Prof. Esp. Thalles Canela

- **Graduado:** Sistemas de Informação - Wyden Facimp
- **Pós-graduado:** Segurança em redes de computadores - Wyden Facimp

- **Consultor de Tecnologia - [aXR6] Cyber Security e NtecSoftware**
- **Professor no Senac (contratado)**
- **Professor na Wyden Facimp (contratado)**
  - **Pós-graduação:** Segurança em redes de computadores - Wyden Facimp
- **Professor na Wyden Facimp (Efetivado)**
  - **Graduação:** Análise e desenvolvimento de sistemas - Wyden Facimp

Redes sociais:

- **Linkedin:** https://www.linkedin.com/in/thalles-canela/
- **YouTube:** https://www.youtube.com/aXR6CyberSecurity
- **Facebook:** https://www.facebook.com/axr6PenTest
- **Instagram:** https://www.instagram.com/thalles_canela
- **Github:** https://github.com/ThallesCanela
- **Github:** https://github.com/aXR6
- **Twitter:** https://twitter.com/Axr6S

Concorrência, Paralelismo, Processos, Threads, programação síncrona e assíncrona

Concorrência, paralelismo, processos, threads, programação síncrona e assíncrona, são assuntos que permeiam o dia a dia dos desenvolvedores. A ideia desse artigo é descomplicar um pouco o que esses conceitos significam e como eles se relacionam.

Monotarefa vs Multitarefa

Os primeiros sistemas operacionais suportavam a execução de apenas uma tarefa por vez. Nesse modelo, o processador, a memória e os periféricos ficavam dedicados a uma única tarefa. Tínhamos um fluxo bem linear, como pode ser visto nesse diagrama:

Monotarefa vs Multitarefa

Apenas no término da execução de uma tarefa que outra poderia ser carregada na memória e então executada.

O problema desse modelo é que enquanto o processo realizava uma operação de I/O para, por exemplo, ler algum dado do disco, o processador ficava ocioso. Ademais, uma operação do processador é infinitamente mais rápida que qualquer uma de leitura ou escrita em periféricos.

Para se ter ideia, quando falamos de uma operação que a CPU executa, lidamos com nanosegundos, enquanto em uma operação de rede consideramos milesegundos.

Monotarefa vs Multitarefa

A solução: permitir ao processador suspender a execução de uma tarefa que estivesse aguardando dados externos ou algum evento e passar a executar outra tarefa.

Alternativa: Em outro momento de tempo, quando os dados estivessem disponíveis, a tarefa suspensa poderia ser retomada do ponto exato de onde ela havia parado.

O que acontece: Nesse modelo, mais de um programa é carregado na memória.

Mecanismo que permite a retirada de um recurso: preempção.

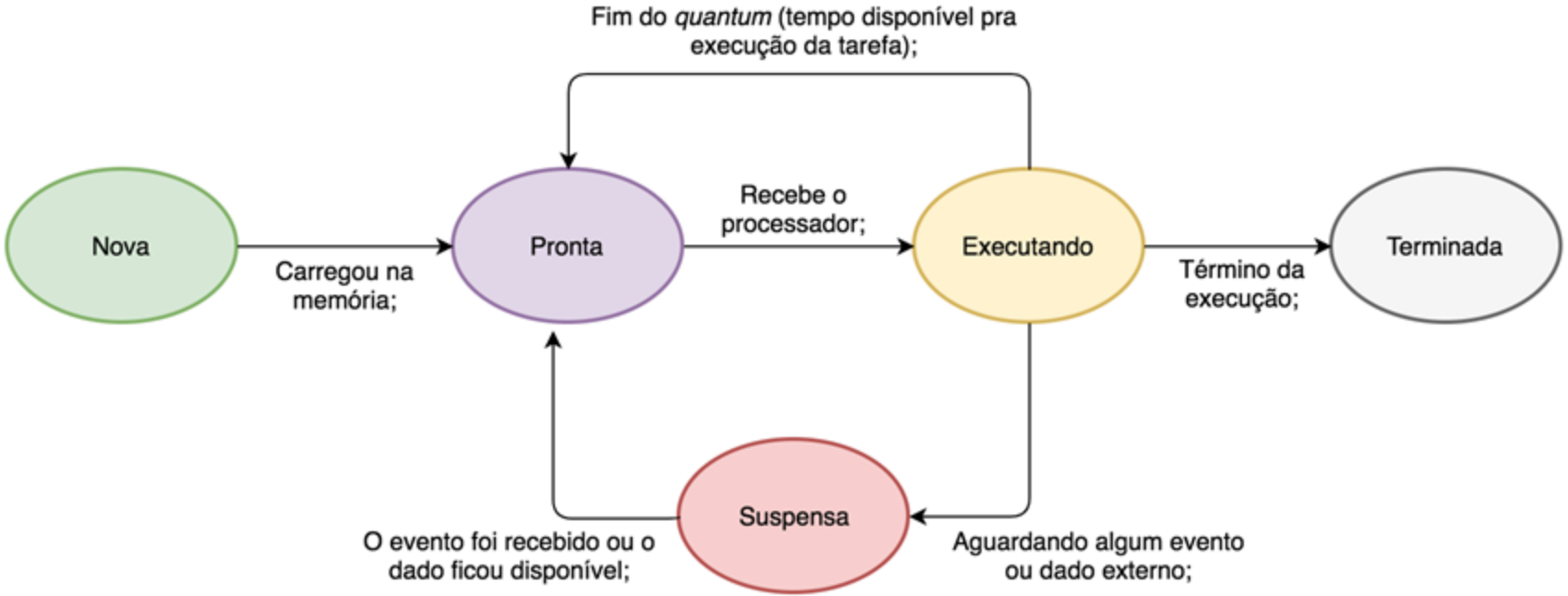
Monotarefa vs Multitarefa

Sistemas preemptivos são mais produtivos, ademais, várias tarefas podem estar em execução ao mesmo intervalo de tempo alternando entre si o uso dos recursos da forma mais justa que for possível. Nesse tipo de sistema as tarefas alteram de estado e contexto a todo instante.

Os estados de uma tarefa num sistema preemptivo:

- **Nova:** A tarefa está sendo criada (carregada na memória);
- **Pronta:** A tarefa está em memória aguardando a disponibilidade do processador para ser executada pela primeira vez ou voltar a ser executada (na hipótese de que ela foi substituída por outra tarefa, devido à preempção);
- **Executando:** O processador está executando a tarefa e alterando o seu estado;
- **Suspensa:** A tarefa não pode ser executada no momento por depender de dados externos ainda não disponíveis (dados solicitados à rede ou ao disco, por exemplo.);
- **Terminada:** A execução da tarefa foi finalizada e ela já pode sair da memória;

O diagrama de estado das tarefas com preempção de tempo:



O que é um processo?

Um processo pode ser visto como um container de recursos utilizados por uma ou mais tarefas.

Processos são isolados entre si (inclusive, através de mecanismos de proteção a nível de hardware), não compartilham memória, possuem níveis de operação e quais chamadas de sistemas podem executar.

Como os recursos são atribuídos aos processos, as tarefas fazem o uso deles a partir do processo.

O que é um processo?

Um processo é uma entidade ativa, é a “instância” de um programa (entidade passiva). Podemos fazer analogia com orientação a objetos onde o programa seria a estrutura da classe e um processo seria um objeto instância dessa classe.

O kernel do sistema operacional possui descritores de processos, denominados PCBs (Process Control Blocks) e eles armazenam informações referentes aos processos ativos e cada processo possui um identificador único no sistema, conhecido como PID (Process IDentifier).

As tarefas de um processo podem trocar informações com facilidade, pois compartilham a mesma área de memória.

O que é um processo?

No entanto, tarefas de processos distintos não conseguem essa comunicação facilmente, pois estão em áreas diferentes de memória. Esse problema é resolvido com chamadas de sistema do kernel que permitem a comunicação entre processos (IPC - Inter-Process Communication).

O que é uma thread?

Os processos podem ter uma série de threads associadas e as threads de um processo são conhecidas como threads de usuário, por executarem no modo-usuário e não no modo-kernel. Uma thread é uma “linha” de execução dentro de um processo. Cada thread tem o seu próprio estado de processador e a sua própria pilha, mas compartilha a memória atribuída ao processo com as outras threads “irmãs” (filhas do mesmo processo).

O núcleo (kernel) dos sistemas operacionais também implementa threads, mas essas são chamadas de threads de kernel (ou kernel-threads). Elas controlam atividades internas que o sistema operacional precisa executar/cuidar.

Concorrência e paralelismo

Concorrência é sobre lidar com várias coisas ao mesmo tempo e paralelismo é sobre fazer várias coisas ao mesmo tempo. Concorrência é um conceito mais a nível de software e paralelismo mais a nível de hardware.

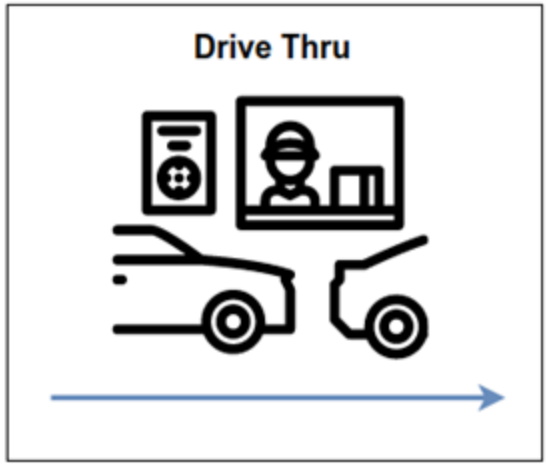
Concorrência é sobre a execução sequencial e disputada de um conjunto de tarefas independentes.

Sob o ponto de vista de um sistema operacional, o responsável por esse gerenciamento é o escalonador de processos.

Já sob o ponto de vista de concorrência em uma linguagem de programação como Go, por exemplo, o responsável é o scheduler interno da linguagem.

Escalonadores preemptivos (como é o caso dos sistemas operacionais modernos)

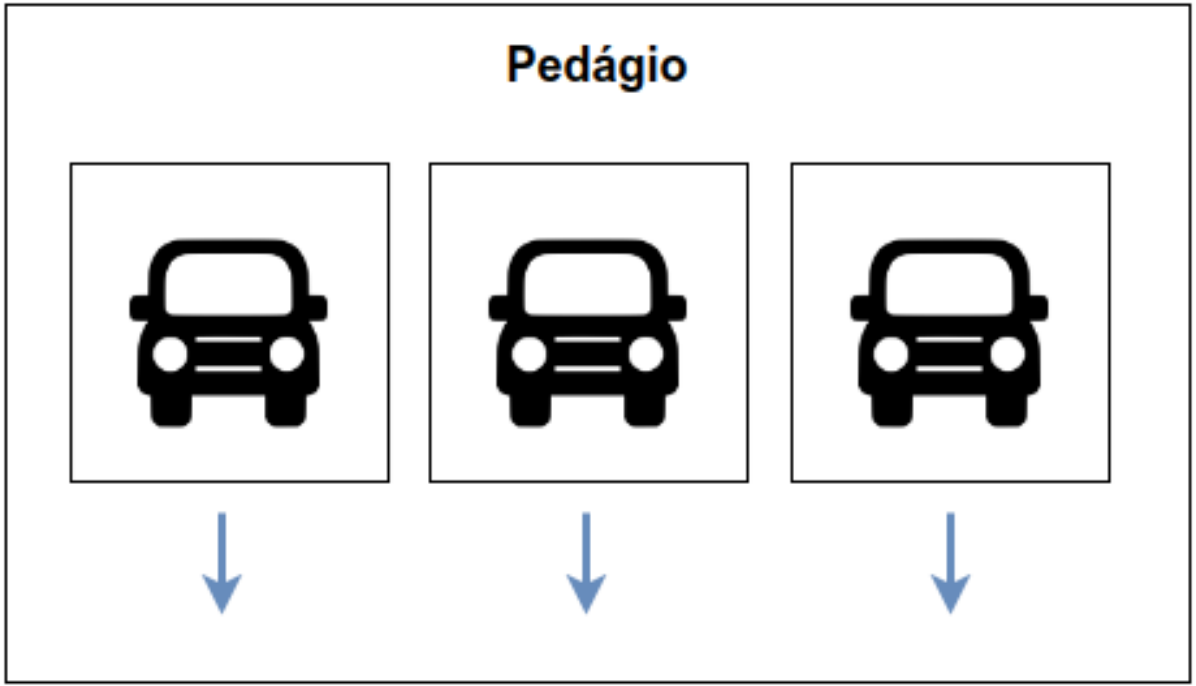
Favorecem a concorrência pausando e resumindo tarefas (no caso de sistemas operacionais estamos falando de processos e threads no que chamamos de trocas de contexto) para que todas tenham a oportunidade de serem executadas.



Paralelismo

Execução paralela de tarefas, ou seja, mais de uma por vez (de forma simultânea), a depender da quantidade de núcleos (cores) do processador. Quanto mais núcleos, mais tarefas paralelas podem ser executadas. É uma forma de distribuir processamento em mais de um núcleo.

Paralelismo é aquele pedágio que permite que carros progridam em diferentes fluxos simultaneamente:



Síncrono e assíncrono

Síncrono e assíncrono são modelos de programação que estão intimamente ligados ao fluxo de execução, eles determinam como o código será escrito e como ele rodará.

No modelo síncrono:

Uma operação precisa ser finalizada para que outra tenha a oportunidade de ser executada. É um modelo linear, previsível, onde a execução acontece etapa por etapa e ele é base padrão da maior parte das linguagens de programação.

No modelo assíncrono:

Uma operação não precisa esperar a outra ser finalizada, ao contrário disso, elas alternam o controle da execução entre si. É um modelo não previsível e que não garante a ordem da execução. É um modelo que favorece a concorrência.

Fazendo uma analogia:

N**o modelo síncrono:** se você precisa colocar roupas para lavar e lavar louças, primeiro você poderia colocar as roupas para lavar e esperaria o tempo que fosse necessário até finalizar, para só depois lavar as louças.

No modelo assíncrono: você poderia colocar as roupas na máquina de lavar roupas e já começaria a lavar as louças no mesmo instante, pois você não precisa ficar ocioso esperando a máquina de lavar processar seu resultado para desempenhar outra tarefa, você pode pegar o “resultado” da máquina de lavar em um momento futuro.