Sistemas Operacionais

Introdução - Estrutura de um SO

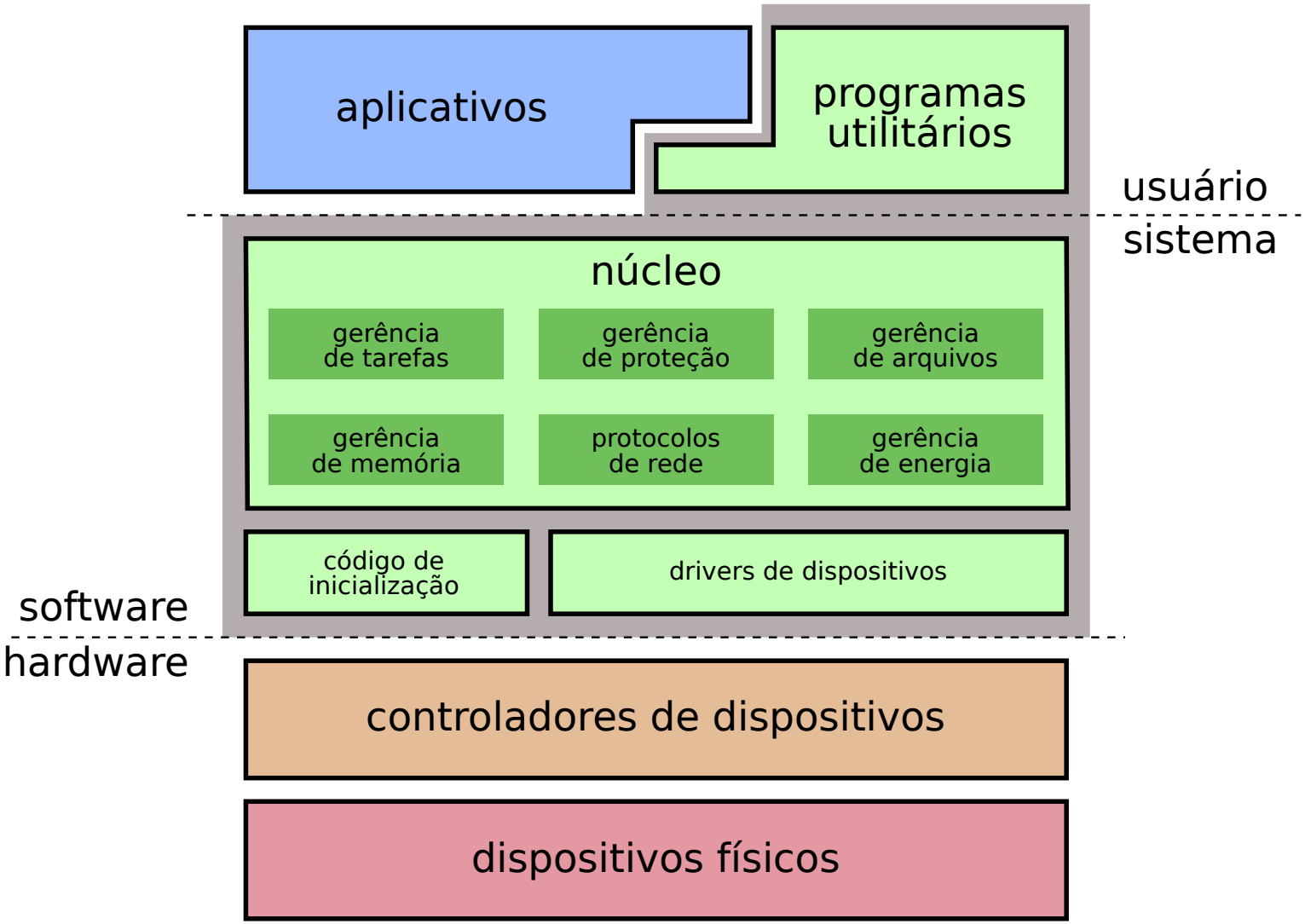
Prof. Carlos Maziero

DInf UFPR, Curitiba PR

Julho de 2020

Conteúdo

Estrutura de um SO



Estrutura de um SO

Componentes mais relevantes:

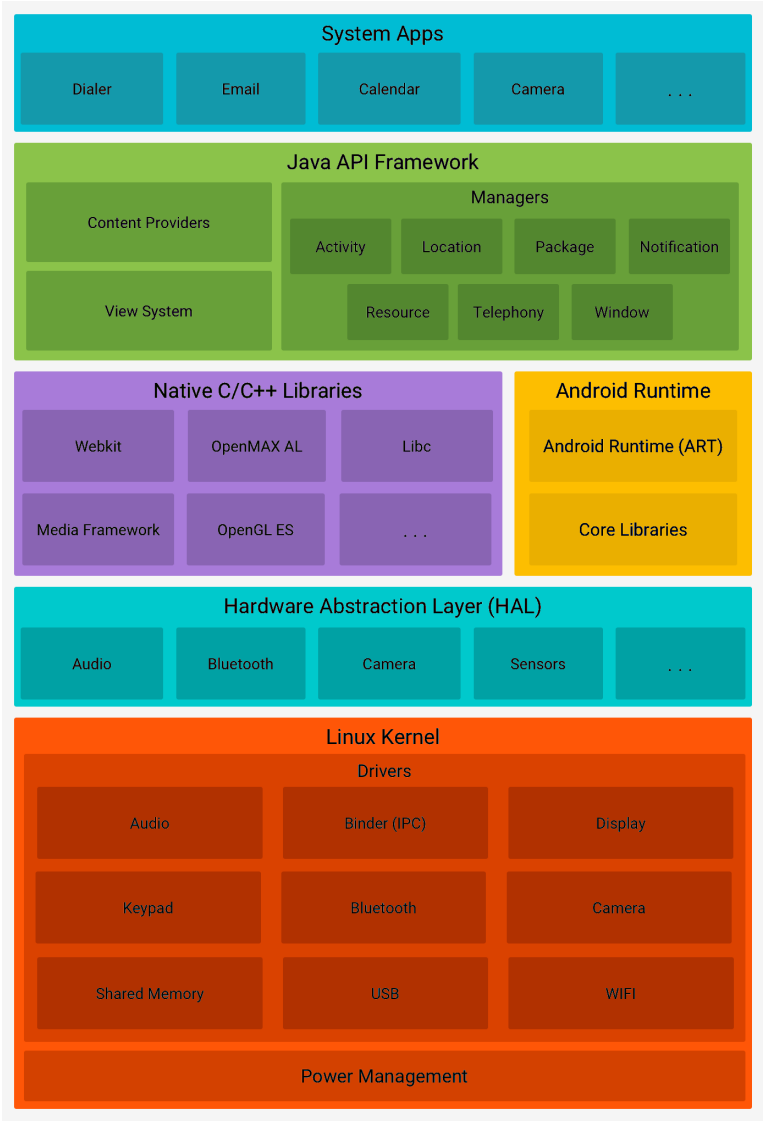
Núcleo : gerência dos recursos do hardware usados pelas aplicações. Também implementa as principais abstrações utilizadas pelos aplicativos.

Inicialização : reconhece os dispositivos instalados e carrega o núcleo do sistema na memória.

Drivers : módulos de código para acessar os dispositivos físicos.

Utilitários : funcionalidades complementares: formatação de discos, *shell* de comandos, interface gráfica, etc.

Exemplo: Android



Estrutura de um SO

Política

Aspecto abstrato de alto nível: decidir a quantidade de memória para cada aplicação, o próximo pacote de rede a enviar, etc.

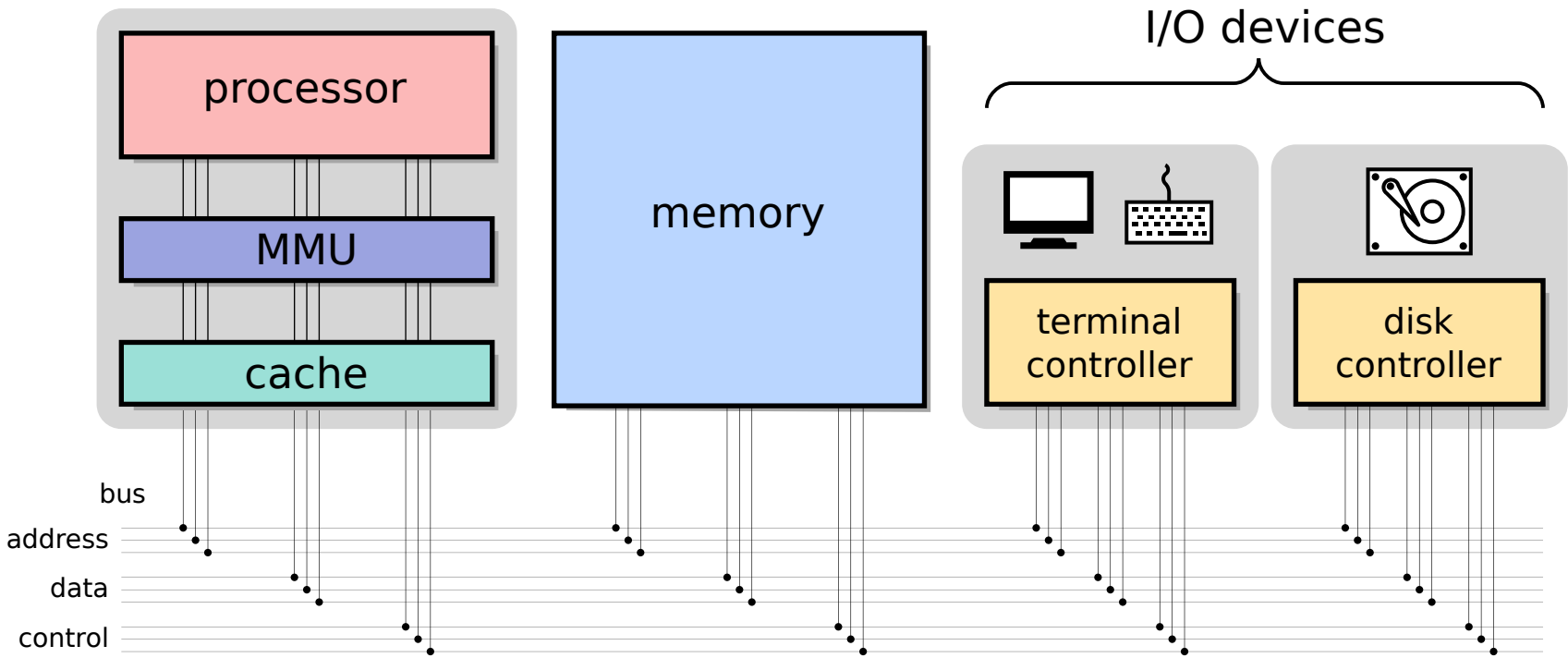
Mecanismo

Procedimento de baixo nível usado para implementar políticas: como iniciar um processo, enviar um pacote de rede, etc.

Filosofia da estrutura: separar políticas de mecanismos

- As políticas devem ser independentes dos mecanismos
- Os mecanismos devem ser genéricos para várias políticas

Arquitetura de um computador



Endereços de dispositivos

dispositivo	**endereços de acesso**
teclado	0060h-006Fh
barramento IDE primário	0170h-0177h
barramento IDE secundário	01F0h-01F7h
porta serial COM1	02F8h-02FFh
porta serial COM2	03F8h-03FFh

Interrupções, exceções e traps

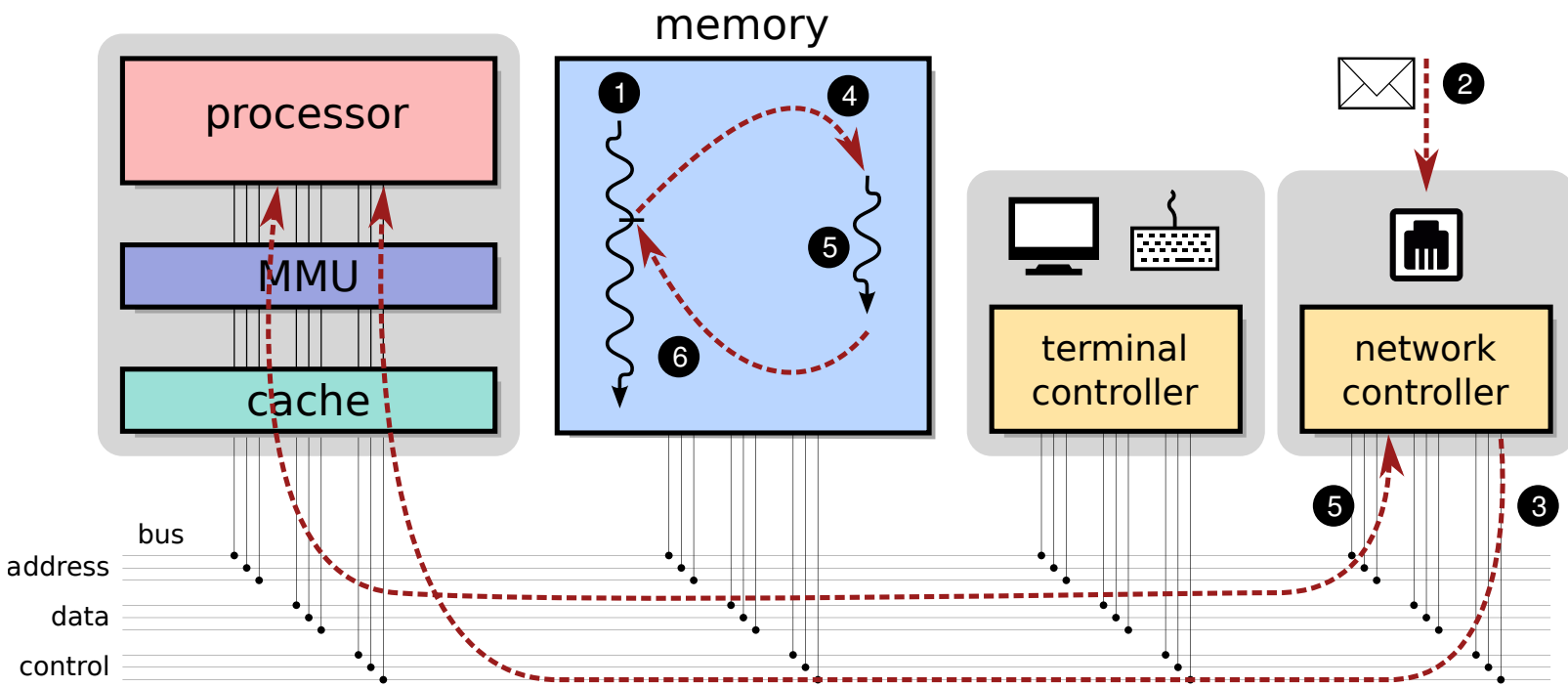
Mecanismos de hardware usados para desviar a execução do processador em caso de eventos:

Interrupção : desvia a execução por evento externo, gerado por um periférico

Exceção : desvia a execução por evento interno (erro numérico, etc)

Trap : desvia a execução a pedido do software

Interrupções e exceções



Roteiro de uma interrupção

1. O processador está em um fluxo de execução;
2. Um pacote vindo da rede é recebido pela placa Ethernet;
3. O controlador Ethernet envia uma IRq ao processador;
4. O processamento é desviado para a rotina de tratamento da interrupção;
5. A rotina de tratamento transfere os dados do pacote do controlador para a memória;
6. A rotina de tratamento finaliza e o processador retorna à execução do programa.

Interrupções típicas PC

IRQ	Descrição
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
...	...
19-31	Intel reserved
32-255	maskable interrupts (devices & exceptions)

Níveis de privilégio

- Implementados pelos processadores modernos
- No Multics: anéis de proteção (0 ... 7)
- Nas CPUs Intel: 4 níveis

Níveis de privilégio

Nível núcleo (*supervisor*, *sistema*, *monitor*):

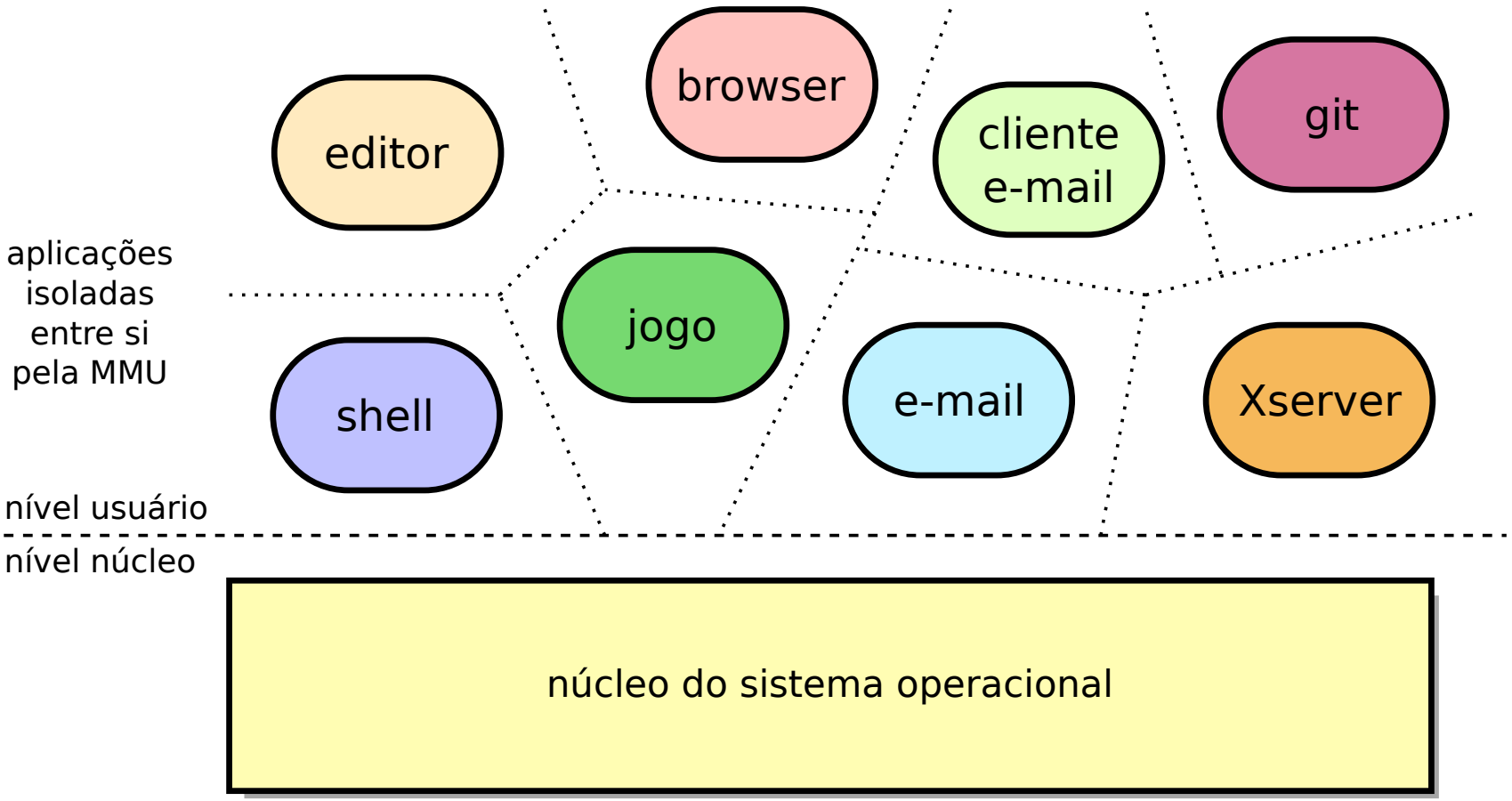
- Acesso amplo aos recursos:
  - todas as instruções do processador
  - todos os registradores
  - todas as portas de entrada/saída
  - todas as áreas da memória
- Nível usado pelo núcleo e os *drivers*.

Níveis de privilégio

Nível usuário (*userspace*):

- Acesso restrito aos recursos:
  - subconjunto das instruções do processador
  - subconjunto de registradores
  - subconjunto de portas de entrada/saída
  - subconjunto de áreas da memória
- Tentativas de acesso inválidas geram exceções
- Nível usado pelos utilitários e aplicações

Separação do núcleo



Chamadas de sistema

São funções que permitem o acesso aos serviços do núcleo:

- Abrir/ler/escrever/fechar arquivos
- Enviar/receber dados através da rede
- Ler teclado
- Escrever dados na tela

Problema:

Como uma aplicação pode invocar uma função do núcleo?

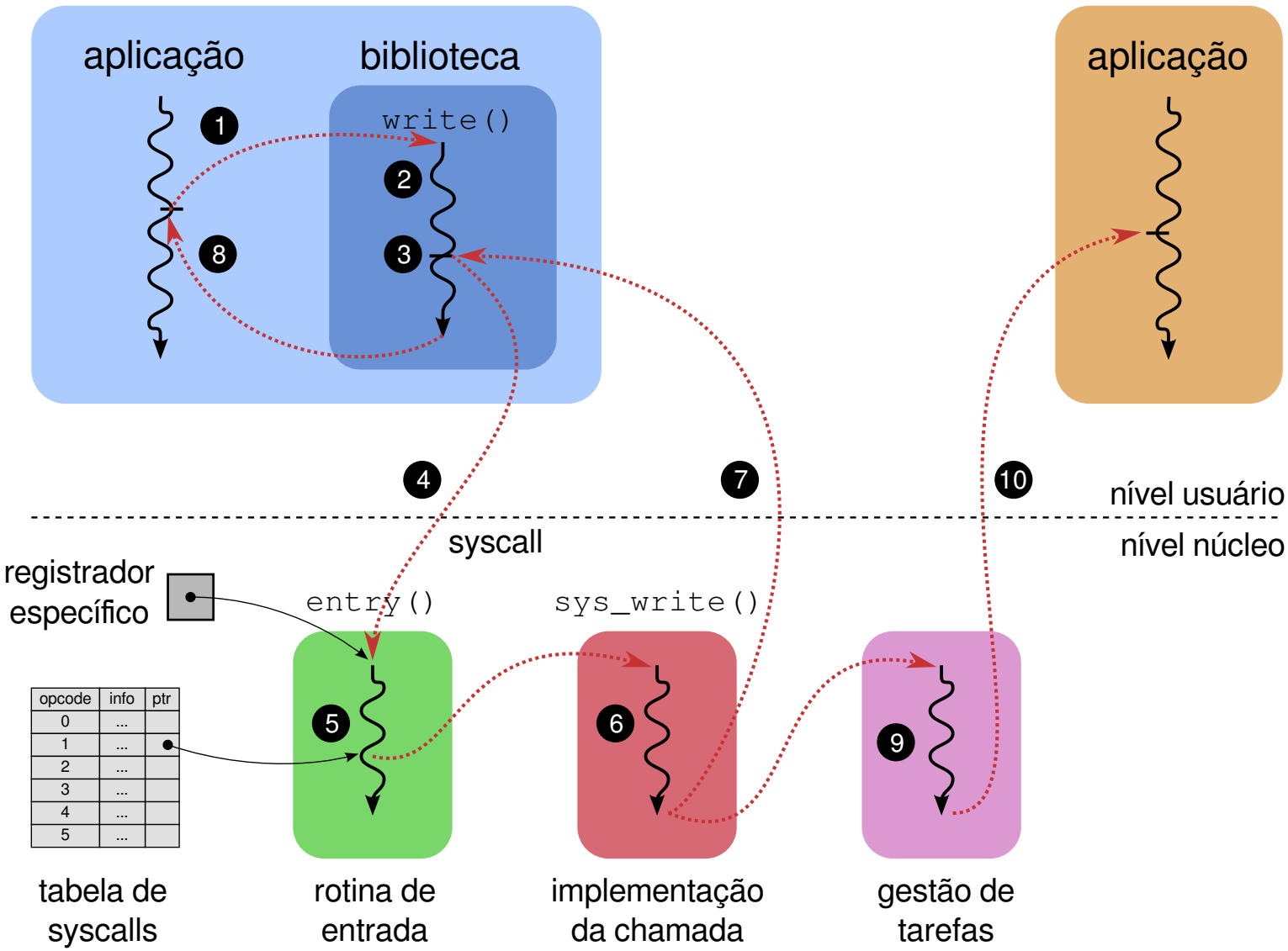
Invocação de uma *syscall* (em C)

```
1  #include <unistd.h>
2
3  int main (int argc, char *argv[])
4  {
5    write (1, "Hello World!\n", 13) ; /* write string to stdout */
6    _exit (0) ;                      /* exit with no error */
7  }
```

Invocação de uma *syscall* (assembly)

```
1  section .data
2  msg db 'Hello World!', 0xA  ; output string (0xA = "\n")
3  len equ 13                  ; string size
4
5  section .text
6
7  global _start
8
9  _start:
10   mov rax, 1                ; syscall opcode (1: write)
11   mov rdi, 1                ; file descriptor (1: stdout)
12   mov rsi, msg              ; data to write
13   mov rdx, len              ; number of bytes to write
14   syscall                   ; make syscall
15
16   mov rax, 60               ; syscall opcode (60: _exit)
17   mov rdi, 0                ; exit status (0: no error)
18   syscall                   ; make syscall
```

Etapas de uma chamada de sistema



Etapas de uma chamada de sistema

1. Aplicação chama `write(...)` da biblioteca.
2. A função `write` preenche os registradores da CPU.
3. A função `write` invoca uma chamada de sistema.
4. A CPU vai para o núcleo e ativa a rotina de entrada (`entry`).
5. A rotina de entrada consulta a tabela de *syscalls* e ativa a função `sys_write`.
6. A função `sys_write` efetua a operação solicitada.
7. A CPU retorna à função `write`, em modo usuário.
8. A função `write` retorna ao código principal da aplicação.

Conjunto de chamadas de sistema

Processos: criar, carregar código, terminar, esperar

Memória: alocar/liberar/modificar áreas de memória

Arquivos: criar, remover, abrir, fechar, ler, escrever

Comunicação: criar/destruir canais, receber/enviar dados

Dispositivos: ler/mudar configurações, ler/escrever dados

Sistema: ler/mudar data e hora, desligar o sistema

Cada SO define seu próprio conjunto de *syscalls*:

- Sistemas Windows: *Win32*
- Sistemas UNIX: *POSIX*