Universidade de Brasília

Instituto de Ciências Exatas

Departamento de Ciência da Computação

Fundamentos de Sistemas Operacionais

Implementação Pseudo-SO

Nome:	*Matrícula:*
Vinícius Caixeta	18/0132199
Johannes Peter Schulte	15/0132662
Igor Bispo de Moraes	17/0050432
Matheus Arruda Aguiar	18/0127659

Outubro de 2021

1 Introdução

Sistemas Operacionais são programas complexos que a partir de um conjunto de componentes executam tarefas cruciais para o funcionamento de computadores. Esses componentes possuem como função, principalmente, o gerenciamento de recursos. Desta forma, este trabalho tem por objetivo a construção de um pseudo sistema operacional, implementando módulos que simulam o funcionamento do gerenciador de processos, gerenciador de memória, gerencia de arquivos, gerenciador de entrada e saída além de um módulo que simula o funcionamento das filas de processos, fazendo uma diferenciação entre processos em tempo real e processos de usuário.

2 Ferramentas e Linguagens

Para esta implementação foi utilizada a linguagem Python versão 3.8.10, onde cada módulo possui uma classe representante de determinado gerenciador, sendo a *dispatcher* responsável por iniciá-los. Na realização de operações como inicialização de processos e arquivos criou-se os arquivos processes.txt e files.txt, os quais possuem informações de dados de processos, volume do disco rígido, arquivos existentes e operações de alocamento e exclusão de arquivos. Para executar o programa é preciso digitar no terminal: "python3 dispatcher.py input/processes.txt input/files.txt".

3 Descrição das Soluções

O pseudo-SO implementado foi dividido em cinco módulos referentes aos componentes principais, assim como o módulo principal do *dispatcher* que age como o processo principal do sistema operacional, fazendo a junção de todos os módulos.

3.1 O pseudo-SO

A inicialização do sistema operacional é feita pelo processo principal *dispatcher*, que recebe como entrada informações sobre cada processo que deve ser executado, assim como as operações de arquivo que devem ser feitas.

Como o objetivo da implementação é apenas de simular o funcionamento de um sistema operacional, o programa não lida com paralelismo real, apenas faz uma simulação de paralelismo. Para isso, o *dispatcher* inicializa uma variável *exec_time* que representa o tempo de execução em segundos, a qual também é utilizada para o tempo de *quantum*.

Desta forma, depois de inicializar as classes dos outros módulos, o *dispatcher* possui um *loop* principal para a execução dos processos. Essa estruturam de repetição executa diante de duas condições: ainda existem processos não inicializados, ou ainda existem

processos sendo executados. Cada iteração representa 1 segundo de tempo da execução do sistema operacional, onde as filas de tempo real e processo de usuário são executadas de forma paralela, como se houvessem dois processadores.

3.2 Módulo de Processos

O módulo de processos tem como objetivo cuidar das classes e estruturas de dados relativas ao processo, assim como métodos para lidar com essas estruturas.

A lista de processos do SO foi implementada utilizando uma classe chamada *ProcessList*. Essa classe lê o arquivo de processos, e adiciona em uma lista seguindo o padrão **<tempo de inicialização>, <prioridade>, <tempo de processador>, <blocos em memória>, <número-código da impressora requisitada>, <requisição do scanner>, <requisição do modem>, <número-código do disco>**, sendo que, cada linha do arquivo de processos, corresponde a um processo. Durante a inicialização da classe, é atribuído um PID para cada processo indo de 0 até **n**, em que **n** é o número de processos.

Quando um processo i-ésimo é executado, o tempo restante do processo com PID i é decrescido em um segundo, *quantum* adotado pelo SO, e o contador de instruções é acrescido em um para indicar que a próxima instrução deverá ser executada na próxima chamada de execução.

O módulo dos processos é utilizado diretamente pelo módulo *dispatcher* e pelo módulo de filas. A interação entre esses será descrita com profundidade nos módulos respectivos.

3.3 Módulo de Filas

O módulo de filas deve representar duas filas distintas, a fila contendo processos de tempo real e a fila contendo os processos de usuários. Os processos de tempo real devem ser inseridos na fila seguindo a política de escalonamento FIFO (*First In First Out*, enquanto os processos de usuários são divididos em múltiplas filas a partir de suas prioridades apresentadas na inicialização.

As filas foram implementadas por uma classe composta por quatro listas e um valor representando a quantidade de processos atuais. As quatro filas são preenchidas baseadas nas prioridades, onde a fila *priority0* acomoda os processos de tempo real, com prioridade 0, a fila *priority1* com processos de usuários de prioridade 1, e assim por diante. O valor *max* da classe é inicializado com o valor 1000, o máximo de processos que podem estar em filas simultaneamente, e é subtraído por 1 sempre que um novo processo entra em uma fila.

A classe também possui as funções de retirar processos finalizados da fila, atualizar a posição de processos de usuários na fila e atualizar as prioridades de processos de usuários. Processos são retirados da fila quando o valor representando a quantidade de instruções

a serem executados chega a 0, fazendo com que a variável *max* seja incrementada. A atualização da posição é feita apenas para processos de usuários, pois não seguem a política FIFO. Desta forma, sempre que o processo em primeiro na fila executa um *quantum*, ele é movido para o fim da fila. Para evitar *starvation*, processos que estão a mais de 10 segundos sem serem executados, são movidos para a fila de prioridade superior. Isso ocorre apenas para processos com prioridade 2 e 3, visto que processos já nas filas de prioridade 0 e 1 não podem alterar suas prioridades.

3.4 Módulo de Memória

A memória principal do sistema operacional é representada por um conjunto de 1024 blocos contíguos, onde cada processo aloca apenas segmentos contíguos quando inserido em uma fila, e os blocos são liberados apenas quando o processo finaliza sua execução e é removido da fila. Os primeiros 64 blocos devem ser reservados apenas para processos de tempo real, enquanto os outros 960 são utilizados por processos de usuários.

A classe utilizada para o módulo de memória é inicializada por uma lista vazia de 1024 blocos. Sempre que o *dispatcher* verifica que um processo chegou no seu tempo de inicialização, o módulo de memória executa três passos: verifica se o tamanho total de blocos a ser alocado é maior que o total de blocos possíveis de serem alocados na memória, caso negativo verifica se a quantidade de blocos a serem alocados está disponível em espaço contíguo na memória, e por fim, caso seja possível, faz a alocação dos blocos para que o processo possa ser inicializado. Essas verificações são feitas respeitando o espaço de memória para cada tipo de processo.

3.5 Módulo de Recursos

O módulo de recursos tem como objetivo o tratamento de alocação e liberação de recursos de entrada e saída para os processos. Os recursos disponiveis neste pseudo-SO são: 1 scanner, 2 impressoras, 1 modem e 2 dispositivos SATA. Na implementação, possui uma classe *Resources* que tem como atributos os recursos citados acima. Todos os processos de usuários tem como possibilidade alocar um desses recursos e o *dispatcher* garante que cada recurso é alocado somente para um processo por vez, mantendo assim, a exclusão mútua dos mesmos.

3.6 Módulo de Arquivos

O sistema de arquivos inicializa contendo uma variável que armazena o tamanho do volume, uma lista representando o mapa de bits de blocos livres e uma lista para guardar dados de cada arquivo no volume contendo nome, posição no volume, tamanho do arquivo e qual processo o criou. As primeiras operações determinam segmentos considerados já

ocupados no volume, portanto não são verificados se existe espaço no volume ou arquivos com o mesmo nome, como nestas operações não é especificado quais processos a criaram a lista de dados considera o processo como -1, ou seja, desconhecido.

Em seguida são realizadas as operações de criação ou exclusão de arquivos, para isso são especificados qual processo realizara a operação, que tipo de operação irá realizar, nome do arquivo e seu tamanho caso queira criá-lo. Primeiro verifica se o processo que irá realizar a operação realmente existe procurando-o na lista de processos, caso tenha que deletar um arquivo a lista de arquivos é vasculhada para determinar se ele existe, então verifica se o processo que irá deletar é de tempo real ou foi responsável por criar o arquivo, se todas as condições forem passadas o mapa de bits é atualizado e o arquivo removido da lista. Na operação de criar um arquivo é verificado se o arquivo já existe no volume, caso não o sistema tenta encontrar no mapa de bits um espaço de blocos vazios contíguos do mesmo tamanho do arquivo, se encontrar o espaço no mapa de bits é atualizado e é adicionado na lista de arquivos os dados do arquivo adicionado.

4 Dificuldades Encontradas

Uma das maiores dificuldades encontrada foi a integração de todos os módulos no *dispatcher*. Como os módulos foram feitos separadamente, foi necessário que suas interfaces fossem implementadas de uma forma que possam ser executados pelo processo principal seguindo o mesmo padrão, como, por exemplo, a estrutura contendo os dados de cada processo.

Além disso, o controle do estado atual de cada processo também rendeu dificuldades, principalmente a divisão de processos não inicializados e processos já inseridos em filas. A forma como isso foi resolvido foi o uso de uma variável contendo a quantidade de processos que ainda precisam ser inicializados, e uma variável contendo a quantidade de processos já inseridos em alguma fila. Essas variáveis foram então utilizadas como condição de saída da estrutura de repetição principal do programa.

5 Funções de Cada Aluno

O trabalho foi dividido entre os integrantes por módulos, onde cada um escolheu livremente o módulo a ser implementado. Como um integrante do grupo não participou da implementação, um integrante se voluntariou a fazer dois módulos.

- Johannes: Implementação do gerenciamento de memória e gerenciamento das filas de prioridade.
- Vinícius: Implementação do sistema de arquivos.

- Matheus: Implementação do módulo de recursos, E/S.
- Igor: Implementação do módulo de processos.