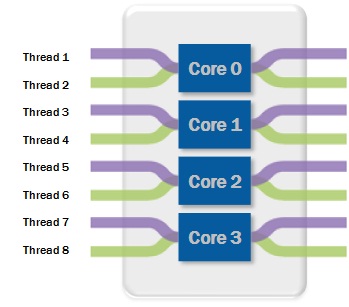
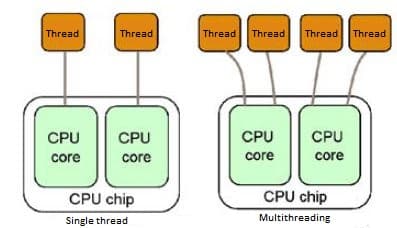
Para entendermos melhor sobre generators, precisamos primeiro entendermos alguns conceitos básicos da computação e das necessidades que deram origem a essas ferramentas.

# Threads

**O Que São Threads?**

* **Threads** são as menores unidades de execução que um sistema operacional pode gerenciar. Pense nelas como “linhas de execução” que podem ser usadas para dividir o trabalho em um programa.
* Um processo (por exemplo, um programa em execução) pode ter uma ou mais threads. Cada thread dentro de um processo compartilha o mesmo espaço de memória, mas executa de forma independente.
* **Exemplo na Vida Real**: Imagine que você está cozinhando. Se você preparar um prato por vez (uma tarefa por vez), é como usar uma única thread. Se você tiver várias pessoas (threads) ajudando na cozinha ao mesmo tempo, cada uma fazendo uma tarefa diferente, você está usando múltiplas threads.



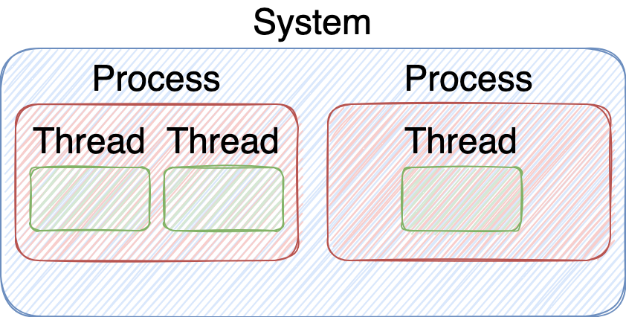
Alternativo as threads, temos os processos.

**O Que São Processos?**

**Processos** são instâncias independentes de programas em execução, cada um com seu próprio espaço de memória. Eles geralmente não compartilham recursos diretamente, e todos os processos podem armazenar inumeras threads.

* Imagine que um navegador da web (um processo) precisa carregar várias páginas ao mesmo tempo. Ele pode usar múltiplas threads:
  + Uma thread para gerenciar a interface do usuário.
  + Outra thread para carregar cada página da web.
  + Outra para executar scripts em segundo plano.

Nesse caso, todas essas threads compartilham o mesmo espaço de memória, permitindo ao navegador executar as tarefas de forma eficiente.

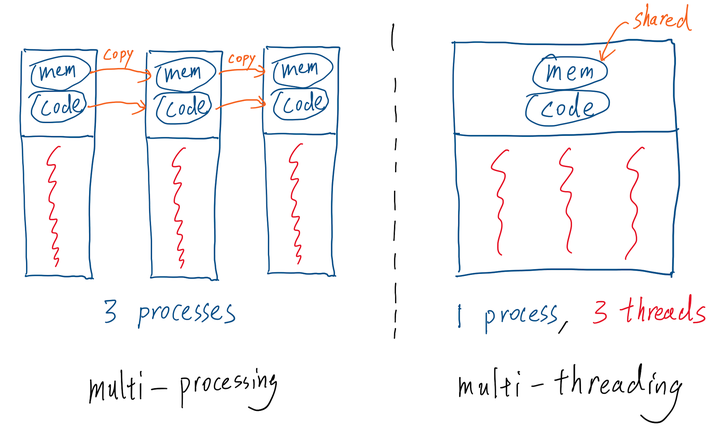


Ao iniciarmos aplicativos como o **navegador**, **IDE**, **agentes**, **aplicações**, **bancos de dados** e outros serviços, o sistema operacional cria um processo para cada um desses programas, fornecendo os recursos necessários para sua execução. Isso inclui **espaço de memória isolado**, **threads**, **contextos** e a gestão do **próprio ciclo de vida** do processo.

# Multithreading

**Multithreading** é uma técnica de programação que consiste na criação de múltiplas threads (fluxos de execução independentes) dentro de um único processo. Cada thread pode ser responsável por diferentes tarefas ou partes de uma tarefa mais ampla. Este método pode ser aplicado **tanto em contextos concorrentes quanto paralelos**. Em sistemas com **um único núcleo do processador, o multithreading facilita a concorrência** (alternância rápida entre threads para criar a ilusão de simultaneidade). Já em sistemas **multiprocessadores, ou com múltiplos núcleos**, o multithreading pode alcançar paralelismo real, com threads sendo executadas simultaneamente em núcleos distintos da CPU\*\*, otimizando o uso dos recursos e melhorando a eficiência e o desempenho do processo.

Para ilustrar o conceito de multithread, pense em seu **restaurante favorito**. Aqui, o **processo é o restaurante funcionando**, com o objetivo de **servir comida aos clientes**. Durante um horário de pico, como o almoço em um dia útil, as **threads são como os funcionários da cozinha**. Cada cozinheiro (thread) é **responsável por preparar um prato diferente simultaneamente**, acelerando o atendimento dos pedidos. Dessa forma, vários pratos são preparados ao mesmo tempo, aumentando a eficiência e diminuindo o tempo de espera dos clientes.



A partir disso, podemos entrar nos conceitos de concorrência e paralelismo.

# Concorrência

Imagine que você está preparando um churrasco sozinho. Você é responsável por organizar a geladeira, fazer os cortes de carne, preparar os vegetais para os amigos vegetarianos, fazer caipirinhas e gelar a cerveja. Você alterna entre essas tarefas, trabalhando um pouco em cada uma, apesar de ser responsável por todas elas.

Este cenário é um exemplo de concorrência, onde você está **gerenciando várias tarefas, mas não necessariamente trabalhando em mais de uma delas simultaneamente**. Você se alterna entre as tarefas, criando a impressão de que tudo está progredindo ao mesmo tempo.

Texto

Descrição gerada automaticamente com confiança médiaConcorrência é sobre lidar com muitas tarefas ao mesmo tempo, mas não de forma simultânea. É a habilidade de uma aplicação gerenciar múltiplas tarefas e instruções em segundo plano, mesmo que essas instruções não estejam sendo processadas ao mesmo tempo, ou executadas em outros núcleos do processador.

>>>>

# Paralelismo

Ainda estamos no exemplo do churrasco. Desta vez você tem amigos para ajudar: um corta a carne, outro acende a churrasqueira, outro gela a cerveja e mais um faz a caipirinha. Todas essas tarefas estão ocorrendo em paralelo, com cada pessoa responsável por uma parte do processo.

Isso ilustra o paralelismo. **Múltiplas tarefas** e instruções ocorrendo **simultaneamente**, executadas por **múltiplos núcleos de processadores**.

Diferentemente da concorrência, onde se trata de gerenciar várias tarefas ao mesmo tempo, mas com apenas uma ativa por vez, **o paralelismo envolve fazer várias coisas ao mesmo tempo.**

Texto

Descrição gerada automaticamenteO paralelismo é empregado em situações onde o desempenho e a eficiência são críticos, e há recursos suficientes, como múltiplos núcleos de CPU, para executar diversas tarefas simultaneamente.



>>>>

Agora perceba, concorrência se trata de 1 processo com multiplas threads, e paralelismo, diversos processos com diversas threads.

Também, precisamos entender um pouco agora sobre iteratores, portanto, voltamos ao nossos estudos em PHP.

# Iterator

Um **Iterator** em PHP é uma interface que fornece um modo de acessar sequencialmente os elementos de uma coleção (como um array ou um objeto) sem expor a estrutura interna dessa coleção. Ele é usado principalmente em loops, como foreach, para permitir a iteração sobre objetos de forma estruturada.

**Como o Iterator Funciona**

Em PHP, a interface Iterator define um conjunto de métodos que uma classe deve implementar para ser usada como um iterador:

1. **current()**: Retorna o elemento atual no iterador.
2. **key()**: Retorna a chave (ou índice) do elemento atual.
3. **next()**: Avança para o próximo elemento.
4. **rewind()**: Retorna o iterador para o seu primeiro elemento.
5. **valid()**: Verifica se o elemento atual é válido (existe ou não chegou ao fim).

Texto

Descrição gerada automaticamenteTexto

Descrição gerada automaticamenteEx:

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

Vamos agora dar continuida nos estudos com PHP e o que isso tem a ver com generators e fibers.

# generators

Generators e iterators em PHP estão intimamente relacionados porque **generators** são uma forma simplificada e eficiente de criar **iterators**. Em outras palavras, quando você usa um generator, você está criando automaticamente um iterator sem precisar implementar manualmente a interface Iterator e escrever métodos como current(), next(), key(), rewind(), e valid().

A ideia básica por trás dos geradores é que uma função não retorna um único valor, mas retorna uma *sequência de valores* , onde cada valor é emitido um por um. Ou em outras palavras, os geradores permitem que você implemente iteradores mais facilmente.

Texto

Descrição gerada automaticamenteExemplo:

Nesse exemplo, demonstra o uso de um **generator** em PHP para iterar de forma eficiente por uma sequência de números, de 1 a 1.000.000, sem carregar todos os números na memória de uma só vez.

* A função xrange é definida para gerar uma sequência de números de $start até $end, com um incremento de $step (o valor padrão de $step é 1).
* Em vez de retornar um array com todos os números, a função usa a palavra-chave yield para gerar cada número um por vez.
* **yield $i**: Cada vez que yield é chamado, a execução da função xrange é pausada e o valor de $i é retornado ao código que está iterando sobre ela. Quando o loop foreach pede o próximo valor, a execução da função continua de onde parou.
* o foreach itera sobre o generator xrange(1, 1000000).
* Em cada iteração, o número atual gerado por yield é atribuído a $num e impresso com echo.
* **Eficiência**: O grande benefício deste código é que ele não cria um array com 1.000.000 de números na memória. Em vez disso, cada número é gerado e usado conforme necessário, economizando uma quantidade significativa de memória.

Ou seja, os dados vão sendo gerados não todos de uma vez, mas 1 por 1 de forma sequencial e sob demanda.

Mas a frente, veremos como podemos pausar 1 função, começar outra, pausa-la novamente, começar outra e depois voltar aleatoriamente para outra (mas de forma sequencial), dando assim, a imprenssão de “multiprocessamento”.

# a real sobre concorrência e paralelismo no php

**Multithreading Real**: Para fazer multithreading real no PHP, você precisaria usar a extensão pthreads. No entanto, essa extensão não é suportada em ambientes de hospedagem comuns e só funciona no PHP rodando em modo CLI (linha de comando). Além disso, a configuração e o uso de threads em PHP podem ser complexos e não são ideais para todos os projetos.

**Paralelismo Real**: Da mesma forma, o PHP pode realizar paralelismo real usando processos (com pcntl\_fork() em sistemas Unix-like) ou threads (com pthreads), mas isso geralmente é limitado a casos específicos, como scripts em linha de comando que processam tarefas intensivas de CPU.

Sim, **Generators** e **Fibers** são usados para **simular concorrência** no PHP. E quanto ao paralelismo no PHP, ele é “inexistente” visto que PHP não executa tarefas simultaneamente em múltiplos núcleos de CPU sem o uso de extensões específicas como pthreads ou processos paralelos com pcntl\_fork().

# Palavra chave “yield”

palavra-chave **yield** é o coração dos **Generators** no PHP. Ela funciona de forma semelhante ao return, mas com uma diferença fundamental: em vez de parar a execução da função e retornar um valor, yield **pausa** a execução da função, salva o estado atual, e depois permite que a execução seja retomada posteriormente.

* Texto

  Descrição gerada automaticamenteQuando uma função que usa yield é chamada, ela retorna um **objeto Generator**. Você pode iterar sobre esse objeto para obter os valores produzidos por yield.

# Concorrência cooperativa

Os **Generators** no PHP não são usados diretamente para criar paralelismo ou concorrência, mas podem ser uma ferramenta útil para implementar **concorrência cooperativa**.

* **Concorrência Cooperativa**: Refere-se a um modelo de execução em que uma tarefa "cede" o controle de volta ao sistema, permitindo que outra tarefa seja executada. No PHP, os Generators, através da palavra-chave yield, podem pausar a execução de uma função, tornando-os úteis para alternar entre diferentes tarefas de maneira controlada. Você pode usar Generators para criar tarefas que pausam e retomam, permitindo que o script execute múltiplas tarefas sem bloqueio. Isso não é concorrência verdadeira, mas uma forma de alternar entre tarefas de forma controlada.

# pra que serve ?

A real dos generators é que só vamos utilizar ele caso tivermos um site que tenha milhões de acessos, e em alguma parte dele temos que gerar algo ou carregar algo que precise de um alto processamento de dados, isso sem deixar o usuário esperar e permitir que ele faça outras coisas enquanto o arquivo carrega.

Não é necessário usar Generators apenas em sites de **altíssimo tráfego**. Eles também são úteis em outras situações, como:

* **Relatórios Extensos**: Se sua aplicação precisa gerar relatórios com muitos dados, os Generators podem ajudar a evitar estouros de memória.
* **Processamento de Arquivos Grandes**: Quando você precisa processar arquivos grandes, como logs ou arquivos CSV, sem carregar todo o arquivo na memória.
* **Consultas a Bancos de Dados**: Quando você deseja buscar registros em lotes para evitar problemas de desempenho.

**Exemplo em Uma Aplicação Real**

Pense em uma **aplicação de e-commerce** com um painel de administração onde o administrador pode visualizar um relatório detalhado de pedidos. Se o administrador solicitar um relatório de um ano inteiro, a quantidade de dados pode ser enorme.

* **Sem Generators**: O servidor pode travar ou demorar muito para carregar todos os dados.
* **Com Generators**: O servidor pode carregar e exibir os dados em lotes, de forma mais eficiente, sem sobrecarregar a memória.

Alguns frameworks como Laravel e já vem com essas funcionalidades embutidas.

# Como montar um generator

Os Generators são projetados para produzir valores sob demanda. Eles funcionam como um iterador que pausa sua execução cada vez que encontram um yield e a retomam quando a próxima iteração é solicitada. Portanto, para extrair os valores, você precisa iterar sobre o objeto Generator.

* **Sem Iteração**: Se você criar um Generator e não iterar sobre ele, o Generator não será executado e nenhum valor será produzido.
* **Com Iteração**: Iterar sobre o Generator faz com que o PHP execute o código até o próximo yield e produza o próximo valor.

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamentePodemos também fazer N maneira de iterar sobre o generators, e todas elas sempre envolveram loops.

Se iniciarmos um generator sem fazer loop, significa que ele irá retornar apenas um valor, podemos ainda chamar métodos, mas se quisermos mais dados, precisamos de loops.

Texto

Descrição gerada automaticamente