FACULDADE DE CIÊNCIAS E TECNOLOGIA
DEPARTAMENTO DE INFORMÁTICA

Fundamentos de Sistemas de Operação

LEI - 2023/2024

Vitor Duarte
Mª. Cecília Gomes

1

Aula 2

- Arranque *(Boot-strap)*
- SO para partilha e gestão de recursos
- Processos e chamadas ao SO

- OSTEP: cap. 2, 3, 4.0 - 4.4
- (também de AC: Dive Into Systems cap. 13)

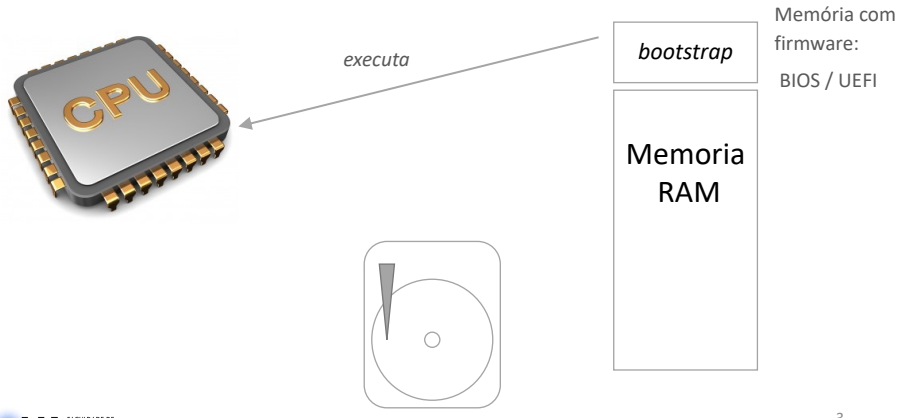
FACULDADE DE CIÊNCIAS E TECNOLOGIA
DEPARTAMENTO DE INFORMÁTICA

2

2

Arranque do SO

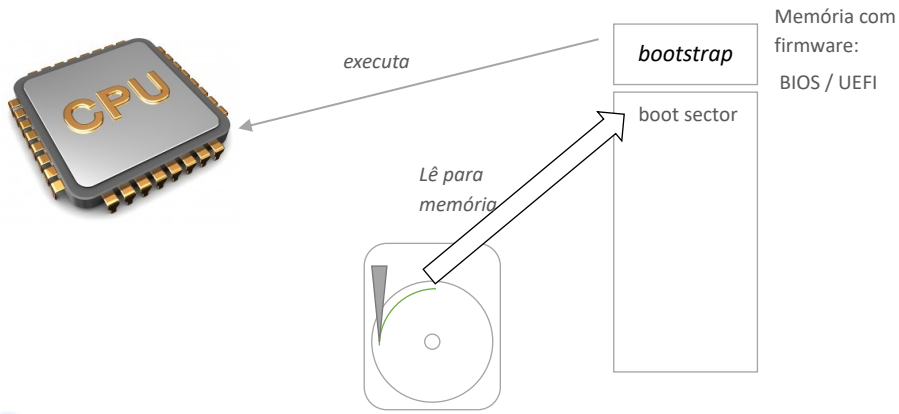
- Como iniciar a execução do SO?
  - No inicio: era mandado carregar "à mão" via consola
  - Agora: rotina de *Boot* numa memória ROM (p.ex.)



3

Arranque do SO

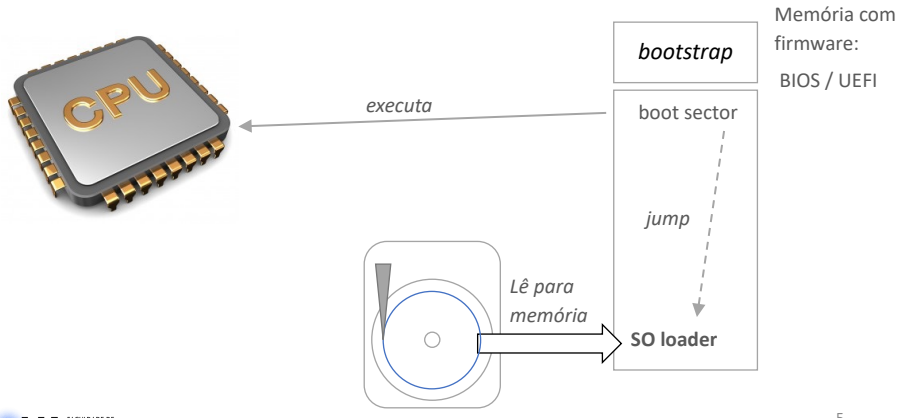
- Carrega em memória o primeiro sector, de um dispositivo identificado como de boot



4

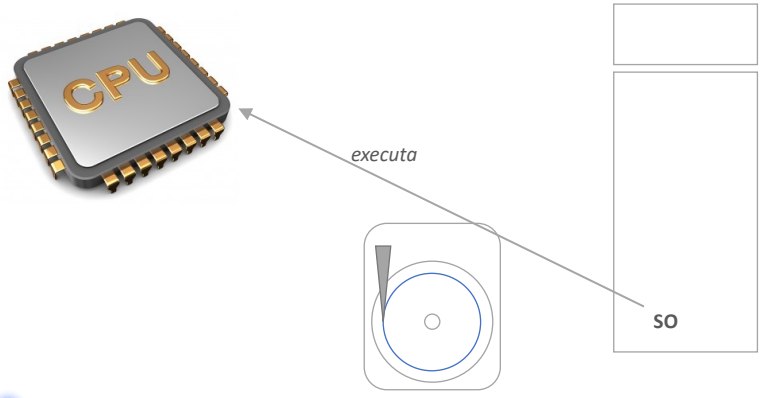
Arranque do SO

- Inicia a execução desse código, responsável por carregar para memória o SO, ou o loader do SO
  - Depois começa a executar esse código



Arranque do SO

- Mais código será carregado do disco, todas as inicializações feitas, até o SO estar completo e criar o primeiro processo,...



Principais características do UNIX (~1970)

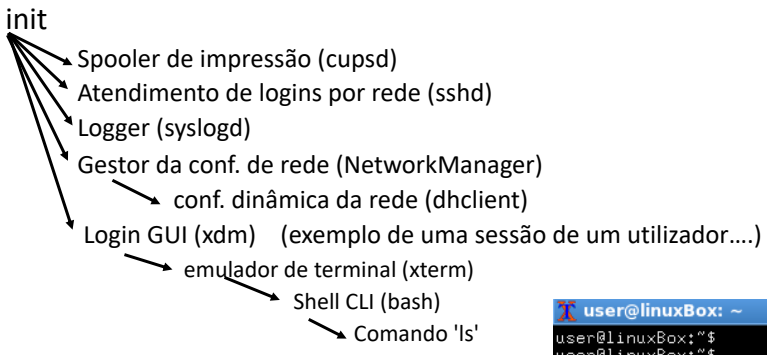
- Suporta múltiplos processos
- Suporta múltiplos utilizadores
- Escalonamento com *time-slice*
- Organiza o espaço nos discos num sistema de ficheiros hierárquico
- Possui mecanismos de permissões associado aos processos e aos ficheiros

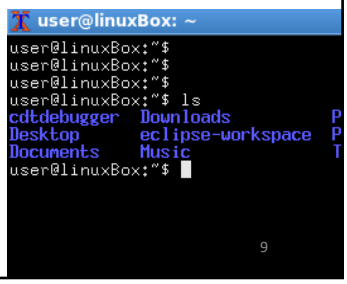
Em resumo, o que hoje todos esperamos de um SO!

Utilizadores e permissões

- Utilizadores: identificação, direitos ou permissões
  - O que pode ler, escrever, executar, que tempo de CPU, espaço em memória, espaço em disco, etc…
- Cada utilizador tem de identificar-se perante o SO: ***login***
- Cada processo, em princípio, recebe os direitos do utilizador que o mandou executar

Árvore de processos (exemplo)





9

Lista de processos (Unix)

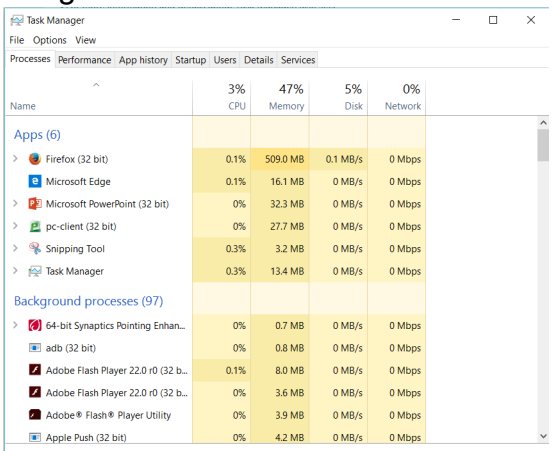
- Visão dada pelo comando `ps` (*process status*)
  Exemplo: `ps -ef`

UID	PID	PPID	C	STIME	TTY	TIME	CMD
root	1	0	0	Feb21	?	0:03	init [3]
cups	513	1	0	Feb21	?	5:03	cupsd
root	519	1	0	Feb21	?	0:16	/usr/sbin/sshd
root	660	1	0	Feb21	tty1	0:00	/sbin/getty tty1
root	661	1	0	Feb21	tty2	0:00	/sbin/getty tty2
vad	11625	519	0	17:54	?	0:00	/usr/sbin/sshd
vad	11633	11625	0	17:54	pts/1	0:00	-bash
vad	11663	11633	0	17:55	pts/1	0:00	ps -ef

10

Lista de processos (MS-Windows)

- Exemplo de uma visão data pelo MS-Windows Task Manager

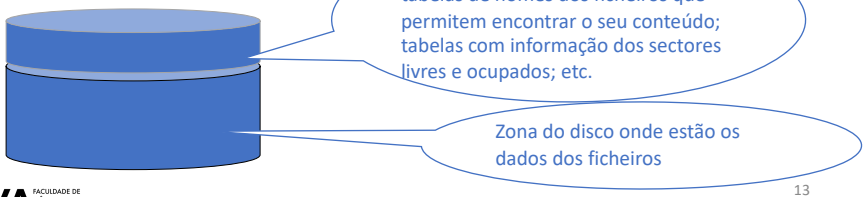


SO como uma máquina virtual

- O SO oferece a cada processo uma máquina (virtual) com:
  - CPU: para executar instruções
  - Memória: para guardar código, dados e pilha (programa)
  - Formas de comunicar (I/O):
    - Acesso aos periféricos
    - Acesso a armazenamento persistente
    - Comunicar com outros processos
- O SO oferece uma interface de programação (API) de mais alto nível do que o *hardware*.
  - Abstrai o hardware, escondendo dos programas os detalhes e variedades dos dispositivos

SO como gestor de recursos

- O SO tem de garantir a partilha justa e segura dos recursos físicos entre os processos concorrentes.
- Criar novos recursos que facilitam a programação
  - p.e. mecanismo de comunicação entre processos
- Exemplo: Armazenamento na forma de ficheiros, que só o SO pode manipular
  - Mais fácil de usar
  - Sem SO seria impossível gerir o espaço livre/ocupado, impedir a destruição das zonas do disco com metadados , ou impor permissões (p.e. aceder a ficheiros não permitidos)



13

Caraterísticas desejáveis

- Facilidade de utilização e portabilidade
  - na utilização dos recursos (hardware)
  - acesso a novos serviços do SO (que facilitam a programação)
  - adaptação a novo hardware
- Eficiência
  - utilizador – melhores tempos de resposta
  - sistema – melhor utilização de recursos e débito de trabalhos
  - serviços para avaliação e reconfiguração
- Fiabilidade
  - limitar falhas/erros nos programa e suas consequências
  - limitar falhas/erros no hardware e suas consequências
  - serviços de recuperação

14

Duas Abstrações Suportadas pelo SO

- Processo
  - Representa e permite controlar uma instância de um programa em execução. Oferece uma máquina virtual ao programa e esconde os detalhes da partilha e gestão dos recursos necessários à execução
  - Operações: criar, destruir, ...
    - Interface Unix/Posix: **`fork`**, **`exec`**, **`kill`**, ...
- Ficheiro
  - Representar uma sequência de bytes, normalmente persistente, sob um nome. Acedidos via canais (ou *streams*). Esconde os detalhes de manipulação e gestão/partilha de periféricos (p.e. discos)
  - Operações: criar, destruir, pedir acesso (canal), ler, escrever, ...
    - Interface Unix/Posix: **`creat`**, **`unlink`**, **`open`**, **`close`**, **`read`**, **`write`**, ...

Atributos dos Processos

- Para gerir os processos, o SO tem de manter uma tabela de descritores desses processos. Cada descritor inclui:
  - Identificador (*process identifier* - *PID*)
  - Identificador do utilizador (UID)
  - Localização da imagem do programa em memória
    - Código, dados e pilha (e.g. tabela de páginas)
  - Estado do CPU (quando não executa)
    - Conteúdo dos registos
  - Estado do I/O
    - Tabela de canais de I/O: ficheiros abertos e posição em que vai a leitura / escrita
- O SO encarrega-se de gerir o espaço de memória em que os processos residem e comutar o CPU entre estes.

Atributos dos Ficheiros

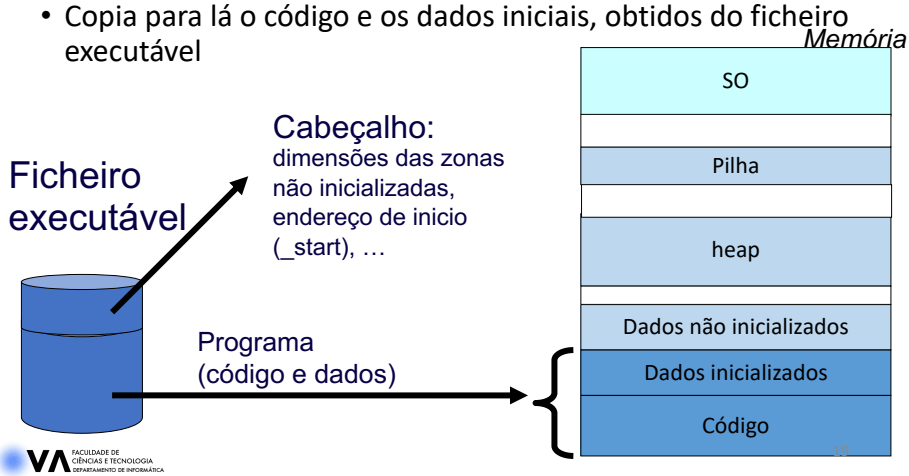
- Um ficheiro é uma sequência, logicamente, contígua de dados (bytes), acessível através de um identificador único, o **nome**.
  - Uma tabela de nomes é vulgarmente chamada **diretoria** ou **diretório**
- O nome identifica um conjunto de atributos (ou meta-dados). Por exemplo:
  - UID do dono do ficheiro
  - Modos de acesso permitidos (quem pode ler, escrever, etc.)
  - Identificação do disco e dos blocos que contém os dados deste ficheiro
  - Um ficheiro pode conter quaisquer dados (texto, imagem, som, etc.) ou programas. Para o SO são só bytes!
- O SO encarrega-se de gerir os discos em que estes residem. A organização do disco é escondida dos programadores e utilizadores.

Criação de Processos

- O sistema operativo cria um descritor, com atributos para definir os recursos virtuais e atribuir os recursos físicos
  - Define PID e guarda UID do utilizador
  - Define o mapa de memória (endereços virtuais e mapeamento na memória disponível); iniciado com a imagem do programa e seu estado inicial
  - Define canais de entrada/saída iniciais
    - Normalmente começa com alguns para interagir com o utilizador: Standard IN, OUT e ERROR
  - Define o estado inicial do CPU para começar a executar o programa
    - Este é carregado no CPU real quando for para começar a executar o programa

Carregamento da imagem

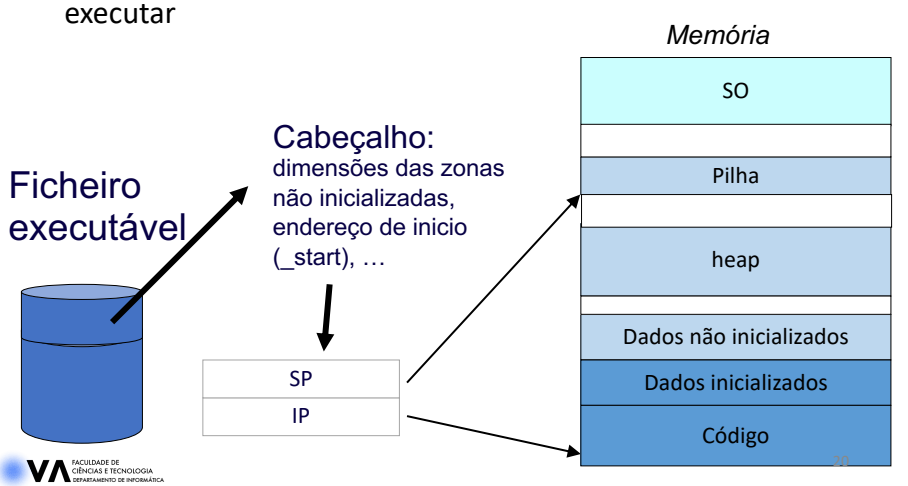
- O SO atribui memória ao processo de entre a livre. Define o endereçamento virtual do processo (p.e. tabela de páginas)
- Copia para lá o código e os dados iniciais, obtidos do ficheiro executável



19

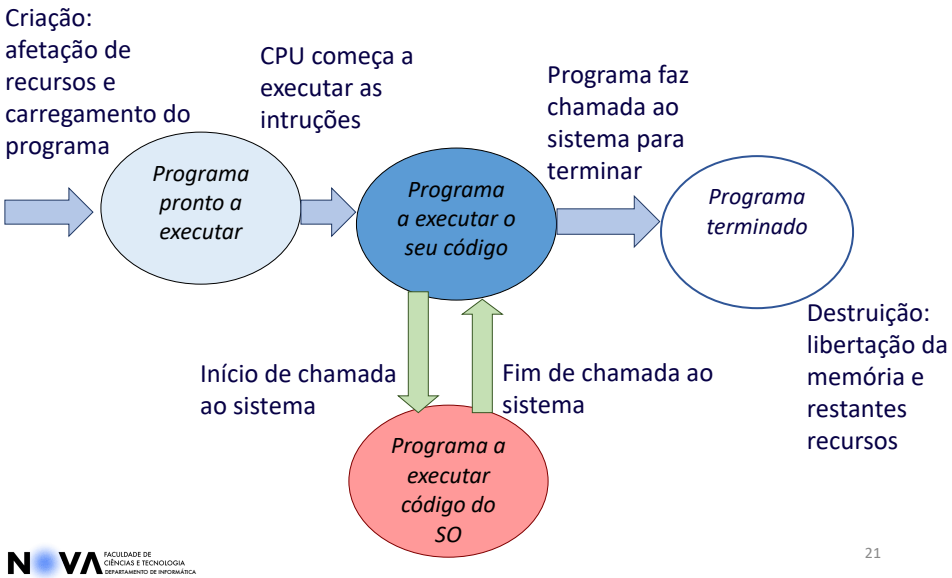
Inicio do estado do CPU

- O estado do CPU inclui os valores iniciais de IP e SP por forma a que o programa, acabado de carregar, possa executar



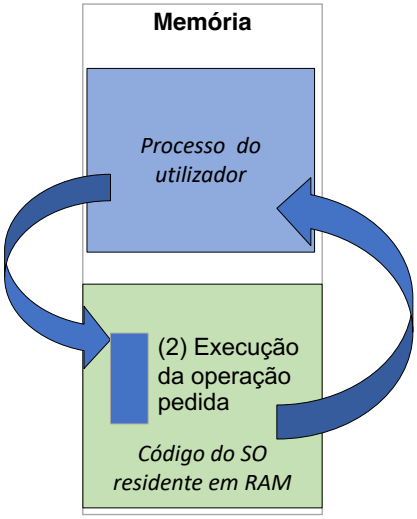
20

Vida de um processo (simplificado)



21

Pedidos ao SO



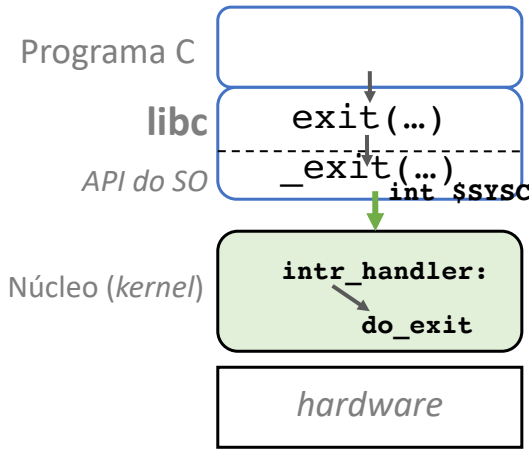
(1) Invocação dos serviços do SO através de uma instrução especial; guarda o valor corrente do IP (ou PC), troca para modo supervisor; o SO pode ainda guardar o restante estado do processo

(3) O estado do processo é reposto; instrução de retorno muda para modo utilizador e coloca no IP (ou PC) o valor guardado em (1)

22

Chamada ao sistema `_exit`

```
void _exit(int status);
```



Exemplo:

```
_exit: …
  mov $EXIT_SCALL,%eax
  mov 4(%ebp), %ebx
  int $SYSCALL
  …
```

No caso do linux/x86:
EXIT_SCALL=1
SYSCALL=0x80

(linux/64bits usa instrução syscall)

Pilha de software – exemplos IO

- Java - Exemplos de classes: InputStream, FileInputStream, FileOutputStream, PrintStream,...
  - read(), write(), close(), println(), printf(), ...
- C - Exemplo sobre o tipo FILE:
  - fopen, fread(), fwrite(), fclose(), fprintf(), printf(), fscanf(), scanf(),...
- Todas têm de se basear no SO:
- Exemplo Unix/Posix: open(), close(), read(), write()

Helloworld.c

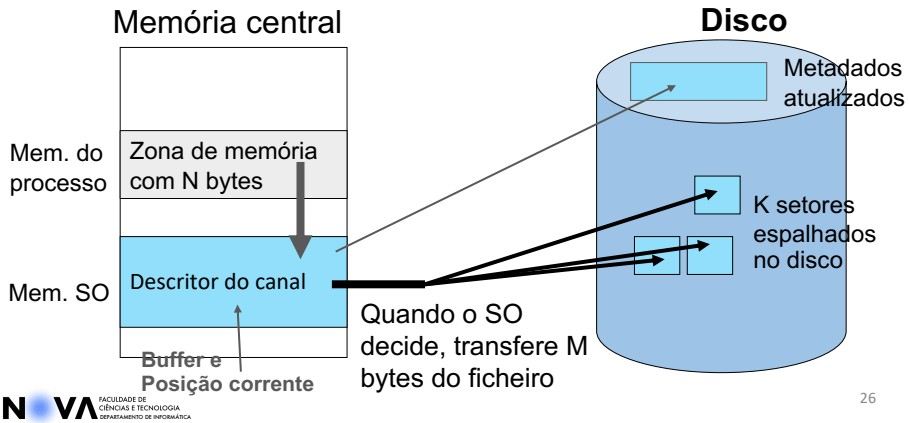
```
#include <stdio.h>
int main( int argc, char*argv[]) {
   printf("Hello world!\n");
   return 0;
}
```

Dentro da libc:

```
printf(...)
     fprintf( stdout, ... )
          write( 1, ... )   // interface
               int 0x80     // com SO
```

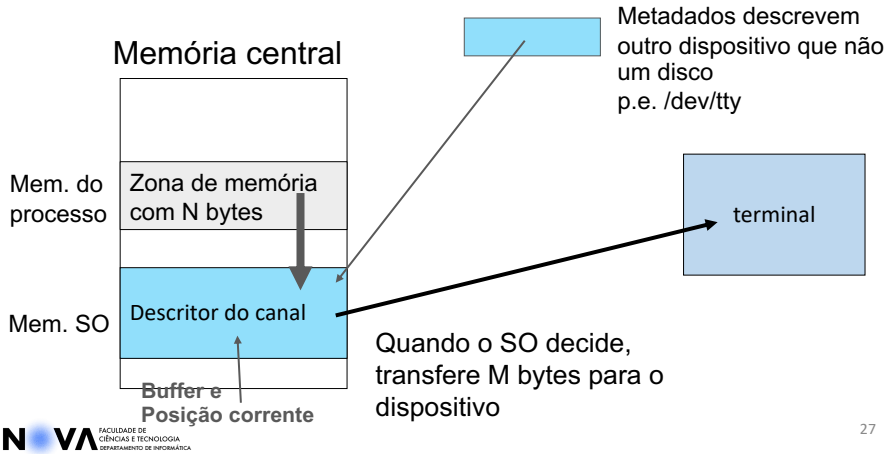

Escrita no ficheiro

- A visão do processo é de sequência contígua de bytes
- A realidade é mais complicada



Escrita no "ficheiro"

- As noções de canal e ficheiro escondem outros dispositivos
- Para o processo é o mesmo...



27

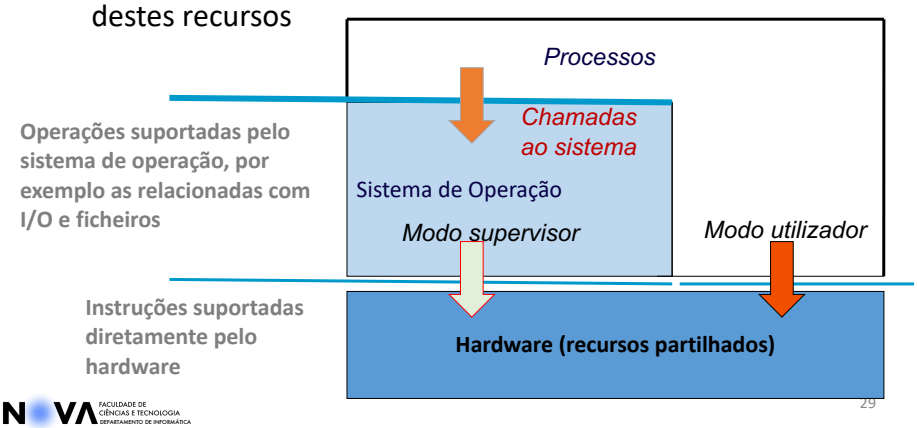
Mecanismos e Políticas

- **Como fazer**: O SO oferece mecanismos: funcionalidades e protocolos que permitem as operações básicas
  - Baseados em *software* e em *hardware*
  - Exemplos: criar processo, criar canal de IO, escrever bytes para canal, trocar CPU de contexto entre processos, mapear endereços virtuais em reais,...
- **O que fazer ou quando**: As decisões de gestão e objectivos do SO depende das políticas implementadas pelos algoritmos no SO
  - Usam os mecanismos
  - Exemplo: onde mapear um novo processo, decidir quando escrever dados nos canais para os discos, quando trocar de processo, que processo executar a seguir (escalonamento), etc...
- Modularidade: independência entre estes dois níveis

28

Execução direta limitada

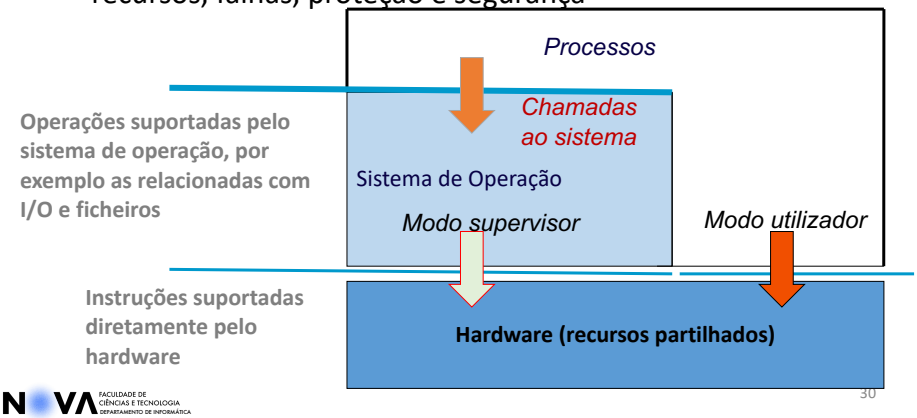
- Só o SO tem acesso a alguns recursos (em modo supervisor)
- Esses recursos só podem ser usados pedindo ao SO
- O hardware ajuda o SO a garantir a proteção e a partilha destes recursos



29

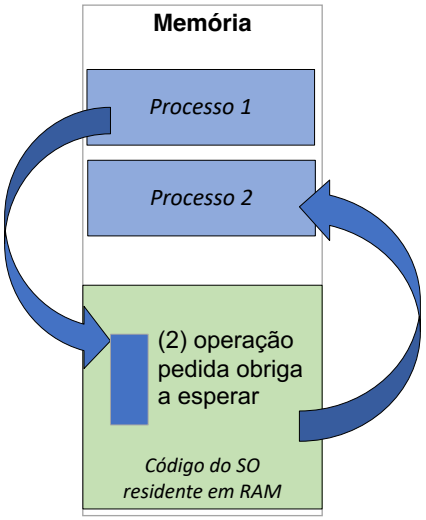
SO em execução

- O SO estende a máquina física com operações de mais alto nível e esconde a especificidade da máquina real
- Os SOs controlam a execução de programas, a gestão de recursos, falhas, proteção e segurança



30

Comutação entre processos

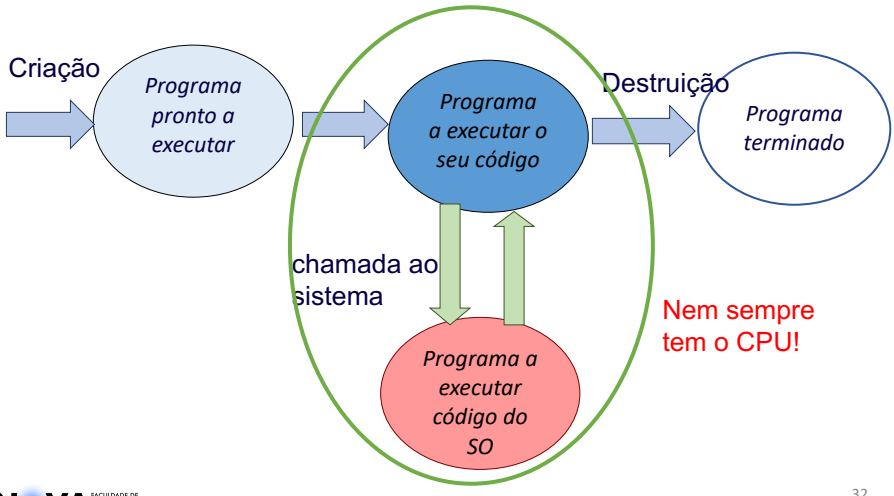


(1) Invocação dos serviços do SO através de uma instrução especial;

(3) Se processo 1 tem de aguardar, então, guarda contexto do processo 1 no seu descritor; Muda para descritor do processo 2; Retoma o contexto do processo 2

31

Vida de um processo (visão anterior)

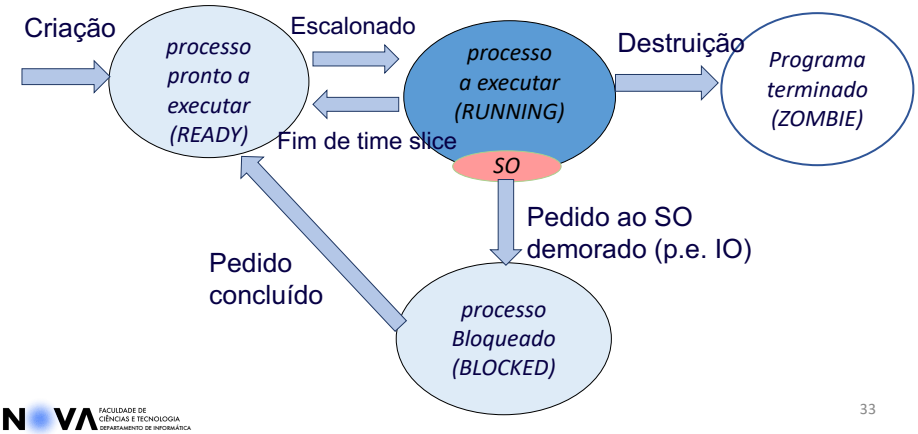


Nem sempre tem o CPU!

32

Estados de um processo

- A repartição do CPU é feita tirando partido do facto de os periféricos serem muito lentos e os processos terem de pedir ao SO as transferências



33

Interacção CPU-Periféricos

- Os periféricos/dispositivos e o CPU executam em paralelo
- Cada controlador supervisiona um tipo de periférico.
- Cada controlador tem um "buffer" e/ou possibilidade de DMA
- O CPU pede transferência de dados usando DMA entre memória e o buffer nos controladores
- O controlador informa o CPU quando acaba a transferência através de uma interrupção.
  - interrompe o processo corrente e SO executa

34

O SO é "interrupt-driven"

- Quando ocorre uma interrupção (trap/excepção) o controlo é transferido para uma rotina de serviço no SO
  - Guarda contexto mínimo, muda para modo supervisor
- Dependendo da interrupção:
  - Atende uma chamada ao SO
  - Trata uma excepção (p.e. erro)
  - Trata IO, desencadeando nova transferência / assinala o processo respetivo pronto a continuar
  - Trata fim de *time-slice* chamando o escalonador
- execução continua no ponto onde se encontrava, o processo corrente, ou no novo processo escolhido