Segundo trabalho prático de implementação
Controle de elevadores

Computação Concorrente (MAB-117) — 2015/2
Prof. Silvana Rossetto

[1]DCC/IM/UFRJ
12 de janeiro de 2016

1. Descrição do problema

Considere um edifício com N andares e M elevadores. O problema consiste em implementar uma aplicação concorrente (Sistema de Controle dos Elevadores - SCE) para controlar a operação dos elevadores. O funcionamento de cada elevador deve ser gerenciado por uma thread (fluxo de execução independente). Os M elevadores devem dividir a tarefa de atender a todas as requisições dos usuários em cada um dos N andares do edifício. Para isso precisam sincronizar suas ações e compartilhar o estado global (conjunto de requisições dos usuários).

Quando o usuário precisa usar o elevador — a partir de qualquer andar — ele informa qual é o andar de destino. O SCE decide então qual elevador irá atendê-lo. O mesmo elevador pode atender a vários usuários ao mesmo tempo. A única restrição é que os elevadores possuem uma capacidade máxima (número máximo de pessoas que podem usar o elevador ao mesmo tempo) que sempre deve ser respeitada. Assim, por exemplo, se a capacidade máxima de cada elevador for de 8 pessoas e há 10 pessoas esperando em um determinado andar, um elevador só poderá levar 8 dessas 10 pessoas, as outras 2 deverão ser atendidas por outro elevador.

Para cada andar no edifício deverá existir uma fila de espera com todas as requisições dos usuários que estão aguardando nesse andar. Uma requisição será uma tupla com as seguintes informações: **identificador único da requisição** (gerado pelo sistema) e **andar de destino**. A *thread main* será responsável por incluir novas requisições nessas filas, enquanto as *threads dos elevadores* serão responsáveis por tratar e retirar as requisições dessas filas. Uma mesma requisição não pode ser tratada por mais de um elevador.

O funcionamento básico dos elevadores deverá ser o seguinte:

```
enquanto há requisições pendentes em algum andar do edifício:
   desloca-se até esse andar
   seleciona até C requisições
   calcula seu trajeto
   desloca-se parando nos andares de destino das requisições
fim enquanto
```

Durante seu trajeto, o elevador não recebe novas requisições (entrada de novos usuários), apenas atende aquelas que já estão com ele. C é a capacidade máxima do elevador. Sempre que o elevador termina de atender a uma lista de requisições, ele permanece no andar onde deixou o último usuário.

Para economizar energia elétrica, a seguinte otimização deverá ser implementada: quando há requisições pendentes em mais de um andar, o elevador deverá selecionar o

andar de partida que esteja mais próximo do andar onde ele está no momento. Se houver mais de uma alternativa, ele deverá escolher o andar que tem o maior número de requisições pendentes. Se ainda houver mais de uma alternativa, ele deverá selecionar qualquer uma delas.

A thread principal do programa é responsável por criar e disparar as threads dos elevadores e, em seguida, receber as requisições dos usuários e inseri-las nas filas de cada andar. **Para facilitar os testes, quando não houver mais requisições dos usuários, a thread principal deverá avisar as threads dos elevadores para que as mesmas encerrem suas execuções e o programa termine**.

O trabalho poderá ser implementado nas linguagens C ou Java.

2. Entrada

Para cada execução do programa, os seguintes dados deverão ser fornecidos como entrada, nesta ordem: (a) valores de N ($5 <= N <= 100$), M ($5 <= M <= 20$), C ($5 <= C <= 20$); (b) andar inicial dos M elevadores; (c) uma linha para cada andar com a lista de destinos dos usuários (o primeiro valor em cada linha será o número de usuários aguardando).

O número do primeiro andar deverá ser 0 e do último $N - 1$.

Exemplo de entrada

```
5 2 6
0 4
6 1 4 2 3 3 4
4 0 3 0 5
8 1 6 5 3 4 4 3 5
5 0 5 0 0 0
3 0 1 1
```

Nesse caso, a primeira linha informa que teremos 5 andares, 2 elevadores e no máximo 6 usuários por elevador. A segunda linha informa que o primeiro elevador começará no andar 0 (térreo) e o segundo elevador começará no andar 4 (último andar). As cinco linhas seguintes (uma para cada andar) informam, respectivamente, o número de usuários aguardando em cada andar e os andares de destino desses usuários.

A thread principal deverá acrescentar um identificador único para cada requisição, seguindo a ordem em que elas foram lidas, e armazená-la na fila de requisições do respectivo andar.

3. Saída

As threads deverão registrar em arquivos de saída distintos (um para cada thread, incluindo a *main*), cada ação realizada e informações sobre o estado corrente da aplicação. O log de saída deverá ser projetado de forma a permitir avaliar a corretude da aplicação e as otimizações realizadas.

4. Etapas do trabalho

A execução do trabalho deverá ser organizada nas seguintes etapas:

1. Compreender o problema, pensar e esboçar uma solução concorrente;
2. Projetar as estruturas de dados e as funçoes que deverão ser implementadas;
3. Construir um conjunto de casos de teste para avaliação da aplicação;
4. Implementar a solução projetada, avaliar a corretude do programa, refinar a implementação e refazer os testes;
5. Redigir os relatórios e documentar os códigos.

5. Artefatos que deverão ser entregues

1. Relatório: documentação do projeto da solução (esboço das estruturas de dados e lógica principal dos algoritmos da aplicação), testes realizados e resultados obtidos. O relatório deverá conter informações suficientes para o professor compreender a solução proposta sem precisar recorrer ao código fonte do programa.
2. Código fonte e roteiro para compilação.

Enviar para o email `computacao.concorrente.ufrj@gmail.com` com o assunto "Trabalho 2 - 2015/2"e dentro do corpo da mensagem os nomes completos dos autores.

6. Data de entrega e critérios de avaliação

O trabalho poderá ser feito em **dupla** e deverá ser entregue até o dia **5 de fevereiro**.

Os seguintes itens serão avaliados com o respectivo peso:

- Compilação e execução correta da solução: 4 pontos
- Relatório (incluindo projeto da solução e testes realizados): 2 pontos
- Modularidade, organização e documentação do código fonte: 1 ponto
- Otimizações, interfaces do programa, e logs de saída: 2 pontos
- Avaliação geral do trabalho: 1 ponto

Os alunos integrantes da equipe poderão ser chamados pelo professor para apresentar/explicar o trabalho.