Simulação de Algoritmos de

Escalonamento

Gabriel de Souza Ferreira – 17250447
Jean Marcelo Mira Junior – 16102369

Resumo. *O presente trabalho da disciplina de sistemas operacionais tem por finalidade explorar o escalonamento de um conjunto de processos/threads através do uso de algoritmos de escalonamento de processos conhecidos na literatura.*

Palavras chave: threads, simulação, cpp.

Introdução

Esse trabalho tem como objetivo o desenvolvimento de um programa capaz de realizar o escalonamento de um conjunto de processos, utilizando diferentes tipos de prioridade de execução conhecidos pela literatura. Sendo assim, foram criadas três classes para estruturação de dados: Entrada, Processo e Escalonamento. A classe Entrada é responsável por fazer a leitura de um arquivo .txt com parâmetros de entrada e armazená-los dentro de variáveis que serão utilizadas no programa. A classe Processo armazena os dados de cada processo. Já a classe Escalonamento possui os diferentes métodos de escalonamento que serão executados.

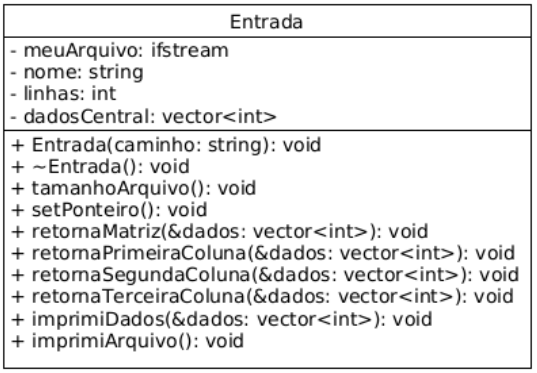

Figura 1: Diagrama UML classe Entrada.

Discussão

A seguir será demonstrado a lógica por trás de cada simulação de escalonamento, apresentando os passos e a conclusão.

Classe Entrada - explorando entrada.h e entrada.cpp:

Esta classe é responsável por fazer toda análise de entrada de dados através de arquivos .txt que contém os dados dos processos separados por linha. Aprofundando na funcionalidade desta classe simples podemos verificar que ela tem como função ler, identificar e retornar para o usuário os valores de tempo de criação, tempo de duração e prioridade estática no momento de execução.

Classe Processo - Explorando processo.h e processo.cpp:

Esta classe é responsável por armazenar e organizar todos os parâmetros iniciais de cada processo, além de disponibilizar algumas variáveis para o controle do mesmo. Os parâmetros iniciais são utilizados para tomada de decisão no momento em que é feito o escalonamento dos processos pela classe "Escalonamento". As variáveis são utilizadas para o controle dos processos e para armazenar os valores de saída apresentados no programa. Dentre as variáveis temos, por exemplo, a variável "estado" que memoriza o estado atual para cada processo e todos os estados durante a existência do processo. Outros tipos de variáveis desta classe são as que contêm os valores de saída, como: tempo transcorrido; tempo médio de espera e número de trocas.

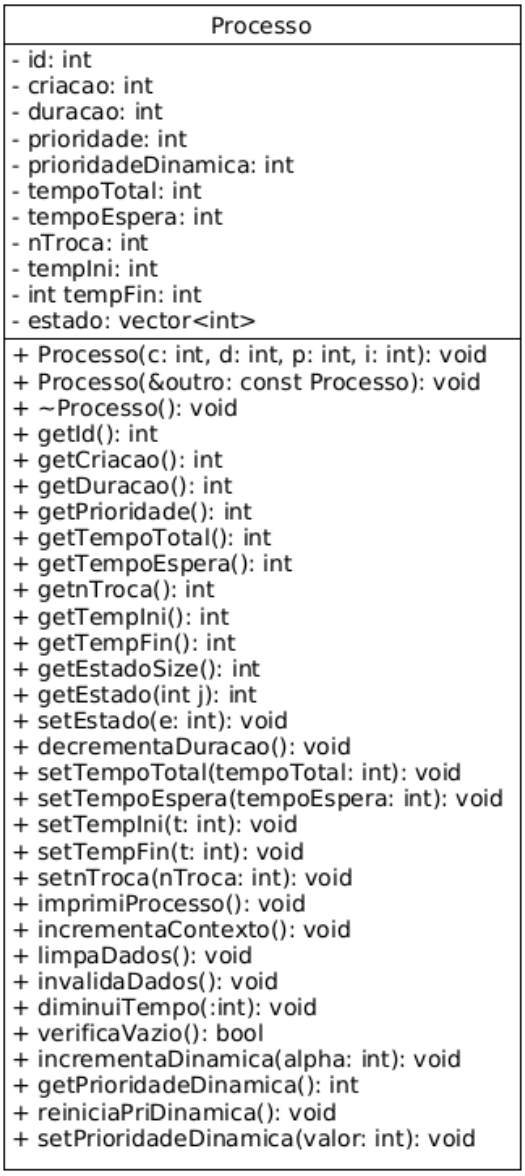

Figura 2: Diagrama UML classe Entrada.

Classe Escalonamento - Explorando escalonamento.h e escalonamento.cpp:

Esta classe é responsável por fazer os algoritmos de escalonamento, contendo dentro dela um vetor de processos no qual guarda todos os processos passados pelo usuário por um arquivo de texto. A seguir será explicado a lógica utilizada para resolver cada método de escalonamento, figura 3.

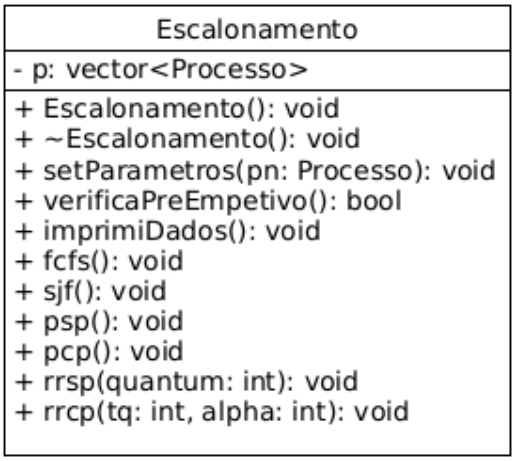

Figura 3: Diagrama UML classe Entrada.

A função void fcfs() diz respeito ao método de escalonamento FCFS (First Come, First Served), o conceito por trás deste método é baseado no tempo de criação do processo. Resumindo, os processos executam do início até o fim sem interrupções tendo como critério de execução o momento em que o processo foi criado, ou seja, os processos com menor tempo de criação executam primeiro. Para gerar essa análise foi feito um código de ordenação dos processos baseado no tempo de criação, para achar o tempo de espera e o tempo de execução total baseado que os processos já estão devidamente ordenados utilizamos a duração do processo anterior, e a sua própria duração. Logo após preencher os dados de tempo de espera e tempo total de execução o programa adiciona para cada processo em seu vetor de estados o estado em que o processo está durante a execução, tendo por finalidade este vetor de estados a impressão na tela do diagrama de tempo de cada execução.

A função void sjf() diz respeito ao método de escalonamento Shortest Job First, o conceito por trás deste método é baseado no tempo de criação adicionando o fator de tempo de duração. Se um processo for criado ao mesmo tempo que o outro ou seja se o processo coexistir ao mesmo tempo que outro deve ser executado primeiro aquele que tiver menos tempo de execução. O programa executa do começo ao fim sem interrupções, portanto é não preemptivo. Para gerar essa análise foi feito um código de ordenação dos processos baseado no tempo de criação, depois é ordenado os processos com base na criação comparando o tempo de duração. Como no processo anterior, para gerar o tempo de espera e tempo total de execução baseando se que os processos já estão devidamente pode se ordenar utilizamos a duração do processo anterior, e a sua própria duração. Logo após preencher os dados de tempo de espera e

tempo total de execução, o programa adiciona para cada processo em seu vetor de estados que irá imprimir na tela do diagrama de tempo de cada execução.

A função void psp() escalona os processos baseada no método de escalonamento Por prioridade sem preempção onde a prioridade de execução do processo é a própria prioridade pré-definida para cada processo, ou seja, no momento que for feito o escalonamento, o processo que possui a maior prioridade será o que vai ser executado primeiro. O "sem preempção" diz respeito a troca de contexto após o início do processo, isso significa que, assim que iniciado, o processo não irá fazer a troca de contexto até sua finalização.

A função void Escalonamento::pcp() refere-se ao escalonamento com prioridade e com preempção por prioridade ou seja o critério para execução é a prioridade estática do processo a partir da criação do mesmo, dependendo da prioridade o processo tem direito a interromper outros processos. O processo para desenvolver o escalonamento é parecido com outros desenvolvidos, o grande diferencial desta função é que a análise é feita baseando se no tempo total que os processo tem,e a partir disso verificasse a cada instante de tempo qual processo tem prioridade deste modo podendo aumentar o número de contextos dos processos que deixaram de executar porque outro processo detém maior prioridade naquele instante de tempo, além disso mudar o processo que executa.

A função void rrsp() aborda os conceitos do escalonamento pelo método Round-Robin com quantum = 2s, sem prioridade. Tem como princípio uma fila circular, onde os processos são adicionados à fila de execução com base no tempo de criação e executam com o tempo "quantum" de 2 segundo por processo. Inicialmente foi ordenado os processos por ordem de criação e depois calculado o tempo máximo de execução de todos os processos, a partir disso se desenvolveu uma lógica que coloca os arquivos em uma fila assim que eles são criados e executa de 2 em 2 segundos aumentando o número de trocas de contexto conforme ele ainda tem a possibilidade de executar.

A função void rrcp() é a implementação do método de escalonamento "Round-Robin com prioridade de envelhecimento" onde o que define qual processo será executado primeiro é a variável envelhecimento de cada processo. O valor inicial da variável envelhecimento de cada processo é o valor da prioridade do mesmo. Quando um processo está executando, sua variável envelhecimento é reiniciada. Já os outros processos, que estão no estado aguardando, tem a sua variável envelhecida em "alpha" a cada quantum passado. Como este método possui prioridade, ele não necessita terminar a execução do processo para fazer a troca de contexto, sendo assim, quando a variável envelhecimento for maior o processo inicia a sua execução.

Conclusão

Durante a implementação do algoritmo, nos deparamos com alguns problemas para ordenar os processos, sendo que quando criado, um processo teria que ficar aguardando até ter a prioridade ou até a finalização do processo em execução no momento. Para solucionar este problema, tivemos que criar um vetor de estados para cada processo. Este vetor, além de fazer o controle dos processos, nos auxiliou no final do programa no momento em que foi feito o comparativo entre os diferentes tipos de escalonamento, visto que o mesmo armazenava todos os estados para cada processo durante toda a execução do programa, sendo esses estados: em execução, esperando e não iniciado ou finalizado. A partir desses dados, pode ser feito também o diagrama de tempo de execução e a análise destes. Podemos concluir que os diferentes tipos de escalonamento podem ser empregados dependendo de sua aplicação pois, em determinados escalonamentos, os processos são executados quase que em paralelo e em outros de forma sequencial.