Projeto da solução

A idéia principal do projeto é a *Main* ler o arquivo de entrada, disparar as threads *Elevadores* e a partir de então concorrer por CPU com os *Elevadores* para inserir as tarefas dentro do *Monitor*. E os *Elevadores* (também disputando CPU) tentarem pegar uma tarefa, que esteja mais próxima de seu andar atual, dentro do *Monitor*. Um clássico problema de Produtor X Consumidor, onde a *Main* é a thread produtora e os *Elevadores* são as threads consumidoras.

- **Main**

Iniciamos com a thread principal que recebe o arquivo de entrada contendo as informações necessárias para o funcionamento dos elevadores.

Ela lê e armazena o arquivo inteiro em um array de String separando por linhas(*String[] linhas*). Cada linha do arquivo é uma posição do array.

Guarda os valores das três primeiras linhas em variáveis auxiliares que serão usadas nos próximos passos (número de andares, número de elevadores e número de requisições) convertendo do tipo String para inteiro.

Cria um objeto monitor e um vetor de M(número de elevadores) objetos elevador.

Inicializa as threads elevadores passando os parâmetros (id do elevador, andar de partida, o objeto monitor e o número de elevadores).

Após isso, a main coloca um id em cada tarefa através de um loop *for* que roda tantas vezes quanto o número total de tarefas e tenta inserir na matriz de tarefas que está dentro da classe *Monitor* chamando o método *insereTarefa* que será explicado mais abaixo.

Depois desse loop acabar, a *main* espera a execução das threads através do *join.*

Assim que as threads acabam a *main* imprime na tela o tempo total gasto pela aplicação, e uma mensagem sinalizando que o programa terminou.

- **Elevador**

No método *run* de cada thread elevador criada, há um loop *while* que verifica a possibilidade de acesso à matriz de tarefas no monitor (através do método “*acessaTarefas*”), os Elevadores começam a tentar pegar alguma tarefa (se já/ainda houver alguma tarefa pronta que não foi acessada). Após o retorno de *getTarefa (int[])*, começa a deslocar o elevador chamando o método *deslocaElevador* que será explicado mais abaixo.

- **insereTarefa, splitTarefa e strToInt** (Monitor)

O método insereTarefa trata a tarefa, recebida por parâmetro, que ainda é uma String, chamando o método *splitTarefa* e coloca numa matriz de inteiros(int[][] matriz_tarefas).

O *splitTarefa* separa a String recebida pelos espaços e chama o método *strToInt*
que passa a tarefa de String para inteiro.

Após isso, *insereTarefa* volta a sua execução e a variável *tarefas_disponíveis* é incrementada (informando que já existe tarefa pronta, ou seja, tratada por esse métodos, para ser usada). As threads elevadores que possam já estar esperando são notificadas que já existem tarefas disponíveis na matriz através do *notifyAll.*

- **getTarefa** (Monitor)

O método inicializa as variáveis que serão utilizadas e depois, após uma conferencia, verifica se há alguma tarefa disponível na matriz de tarefas, caso não exista, as threads ficam bloqueadas no wait (não há tarefas disponíveis ainda).

Ao serem notificadas pela main (através do método *insereTarefa*), os elevadores voltam a conferir se existe tarefa para a execução (verificam de novo pois enquanto voltam a disputar pelo synchronized outros elevadores podem vir antes, pegar as tarefas e voltar a não ter nenhuma disponível).

Como o bloco desse método é synchronized, um por um, os elevadores terminam de executar o método que escolhe a tarefa selecionando a primeira que tiver a partida inicial mais próxima do andar atual do elevador.

Assim que uma tarefa foi pega, a variável *tarefas_acessadas* é incrementada (essa variável é usada pelo método acessaTarefas que libera o acesso das threads à lista de tarefas).

A tarefa é passada ao elevador e impossibilitada de ser acessada novamente (a tarefa não será mais acessada porque seu id é zerado e ao procurar uma tarefa, o método não permite acesso a tarefas com o id = 0).

- **deslocaElevador** (Elevador)

Esse método salva, num vetor de inteiros(int[] *tarefaOrdenada*), a tarefa ordenada e sem repetição (usando métodos *ordenaTarefa* que chama dentro dele o método *excluiIguais*).

Cria um arquivo de saída (um para cada elevador) e escreve, no padrão exigido, as três primeiras linhas.

Num loop *for*, que percorre todo o vetor de tarefas, simulamos o elevador chegando aos andares escrevendo no arquivo cada

atualização da lista de andares (a cada iteração, atualizamos a lista de andares criando novas listas e escrevemos no arquivo).

Após o término de uma lista, o método confere se ainda existe tarefa não acessada no monitor (mais uma vez fazendo uso do método *acessaTarefas*). Se existir, o elevador faz tudo de novo (volta a executar o método *run)*. Se não existir, o arquivo do elevador é finalizado (escrevendo "fim") e informando no terminal quantas tarefas realizou.