Exercícios de Teste/Exame

Sistemas Distribuídos

Exercício

II

Considere um serviço de transferência de passageiros. Assuma 5 terminais, um percurso circular (ou seja, 1–2–3–4–5–1 repetidamente), e um shuttle com capacidade para 30 passageiros. O shuttle pára em cada terminal para permitir saída e entrada de passageiros. Por questões de rentabilidade, o shuttle só viaja com pelo menos 10 passageiros, esperando por mais se necessário. A viagem entre terminais demora 5 minutos. Cada passageiro utiliza uma aplicação (cliente) que interage com o servidor que controla o sistema, devendo permitir: o passageiro requisitar ao servidor uma viagem entre o terminal onde está (origem) e o terminal de destino; o servidor informar o passageiro que pode entrar no shuttle; o servidor informar o passageiro que chegou ao seu destino.

1 Apresente uma classe (para ser usada no servidor) que implemente a interface abaixo, tendo em conta que os seus métodos serão invocados num ambiente multi-threaded.

```
interface Controlador {
  void requisita_viagem(int origem, int destino);
  void espera(int destino);
}
```

O método `requisita_viagem` deve bloquear até o passageiro poder ser informado que pode entrar no shuttle (o shuttle chegou à origem e há lugar disponível). O método `espera` deve bloquear até o shuttle ter chegado ao terminal `destino`. Caso haja mais passageiros num terminal do que lugares disponíveis, os passageiros devem entrar por ordem de requisição. Nota: esta interface deve considerar o uso dos seus métodos apenas por threads que representam passageiros; considere a possibilidade de criar threads auxiliares na sua implementação.

2 Implemente o programa servidor usando threads, sockets TCP, e a classe desenvolvida na pergunta anterior.

Exercício

1. Fazer uma leitura atenta e perceber o que é a base do exercício, ou seja que entidades e acções existem:
   - shuttle com percurso circular que transporta passageiros entre terminais
   - controlador permite aos passageiros:
     - requisitar viagens;
     - saber quando entrar no shuttle (no terminal de origem);
     - saber quando sair do shuttle (no terminal de destino).
2. Perceber que condições estão associadas a cada acção:
   - capacidade máxima: 30 passageiros;
   - o shuttle só viaja com pelo menos 10 passageiros, esperando por mais se necessário;
   - caso haja mais passageiros num terminal do que lugares disponíveis, os passageiros devem entrar por ordem de requisição.

Exercício - Versão 1

- Passageiros **entram** e **saem** no terminal certo.
- Shuttle com **capacidade ilimitada**.
- Shuttle **parte** quando tiverem saído todos os passageiros que têm como destino esse terminal.

Exercício - Versão 2

- Passageiros **entram** e **saem** no terminal certo
- Shuttle com **capacidade limitada** (30 passageiros).
- Shuttle **parte** quando tiverem saído todos os passageiros que têm como destino esse terminal.
- Shuttle **parte** quando estiver cheio.

Exercício - Versão 3

- Passageiros **entram** e **saem** no terminal certo
- Shuttle com **capacidade limitada** (30 passageiros).
- Shuttle **parte** quando tiverem saído todos os passageiros que têm como destino esse terminal
- Shuttle **parte** quando (estiver cheio || (10 passageiros && não há passageiros à espera nesse terminal))

Exercício - Versão 4

- Passageiros **entram** e **saem** no terminal certo
- Shuttle com **capacidade limitada** (30 passageiros).
- Shuttle **parte** quando tiverem saído todos os passageiros que têm como destino esse terminal.
- Shuttle **parte** quando estiver (cheio || (10 passageiros && não há passageiros à espera nesse terminal))
- Caso haja mais passageiros num terminal do que lugares disponíveis, os passageiros devem **entrar** por **ordem** de requisição

Optimizações

- Utilizar signal() em vez de signalAll(), quando possível. (versão 5)
- Considerar variáveis de condição por terminal. (versão 6)
- Thread que faz avançar o shuttle (versão 7).