Programação de Sistemas

Arquitectura dos Sistemas Operativos

Introdução (1)

- Um sistema operativo de uso geral é formado por diversas componentes:
  - Gestor de processos (PS-process scheduler): multiplexa o processador entre entidades lógicas designadas por processos (que serão tema central do próximo capítulo).
  - Gestor de memória (MM-memory manager) : controla acesso dos processos à memória central RAM.
  - Comunicação entre processos (IPC-interprocess communication).
  - Entradas/saídas (I/O-input/output) *: comunicação com dispositivos do computador (disco,…) e exteriores ao computador.
  - Interpretador de comandos: aplicação de interface para utilizadores (dado no 1º laboratório).

* Descrito de forma muito resumida nesta disciplina

Introdução (2)

- Gestor de ficheiros (FS-file system): organiza logicamente os dispositivos de memória de massa (disco,…).
- Funções de sistema (SCI): interface de serviços para as aplicações informáticas.

[Def] Núcleo ("kernel") do SO: conjunto dos componentes essenciais (ex: PM, MM, IPC e FS) que residem na memória central.

[Def] Espaço do núcleo (kernel space): zona de memória onde o núcleo executa e disponibiliza serviços.

[Def] Espaço de utilizador (user space): zona de memória onde os processos de utilizador correm.

Introdução (3)

[Def] Serviço: trabalho descrito de forma genérica, a executar pelo SO a pedido de uma aplicação ou de outro serviço.
Exemplo: impressão de um documento numa impressora.

[Def] Utilitário: trabalho separado do SO, normalmente usado para apoiar gestores e utilizadores.
No Linux os utilitários são categorizados em

- Gestão do sistema de ficheiros (`read`,…)
- Comunicações locais e de rede
- Editores
- Filtros e processadores de texto
- Compiladores (`gcc`,…)

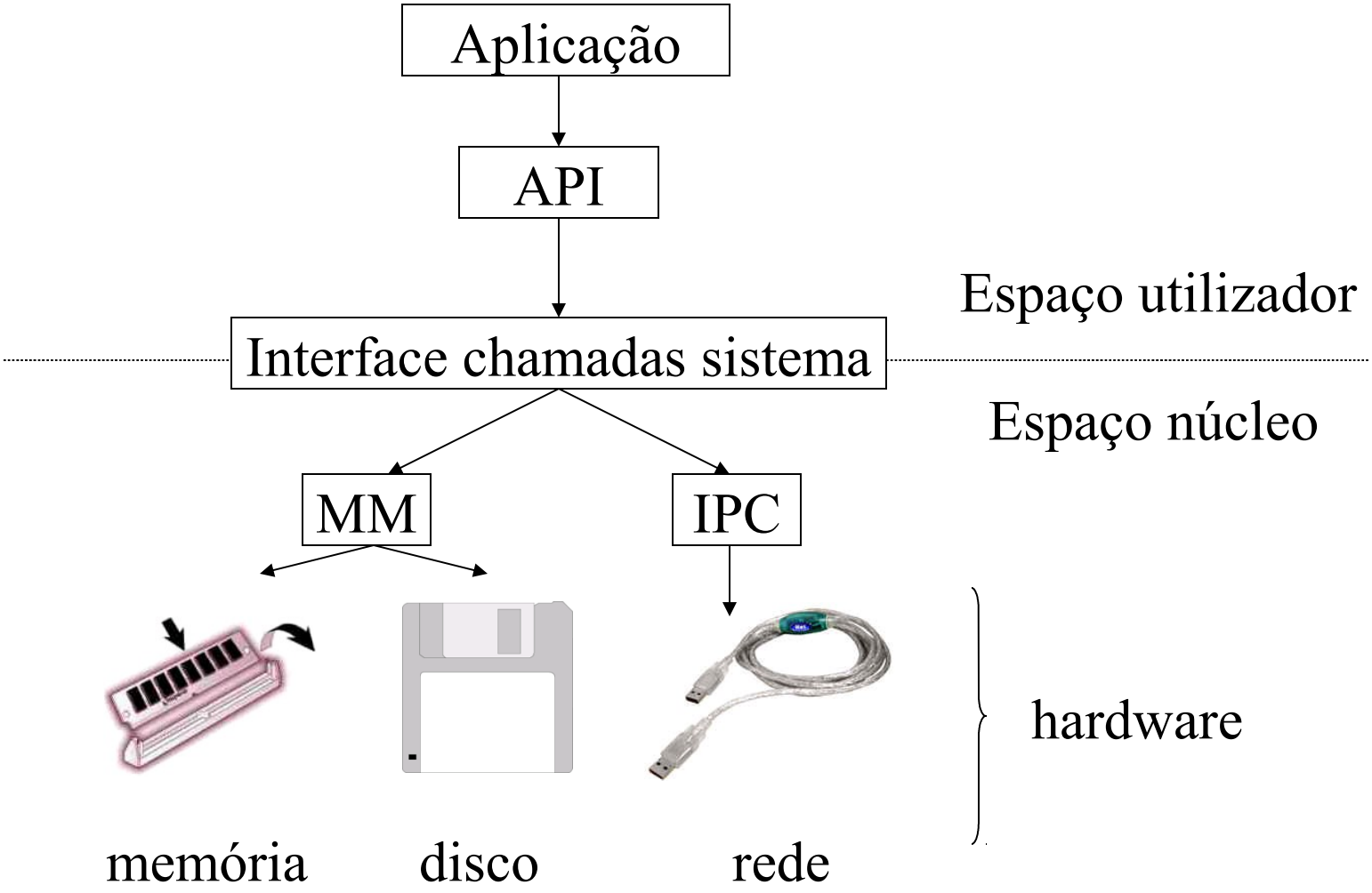
Introdução (4)

[Def] Função de sistema: tarefa que implementa uma das partes constituintes dos serviços ou utilitários do SO.
As funções manipulam estruturas de dados internas ao sistema operativo.
Exemplo: criação de um processo, que é representado internamente por uma tabela armazenada em listas.

[Def] API (Application Programming Interface): conjunto de funções do sistema.

- norma POSIX (Portable Operating System Interface) do IEEE para o Unix.
- Windows API.

Introdução (5)

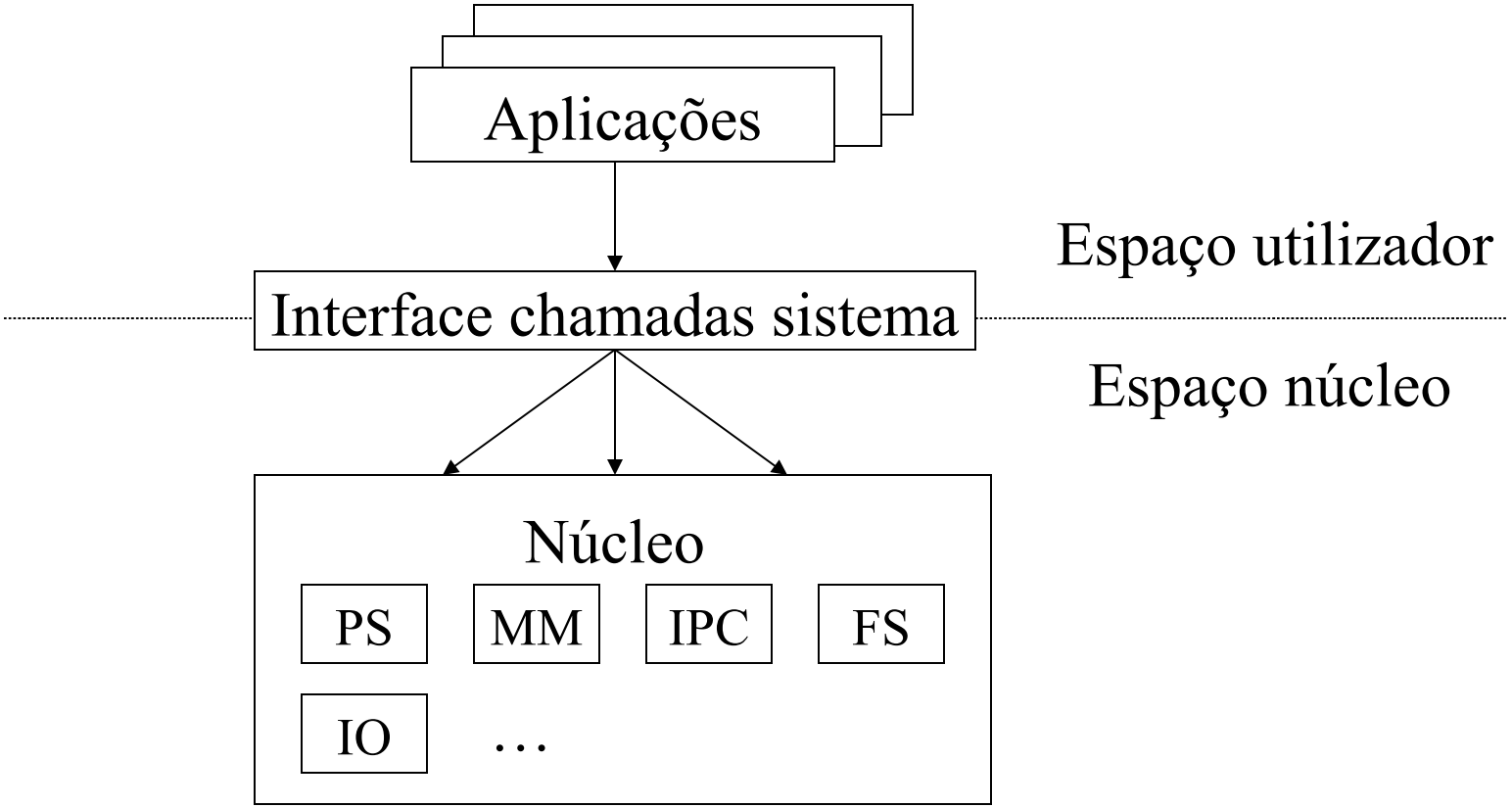


Arquitectura (1)

- Os diversos componentes de um SO podem ser organizados de acordo com a arquitectura adoptada.

A. <u>Monolítico</u>
SO formado por um único módulo (idealmente desenvolvido por abstracção de dados-ADT). O SO reside numa única zona de memória.
Exemplos: MSDOS, Unix inicial
Vantagens: fácil de implementar, bom desempenho.
Inconvenientes: difícil de estender e alterar.

Arquitectura (2)



Arquitectura (3)

B. Camadas ("layered")
Componentes distribuídos por diversas camadas, cada uma agrupando componentes com funcionalidades semelhantes.

- A camada inferior é constituída por hardware e a camada de topo pelas aplicações.
- Componentes de uma camada interagem apenas com as camadas de nível imediatamente inferior.

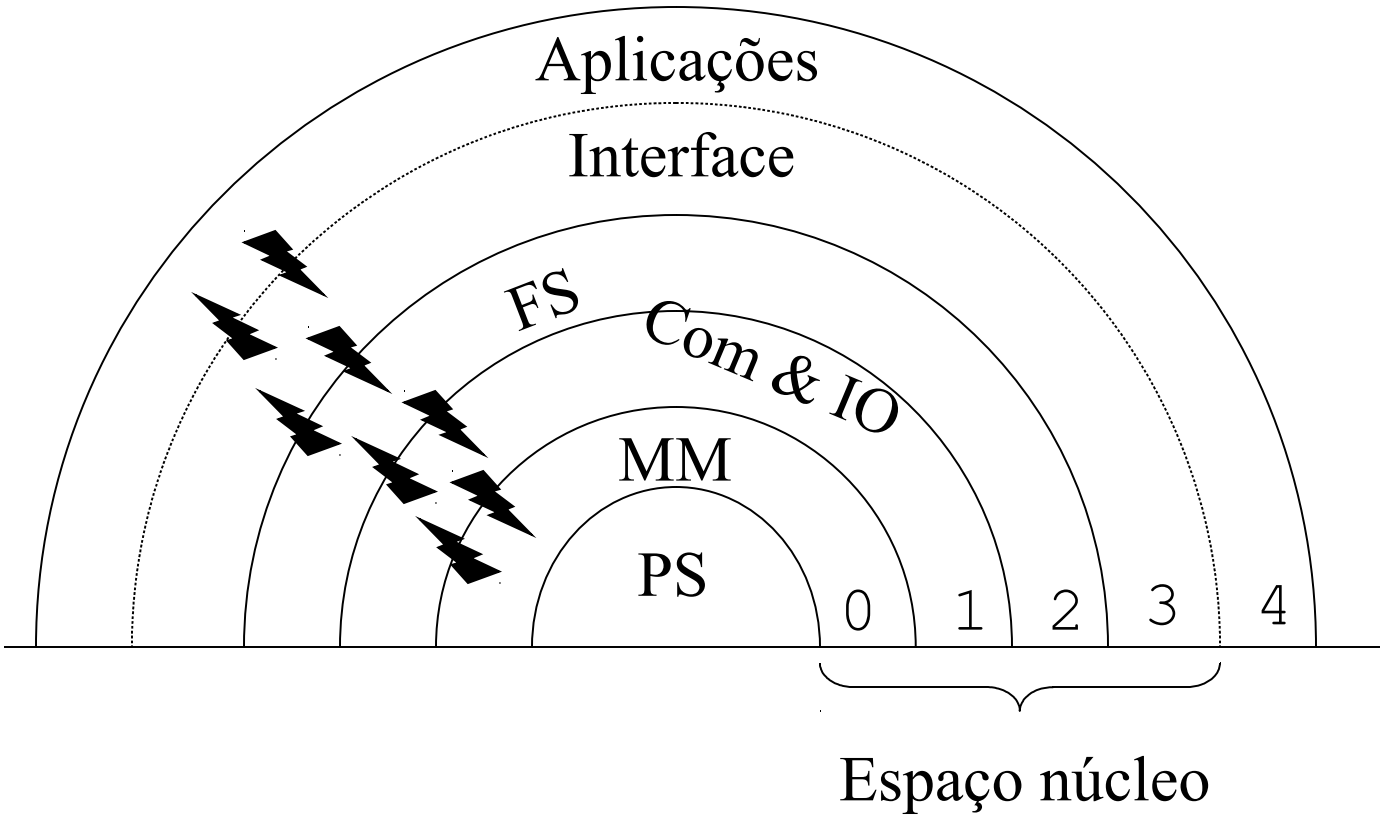
Exemplos: THE (1º SO por camadas), Multics

Vantagens: bem estruturado, fácil de alterar.

Inconvenientes: acesso lento a serviços disponibilizados em camadas distantes.

Arquitectura (4)

Níveis de um SO



Arquitectura (5)

C. Micronúcleo

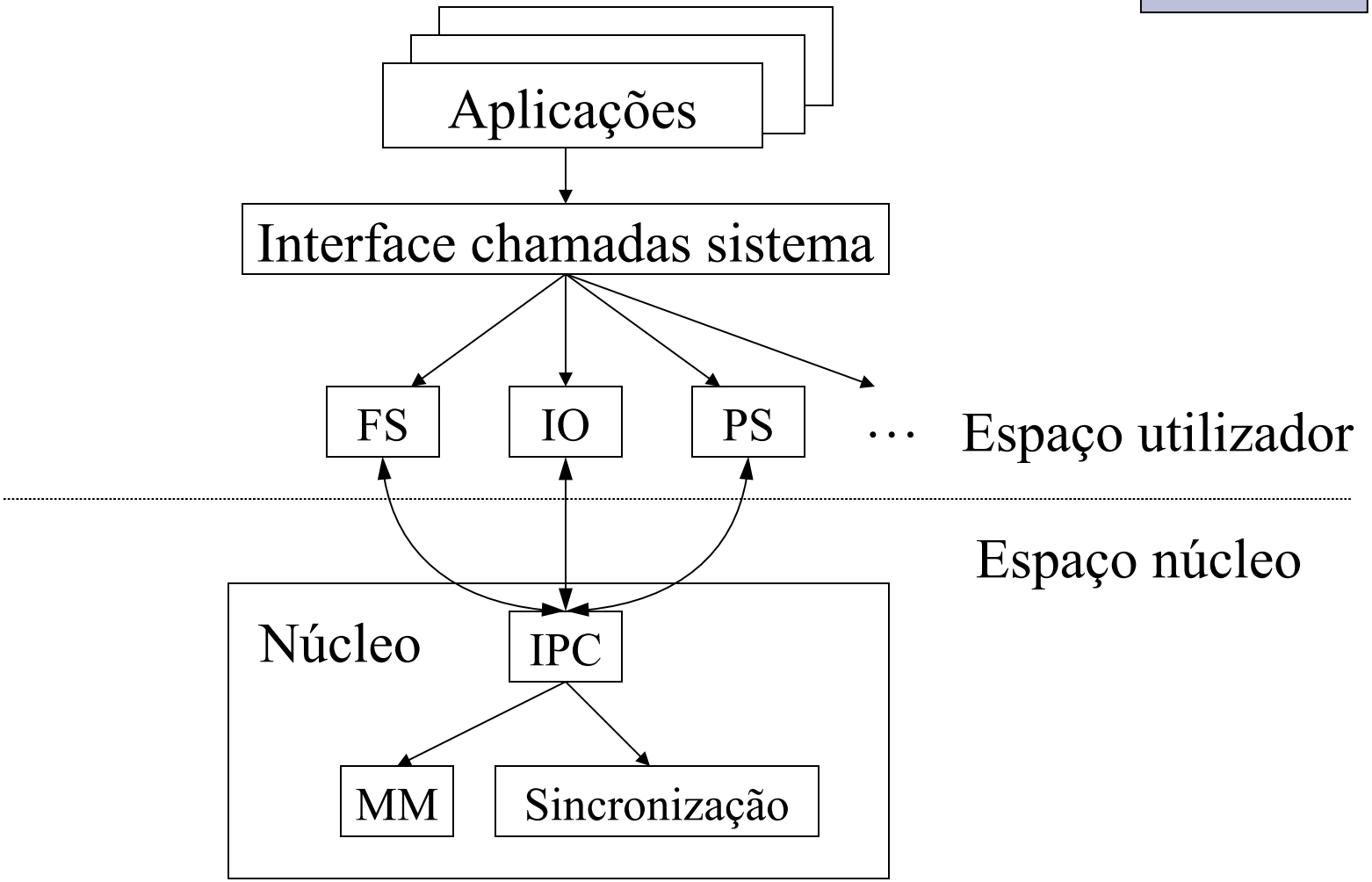
Caso especial do SO por camadas, com apenas 2 (a debaixo com serviços essenciais do núcleo, de cima com restantes serviços).

Exemplos: Windows NT, Mach

Vantagens: erros nos serviços a nível de utilizador não indisponibilizam o SO.

Inconvenientes: núcleo não corre sozinho, aumento da intecomunicação entre processos.

Arquitectura (6)



Arquitectura (5)

C. Distribuída

Cada component/serviço do SO é um processo separado, existente:

- Na mesma máquina
- Em máquinas diferentes

Interação é por canis/mensagens

Sistema de ficheiros distribuíos

Gestão de memória distribuída

Gestão de processos distribuída

Exemplo Amoeba

Arquitectura (7)

D. Máquina virtual

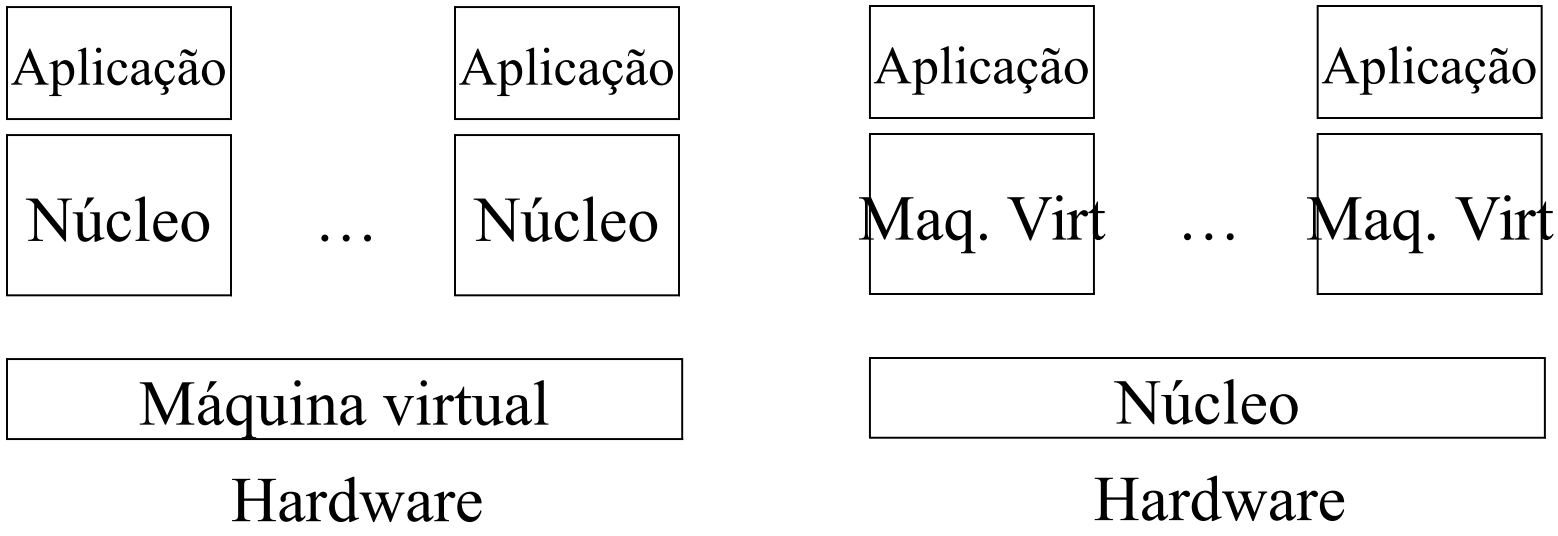
Caso especial do SO por camadas, na parte inferior um SO básico, por cima um *virtualizador* e no topo vários SOs.
Sistema: XEN / VirtualBox

Processo: java, python, CLR

Vantagens: permite o mesmo computador correr vários SOs em simultâneo, facilita portabilidade de aplicações.

Inconvenientes: baixo desempenho. (+-)

Arquitectura (8)



Interrupções (1)

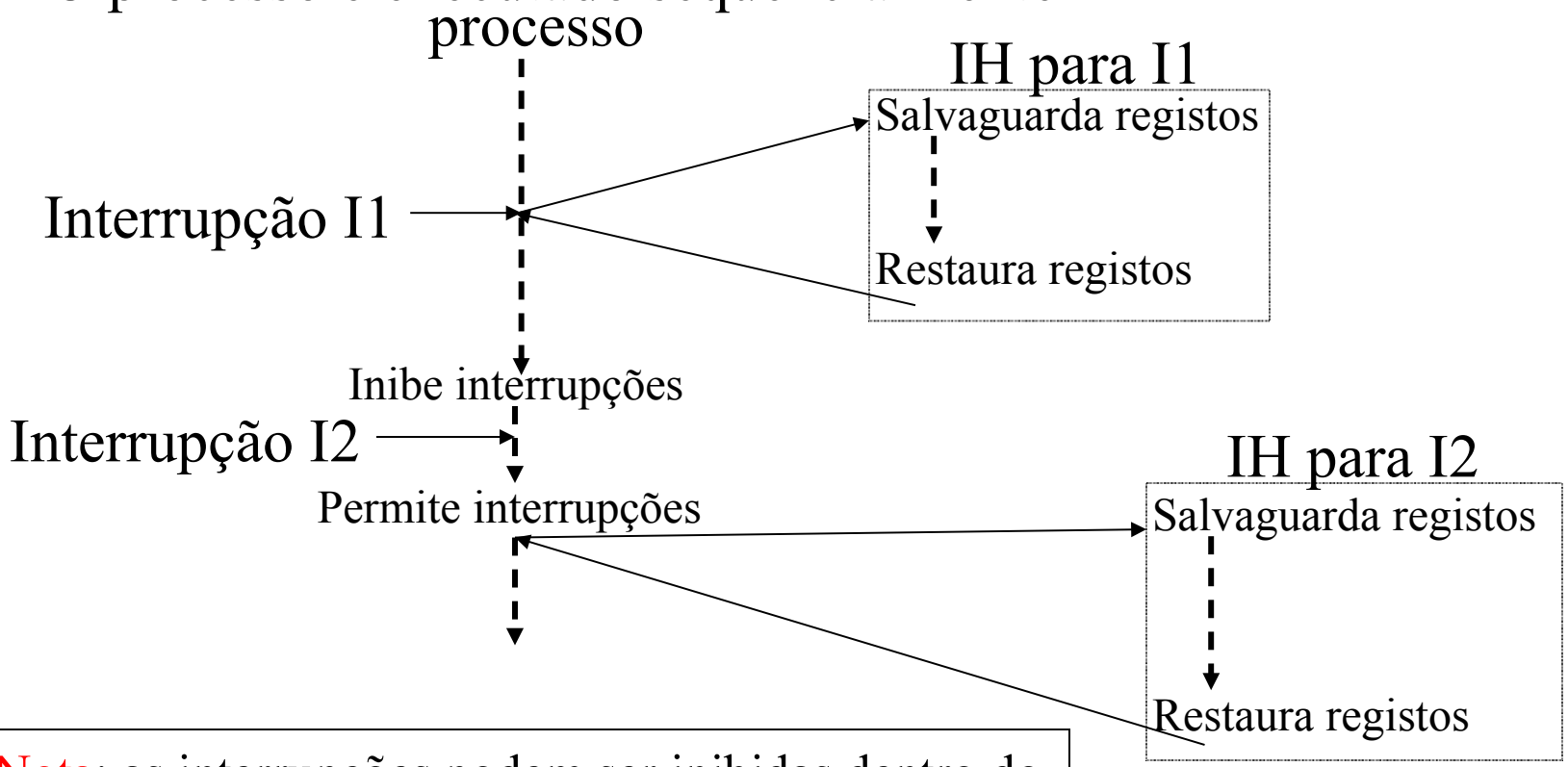
- Um sistema operativo é guiado por interrupções ("interrupt-driven"), de vários tipos
  - Dispositivos entrada/saída, para indicar que os dados foram transferidos.
  - Temporizadores, para indicar que terminou a fatia de tempo concedida a um processo.
  - Hardware, por exemplo tentativa de acesso a memória (inexistente ou não satisfazendo condições de acesso).
  - Programas (ex: divisão por zero, mudança de modo).

Interrupções (2)

- Durante alguns períodos, o sistema operativo pode inibir ("disable") que interrupções interrompam a execução das instruções.
- Na maior parte das vezes, há permissão ("enable") para as interrupções transfirem o CPU para uma área de tratamento da interrupção ("interrupt hander").
- O tratamento da interrupção deve iniciar pela salvaguarda dos registos do processo (acumulador, ponteiro para a pilha,…). Mais tarde, os registos carregados com esses valores, para que seja possível retornar a execução do programa como se não tivesse havido interrupção (o único efeito é demorar um pouco mais a terminar o programa).

Interrupções (3)

- O processo é executado sequencialmente



Nota: as interrupções podem ser inibidas dentro do IH, ou autorizadas apenas as de prioridade superior

Chamadas de sistema (1)

- Uma aplicação interage com o SO através de chamadas de sistema ("system call").
- Algumas chamadas do sistema podem ser introduzidas pelo utilizador. Por exemplo, a instrução em C `count=read(fd,buffer,nbytes)` é equivalente ao comando do interpretador de comandos Shell (palavras colocadas numa tabela ou em várias variáveis)

```
read [-u fd] [-n nbytes] [-a aname] [nome1] ...
```

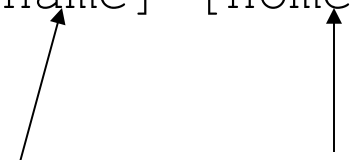


Tabela onde são colocadas as palavras

1ª palavra

Chamadas de sistema (2)

- No Linux, as chamadas ao sistema são agrupadas em 5 categorias:
  - Controlo do processo: `fork`, `execute`, `wait`,...
  - Manipulação de ficheiros: `open`, `read`, `set`,...
  - Manipulação de dispositivos: `request`, `read`, ...
  - Informação de manutenção: `date`, `ps` , ...
  - Comunicações: `send`, ...
- Unix iniciais definiam 60 chamadas ao sistema
- Linux (ver códigos em `/usr/include/asm/unistd.h`):
  - Kernel 2.0 ≈ 160
  - Kernel 2.2 ≈ 190
  - Kernel 2.4 ≈ 220
  - Kernel 2.6 define 286

Chamadas de sistema (3)

Execução da função `read` do C

- {1-3} Carrega na pilha os parâmetros da função, da direita para a esquerda (nota: funções com número variável de parâmetros, ex: `printf`, inserem à esquerda os parâmetros fixos).
- {4} Chama `nread = fread(inputbuf, OBJSIZE,numerobjs,fileptr)`
  - definida na biblioteca `/usr/include/stdio.h`
  - código residente no arquivo `/usr/lib/libc.a`
- {5} Inicializa registos (EAX no Pentium, valor indicado no `unistd.h` para __NR_read é 3)

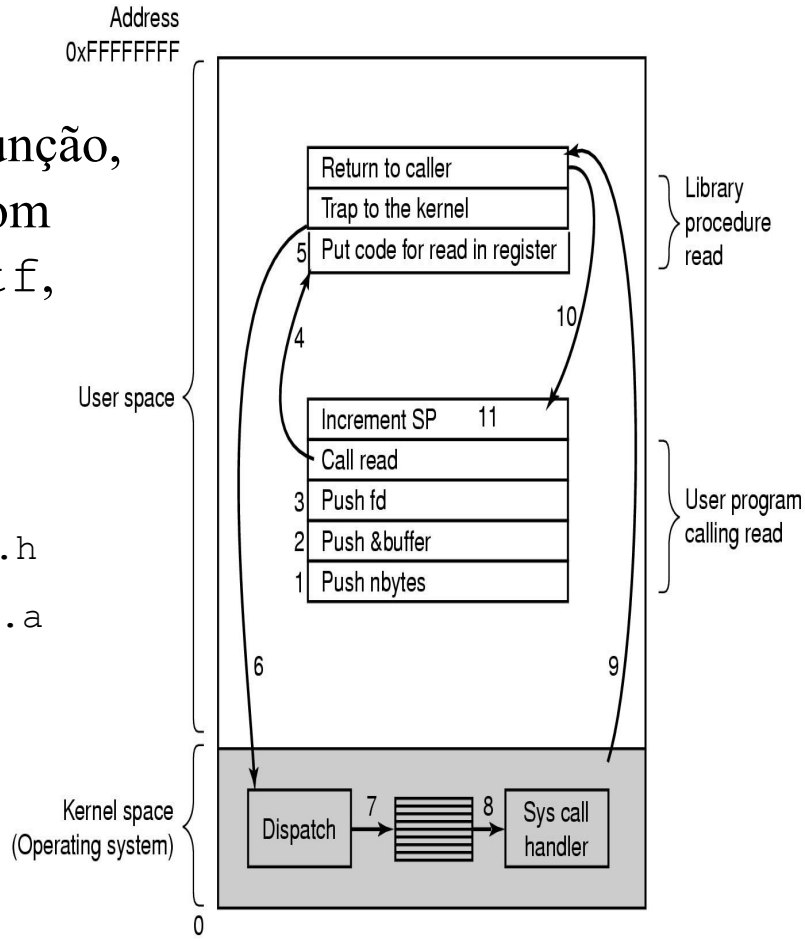


Figura 1-17, Modern Operating Systems

Chamadas de sistema (4)

- {6} Executa um TRAP (Pentium: instrução `INT 80h`, ARM: instrução `swi`), que
  - salva contexto corrente (registos)
  - passa de modo U para modo S
  - transfere PC-"program counter" para localização específica do núcleo.
- {7-8} De acordo com valores de {5}, o despacho invoca o manipulador de dispositivo ("device handler") correspondente, com prefixo `sys_` (neste caso,
  `asmlinkage int sys_read(unsigned fd,char *buf,int count)`
- {9} O manipulador de dispositivo entra em bloqueio até conclusão da transferência de dados.
  Quando a transferência for concluída é executada a instrução `iret`, de retorno para a função de biblioteca.

Chamadas de sistema (5)

- {10} Retorna à instrução `read` do C

Chamadas a funções de sistema retornam inteiro indicando o resultado

- 0 determina execução com sucesso
- Código da causa do erro afixada na variável de ambiente `errno`
- Nome lógido para códigos de erro lista no ficheiro `/usr/include/asm-generic/errno-base.h`

  Ex: `#define EACCES 13 /* Permission denied */`
- Mensagem explicativa enviada para `stderr` pela função biblioteca (parâmetro acrescentado à mensagem)

```
#include <stdio.h>
void perror(const char *string);
```

Chamadas de sistema (6)

- Comando `strace [opções] prog` imprime no terminal todas as chamadas de sistema durante a execução de `prog`.
  - `-f` acompanha cada processo lançado, indicando no início da linha o PID do processo que faz a chamada de sistema.
  - `-t` imprime, no final da linha a duração de cada chamada do sistema na forma <0.000104>.

```
asterix.ist.utl.pt> strace -f -T testaWait
execve("./testaWait", ["testaWait"], [/* 48 vars */]) = 0
uname({sys="Linux", node="asterix.ist.utl.pt", ...}) = 0 <0.000022>
…
[pid  4145] getpid()                    = 4145 <0.000017>
…
[pid  4146] getppid()                   = 4145 <0.000013>
[pid  4146] exit_group(20)              = ?
Process 4145 resumed
```

Tabelas de controlo do SO

- Normalmente, o SO controla 4 componentes base:
  - Processos
  - Memória
  - Dispositivos
  - Ficheiros

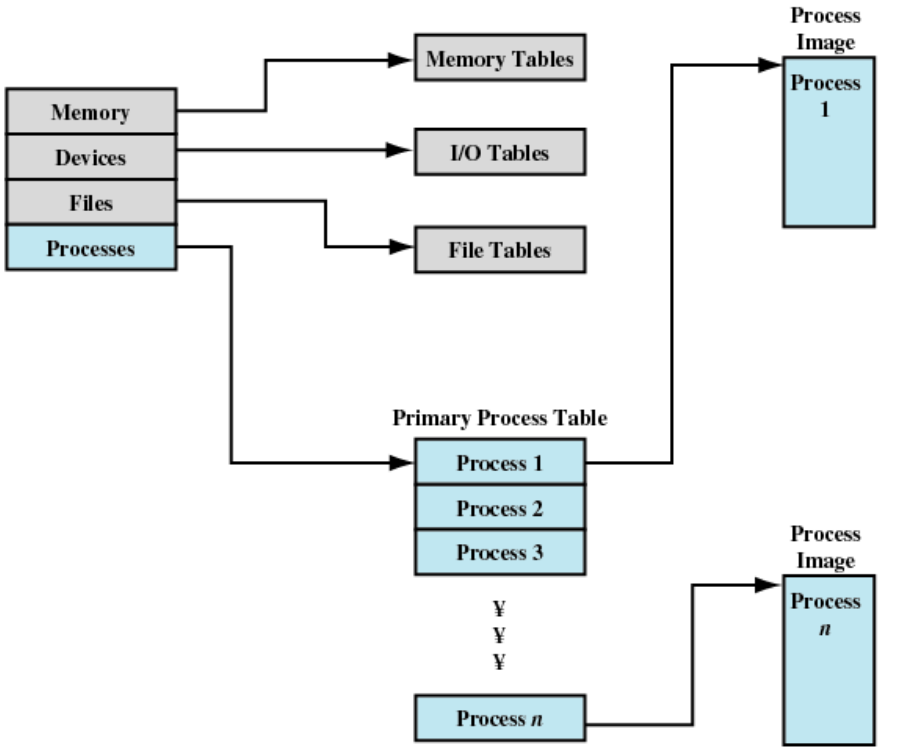


Figure 3.11 General Structure of Operating System Control Tables

Manual

- O comando `man` disponibiliza informação sobre o Linux.
- As páginas estão organizadas por 9 secções:
  1. Comandos de utilizador
  2. Chamadas de sistema
  3. Funções de biblioteca do C
  4. Controladores de dispositivos
  5. Ficheiros de configuração
  8. Comandos de manutenção
- O `man` procura a página a partir da secção 1. O utilizador pode indicar a secção entre o comando e o tópico.

  `man 1 read` : página do comando de utilizador `read`

  `man 2 read` : página da chamada de sistema `read`

  `man 3 fread` : página da função de biblioteca `fread`