Marcelo Gaioso Werneck

NºUSP – 8061963

Sistemas Operacionais

Professora: Gisele da Silva Craveiro

Exercício Programa v1 e v2 (EP's)

Relatório de Qualidade

Sumário

Capítulo 1 - Escalonamento da CPU

Estrutura do EP

A estrutura do programa foi constituída a partir do artigo fornecido pela professora como apoio a construção do exercício. A representação de um processo para o simulador é reconhecida por uma entrada na jobtable, que no meu caso foi feita como uma matriz Nx5 de inteiros, onde:

-0 = ID do processo; (identificação)

-1 = Tempo de Chegada do Processo;

-2= CPU Burst; (quantos ciclos de CPU ele necessita – informal)

-3= Prioridade do processo;

-4 = Tempo de espera; (tempo que ele ficou esperando na fila para até ser executado);

-5 = Tempo finalizado.

Segundo o artigo, é necessária também uma lista de eventos, estes dos quais irão ocorrer em um próximo tempo, como por exemplo, a chegada de um processo, a requisição de CPU, etc. E para essa lista, foi criado uma LinkedList<Evento> nomeada de "lista" simplesmente. Além disso, quando um processo requisita a CPU e a mesma encontra-se ocupada, é necessário ter algum lugar para agenda-los, e para isso foi criado uma outra LinkedList<Evento>, usado especificamente como espera da CPU.

Esse foi o básico recomendado pelo artigo, mas tem também a estrutura que foi criada devido ao escalonamento da CPU. Existem 6 algoritmos de escalonadores implementados no programa, estes são: "First Come First Served (FCFS)" ; "Shorted Job First (SJFn)(Não Preemptivo)" ; "Shorted Job First (SJFp)(Preemptivo) ; "Round Robin (RR)" ; "Múltiplas Filas por Prioridade (MFp)" e por fim "Múltiplas Filas por realimentação (MFr)".

Foram criadas várias variáveis para o controle de tempo, quantum do RR, tipo de escalonador, etc. Mais duas LinkedList<Evento> para os escalonadores MF's . Isso tudo necessário para a estrutura em termos lógicos.

Agora quanto a estrutura em termos de implementação, de código, o programa foi separado/dividido em 7 classes: Main, Arquivo, Evento, Escalonador, Estrutura, Grafica e Sobre. Sendo essas ultimas duas interfaces gráficas para o programa.

-Main: É a classe executável, assim como a Grafica. Em seu método main(String[] args) contém apenas a chamada ao método principal da Grafica(interface gráfica do aplicativo). Mas possui os métodos mais usados e importantes do programa: void Executa() e void Agendador().

-void Agendador(): é o método capaz de remover um evento da event list e executar aquele evento.

-void Executa(): é o encarregado de botar o sistema para funcionar, ele chama o método que cria a jobtable, escreve no log, e enquanto o numero de processos ativos for > 0 ele chama o Agendador(), sem dizer que incrementa o tempo.

-Estrutura: É a classe mais importante, sem ela nada existe. Nela que estão as listas, as variáveis globais, a jobtable e assim por diante. Ela contem o método de criar lista de eventos, criar a jobtable, de criar, escrever e finalizar o log.

-Evento: Nessa classe contém todas as ações relativas a execução do evento, todas elas são invocadas pelo método executa, que verifica qual a identificação do evento e chama o método da ação correspondente. O evento é composto por:

-ID : Identificação do evento, este pode ser:

* ID = 1 > evento "inciailização" ;
* ID = 2 > evento "requisitando memoria" ;
* ID = 3 > evento "requisitando processador" ;
* ID = 4 > evento "saida da CPU para a finalização" .

--time: Tempo que o evento será executado.

--processo: Posição relativa a jobtable do processo ligado ao evento.

Implementação da Estrutura

Na classe contem os métodos : void inicio(); void requisitaCM(); void requisitaCPU(); void escalonador(int esc); void finalizacaoPadrao(). Onde são ações que os mesmos podem fazer sobre o processo.

Escalonador: É a classe correspondente a todos os algoritmos de escalonamento da CPU. Em que cada método dele é o nome de um escalonador, e todos eles executam ações especificas ao seu tipo.

Arquivo: É a classe que simplesmente está encarregada da abertura dos arquivos, ela contém os métodos que retornam o arquivo de entrada e o log.

Grafica: Classe principal da interface gráfica, nela estão todos os códigos sobre painéis, botões, spinners, ações, escolha de arquivos, etc. É o centro do programa para o usuário, e é por onde ele interasse com o sistema, selecionando os arquivos, alterando os valores das variáveis, escolhendo o escalonador, etc. Cada ação realizada pelo usuário(click de botão, teclando na textBox, acionando a Checkbox, etc) é um método em seu código.

Sobre: Classe de interface gráfica também, mas é simplesmente para uma nova janela que mostra informações sobre o programa, autor, fins, professora, e links para esse relatório e o manual de compilação.

Essa parte foi a descrição da estrutura do programa. Agora sobre o funcionamento do programa, é da seguinte forma:

É executada a classe Main, e ela chama a main da classe Grafica, que abre a interface gráfica. A partir daí, todas as alterações realizadas nos butões e escolha de variáveis terão impacto sobre o código. Logo de inicio tem o titulo e do lado direito tem um botão “Sobre”, que quando clicado abrirá uma nova janela (painel) com informações adicionais sobre o programa.

No painel “Sobre” também está dois links para o Manual de Compilação e esse Relatório de Qualidade. Se o painel for fechado, o programa ainda continua na execução.

Deve-se ser inserido um nome para um arquivo de entrada, ou então clicando no botão ao lado e escolhendo o caminho do mesmo. Se o arquivo de entrada não estiver na mesma pasta do aplicativo, é necessário escrever o caminho completo. No manual de compilação estão as clausulas e lá é especificado de como deve ser esse arquivo de entrada.

Como arquivo de saída, não é necessariamente escolher um arquivo, pois ele irá criar um automaticamente com o nome que for digitado, no caminho do qual está localizado o aplicativo. Se for escolhido um arquivo, ele irá sobrepor às informações.

Uma vez que o arquivo foi escolhido corretamente, ele irá automaticamente iniciar as variáveis do tipo File para a construção da jobtable e do log.

Há um GroupButtons, que deixa você selecionar apenas uma opção: “Log completo” ou “Apenas Resultados”, se “Log Completo” for selecionado, ele irá gravar no arquivo todas as informações relativas as ocorrências dos eventos, e somente no fim aparecerá as informações relativas a cada processo como resultado do escalonador. Agora se “Apenas Resultados” for selecionado, gravará-se apenas os resultados dos escalonadores.

Há um checkBox que pode ser selecionado ou não, caso você aperte o botão “Gerar” com ele ativo, no fim da execução do programa, se tudo ocorreu certamente, ele irá abrir o arquivo de saída automaticamente.

Na parte de baixo, está outro Groupbuttons do qual é para se escolher o escalonador que deseja que seja usado. Também possui Spinners, caixa do qual é possível selecionar a variável digitando ou clicando nas setas para incrementar ou decrementar seu valor. O valor mínimo permitido é 1 para todas.

O spinner Quantum é relativo ao escalonador RR, que também é usado na primeira fila do algoritmo MFr, os Spinners das prioridades são usados apenas no escalonadores MFp e o ultimo Spinner de Quantum 2 é usado apenas no escalonador MFr.

Funcionamento do programa

O botão “Gerar” quando clicado executa o método void Execute() da classe Main. Que dá inicio oficialmente ao programa.

Antes de qualquer coisa, ele vai abrir o arquivo de entrada, checar o primeiro numero, que corresponde ao numero de processos do programa. Com esse valor, ele vai criar uma matriz Nx7, e adicionar todos os valores do arquivo nessa matriz, em ordem. Essa é a jobtable. Os valores das ultimas colunas são deixadas para serem utilizadas depois.

Ele em seguida vai abrir o arquivo de log, e já escrever qual escalonador será utilizado. Depois ele invoca o método void CriaListaEventos(jobtable), que como o nome diz, ele vai varrer a jobtable, verificar o tempo atual, e se tiver algum processo que chega nesse tempo ele vai adicionar a lista de eventos como o ID = 1. Inicialização.

Ele então, se a lista não tiver vazia, chama o agendador(), que por sua vez vai retirar o primeiro elemento da lista e executá-lo. Como o ID = 1 ele vai inicializar oficialmente o processo (é algo mais fictício, e também para deixar igual ao do artigo), e mudar o ID para 2, e assim adicionar no final da lista de eventos.

Novos processos chegando por causa do tempo, sempre irão para o final da lista. O ID = 2 corresponde a requisição de CM. Que é concedida sem dificuldades (considerar como ilimitada), alterado para ID = 3 e adicionada ao final da lista novamente. Quando esse evento chega novamente a primeira posição e é executado, o método verifica se a variável boolean CPUlivre = true ou false, caso seja true, é alterada para false, o ID é modificado para 4 e é novamente adicionado a lista. Caso seja false, então depende do escalonador.

Um processo requisitando a CPU pode Preemptar o processo que está utilizando-a no momento, ele normalmente vai para a fila da CPU, mas caso o escalonador seja Multiplas Filas Prioridade, ele pode ir para alguma outra fila a depender das Priodades estabelecida. Então no método é verificado qual é o escalonador então realiza a determinada função.

Isso continua até chegar um evento do com ID = 4 para ser executado. Nesse momento ele chama o escalonador, que realiza sua função: gerenciar os processos que estão nas filas e o que está executando na CPU.

FCFS: Ele executa o processo até finalizar, então um while(CPUBurst > 0) é o sulficiente.

SJFn: Toda vez quando um processo finalizar, ele escolhe qual será o próximo a ser executado a partir do menor CPUburst dos processos que estão na fila da CPU.

SJFp: Ele é igual ao anterior com execção que o processo que esta na CPU pode ser preemptado caso um outro processo requisite a CPU e o mesmo tenha um CPUBurst menor. Caso isso ocorra o processo executando é parado e vai para a fila da CPU.

RR: Ele usa um quantum de tempo para percorrer os processos da Fila da CPU e executando um de cada vez. Por quantum tempo for estabelecido. O processo pode acabar ou não no meio de um ciclo, caso acabe ele é finalizado, caso não acabe ele é preemptado e adicionado no fim da fila da CPU, então é carregado o primeiro da fila de espera.

MFp: Usa 3 filas para gerenciar os processos, essas filas são determinadas pelas prioridades: Fila 0 é para os processos com prioridades <= Prioridade 1 (determinado pelo usuário. Fila 1 é para os processos com prioridades > Prioridade 1 e < Prioridade 2. E por fim os da Fila 2 são para os > Prioridade 2. É executado usando algoritmo RR. E Somente executa processos das filas 2 e 1 quando a fila 0 estiver vazia. Idem para a fila 2 com a 1.

MFr: Usa também as 3 filas. Mas dessa vez todos os processo começam da fila 0, caso o processo não finalize pelo quantum pré-estabelecido, ele é colocado na fila1, que por sua vez faz uso do Quantum 2, caso esse ainda não termine, então é colocado na fila 2, ai é executado usando FCFS mas somente quando a Fila 0 e 1 estiverem vazias(como o escalonador passado).

Uma vez que isso é realizado a todos os processos, o programa escreve os valores dos quais todos os processos finalizaram, relevante o tempo de espera, e o tempo finalizado. Por isso tem o tempo médio de espera. Tempo total de espera / numero de processos. Então os arquivos são fechados e o programa é finalizado.

Capítulo 2 - Gerenciamento de Memória

O que foi acrescentado na estrutura?

Foi muito fácil essa nova parte, na estrutura apenas adicionada uma nova classe chamada Memoria, em que nessa classe possui um array de tamanho a determinar pelo usuário, possui também uma a tabela de pagina, o frame e variáveis inteiras como, por exemplo, o espaço livre.

Nessa classe foi implementado métodos que busca menor buraco na CM, ou seja, Best-fit, que dado um tamanho mínimo ele percorre um array e separa o menor buraco possível para aquele tamanho mínimo.

Ainda na classe, possui o método de liberar memória, e o de requisitar memória, em que consegue ao processo o tanto de memória sulficiente para sua execução, e libera só tira-o da memória e procura um novo processo na lista de espera à memória que tenha espaço livre o bastante para entrar na memória (variação do FIFO), tornando assim o uso da memória continuo. O contra é que pode haver Starvation, ou seja, processos que requerem muita memória podem demorar muito, ou até não executar por ter outros menores na sua frente.

Com relação ao programa em geral, na interface foi adicionado novos RationButtons, labels e spinners para a contagem do tanto de memória total será considerada e o tamanho do frame na parte de Paginação.

O novo funcionamento do programa

Muda que, em seu funcionamento de escalonamento de CPU, no evento requisitaCM(que já existia, apenas concedia memória ilimitada no começo) foi adicionado a tarefa de escalonar e alocar memória para o determinado processo, dependendo de seu tamanho.

Quando clicar no botão “Gerar” ele não vai simplesmente escalonar o processo na CPU pelo algoritmo escolhido, como também gerenciar e alocar a memória usando o método escolhido pelo usuário.

Ainda, no log, aparecerá dados específicos de quanta memória ainda há sobrando, toda vez que um processo estiver fazendo a requisição dela (também mostra a quantidade de páginas do processo). Além de que no final, no resumo aparecerá quanto cada processo possuía de paginas ou seu tamanho, dependendo do método de gerencia de memória escolhido.