

# Sistemas Hardware-Software

Aula 19 - Introdução a sincronização

**Engenharia**

Fabio Lubacheski

Maciel C. Vidal

Igor Montagner

Fábio Ayres

# Comunicação entre as Threads

- As threads podem trabalhar **cooperativamente** na resolução de um problema;
- As variáveis globais são utilizadas para trocar informações e influenciar na execução das outras threads.

# Tarefas paralelas

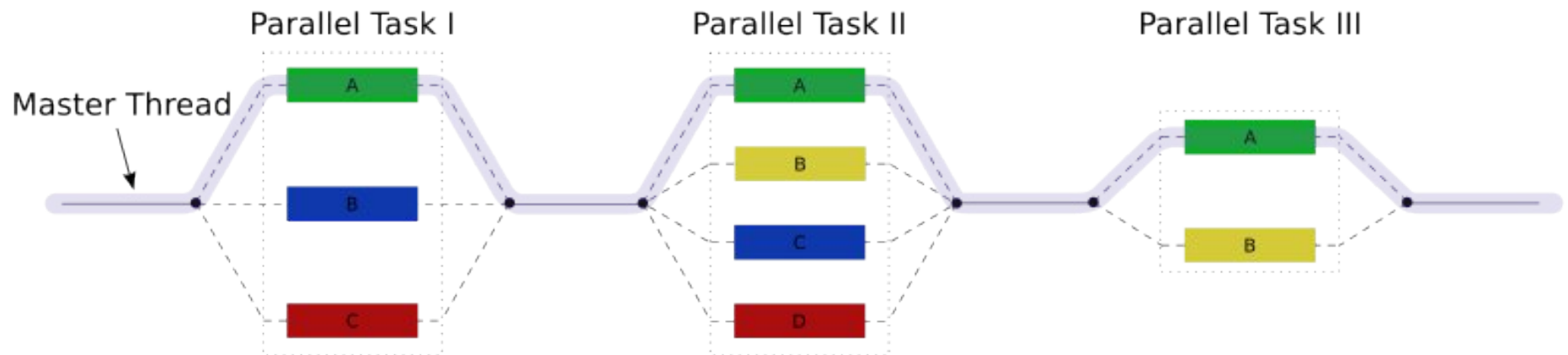
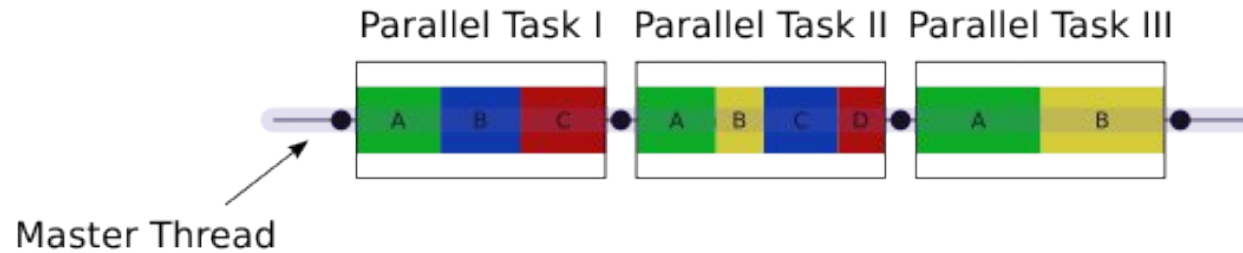


Figura: [https://en.wikipedia.org/wiki/File:Fork\\_join.svg](https://en.wikipedia.org/wiki/File:Fork_join.svg)

# POSIX threads

O padrão POSIX define também uma API de threads (*pthread*) que inclui

- Criação de threads
- Controle a acesso de dados (usando **mutex Semáforos Binários**)
- Sincronização (usando **Semáforos Contadores**)



# Atividade prática

## Aquecimento (20 min)

1. Utilização da API pthreads
2. Dividir uma tarefa em pedaços para executar.

# Correção

## Aquecimento

1. Utilização da API pthreads
2. Dividir uma tarefa em pedaços para executar.

# Conceito : Race Condition

"Ocorre quando a saída do programa depende da ordem de execução das threads"

Em geral ocorre quando

- uma variável é usada em mais de uma thread e há pelo menos uma operação de escrita (**variável compartilhada**).
- trabalhamos com os mesmos **arquivos** simultaneamente em várias threads

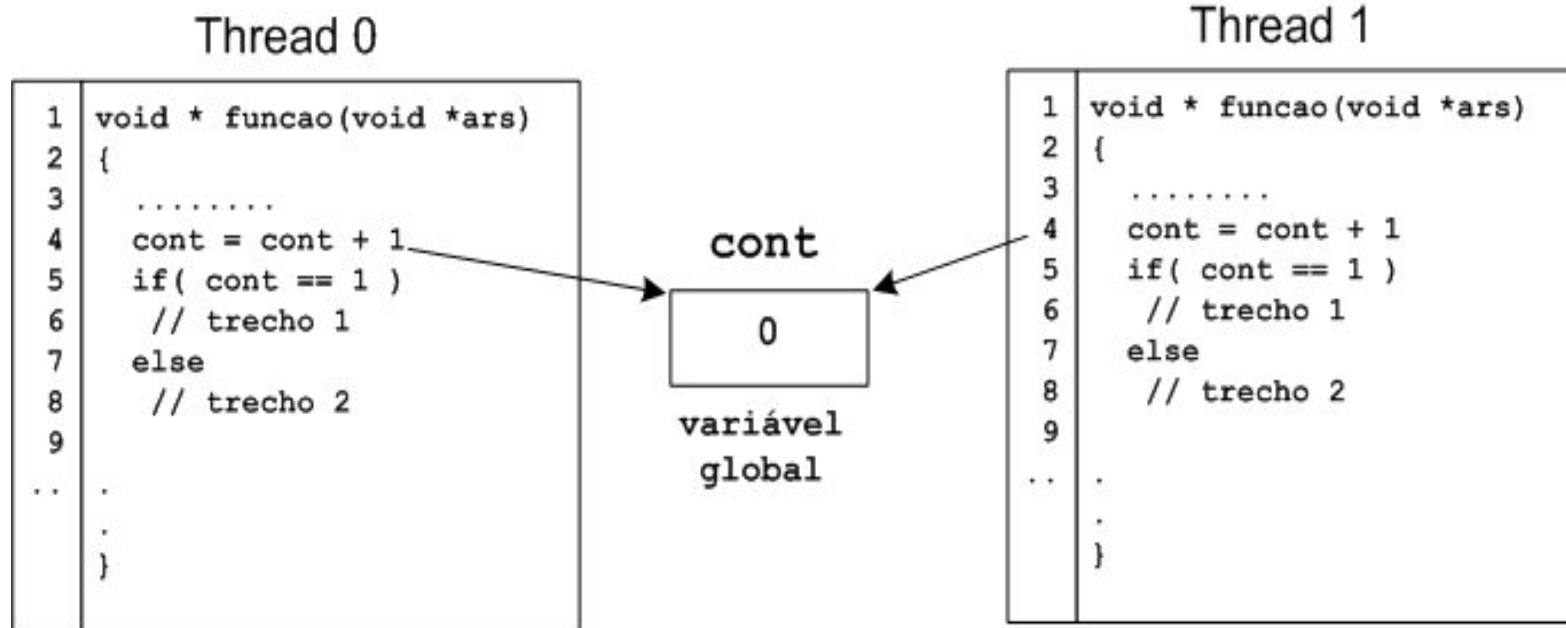
# Conceito : Região Crítica

"Parte do programa que só pode ser rodada uma thread por vez"

- elimina situações de concorrência
- elimina também todo o paralelismo e pode se tornar gargalo de desempenho

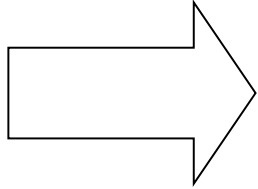


# Exemplo de Região Crítica



- A variável **cont** é **compartilhada** e incrementada nos dois processos para que somente um processo execute o **trecho 1**;

## Exemplo de Região Crítica

`cont=cont+1` 

```
mov 0x0(%rip),%eax  
lea 0x1(%rax),%esi  
mov %esi,0x0(%rip)
```

O que acontece se as threads **T0** e **T1** forem escalonados na linha acima e as duas threads executarem ao mesmo tempo a instrução **cont=cont+1** ?

## Exemplo de Região Crítica

```
mov 0x0(%rip),%eax //thread 1
lea 0x1(%rax),%esi //thread 1
mov 0x0(%rip),%eax //thread 2
lea 0x1(%rax),%esi //thread 2
mov %esi,0x0(%rip) //thread 1
mov %esi,0x0(%rip) //thread 2
```

# Implementando uma Região Crítica

- A **região crítica** é garantida através de um **protocolo** de sincronização denominado **Exclusão Mútua**.
- O protocolo de **Exclusão Mútua** deve garantir que se um processo estiver usando um recurso compartilhado os demais serão impedidos de fazer a mesma coisa.
- Implementação do **Protocolo de Exclusão Mútua**
  - Adquire o controle Exclusivo
  - **Região Crítica**
  - Libera o controle Exclusivo

# Semáforo Mutex (Mutual Exclusion)

- Para implementar o **Protocolo de Exclusão Mútua** utilizaremos **semáforos**, que são mecanismos de sincronização que permitem gerir o acesso a recursos em **modo exclusivo** e em **modo de cooperação**;
- **Semáforos Mutex** são denominados **semáforos binários** pois assumem somente dois valores **0** ou **1**.
- Um **semáforo Mutex** é representado por uma variável inteira **não negativa** que só pode ser manipulada pelas primitivas **Lock** e **Unlock**.

# Mutex – Representação Conceitual

**Lock**(Mutex s)

```
{  
    if( s = 1 )  
        s = 0  
    else  
        "Bloqueia a thread"  
}
```

**Unlock**(Mutex s)

```
{  
    if("existe uma thread  
        bloqueada" )  
        "desbloqueia a  
        thread"  
    else  
        s = 1  
}
```

# Atividade prática

## Sincronização usando mutex (20 minutos)

1. Utilização da API pthreads para criar mutex
2. Entender quando usá-los e como diminuir seu custo

# Correção

## Sincronização usando mutex

1. Utilização da API pthreads para criar mutex
2. Entender quando usá-los e como diminuir seu custo



# Semáforo Mutex (Mutual Exclusion)

- Caro, mas muito útil quando somos obrigados a compartilhar um recurso
- Ideal é usar **Lock/Unlock** o mínimo possível
- Criar cópias privadas de uma variável compartilhada pode ajudar

# Insper

[www.insper.edu.br](http://www.insper.edu.br)