TRABALHO 1 - ESCALONADORES

Matheus Caldas

1312760

Este primeiro trabalho de sistemas da computação consistia em criar e saber trabalhar com dois escalonadores. Um seria por prioridade dinâmica e outro por Round Robin, ou seja, divisão de tempo igual entre os processos. Optei por criar dois programas separados.

De comum eles possuem funções básicas de exibição e os handlers de sinais que servem para tratar simulações de I/O. As simulações serão explicadas melhor mais a frente. A informação de cada processo também é bastante semelhante, a diferença é que no caso do escalonamento por prioridade o processo obviamente tem um campo para indicar qual a sua prioridade.

A respeito do **interpretador**, optei por fazer simplesmente uma função que lê um arquivo de configuração que vai indicar quais programas que devem ser executados. No caso, foram utilizamos os programas: “programa1”, “programa2”, “progrma3”, “programa4”, “programa5”, “programa6” e “programa7”.

Claramente, como o arquivo de configuração é diferente para cada tipo de escalonamento os interpretadores são diferentes, embora eles façam a mesma coisa - Leiam as configurações e montem um vetor de processos a serem executados -.

Aqui vale ressaltar que optei trabalhar com vetor pela simplicidade, e resolvi os problemas de **filas (“first in, first out”)** utilizando variáveis que indicavam a prioridade e/ou número de vezes que aquele já foi executado. Assim foi possível saber a sua “posição” na fila. O ponto negativo é que era preciso iterar sobre o vetor várias vezes, mas como eram apenas 7 processos, o custo disso era bem baixo em comparação a fazer toda implementação de uma fila propriamente dita.

Os programas foram criados de forma a serem heterogêneos, alguns com grandes loops com operações, outros com escritas em arquivo e outros que executam rapidamente. Isso foi pensado para simular os casos onde o processo ganha um tempo de execução (no caso 2segundos) e após esse tempo ele ainda não terminou sua execução. Dessa forma, ele deve ir pra fila de prontos e após ser chamado novamente pelo escalonador continuar o seu processo.

A maior dificuldade encontrada foi a respeito da **simulação de I/O**. As escritas em arquivo serviam apenas para “gastar tempo de execução”, a simulação na verdade foi feita nos programas “programa5” e “programa7” onde eles enviavam um sinal ao seu pai (escalonador) indicando que entraram em modo I/O. Enquanto isso, um handler no pai, tratava essa notificação alterando o valor de uma flag. Dessa forma era possível saber se o programa entrou em simulação de I/O e quando ele saiu. Quando entrava em I/O, ele ia para fila dos processos em espera, quando saia, ia para fila dos processos em pronto.

Dessa maneira, mesmo que um processo tivesse 2 segundos para executar, ele podia entrar em I/O em por exemplo 1 segundo e passava o controle pro escalonador que podia já botar outro processo executando em seu lugar, fazendo máximo uso da CPU.

Inicialmente foi pensado em dar um sleep de 2 segundos no escalonador para depois ele ver se o processo tinha terminado, finalizado com erro ou se tinha se encerrado corretamente, saindo portanto da fila de prontos. O grande problema é que ele dormindo não receberia o sinal que a simulação de I/O foi iniciada. Assim, para resolver esse problema optei por fazer vários sleeps em micro segundos, dessa forma o pai acordaria e veria se chegou alguma notificação de entrada de I/O. Se não chegasse nada até o final, ele faria verificação do que ocorreu após os 2 segundos normalmente como previsto inicialmente.

Os dados de saída de ambos os escalonadores são basicamente iguais, sendo exibidos na saída do terminal. A cada 2 segundos (ou até entrar em I/O), o escalonador imprime quem são os processos em espera, quem está executando, quais são os processos prontos na fila de execução, o número de vezes que foram executados e no caso do de prioridade, qual a sua prioridade.

Dessa forma, foi possível debugar e entender exatamente qual a ordem dos processos, quem estava executando primeiro e quem deveria ser o próximo processo a ser executado.

Para os escalonadores decidirem quais seriam os próximos processos, eles chamavam funções auxiliares. No caso do Round Robin, as funções verificavam quantas vezes cada um já tinha sido executado e operava de forma a dar “chances” iguais de forma circular.

No caso do de prioridade, ele verificava a prioridade e após uma execução sua prioridade caia, garantindo assim uma prioridade dinâmica dos processos. Além da prioridade, caso houvesse empate, o número de execuções de cada um era comparado, garantindo uma divisão justa.

Para **testar as execuções**, primeiro é necessário (caso ainda não estejam), compilar os 7 programas. Em seguida, compilar os escalonadores e executá-los. Ao que parece as saídas foram corretas e o programa executou corretamente.

No arquivo saidaPrioridade.txt e saidaRound.txt estão as saídas exibidas no terminal após a execução dos respectivos escalonadores. Os arquivos gerados após execução file1 e file7 são arquivos gerados para “gastar tempo de execução”, e devem ser ignorados.