

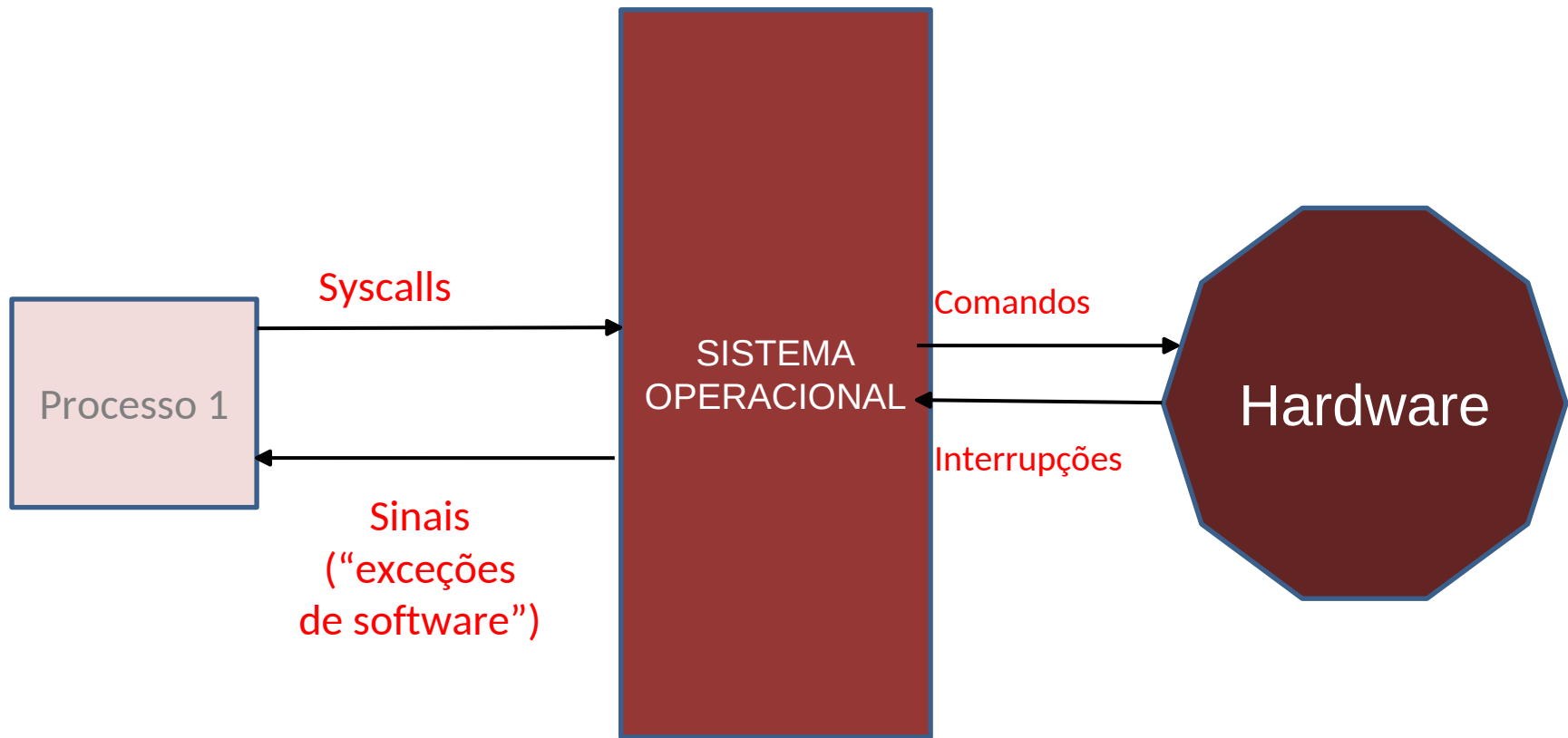
Sistemas Hardware-Software

Aula 14 – Processos

2020 – Engenharia

[Igor Montagner](#), Fábio Ayres

Chamadas de sistema



POSIX - Arquivos

- open, close, read, write
- Todo arquivo pertence a um usuário e a grupo.
- Permissões: **dono do arquivo, membros do grupo dono e resto.**
 - Codificação das permissões:
 - 4 para leitura
 - 2 para escrita
 - 1 para execução

Permissões na prática: Android

FEATURES TESTING BEST PRACTICES

AOSP > Secure > Features

☆☆☆☆☆

Application Sandbox

The Android platform takes advantage of the Linux user-based protection to identify and isolate app resources. This isolates apps from each other and protects apps and the system from malicious apps. To do this, Android assigns a unique user ID (UID) to each Android application and runs it in its own process.

Android uses this UID to set up a kernel-level Application Sandbox. The kernel enforces security between apps and the system at the process level through standard Linux facilities, such as user and group IDs that are assigned to apps. By default, apps can't interact with each other and have limited access to the operating system. For example, if application A tries to do something malicious, such as read application B's data or dial the phone without permission (which is a separate application), then the operating system protects against this behavior because application A does not have the appropriate user privileges. The sandbox is simple, auditable, and based on decades-old UNIX-style user separation of processes and file permissions.

<https://source.android.com/security/app-sandbox>

Até agora

Um programa tem acesso total aos recursos da máquina:

- Pode ocupar toda RAM (acessar todos os 2^{64} endereços de memória)
- Tem uso exclusivo de todos os registradores
- Tem uso exclusivo do tempo da CPU
- Tem acesso instantâneo ao disco e à rede

Até agora...

Um programa tem acesso total aos recursos da máquina:

- ~~Pode ocupar toda RAM (acessar todos os 2^{64} endereços de memória)~~
- ~~Tem uso exclusivo de todos os registradores~~
- ~~Tem uso exclusivo do tempo da CPU~~
- ~~Tem acesso instantâneo ao disco e à rede~~

Como o SO faz isso?

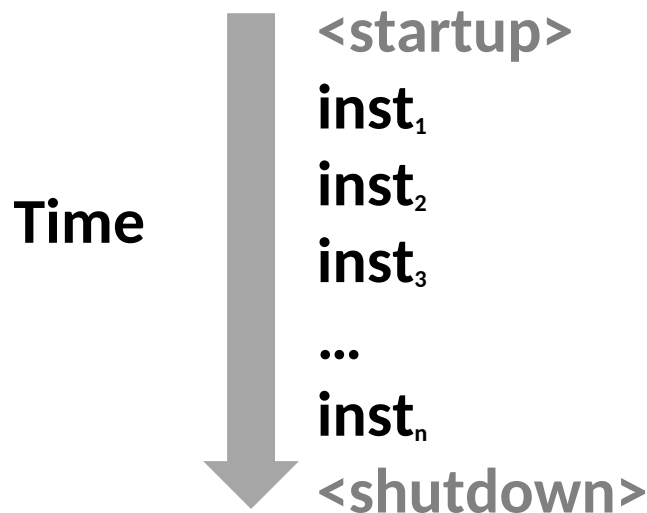
Hoje

- 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

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

Physical control flow



Alterando o fluxo de controle

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

- Saltos (*jumps*: **jmp**) e desvios (*branches*: **je**, **j1**, **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



Contextos de aplicação

Embarcados:

- 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

Sistemas Operacionais de uso geral

Kernel: software do sistema que gerencia

- Programas
- Memória
- Recursos do hardware

Roda com privilégios totais no hardware.

Handlers para todas interrupções

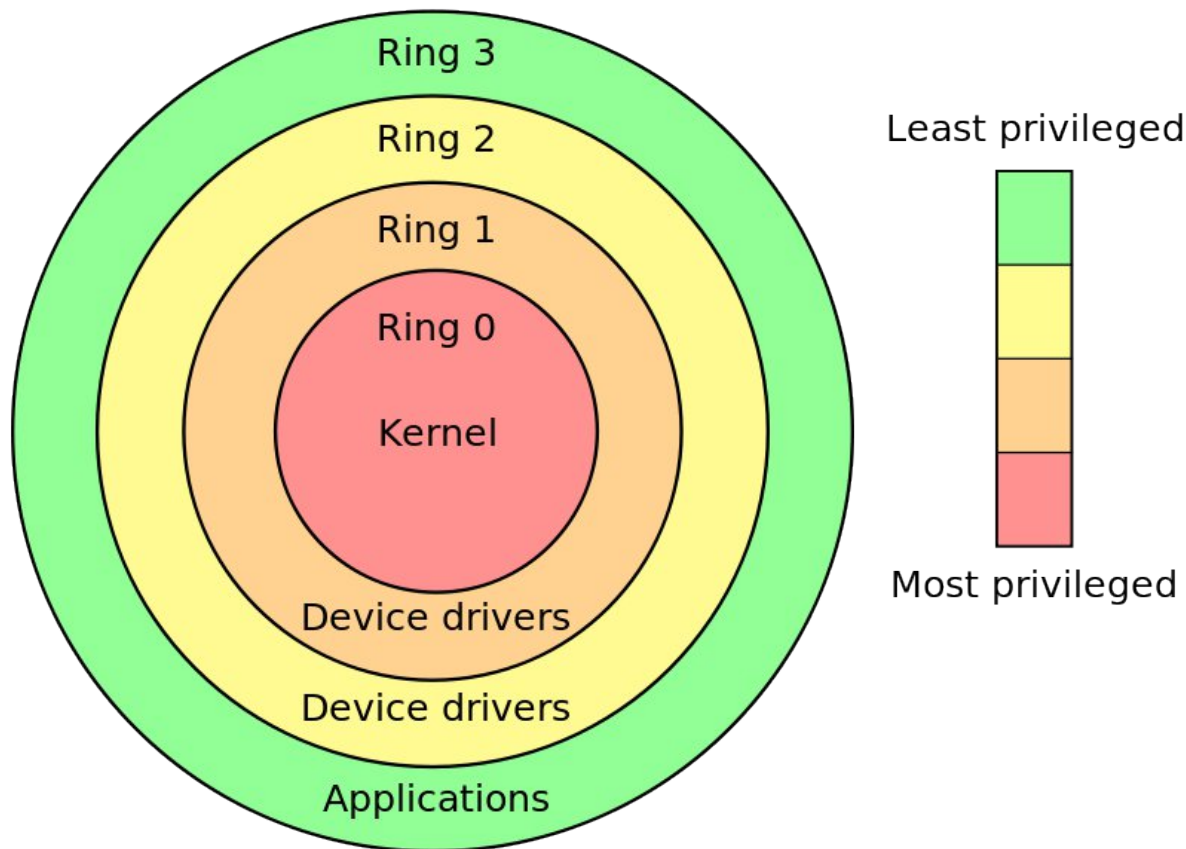
Sistemas Operacionais de uso geral

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

Roda com privilégios limitados (ring 3 ou *user land*).
Interage com o hardware por meio de chamadas ao kernel.

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

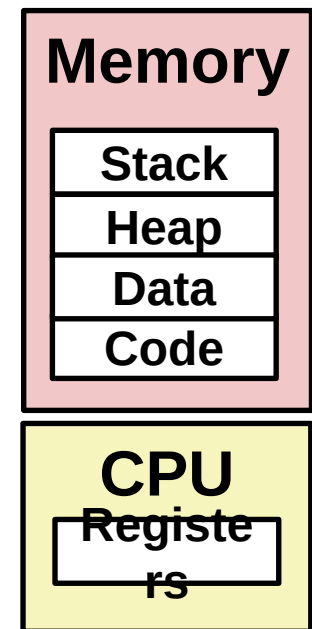
Níveis de proteção em x86



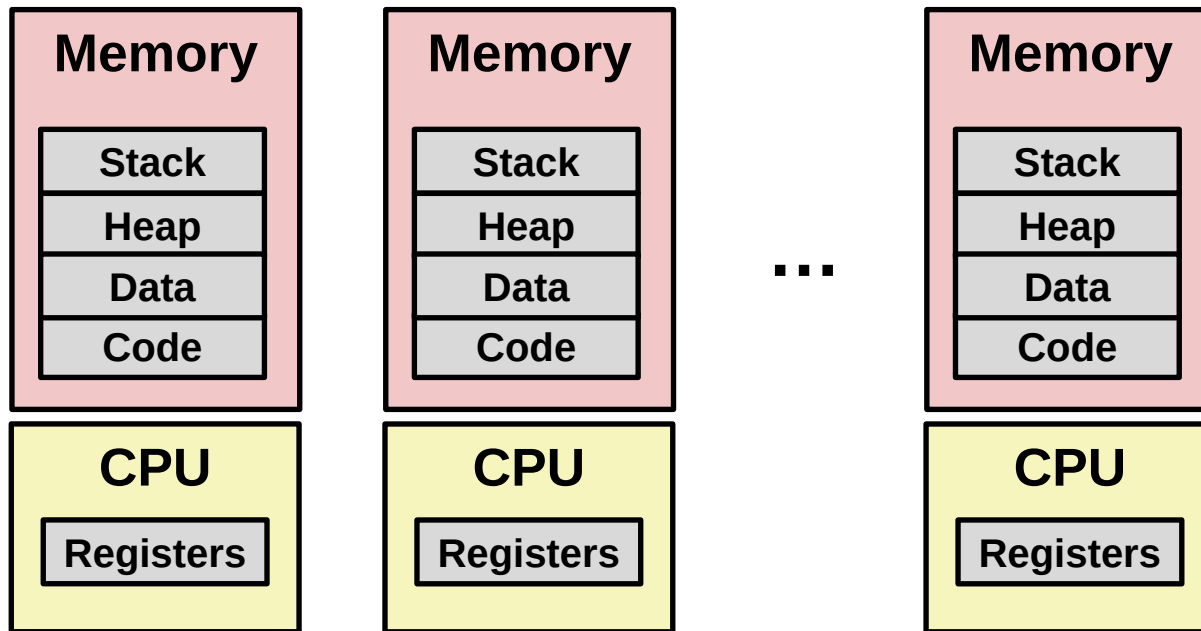
https://en.wikipedia.org/wiki/Protection_ring

Processos

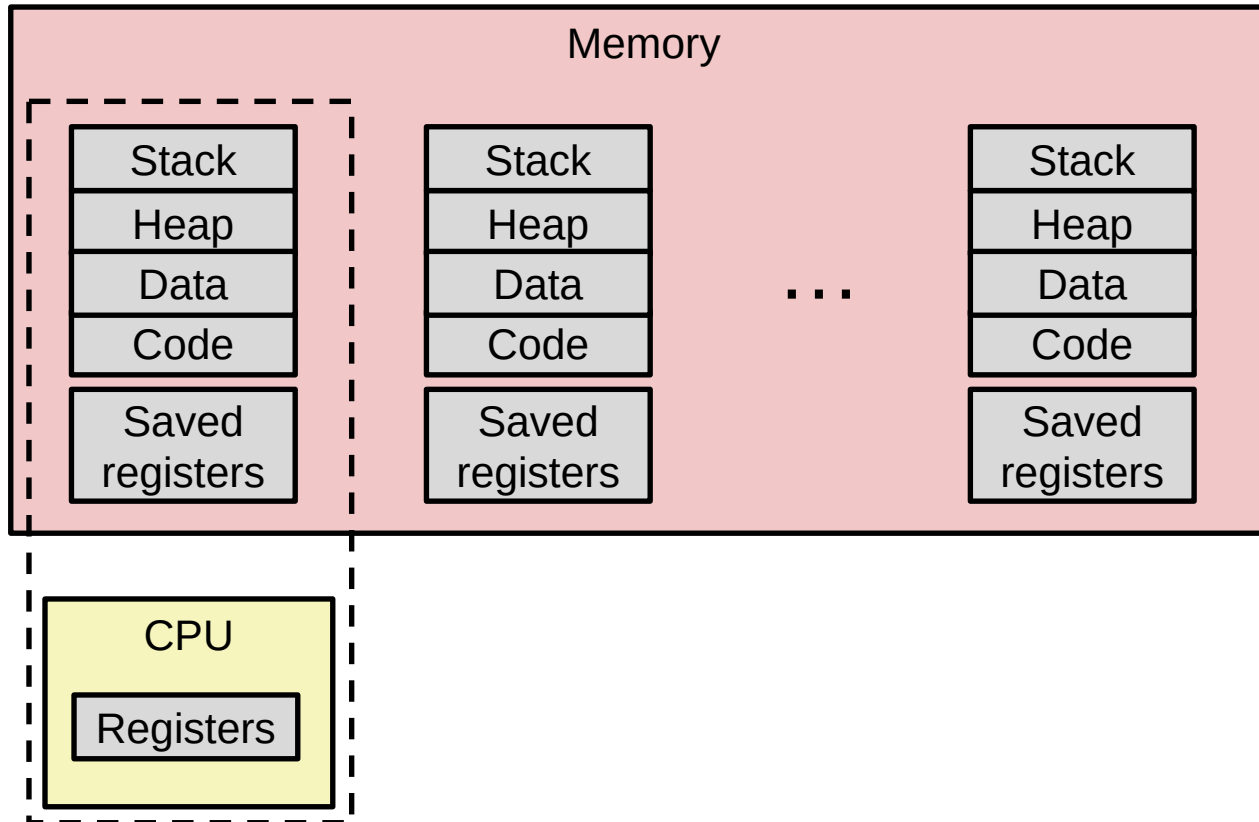
- Fluxo de controle lógico
 - Cada programa parece ter uso exclusivo da CPU
 - Provido pelo mecanismo de *chaveamento de contexto*
 - Espaço de endereçamento privado
 - Cada programa parece ter uso exclusivo da memória principal
 - Provido pelo mecanismo de *memória virtual*



A ilusão do multiprocessamento

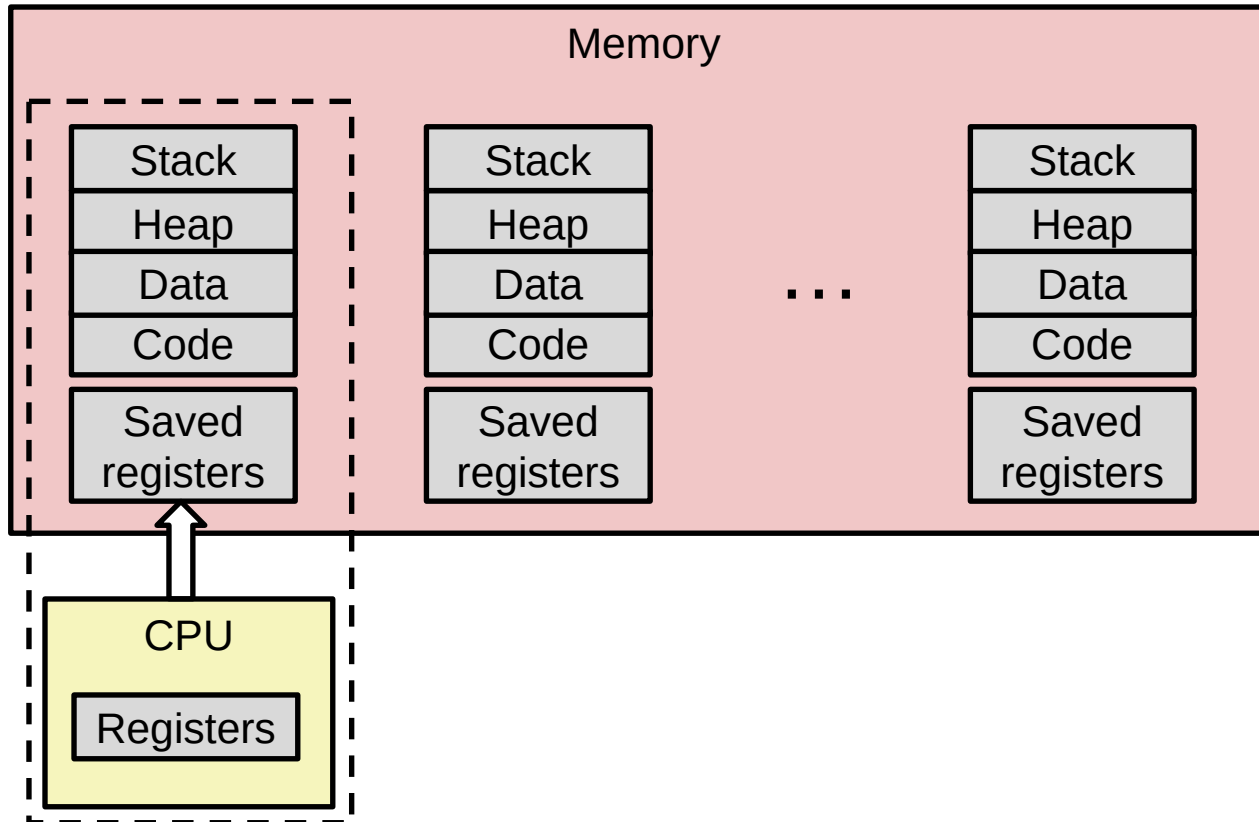


A realidade do multiprocessamento



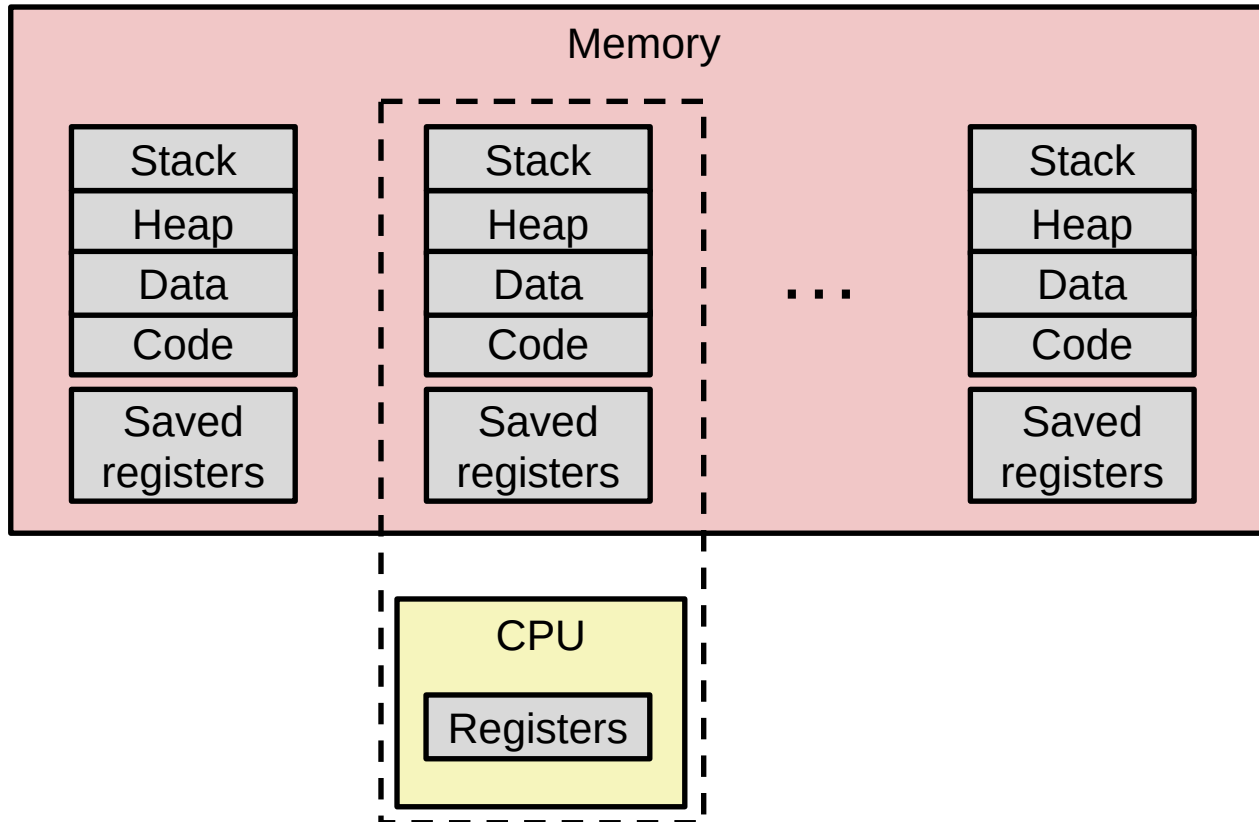
- Execução de processos intercalada
- Espaços de endereçamento gerenciados pelo sistema de memória virtual
- Valores de registradores para processos em espera são gravados em memória

A realidade do multiprocessamento



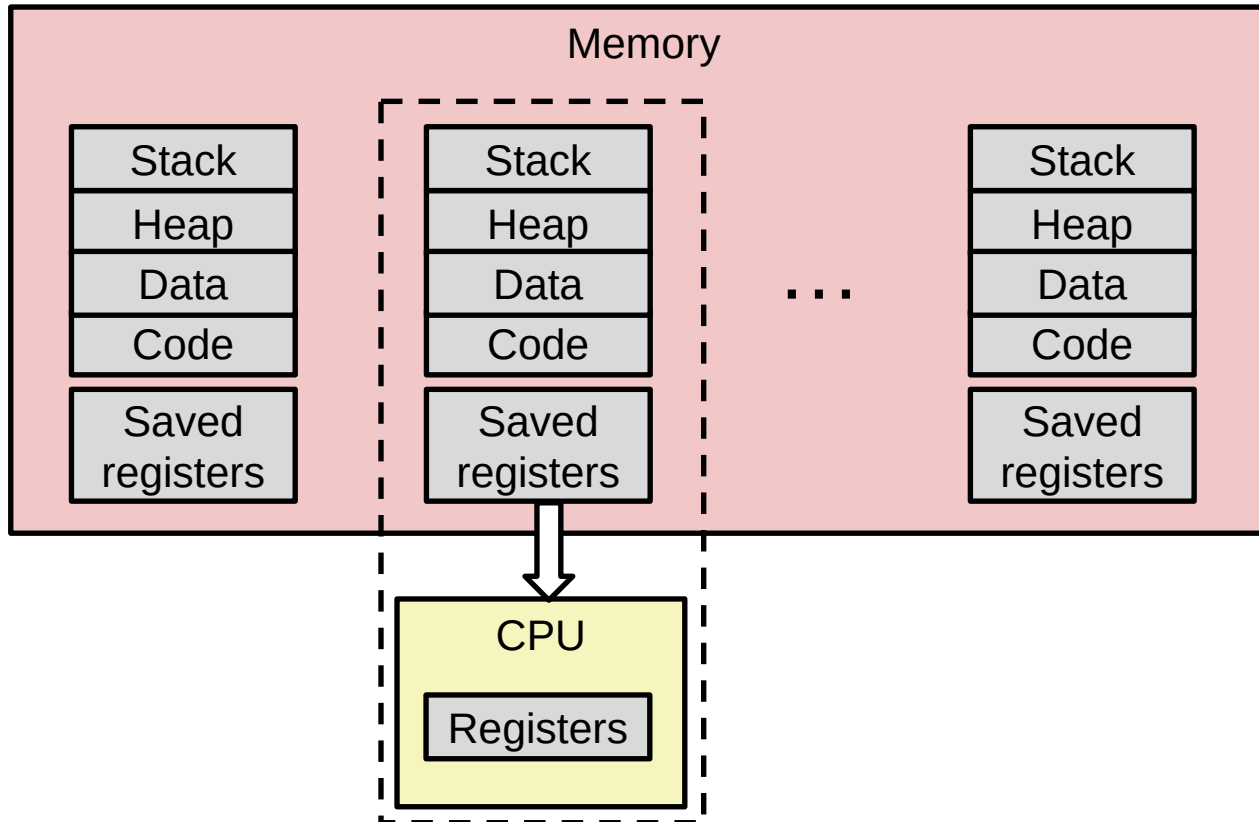
- Grava registradores na memória

A realidade do multiprocessamento



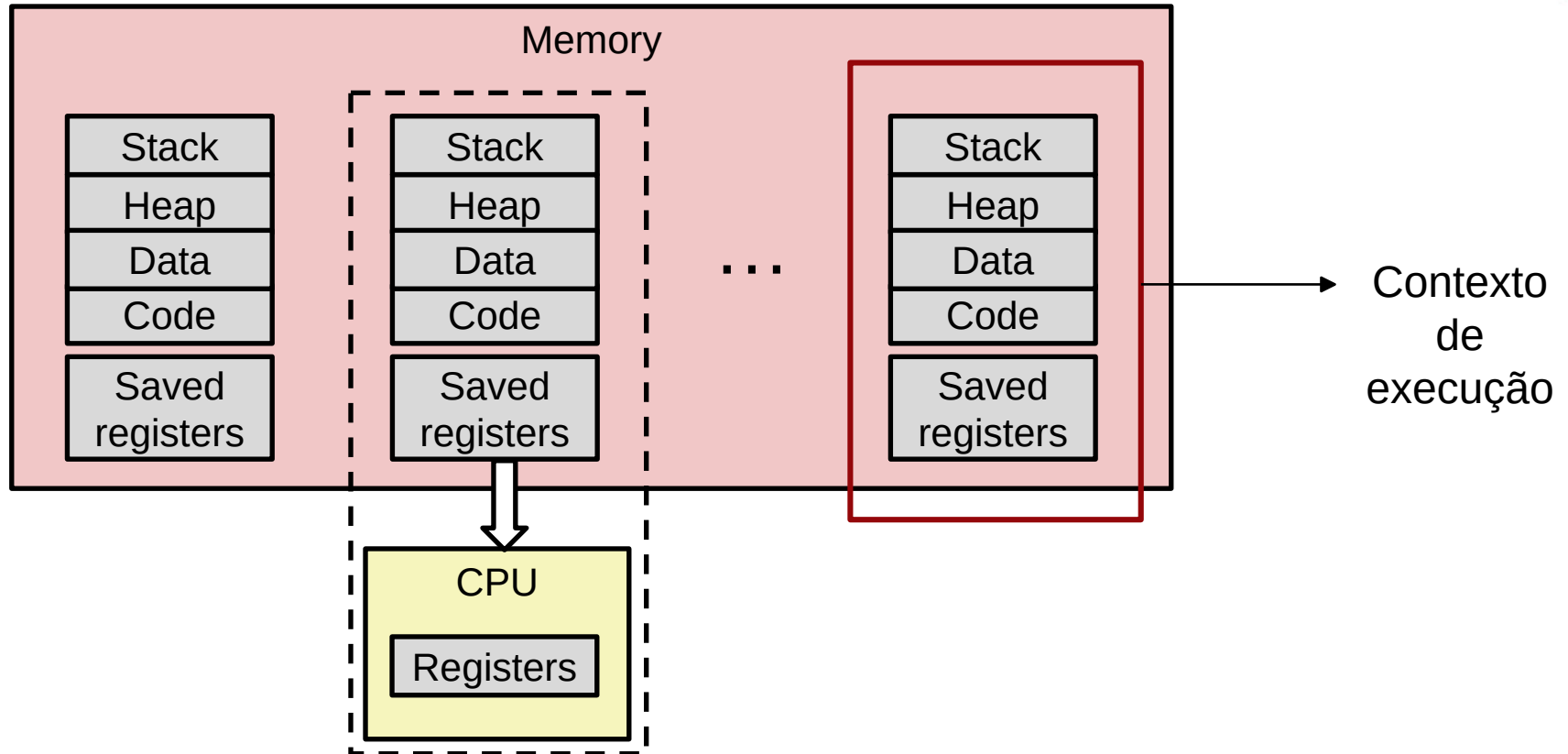
- Escolhe próximo processo a ser executado

A realidade do multiprocessamento



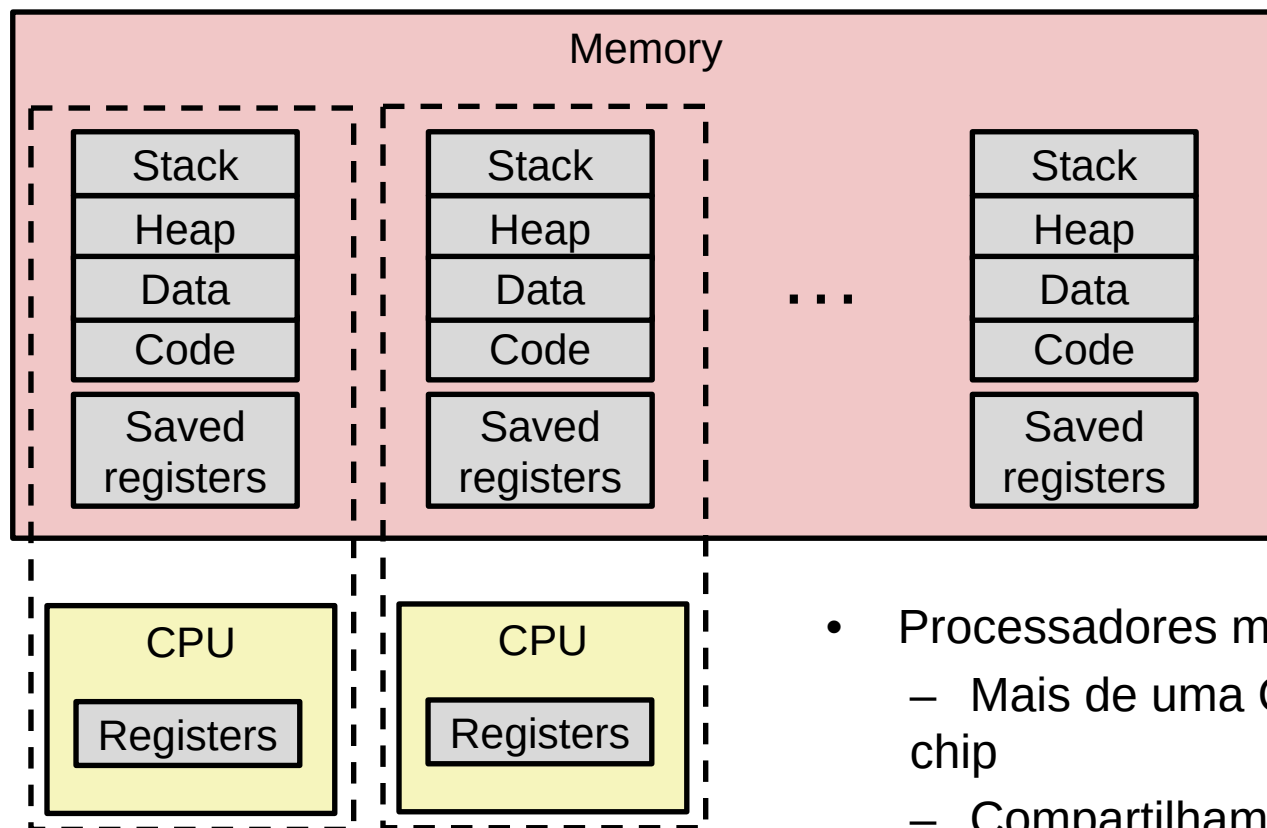
- Carrega registradores gravados e troca de espaço de endereçamento (*context switch* – chaveamento de contexto)

A realidade do multiprocessamento



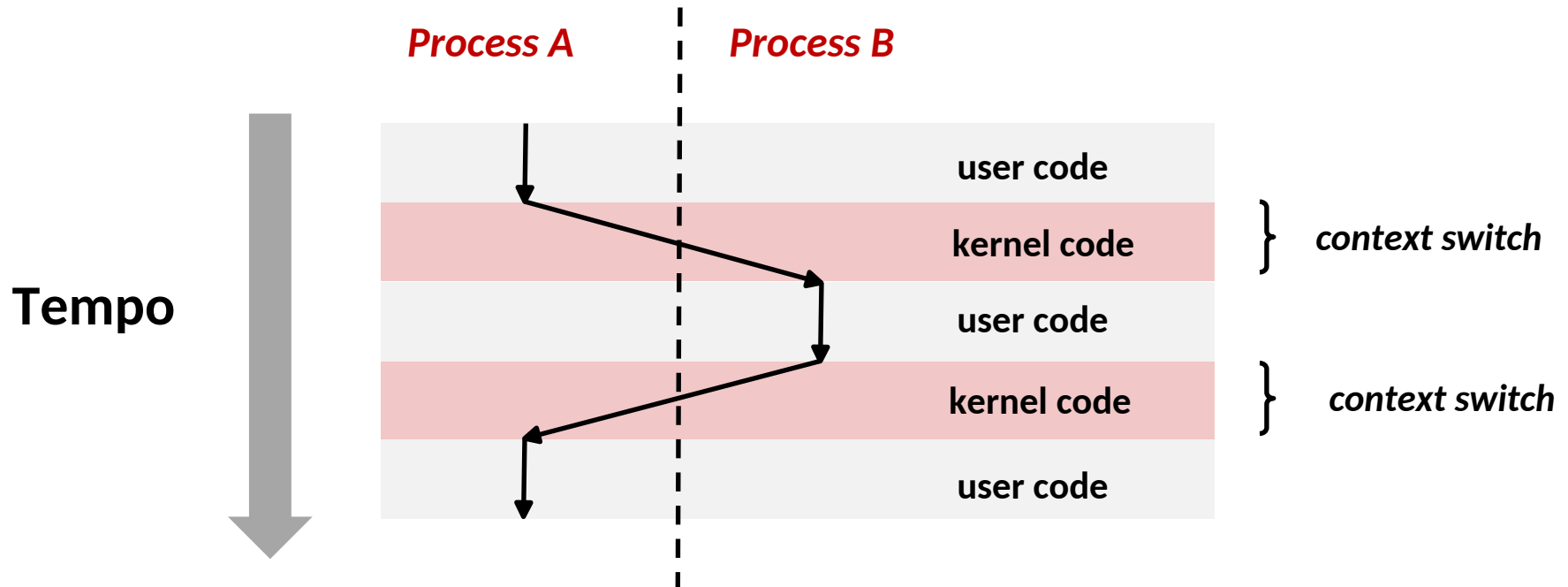
- Carrega registradores gravados e troca de espaço de endereçamento (*context switch* - chaveamento de contexto)

A realidade moderna do multiprocessamento

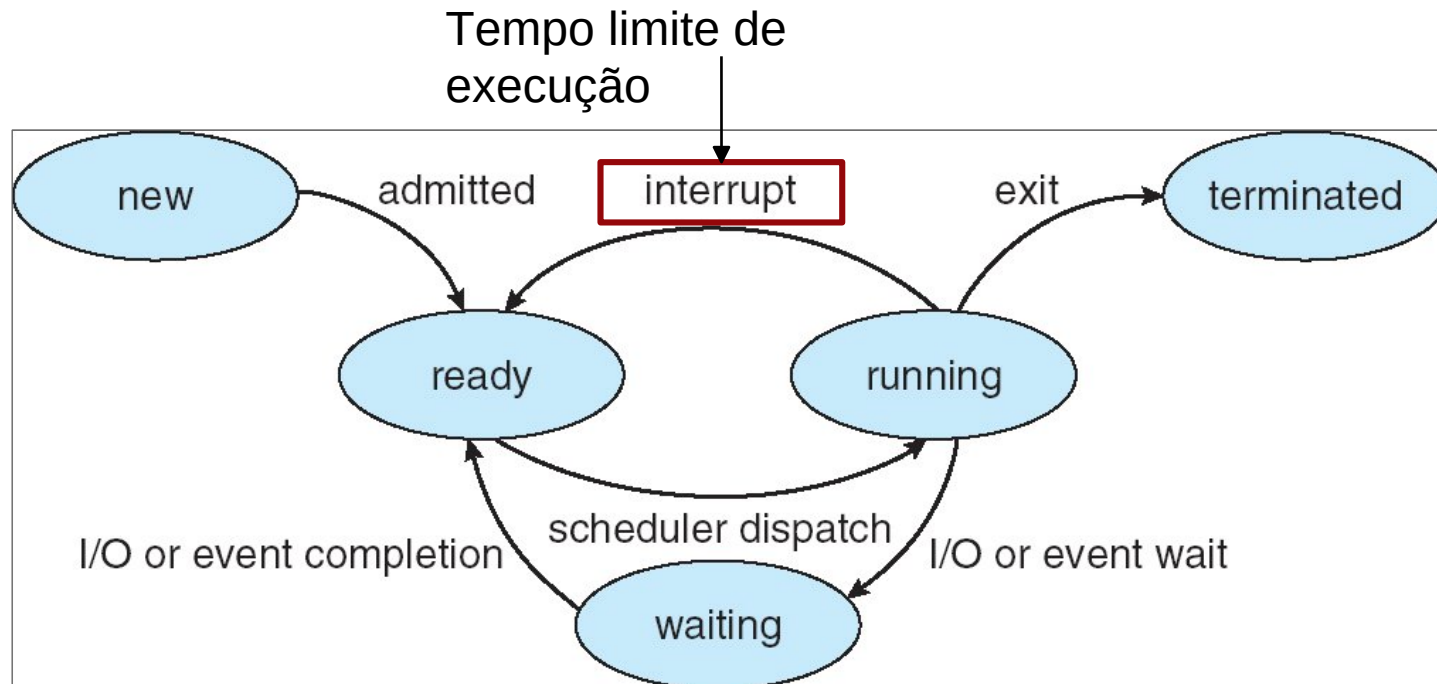


- Processadores multicore
 - Mais de uma CPU em um mesmo chip
 - Compartilham memória principal e parte do cache (cache L3)
 - Cada core pode executar um processo separado
 - Agendamento de processos em cores feito pelo kernel

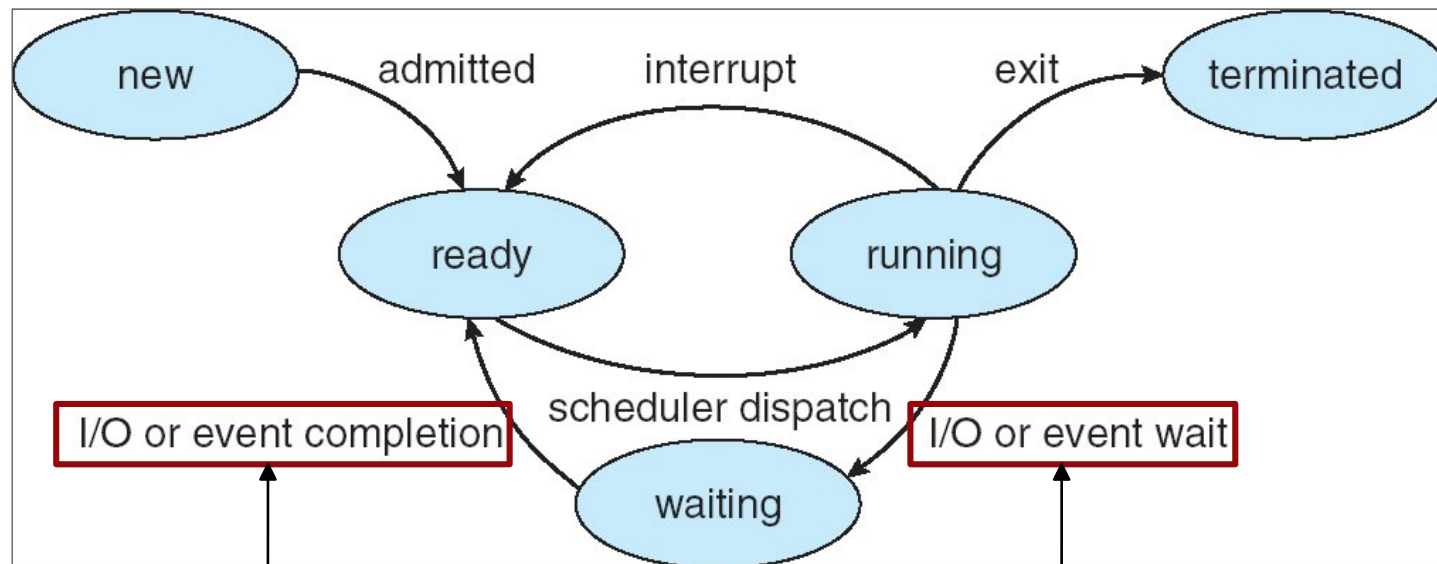
Chaveamento de contexto



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

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)

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>

int main() {
    int rodando = 1;
    pid_t filho;

    filho = fork();

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

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

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