura 1 - Loop de execução dos processos

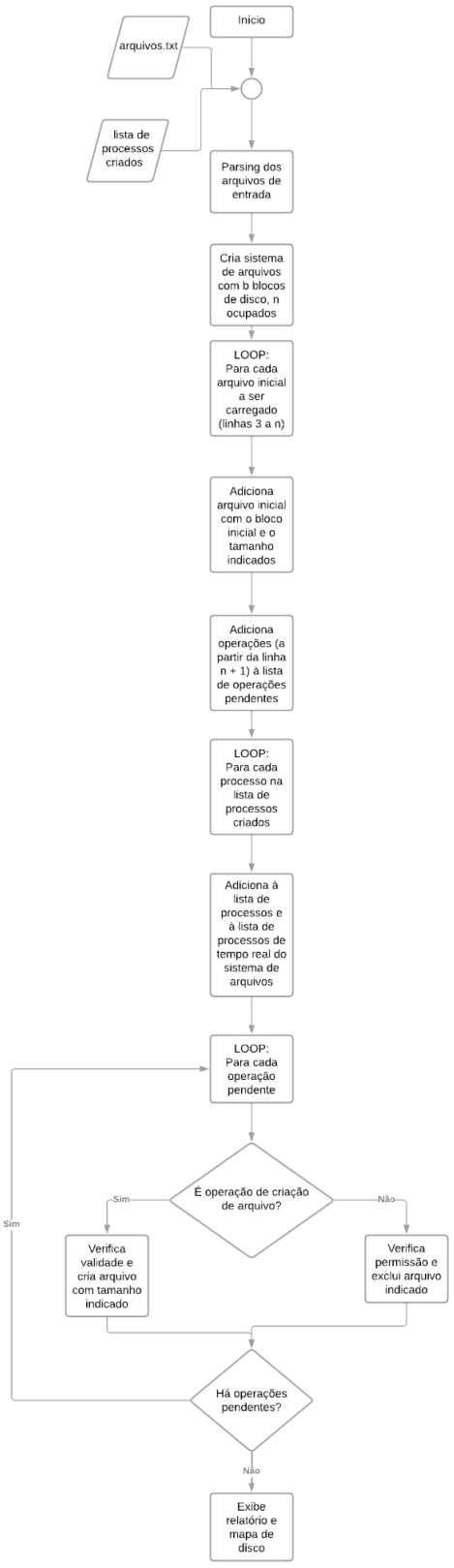

Figura 2 - Loop de execução dos arquivos

Dificuldades encontradas

Não é especificado na definição do trabalho como exatamente lidar com a alocação de recursos e deadlocks. A simples alocação ordenada de recursos impede a ocorrência de ciclos no grafo de alocação de recursos. Para manter consistência no caso de um processo bloqueado que tenta pedir recursos diversas vezes, essa requisição deve ser um método reentrante. Após implementação com essas considerações, o sistema funciona corretamente.

A simulação não provê boa forma de lidar com interrupções na hora que um processo de maior prioridade chega na fila. Foi difícil encontrar uma solução que não fosse muito ruim em qualidade de código. Foi decidido que o loop principal do simulador é um oráculo que sabe quando os processos vão chegar, então prevê quando um processo vai ser interrompido por outro de maior prioridade chegando.

Comunicação entre as operações do sistema de arquivos com o módulo dos processos. Para saber as permissões de um processo é necessário diferenciar, a partir do pid, se o processo é ou não de tempo real. Por simplificação da simulação, operações no sistema de arquivos deve ser separado da execução dos processos. No entanto, isso significa que não é útil a tabela de processos guardada pelo gerenciador de processos, pois quando as operações em arquivos forem executadas (seja antes ou depois), a tabela de processos estará vazia.

Para resolver esse problema, os processos foram ordenados por tempo de chegada e seu índice nessa lista foi assumido como seu pid, separando a lógica da tabela de processos e de permissões no sistema de arquivos.

Papel dos alunos

Artur - Gerenciador de processos, escalonador e testes
Paulo - Gerenciador de recursos e de memória, e testes
Bernardo - Sistema de arquivos, loop principal e testes

Bibliografia

[1] Fundamentos de sistemas operacionais / Abraham Silberschatz, Peter Baer Galvin, Greg Gagne. - 9. ed. - Rio de Janeiro: LTC, 2015.