Programação concorrente: introdução

- O mundo real funciona concorrentemente: várias atividades podem ser executadas em paralelo. Ex.: uma pessoa pode estar
  - respirando, e,
  - falando, e
  - escrevendo, e
  - lendo, etc.
- Computadores também operam concorrentemente. Ex.: um computador pode estar
  - compilando um programa, e
  - recebendo uma mensagem, e,
  - imprimindo um arquivo, e,
  - tocando música, etc.

Programação concorrente: motivações para estudo

1) Muitos domínios de problemas prestam-se naturalmente à concorrência, de uma maneira muito similar ao fato da recursão ser natural ao idealizar-se a solução de alguns problemas. Exs.: vôos de aeronaves em área controlada, estações repetidoras, várias máquinas de uma instalação manufatureira.
2) Computadores de múltiplos processadores são amplamente utilizados atualmente, criando a necessidade de que o software faça uso efetivo de sua capacidade.

Objetivos da programação concorrente

- Reduzir o tempo total de processamento
  - múltiplos processadores
- Aumentar confiabilidade e disponibilidade
  - processadores distribuídos
- Obter especialização de serviços
  - sistemas operacionais
  - simuladores
- Implementar aplicações distribuídas
  - correio eletrônico

Vantagens da programação concorrente

- Aumento de desempenho, pois aumenta-se a quantidade de tarefas sendo executadas em determinado período de tempo.
- Possibilidade de uma melhor modelagem de programas, pois determinados problemas computacionais são concorrentes por natureza.

Programação Concorrente

- Paradigma de programação que faz uso da **execução concorrente** (simultânea) de várias tarefas computacionais interativas, que podem ser implementadas como programas separados ou como um conjunto de *threads* criadas por um único programa.

Programação Concorrente

- Essas tarefas podem ser executadas por um único processador, vários processadores em um único equipamento (*multicore*) ou processadores distribuídos por uma rede (sistemas distribuídos).

Programação concorrente

- O termo programação concorrente é usado no sentido abrangente, para designar a programação paralela e a programação distribuída, porém foca mais na **interação entre as tarefas.**
- A interação e a comunicação correta entre as diferentes tarefas, além da coordenação do acesso concorrente aos recursos computacionais são as principais questões discutidas durante o desenvolvimento de sistemas concorrentes.

Programação concorrente

- Uma **unidade concorrente** é um componente de um programa que não exige a execução sequencial, ou seja, que sua execução seja realizada antes ou após a execução de outros componentes do programa.
- Concorrência relaciona-se com fluxo de controle: em um programa, existe mais de um fluxo de controle ativo.

Fluxo de execução

Execução sequencial

- **Comandos de controle de fluxo de execução**
  - Sequencial
  - Condicional
  - Iterativo
- **Requisição de execução de unidades**
  - explícita: chamada de métodos
  - implícita: ativação de exceções
- **Programa controla a ordem de execução**

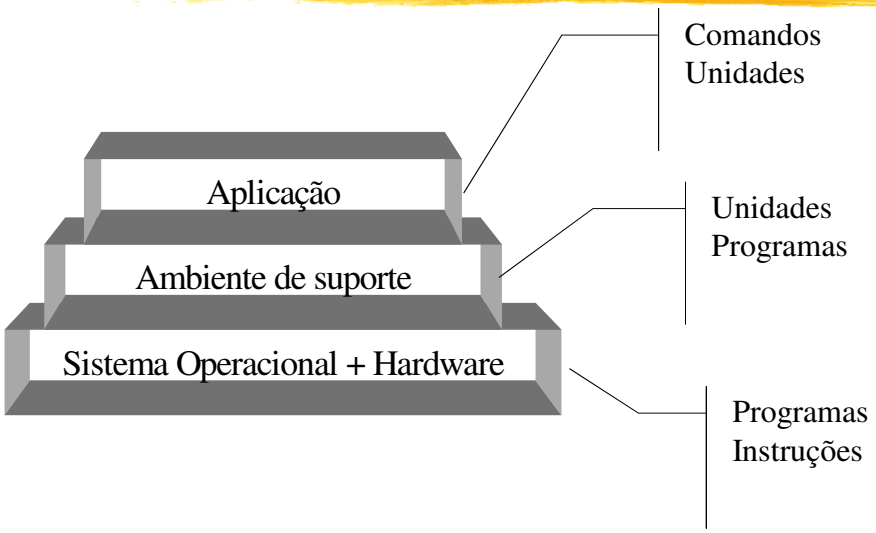
Execução concorrente

- **Cada tarefa é uma unidade de execução autônoma (*thread*)**
- **Tarefas podem ser totalmente independentes**
  - Exemplo: execução de um mesmo método sobre dois objetos (da mesma classe)
- **Tarefas podem necessitar comunicação**
- **Programa não controla a ordem de execução**

Níveis de concorrência

- Instrução de máquina
  - Executando 2 ou mais instruções de máquina simultaneamente
- **Instrução (comando)**
  - Executando 2 ou mais instruções simultaneamente
- **Unidade (processos)**
  - Executando 2 ou mais unidades de subprograma simultaneamente
- Programa
  - Executando 2 ou mais programas simultaneamente

Níveis de controle de concorrência



Categorias de concorrência

- Concorrência física: diversas unidades do mesmo programa literalmente executam simultaneamente, supondo que mais de um processador esteja disponível.
- Concorrência lógica: programador e programa supõem a existência de múltiplos processadores fornecendo concorrência real quando, de fato, a execução real dos programas está se dando intercaladamente em um único processador.

Conceitos fundamentais

- **Tarefa**: unidade de programa que pode estar em execução concorrente com outras unidades do mesmo programa.
- Implicitamente iniciada, quando unidade de programa invoca uma tarefa, ela não precisa esperar que esta conclua sua execução antes de prosseguir por si mesma; e quando a execução da tarefa é concluída, o controle pode ou não retornar à unidade.
- Tarefas podem se comunicar entre si via variáveis não-locais compartilhadas, pela passagem de parâmetros e pelos parâmetros.

Conceitos fundamentais

- **Sincronização**: mecanismo que controla a ordem de execução das tarefas.
- **Tipo cooperação**: qdo tarefa A precisa aguardar que B conclua alguma atividade específica antes de prosseguir sua execução. Impõem aos usuários de uma mesma estrutura de dados compartilhada o uso cooperativo do recurso.

Conceitos fundamentais

- **Sincronização**: mecanismo que controla a ordem de execução das tarefas.
- **Tipo competição**: qdo ambas tarefas requerem o uso de algum recurso que não pode ser usado simultaneamente. Impede que 2 tarefas acessem a mesma estrutura de dados compartilhada exatamente ao mesmo tempo – o que poderia destruir sua integridade. Para isso deve-se garantir acesso mutuamente exclusivo aos dados compartilhados.

Exemplo de Sincronização Tipo Cooperação

- Problema do produtor-consumidor, originário do desenvolvimento de SOs, nos quais uma unidade de programa produz algum valor de dados ou recurso e outra unidade o usa. Os dados são colocados em um retentor (*buffer*) de armazenamento pela unidade produtora e removido dela pela consumidora, de forma sincronizada. Não se permite remoção no buffer vazio, nem colocação no buffer cheio.

Métodos para acesso mutuamente exclusivo a um recurso

- **Semáforo** – estrutura de dados que consiste em um número inteiro e uma fila que armazena descritores de tarefas (armazena todas as informações relevantes sobre o estado de execução de uma tarefa).

Métodos para acesso mutuamente exclusivo a um recurso

- **Monitor** – técnica para sincronizar 2 ou mais tarefas q compartilham um recurso em comum (dispositivo de hardware ou região da memória). Assim, o programador não precisa ter acesso às primitivas para tal, tendo que realizar o bloqueio e desbloqueio de recursos manualmente.

Métodos para acesso mutuamente exclusivo a um recurso

- **Passagem de mensagens –** técnica que resolve o problema de o que fazer quando múltiplos pedidos simultâneos são feitos por outras tarefas para se comunicar com uma em especial. Usa forma de não-determinismo para assegurar justiça na escolha de qual pedido será atendido primeiro.

Execução concorrente

- Execução concorrente, também conhecida como execução paralela, não significa execução simultânea.
- O programa geralmente não possui controle sobre a ordem e o tempo de execução das unidades concorrentes.

Execução concorrente

- A execução de unidades concorrentes admite as seguintes possibilidades:
  - **Pseudo-paralela**: Execução em um único processador;
  - **Paralela**: Execução em vários processadores que compartilham uma única memória;
  - **Distribuída**: Execução em vários processadores independentes, sem compartilhamento de memória.

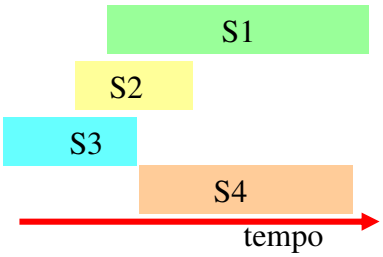
Unidades concorrentes: código e dados

- A informação exigida para a execução de uma unidade pode ser decomposta em duas partes:
  - uma parte permanente, que consiste da estrutura dos dados e da sequência de instruções a serem executadas (código reentrante).
  - uma parte temporária, que consiste de dados e outras informações contextuais que variam a cada execução do componente .
- Além disso, deve ser considerada a possibilidade de acesso a dados não locais, isto é, não definidos na unidade em questão.

Exemplo de concorrência

Os comandos S1 a S4, representam p.ex. chamadas de procedimentos, podem ser executados em paralelo. O ponto "coend" indica que todas as unidades devem terminar a sua execução antes de passar para o ponto seguinte.

```
--------- comando anterior
        Cobegin
                S1;
                S2;
                S3;
                S4;
        Coend;
---------- comando seguinte
```



Exemplo em Go

```
package main
import "fmt“
func f(from string) {
    for i := 0; i < 3; i++ {
        fmt.Println(from, ":", i)
    }
}
func main() {
    f("direct")
    go f("goroutine")
    go func(msg string) {
        fmt.Println(msg)
    }("going")
    fmt.Scanln()
    fmt.Println("done")
}
```

```
$ go run goroutines.go
direct : 0
direct : 1
direct : 2
goroutine : 0
going
goroutine : 1
goroutine : 2
<enter> done
```

Programação concorrente: algumas linguagens

1) Java e C# – modelo de memória compartilhada com bloqueio via monitores
2) Erlang – Ericsson, modelo de troca de mensagens
3) Occam – nativa para microprocessadores INMOS, criada para computação paralela
4) GoLang – gorotina não compartilha dados/memória
5) Rust – concorrência sem disputa de dados, referências compartilhadas entre várias threads são somente para leitura.

Programação Concorrente em Outros Paradigmas

- Paradigma Imperativo (procedimental e orientado a objetos, p.ex.)
  - Concorrência explícita.
  - Programação mais complexa, deixando a cargo do programador todo o controle.
- Paradigma Funcional e Lógico
  - Explora concorrência automaticamente, sem controle do programador.