Pseudo-SO Grupo 10

Kesley Kenny Vasques Guimarães, 18/0021231
Pedro Henrique de Brito Agnes, 18/0026305

Dep. Ciência da Computação - Universidade de Brasília (UnB)

1 Introdução

Para o projeto final da disciplina de Introdução a Sistemas Operacionais, foi desenvolvido um pseudo-sistema operacional para a simular o funcionamento de um sistema operacional ao ficar responsável por exercer todas as suas funções. O programa realiza a gerência de processos que são recebidos por meio de um arquivo, gerência de memória, de acordo com espaço máximo definido, gerência de arquivos, onde são disponibilizadas as funções de *create* e *delete* para a manipulação do espaço definido do disco de acordo com o segundo arquivo de entrada. Por fim, também é feita a gerência de recursos, que são limitados e a gerência das filas para o escalonamento dos processos.

2 Ferramentas/Linguagens Utilizadas

Para o desenvolvimento do projeto, foi utilizada a linguagem de programação *Groovy*, que é open source e utiliza o mesmo ambiente de execução que *Java*, a JVM (*Java virtual machine*). Para a execução do código, é necessário ter instalado o Apache Groovy, onde é garantido de rodar corretamente em versões 3.0.4 ou superiores, assim como a JVM 8 ou superior.

2.1 Preparação do Ambiente

A forma mais fácil de instalar o *Groovy* seria baixando o zip na página de download: `https://groovy.apache.org/download.html`, extraindo o conteúdo e adicionando ao *PATH* a pasta `bin` ou executando a partir do caminho completo. Já para a instalação da JVM, pode ser seguido o tutorial da própria página de download: `https://www.java.com/en/download/` (lembrando a partir de quais versões em que é garantido o correto funcionamento do programa). Após instalado, pode ser coferida a versão de ambos pelo seguinte comando no terminal:

```
groovy --version
```

2.2 Executando

Após o ambiente preparado, será possível executar o código a partir da **pasta raiz do projeto**, utilizando o seguinte comando no terminal:

```
groovy pso/dispatcher.groovy <arquivo_de_processos> <arquivo_de_arquivos>
```

Também foram disponibilizados arquivos de exemplo na pasta `sample` do projeto, o que permite a execução do seguinte comando:

```
groovy pso/dispatcher.groovy sample/*.txt
```

3 Fundamentação Teórica e Prática da Solução

Para o desenvolvimento do projeto, foi necessário utilizar todo o conhecimento adquirido na disciplina de Introdução a Sistemas Operacionais, assim como os conteúdos de disciplinas anteriores como Estrutura de Dados.

3.1 Visão Geral

O processo principal do pseudo-SO é o *dispatcher*, que é o despachador, responsável por obter a lista de processos e informações do sistema de arquivos, assim como instruções a serem executadas nele. Durante o início da execução, o processo vai criar um objeto do tipo `Manager`, que agrupa os módulos e as operações a serem utilizadas durante o gerenciamento realizado na execução, o que vai inicializar todos os módulos que precisam ser incializados. Também será inicializado o sistema de arquivos nesta etapa, com base nas informações obtidas no segundo arquivo de entrada.

Após obter todas as informações necessárias, o processo vai entrar em loop para despachar os processos com base no tempo de inicialização definido por cada um, assim mandando-os para a estrutura de filas. Após todos os processos finalizarem a execução, o loop é quebrado e serão computadas as instruções que cada um realizou no sistema de arquivos, mostrando o mapa de alocação do disco ao final.

3.2 Processos

O módulo de processos trata-se apenas da classe que vai definir as propriedades de cada um, assim como agrupara as operações realizadas neles. A estrutura dos processos pode ser vista a seguir:

```
class Processo {
    int pid
    int tempoInicio
    int prioridade
    int tempoUsado
    int offset
    int blocks
    int impressora
    int scanner
    int modem
    int sata
    int instrucaoAtual
}
```

Cada um dos processos criados durante a execução do programa seguirá a estrutura descrita acima, sendo um objeto do tipo `Processo`.

3.3 Filas

Para as filas, foi criada uma classe `Fila`, que vai definir a estrutura de dados de cada fila de processos, agrupando as operações possíveis de realizar, como a inserção de um processo e a remoção, seguindo o algoritmo FIFO. Também foi criada a classe `Escalonador`, que vai fazer o gerenciamento das filas usadas na execução, assim como atribuir o quantum definido aos processos para a posse da CPU.

O método de escalonamento definido no projeto foi o de múltiplas filas com realimentação, onde são usadas 6 filas no total. A primeira fila pode ser interpretada como uma fila global, que é a de processos prontos para executarem, que entram nela no momento em que o *dispatcher* despacha os processos para execução. Assim, o escalonador vai pegar o primeiro processo da fila e avaliar a sua prioridade para mandar para a fila de tempo real ou para a fila de usuário. Sempre que existirem processos na fila de tempo real, como eles não podem ser preemptados, eles assumem controle completo da CPU até finalizarem a sua execução por completo. Já os processos na fila de usuário podem ser preemptados e são classificados novamente entre mais 3 filas.

Todos os processos que chegam na fila de usuário têm a sua prioridade avaliada novamente para entrarem em uma das filas que seguem. Caso a prioridade seja 1, o processo será encaminhado para a fila de maior prioridade de usuário, caso seja 2, cairá na segunda maior prioridade e, por fim, se tiver prioridade 3, cairá na última fila, com a menor prioridade. A partir do momento em que os processos estão em uma das 3 filas de prioridades distintas, eles começam a ser executados de acordo com o algoritmo FIFO, onde é atribuído o quantum de 1 segundo em qualquer fila e caso o processo não tenha finalizado ainda, ele volta para o final da próxima fila em relação a prioridade. No momento que um processo chega na fila de menor prioridade, ele ficará lá até finalizar a sua execução, sendo mandado para o fim da fila toda vez que seu quantum é esgotado.

3.4 Memória

O módulo de gerenciamento de memória, `Memoria` é a unidade responsável por abstrair detalhes mais específicos da memória, assim como prover serviços de alocação e desalocação de blocos de memória para os processos. A memória disponível no programa é de 64 blocos para processos de tempo real e 960 blocos para processos de usuário. A memória é alocada para o processo no momento que ele é removido da fila de pronto, assim o gerenciador vai avaliar se há espaço disponível e avisar ao escalonador se foi possível realizar a alocação ou não, para assim o processo prosseguir para as filas seguintes ou voltar para o final da fila de prontos.

3.5 Recursos

Similarmente a memória, se um processo precisar de um recurso para a sua execução, o escalonador vai acionar o gerenciador de recurso, chamado `Recurso` para verificar a disponibilidade do mesmo e, caso esteja disponível, aloca o recurso e manda para uma das filas conforme a prioridade, caso contrário, volta para o final da fila de pronto. O gerenciador

de recursos vai disponibilizar serviços como a alocação dos recursos limitados aos processos, assim como a desalocação, e vai manter registro de qual processo está em posse que qual recurso, garantindo que apenas um processo possa usar um recurso de cada vez.

3.6 Arquivos

O módulo responsável pelo sistema de arquivos é o `SistemaArquivos`, que vai disponibilizar as funções de `create` e `delete` e vai imprimir a operação se obteve sucesso ou falha. A quantidade de armazenamento deisponível é definida no início da execução do pseudo-SO a partir dos arquivos de entrada, assim como os blocos iniciais ocupados e, com isso ele vai sempre avaliar se a operação solicitada é válida, se o arquivo a ser deletado existe, se o processo possui acesso ou até se há espaço suficiente para a criação de novos arquivos, seguindo o algoritmo *first-fit*, que apesar de não ideal, foi essencial para o melhor entendimento do porque não é mais usado.

Para a definição de um arquivo, foi criada a classe `Arquivo`, que basicamente vai armazenar as propriedades do arquivo de forma a facilitar a avaliação pelo `SistemaArquivos` sobre permissão ou nome. A classe é a seguinte:

```
class Arquivo {
    char nome
    int processoCriador
}
```

4 Dificuldades

Foram observadas algumas dificuldades durante a implementação do trabalho. Primeiramente, a dupla precisou entrar em um consenso de escolha da linguagem que seria utilizada. A escolha do groovy está relacionada com a familiaridade da sintaxe que ele possui com Java, uma linguagem que ambos dominavam e, que se aplicava bem no contexto de aplicação e na complexidade exigida pelo projeto. Ao longo do projeto foram encontrados outros desafios. O principal, a dificuldade da dupla em interligar as múltiplas funcionalidades que o processo executa. Essa dificuldade impactou em múltiplas refatorações, além de, um reestruturação inteira da arquitetura dos diretórios.

Dentre as várias dificuldades encontradas, também teve um problema no sistema de arquivos para a execução das operações disponibilizadas aos processos, no momento da identificação de acesso, para encontrar o processo criador do arquivo. Para solucionar o problema, foi criada uma classe `Arquivo` que agrupa as propriedades do arquivo, basicamente tendo um nome e o *pid* do processo criador. Assim, o mapa de alocação de disco foi definido como um array dessa estrutura `Arquivo` definida.

Sem precisar refletir muito, é possível concluir que a parte que apresentou os maiores desafios durante a implementação do projeto foi o gerenciador das filas com realimentação, que foi chamado de `Escalonador`. Cada fila foi instanciada como sendo do tipo *Fila*, que foi a estrutura de dados criada para esse objetivo, permitindo as operações de inserção no final e remoção do início, sendo todos os elementos objetos da classe `Processo`, que também foi criada pelo grupo para o trabalho. Esse módulo também teve que usar funcionalidades de muitos outros para a sua própria funcionalidade conforme especificações, sempre fazendo a verificação para encontrar a disponibilidade de recursos antes de mandar o processo para

execução, atribuindo um quantum justo evitando ao máximo o *starvation*, garantindo sempre que os processos serão executados em algum momento. Para a solução de cada problema dado neste tópico, foi necessário o uso de todo o conhecimento adquirido desde o início do curso, utilizando estratégias de mudança de prioridade, de escalonamento, entre muitos outros.

5 Papel de Cada Integrante do Grupo

Todos os integrantes do grupo ajudaram na implementação do pseudo-SO, onde as atividades foram divididas o mais igualmente possível entre os dois.

- O Kesley ficou responsável pela implementação do gerenciador de memória, assim como o gerenciador de recursos e parte do gerenciador de processos. Também foram feitas as integrações dos módulos desenvolvidos com o restante do código, onde deveriam avisar ao escalonador, por exemplo quando não foi possível alocar. Também ajudou na correção de bugs e melhorias do código no geral.
- O Pedro começou a implementação do projeto, criando a estrutura original do repositório. Assim, foi implementado boa parte da função principal, localizada no arquivo `dispatcher.groovy`, assim como o `Manager`, que foi definido como o módulo responsável por acessar todos os recursos do sistema operacional. O aluno também ficou responsável pela implementação do sistema de arquivos e o escalonador juntamente com o Kesley.

Referências

[1] Tanenbaum, A. S., Bos, W. *Sistemas Operacionais Modernos*. Person, São Paulo, 4° ed., 2016