Sistemas Distribuídos

Guiões Práticos

Grupo de Sistemas Distribuídos

2008/9

Capítulo 1

Criação de Threads

1.1 Conceitos relevantes

- fios de execução concorrentes de um programa

1.2 Classes, interfaces e métodos mais relevantes

- java.lang.Runnable
  - interface implementado por classes cujas instâncias são executadas por uma thread
  - classes que implementem o interface têm que implementar o método run()
- java.lang.Thread
  - implementa java.lang.Runnable
  - classes que extendam Thread devem reimplmentar o método run()
  - outros métodos relevantes: Thread(), start(), sleep(), join()

1.3 Exercícios propostos

1. Crie uma thread que imprima e incremente um contador a cada segundo, enquanto outra thread repete o standard input para o standard output. Nota: o corpo (ou thread) principal do programa espera implicitamente que as threads terminem a sua execução.

2. Crie dez threads que incrementem dez milhões de vezes um contador partilhado. No final da execução das threads, o corpo ou thread principal do programa deve imprimir o valor final do contador. Nota: a thread principal do programa tem que esperar que as threads criadas terminem, antes de imprimir o valor do contador.

Capítulo 2

Exclusão Mútua

2.1 Conceitos relevantes

- corrida e zona crítica
- situação de deadlock

2.2 Construções da linguagem

- mecanismo intrínseco de exclusão mútua disponível em todos os objectos
- synchronized *tipo* metodo (*argumentos*) { *statements* }
- synchronized (objecto) { *statements* }
- aquisição reentrante do acesso às regiões críticas protegidas pela exclusão mútua

2.3 Exercícios propostos

1. Modifique o exercício anterior — incremento concorrente de um contador partilhado — de modo a garantir a execução correcta do programa.
2. Implemente uma classe Banco que ofereça os métodos de consulta, crédito e débito de valores sobre um número fixo de contas (com saldo inicial nulo). Utilize exclusão mútua ao nível do objecto Banco.
3. Acrescente o método transferir à classe Banco como composição das operações de débito e crédito de um valor sobre duas contas.
4. Reimplemente a classe Banco utilizando exclusão mútua ao nível das contas inviduais.

Capítulo 3

Variáveis de Condição

3.1 Conceitos relevantes

- suspensão/retoma de execução dentro de zona crítica

3.2 Construções de linguagem

- mecanismo intrínseco de variável de condição em todos os objectos
- métodos wait(), notify(), notifyAll()

3.3 Exercícios propostos

- Reimplemente a classe Banco de modo a bloquear as operações que conduzam a saldos negativos.
- Implemente uma classe BoundedBuffer que ofereça as operações `void put(int v)` e `int get()` sobre um array cujo tamanho é definido no momento da construção de uma instância. O método `put()` deverá bloquear enquanto o array estiver cheio e o método `get()` deverá bloquear enquanto o array estiver vazio. Os métodos oferecidos podem estar sujeitas a invocações de threads concorrentes sobre uma instância partilhada. A classe BoundedBuffer deverá garantir a correcta execução em cenário multi-thread.
- Implemente uma classe Barreira que ofereça um método `esperar()` cujo objectivo é garantir que cada thread que o invoque se bloqueie até que o número de threads nesta situação tenha atingido o valor `N` passado ao construtor de uma sua instância.
- Implemente o mecanismo tradicional de semáforos à custa do mecanismo de variáveis de condição.

Capítulo 4

Mecanismos Explícitos de Sincronização

4.1 Conceitos relevantes

- situação de *starvation*

4.2 Classes e métodos relevantes

- java.util.concurrent.locks.ReentrantLock
  - métodos lock(), unlock(), newCondition()
- java.util.concurrent.locks.Condition
  - métodos relevantes: await(), signal(), signalAll()

4.3 Exercícios propostos

1. Reimplemente a classe BoundedBuffer de modo a distinguir as situações de bloqueio pelo array estar vazio e estar cheio.
2. Implemente a classe RWLock com os métodos *readLock()*, *readUnlock()*, *writeLock()* e *writeUnlock()* de modo a permitir o acesso simultâneo de múltiplos leitores a uma dada região crítica, ou em alternativa, o acesso de um único escritor.
3. Procure reimplementar a classe RWLock de modo a evitar o fenómeno de starvation dos escritores.

Capítulo 5

Sockets Cliente/Servidor

5.1 Conceitos relevantes

- paradigma cliente/servidor
- comunicação em rede, endereço e porto, fiabilidade
- heterogeneidade e representação externa de dados

5.2 Classes e métodos relevantes

- java.net.io.*
- java.net.ServerSocket
  - métodos relevantes: ServerSocket(), accept(), close()
  - outros métodos: setReuseAddress(), bind()
- java.net.Socket
  - métodos relevantes: Socket(), connect(), read(), write(), getInputStream(), getOutputStream()
  - outros métodos: shutdowInput(), shutdownOutput()

5.3 Exercícios propostos

1. Implemente um servidor que aceite a ligação de um cliente de cada vez, e que devolva ao cliente cada linha de texto que este lhe envie. Nota: pode testar o servidor desenvolvido recorrendo ao comando *telnet*.
2. Implemente um cliente para o servidor de eco desenvolvido no exercício anterior.

3. Implemente um servidor que aceite a ligação de um cliente de cada vez. O servidor receberá de cada cliente, uma sequência de inteiros — pode optar tanto pelo formato binário como de texto – que terminará quando detectar a situação de *end of file* na *stream* de leitura do socket. No final da leitura dos valores inteiros, o servidor devolve ao cliente a soma correspondente.

4. Implemente um cliente para o servidor de soma desenvolvido no exercício anterior.

5. Implemente um servidor e um cliente da classe Banco desenvolvida anteriormente.

Capítulo 6

Sockets Cliente/Servidor em Multi-Tarefa

6.1 Conceitos relevantes

- múltiplas threads de comunicação

6.2 Exercícios propostos

1. Reimplemente o servidor de Banco do exercício anterior de modo a que este aceite a conexão simultânea de múltiplos clientes.
2. Implemente um servidor de conversação que aceite a conexão de múltiplos clientes. Cada mensagem enviada por um cliente é difundida por todos os clientes ligados.
3. Acrescente a funcionalidade de registo de *nicks* e do envio de mensagens privadas ao serviço de conversação desenvolvido no exercício anterior.