

# Sistemas Hardware-Software

Aula 18 - Introdução a concorrência

**Engenharia**  
Fabio Lubacheski  
Maciel C. Vidal  
Igor Montagner  
Fábio Ayres



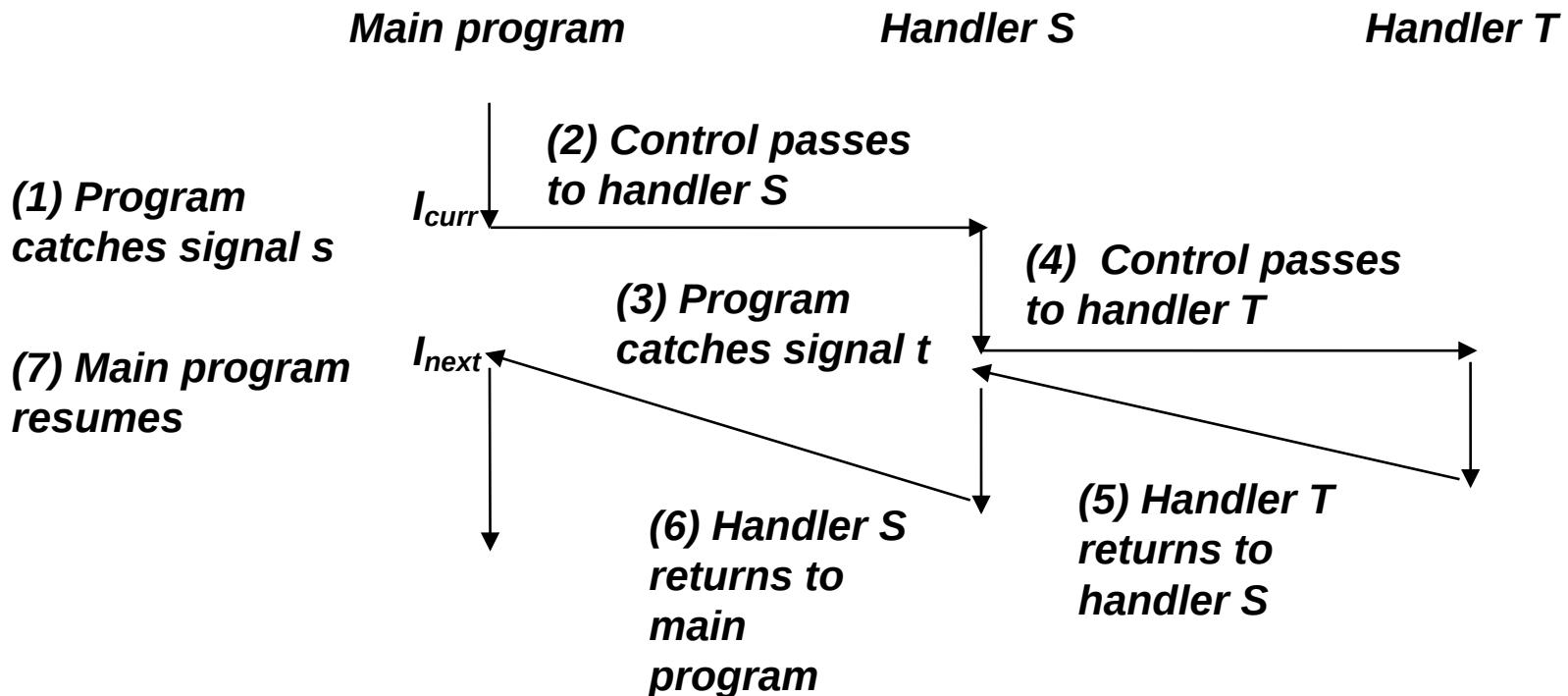
# Correção

## Bloqueando sinais

1. Sinais diferentes sendo capturados pelo mesmo processo
2. Bloqueando sinais durante a execução do handler

# Sinais concorrentes

Handlers podem ser interrompidos por outros handlers!



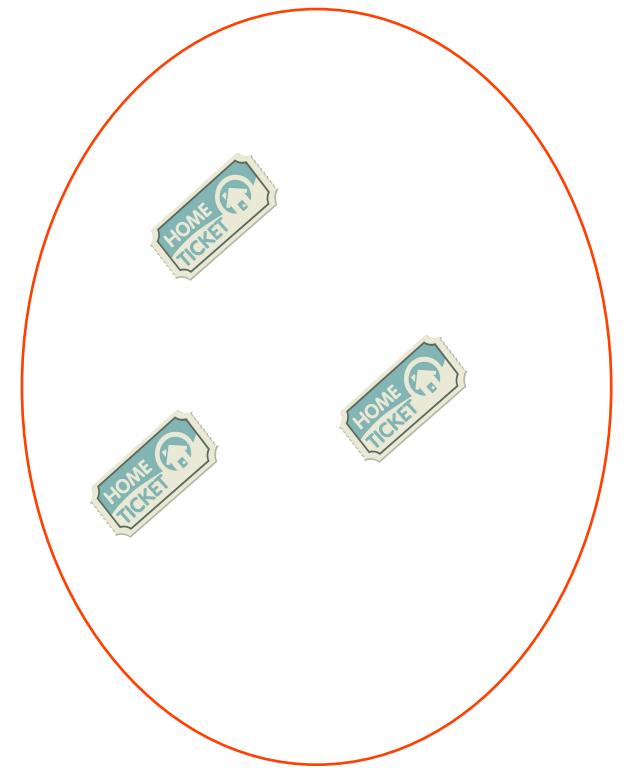
Mas não pode haver mais de um handler do mesmo sinal rodando!

# Até agora vimos no multiprocessamento

## Processos:

- Colaboração para resolver um problema é limitada
- Compartilhamento de dados é um complicador
  - Concorrência por recursos
- Sincronização entre processos limitada

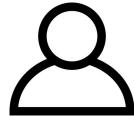
# Situação 1 – compra de ingressos



# Situação 1 – compra de ingressos



- Existe uma quantidade limitada de ingressos
- Não posso vender o mesmo ingresso para duas pessoas diferentes
- Muitos pedidos de ingressos
- Podem chegar a qualquer momento



# Situação 1 – compra de ingressos



- Existe uma quantidade limitada de ingressos
- Não posso vender o mesmo ingresso para duas pessoas diferentes
- Muitos pedidos de ingressos
- Podem chegar a qualquer momento



**Concorrência por um recurso compartilhado que só pode ser usado por uma tarefa por vez.**

# Situação 2 – busca em fotos

Objetivo: contar pessoas nas fotos



1. Preciso esperar a primeira foto para começar analisar a segunda?
2. Consigo responder antes de acabar todas?

# Situação 2 – busca em fotos

Partes do programa são independentes:

- Análise de uma imagem não depende das outras

Partes são síncronas

- Só consigo finalizar se todas estiverem prontas

# Situação 2 – busca em fotos

Partes do programa são independentes:

- Análise de uma imagem não depende das outras

**Tarefas precisam de sincronização**

Partes são síncronas

- Só consigo finalizar se todas estiverem prontas

# Multiprocessamento

Divisão de um programa em várias tarefas (**multiprocessamento**) podem acontecer em **paralelo** ou **concorrente**. E possuem dois pontos chaves:

- **Compartilhamento de recursos**
  - Tarefas usam os mesmos dados
- **Sincronização de tarefas**
  - Algumas tarefas dependem das outras

# Multiprocessamento ...

- ... é emocionante!
  - uma das áreas mais interessantes da computação!
- ... é frustrante!
  - É difícil.
  - Muito difícil.
- ... é inevitável!
  - computação paralela em todo lugar, do laptop ao datacenter
  - é um conhecimento fundamental (e um diferencial de mercado!) para engenheiros de computação

# Implementando o multiprocessamento

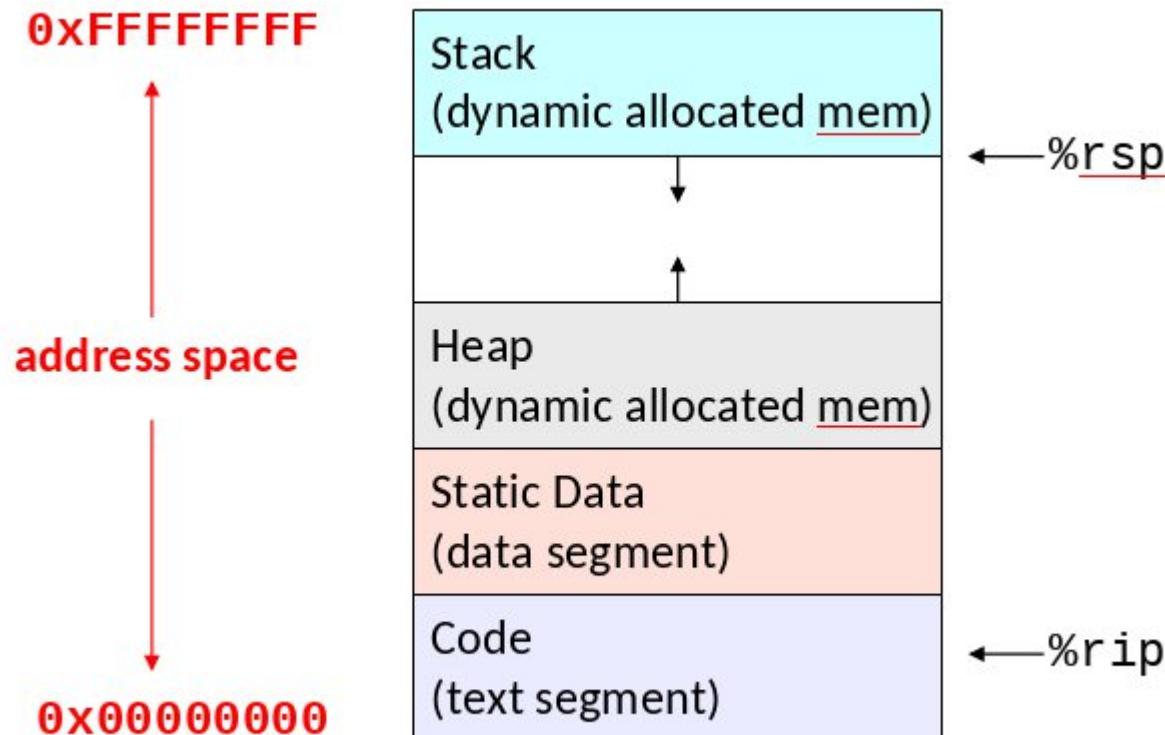
- **Processos**

- Espaços de endereçamento separados e comunicação custosa entre processos
- Compartilham algumas estruturas (tabela de descritores de arquivo, etc)

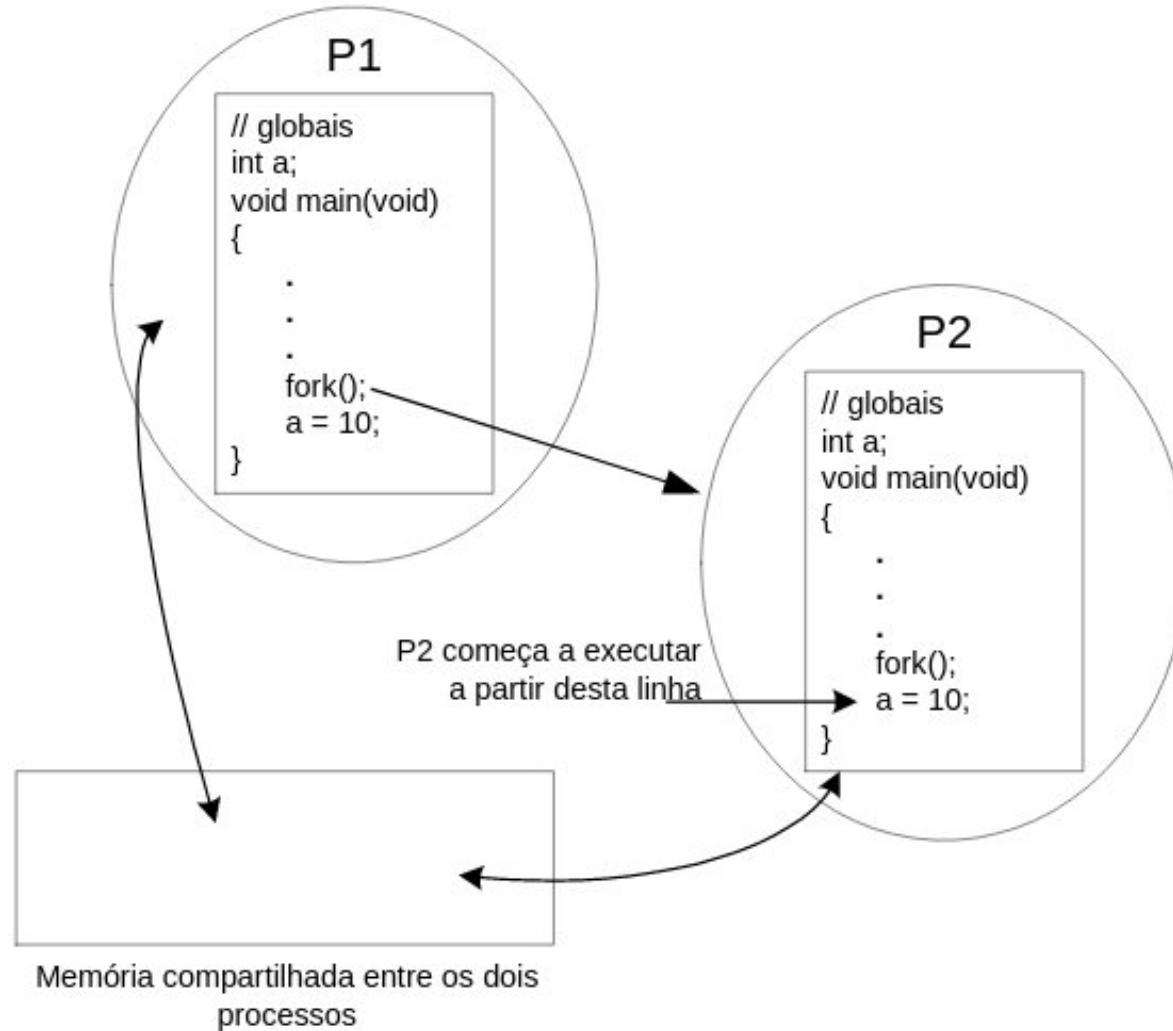
- **Threads**

- Executam no mesmo processo
- Mesmo espaço de endereçamento
  - Compartilham memória usando variáveis

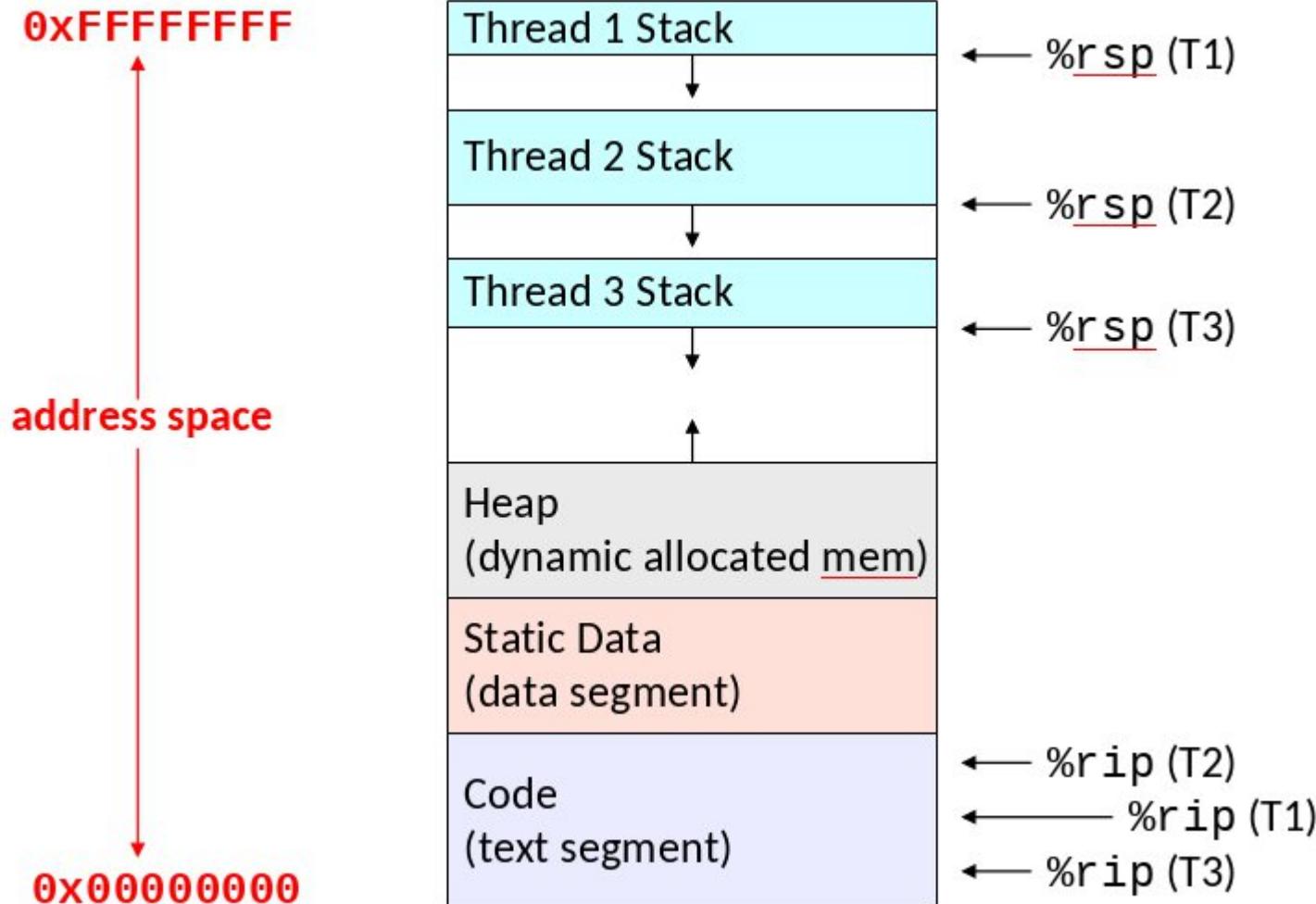
# Processos



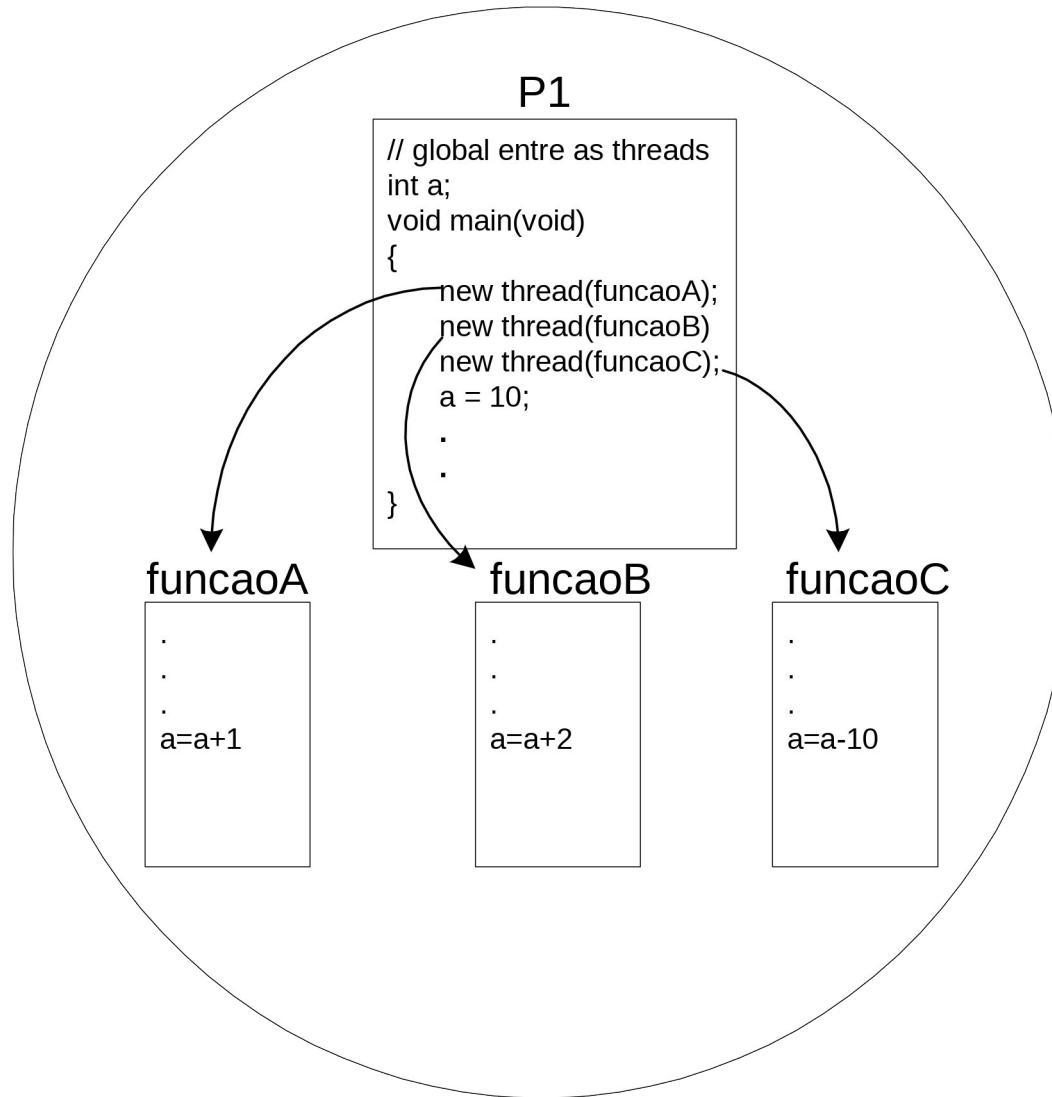
# Processos



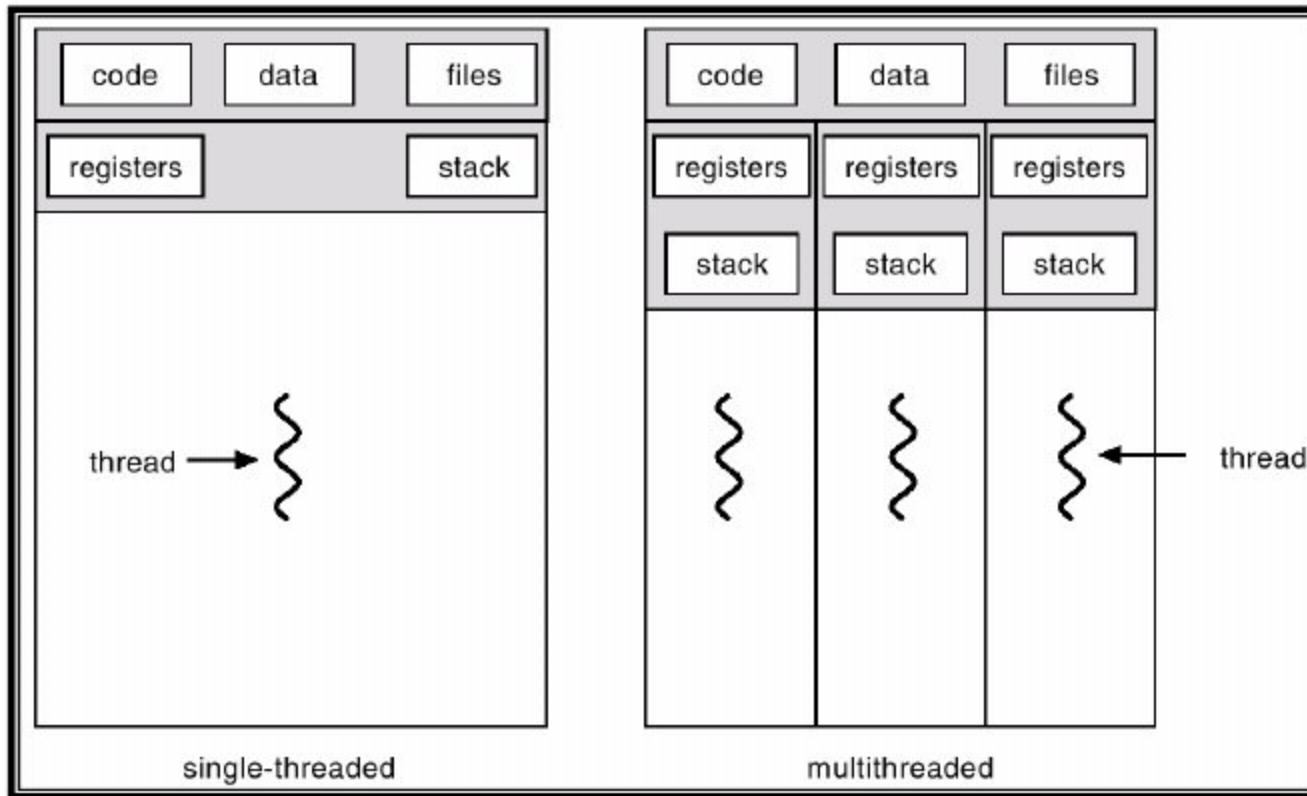
# Threads



# Threads



# Processos e threads

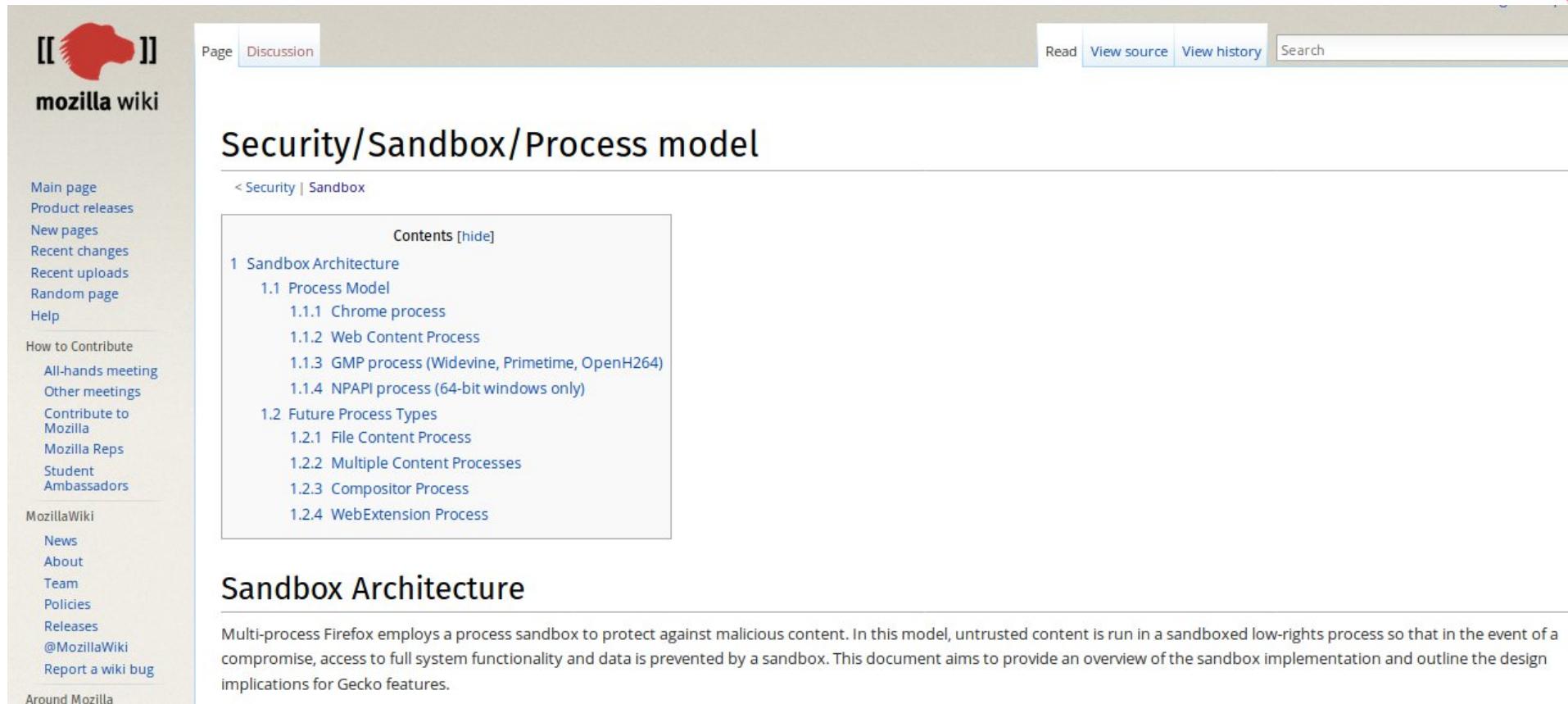


# Processos e threads

- Processos
  - Comunicação entre processos
  - **Possível distribuir em várias máquinas**
- Threads
  - Mais barato de criar e destruir
  - Sempre pertencem a um único processo
  - **Sincronização para acessar recursos compartilhados**

Troca de contexto ocorre de maneira igual nos dois casos!

# Exemplos de implementações - Processos



The screenshot shows a Mozilla Wiki page titled "Security/Sandbox/Process model". The page content is organized into sections: "Sandbox Architecture" (with sub-sections for Process Model, Future Process Types, and various content processes), "File Content Process", "Multiple Content Processes", "Compositor Process", and "WebExtension Process". The page also includes a sidebar with links to Main page, Product releases, New pages, Recent changes, Recent uploads, Random page, Help, How to Contribute, MozillaWiki, and Around Mozilla.

Page Discussion Read View source View history Search

## Security/Sandbox/Process model

< Security | Sandbox

Contents [hide]

- 1 Sandbox Architecture
  - 1.1 Process Model
    - 1.1.1 Chrome process
    - 1.1.2 Web Content Process
    - 1.1.3 GMP process (Widevine, Primetime, OpenH264)
    - 1.1.4 NPAPI process (64-bit windows only)
  - 1.2 Future Process Types
    - 1.2.1 File Content Process
    - 1.2.2 Multiple Content Processes
    - 1.2.3 Compositor Process
    - 1.2.4 WebExtension Process

# Exemplos de implementações -Processos



## The Chromium Projects

Home  
Chromium  
Chromium OS

Quick links  
Report bugs  
Discuss  
Sitemap

Other sites  
Chromium Blog  
Google Chrome  
Extensions

Except as otherwise [noted](#), the content of this page is licensed under a [Creative Commons Attribution 2.5 license](#), and examples are licensed under the [BSD License](#).

For Developers > Design Documents >

## Process Models

**Contents**

- [1 Overview](#)
- [2 Supported models](#)
  - [2.1 Process-per-site-instance](#)
  - [2.2 Process-per-site](#)
  - [2.3 Process-per-tab](#)
  - [2.4 Single process](#)
- [3 Sandboxes and plug-ins](#)
- [4 Caveats](#)
- [5 Implementation notes](#)
- [6 Academic Papers](#)

This document describes the different process models that Chromium supports for its renderer processes, as well as caveats in the models as it exists currently.

### Overview

Web content has evolved to contain significant amounts of active code that run within the browser, making many web sites more like applications than documents. This evolution has changed the role of the browser into an operating system rather than a simple document renderer. Chromium is built like an operating system to run these applications in a safe and robust way, using multiple OS processes to isolate web sites from each other and from the browser itself. This improves robustness because each process runs in its own address space, is scheduled by the operating system, and can fail independently. Users can also view the resource usage of each process in Chromium's Task Manager.

# Exemplos de implementações - Threads

Processamento de dados em aplicações gráficas



Computação paralela em que é importante/necessário compartilhar dados

# POSIX threads

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

- Criação de threads
- Sincronização (usando semáforos)
- Controle a acesso de dados (usando mutex)

# Atividade prática

## Criação de threads (15 minutos)

1. Utilização da API pthreads
2. `pthread_create/pthread_join`



# Atividade prática

**Criação de threads com argumentos (20 minutos)**

1. Utilização da API pthreads
2. `pthread_create/pthread_join`
3. recebendo argumentos



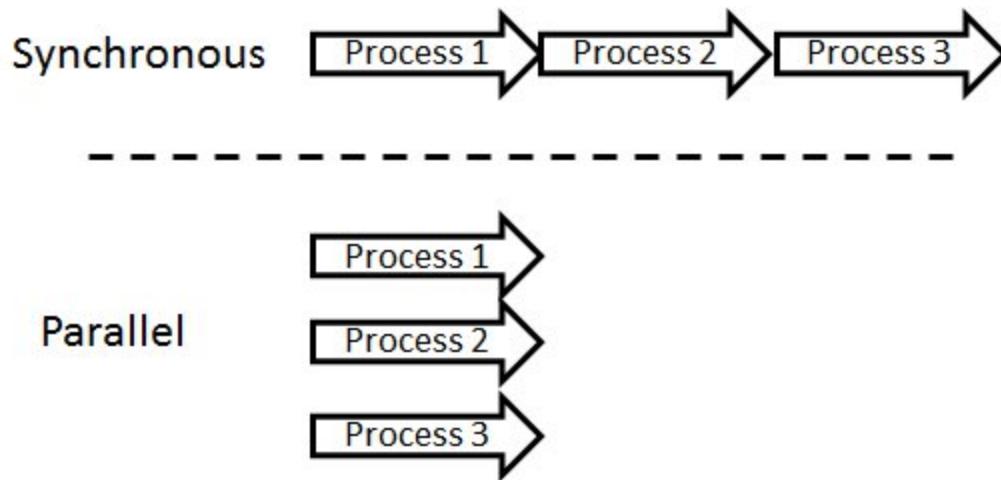
# Atividade prática

## Criação de threads com argumentos e valor de retorno (10 min)

1. Utilização da API pthreads
2. pthread\_create/pthread\_join
3. recebendo argumentos
4. e produzindo resultados

# Próxima aula

- reorganizando código sequencial em paralelo



Fonte: <https://www.packtpub.com/books/content/asynchrony-action>

# Insper

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