Relatório de implementação - Escalonadores de processos

Higor Celante [1], **Lucas de Santana Rocha**[1], **Brendow Paolillo Castro Isidoro**[1]

[1] Departamento de Computação – Universidade Tecnológica Federal do Paraná (UTFPR)
Caixa Postal 87301-899 – Campo Mourão – PR – Brazil

brendowpaolillo@gmail.com, {higorcelante, lucasrocha.2017}@alunos.utfpr.edu.br

Abstract. *The following document presents the implementation report of a process scheduler manager algorithm. This text explains how the implementation of three algorithms used to schedule processes inside the CPU, Round Robin, Shortest Job First (SJF) and the Dynamic Priority Scheduler were implemented.*

Resumo. *O documento que se segue apresenta o relatório de implementação de um algoritmo gerenciador de escalonadores de processo. É explicado nesse texto como foi realizada a implementação de três algoritmos utilizados para escalonar processos dentro da CPU, Round Robin, SJF(Shortest Job First) e o Escalonador com prioridade dinâmica.*

1. Introdução

Escalonadores de processos são algoritmos que otimizam o desempenho do processador, sem o auxilio desses programas não seria possível alcançar o desempenho que os computadores atuais possuem. Em primeiro lugar é apresentado o gerenciador dos escalonadores na seção 2, em seguida é explicado a implementação do escalonador Round Robin na seção 3. Na seção 5 é discutido sobre o escalonador Shortest Job First, e por fim dissertamos o funcionamento do escalonador com prioridade dinâmica.

2. Gerenciador dos Escalonadores

O gerenciador funciona juntamente dos escalonadores, fazendo o tratamento das filas de prontos, bloqueados, finalizados e em execução. Verificando os estados de cada processo e definindo o fluxo dos escalonadores.

3. Round Robin

A classe 'RR' que se encontra no arquivo 'src/schedulers/RR.py' é responsável por fazer alterações apenas no bloco de controle do processo. Todo o fluxo de execução referente ao Round Robin se encontra no arquivo 'Gerenciador.py'. A seguir é descrito o fluxo de execução do laço referente ao Round Robin.

3.1. Entrada de Processos Bloqueados

Inicialmente é feito a checagem do tamanho da lista de bloqueados, caso seu tamanho seja maior que '0', é checado em cada BCP na lista de bloqueados seu 'timeBlockRemain', que é responsável por controlar o tempo que o processo fica na lista de bloqueados. Caso 'timeBlockRemain' seja igual a '0' o BCP é retirado da fila de bloqueados e colocado na lista de prontos.

3.2. Entrada de Processos Não Criados

O método **criaListaProntos()** é responsável por criar os processos no seu devido *time stamp* de entrada. Cada iteração do laço é referente à 1 *time stamp*, sendo assim o método **criaListaProntos()** é chamado em toda iteração para checar se ainda há processos à serem criados.

3.3. Consumo de Processos

Se a lista de prontos for maior que '0', a execução chama o método **updateWaitingTime()** da fila de prontos, que atualiza o tempo de espera de todos os processos, exceto o processo da vez que será consumido. Para atualizar o BCP do processo da vez é chamado o método **RR.updateBCPQt(processo)** que consome '1' quantum do total da execução do processo e atualiza seu estado. Em seguida é chamado o método **checkStatus()** que checa o estado do processo, caso tenha terminado sua execução é retirado da lista de prontos e colocado na lista de terminados, retornando a lista de terminados para o **Gerenciador**. Caso o processo não tenha terminado é checado se há saída para **I/O**, se houver o **timeBlockRemain** é atualizado e o processo vai para a fila de bloqueados e é retirado da fila de prontos. Se não houver saída para **I/O** e o processo já tiver consumido todo o seu quantum, o estado do processo é alterado para '0' (pronto) e o processo passa a vez.

3.4. Ociosidade

Se não houver processos à serem consumidos, é comparado o tamanho da lista de terminados com a quantidade total de processos. Se os valores forem iguais a execução é terminada, se não, a execução continua.

3.5. Atualização da Fila de Bloqueados

Se houver processos na fila de bloqueados, o **timeBlockRemain** de cada processo é decrementado. E por fim o *time stamp* é incrementado.

4. Shortest Job First

O Shortest Job First (SJF) é implementado no arquivo "src/schedulers/SJF.py", no qual faz o tratamento do BCP do processo que está sendo executado. A classe também é responsável pelo método que seleciona o processo mais apto para a execução (com menor burst time).

4.1. Fluxo principal

O fluxo principal é feito pelo gerenciador, no qual trata as filas necessárias para a execução do SJF.

4.1.1. Lista de prontos

A lista de prontos é criada na chamada da função "self.criaListaProntos()", no qual adiciona todos os processos que são criados no timestamp atual.

4.1.2. Seleção do processo

Para a seleção do processo, verifica se há algo na lista de prontos e se há um processo executando. O método do SJF, "SJF.selectProc()", é realizado na lista de prontos retornando o index e o burst time do processo mais apto da lista de prontos. Caso haja um processo em execução, será comparado o burst time com o processo mais apto, e será trocado caso seja maior que o selecionado. Caso não haja um processo, o processo mais apto é enviado para a execução.

4.1.3. Execução

A execução verifica o estado do processo, se for 0 ou 1, ele realizará a execução do SJF. Caso seja -1 ele será adicionado a fila de bloqueados. Caso seja 2, o processo é enviado para a fila de finalizados.

4.1.4. Bloqueio

A fila de bloqueio é percorrida todo ciclo de CPU, verificando se algum processo chegou ao final do bloqueio, movendo ele pra lista de prontos, ou se ele possui algum tempo de bloqueio restante para ser consumido.

4.1.5. Finalização

A execução do escalonador será terminada ao verificar se a quantidade de processos na fila de finalizados é a mesma da quantidade total de processos inicializados.

5. Prioridade Dinâmica

O último algoritmo escalonador de processos implementado é o escalonador de processos com prioridade dinâmica. Esse algoritmo foi implementado utilizando duas filas com prioridades distintas, fila A e fila B, e uma fila de Bloqueados onde são deixados os processos que saem do processador para realizar operações de I/O. Na Fila A são colocados os processos recém chegados no escalonador e que não conseguiram utilizar todo o tempo do processamento anteriormente. Já na fila B são colocados os processos que na última execução utilizaram todo tempo disponível no processador, nessa fila, a ordem de execução é dada conforme o algoritmo Round Robin.

5.1. Fluxo de Execução

Todo fluxo de execução do escalonador com prioridade dinâmica é realizado dentro da função exec_loop na classe Manager. Em primeiro lugar, é verificado se algum processo acabou de ser criado e incluído na lista de prontos, caso tenha sido criado é realizado a reordenação da fila A por prioridade dentre os processos. A próxima operação realizada é a verificação se algum processo saiu da lista de bloqueados e foi para a lista A, a seguir é realizada a mesma operação que é feita quando um processo chega no escalonador.

A seguir temos que escalonar qual processo irá assumir um lugar no processador, para isso temos que seguir algumas regras: a fila A tem prioridade de execução sobre a fila B, ou seja, processos da Fila B só executarão caso não haja processos remanescentes na Fila A.

A próxima verificação a ser realizada é referente a conferir se o processo da vez deverá sair do processador para realizar uma operação de I/O. Caso o processo deva sair pra I/O, o processo é bloqueado e as informações do seu bcp são atualizadas. Caso contrário é realizada a execução do processo e também é realizada a verificação se um processo, após o processamento, acabou toda sua execução ou se ainda tem ciclos para executar.

Por fim, após um processo ter utilizado todo seu quantum em execução, o processo é direcionado para a Fila B onde espera pela sua próxima vez de execução.

6. Visualização

Ao final da execução o programa imprime a execução dos processos e gera uma imagem com o diagrama de Gantt, com o nome aleatório, de cada escalonador.

7. Conclusão

Escalonadores são algoritmos que otimizam o desempenho do processador, por conseguirem tratar processos simultaneamente, evitando que a CPU entre em ócio ou bloqueio devido ao estado do processo.
O objetivo deste projeto foi implementar os algoritmos de Shortest Job First, Round Robin e Prioridade Dinâmica.
Os algoritmos foram desenvolvidos e testados com processos estabelecidos na descrição do projeto e outros definidos durante o desenvolvimento. Afim de verificar todos os casos de falhas.

Referências

[1] Gate Vidyalay: Turn Around Time — Response Time — Waiting Time
https://www.gatevidyalay.com/turn-around-time-response-time-waiting-time/ pages

[2] Wikipedia: Escalonamento de Processos
https://pt.wikipedia.org/wiki/Escalonamento$_d e_p rocessos pages$

[3] GeeksForGeeks: CPU Scheduling in Operating Systems
https://www.geeksforgeeks.org/cpu-scheduling-in-operating-systems/ pages

[4] Alex Coletta: Escalonamento de Processos
https://alexcoletta.eng.br/artigos/escalonamento-de-processos/ pages