

Sistemas Operativos

Licenciatura Engenharia Informática
Licenciatura Engenharia Computacional

Ano letivo 2023/2024

Nuno Lau (nunolau@ua.pt)

Interrupções/Excepções

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

Linguagem de alto nível:

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

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

Continuaremos a ter controlo do SO, podemos utilizar o CPU para outros tarefas. Mesmo quando temos algo em loop a decorrer no CPU

Reconismo 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)

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

→ Processos de utilizadores

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

→ O SO é executado periodicamente! //
→ Ele vai ver se está tudo OK! //

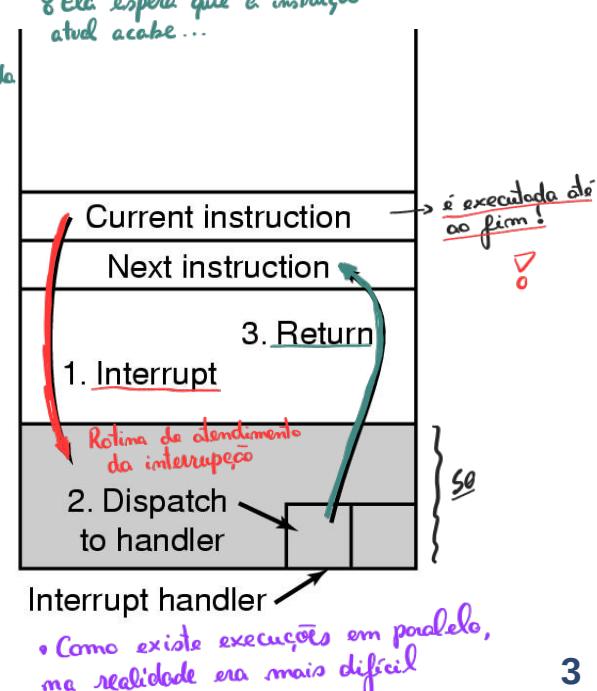
→ Todos os processos do utilizador têm um tempo limite de CPU.
→ Vem uma rotina de atendimento de interrupção e depois volta a executá-lo no mesmo sitio

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

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 tivesse passado* → *Numa quebram 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
 - 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



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

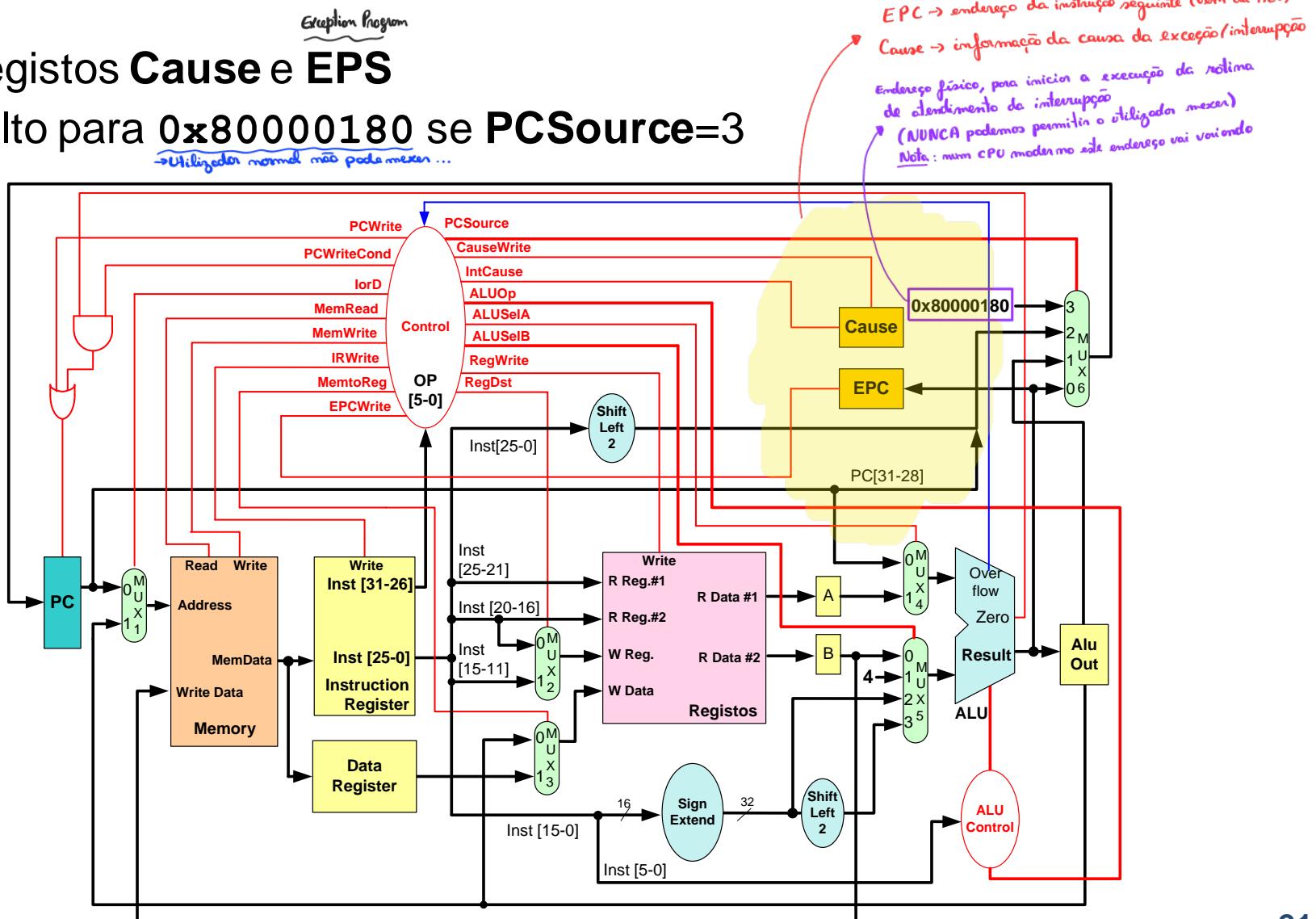
→ Interrupt entre processadores

Tipos de interrupção:

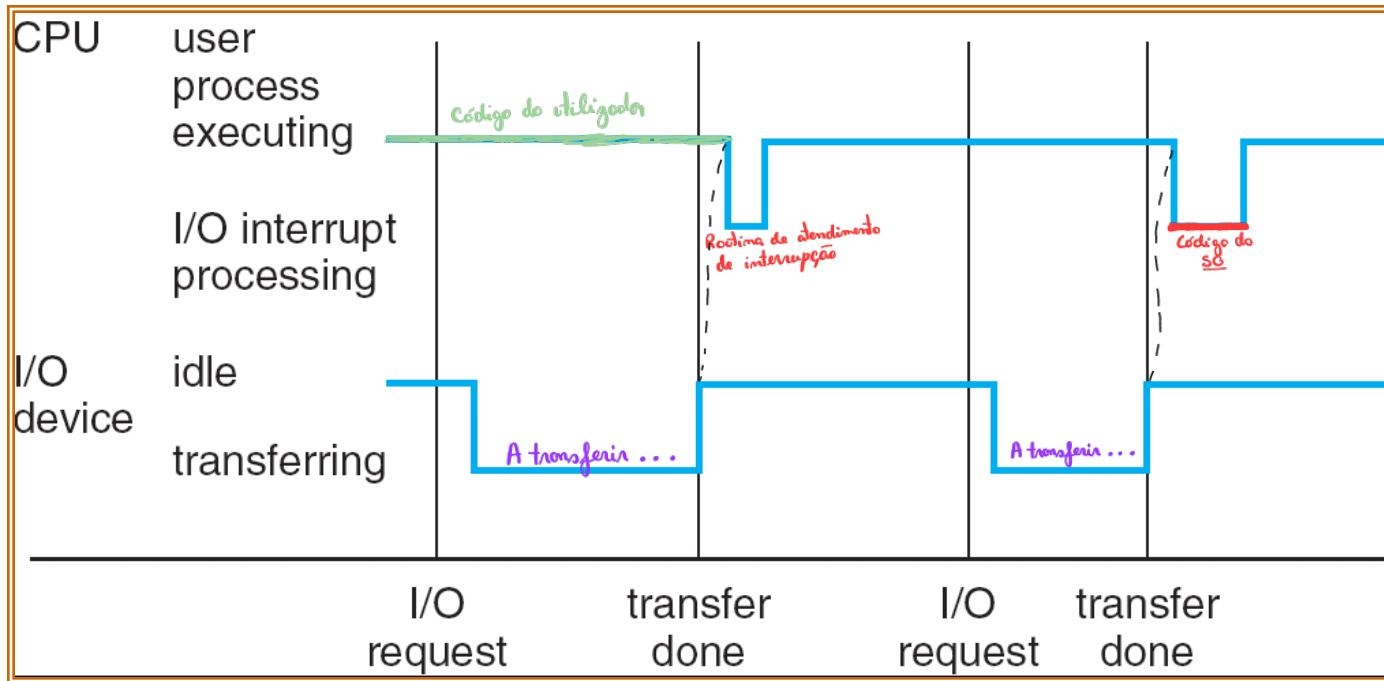
- I/O (dados para enviar ou pronto para receber mais)
 - ex: teclado/rato
 - Para e escreve
- Timers (periodicamente, RTC - Real Time Clock)
- Entre processadores (precisa de atenção de outro)

Interrupções/Excepções no MIPS

- Registros **Cause** e **EPS**
 - Salto para 0x80000180 se **PCSource=3**



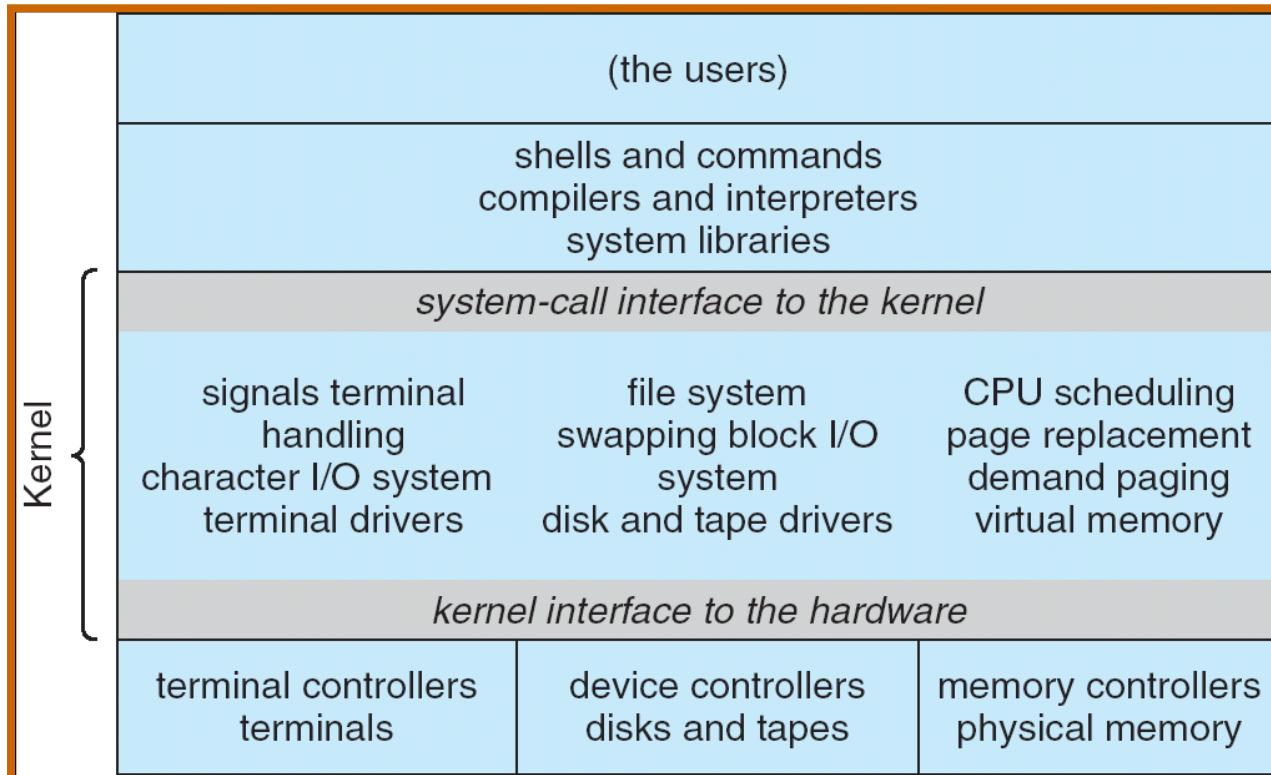
Atendimento de uma interrupção



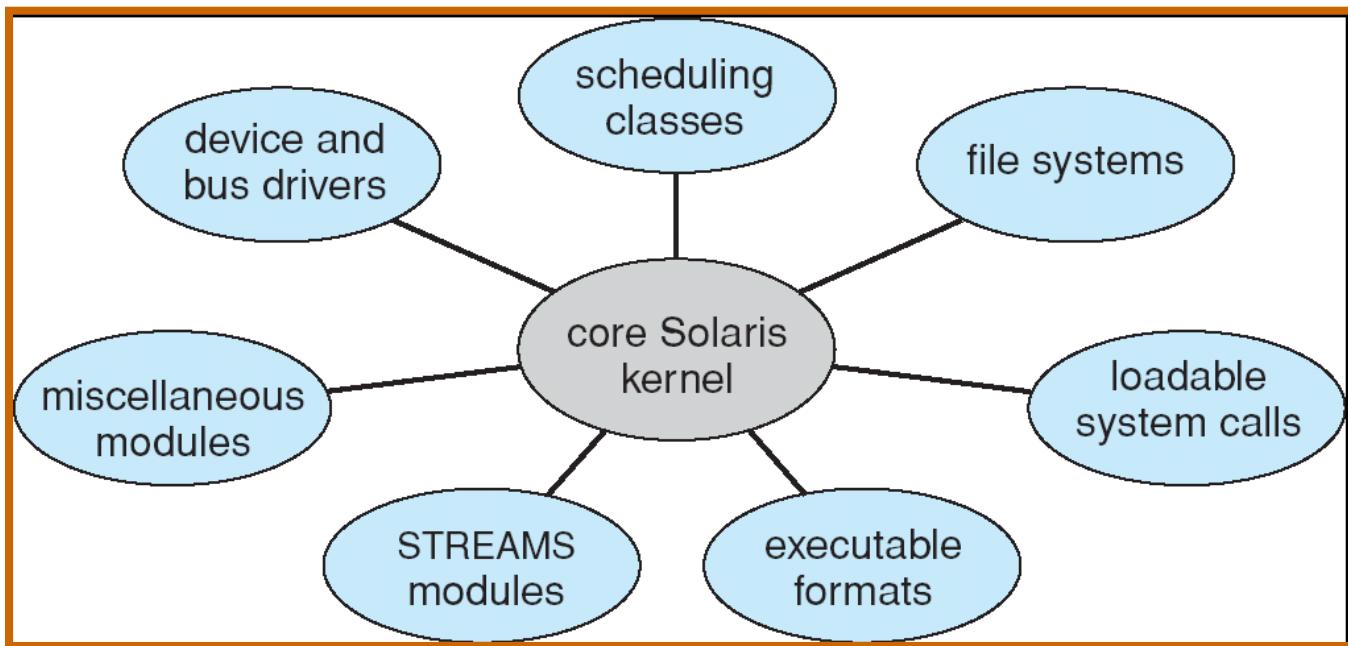
- Dispositivo de I/O envia interrupção ao CPU quando “*transfer done*”
- CPU executa rotina de atendimento à interrupção, no âmbito do SO, e volta a executar processo do utilizador

Estrutura do SO

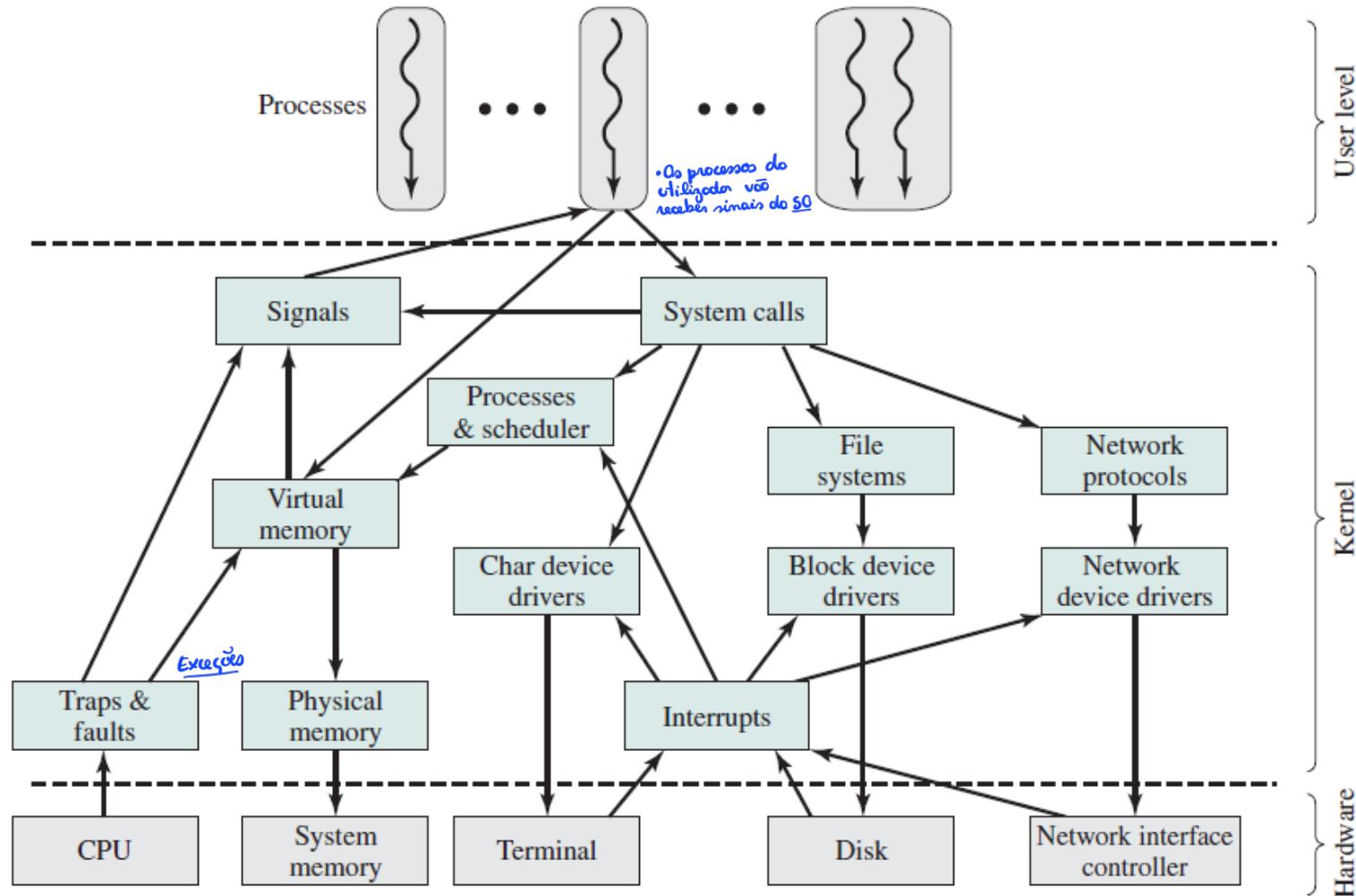
- **Monolítico**
 - Sistema operativo contém todas as funcionalidades de forma estática
 - Permite código otimizado, pouco flexível, ocupa mais memória
 - Ex: MS-DOS; UNIX
- **Modular**
 - Sistema operativo permite adição/configuração de funcionalidades através de integração de módulos
 - Custo/Overhead da API, mais flexível, menos memória
 - Ex: Solaris, Linux
- **Microkernel**
 - Kernel apenas com serviços básicos: *thread, address space, ipc*
 - 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



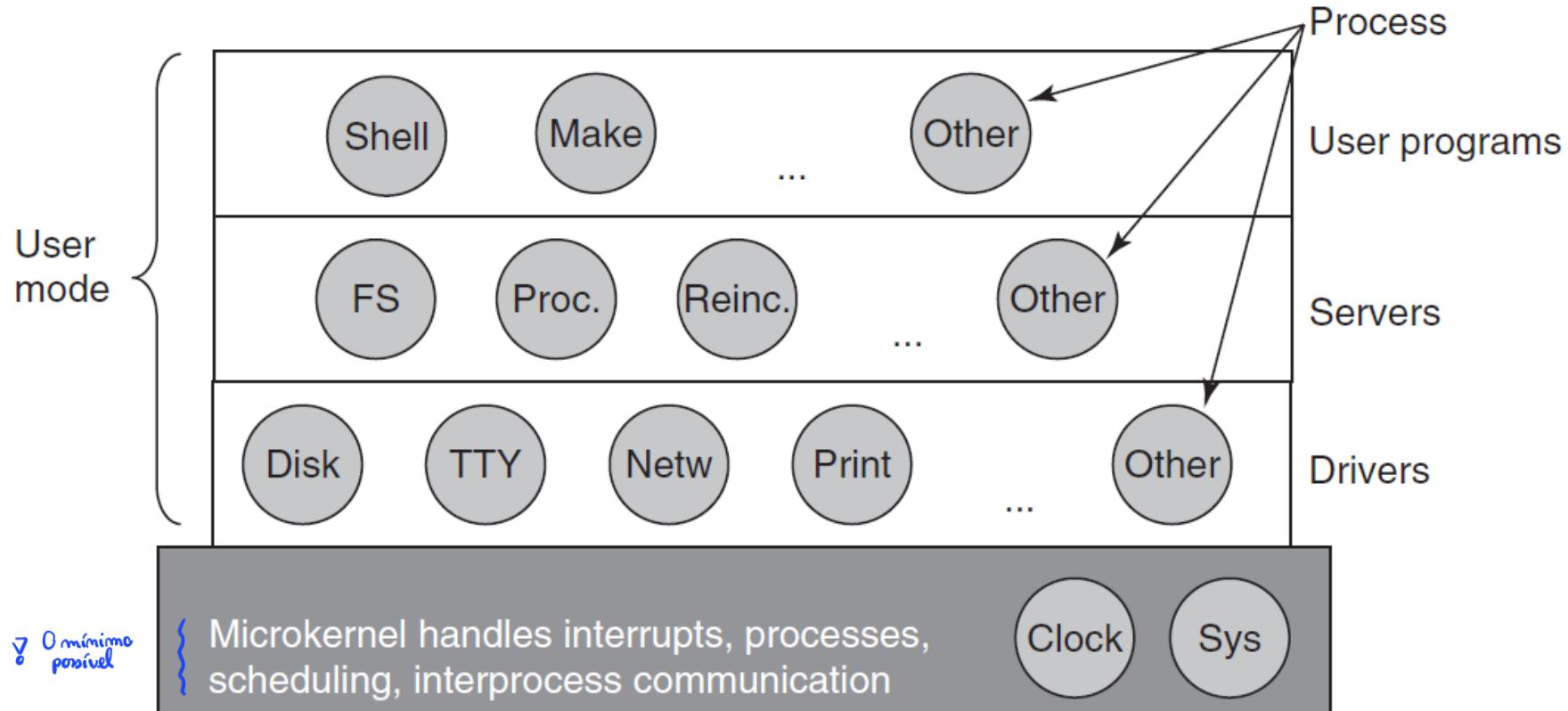
Abordagem Modular (Solaris)



Linux (Parecido ao Unix)



Microkernel (MINIX)



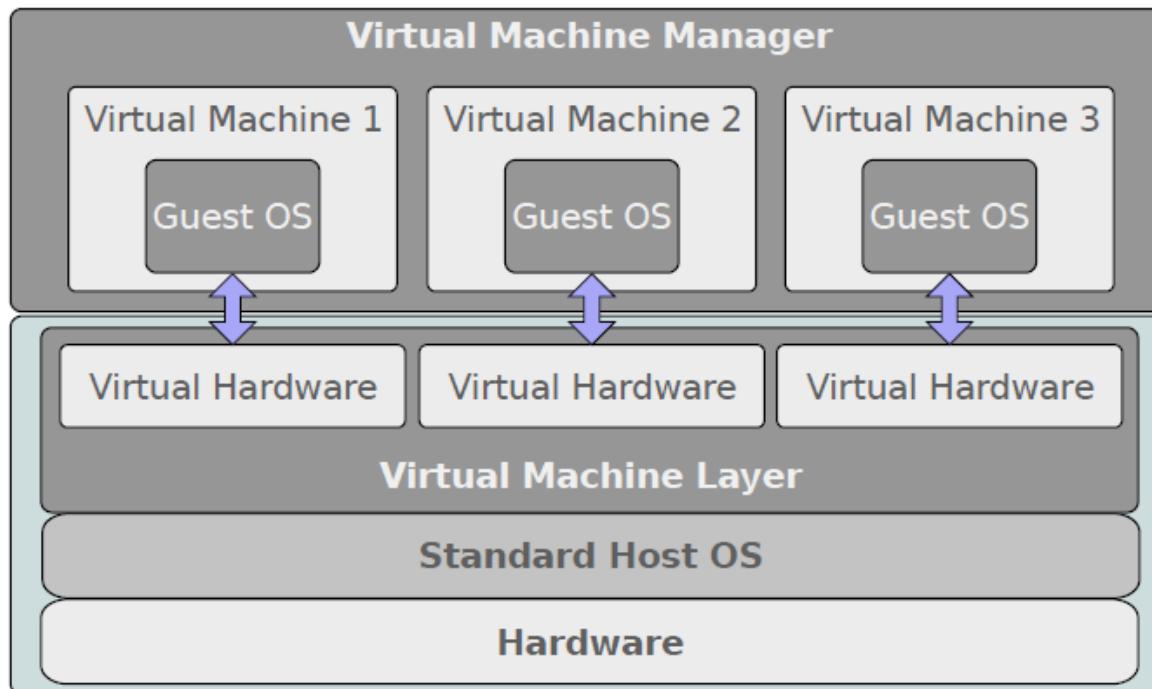
- Open Source project (MINIX 3 – www.minix3.org)
- **Kernel has ~4k lines of code**
 - Linux has 5M lines of code

Máquinas Virtuais

- As máquinas virtuais levam a organização em camadas até ao limite
- Ambientes de execução virtuais, possivelmente a executar SO distintos, em que os recursos do computador (CPU, memória, discos, etc) parecem ser exclusivos apesar de serem partilhados *{estamos a partilhar}*
- O SO cria a ilusão de que cada processo corre no seu processador com a sua memória
- Vários sistemas operativos podem executar concorrentemente em máquinas virtuais distintas tendo cada um o seu conjunto de processos em execução

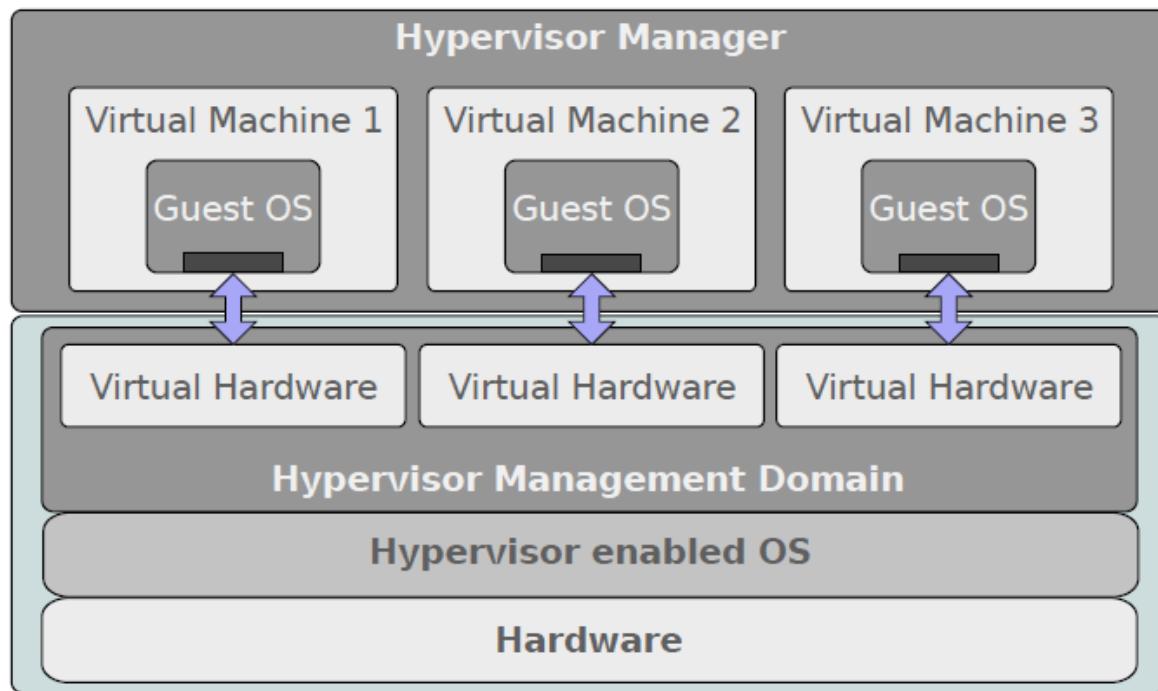
Full Virtualization

- Emulação do hardware
 - Através do *host OS*
- Transparente para o(s) *guest SO(s)*
- Exemplos: Virtual Box, VMWare



Paravirtualization

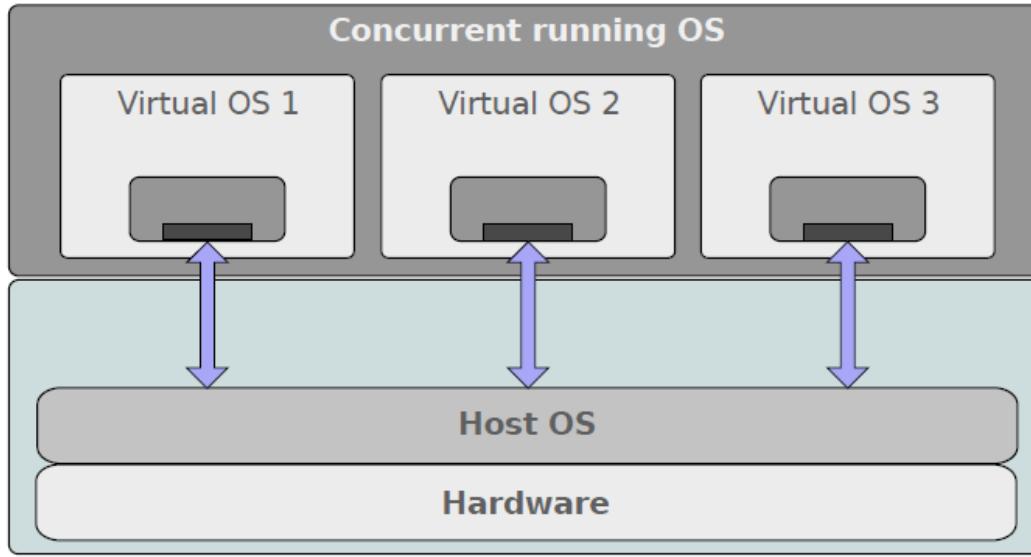
- Execução concorrente de vários SOs
- Necessita de suporte de hardware e dos SOs
- Exemplos: Xen, UML

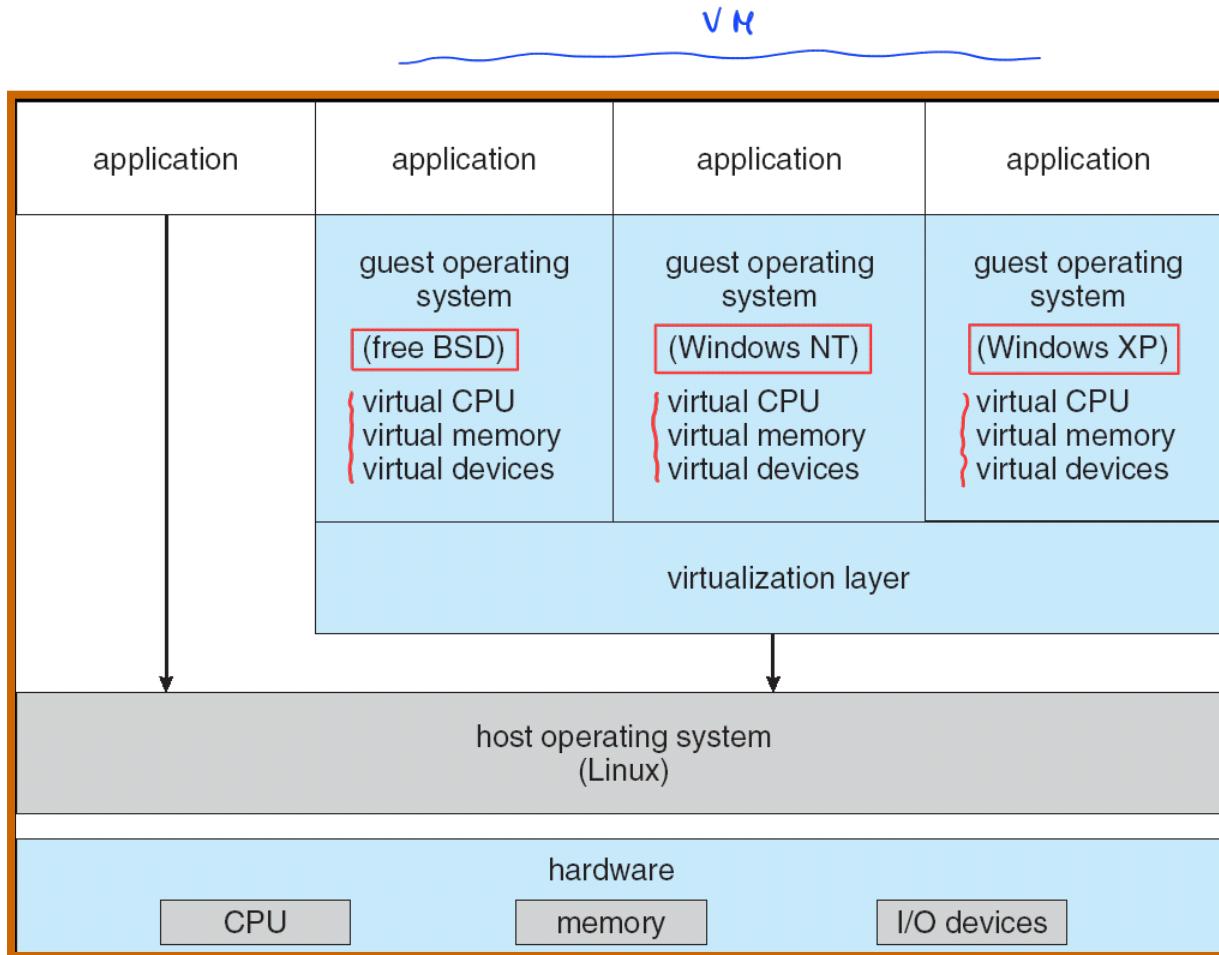


UPC, ASO, Serral

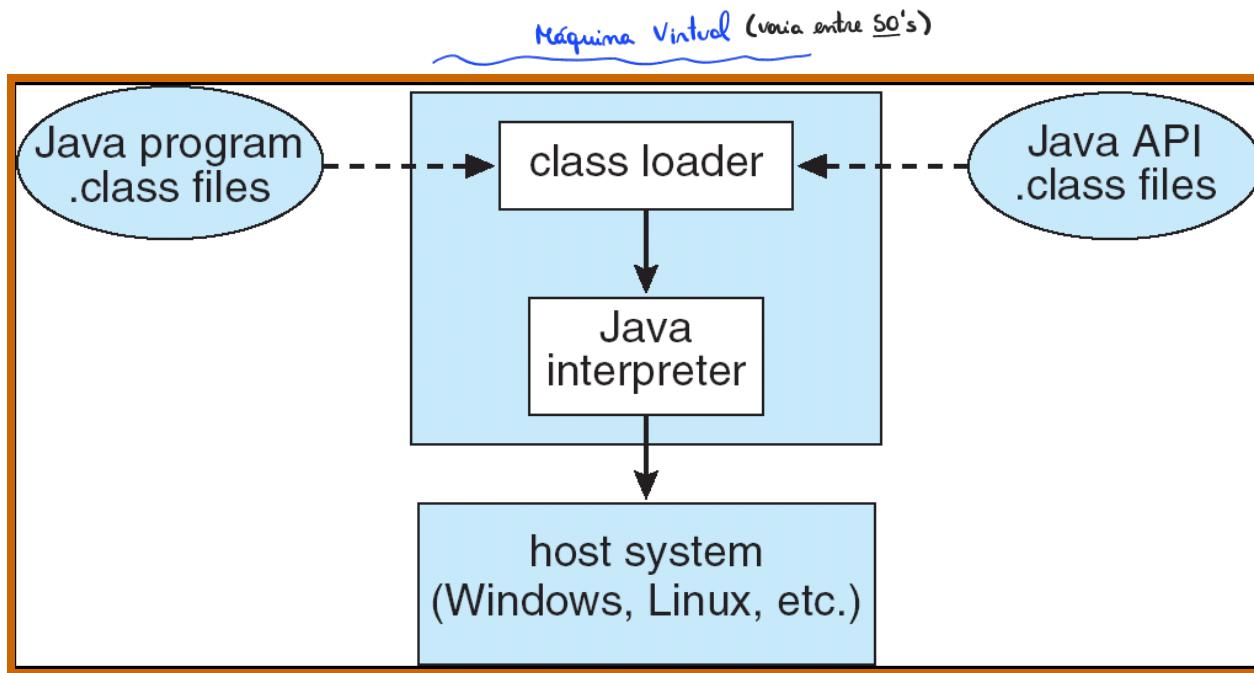
OS Virtualization

- Colaboração entre *host* e *guests*
 - Acesso direto ao hardware dos *guests*
 - Pode executar em *userspace*
- Necessita de suporte do SO
 - *Host* e *guests* usam o mesmo SO
- Exemplos: Docker.io, Solaris containers ☺

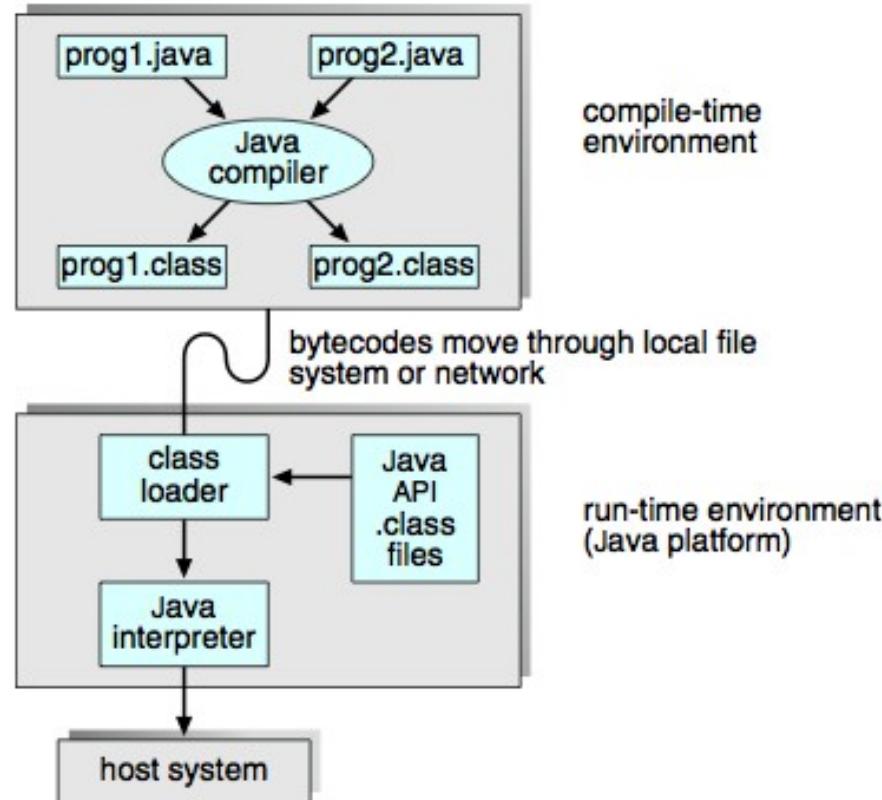




Java Virtual Machine



Desenvolvimento em Java



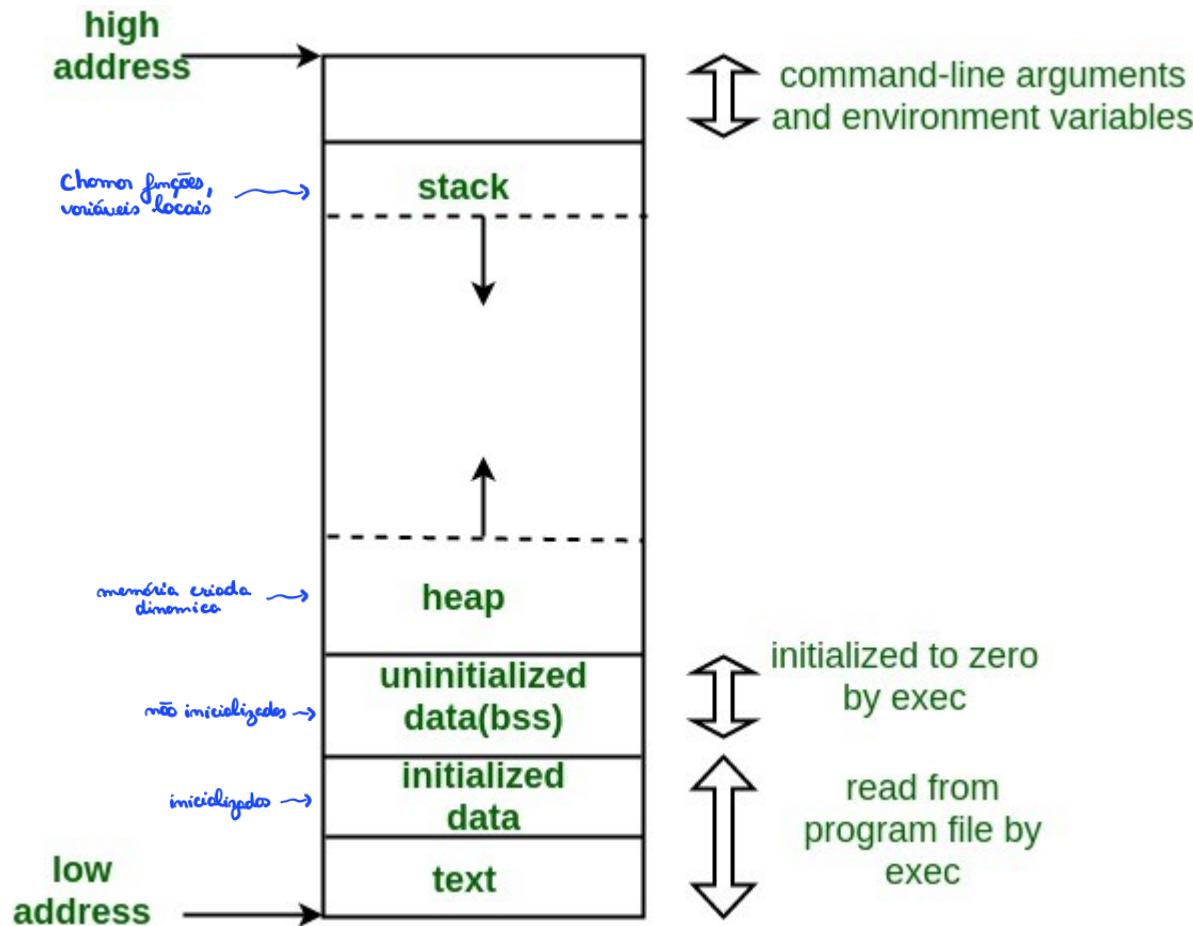
Módulos em Linux

- O Linux pode carregar e remover módulos do *kernel* durante a sua execução
 - São necessárias permissões de *superuser*
 - Comandos principais:
 - **lsmod**, **modinfo**, **modprobe**, **rmmod**
- módulos ativos*

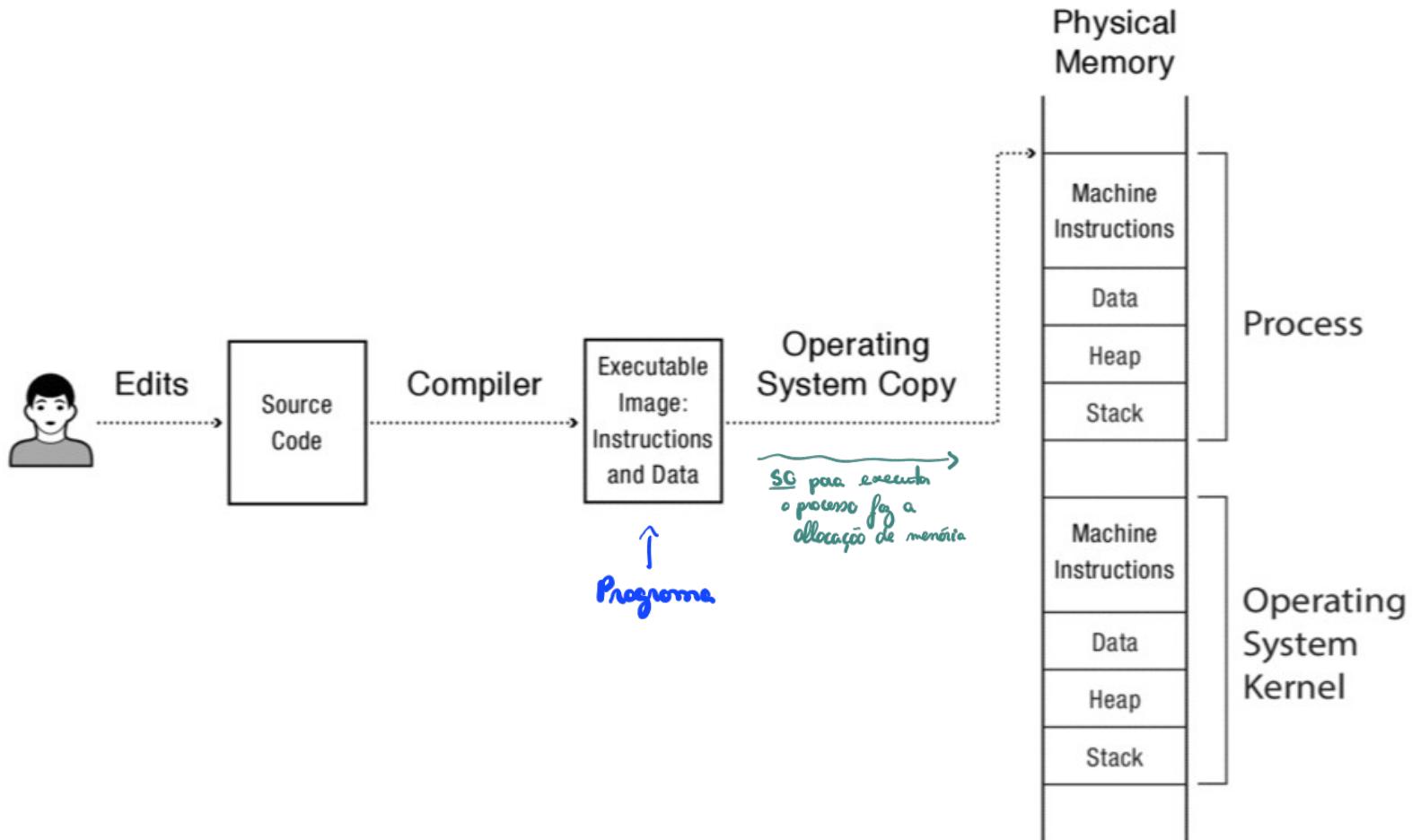
Processo

- Programa em execução
- Criar um processo
 - Inicialização do Sistema → Avançar o SO
 - Execução de chamada ao sistema por processo em execução
 - Pedido do utilizador para criar novo processo → Cada comando no bash cria um novo processo
 - Início de um *batch script* → "bash" do windows
- Processos podem correr em:
 - *foreground*: interage com utilizador → o I/O está ativo... e.g.: sleep 10
 - *background*: executa sem interação, *daemon*
→ Podem ter output...

Memória de um processo



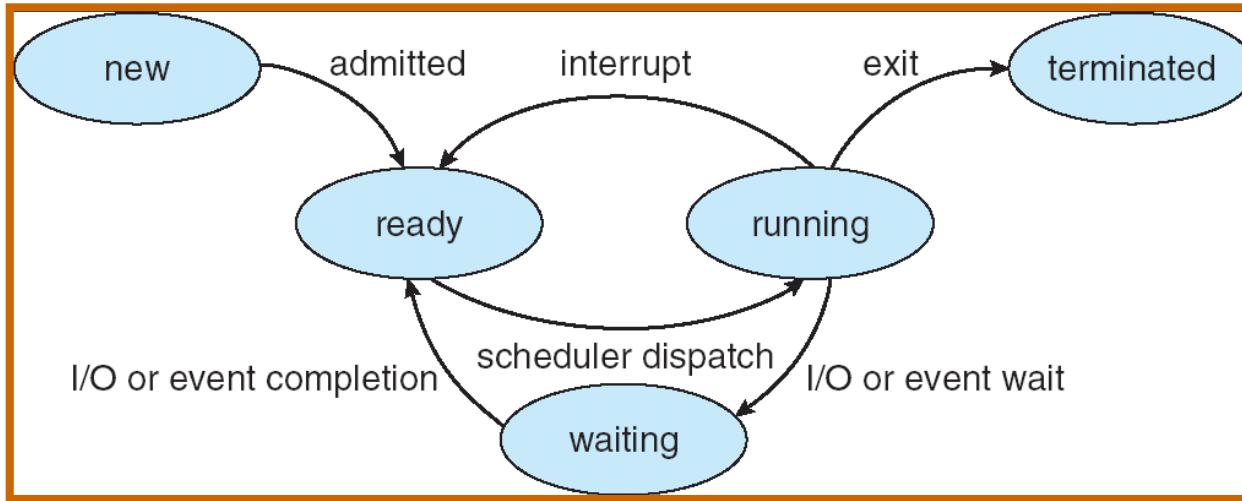
Memória processo/OS



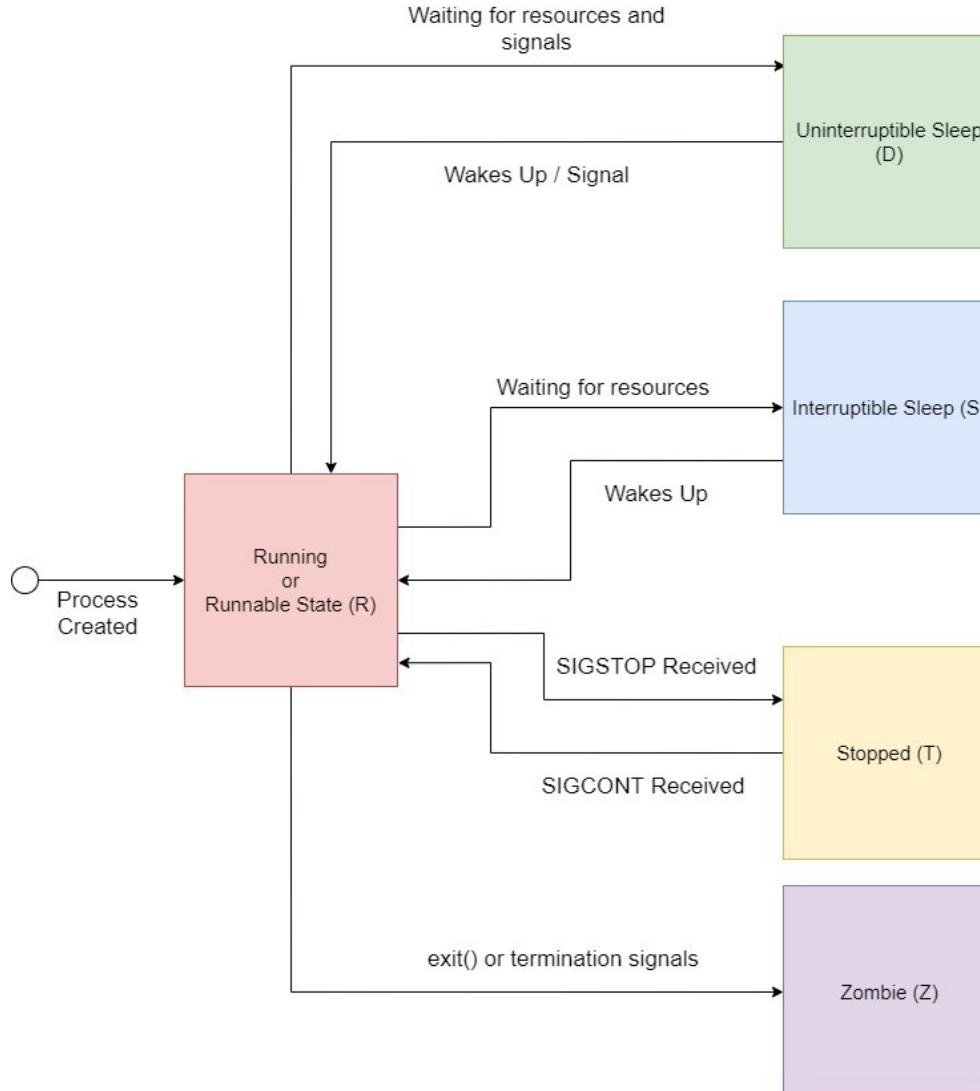
Estados de um processo

- Um processo em execução pode estar nos seguintes estados
 - New *~ Quando está a ser criado*
 - Running *→ Executor*
 - Waiting *~ Espera de alguma coisa... eg.: I/O, sleep, ...*
 - Ready *~ Eles por sua vontade estão em Running, mas à falta de recursos e eles têm de esperar ...*
 - Terminated *~ Acabou...*

Estados de um processo



Estados de um processo



Process Control Block

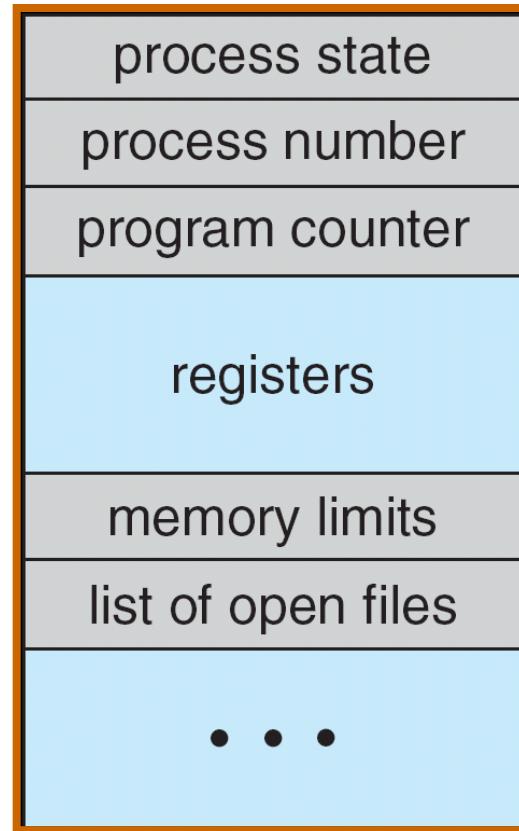
→ Estrutura de gestão dos processos! //

- O Process Control Block (PCB) é a estrutura que, no SO, armazena a informação sobre um processo
- Inclui os seguintes dados:
 - Estado do processo
 - *Program Counter* ← só nos casos Ready, Waiting
→ têm de voltar na próxima instrução
 - Registos do CPU ← nos mesmos casos...
 - Tipo de escalonamento → "escalonar" = "temporizar" ⇒ quem decide que processo deve ser executado agora, ou seja, dos que estão Ready...
 - Informação sobre a memória do processo
 - Informação sobre I/O
 - Accounting ← Tempo de CPU, ...

Temos que garantir que
restaura o contexto de execução
do processo



Process Control Block

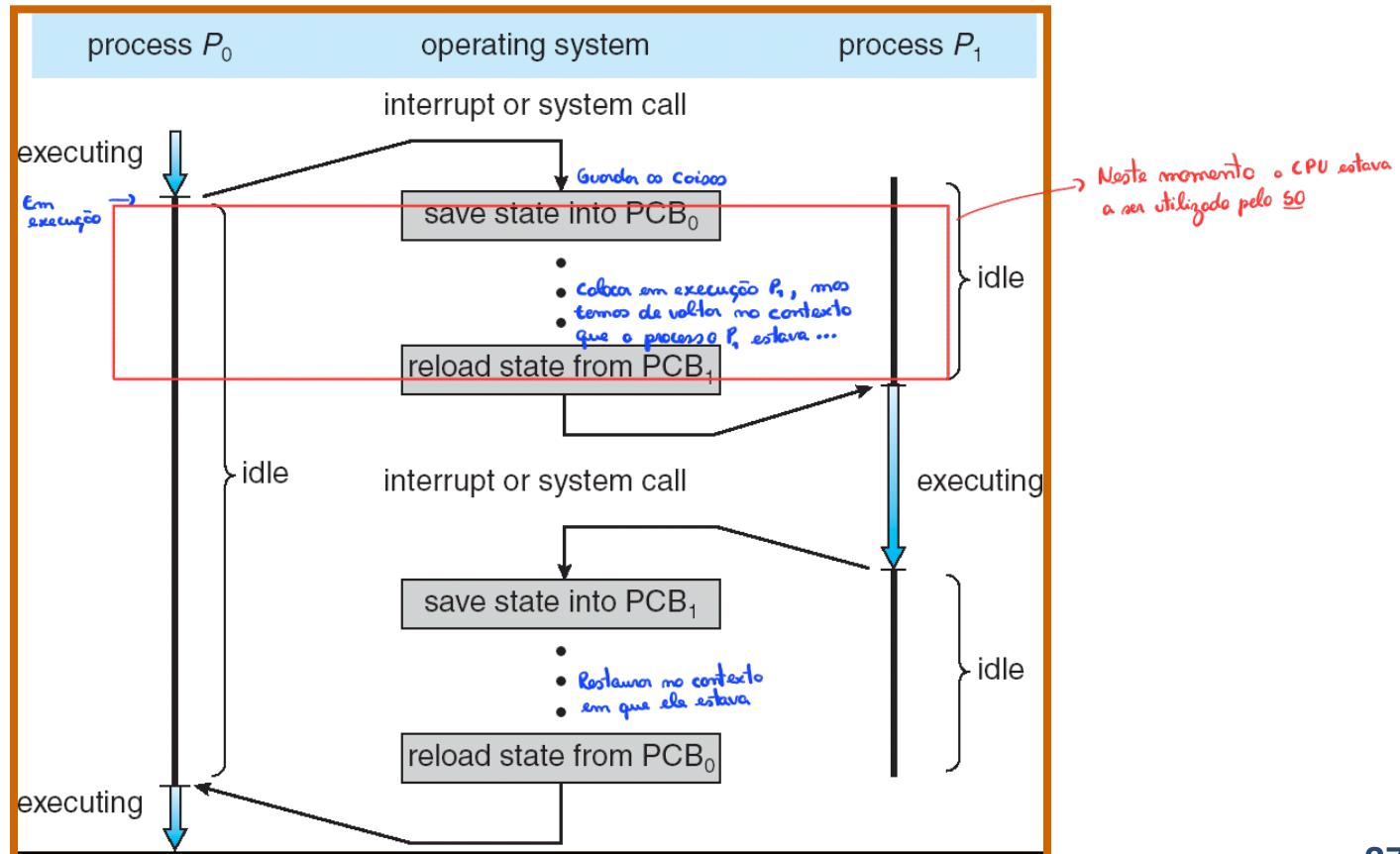


Process Control Block

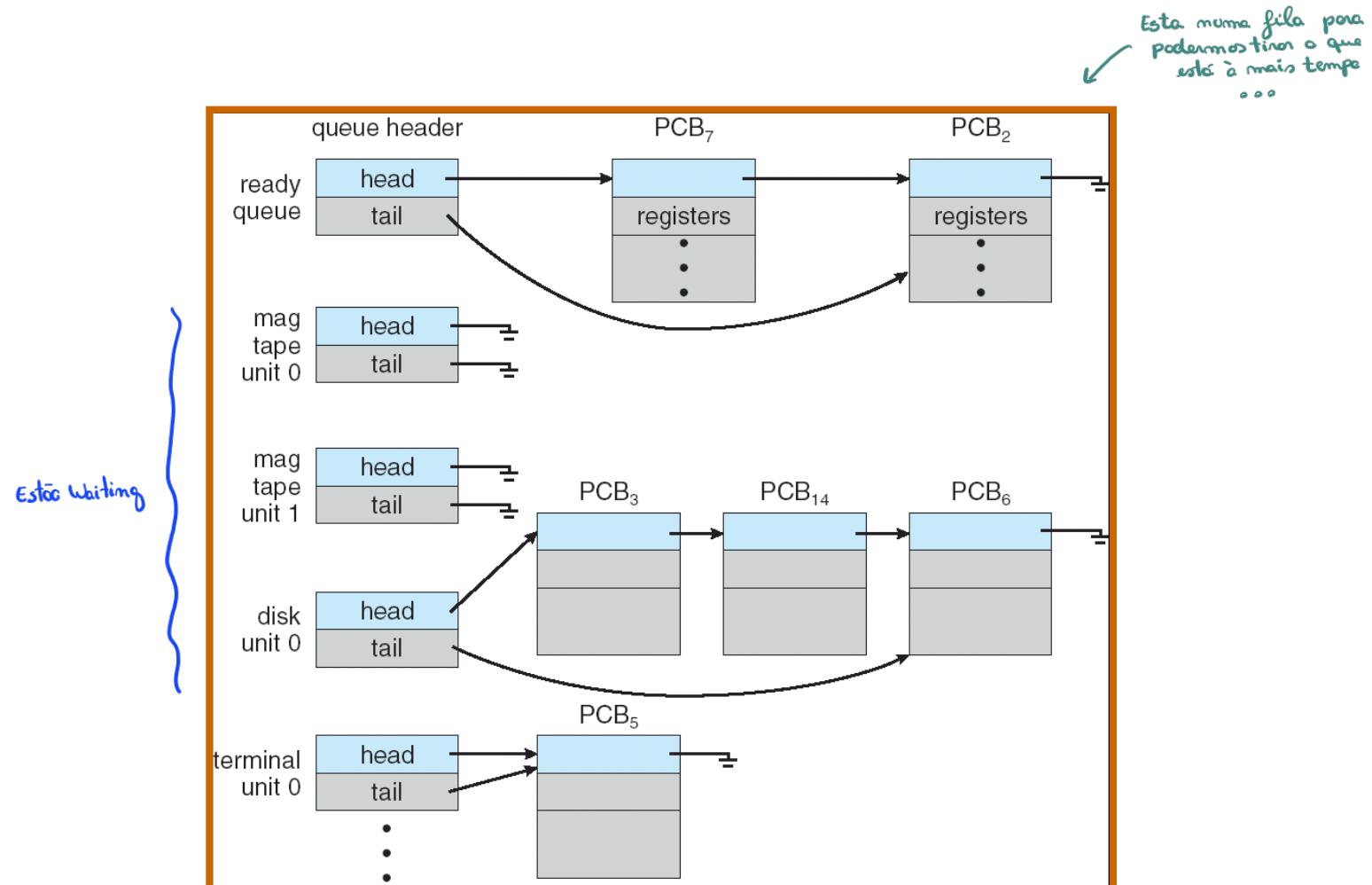
Process management	Memory management	File management
Registers Program counter Program status word Stack pointer Process state Priority Scheduling parameters Process ID Parent process Process group Signals Time when process started CPU time used Children's CPU time Time of next alarm	Pointer to text segment Pointer to data segment Pointer to stack segment	Root directory Working directory File descriptors User ID Group ID

Mudança de contexto

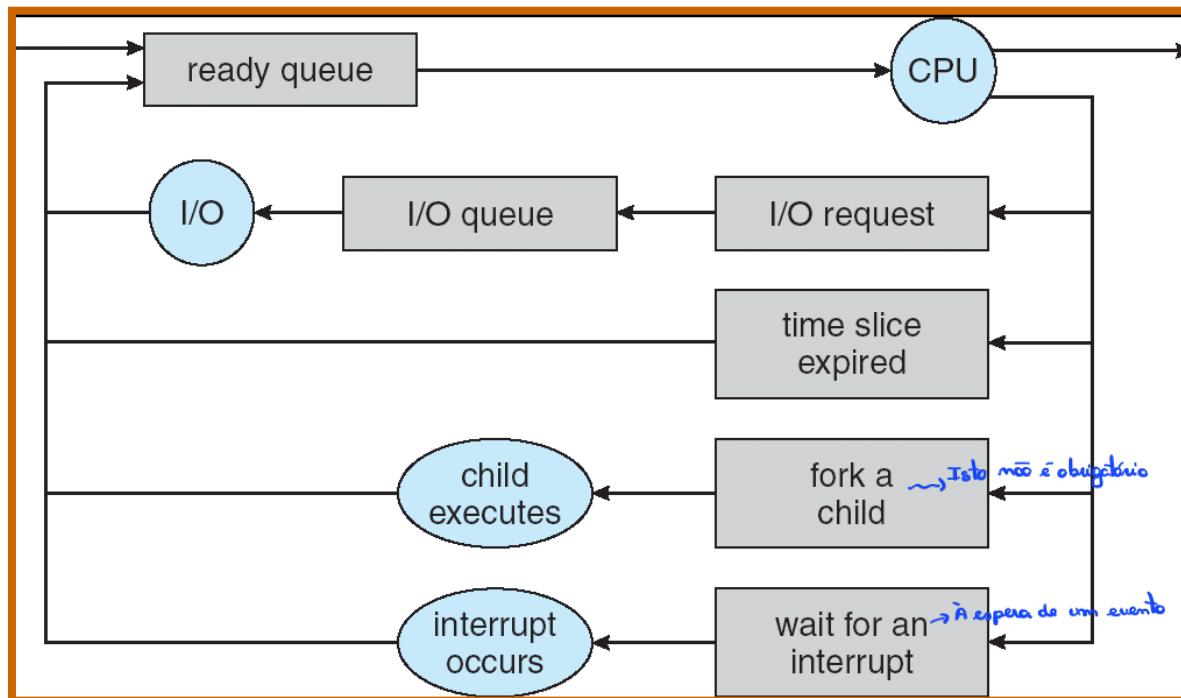
- Quando o SO troca o processo que está no estado *Running*



Estruturas de suporte ao escalonamento



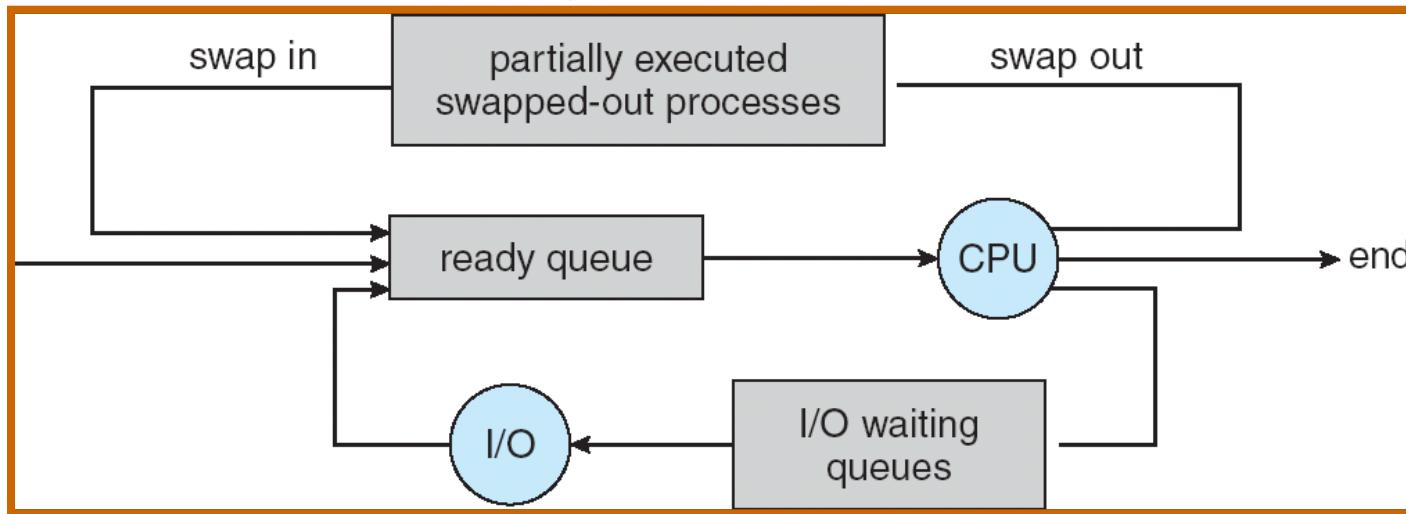
Representação do escalonamento



Representação do escalonamento

! Gestão da memória

Desloca processos para o Disco // Quando a situação melhora fazemos swap in



Tópico prático

Directória /proc

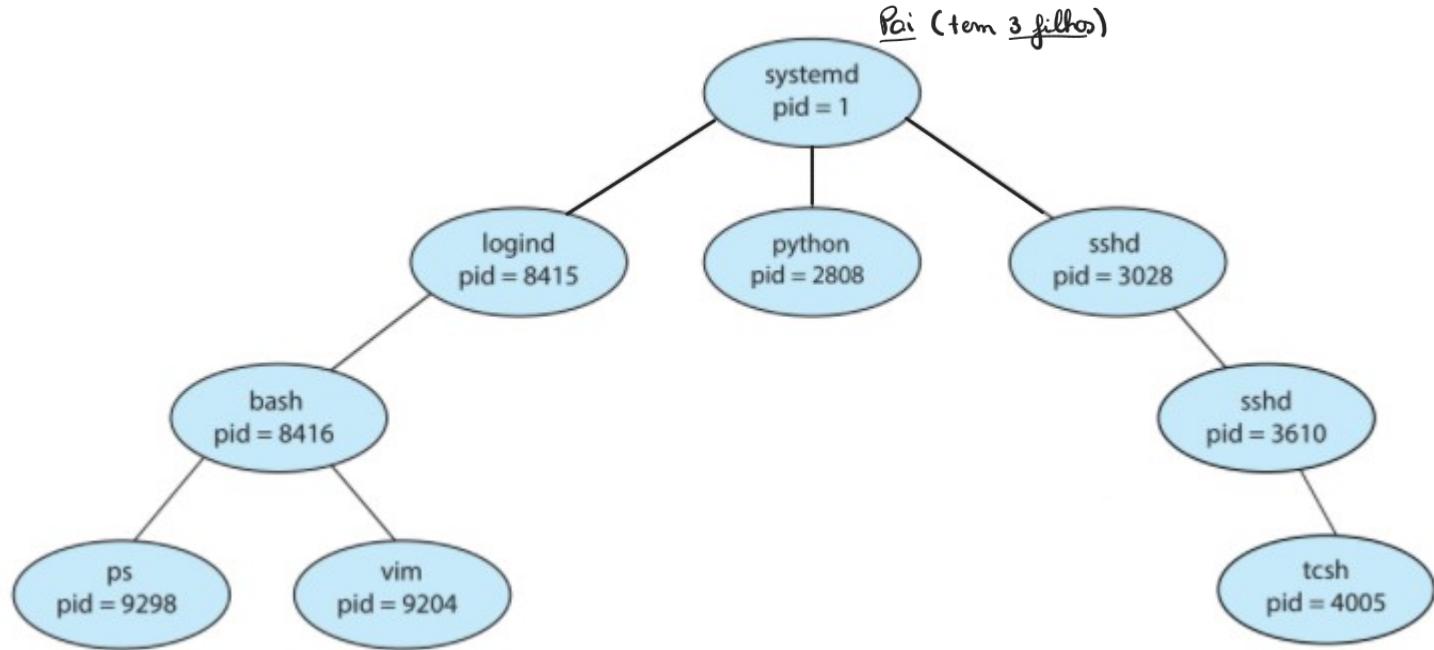
- directória virtual (ficheiros só existem em memória)
- podem ser consultadas várias informações sobre o Sistema
 - Permite alterar parâmetros do kernel
- Contém informações gerais:
 - Ex: **cpuinfo**; **meminfo**; etc
- Uma directória para cada processo
 - Ex: **root**; **cwd**; ficheiros abertos; mapa de memória; etc
- Mais informações em:

<http://www.tldp.org/LDP/Linux-Filesystem-Hierarchy/html/proc.html>

Árvore de processos

- Quando um processo cria um novo processo
 - Processo criador é designado de **processo pai**
 - Novo processo é designado de **processo filho**
- Pode ser formada uma hierarquia de processos
- Hierarquia de processos
 - Processo pode saber **pid** do pai
 - Quando filho morre é enviado o sinal **SIGCHLD** ao pai
espécie de interrupt
 - Pai recolhe *exit code* dos filhos
 - Quando pai morre, filho é herdado pelo processo 1 (**init**)
 - A partir do kernel 3.4, um processo pode nomear-se como pai dos processos orfãos seus descendentes (ex: **systemd**, **upstart**)

Árvore de processos



Árvore de processos



Tipos de processos

- I/O intensivos
 - Fazem muitas chamadas ao sistema relacionadas com I/O
 - Muitos pequenos períodos de utilização do CPU
- CPU intensivos
 - Fazem poucas chamadas I/O
 - Poucos e longos períodos de utilização do CPU
- Num sistema com *timesharing* e de modo a otimizar a utilização do CPU é positivo que a lista de processos em execução seja equilibrada entre os 2 tipos 

Criação de processos

- Um processo pode criar novos processos
 - O processo criador designa-se de processo pai e os criados de processos filhos
 - Os filhos podem, por sua vez, criar novos processos
- Partilha de recursos
 - Pai e filhos partilham recursos *⇒ Por exemplo: ficheiros abertos! //*
 - Filhos partilham um subconjunto dos recursos do pai
 - Pai e filhos não partilham recursos
- Execução
 - Pai e filhos executam em paralelo
 - Pai espera que filho(s) terminem

Criação de processos

