UNIVERSIDADE FEDERAL DE SÃO JOÃO DEL-REI
CIÊNCIA DA COMPUTAÇÃO

Gerenciamento De Processos Em Um Sistema Operacional

Felipe Maxsuel Carvalho 182050047
Samuel de Souza Carvalho 182050048
Wagner Lancetti 182050040

Primeiro trabalho prático da disciplina de Sistemas Operacionais do curso de Ciência da Computação da Universidade Federal de São João del-Rei.

São João del-Rei
Novembro de 2020

Sumário

1 Introdução

Em dias atuais cada vez mais estamos inserindo dispositivos inteligentes que facilitam nossas vidas, entretanto com o passar do tempo amplia-se a nescessidade desses executarem mais processos ao mesmo tempo. Dessa forma várias soluções são propostas, como o aumento da memória do sistema, CPUs mais potentes e outras, mas principalmente levamos em consideração a melhoria no gerenciamento de processos, pelo qual o sistema operacional é responsável.

A multiprogramação consiste em fazer com mais processos ocupem a memória, permitindo que a CPU passe menos tempo ociosa, dando a impressão ao usuario que tudo está sendo executado ao mesmo tempo. Cada sistema adota melhorias para isso dependendo do seu objetivo, sendo assim podemos ver que a complexidade de melhorar ainda mais a multiprogramação é muito grande. Todavia, o nosso objetivo principal é entender como funciona esses sistemas operando no gerenciamento de processos de forma básica.

2 Problema Proposto

Este trabalho prático tem como objetivo entender melhor sobre gerenciamento de processos em um sistema operacional, colocando em prática alguns dos ensinamentos que foram passados em aula. Para isso, foi pedido a criação de um sistema que simula algumas das principais funções de um gerenciador de processos, nas quais criação de um processo, substituição do processo atual por outro processo, transição de estados de um processo, escalonamento e troca de contexto. Além disso, o sistema deve realizar a comunicação entre dois processos diferentes.

3 Implementação

Para a solução do problema foi feita a implementação utilizando a linguagem C. A execução do código baseia-se no arquivo /manager.c/, que controla a execução dos processos, as transições de estados e o escalonamento desses processos.

O processo /commander/ cria o processo manager, que será executado pelo arquivo /manager.c/ e envia comandos para serem interpretados por esse processo a cada segundo. A comunicação entre eles é feita através do pipe.

O processo /manager/ possui diversas estruturas de dados, as especificações delas serão discutidas em seções abaixo. Para implementar essas estruturas foram utilizados listas, filas e filas circulares. Listas foram utilizadas nas estruturas que não sofreram remoção constantes, como é o caso do arquivo de programa e da TabelaPcb. Filas foram utilizadas para os vetores que precisam fazer remoção apenas no começo, como é o caso do vetor de EstadoBloqueado, na qual a remoção acontece sempre no primeiro processo bloqueado. Já

filas circulares foi utilizado apenas em EstadosProntos, visto que a política adotada no escalonador foi de fila circular, fica mais simples percorrer esse vetor, não sendo necessário colocar o processo que foi executado no final da fila, visto que após passar para a próxima celula, imediatamente esse processo será o último.

Abaixo serão dicutidos, em detalhes, a implementação de cada funcionalidade do sistema.

3.1 Commander

O commander é um processo mais simples, ele inicialmente cria um pipe para poder se comunicar com o process manager. Ele recebe instruções pela entrada padrão e, usando o pipe, ele repassa esses comandos para o processes manager a cada um segundo.

O commander reconhece quatro tipos de comandos da entrada padrão, sendo eles Q, U, P e T. Cada um desses comandos possuem funcionalidades diferentes e elas são interpretadas e executadas pelo processes manager. O comando Q é responsável pelo fim da unidade de tempo, com isso ele lê a próxima instrução do arquivo do processo simulado, executa-a, e realiza o escalonamento, colocando um novo processo na CPU; o comando U desbloqueia o primeiro processo da fila de bloqueados; o comando P é responsável por imprimir o estado atual do sistema, para isso ele chama o reporter; por fim o comando T realiza a chamada do reporter e finaliza o simulador.

3.2 Process Mananger

O Process Mananger tem como principal função realizar todo o gerenciamento de processos do sistema, para isso ele utiliza de estruturas de dados e funções. As estruturas e funções serão descritas abaixo:

3.2.1 Estrutura de dados

O programa contém como principal estrutura de dados o **ProcessManager**, nela está incluída todas as estruturas nescessárias do programa. São elas:

- **CPU**: Essa estrutura fica responsável pela execução dos processos simulados. Para criação da estrutura de dados CPU foi criada uma struct que contém um inteiro, *valor_inteiro*, utilizado para armazenar o valor inteiro do processo simulado e modificá-lo quando receber instruções que indiquem modificações, como as instruções S, A e D que, respectivamente, indicam atualizar o valor da variável, somar a um outro inteiro e subtrair o valor atual com um outro valor inteiro. A CPU possui um inteiro *tempo* que representa o tempo do processo, sendo atualizado toda vez que o programa realiza o escalonamento; um inteiro *contador_de_programa* para armazenar a linha que o programa está; um ponteiro, *apontador*, que irá apontar

para o arquivo de programa do processo em execução; e, por fim, *Tempo_Atual* que é um inteiro representando quantas unidades de tempo a CPU já executou.

- **TabelaPcb**: Essa estrutura é utilizada para armazenar todos os processos. Quando um processo simulado executa a instrução F, um novo processo é criado, ele então é adicionado na tabela. Como essa estrutura irá guardar processos, ela precisa de dados que reconheçam um processo, sendo eles: O *id* do processo, que será arbitrário; *id_pai* que representa o ID do processo pai, sendo o ID do processo que executou a instrução F; *contador_de_programa* que armazena o valor referente ao número de instruções executadas naquele arquivo; *inteiro* que é a variável do inteira do processo simulado; *prioridade* do processo que também será um valor arbitrário; *estado* um inteiro que indica em qual estado o processo está, sendo eles executando(2), pronto(1) e bloqueado(0); *tempo_inicial* que indica em que unidade de tempo o processo começou a ser executado; *tempo_utilizado* que indica quantas unidades de tempo esse processo usou a CPU; *arquivo_do_programa* que aponta para o vetor que contém as instruções que serão executadas pelo processo; e, por fim, os apontadores *prox* e *anterior* do mesmo tipo da TabelaPcb, que constroem uma lista duplamente encadeada nessa estrutura.

- **Vetor**: Essa estrutura de dados irá converter o arquivo de entrada (do tipo *FILE*) para um vetor. Os campos dessas estrutura consistem em *instrução*, que irá guardar qual instrução o processo simulado irá executar; *valor* um inteiro que será utilizado em instruções que fazem modificações no inteiro do processo simulado; *str* uma stirng, para guardar o nome do arquivo quando receber uma instrução R; e um apontador *prox*, que faz desse vetor uma lista simplesmente encadaeada.

- **ApontaTabela**: Essa estrutura de dados é usada nos vetores *EstadoPronto*, *EstadoBloqueado* e *EstadoExecutando* dentro da estrutura ProcessManager. Ela consiste em três ponteiros: *apontador* que irá apontar para um processo na TabelaPcb e os ponteiros *prox* e *anterior*, que fazem essa estrutura virar uma fila duplamente encadeada.

- **EstadoPronto**: Essa estrutura é do tipo ApontaTabela, ela é utilizada para guardar os processos que estão prontos para serem executados, e aponta para eles na TabelaPcb. A implementação dessa estrutura foi feita usando um fila circular, então esse apontador fica se movimentando junto com o ponteiro do EstadoExecutando, ou seja, EstadoPronto também irá apontar para o processo que está na CPU no momento.

- **EstadoExecutando**: Essa estrutura é do tipo ApontaTabela, ela é utilizada para apontar na TabelaPcb qual processo está sendo executado no momento. Ele também

é responsável em passar para a CPU qual é o processo que será executado, além de receber os dados da CPU e atualizá-los na TabelaPcb.

- **EstadoBloqueado**: Essa estrutura é do tipo ApontaTabela, ela é utilizada indicar quais processos estão bloqueados na TabelaPcb, além de apontar para eles. A implementação dessa estrutura foi feita utilizando a ideia de FIFO (*First In First Out*), ou seja, o primeiro processo a ser bloqueado, será o primeiro que será desbloqueado.

3.2.2 Funções

As funções presentes no código estão comentadas, facilitando também a compreensão delas.

- **Guarda_Arquivo**: Essa função tem a finalidade de ler o arquivo enviado pra ela e guardar seus dados dentro da estrutura Vetor. Inicialmente é lido a instrução e seu valor é guardado no campo *Instrucao* da estrutura. Após isso é feito a verificação se existe outro dado para ser lido na frente, caso seja, esse dado é guardado de acordo com seu tipo, se for um inteiro guarda no campo inteiro, ou se for o nome de um arquivo guarda na string. Após isso, se ainda tiver algo para ser lido, cria-se uma lista com essas novas leituras.

- **Inicializa_Dados**: Essa função inicializa os dados da estrutura *ProcessManager*. Como o primeiro processo é estático, lido do arquivo Init, os dados são inicializados já com esse processo adicionado.

  As estruturas internas são inicializadas e, após isso, a TabelaPcb guarda o primeiro processo, Init, na primeira celula; a estrutura EstadoBloqueado é inicializada vazia; a estrutura EstadoExecutando aponta para esse processo do arquivo Init, que será convertido em vetor pela função *Guarda_Arquivo*; a estrutura EstadoPronto inicializa com o primeiro processo; e a CPU recebe os dados do EstadoExecutando;

- **Criacao_de_processo:** O objetivo dessa função é a criação de um novo processo, idêntico ao processo que está criando. Essa função é chamada decorrente da instrução F, executada por algum processo simulado.

  Inicialmente essa instrução verifica se existem mais do que um procesos na TabelaPcb. Caso exista apenas um processo, pode-se usar o próprio ponteiro da TabelaPcb para criação de um novo processo; do contrário é necessário utilizar um ponteiro auxiliar para chegar até o último processo adicionado na tabela. O novo processo adicionado é, praticamente, idêntico ao que executou a instrução F, diferenciando-se nos seguintes campos:

- **Arquivo_do_Programa:** É o mesmo arquivo, mas o processo novo irá executar na próxima instrução após a chamada, enquanto que o processo que executou a instrução F dará um salto de n linhas pra frente.
- **Id:** O ID do novo processo é arbitrário, e será diferente do processo que executou a instrução.
- **Id_pai:** O processo novo precisa guardar o ID do processo o criou.
- **Tempo_utilizado:** O tempo de CPU do novo processo será 0, visto que ele ainda não foi escalonado nenhuma vez.
- **Tempo_inicial:** O tempo inicial é inicializado com o tempo que atual da CPU, ou seja, em qual unidade de tempo o sistema está.
- **Estado:** O estado do novo processo será Pronto(1), enquanto que o processo que executou a instrução F é Executando(2).

Após adicionar o processo na TabelaPcb, é necessário colocar o processo na lista de prontos. Para isso é verificado se a lista de prontos possui um ou mais processos prontos. Caso exista só um, basta colocar o processo na frente do que já existe e respeitar a ordem de fila circula. Caso exista /n/ processos, o novo processo será colocado entre o processo atual e o último processo lido, respeitando a implementação de fila circular, e também mantendo a ordem de execução dos processos.

- **Troca_de_Imagem:** Essa função tem a finalidade de trocar o arquivo atual do processo por outro arquivo, sendo seu nome lido após a instrução R.

  A função recebe o novo arquivo do processo como parâmetro e o que estava sendo executado é retirado da CPU. Esse arquivo é convertido em vetor pela função Guarda_Arquivo e colocado na CPU, para que ela possa atualizar o processo na TabelaPcb.

- **Gerenciamento_de_transicao:** Essa função tem a finalidade de realizar as transições de estados dos processos, de executando pra bloqueado, de bloqueado para pronto e de executando para terminado. Ou seja, essa função atende a três instruções que o sistema pode enviar, são elas:

  - **B:** A instrução B vem de um processo simulado, sua funcionalidade é que o processo em execução seja bloqueado.

    Para realizar esse bloqueio é necessário atualizar a estrutura de EstadoBloqueado, EstadoPronto e também atualizar seu estado na TabelaPcb. Como EstadoBloqueado implementa uma fila simples (FIFO), basta colocar esse processo no fim da fila, ou na cabeça, caso não exista outro processo bloqueado; já para atualizar EstadoPronto é necessário retirar esse processo e deixar a estrutura de acordo com uma fila circular.

- **E:** A instrução E vem de um processo simulado, sua funcionalidade é terminar o processo que está em execução.

  Para finalizar o processo é necessário retirar ele da estrutura EstadoPronto, bem como da TabelaPcb. Como dito na instrução B, para retirar ele de EstadoPronto é necessário manter a estrutura como uma fila circular, então os processos que ainda ficarem lá na estrutura devem formar um círculo; já para remover esse processo da TabelaPcb, basta utilizar o apontador do EstadoExecutando que consegue-se o acesso à esse processo na TabelaPcb, dessa forma, basta atualizar a tabela mantendo a estrutura como uma lista duplamente encadeada.

  É importante lembrar, que se remover todos os processos da TabelaPcb o programa não poderá realizar mais nenhuma instrução do tipo Q e U, vindo do commander, visto que elas executam em cima de processos. Apenas serão aceitas instruções do tipo P e T.

- **U:** Essa instrução, vinda do /commander/, tem a finalidade de desbloquear o primeiro processo da fila de bloqueados .

  Para desbloquear um processo, primeiramente é necessário que haja um processo para ser desbloqueado, do contrário, irá retornar uma mensagem de error para o usuário. Considerando que haja um processo, é necessário atualizar a estrutura EstadoBloqueado, EstadoPronto além de alterar o estado do processo na TabelaPcb. Como EstadoBloqueado implementa uma fila, basta remover o primeiro processo e atualizar o novo primeiro processo bloqueado; já para atualizar EstadoPronto é necessário adicionar esse processo de tal forma que a fila circular seja mantida; para atualizar a TabelaPcb basta usar o ponteiro EstadoPronto -> anterior -> apontador, visto que o processo estará na celula anterior ao atual processo executando, e atualizar seu estado para Pronto(1).

- **Troca_de_contexto:** Essa função tem a finalidade de guardar os dados da CPU, atualizados, na TabelaPcb do processo que está em execução, além de colocar um novo processo dentro da CPU.

  Inicialmente ela verifica se a CPU não leu uma instrução do tipo E, ou seja, terminou o processo que estava sendo executado, visto que não há motivos de atualizar os dados de um processo que não existe mais. Em seguida ela verifica se há outro processo pronto para ser executado, caso haja, esse processo é colocado na CPU para execução, do contrário o programa não irá escalonar nenhum outro processo, manterendo aquele que já estava executando.

- **Escalonador:** Essa função tem a funcionaliade de verificar os processos que ainda podem ser executados, caso haja algum, ele chama a função Troca_de_contexo para trocar a execução e atualizar os valores, do contrário é feito a verificação se existe

algum processo bloqueado e, caso houver, é necessário aguardar que esse processo seja desbloqueado para ser executado. Além dessas verificações, o escalonador também verifica se ainda existem processos no sistema, caso não exista é enviada uma mensagem pro usuário e, a partir dai, o programa só irá executar instruções P e T, vindas do /commander/, sem enviar mensagem de error pro usuário.

Essa função também atualiza os tempos dentro da CPU, tanto o tempo total da CPU, o contador de programa do processo e o tempo que o processo passou dentro da CPU.

- **Reportar:** Essa função tem a finalidade mostrar o estado do sistema. Ela vai mostrar todos os processos que estão prontos, todos que estão bloqueados e o processo que está atualmente na CPU. Caso não exista nenhum processo nessas estruturas, as mensagens serão para indicar isso.

4 Resultados e Discussões

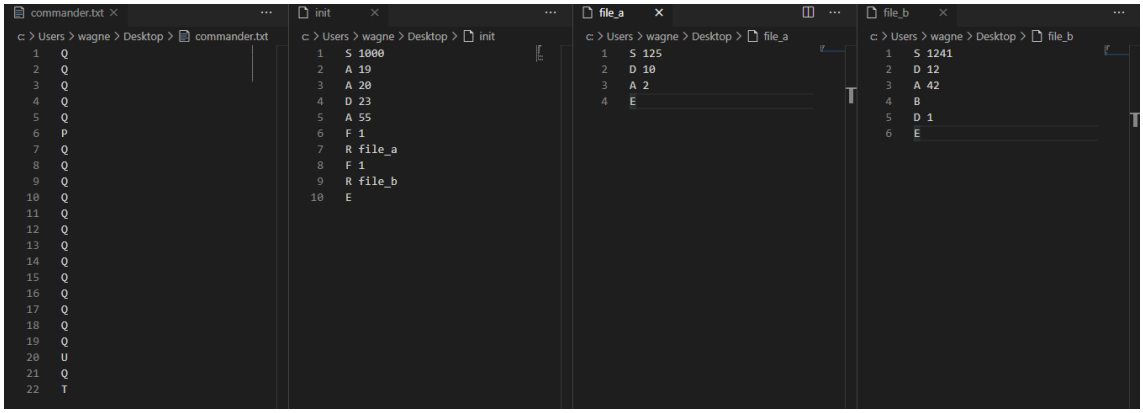

Figura 1: Foto dos Arquivos testados.

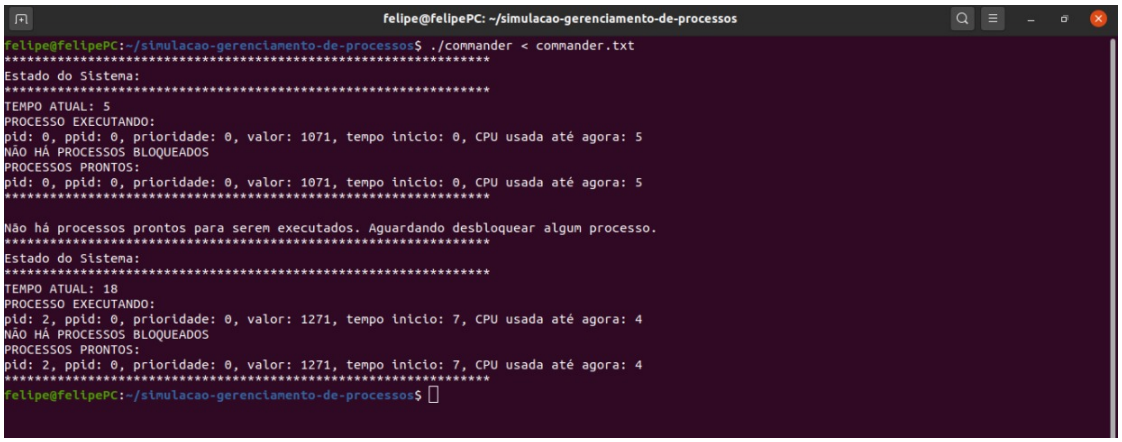

Figura 2: Foto reprodução dos testes.

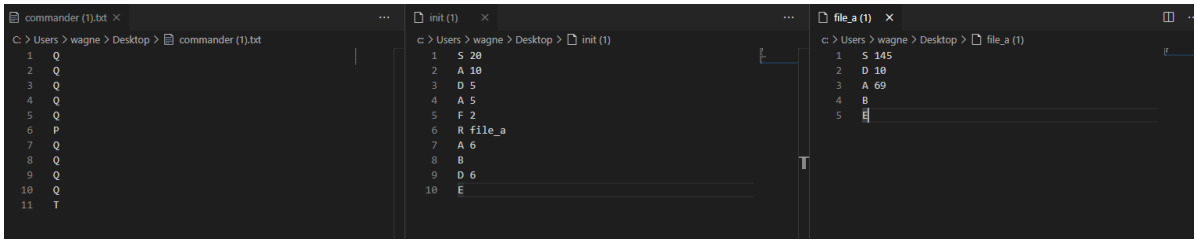

Figura 3: Foto dos arquivos de teste 2.

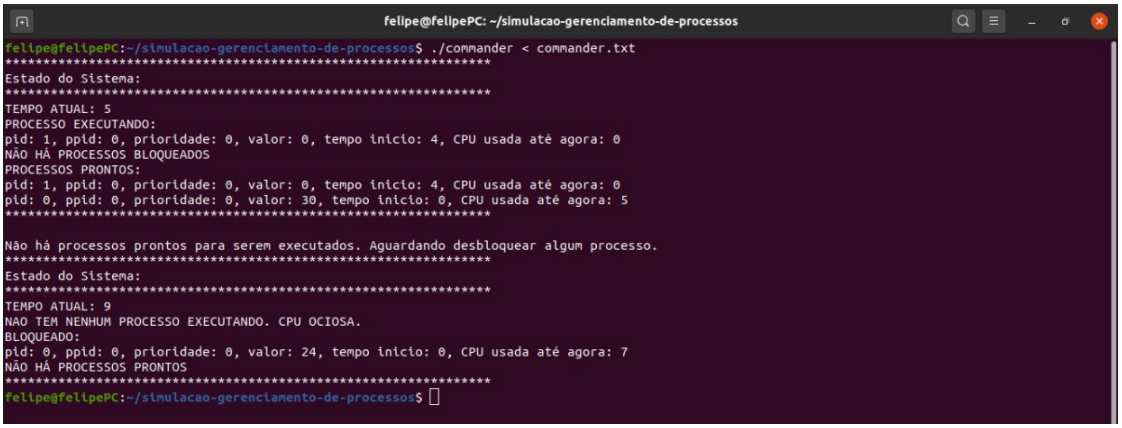

Figura 4: Foto reprodução dos testes 2.

Como pôde ver nas execuções 1 e 2, os resultados foram dentro do esperado. Note que tanto na execução 1 quanto na execução 2 aparecem mensagens de erro, isso acontece pois o programa está recebendo uma instruções Q sem ter processo para ser executado. Análogo à isso, caso o programa receba instruções do tipo U sem ter processos para serem desbloqueados, outra mensagem de error irá aparecer. Esses erros são apenas mensagens para informar o usuário que a instrução enviada não é a correta, isso não afeta a execução geral do programa.

5 Conclusão

A implementação do trabalho não foi um grande problema e os resultados ficaram dentro do esperado. Com a implementação desse sistema pôde-se ter uma ideia do funcionamento de um gerenciador de processos simples e, como a implementação deu-se exclusivamente por ponteiros, pôde-se exercitar bastante conteúdos aprendidos ao longo do curso. Ademais, pode-se visualizar a transição entre estados de processo, mostrada em aula, entre as filas de prontos, bloqueados e executando, além de entender o funcionamento da CPU em um gerenciador de processos.

A principal dificuldade encontrada na implementação desse código foram os ponteiros, como foi feito usando, quase exclusivamente, listas e filas a quantidade de ponteiros para implementar as funções foi grande e não se perder no meio desses ponteiros tornou-se difícil.

Além disso, a implementação da criação e comunicação entre os processos, utilizando chamadas de sistemas, foi um grande empecilho na implementação.

Referências

[1] Andrew S. Tanenbaum. Sistemas Operacionais Modernos - 3º Edição. Pearson 2010.

[2] Materiais postados no Campus Virtual da disciplina de Sistemas Operacionais.