Computação Paralela e Distribuída – aula prática nº 8

Gestão de processos e sincronização em Java

- Problema dos leitores-escritores

Construção de uma versão simplificada de um monitor transaccional para contas bancárias

- Gestor de locks e de contas bancárias

Os slides da 1ª parte são adaptações dos que estão disponíveis no site do livro "Operating System Concepts with Java 8thEd" de Silberschatz et al, Ed. Wiley

O problema da gestão das contas bancárias

- Nas próximas 3 aulas vamos fazer um problema que ilustra os conceitos dados nas aulas teóricas sobre transacções
- Envolve processos /threads e comunicação entre processos
  - Linux / C / pthreads / sockets
  - Java / threads / sockets
- Planeamento
  - Semana de 22 de Abril: gestor de dados e gestor de locks
  - Semanas de 29 de Abril e 6 de Maio: gestor de transacções

1ª parte: gestão de contas [1]

- Pretende-se que construa um processo servidor que
  - Gere um conjunto de contas bancárias. O nº de contas é fixo e igual a NCONTAS
  - As contas estão representadas por um vector de reais que representa o saldo da conta
  - O nº da conta é o índice no vector
- Operações definidas (para já)
  - writeAccount (+ nº conta, + valor)
  - readAccount (nº conta, - saldo

1ª parte: gestão de contas [2]

- Mensagens trocadas entre os clientes e o gestor de contas são strings terminadas por EOL. O parâmetro status das respostas é ‘ok’ ou ‘not_ok’
- readAccount
  - Pedido: ler nº_conta
  - Resposta: status valor
- writeAccount
  - Pedido: escrever nº_conta novo_saldo
  - Resposta: status

2ª parte: gestão de locks [1]

- Pretende-se que construa um processo servidor que gere locks
  - Cada lock corresponde a uma dada conta. Há por isso NLOCKS (igual a NCONTAS). Há processos que consultam o saldo (leitores) e processos que alteram o saldo (escritores)
- Há dois tipos de locks
  - WriteLocks: usados pelos escritores; só podem ser concedidos em exlusividade - exigem que mais nenhum processo (leitor ou escritor) detenha um lock sobre a conta
  - ReadLocks: usados pelos leitores; vários leitores podem ter ter locks de leitura em simultâneo; isto é um lock de leitura pode ser concedido desde que nenhum processo detenha um lock de escrita

2ª parte: gestão de locks [2]

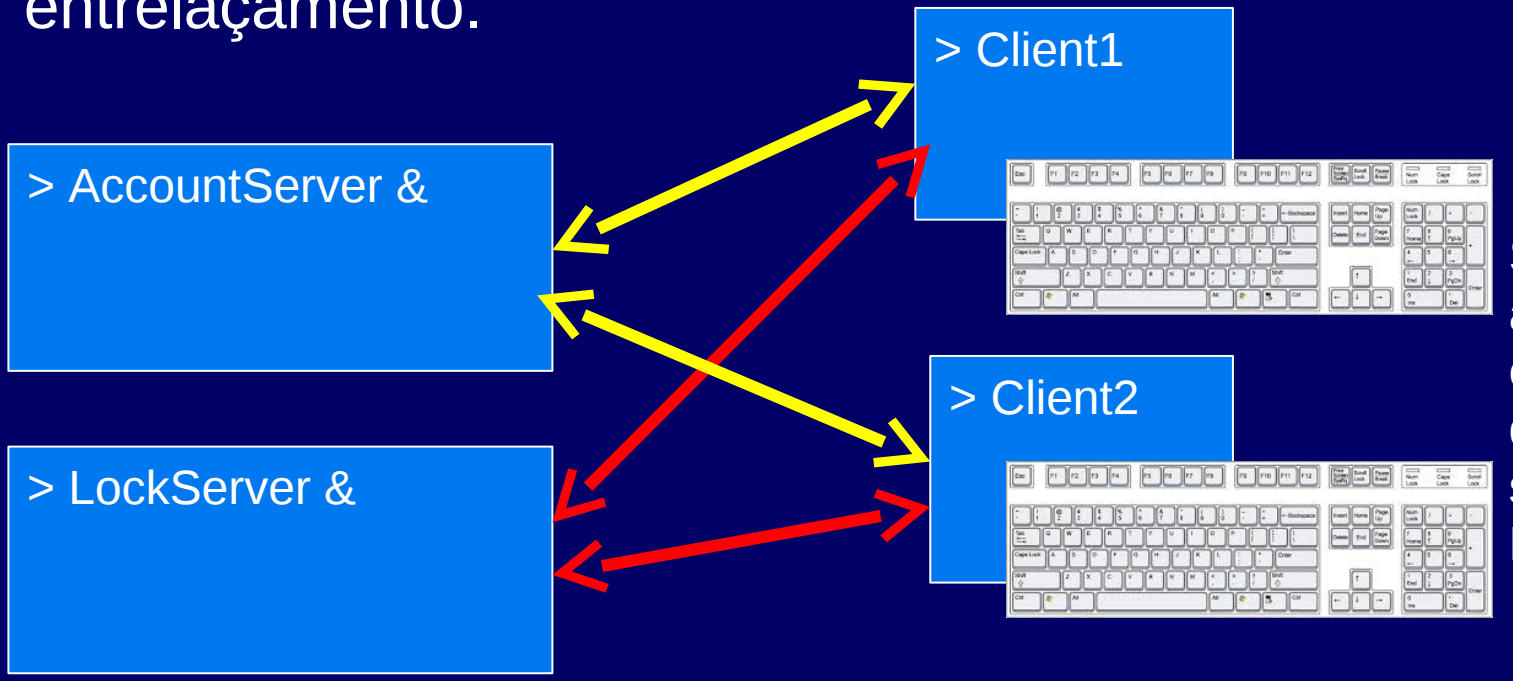
- Trata-se de uma versão do problema dos leitores escritores já visto. Pretende-se que use a versão que impede que os escritores sejam vítimas de starvation
- Operações definidas
  - lockWrite (+ nº lock)
    - Pedido: lockW nº_lock; Resposta (quando possível) ok; not_ok em caso a ver à frente
  - unlockWrite (+ nº lock)
    - Pedido: unlockW nº_lock; Resposta ok; not_ok se não existe
  - lockRead (+ nº lock)
    - Pedido: lockR nº_lock; Resposta (quando possível) ok; not_ok em caso a ver à frente
  - unlockRead (+ nº lock)
    - Pedido: unlockR nº_lock; Resposta ok; not_ok se não existe

2ª parte: gestão de locks [3]

- O servidor bloqueia os clientes até lhes poder dar uma resposta positiva
- Como resolver os casos de espera eterna (deadlock ou erro) ?
  - Timeout ao fim de N segundos
  - Valor a ajustar
  - Resposta Not_ok

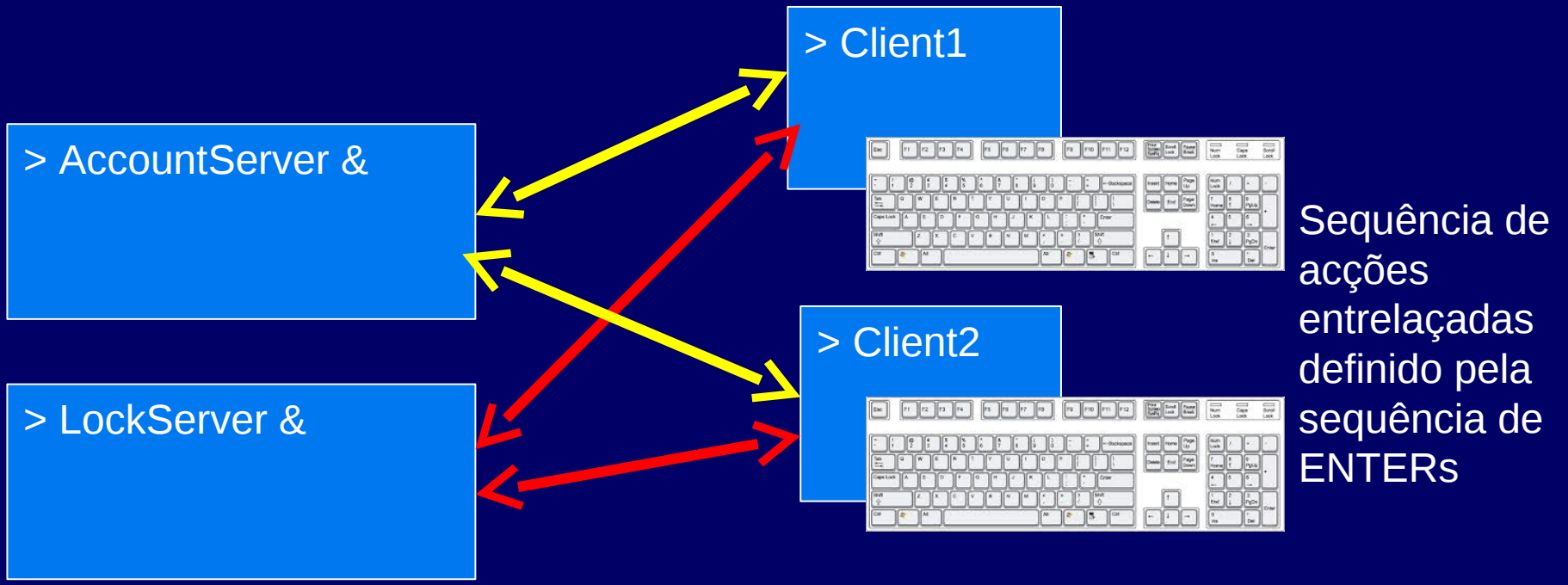
3ª parte: Teste do sistema [1]

- Pretende-se que teste o sistema através de clientes que invocam as operações dos servidores atrás definidas
- Deve colocar no final das operações de readAccount, writeAccount, lockRead, lockWrite, unlockRead e unlockWrite uma leitura do teclado que permita vários tipos de entrelaçamento.



3ª parte: Teste do sistema [2]

- Teste 1: exclusividade dos writeLocks
- Teste 2: concorrência dos readLocks
- Teste 3: não starvation dos escritores
- Teste 3: não bloqueio em caso de deadlock



3ª parte: Teste do sistema [3]

- Detecção de anomalias
  - Teste 1: actualização perdida: C2 lê, C1 escreve, C2 escreve
  - Teste 3: leitura suja: C2 escreve, C1 Lê, C2 escreve
  - Teste 3: leitura irrepetível: C1 lê, C2 escreve, C1 lê