

Sistemas Hardware-Software

Aula 21 – Programação concorrente

2020 – Engenharia

Igor Montagner, Fábio Ayres [<igorsm1@insper.edu.br>](mailto:igorsm1@insper.edu.br)

Até agora

- Chamadas de Sistema POSIX
 - Arquivos, permissões e usuários
 - Gerenciamento de processos
 - Redirecionamento de arquivos (Entrada/Saída)
- Processo
 - Bloco básico de execução
 - Isolamento total de memória
 - Comunicação via arquivos (sockets/pipes/etc)

Processos

- Colaboração para resolver um problema é limitada
- Compartilhamento de dados pode ser importante
 - Concorrência por recursos
- Sincronização entre tarefas

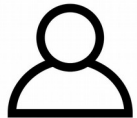
Situação 1 – compra de ingressos



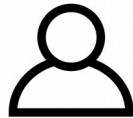
Situação 1 – compra de ingressos



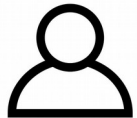
- Existe uma quantidade limitada de ingressos
- Não posso vender o mesmo ingresso para duas pessoas diferentes
- Muitos pedidos de ingressos
- Podem chegar a qualquer momento



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

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

analisar a segunda?

2. Consigo responder antes de acabar todas?

Tarefas precisam de sincronização

Programação concorrente

Divisão de um programa em várias tarefas que envolvem

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

Programação concorrente...

- ... 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

O desafio cognitivo da computação concorrente

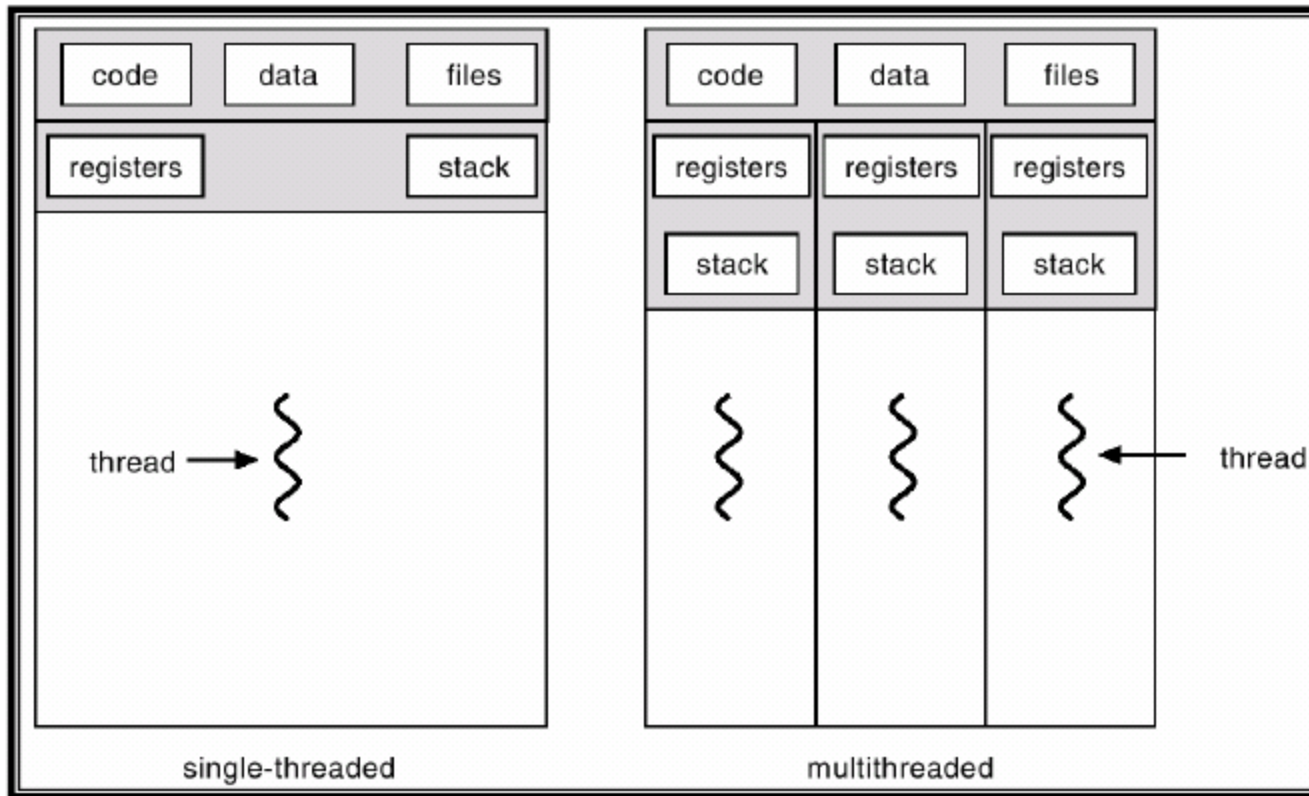
- Muitas coisas ocorrendo ao mesmo tempo!
- Não dá para saber quem acontece primeiro!



Processos e threads

- Processos
 - Execução paralela ou concorrente
 - **Espaços de endereçamento separados**
 - Compartilham algumas estruturas (tabela de descritores de arquivo, etc)
- Threads
 - Executam no mesmo processo
 - Mesmo espaço de endereçamento
 - **Compartilham memória**

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!

Processos



Main page
Product releases
New pages
Recent changes
Recent uploads
Random page
Help

How to Contribute
All-hands meeting
Other meetings
Contribute to Mozilla
Mozilla Reps
Student Ambassadors

MozillaWiki
News
About
Team
Policies
Releases
@MozillaWiki
Report a wiki bug

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

Sandbox Architecture

Multi-process Firefox employs a process sandbox to protect against malicious content. In this model, untrusted content is run in a sandboxed low-rights process so that in the event of a compromise, access to full system functionality and data is prevented by a sandbox. This document aims to provide an overview of the sandbox implementation and outline the design implications for Gecko features.

Processos



The Chromium Projects

Search this site

[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.

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 (*pthread*) que inclui

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

Atividade

Vamos criar algumas threads e resolver problemas simples.

Parte 1

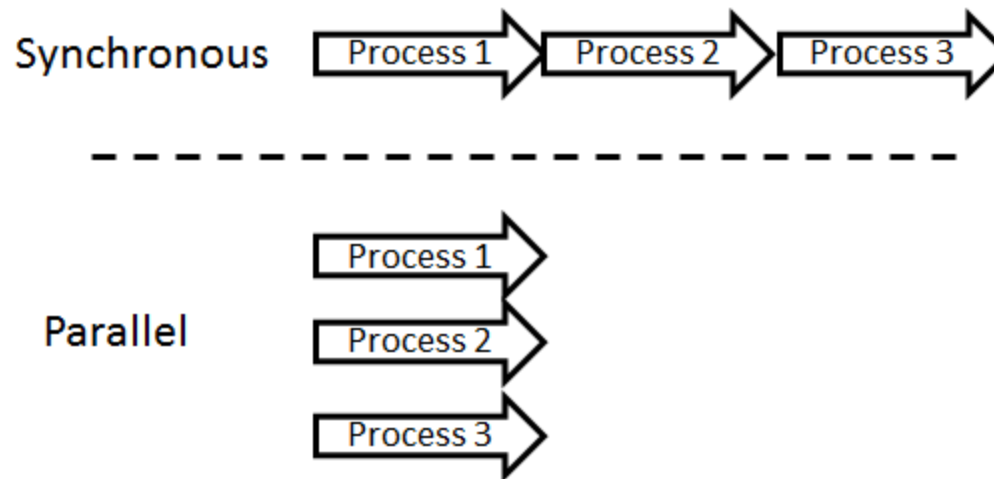
- Cada thread recebe um id diferente
- Não existe ordenação.
- Cada vez os prints aparecem em uma ordem

Partes 2 e 3

- Threads recebem um endereço de memória como parâmetro
- Podemos passar vários argumentos usando *struct*
- Podemos devolver valores setando campos deste *struct*
- Os dados passados devem ser alocados dinamicamente

Parte 4

- Vamos discutir o resultado na próxima aula



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

Insper

www.insper.edu.br