Trabalho Prático: Pseudo-SO

André Cássio Barros de Souza
160111943
andreloff15@gmail.com

Carolina Estrella Machado
180074792
carolina@host.com.br

Hélio Santana da Silva Júnior
190108444
helio.ssjr@gmail.com

Raphael Paula Leite Muller
170113477
raphaelplmuller@icloud.com

Vitor Fernandes Dullens
160148260
vitordullens2.0@gmail.com

***Resumo*—Este relatório descreve a implementação de um pseudo-SO multiprogramado composto por um gerenciador de processos, por um gerenciador de memória, por um gerenciador de E/S e por um gerenciador de arquivos. Foi utilizado a linguagem python para a implementação.**

***Index Terms*—sistemas operacionais;**

I. Introdução

Sistemas operacionais(SO) estão a cada dia mais presentes nas vidas das pessoas, tendo em vista que nos últimos anos a explosão dos smartphones e smart tv's proporcionou o contato de bilhares de pessoas com aparelhos que possuem interação com o usuário a um computador complexo através de um sistema operacional. O que muitas pessoas comuns não percebem é que os sistemas operacionais desses dispositivos são muito parecido ou até mesmo os mesmos de computadores pessoais. Existem funcionalidades e processos que são básicos entre todos os sistemas operacionais.

Este trabalho tem como objetivo emular um pseudo sistema operacional com as funcionalidades básicas de um SO. A sessão II irá apresentar as ferramentas utilizadas para construção desse pseudo-SO. A sessão III irá apresentar as restrições e limitações do pseudo-SO bem como as soluções propostas para resolver o problema dado [1]. A sessão IV irá apresentar como foi dividido o trabalho com os integrantes do grupo. E por fim a sessão V irá discorrer sobre as dificuldades encontradas, as soluções adotadas para essas dificuldades e propostas futuras para resolução dos problemas enfrentados.

II. Ferramentas

A linguagem escolhida para o desenvolvimento do pseudo-SO foi a linguagem Python, pelo fato de ser uma linguagem dinâmica e agilizar o desenvolvimento de um código de médio porte. Não foi utilizado nenhuma biblioteca externa do Python. O código desenvolvido foi versionado com o GitHub, e escrito na ferramenta VSCode. Além disso, a ferramenta LiveShare foi utilizada para colaboração entre os membros.

III. Solução

A. Arquitetura do projeto

O pseudo-SO possui quatro módulos: gerenciador de processos, gerenciador de memória, gerenciador de E/S e gerenciador de arquivos. Porém foram implementados apenas os módulos de gerência de processo e gerência de arquivos, além do kernel.

B. Gerência de processos

Os processos prontos para execução podem ser classificados em processos de tempo real, ou seja, aqueles com prioridade zero, ou em processos de usuário, com prioridade maior ou igual a 1. Os processos de tempo real que detém o estado pronto de execução são enfileirados segundo o algoritmo FIFO. Isto é, o primeiro processo a ser enfileirado sempre será o primeiro a ser executado. Os processos de usuário, por sua vez, são enfileirados de acordo com o algoritmo de múltiplas filas com realimentação. Para isso, existem três filas de prioridade implementadas, que organizam os processos de acordo com o seu grau de prioridade, numeradas de 1 a 3. A fila 1 é a mais prioritária, e a fila 3 é a menos prioritária dentre essas.

Na implementação proposta pelo grupo, todos processos de usuário que passam a deter o estado pronto são enfileirados na fila 1 de acordo com sua prioridade. A partir disso, as seguintes situações podem ocorrer:

- Se enquanto um processo de usuário é executado, um processo de tempo real tornar-se pronto, é realizada troca de contexto e o processo de usuário é reposicionado dentro da mesma fila à qual pertence.
- Se enquanto um processo de usuário é executado, um processo de usuário em uma fila superior tornar-se pronto, é realizada troca de contexto e o processo de usuário é mantido na mesma fila à qual pertence.
- Se enquanto um processo de usuário é executado, um processo de usuário de maior prioridade tornar-se pronto na mesma fila, o processo atual é rebaixado para a próxima fila de menor prioridade.
- Se um processo de usuário executar uma requisição de E/S, esse é reposicionado na mesma fila à qual pertence.

Por fim, vale ressaltar que uma fila de menor prioridade só passa a ser executada quando as filas de prioridade maior, incluindo a de processos de tempo real, estiverem vazias. Por exemplo, um processo que esteja na fila 3 só será executado quando ambas filas de processos de tempo real, 1 e 2 estiverem vazias. Se algum processo for inserido nessas enquanto a fila

3 é executada, será realizada troca de contexto imediata, visto que processos de usuário são preemptáveis.

C. Gerência de arquivos

No pseudo-SO implementado, os arquivos são salvos em uma estrutura de memória que consiste em um vetor de blocos que armazenam frações de um arquivo. Essa é gerida segundo o algoritmo *first-fit*. Isso significa que, quando um processo deseja escrever um arquivo na memória, essa é percorrida com origem na posição inicial, de índice 0, até que seja encontrado um conjunto contíguo de blocos livres que possua tamanho igual ao tamanho do arquivo. Caso não haja espaço disponível para um determinado arquivo, o usuário será informado por meio do console de saída.

Além disso, esse módulo implementa uma tabela de dispersão que mapeia cada nome de arquivo salvo no sistema com o endereço de memória do seu primeiro bloco e o PID do processo que o escreveu. Essa é consultada quando um processo requisita a deleção de um arquivo. Primeiramente, realiza-se uma validação para determinar se esse possui permissão para tal. Caso possua, o endereço inicial do arquivo e seu tamanho são utilizados pelo gerenciador de arquivos para realizar a sua remoção.

Em relação às permissões de escrita e remoção de arquivos, os processos podem acessar qualquer arquivo na memória, independentemente de seu tipo. Além disso, todos podem realizar escrita. Contudo, somente processos de tempo real possuem permissão para deletar qualquer arquivo. Um processo de usuário qualquer, por sua vez, só pode deletar arquivos criados pelo próprio.

IV. Divisão de Tarefas

Os alunos André e Carolina fizeram *pair programming* no desenvolvimento do módulo de Gerência de Arquivos, tal como a segunda parte do *output* do trabalho, que diz respeito as operações e seus respectivos resultados além da revisão e colaboração no relatório. Os alunos Raphael e Hélio foram responsáveis pela pesquisa e definição de requisitos das filas de gerência de processos e também da elaboração das sessões referentes a gerencia de processos deste relatório. O aluno Vitor Dullens foi responsável pela implementação do kernel além de ter colaborado na escrita desse relatório.

V. Conclusão

A primeira dificuldade encontrada pelo grupo foi na definição das tarefas, de tal forma que todos pudessem trabalhar de forma independente e mesmo assim produzir uma base de código consistente. Para isso foi realizada uma etapa de planejamento, na qual as funcionalidades foram divididas em subtarefas. Além disso, os integrantes definiram a estrutura geral do código, as estruturas de dados a serem utilizadas, bem como a identificação de quais módulos iriam requerer colaboração no momento de integração do código produzido por cada aluno. Outra dificuldade encontrada foi a definição da política de gerenciamento dos processos de usuário. Isso porque foi necessário realizar diversos testes com o intuito de analisar a distribuição do tempo de CPU entre processos de diferentes prioridades. Com base nessas observações, estabeleceram-se quais seriam os critérios para preempção e movimentação desses processos entre filas. Alguns módulos não foram totalmente implementados, como a fila de prioridade dos processos de usuário, o módulo de gerencia de E/S e memória de processos.

Referências

[1] A. S. Tanenbaum and H. Bos, *Modern operating systems*. Person, 2015, vol. 1.