Instituto de computação uff

Threads

Leonardo Gresta Paulino Murta

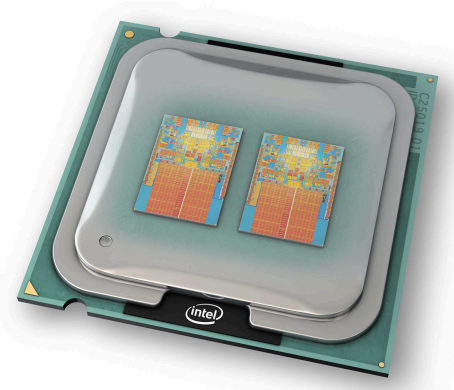
leomurta@ic.uff.br

Aula de hoje

- Estudaremos a execução em paralelo de programas em Java por meio de Threads

Por que usar Threads?

- Threads permitem processamento paralelo
  - Podemos **rodar mais de uma tarefa ao mesmo tempo**
  - Podemos tirar proveito dos vários processadores do nosso computador para **rodar mais rápido uma tarefa**



Processos vs. Threads

Processos

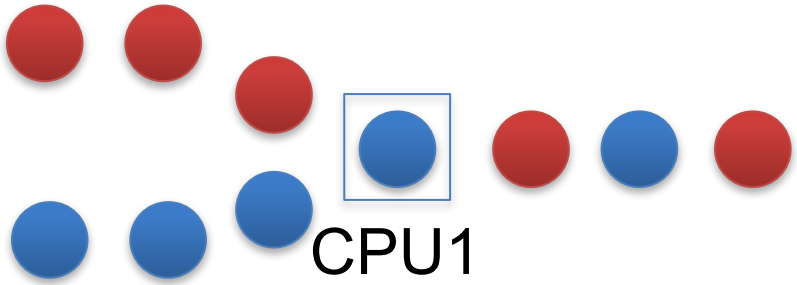
- Ambiente autocontido
- Espaço próprio de memória
- Cada aplicação que roda no sistema operacional é um processo em separado
- A JVM roda seu programa Java em um único processo

Threads

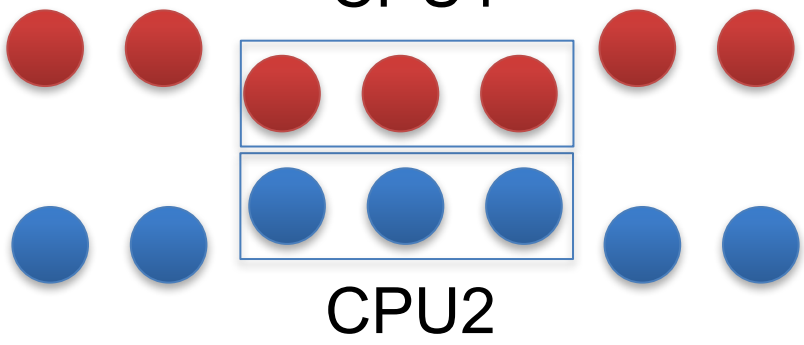
- São processos leves
- Compartilham recursos
- Um processo é composto por uma ou mais threads
- O seu programa Java inicia em uma thread principal mas pode abrir threads adicionais

Execução de Threads

Computador single core
(compartilhamento de tempo)



Computador dual core
(paralelismo real)



Thread 1 Thread 2

Threads em Java

- Interface Runnable
  - Permite criar classes que podem ser executadas em threads separadas
  - Contem um único método, que deve ser implementado com o código da tarefa: ***run()***
- Classe Thread
  - Controla a criação e execução de threads
  - Recebe um objeto Runnable como parâmetro
  - Contém o método ***start()***, que chama o método run() do objeto Runnable em uma thread em separado

Exemplo

```
public class Tarefa implements Runnable {

  private final String nome;

  public Tarefa(String nome) {
    this.nome = nome;
  }

  @Override
  public void run() {
    for (int i = 0; i <= 100; i = i + 20) {
      System.out.println("Tarefa " + nome + ": " + i + "%");
    }
  }
}
```

Exemplo

- Criação das threads

```
Thread t1 = new Thread(new Tarefa("A"));
Thread t2 = new Thread(new Tarefa("B"));

t1.start();
t2.start();
```

Exemplo

Resultado da primeira execução

```
Tarefa A: 0%
Tarefa A: 20%
Tarefa B: 0%
Tarefa A: 40%
Tarefa A: 60%
Tarefa B: 20%
Tarefa A: 80%
Tarefa B: 40%
Tarefa A: 100%
Tarefa B: 60%
Tarefa B: 80%
Tarefa B: 100%
```

Resultado da segunda execução

```
Tarefa B: 0%
Tarefa B: 20%
Tarefa B: 40%
Tarefa A: 0%
Tarefa A: 20%
Tarefa A: 40%
Tarefa A: 60%
Tarefa A: 80%
Tarefa A: 100%
Tarefa B: 60%
Tarefa B: 80%
Tarefa B: 100%
```

Mais métodos da classe Thread

- static void sleep(long millis)
  - Pausa a thread corrente em *milis* milissegundos
- void join()
  - Aguarda a outra thread terminar, bloqueando a thread corrente

O que acontece aqui?

```
Thread t1 = new Thread(new Tarefa("A"));
Thread t2 = new Thread(new Tarefa("B"));

t1.start();
Thread.sleep(1);
t2.start();
```

O que acontece aqui?

```
Thread t1 = new Thread(new Tarefa("A"));
Thread t2 = new Thread(new Tarefa("B"));

t1.start();
Thread.sleep(1);
t2.start();
```

```
Tarefa A: 0%
Tarefa A: 20%
Tarefa A: 40%
Tarefa A: 60%
Tarefa A: 80%
Tarefa A: 100%
Tarefa B: 0%
Tarefa B: 20%
Tarefa B: 40%
Tarefa B: 60%
Tarefa B: 80%
Tarefa B: 100%
```

O que acontece aqui?

```
Thread t1 = new Thread(new Tarefa("A"));
Thread t2 = new Thread(new Tarefa("B"));

t1.start();
t2.start();

System.out.println("Fim da thread principal.");
```

O que acontece aqui?

```
Thread t1 = n
Thread t2 = n

t1.start();
t2.start();

System.out.pr                                    .");
```

```
Fim da thread principal.
Tarefa B: 0%
Tarefa A: 0%
Tarefa B: 20%
Tarefa B: 40%
Tarefa B: 60%
Tarefa B: 80%
Tarefa B: 100%
Tarefa A: 20%
Tarefa A: 40%
Tarefa A: 60%
Tarefa A: 80%
Tarefa A: 100%
```

O que acontece aqui?

```
Thread t1 = new Thread(new Tarefa("A"));
Thread t2 = new Thread(new Tarefa("B"));

t1.start();
t2.start();

t1.join();
t2.join();

System.out.println("Fim da thread principal.");
```

O que acontece aqui?

```
Thread t1 = n
Thread t2 = n

t1.start();
t2.start();

t1.join();
t2.join();

System.out.pr                                    .");
```

```
Tarefa A: 0%
Tarefa A: 20%
Tarefa B: 0%
Tarefa A: 40%
Tarefa A: 60%
Tarefa B: 20%
Tarefa A: 80%
Tarefa B: 40%
Tarefa A: 100%
Tarefa B: 60%
Tarefa B: 80%
Tarefa B: 100%
Fim da thread principal.
```

Modificador synchronized

- Permite delimitar regiões críticas do programa
- Garante que um método nunca será executado por mais de uma thread em paralelo

Exemplo

```
public class Contador implements Runnable {

  private int numero = 0;
  private final Set<Integer> numeros = new HashSet<>();
  private static final int MAX = 10000;

  public int proximo() {
    return numero++;
  }

  public boolean continua() {
    return numero < MAX;
  }

...
```

Exemplo

```
...

  @Override
  public void run() {
    while (continua()) {
      int proximoNumero = proximo();
      if (!numeros.add(proximoNumero)) {
        System.out.println("Colisão: " + proximoNumero);
      }
    }
  }
}
```

O que acontece aqui?

```
Runnable contador = new Contador();

for (int i = 0; i < 10; i++) {
  Thread t = new Thread(contador);
  t.start();
}
```

O que acontece aqui?

```
Runnable contador = new Contador();

for (int i = 0; i <
  Thread t = new Thr
  t.start();
}
```

```
Colisão: 151
Colisão: 718
Colisão: 874
Colisão: 650
Colisão: 1034
Colisão: 1113
Colisão: 1306
Colisão: 151
Colisão: 1578
Colisão: 1754
Colisão: 2006
...
```

E agora?

```
public class Contador implements Runnable {

  private int numero = 0;
  private final Set<Integer> numeros = new HashSet<>();
  private static final int MAX = 10000;

  public synchronized int proximo() {
    return numero++;
  }

  public boolean continua() {
    return numero < MAX;
  }

...
```

E agora?

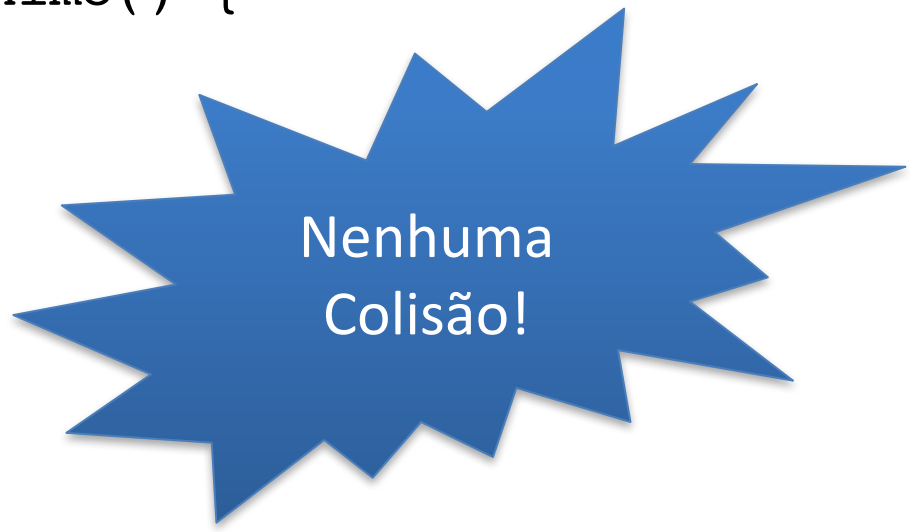
```
public class Contador implements Runnable {

  private int numero = 0;
  private final Set<Integer> numeros = new HashSet<>();
  private static final int MAX = 10000;

  public synchronized int proximo() {
    return numero++;
  }

  public boolean continua() {
    return numero < MAX;
  }

...
```



Exercício

- Faça uma implementação recursiva (e ineficiente) de Fibonacci:

$$fib(n) = \begin{cases} 0, & n = 0 \\ 1, & n = 1 \\ fib(n-1) + fib(n-2), & x \geq 2 \end{cases}$$

- Paralelize essa implementação usando duas threads
  - Ficou mais rápido?

Instituto de computação uff

Threads

Leonardo Gresta Paulino Murta

leomurta@ic.uff.br