Concorrência e Paralelismo

Sumário!

Sistemas Monotarefa x Multitarefas

Processos, Threads e Corrotinas

Programação Sequencial x Paralela x Concorrente

Concorrência e Paralelismo na Cloud

Monotarefa
x
Multitarefas

Sistemas Monotarefa

- 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.

Sistemas Monotarefa

- 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.

- I/O -> Milissegundo
- CPU -> Nanossegundos

- Milissegundos (10^-3 segundo)
- Nanossegundos (10^-9 segundo)

Sistemas Multitarefas

- A solução encontrada para resolver esse problema foi permitir ao processador suspender a execução de uma tarefa que estivesse aguardando dados externos ou algum evento e passar a executar outra tarefa.

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

- Nesse modelo, mais de um programa é carregado na memória.

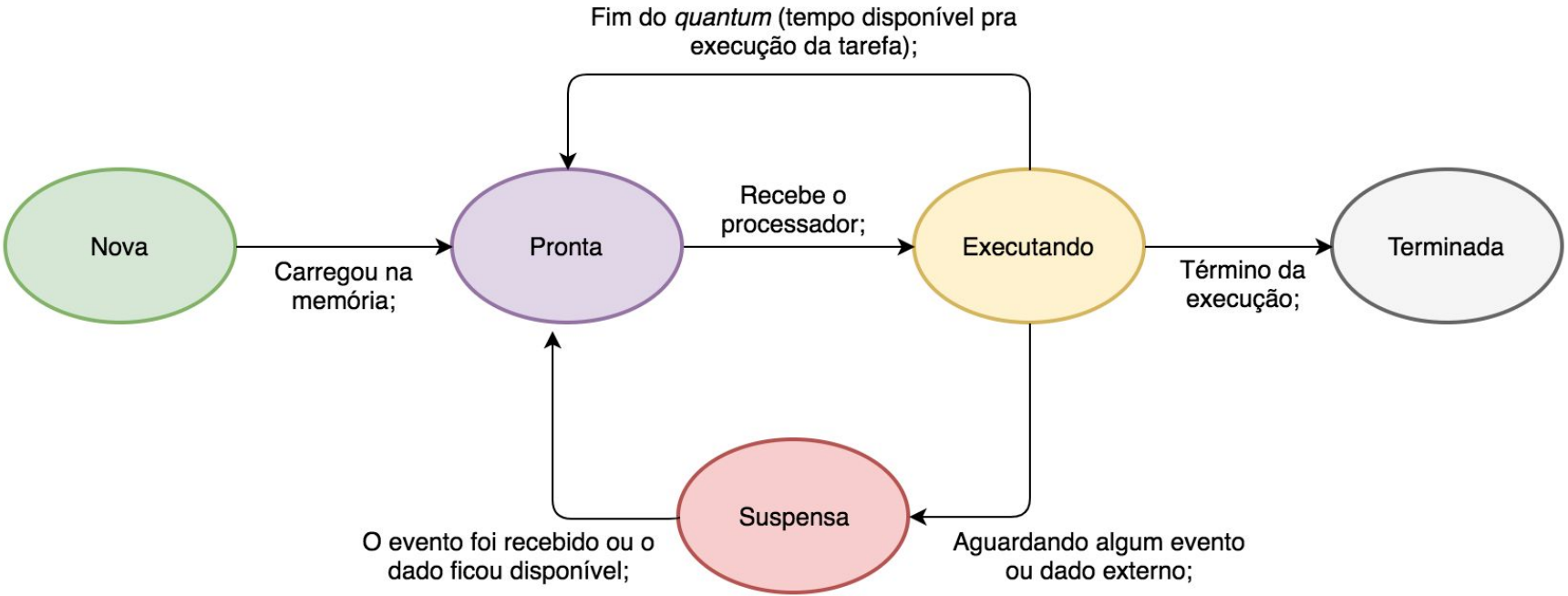
- O mecanismo que permite a retirada de um recurso (o processador, por exemplo) de uma tarefa, é chamado de preempção.

Sistemas Multitarefas

- 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.

- O SO possui um recurso chamado de “escalonador de processos” que é o responsável por “emprestar” a CPU para as tarefas.

Sistemas Multitarefas



Processos, Threads e Corrotinas

Processos

- Programa em execução.

- 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.

- O SO reúne informações do processo em uma estrutura de dados chamada PCB (Process Control Blocks), que por sua vez, são de extrema importância para o escalonador.

- O conjunto de PCBs forma o que chamamos de "tabelas de processos".

Processos

Um PCB é constituído por:

- PID (Identificador do processo);
- Registradores da CPU;
- O espaço de endereçamento do processo;
- A prioridade do processo;
- O estado do processo;
- Informações sobre o escalonamento de processo;
- Informações de entrada/saída;
- O ponteiro para o próximo PCB;

Threads

- Uma thread é uma linha de execução dentro de um processo. Ela é a menor unidade de processamento que pode ser executada de forma independente.

- 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).

- Threads são mais leves que processos porque compartilham muitos recursos do processo pai, tornando a criação e a troca de contexto entre threads mais rápida e menos custosa em termos de recursos do sistema.

Corrotinas

- Rotina nada mais é que uma instrução no código, como um “for”, um “while”, uma chamada de função, método, etc.

- Corrotinas são rotinas que coexistem entre si, de maneira independente, dentro de um processo.

- Funcionam de maneira idêntica ao que foi explicado em “sistemas multitarefas”, contudo, as corrotinas funcionam no nível do processo.

- Em Python e em Node.js, utiliza-se o “Event Loop”, já em Golang usa-se o “Scheduler”.

Sequencial
x
Paralela
x
Concorrente

Programação Sequencial

- Forma mais tradicional de escrever um programa.

- Estruturas do código são dependentes uns dos outros.

Programação Sequencial (Exemplo)

poc_sequential.py > ...

```python
import logging

import requests

logger = logging.getLogger(__name__)

urls = (
    "https://pt.aliexpress.com/",
    "https://m.shein.com/br/",
    "https://shopee.com.br",
    "https://www.enjoei.com.br/",
    "https://magazineluiza.com.br",
)

for url in urls:
    response = requests.get(url, timeout=5)
    logger.info(f"Status Code: {response.status_code} => {url}")

```

Link do gist

Programação Sequencial (Exemplo)

```
(env) → POC python poc_sequential.py
[SEQUENTIAL] 2024-05-27 21:33:13,831 - INFO : Status Code: 200 => https://pt.aliexpress.com/
[SEQUENTIAL] 2024-05-27 21:33:14,812 - INFO : Status Code: 200 => https://m.shein.com/br/
[SEQUENTIAL] 2024-05-27 21:33:15,334 - INFO : Status Code: 200 => https://shopee.com.br
[SEQUENTIAL] 2024-05-27 21:33:15,500 - INFO : Status Code: 200 => https://www.enjoei.com.br/
[SEQUENTIAL] 2024-05-27 21:33:16,155 - INFO : Status Code: 200 => https://magazineluiza.com.br
[SEQUENTIAL] 2024-05-27 21:33:16,155 - INFO : 0:00:03.002740
(env) → POC
```

Programação Paralela

- Paralelismo: Independência.

- Execução simultânea de um conjunto de tarefas, que podem ou não estar relacionadas. É uma forma de fazer muitas coisas ao mesmo tempo e está ligado a forma de se executar tarefas.

- Programação utilizando multi-processing ou multi-threads.

- Cada processo / thread criada será executada em uma CPU a parte de maneira simultânea.

Programação Paralela (Exemplo - Multiprocessing)

poc_multiprocessing.py > ...

```python
import logging
import os
from multiprocessing import Pool

import requests

logger = logging.getLogger(__name__)

urls = (
    "https://pt.aliexpress.com/",
    "https://m.shein.com/br/",
    "https://shopee.com.br",
    "https://www.enjoei.com.br/",
    "https://magazineluiza.com.br",
)


def get_and_print(url):
    response = requests.get(url)
    logger.info(f"Status Code: {response.status_code} => {url}")


pool = Pool(processes=os.cpu_count())
pool.map(get_and_print, urls)

```

Programação Paralela (Exemplo - Multiprocessing)

```
(env) → POC python poc_multiprocessing.py
[MULTIPROCESSING] 2024-05-27 21:34:16,608 - INFO : Status Code: 200 => https://www.enjoei.com.br/
[MULTIPROCESSING] 2024-05-27 21:34:16,981 - INFO : Status Code: 200 => https://magazineluiza.com.br
[MULTIPROCESSING] 2024-05-27 21:34:16,988 - INFO : Status Code: 200 => https://m.shein.com/br/
[MULTIPROCESSING] 2024-05-27 21:34:17,131 - INFO : Status Code: 200 => https://shopee.com.br
[MULTIPROCESSING] 2024-05-27 21:34:17,165 - INFO : Status Code: 200 => https://pt.aliexpress.com/
[MULTIPROCESSING] 2024-05-27 21:34:17,168 - INFO : 0:00:00.769469
(env) → POC
```

Programação Paralela (Exemplo - Multithreading)

poc_multithreads.py > ...

```python
import logging
import os
from multiprocessing.pool import ThreadPool as Pool

import requests

logger = logging.getLogger(__name__)

urls = (
    "https://pt.aliexpress.com/",
    "https://m.shein.com/br/",
    "https://shopee.com.br",
    "https://www.enjoei.com.br/",
    "https://magazineluiza.com.br",
)


def get_and_print(url):
    response = requests.get(url)
    logger.info(f"Status Code: {response.status_code} => {url}")


pool = Pool(processes=os.cpu_count())
pool.map(get_and_print, urls)

```

Programação Paralela (Exemplo - Multithreading)

```
(env) → POC python poc_multithreads.py
[MULTITHREADS] 2024-05-27 21:35:23,394 - INFO : Status Code: 200 => https://www.enjoei.com.br/
[MULTITHREADS] 2024-05-27 21:35:23,661 - INFO : Status Code: 200 => https://magazineluiza.com.br
[MULTITHREADS] 2024-05-27 21:35:23,802 - INFO : Status Code: 200 => https://m.shein.com/br/
[MULTITHREADS] 2024-05-27 21:35:23,852 - INFO : Status Code: 200 => https://shopee.com.br
[MULTITHREADS] 2024-05-27 21:35:23,945 - INFO : Status Code: 200 => https://pt.aliexpress.com/
[MULTITHREADS] 2024-05-27 21:35:23,946 - INFO : 0:00:00.805671
(env) → POC
```

Programação Concorrente

- Concorrência: Interrupção.

- Conjunto de tarefas sendo executadas de forma intercalada em um mesmo intervalo de tempo. É uma forma de lidar com muitas coisas ao mesmo tempo e está ligado a forma de se estruturar tarefas.

- Para que seja possível implementar um modelo de programação concorrente, é preciso que seja possível quebrar uma tarefa em pedaços que podem ser executados independentemente.

- Para isso, é necessário utilizar as corrotinas, que são funções que podem ser suspensas em determinados pontos de execução para serem retomadas depois, mantendo todos os estados de quando foi suspensa.

- Utiliza Event Loops para “escalonar”.

Programação Concorrente (Exemplo)

poc_async.py > ...

```python
import asyncio
import logging

from aiohttp import ClientSession

logger = logging.getLogger(__name__)

urls = (
    "https://pt.aliexpress.com/",
    "https://m.shein.com/br/",
    "https://shopee.com.br",
    "https://www.enjoei.com.br/",
    "https://magazineluiza.com.br",
)


async def get_and_print(session, url):
    async with session.get(url) as response:
        logger.info(f"Status Code: {response.status} => {url}")


async def fetch(loop, urls):
    async with ClientSession(loop=loop) as session:
        tasks = (get_and_print(session, url) for url in urls)
        await asyncio.gather(*tasks, return_exceptions=True)
        loop.stop()


loop = asyncio.get_event_loop()
loop.create_task(fetch(loop, urls))
loop.run_forever()

```

Programação Concorrente (Exemplo)

```
(env) → POC python poc_async.py
[ASYNC] 2024-05-27 21:36:03,825 - INFO : Status Code: 200 => https://www.enjoei.com.br/
[ASYNC] 2024-05-27 21:36:04,146 - INFO : Status Code: 200 => https://magazineluiza.com.br
[ASYNC] 2024-05-27 21:36:04,155 - INFO : Status Code: 200 => https://m.shein.com/br/
[ASYNC] 2024-05-27 21:36:04,271 - INFO : Status Code: 200 => https://pt.aliexpress.com/
[ASYNC] 2024-05-27 21:36:04,484 - INFO : Status Code: 200 => https://shopee.com.br
[ASYNC] 2024-05-27 21:36:04,484 - INFO : 0:00:00.735075
(env) → POC
```

Concorrência e Paralelismo na Cloud