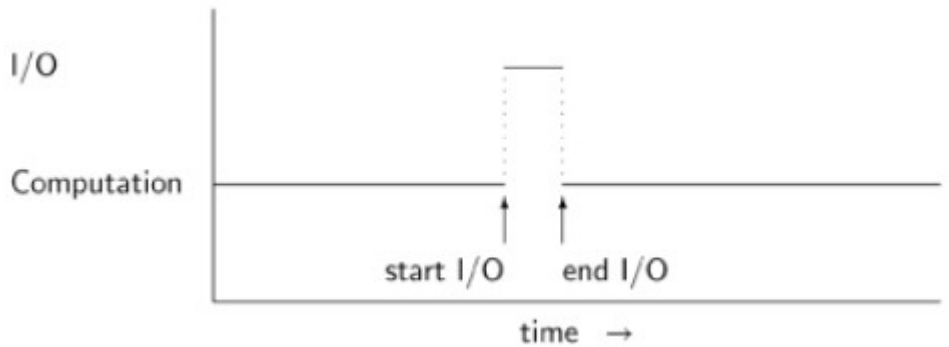
Programação Concorrente em Java

Aula 01

Introdução

Introdução

- Concorrência em Java (Java Concurrency) é um termo que cobre *concorrência* e *paralelismo* na plataforma.
- Isso inclue ferramentas, problemas e soluções.
- "***A concurrent program is a set of sequential programs that can be executed in parallel***" - Ben Ari



Conceitos Básicos

- O que é concorrência?

  - É a habilidade de executar vários programas ou partes de um programa em paralelo. Esses programas, ou partes deles, **concorrem** pelos recursos computacionais.

- Exemplos da vida real?
- Exemplos computacionais?

Uma breve história...

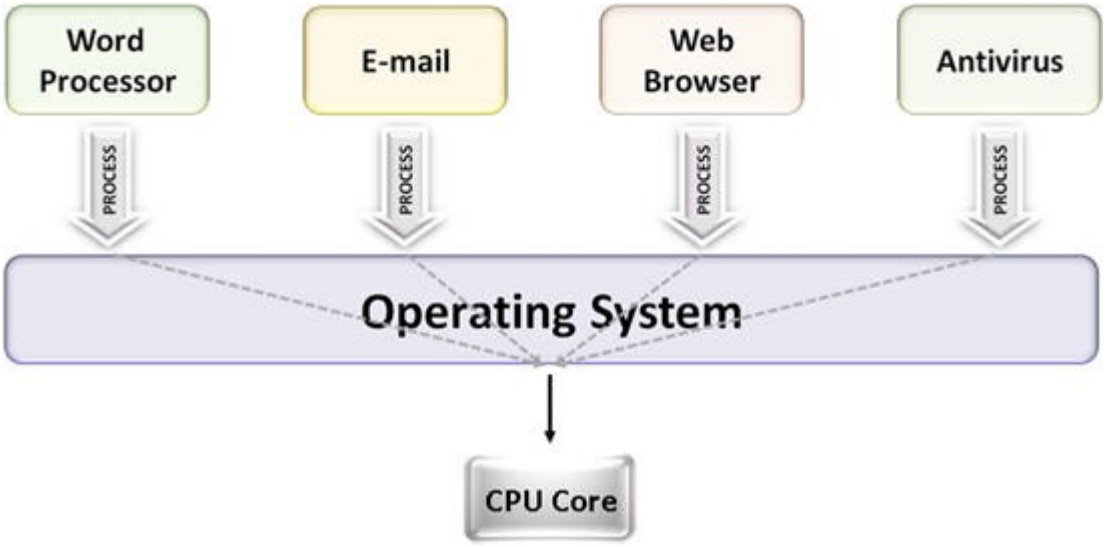
- Antigamente, computadores tinham apenas uma única CPU e só podiam executar uma tarefa de cada vez.
- Depois veio o **multitasking** o qual permitia que os computadores pudessem executar vários programas ou processos “ao mesmo tempo”.
- Com o multitasking, vieram novos desafios (e vários problemas). Programas não podiam assumir que a CPU estivesse disponível o tempo todo.

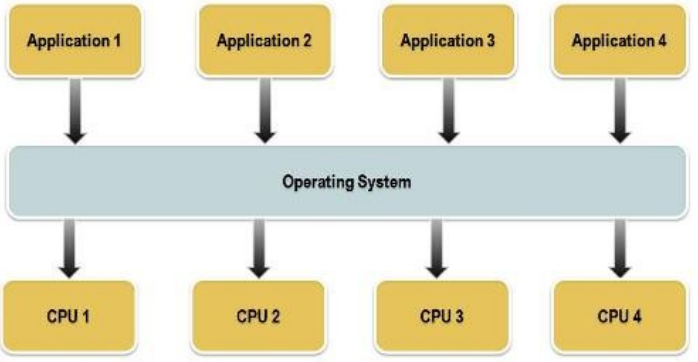
Uma breve história...

- Multitasking
  - Sobreposição de programas.
  - Função central do Kernel de todos os sistemas operacionais modernos.
  - Um SCHEDULER é executado pelo sistema operacional tomando prioridades na execução de processos (*time-slicing*).
  - Se tornou tão popular que deu origem ao Multithreading das linguagens de programação.
    - *“Threads enable the programmer to write concurrent (conceptually parallel) computations within a single program. For example, interactive programs contain a separate thread for handling events associated with the user interface that is run concurrently with the main thread of the computation. It is multithreading that enables you to move the mouse cursor while a program is performing a computation.” -Ben Ari.*

Uma breve história...

- Multitasking



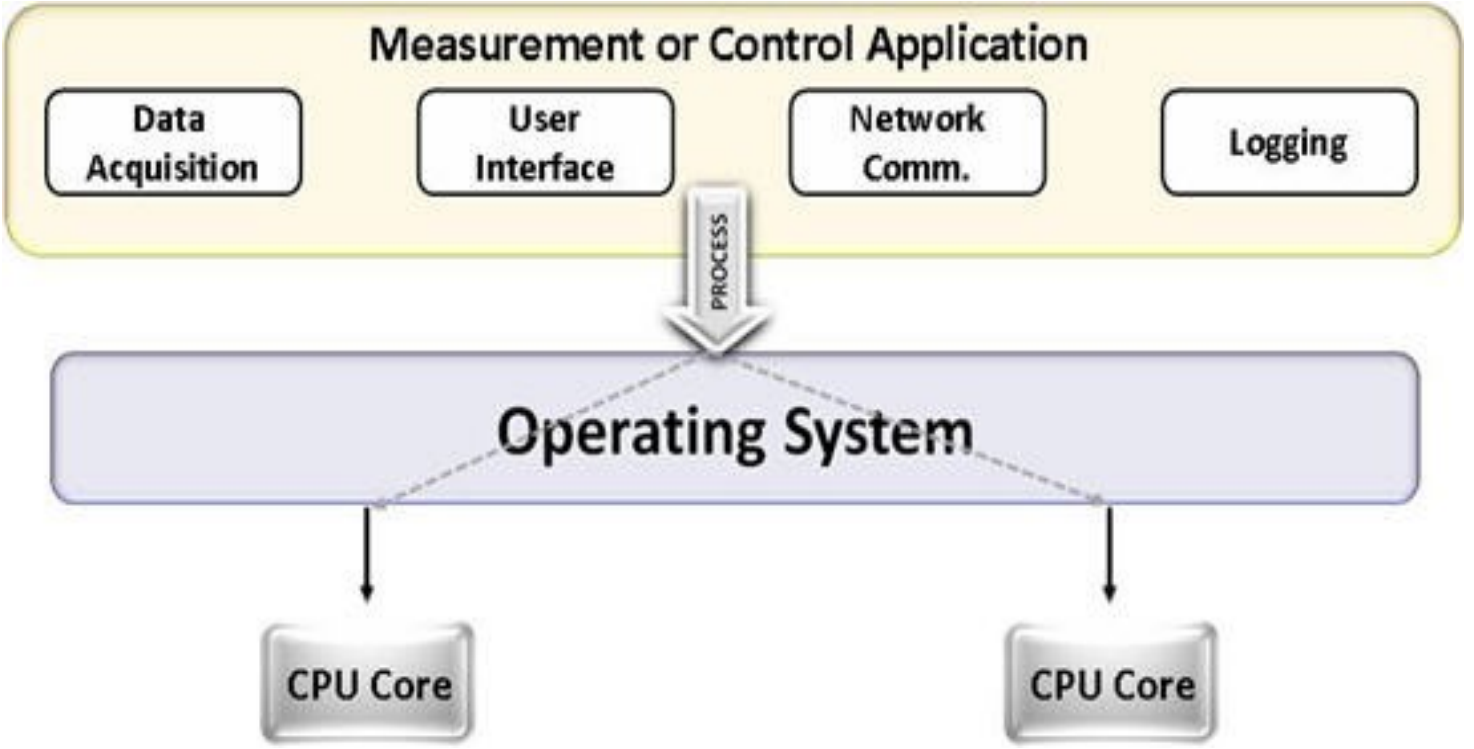


Uma breve história...

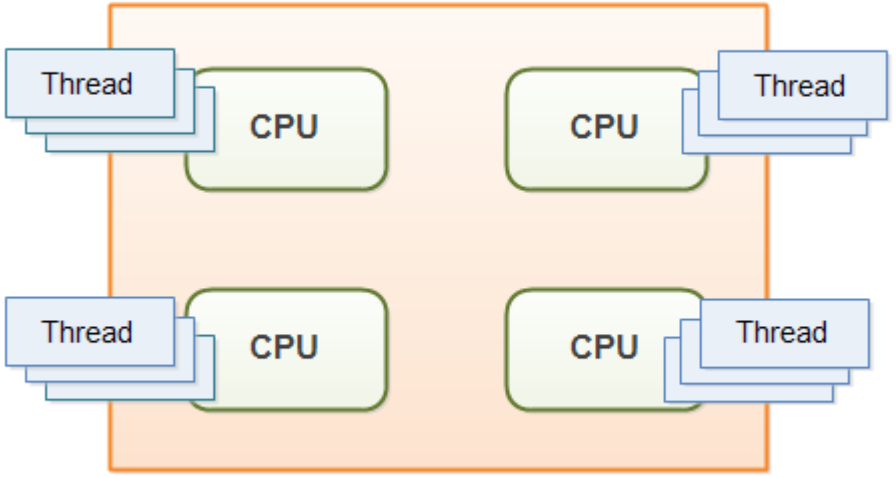
- **Multithreading,** o qual quer dizer que um programa pode ter múltiplas threads (processos independentes).
- Uma thread (fio, na tradução literal) pode ser vista como uma CPU executando o programa. Múltiplas threads, várias CPUs.

Uma breve história...

- Multithreading



Múltiplas Threads



“If a thread reads a memory location while another thread writes to it, what value will the first thread end up reading? The old value? The value written by the second thread? Or a value that is a mix between the two? Or, if two threads are writing to the same memory location simultanously, what value will be left when they are done? The value written by the first thread? The value written by the second thread? Or a mix of the two values written?”

Java Multithreading

- Umas das primeiras linguagens de alto nível a tornar a programação de threads “fácil”;
- Desde o início, possui capacidade para programação em múltiplas threads (**Multithreading**);
- Obviamente, os problemas da programação em Multithreading persistem, mesmo no Java. Cabe ao desenvolvedor evitá-los;

Conteúdo da Disciplina

- A disciplina fará um misto entre:
  - Prática em programação concorrente em Java
  - Teoria de programação concorrente, segunido o livro do Ben-Ari

Concorrência em Java – Conteúdo

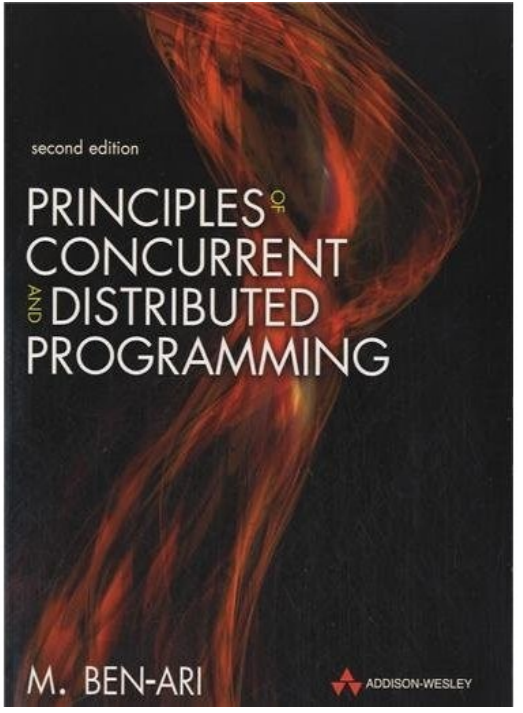
- Criação de Threads;
- Condições de Corrida e Seção Crítica;
- Segurança e Compartilhamento;
- Segurança e Imutabilidade;
- Modelo de Memória;
- **Synchronized**
- **ThreadLocal**
- **Thread Signaling**

Concorrência em Java - Conteúdo

- **Deadlock**
- **Prevenção do Deadlock**
- **Starvation e Fairness**
- Monitor aninhado
- **Slipped Conditions**

Concorrência – Teoria Ben Ari

- 1 - O que é programação concorrente - concorrência, multitarefa, terminologia e desafios.
- 2 - Abstrações e execução concorrente
- 2 - Justificativa da abstração, interleaving e estados atômicos
- 3 - O problema da seção crítica
- 4 - Algoritmos avançados de seção crítica (**TALVEZ**)
- 5 - Semáforos
- 6 - Monitores
- 7 - Consenso



Benefícios de Multithreading

- Melhor utilização dos recursos;
- Em algumas situações, o projeto de programação é mais simples;
- Programas mais responsivos.

Melhor utilização dos recursos

- Imagine uma aplicação que **lê(read)** e **processa(process)** dois arquivos. A operação de leitura leva 5 segundos e a operação de processamento leva dois. Sua execução sequencial seria:

5 seconds reading file A

2 seconds processing file A

5 seconds reading file B

2 seconds processing file B

---

14 seconds total

Melhor utilização dos recursos

- E se invertermos a ordem das operações? Ao colocar as duas leituras no início, aproveitamos o tempo ocioso do processador.

5 seconds reading file A

5 seconds reading file B + 2 seconds processing file A

2 seconds processing file B

---

12 seconds total

Projeto de programação simples

- A aplicação anterior feita em uma única thread necessita que o programador rastreie as operações de leitura e processamento, pra cada arquivo.
- Você pode implementar o mesmo programa fazendo uso de duas Threads, uma para cada arquivo. Sendo assim, cada thread se preocupa apenas **com o seu arquivo.**

Exercício

- Implemente a solução anterior com uma única thread e depois com duas threads.
- Use sleep para simular tempos de leitura e processamento de arquivos.

Programas mais responsivos

- Imagine uma aplicação servidora (**single thread**) em alguma porta esperando requisições:

```
while(server is active){
   listen for request
   process request
 }
```

Programas mais responsivos

- Quando a requisição chega, ele lida com a requisição separadamente e volta a escutar novas requisições.
- Caso a requisição demore muito tempo, o servidor não estará escutando, logo novas requisições deverão esperar.
- Qual seria a solução?

Programas mais responsivos

- Um design alternativo é deixar que uma thread trate em separado cada requisição, deixando o servidor livre para escutar as próximas requisições (o mesmo também acontece em aplicações desktop).

while(server is active){

listen for request

hand request to worker thread

}

Custos do Multithreading

Nem tudo são benefícios. Obviamente, existem custos na abordagem multithreading. Por exemplo...

- **Design mais complexo**: apesar de algumas partes de um programa multithread serem mais simples que um programa singlethread, outras partes podem ser mais complexas…
  - Múltiplas threads acessando dados compartilhados;
  - Iteração entre threads (troca de dados entre elas);
  - Erros de sincronização são complicados de debugar.

Custos do Multithreading

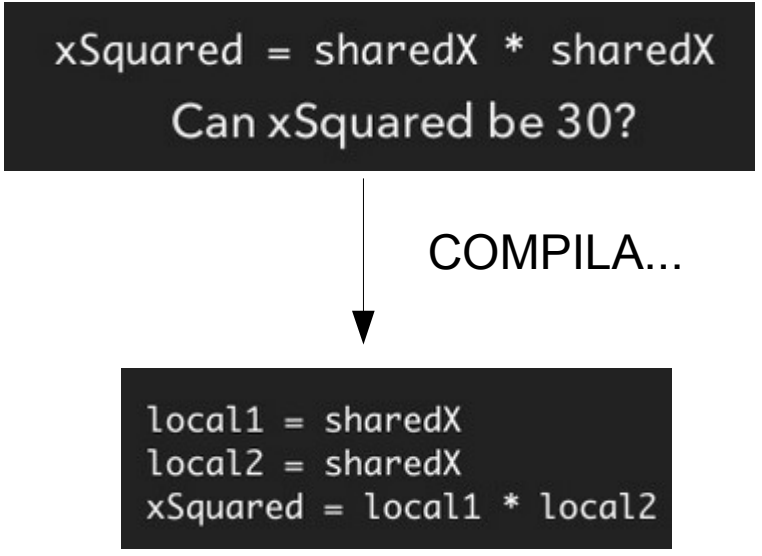
- **Sobrecarga na troca de contexto:**
  - Quando a CPU pausa a execução uma thread para iniciar outra, o estado da thread original deve ser salvo e isso gera overhead (context switch).
  - Mais: **http://en.wikipedia.org/wiki/Context_switch**
- **Consumo de recursos aumentado:**
  - CPU
  - Memória para guardar o estado
  - Recursos dentro do sistema operacional
  - Teste: crie um programa que inicia 1000 threads, e veja o consumo de memória.

Desafios

- Desafios da programação concorrente.
  - Sincronização das tarefas
  - Segurança dos dados
  - Acesso a região crítica de código
  - Veracidade dos dados
  - Paradigmas de programação

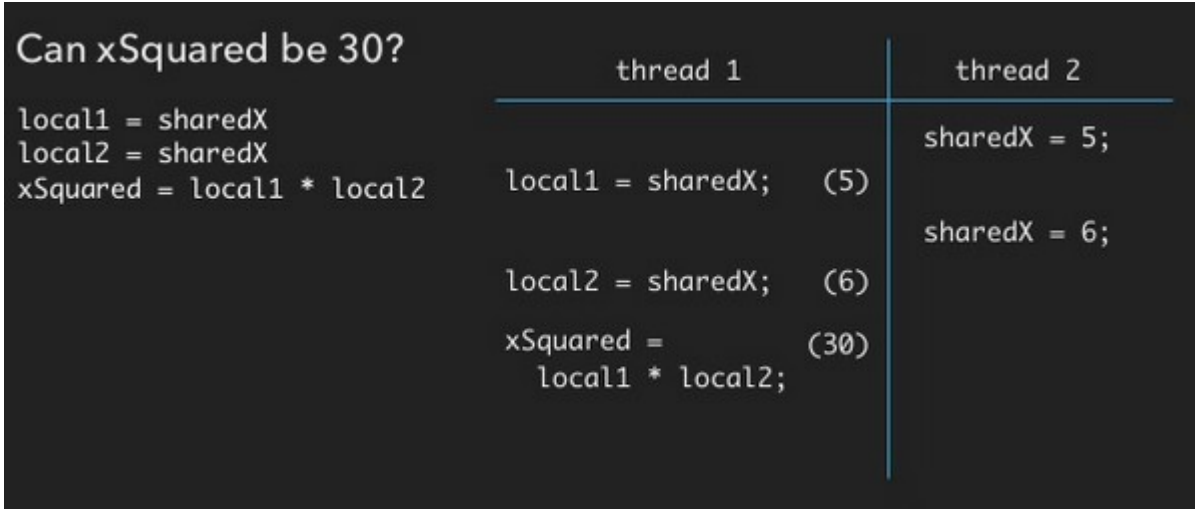
Desafios

- Seja sharedX uma variável compartilhada entre duas threads…
- Seja xSquared a variável local de uma das threads.



Desafios

- O que realmente pode acontecer:



Disciplina - Concorrência

- 2 provas teóricas [0-10]
- 1 seminário (primeira parte) [0-10]
  - O professor escolhe duas linguagens de programação modernas;
  - Explique, com exemplos práticos simples (+3), como a concorrência e o paralelismo é tratado nessas linguagens (ponto extra para +4 exemplos).
  - Apresentação de 15 a 20 minutos
  - Exemplos práticos devem ser enviados a parte (código fonte).
  - Lista de linguagens: https://en.wikipedia.org/wiki/Concurrent_computing
- 1 trabalho prático (segunda parte) [0-10]
- Média final: Soma tudo e divide por 4.

Exercício

- Implemente um programa em Java, o qual faz uso de uma thread. Inicie a thread e imprima algo em sua execução.
- Dispare 5 threads do mesmo programa e compare o tempo de execução com uma abordagem sequencial, onde o mesmo programa, sem threads, é executado 5 vezes. Você deve capturar o tempo inicial de execução e o final. O seu programa pode apenas executar um sleep, simulando uma computação custosa.

Referências

- http://tutorials.jenkov.com/java-concurrency/index.html
- Livro do Ben-Ari, capítulo 1.