Threads

Sistemas Operacionais

Prof. Pedro Ramos
pramos.costar@gmail.com

Pontifícia Universidade Católica de Minas Gerais
ICEI - Departamento de Ciência da Computação

AULA ANTERIOR: ESCALONAMENTO

- Preemptivo vs não preemptivo
- Objetivos do escalonamento:
  - Minimizar a média de tempo de resposta
  - Maximimzar vazão
  - Compartilhar CPU igualmente
  - Outros ?
- **Algoritmos de escalonamento:**
  - Considerar os tradeoffs na escolha da política
  - FCFS
  - RR
  - SJF/SRTF
  - MLQ
  - Loteria

THREADS

- O QUE SÃO THREADS?

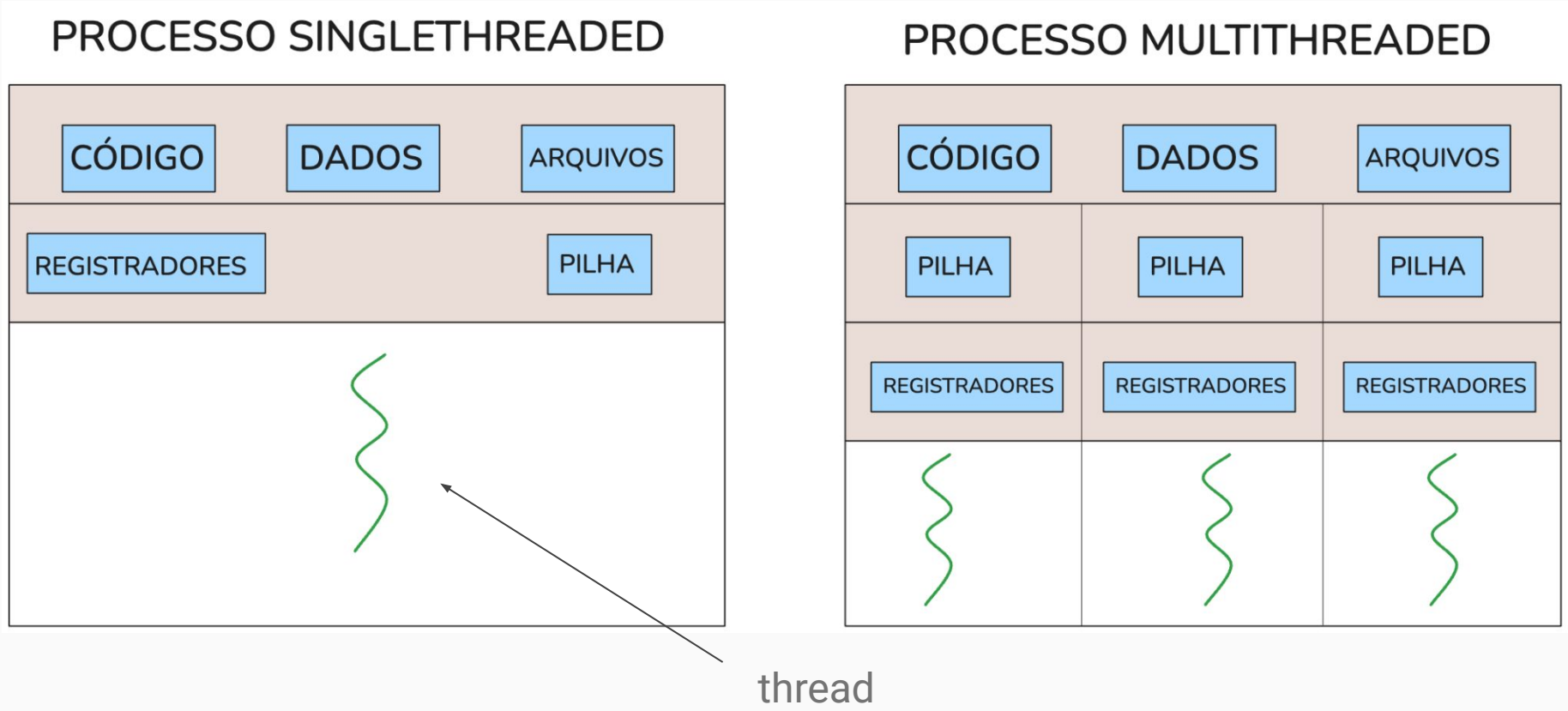
- ONDE IMPLEMENTAR THREADS?
  - NO KERNEL?
  - EM UM PACOTE (LIB) NÍVEL APLICAÇÃO DE USUÁRIO?

- COMO ESCALONAR THREADS (E PROCESSOS) NA CPU?

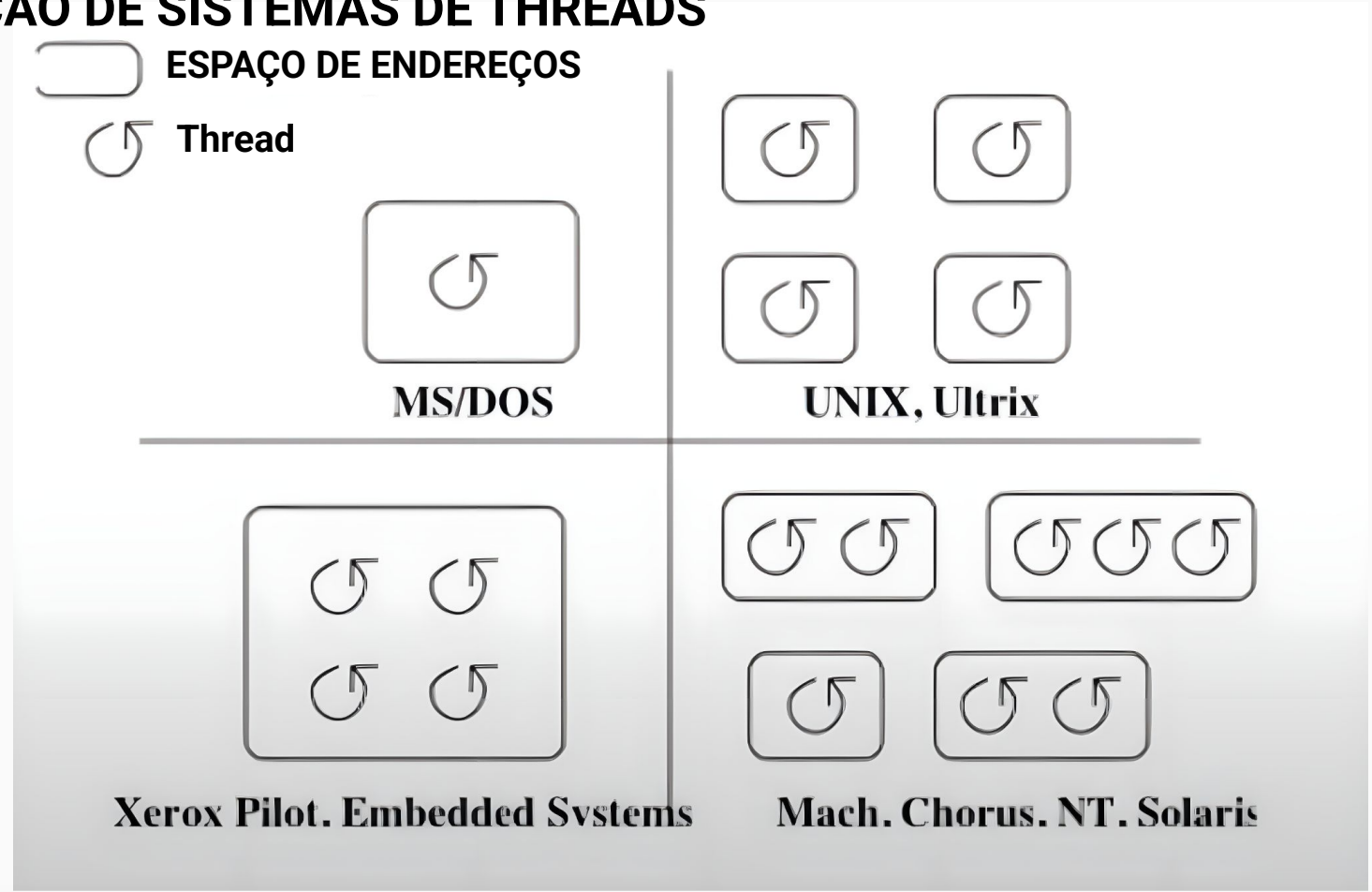
PROCESSOS vs THREADS

- Um processo define um espaço de endereços, texto, recursos, etc.
- Uma **thread** define um **único fluxo de execução sequencial dentro de um processo (PC, pilha, registradores).**
- Threads extraem o FIO DE CONTROLE (*thread of control*) da informação no processo.
- Threads estão vinculadas a um único processo e tem acesso ao espaço de endereços desse processo.
- Cada processo pode ter várias threads de controle dentro de si.
  - O **espaço de endereços é compartilhado** entre as threads
  - **Nenhuma chamada de sistema** é requerida para que threads cooperem
  - Mais simples que enviar mensagens entre processos

THREADS


PROCESSO SINGLETHREADED
CÓDIGO
DADOS
ARQUIVOS
REGISTRADORES
PILHA
thread
PROCESSO MULTITHREADED
CÓDIGO
DADOS
ARQUIVOS
PILHA
PILHA
PILHA
REGISTRADORES
REGISTRADORES
REGISTRADORES

CLASSIFICAÇÃO DE SISTEMAS DE THREADS

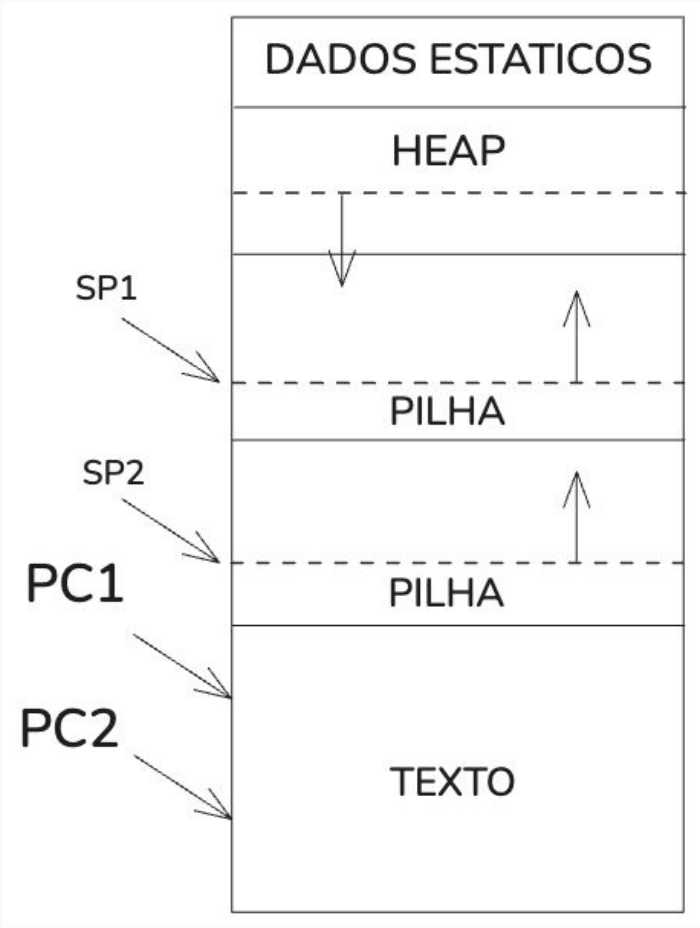


THREADS

```
main()
     global in, out, n, buffer[n];
     in=0; out=0;
     fork_thread(produtor());
     fork_thread(consumidor());
end

produtor()
     repetir
          nextp = novo item produzido
          while (in+1  mod n == out) faça: nada
          buffer[in] = nextp; in = (in + 1) mod n
end

consumidor()
     repetir
          while (in == out) faça: nada
          nextc = buffer[out]; out = (out + 1) mod n
          consome item nextc
```



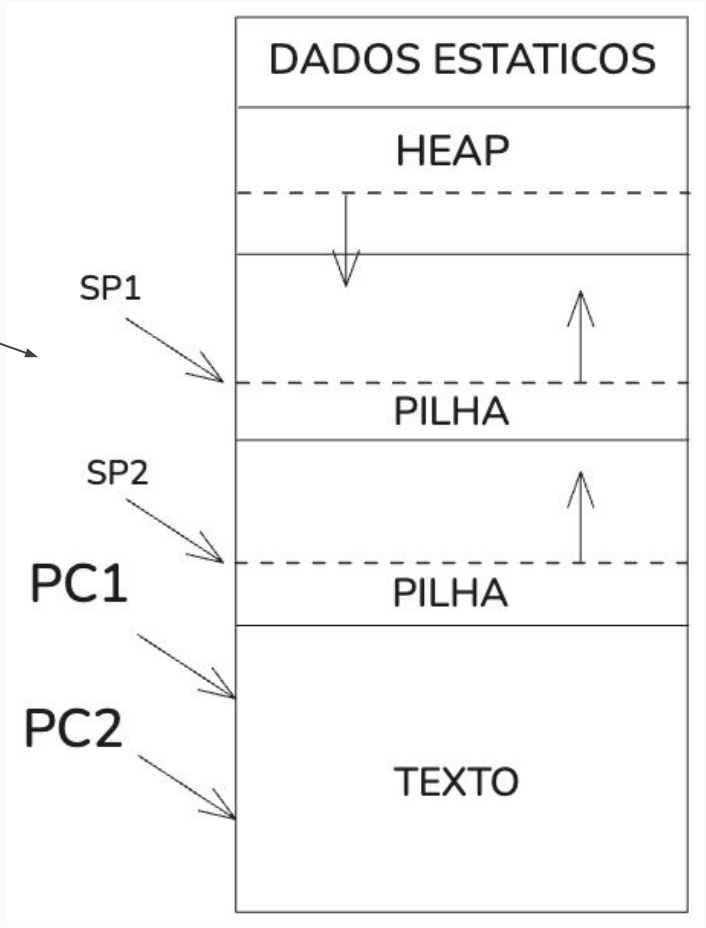
THREADS

```
main()
     global in, out, n, buffer[n];
     in=0; out=0;
     fork_thread(produtor());
     fork_thread(consumidor());
end

produtor()
     repetir
          nextp = novo item produzido
          while (in+1  mod n == out) faça: nada
          buffer[in] = nextp; in = (in + 1) mod n
end

consumidor()
     repetir
          while (in == out) faça: nada
          nextc = buffer[out]; out = (out + 1) mod n
          consome item nextc
```

ONDE GUARDAR A INFORMAÇÃO DAS THREADS? → PCB



THREADS DO KERNEL

- QUEM GERENCIA AS THREADS?
  - Podem ser criadas com uma chamada de sistema direta pro Kernel, ou por bibliotecas de threads (código de usuário)

- Uma thread do Kernel é um processo leve, uma thread que o **SO conhece**.
- Trocar entre threads de kernel do mesmo processo é uma troca de contexto pequena (otimizada pelo SO)
  - **Troca os valores dos registradores**, PC (program counter), e SP (stack pointer)
  - **Informação de memória** (quais páginas aquele processo tem acesso) **não precisa mudar**, pois o espaço de endereços é o mesmo.

THREADS DO KERNEL

- O KERNEL gerencia e escalona as threads (e processos) com os mesmos algoritmos que vimos.

  TROCA ENTRE THREADS DE KERNEL É MAIS RÁPIDA QUE A TROCA ENTRE PROCESSOS!

THREADS DO USUÁRIO

- O SO não conhece threads de nível usuário.
- O SO só conhece os processos que contém as threads.
- O SO escalona o processo, não as threads dentro do processo.
- O programador usa uma biblioteca de threads para gerenciar threads (criar e deletar, sincronizar, e escalonar).
- NÃO PODEM RODAR EM MÚLTIPLAS CPU's, pois o SO enxerga somente 1 processo.

THREADS DO USUÁRIO - VANTAGENS

- **Por que fazer threads de usuário?**

- Operações assíncronas em um mesmo programa
- *Não tem troca de contexto*
- Threads do usuário podem ser escalonadas pelo programador
  - **+ Flexibilidade na escolha do algoritmo** pra cada processo/threads
  - **A thread tem o controle de ceder (*yield*):**
    dizer para a CPU que outras threads podem executar no seu lugar
- Não requerem chamadas de sistema

- Mais rápidas que threads do kernel

THREADS DO USUÁRIO - DESVANTAGENS

- **Sem paralelismo real**
  - Várias threads do usuário no mesmo processo não podem executar ao mesmo tempo em CPU's diferentes
- Como o SO não conhece as threads, pode tomar decisões ruins:
  - Pode rodar um processo cheio de threads ociosas.
  - Se uma thread do usuário faz I/O, o processo inteiro espera.
  - Resolvir isso requer comunicação entre Kernel e gerenciador de threads do usuário
- O SO escalona o processo independente do número de threads do usuário que ele tem,
- Mas para threads do kernel, quanto mais threads um processo criar, mais fatias de tempo o SO vai dedicar a ele.

THREADS DO USUÁRIO - DESVANTAGENS

- Sem paralelismo real
  - Várias threads do usuário no mesmo processo não podem executar ao mesmo tempo em CPU's diferentes
- Como o **SO não conhece as threads**, pode tomar **decisões ruins**:
  - Pode rodar um processo cheio de **threads ociosas**.
  - Se uma thread do usuário faz I/O, o processo inteiro espera.
  - Resolver isso requer **comunicação entre Kernel e gerenciador de threads do usuário**
- O SO escalona o processo independente do número de threads do usuário que ele tem,
- Mas para threads do kernel, quanto mais threads um processo criar, mais fatias de tempo o SO vai dedicar a ele.

THREADS DO USUÁRIO - DESVANTAGENS

- Sem paralelismo real
  - Várias threads do usuário no mesmo processo não podem executar ao mesmo tempo em CPU's diferentes
- Como o SO não conhece as threads, pode tomar decisões ruins:
  - Pode rodar um processo cheio de threads ociosas.
  - Se uma thread do usuário faz I/O, o processo inteiro espera.
  - Resolvir isso requer comunicação entre Kernel e gerenciador de threads do usuário
- O SO escalona o processo **independente do número de threads do usuário que ele tem,**
  ...mas para **threads do kernel, quanto mais threads um processo criar, mais fatias de tempo** o SO vai dedicar a ele.

M:N THREADING

Solução - HÍBRIDA

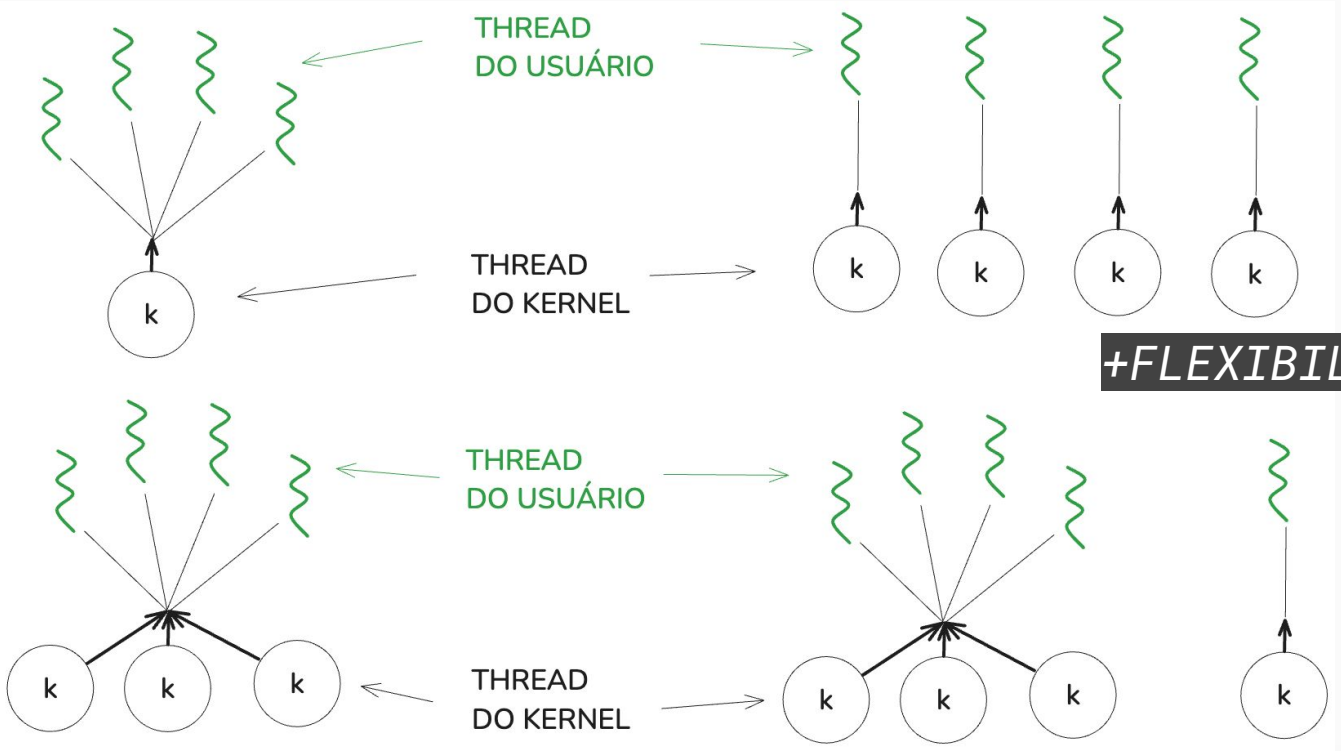
M threads do usuário são mapeadas em N threads do kernel.

⇒ SOLARIS, Windows, JVM

M:N THREADING

Muitos-pra-um, um-pra-um, muitos-pra-muitos, e 2 níveis

N:1 1:1 M:N



BIBLIOTECAS DE THREADS

- Biblioteca de threads provê ao programador uma API para criar e gerenciar threads.

2 formas de implementar:

- Threads do usuário (biblioteca inteira em espaço de usuário)
- Threads do kernel (biblioteca tem suporte no SO)

pthreads

- Provê threads NÍVEL-USUÁRIO ou NÍVEL-KERNEL
- Padrão POSIX de API para criação e sincronização de threads
- API especifica o comportamento das thread, mas a implementação fica a cargo do programador
- Comum em sistemas UNIX (Solaris, Linux, Mac OS)
- WIN32 threads - similar

THREADS NO JAVA

- Threads do Java são gerenciadas pela JVM

- É implementado usando o modelo de threads do SO que executa a JVM

- Podem ser criadas:
  - Estendendo a classe **Thread**
  - Implementando a interface **Runnable**
  - Disparando threads virtuais com **Thread.ofVirtual()**
  - Construtos modernos (java 21+) são implementados em cima desses.

EXEMPLOS

Pthreads:

```
pthread_attr_init(&attr);
pthread_create(&tid, &attr, sum, &param);
```

Win32 threads:

```
ThreadHandle = CreateThread(NULL, 0, Sum, &Param, 0, &ThreadID);
```

Java Threads:

```
Sum sumObject = new Sum();
Thread t = new Thread(new Summation(param, SumObject));
t.start();
```

THREADS: RESUMO

- Thread: Um fluxo único de execução dentro de um processo.
- Trocar entre threads do usuário é mais rápido que trocar entre threads do Kernel, pois não há troca de contexto.
- Threads do usuário resultam em decisões ruins do Kernel, que resulta em execução de processos mais lenta do que seria caso threads do Kernel fossem utilizadas.
- Existem muitos algoritmos de escalonamento. Selecionar um algoritmo é uma decisão de política e deve ser baseado nas características do processo em execução e os objetivos do sistema operacional (minimizar tempo de resposta, maximizar vazão, etc...)

PERGUNTAS?

REFERÊNCIAS

- **TANENBAUM, Andrew.** Sistemas operacionais modernos.
- **SILBERSCHATZ, Abraham et al.** Fundamentos de sistemas operacionais: princípios básicos.
- **MACHADO, Francis; MAIA, Luiz Paulo.** Arquitetura de Sistemas Operacionais.
- **CARISSIMI, Alexandre et al.** Sistemas operacionais.