

Sistemas Operativos

Licenciatura Engenharia Informática
Licenciatura Engenharia Computacional

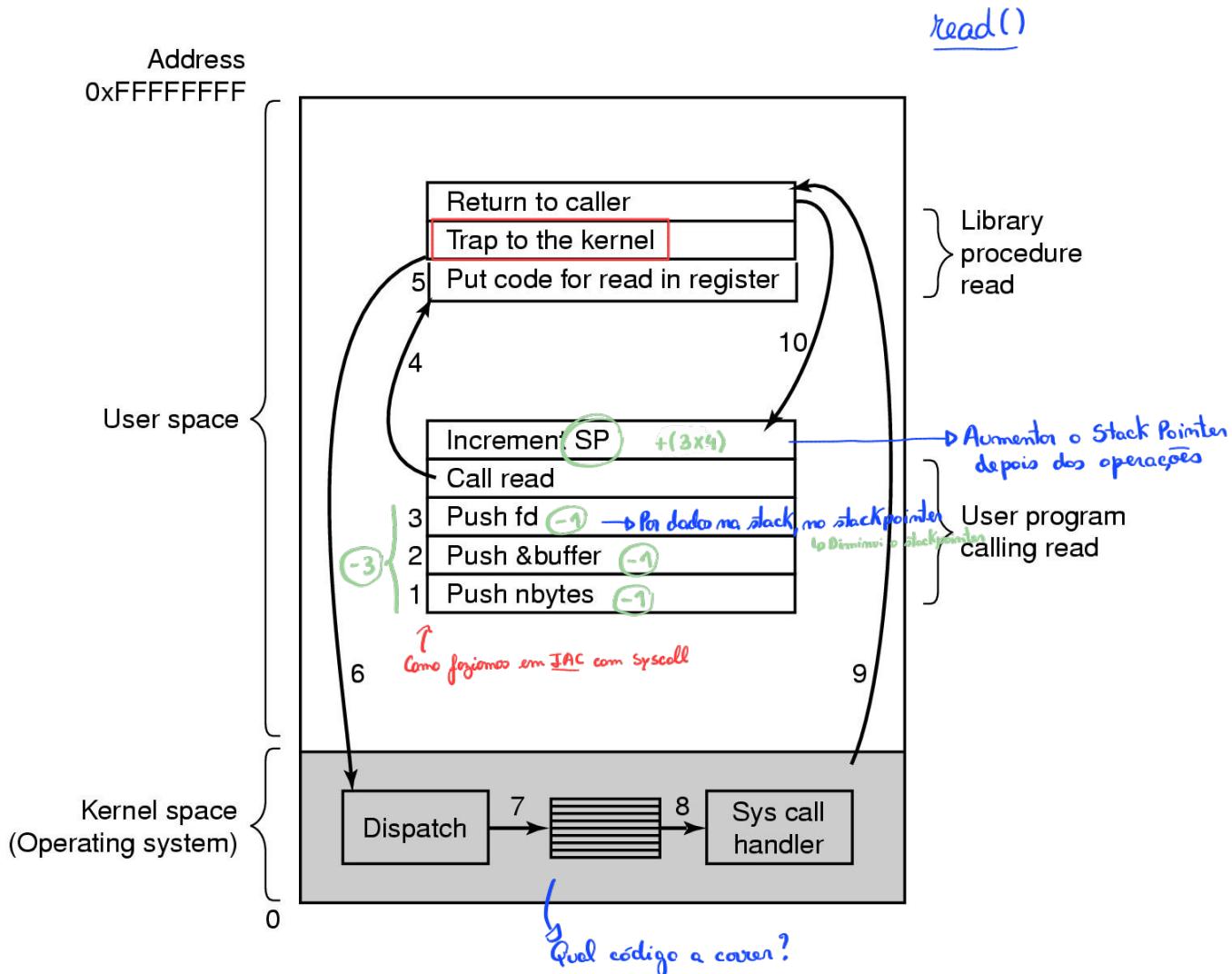
Ano letivo 2023/2024

Nuno Lau (nunolau@ua.pt)

Modos de operação

- De modo a garantir a segurança do sistema, a maioria dos SOs podem executar em 2 modos:
 - Modo de utilizador
 - Com restrições de segurança
 - Acesso a certas instruções e zonas de memória e dispositivos estão interditos
 - Modo de *kernel*
 - Sem restrições de segurança
 - Pode executar todas as instruções e acessos
 - Instruções privilegiadas
 - Chamadas ao sistemas providenciam uma forma segura de alternar entre os 2 modos

Chamadas ao Sistema



Chamadas ao Sistema

UNIX	Win32	Description
fork	CreateProcess	Create a new process
waitpid	WaitForSingleObject	Can wait for a process to exit
execve	(none)	CreateProcess = fork + execve
exit	ExitProcess	Terminate execution
open	CreateFile	Create a file or open an existing file
close	CloseHandle	Close a file
read	ReadFile	Read data from a file
write	WriteFile	Write data to a file
lseek	SetFilePointer	Move the file pointer
stat	GetFileAttributesEx	Get various file attributes
mkdir	CreateDirectory	Create a new directory
rmdir	RemoveDirectory	Remove an empty directory
link	(none)	Win32 does not support links
unlink	DeleteFile	Destroy an existing file
mount	(none)	Win32 does not support mount
umount	(none)	Win32 does not support mount
chdir	SetCurrentDirectory	Change the current working directory
chmod	(none)	Win32 does not support security (although NT does)
kill	(none)	Win32 does not support signals
time	GetLocalTime	Get the current time

Chamadas ao Sistema

Process management

process id	Call	Description
	pid = fork() → novo processo	Create a child process identical to the parent
	pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
	s = execve(name, argv, environp)	Replace a process' core image
	exit(status) → termina o processo	Terminate process execution and return status

→ passou a correr outro programa
(substituir o programa que estamos a executar)

Chamadas ao Sistema

File management

Call	Description
fd = open(file, how, ...)	Open a file for reading, writing or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = <u>lseek(fd, offset, whence)</u>	Move the file pointer → ex: escrever a partir de outra pos.
s = stat(name, &buf))	Get a file's status information

Para ler ou escrever
em outro sitio...

Chamadas ao Sistema

Directory and file system management

Call	Description
<code>s = mkdir(name, mode)</code>	Create a new directory
<code>s = rmdir(name)</code>	Remove an empty directory
<code>s = link(name1, name2)</code>	Create a new entry, name2, pointing to name1
<code>s = unlink(name)</code>	Remove a directory entry
<code>s = mount(special, name, flag)</code>	Mount a file system
<code>s = umount(special)</code>	Unmount a file system

Chamadas ao Sistema

Miscellaneous

Call	Description
<code>s = chdir(dirname)</code>	Change the working directory
<code>s = chmod(name, mode)</code>	Change a file's protection bits
<code>s = kill(pid, signal)</code>	Send a signal to a process
<code>seconds = time(&seconds)</code>	Get the elapsed time since Jan. 1, 1970

Interrupções/Excepções

- Considerar um computador com apenas 1 CPU que está a executar o seguinte código:

Linguagem de alto nível:

```
while(1) {  
    i++;  
}
```

Em um CPU antigo ficava freez e temos de reiniciar o PC.

- Como pode o Sistema Operativo obter o controlo do computador?

→ O SO é executado periodicamente! //

→ Ele vai ver se está tudo OK! //

{ → Todos os processos do utilizador têm um tempo limite de CPU.
→ Vem uma rotina de atendimento de interrupção e depois volta a executá-lo no mesmo sitio

Escalonador: módulo de SO que determina o processo a executar na CPU a cada momento! //

Continuamos a ter controlo do SO, podemos utilizar o CPU para outros tarefas. Mesmo quando temos algo em loop a decorrer na CPU

Reconhecimento de exceção / interrupção: (todos os CPU's têm)
→ todos os processos têm um limite de utilização, para limitar isso temos a rotina de atendimento de interrupção que faz o SO decidir se continua a operar o processo ou muda (para evitar freezes)

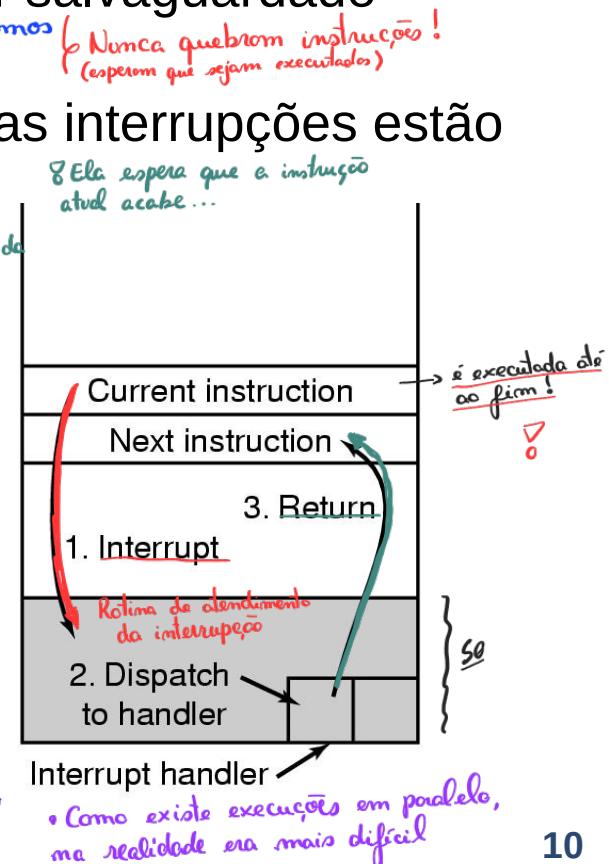
→ Processos de utilizadores

Abandonar e vai correr uma rotina de atendimento de interrupção
código do SO

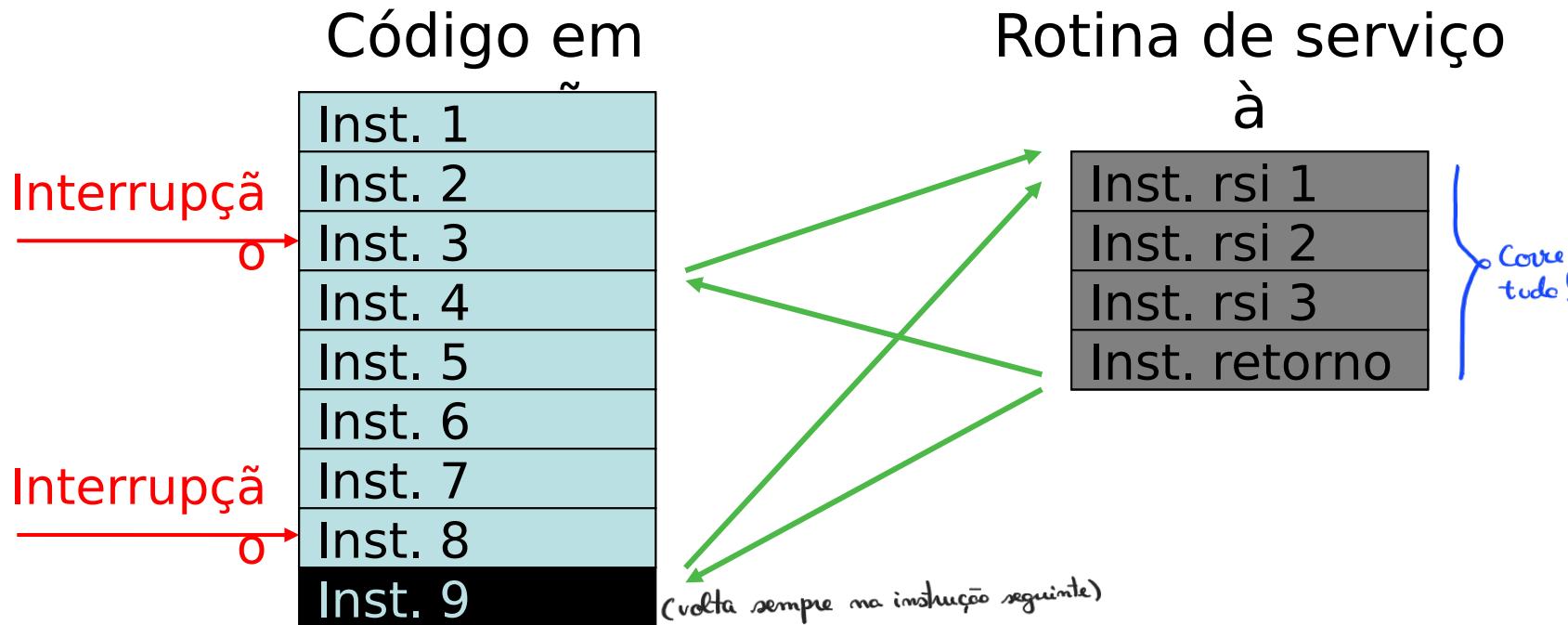
Interrupções/Excepções

- Quando o CPU deteta uma interrupção **abandona o código** que está a executar e **transfere o controlo** para a **rotina de atendimento da interrupção**
- O endereço da instrução interrompida deve ser salvaguardado
 - Porquê? → *No caso de continuarmos a executar o processo, precisamos de voltar como se nada se estivesse passado* *Numca quebrem instruções!* *(esperam que sejam executados)*
- Durante a execução da rotina de atendimento as interrupções estão desativadas *Quando acaba voltamos a ativar!*
 - Problema da interrupção perdida → *Por isso essa rotina tem de ser muito rápida*
- Um **trap** ou **exceção** é uma interrupção gerada por software
 - aceder algo que não lhe pertence (exceção de software)*
Access violation, breakpoint, misaligned access, divide by 0, overflow, illegal instruction, privileged instruction
- Nos Sistemas operativos as interrupções são fundamentais

Os break points funcionam com o mesmo método, o CPU sabe que quando chegar aquela linha para é enviada para uma rotina de atendimento

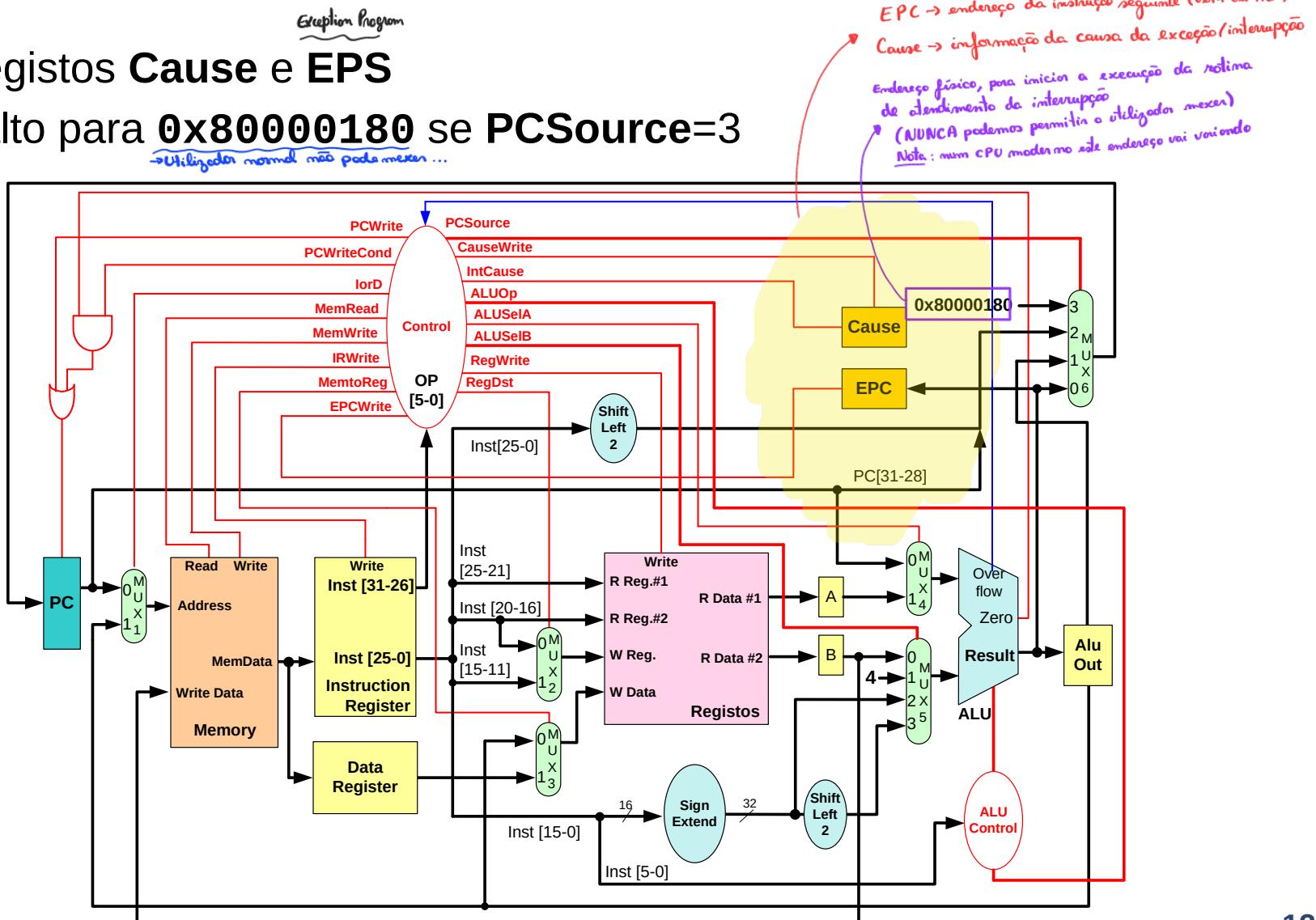


Interrupções/Excepções

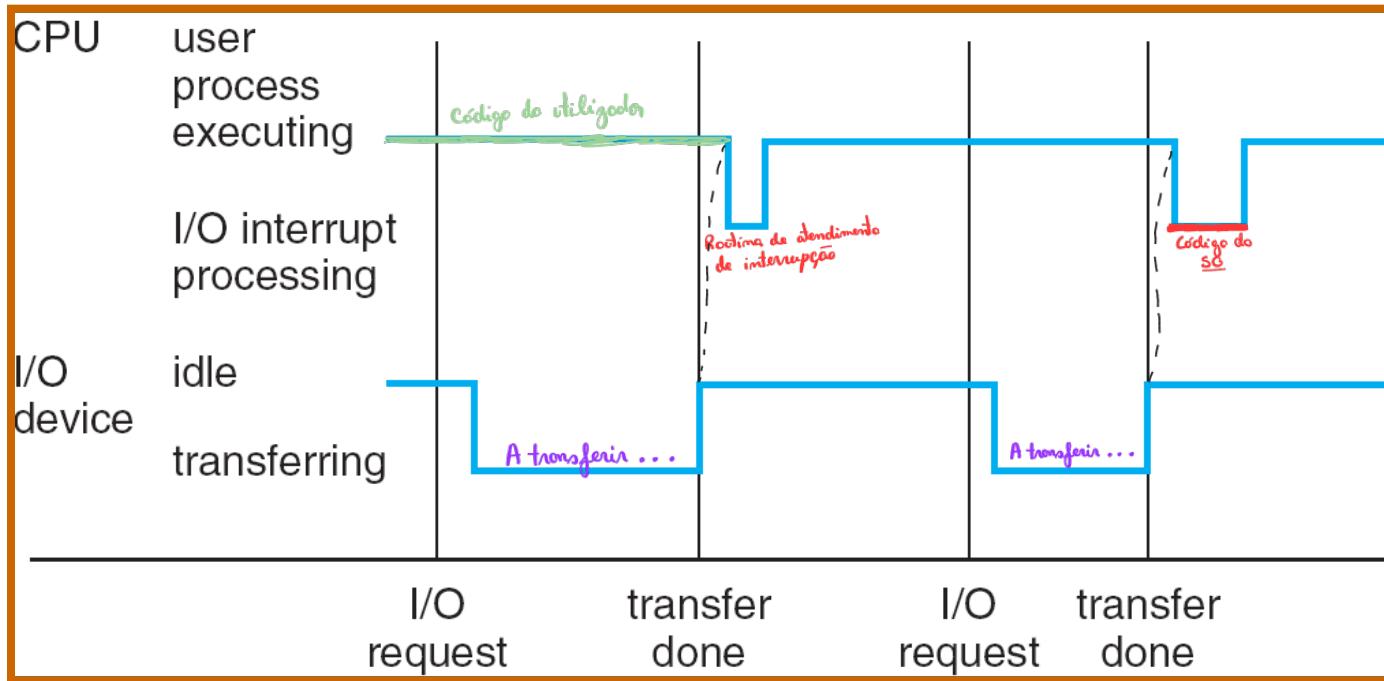


Interrupções/Excepções no MIPS

- Registros **Cause** e **EPC**
- Salto para **0x80000180** se **PCSource=3**
→ Utilizadores normais não podem mexer ...



Atendimento de uma interrupção



- Dispositivo de I/O envia interrupção ao CPU quando “*transfer done*”
- CPU executa rotina de atendimento à interrupção, no âmbito do SO, e volta a executar processo do utilizador

Tipos de Interrupções

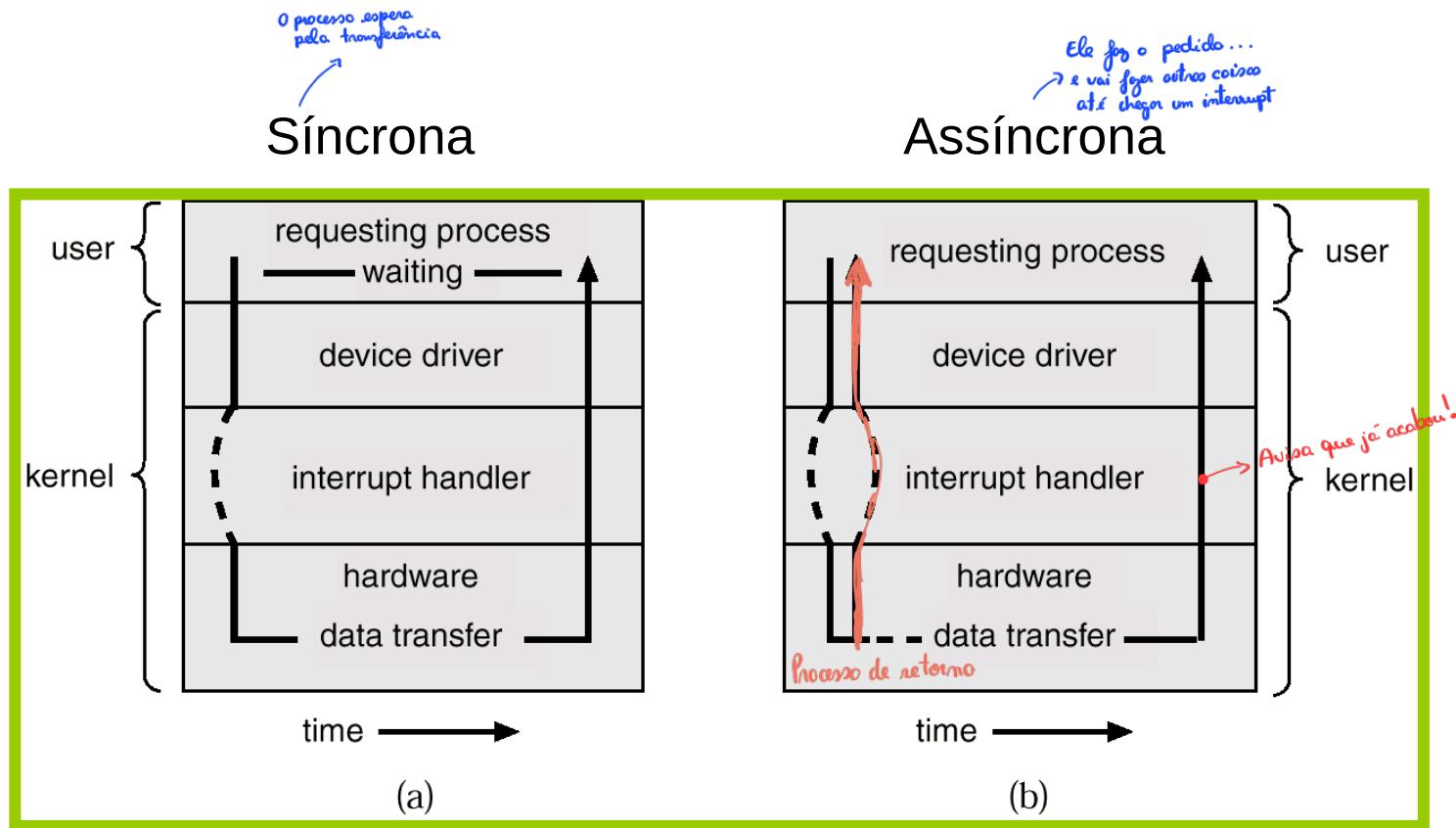
- Interrupções de I/O
 - O dispositivo de I/O pede atenção. A rotina de atendimento deve aceder ao dispositivo para determinar a ação necessária.
 - Teclado, rato, placa de rede, disco, etc.
- Interrupções de *timers*
 - Dizem ao processador que um certo intervalo de tempo já decorreu.
 - Timer local ou timer externo
- Interrupções entre processadores
 - Um processador emite uma interrupção para outro processador num sistema multi-processador

→ Interrupt entre processadores

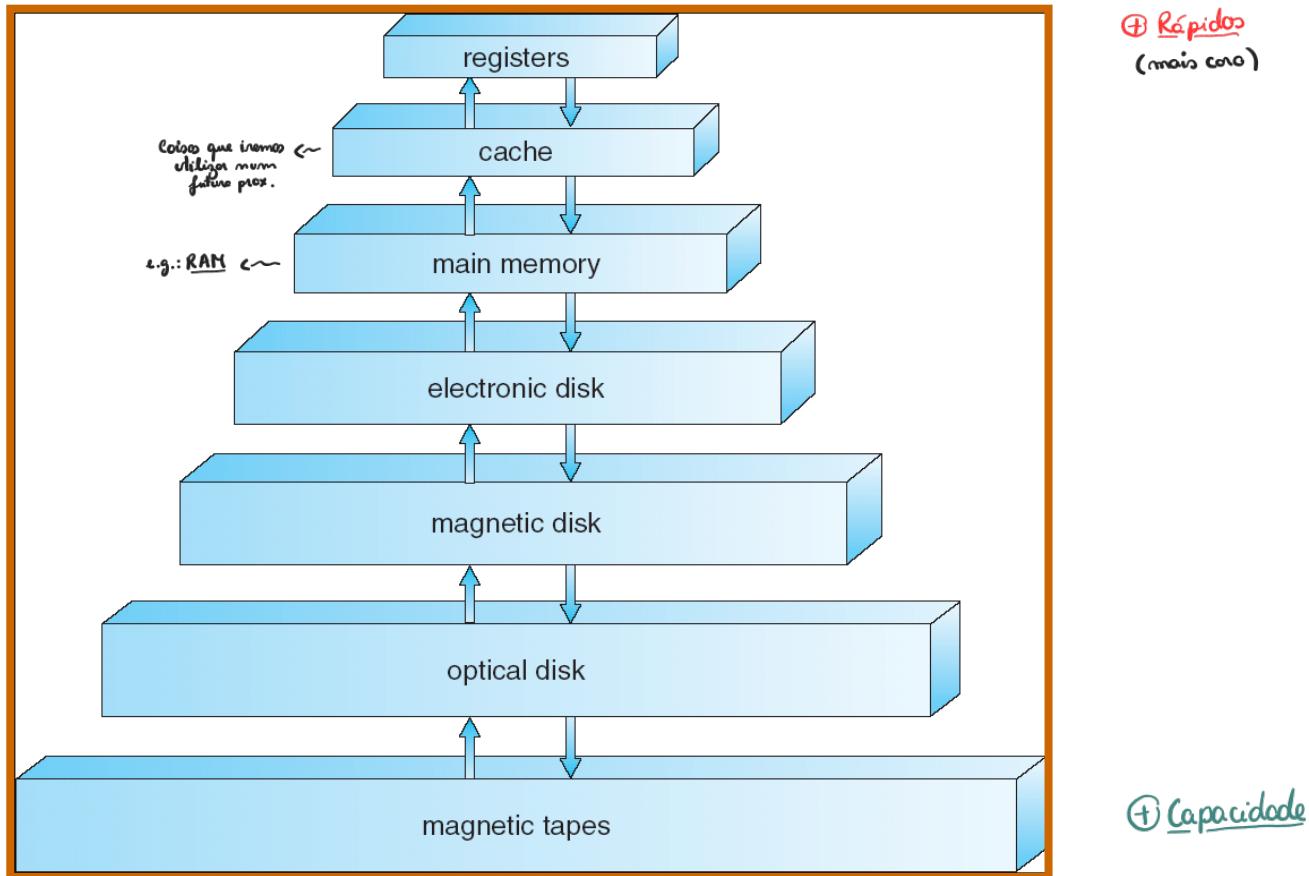
Tipos de interrupção:

- I/O (dados para enviar ou pronto para receber mais)
 - „ex: teclado/rato
 - Para ler e escrever
- Timers (periodicamente, RTC - Real Time Clock)
- Entre processadores (precisa de atenção de outro)

Organização de I/O



Hierarquia de armazenamento



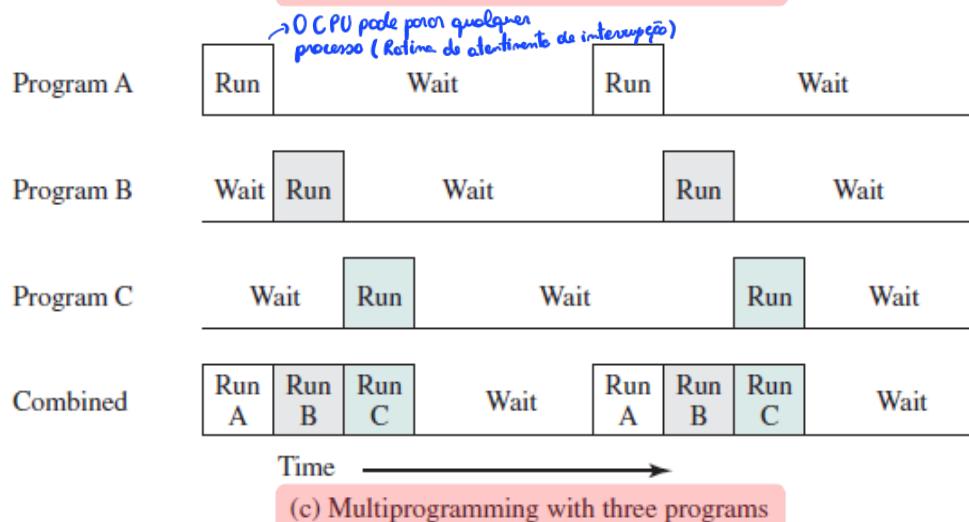
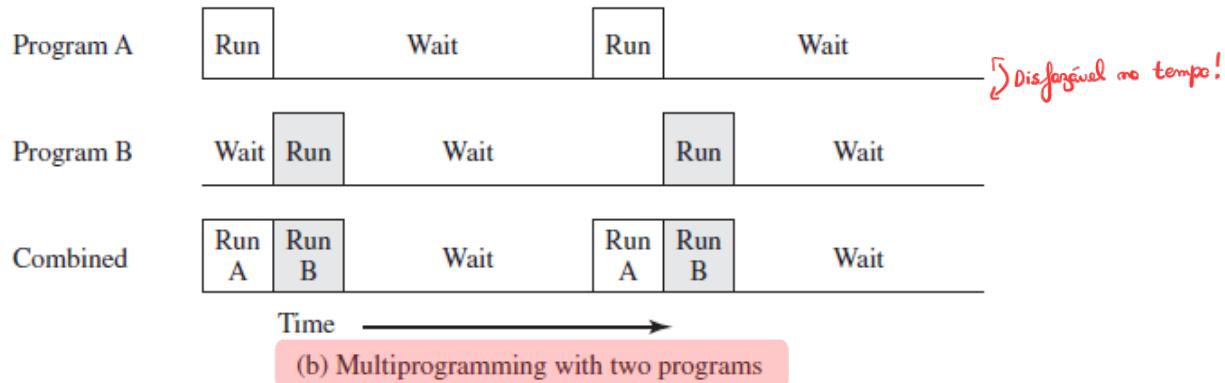
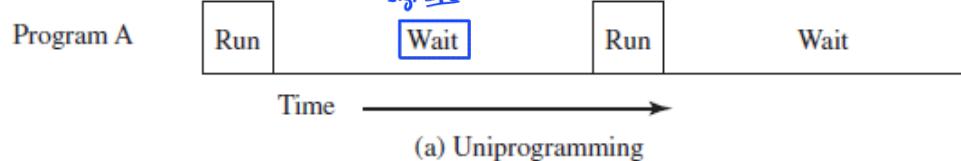
Multiprogramação

~> 1 processo não mantém o CPU constantemente ativo!„
 → Porquê? Eles estão a falar com o SO ou com o I/O e como ler do disco demora mais do que executar instruções o CPU fica parado... „
 ↓ Para aproveitar o CPU

- Permite que mais do que 1 processo (programa em execução) esteja ativo em simultâneo
- A utilização de apenas 1 processo não permite manter o CPU constantemente ativo
- Multiprogramação escolhe um novo processo para ocupar o CPU quando o processo que estava a ser executado tem de esperar (ex: chamada de I/O)
- Os processos ativos são mantidos em memória
- Escalonamento de processos

Multiprogramação

1ms é muito tempo para a CPU



→ O CPU pode parar qualquer processo (Rotina de atendimento de interrupção)

Timesharing

- CPU altera o processo em execução mesmo que este não necessite de esperar
- A cada processo é atribuído um tempo máximo de ocupação consecutiva do processador
Rotin da execução e coloca outro ...
- Se esse tempo é esgotado o SO **muda o contexto** do processador para outro processo
- Diminui muito o tempo de resposta de aplicações interactivas → *Queremos mudar de aplicação rapidamente... E cada aplicação tem um processo! //*
- Permite que vários utilizadores usem o mesmo sistema computacional como se dispusessem do sistema em exclusivo



Listas de interrupções

- Para visualizar configuração de interrupções
 - Windows
 - Aplicação **MSInfo32.exe**
 - Hardware Resources -> IRQs
 - Linux
 - Ficheiro **/proc/interrupts**

Projeto do SO

- Conceitos a separar:
 - **Política:** O que será realizado?
 - **Mecanismo:** Como será realizado?
- Mecanismos determinam como realizar a função, enquanto que a Política define o que vai acontecer
 - A separação permite aumentar a capacidade de adaptação do sistema.
 - Ao integrar mecanismos sem política associada, o sistema torna-se facilmente adaptável a diferentes políticas
 - Ex: Escalonamento do Solaris é baseado em tabelas recarregáveis

Estrutura do SO

• Monolítico

- Sistema operativo contém todas as funcionalidades de forma estática
- Permite código otimizado, pouco flexível, ocupa mais memória
- Ex: MS-DOS; UNIX
(configurável)

acesso completo ao código do SO ?

• Modular

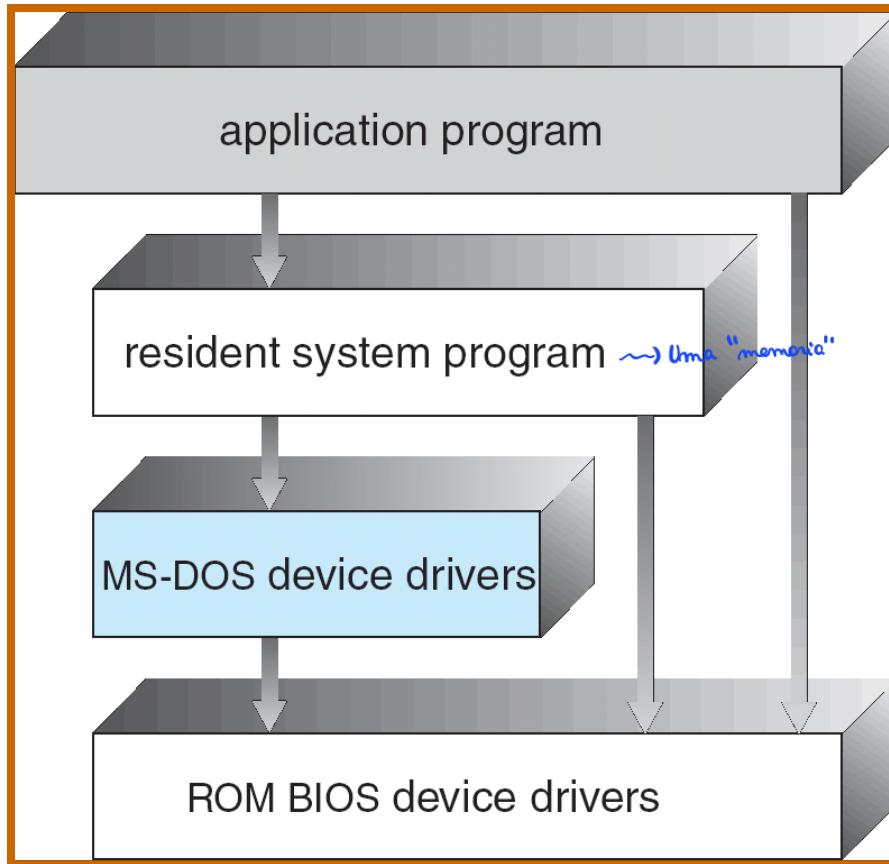
- Sistema operativo permite adição/configuração de funcionalidades através de integração de módulos
- Custo/Overhead da API, mais flexível, menos memória
Application Program Interface adicionar módulos... (nova placa gráfica) apenas o que precisamos ...
- Ex: Solaris, Linux

• Microkernel

- Kernel apenas com serviços básicos: *thread*, *address space*, *ipc*
Núcleo do SO
- Várias funcionalidades associadas ao SO correm em modo utilizador
- Pouca memória, verificável, mudanças de kernel mode para user mode são frequentes
- Ex: MACH, MINIX, QNX

threads espaço de endereçamento interrupts

MS-DOS (Monolítico)



1 aplicação de cada vez ...

Organização em camadas

→Conceito!

