Insper

Sistemas Hardware-Software

Aula 13 – Processos

2021 – Engenharia

Maciel Vidal
Igor Montagner
Fábio Ayres

Sistemas Operacionais

“Permitir que um usuário execute diversos programas de maneira “simultânea” e segura.”

Sistemas Operacionais

Kernel: software do sistema que gerencia

- Programas
- Memória
- Recursos do hardware

Roda com privilégios totais no hardware. Grosso modo, é um conjunto de *handlers* de interrupção.

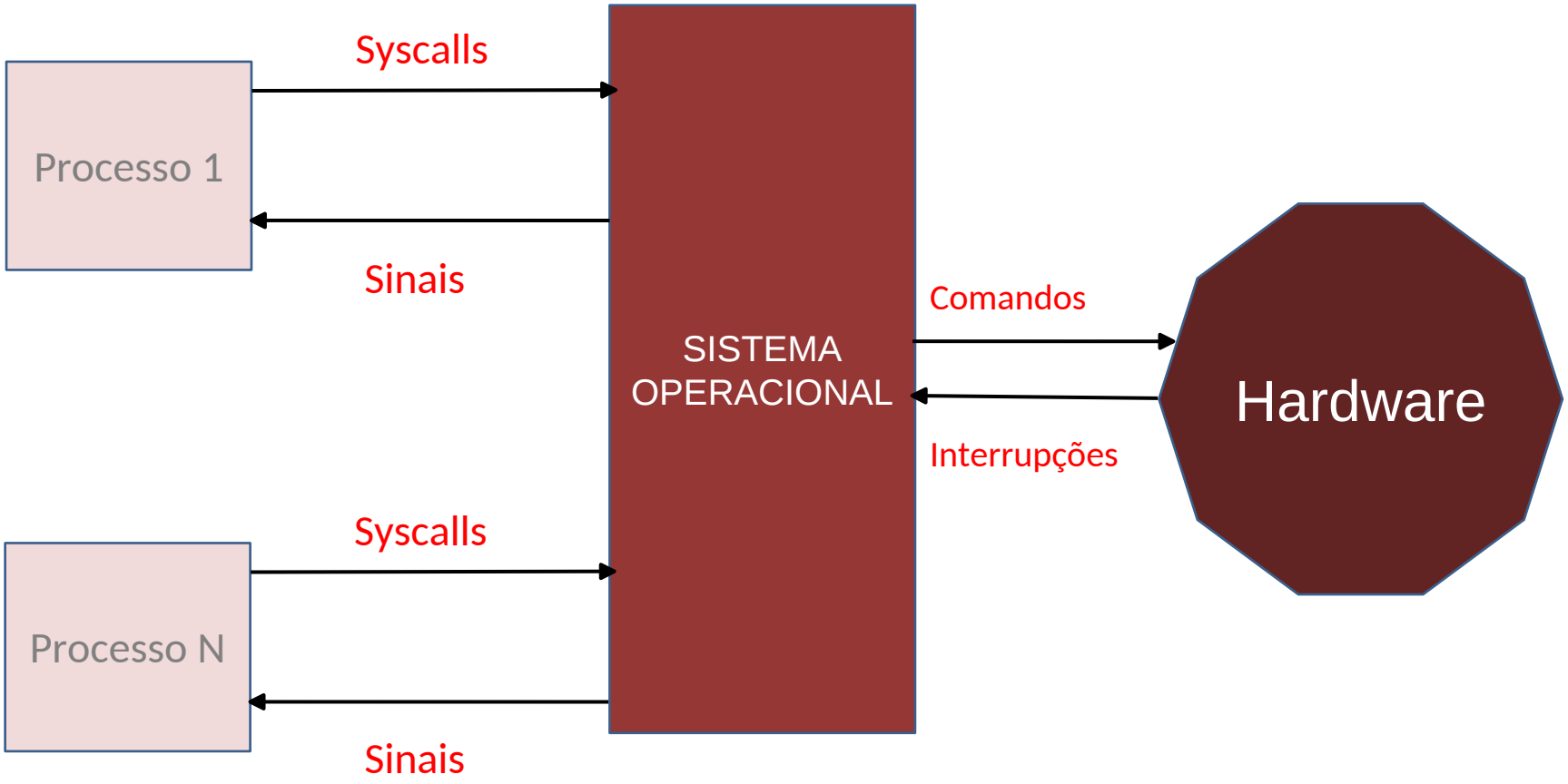
Sistemas Operacionais

Processo de usuário: qualquer programa sendo executado no computador. **A falha de um processo não afeta os outros.**

Roda com **privilégios limitados**. Interage com o hardware por meio de **chamadas ao kernel** para obter

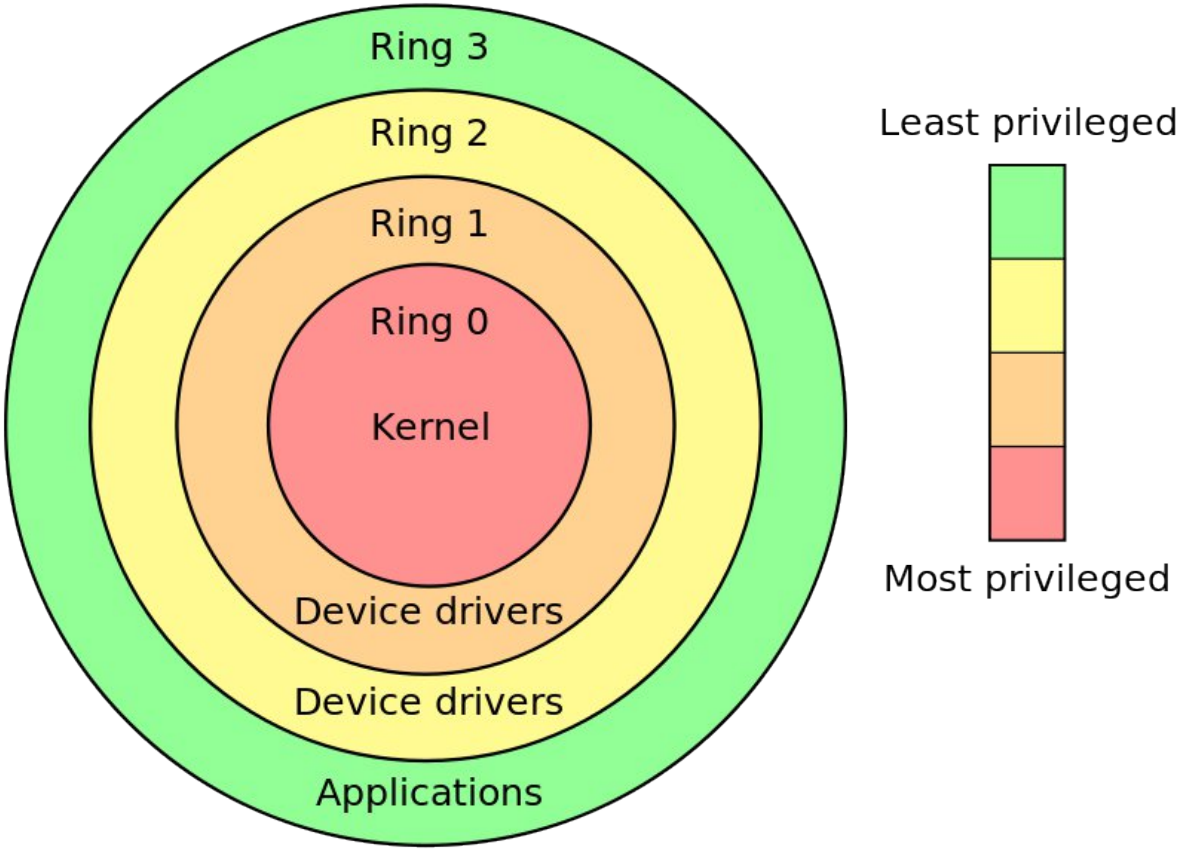
- Memória
- Acesso ao disco e outros periféricos
- Comunicar com outros processos

Interação de processos com Hardware



Insper

Níveis de proteção em x86



https://en.wikipedia.org/wiki/Protection_ring

Insper

POSIX

The **Portable Operating System Interface** (POSIX) is a family of standards specified by the IEEE Computer Society for maintaining compatibility between operating systems. POSIX defines the **application programming interface (API)**, along with command line shells and utility interfaces, for software compatibility with variants of Unix and other operating systems

1) Wikipedia

POSIX

- O SO disponibiliza uma API para interagir com o hardware
- POSIX é uma API que pode ser implementada por vários sistemas diferentes
  - Linux
  - MacOS
  - Haiku
  - Windows
- Sistemas *POSIX compliant* são compatíveis em nível de código fonte
- Arquivo de cabeçalho <unistd.h>

Windows API

- Windows também disponibiliza sua API própria
- WINE: implementação da API windows via POSIX:
    - “Instead of simulating internal Windows logic like a virtual machine or emulator, Wine translates Windows API calls into POSIX calls on-the-fly” (winehq.org)
- Cygwin: implementação de API POSIX em cima da API windows. É necessário recompilar código para funcionar.
- Documentação: https://msdn.microsoft.com/en-us/library/windows/desktop/ff818516(v=vs.85).aspx

POSIX

- Gerenciamento de usuários e grupos
- Manipulação de arquivos (incluindo permissões) e diretórios
- Criação de processos e carregamento de programas
- Comunicação entre processos
- Interação direta com hardware (via drivers)

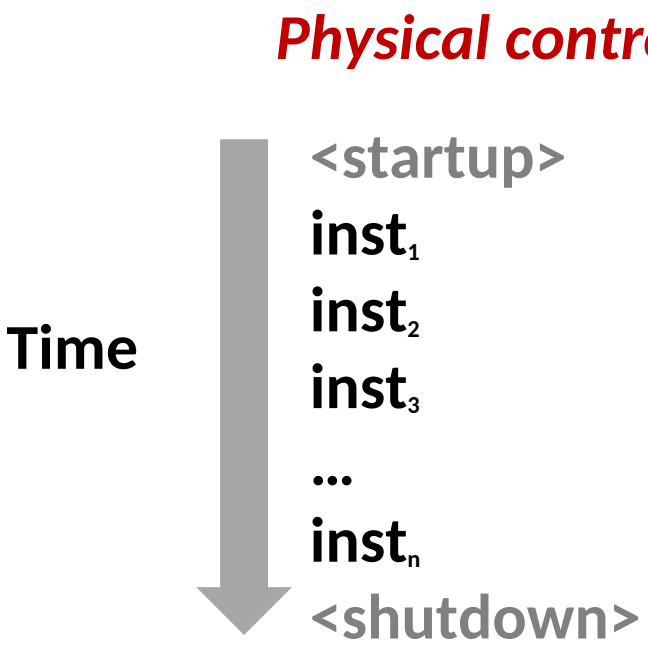
Hoje

- Definir o que é um processo e seu contexto
- Entender quais mecanismos de hardware são usados para alternar entre processos
- Analisar o ciclo de vida de um processo

Processos

Fluxo de controle

- Desde o início até o seu desligamento, a CPU apenas lê e executa uma sequência de instruções, uma por vez
- Esta sequência é o fluxo de controle da CPU



Insper

Alterando o fluxo de controle

Até o momento, temos dois mecanismos para alterar o fluxo de controle:

- Saltos (*jumps*: **`jmp`**) e desvios (*branches*: **`je, jl, jge`**, etc)

- Chamadas (calls) e retornos

Permitem alterar o fluxo de controle em função de mudanças no estado do programa

Alterando o fluxo de controle

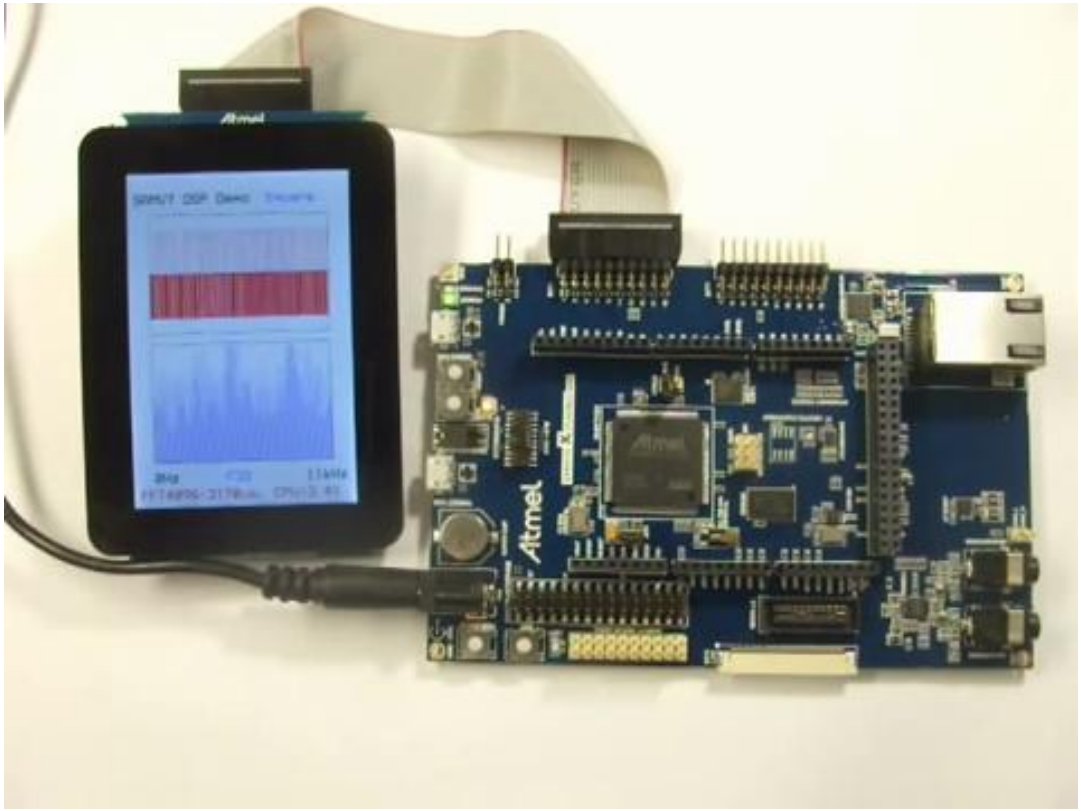
Mas isto não basta: como reagir a mudanças no estado do sistema?

- Dados lidos do disco ou da rede
- Programa executa uma instrução ilegal ou em condições inválidas (como divisão por zero)
- Usuário digita Ctrl-C no teclado
- Timer de sistema notifica o programa

Precisamos de mecanismos para reagir a estes eventos "excepcionais"

Interrupções

Muito usadas em Embarcados



Insper

Contextos de aplicação

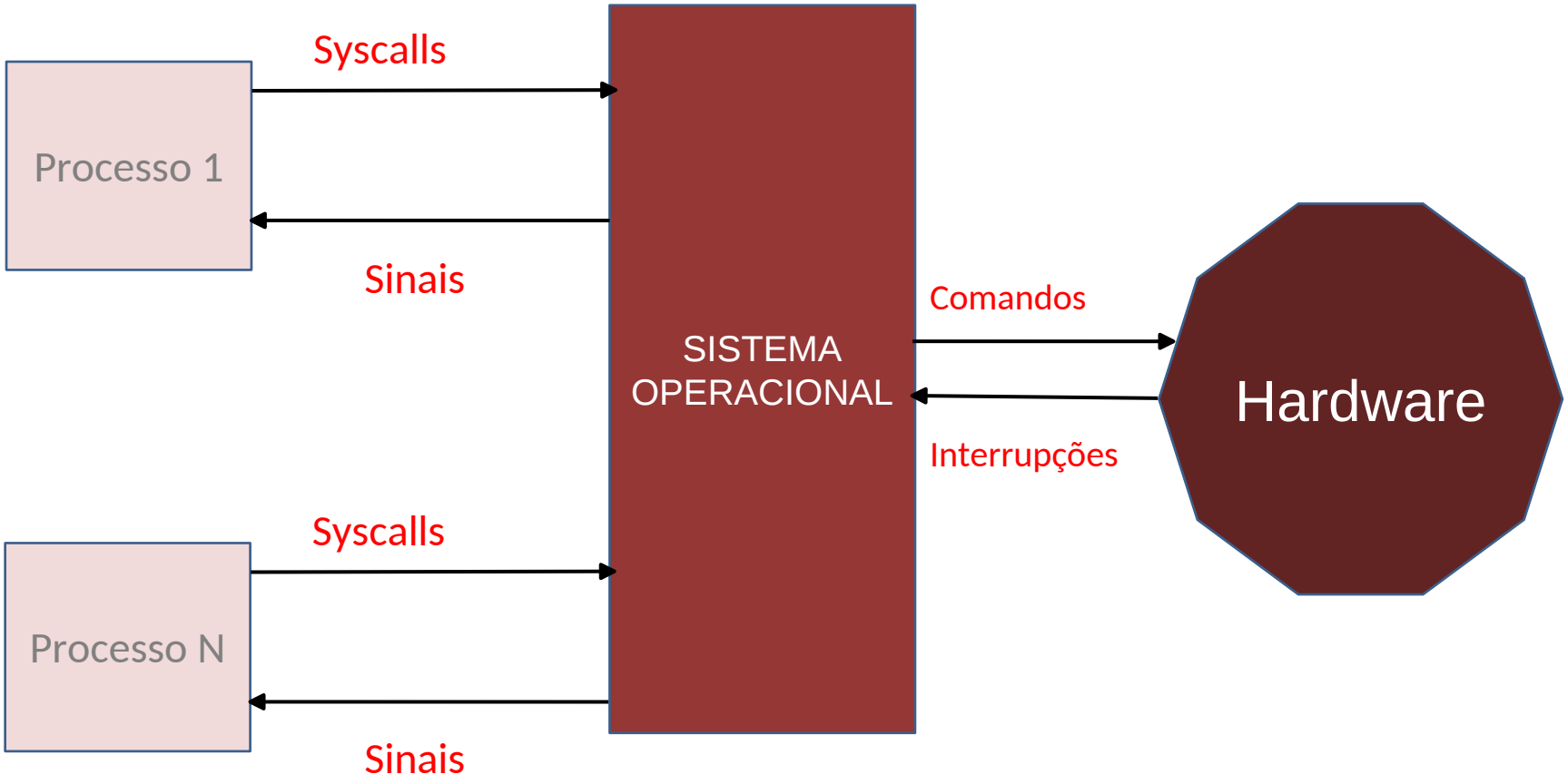
Embarcados (firmware):

- Somente um programa rodando, mas com várias tarefas concorrentes
- Tarefas compartilham espaço de memória

Desktop/Celular:

- Vários programas (não confiáveis) rodando
- Programas começam e terminam a qualquer momento
- Isolamento de memória e recursos

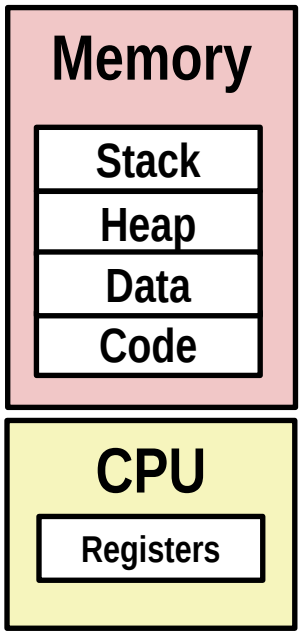
Interação de processos com Hardware



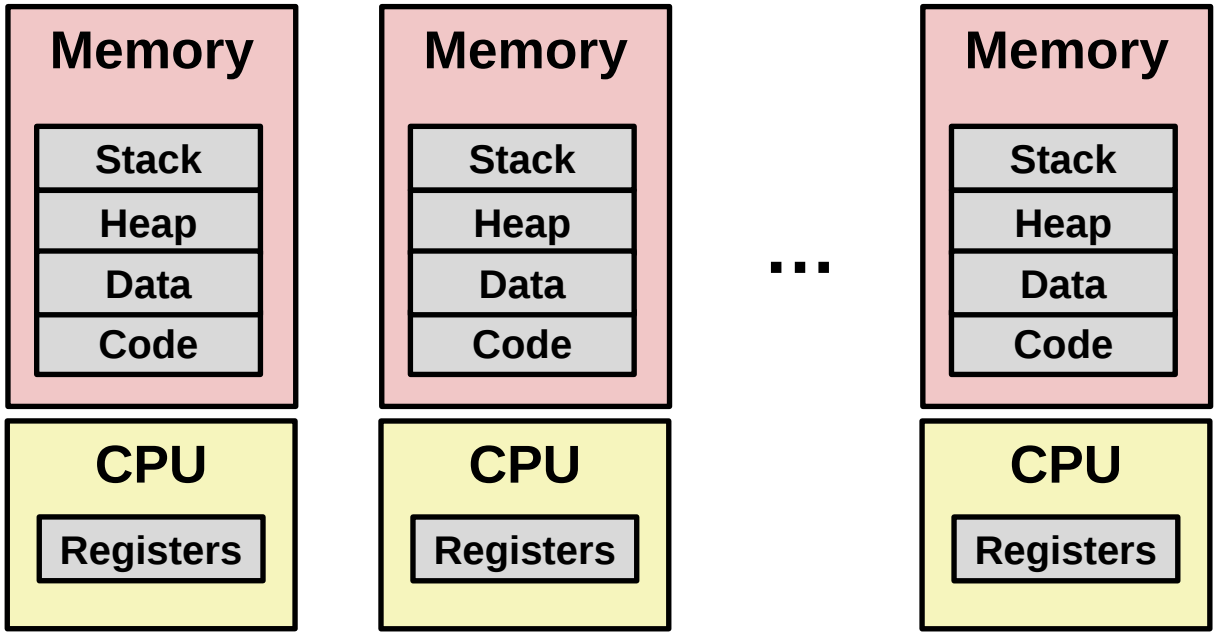
Insper

Processos

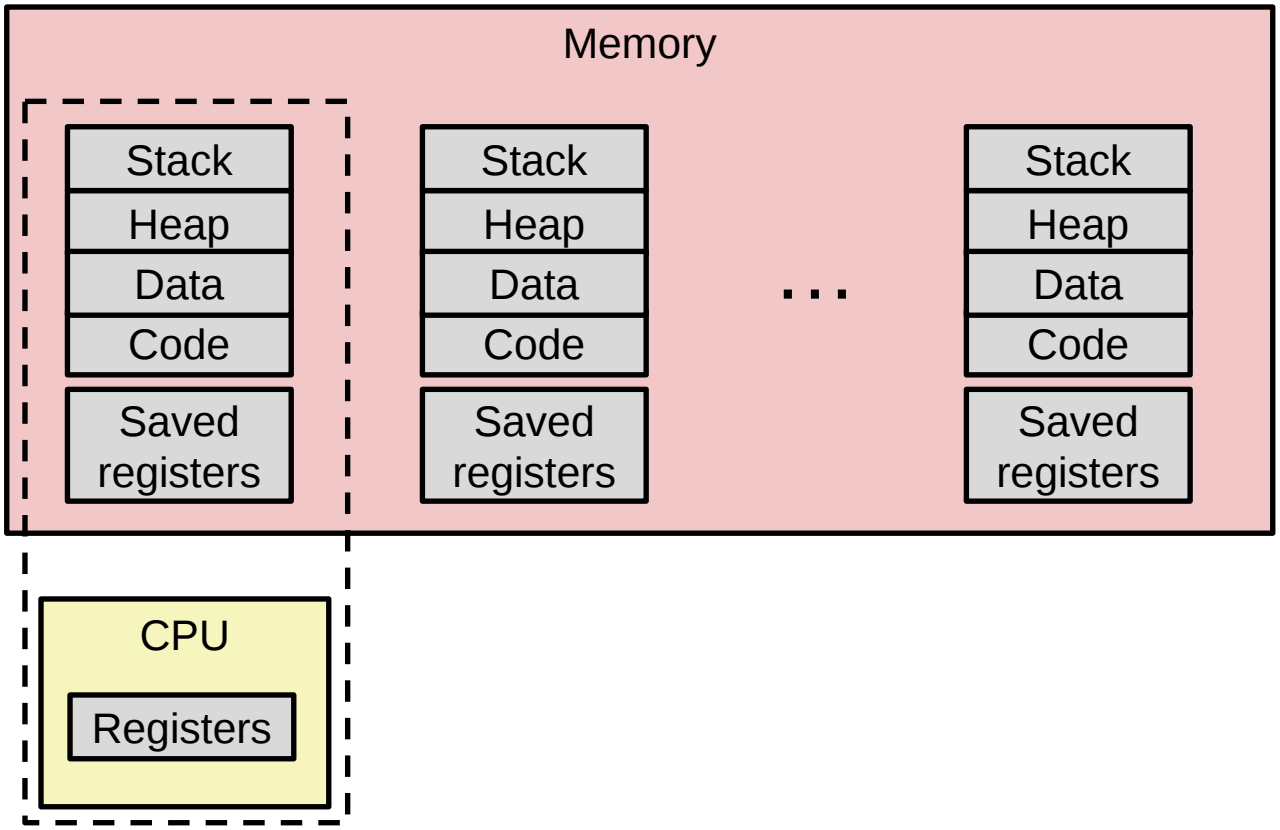
- Fluxo de controle lógico
  - Cada programa parece ter uso exclusivo da CPU
  - Cada programa parece ter uso exclusivo da memória principal



A ilusão do multiprocessamento

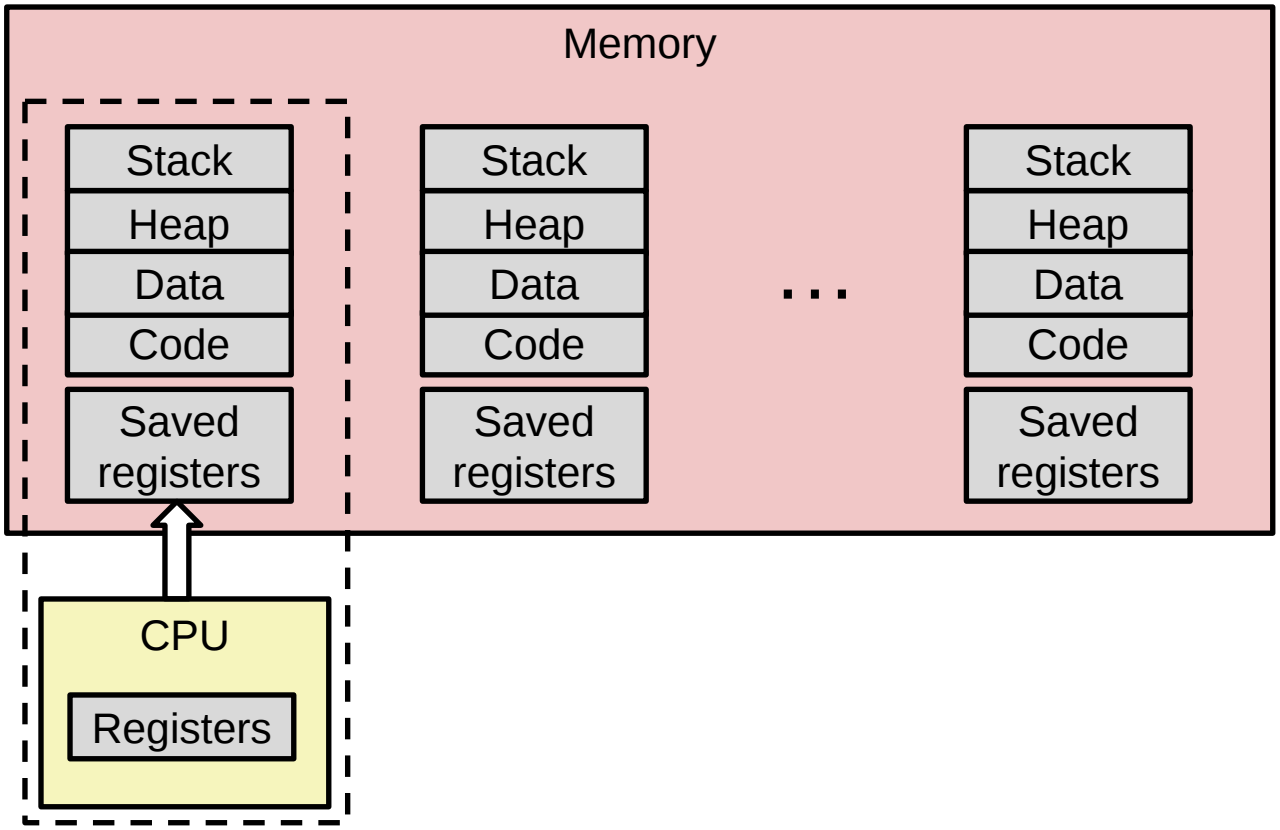


A realidade do multiprocessamento



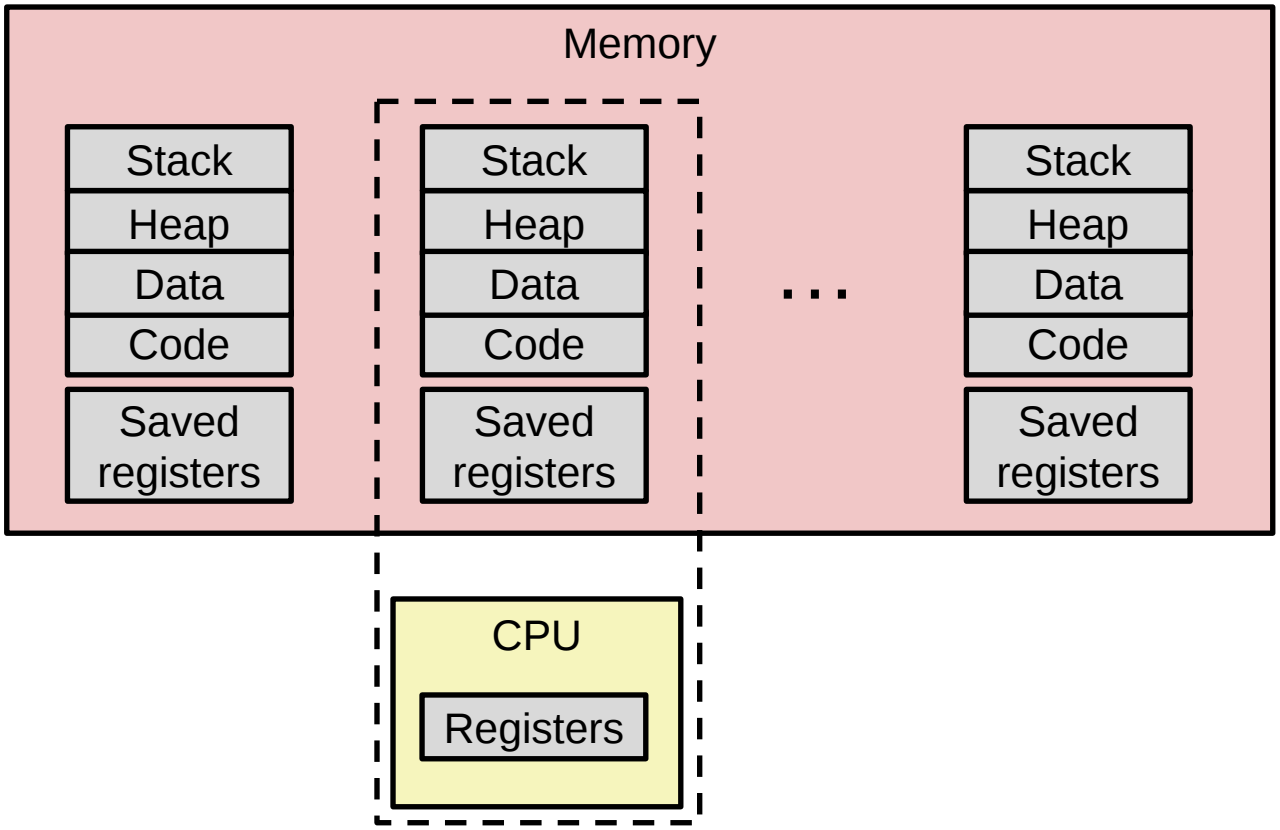
- Execução de processos intercalada
- Espaços de endereçamento gerenciados pelo sistema de memória virtual
- Valores de registradores para processos em espera são gravados em memória

A realidade do multiprocessamento



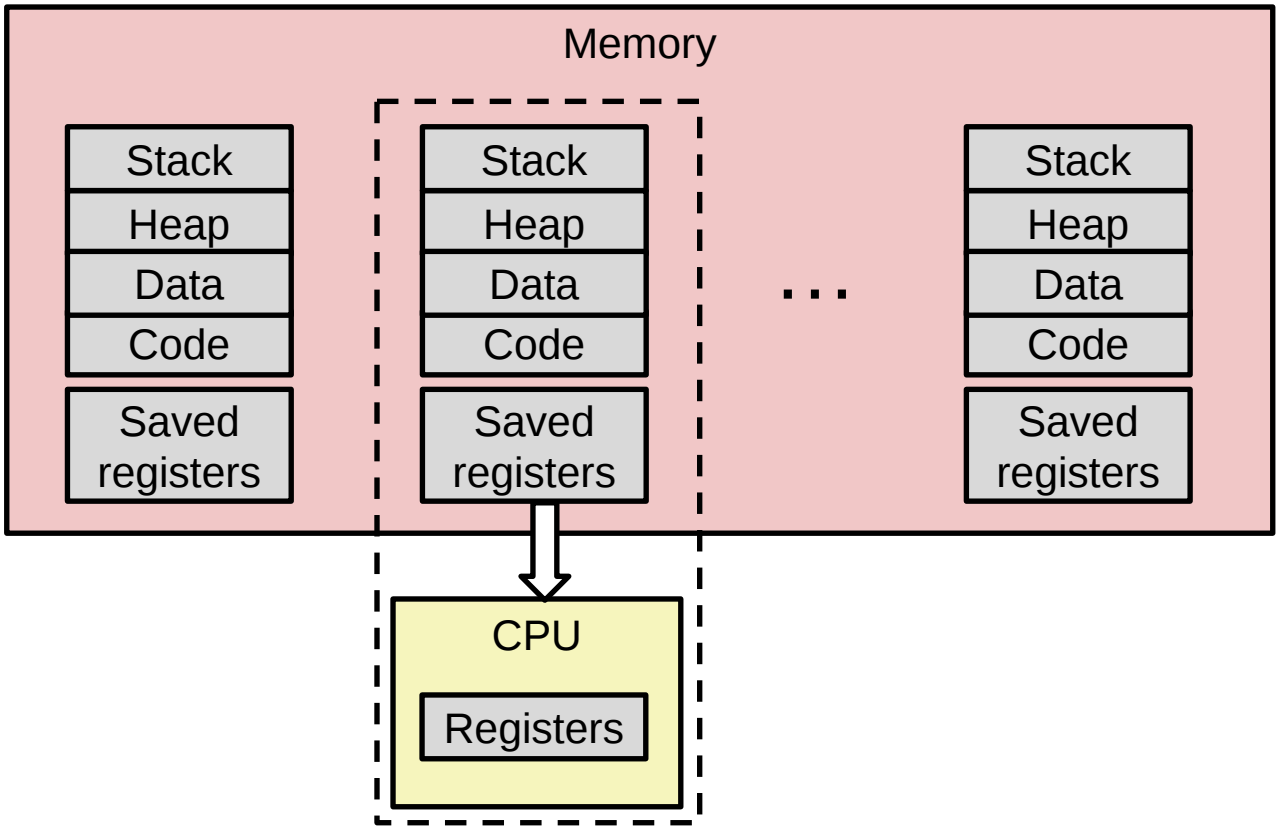
- Grava registradores na memória

A realidade do multiprocessamento



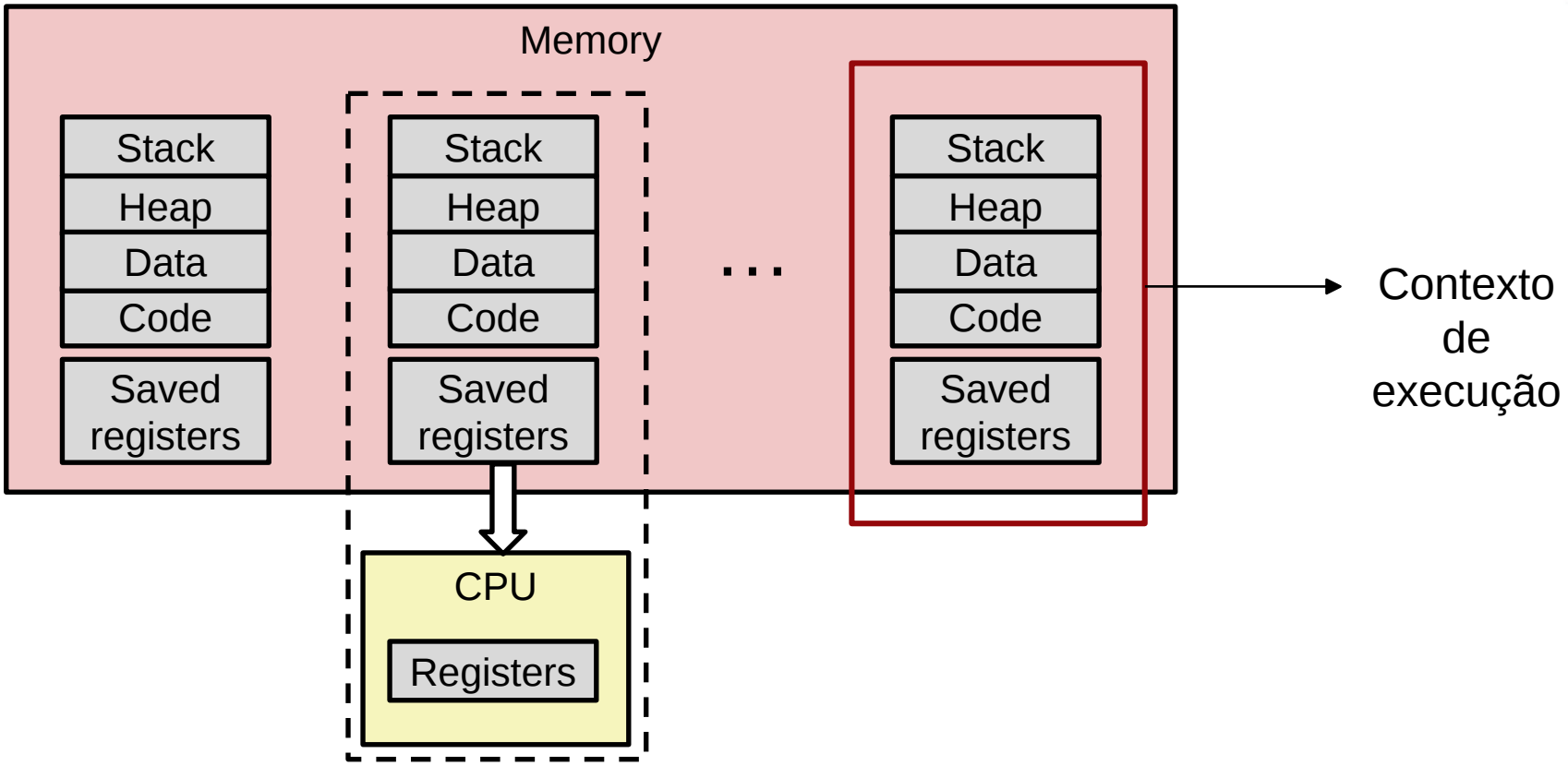
- Escolhe próximo processo a ser executado

A realidade do multiprocessamento



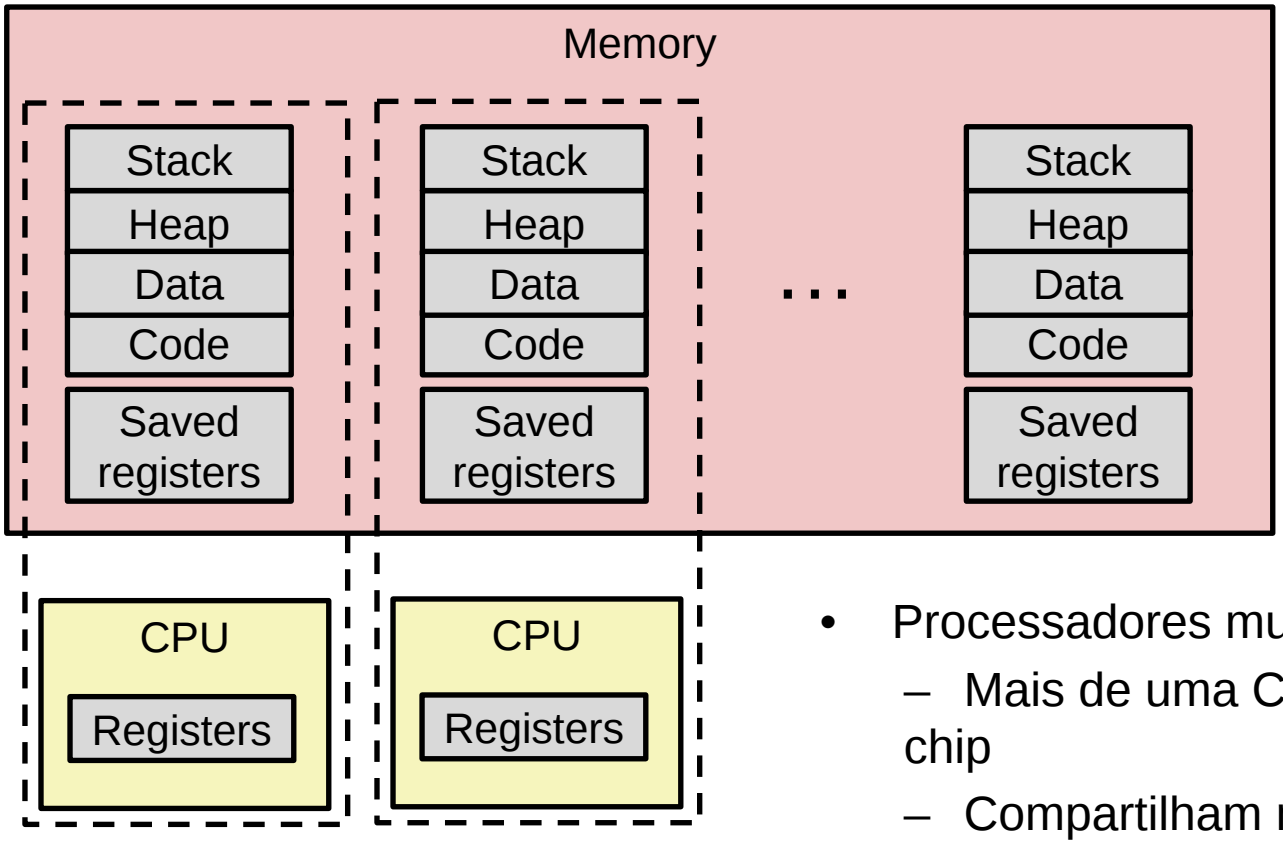
- Carrega registradores gravados e troca de espaço de endereçamento (*context switch* – chaveamento de contexto)

A realidade do multiprocessamento



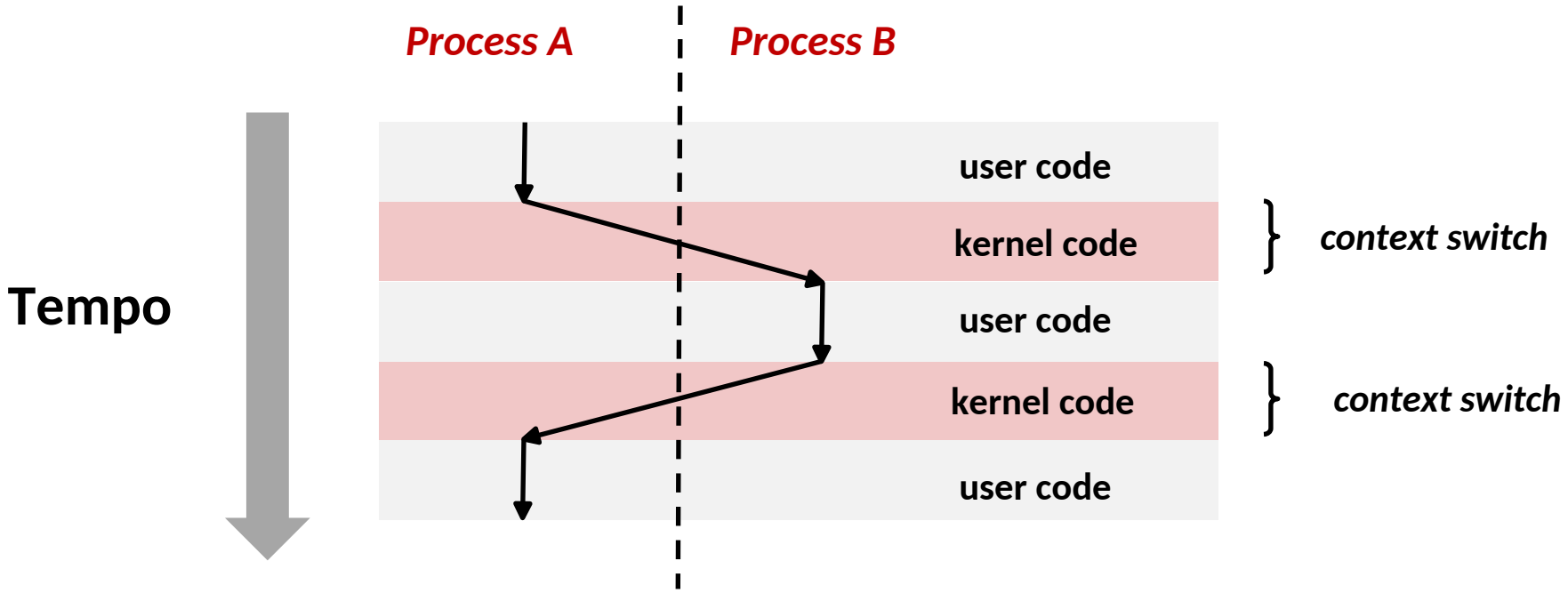
- Carrega registradores gravados e troca de espaço de endereçamento (*context switch* – chaveamento de contexto)

A realidade moderna do multiprocessamento

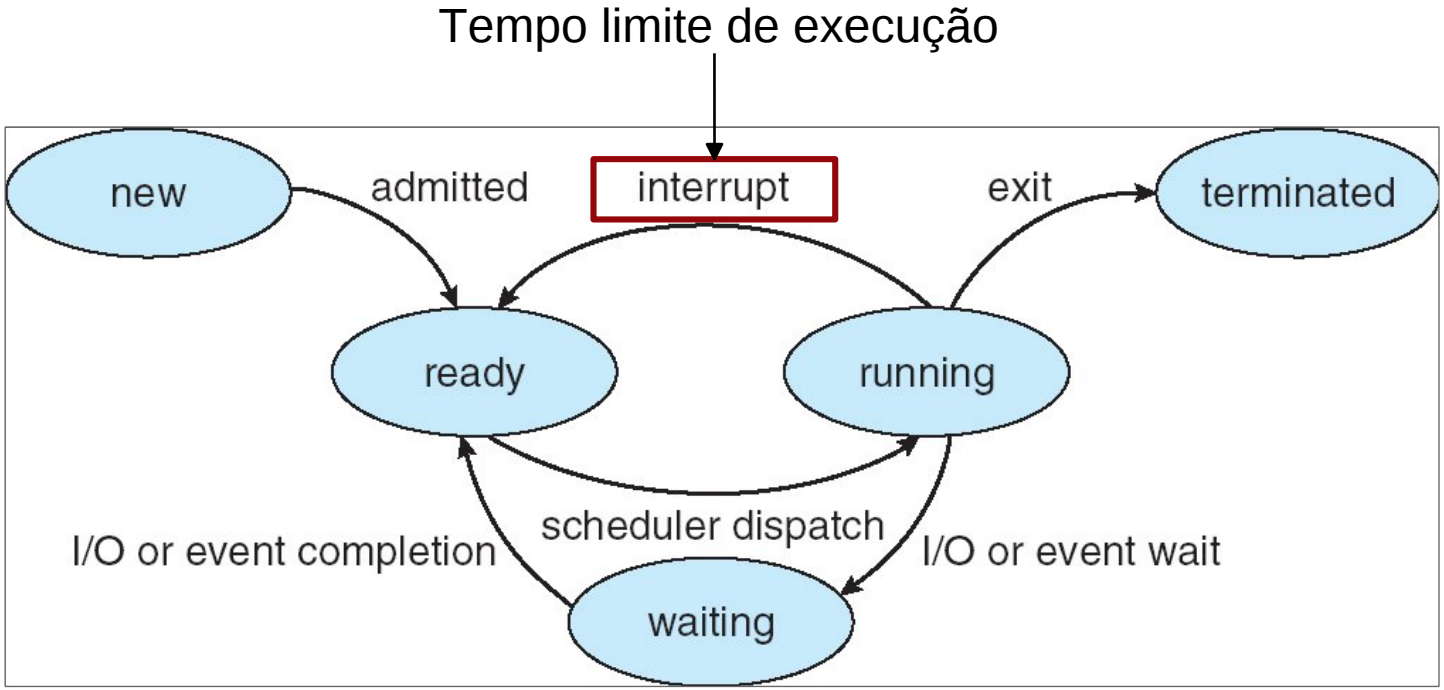


- Processadores multicore
  - Mais de uma CPU em um mesmo chip
  - Compartilham memória principal e parte do cache (cache L3)
  - Cada core pode executar um processo separado
    - Agendamento de processos em cores feito pelo kernel

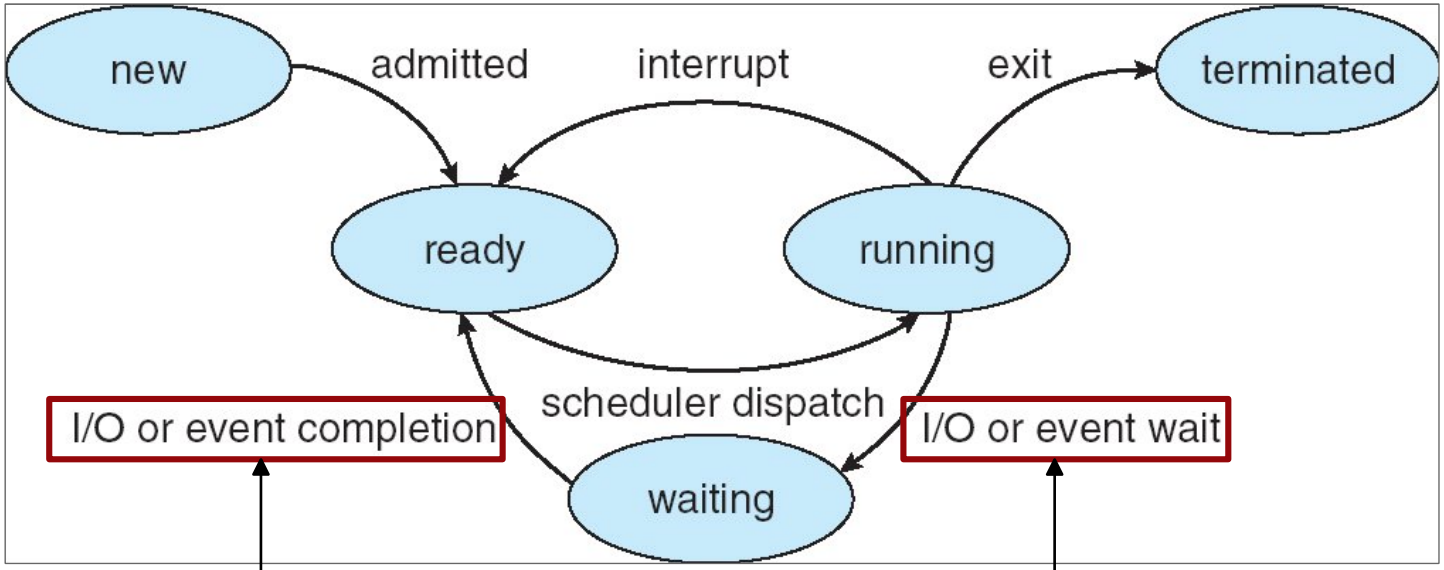
Chaveamento de contexto



Ciclo de vida de processos



Ciclo de vida de processos



Interrupção ligada por algum hardware (disco, rede, usb, etc)

Chamada de sistema Para acessar hardware

Insper

Criação de processos

Criamos processos usando a chamada de sistema *fork*

```
pid_t fork();
```

O fork cria um clone do processo atual e retorna duas vezes

No processo original (pai) `fork` retorna o pid do filho

O pid do pai é obtido chamando

```
pid_t getpid();
```

No processo filho `fork` retorna o valor `0`.
O pid do filho é obtido usando

```
pid_t getpid();
```

O pid do pai pode ser obtido usando a chamada

```
pid_t getppid();
```

Atividade prática

A chamada fork

1. Criação de processos
2. Utilização do manual para dúvidas sobre as chamadas

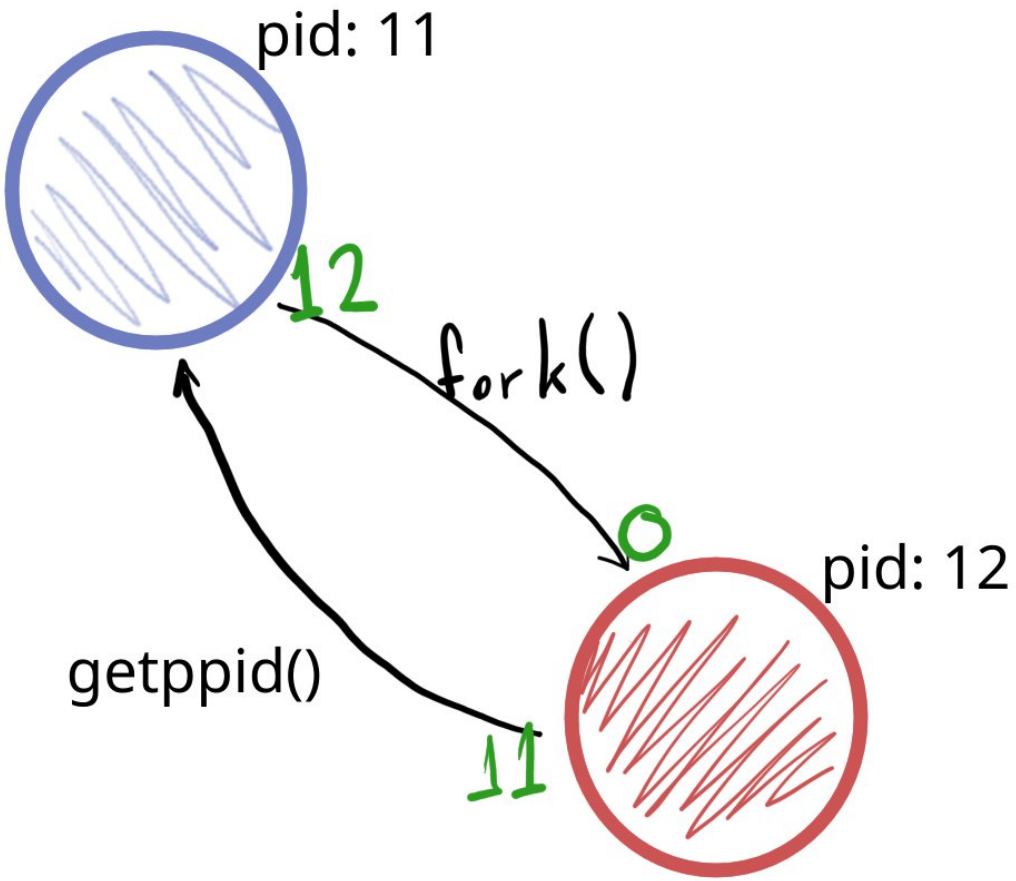
Criação de processos (exercício)

```c
int rodando = 1;
pid_t filho;

filho = fork();

if (filho == 0) {
    printf("Acabei filho\n");
    rodando = 0;
} else {
    while (rodando) {
        printf("Esperando o filho acabar!\n");
        sleep(1);
    }
}
return 0;
```

Parentesco de processos



Insper

Valor de retorno

- Um processo pode esperar pelo fim de outro processo filho usando as funções

```
pid_t wait(int *wstatus);
pid_t waitpid(pid_t pid, int *wstatus, int options);
```

- A primeira espera qualquer um dos filhos, enquanto a segunda espera um filho (ou grupo de filhos) específico.

- Ambas bloqueiam até que um processo filho termine e retornam o pid do processo que acabou de terminar.

- O valor de retorno do processo é retornado via o ponteiro `wstatus.`

E se o processo filho deu ruim?

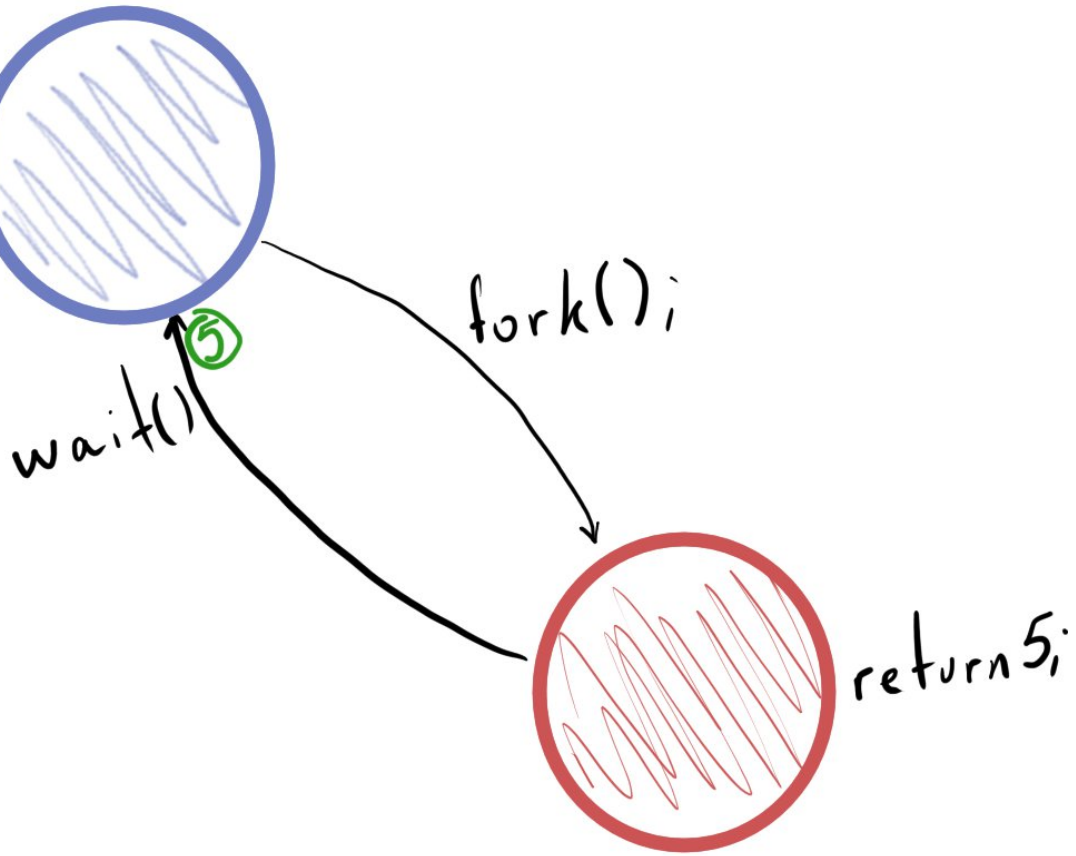
- É possível checar se um processo filho terminou corretamente usando o conteúdo de `wstatus` e as seguintes macros:
- `WIFEXITED(wstatus)`: `true` se o filho acabou sem erros
- `WEXITSTATUS(wstatus)`: valor retornado pelo main
- `WIFSIGNALED(wstatus)`: `true` se o filho foi terminado de maneira abrupta (tanto por um ctrl+c quanto por um erro)
- `WTERMSIG(wstatus)`: código numérico representando a razão do encerramento do filho

Atividade prática

A chamada wait

1. Criação de processos
2. Identificação de término de processos
3. Utilização do manual para dúvidas sobre as chamadas

Parentesco de processos – II



Como executar novos programas?

- **fork** só permite a criação de **clones** de um processo!
- Família de funções **exec** permite o carregamento de um programa do disco
- É permitido setar as variáveis de ambiente do novo programa e seus argumentos.
- Funções da família **exec** nunca retornam: o programa atual é destruído durante o carregamento do novo programa

Insper

www.insper.edu.br