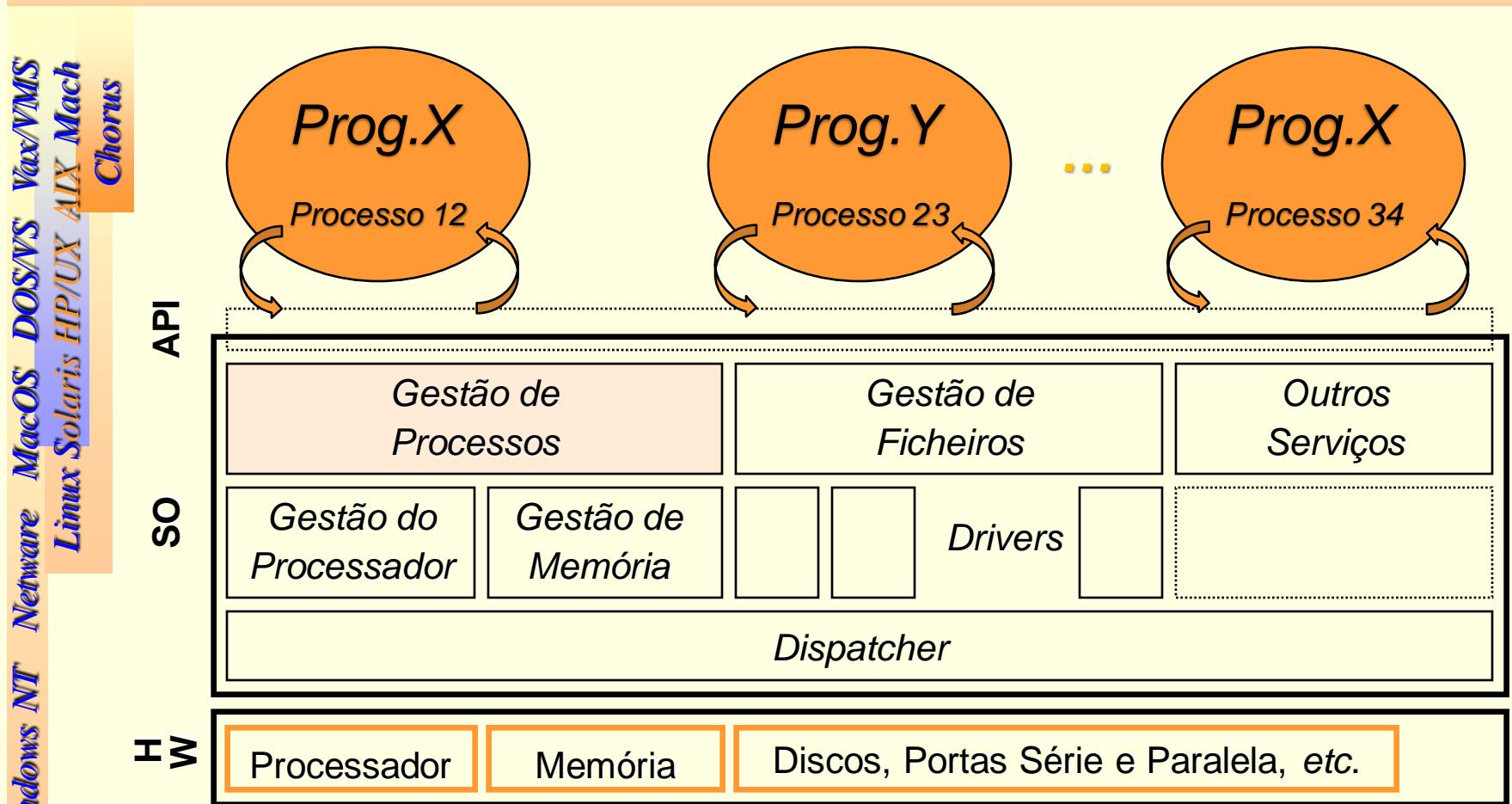


Fundamentos de Sistemas de Operação

Unix Windows NT Netware Mac OS DOS/V/VS Vax/VMS
Linux Solaris HP/UX AIX Mach Chorus

Sistema de Operação:
Processos (introdução)

O SO e os processos



SO: Gestão de processos

□ Gestão de processos

- **Um processo é um programa em execução.** Um processo necessita de recursos para efectuar a computação que lhe está associada:
 - Tempo de CPU; Memória (RAM)
 - Entradas/Saídas: ficheiros, periféricos e mecanismos para comunicação com outros processos
- Um processo é, **por princípio**, isolado dos outros
 - A sua memória é privada (“ninguém” lá pode entrar)
 - Os dados guardados nos registos do CPU não são partilhados com outros processos
 - Por princípio, aquilo que um processo escreve nos “seus” ficheiros é privado – no entanto o utilizador pode dar permissão a outros (utilizadores e seus processos) para acederem a esses ficheiros

SO: Gestão de processos

□ Gestão de processos

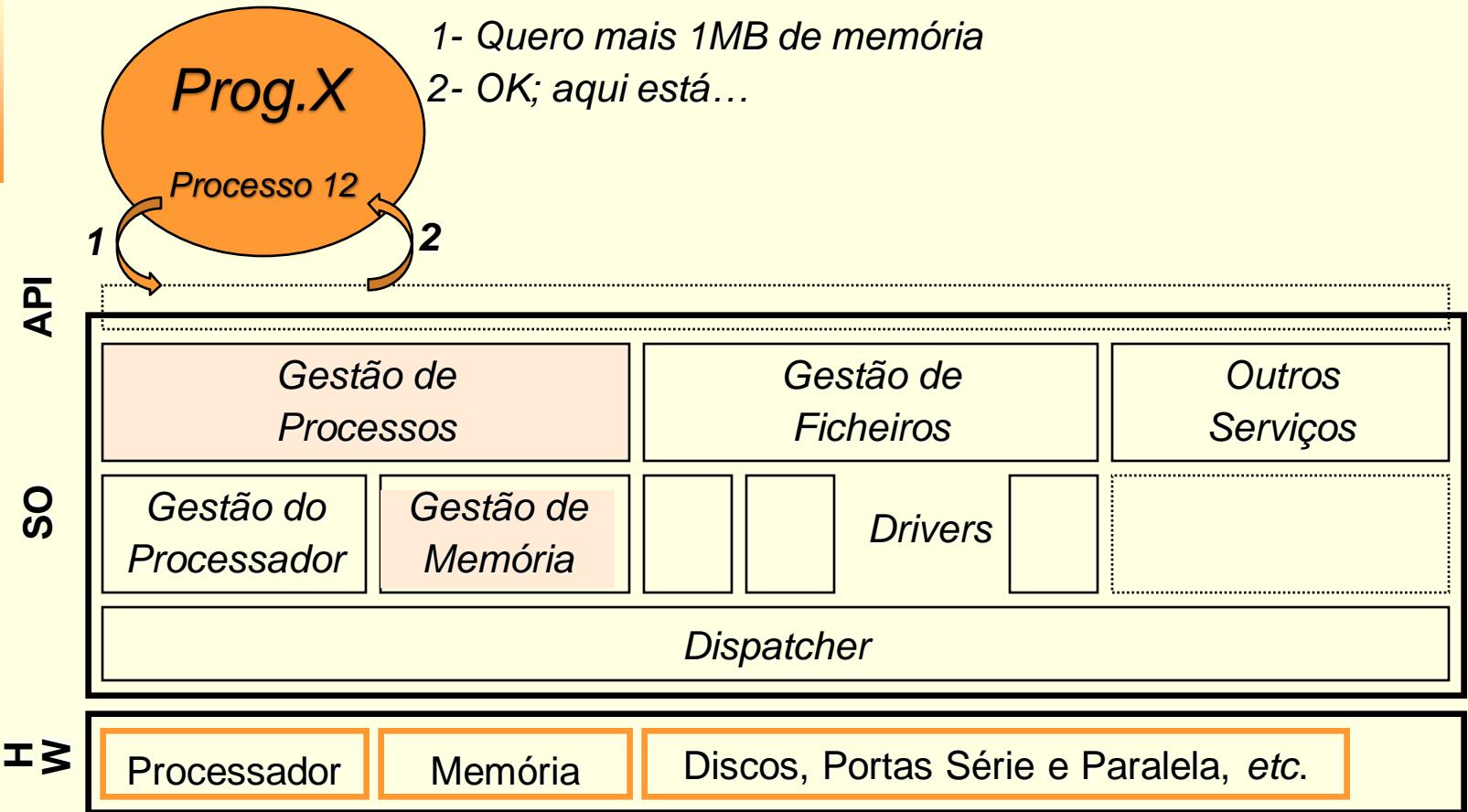
- O SO tem de:
 - Permitir Criar e Destruir processos
 - Executar um outro programa “no interior” do processo (Unix/Linux) ou criar um processo indicando qual o programa a executar (Windows)
 - Permitir Suspender (dando a vez a outro) e Retomar processos
 - Oferecer mecanismos que possibilitem a:
 - Sincronização de processos
 - Comunicação entre processos
- O SO tem de:
 - Permitir a um processo pedir-lhe serviços (API dos processos), e
 - Satisfazer ou recusar (dando erro) o pedido

Fundamentos de Sistemas de Operação

Unix Windows NT Netware Mac OS DOS/V/S Vax/VMS
Linux Solaris HP/UX AIX Mach Chorus

Sistema de Operação:
Gestão de Memória (Parte I)

O SO: memória para os processos (1)



O SO: memória para os processos (2)

VMS
Vax/VMS

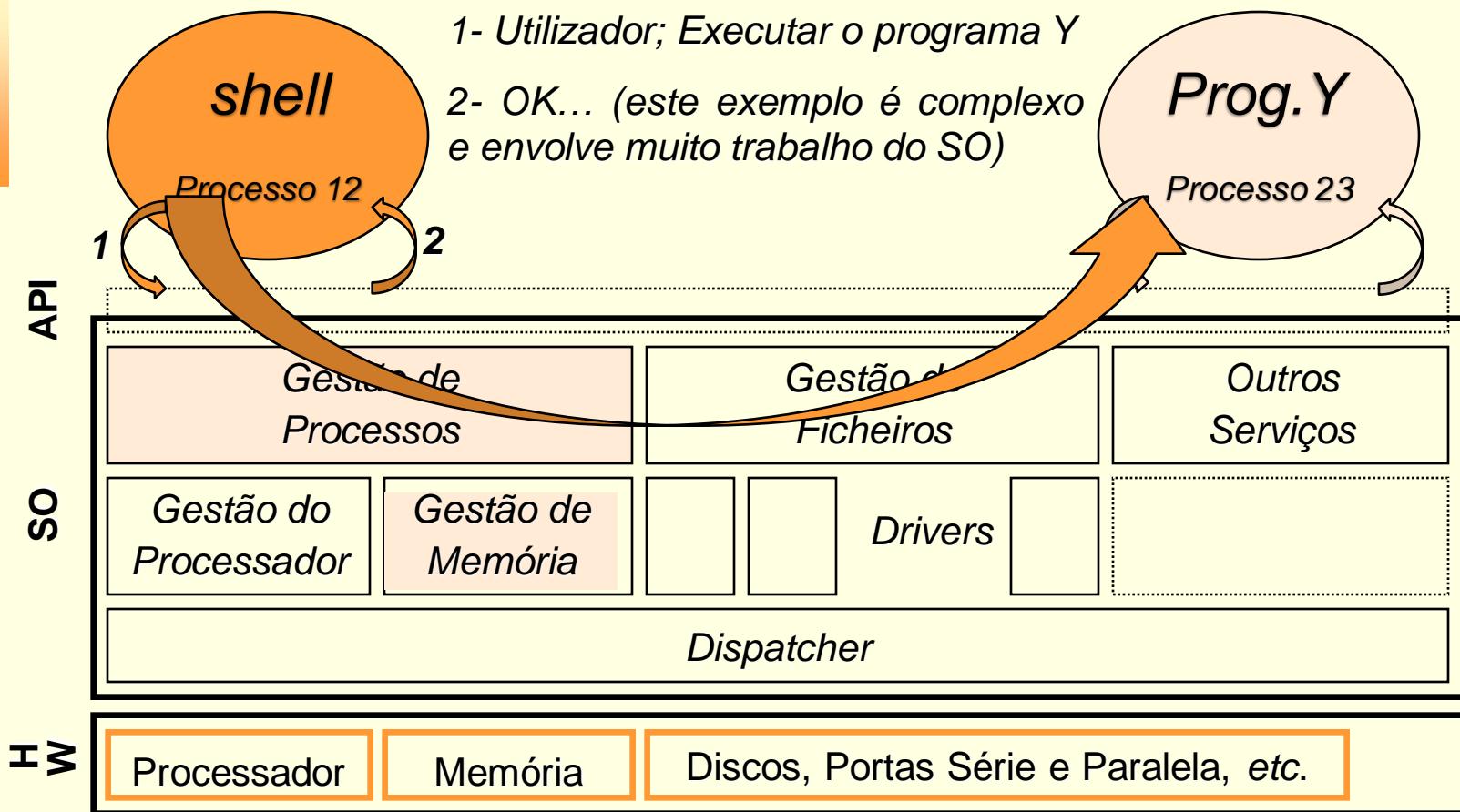
DOS/VS
DOS/VS

MacOS
MacOS

Netware
Netware

Linux Solaris HP/UX AIX Mach
Linux Solaris HP/UX AIX Mach

Chorus
Chorus



O SO: memória para os processos (3)

- Criar um novo processo:
 - Localizar o ficheiro com o programa (executável!)
 - Ler o início do ficheiro (cabeçalho) para obter as informações sobre a quantidade de memória necessária
 - Obter o espaço na RAM
 - Carregar nesse espaço o código máquina e os dados e reservar espaço para o stack e...
 - Preparar para executar a 1^a instrução do código... (a ver mais tarde)
- Tudo feito pelo SO!

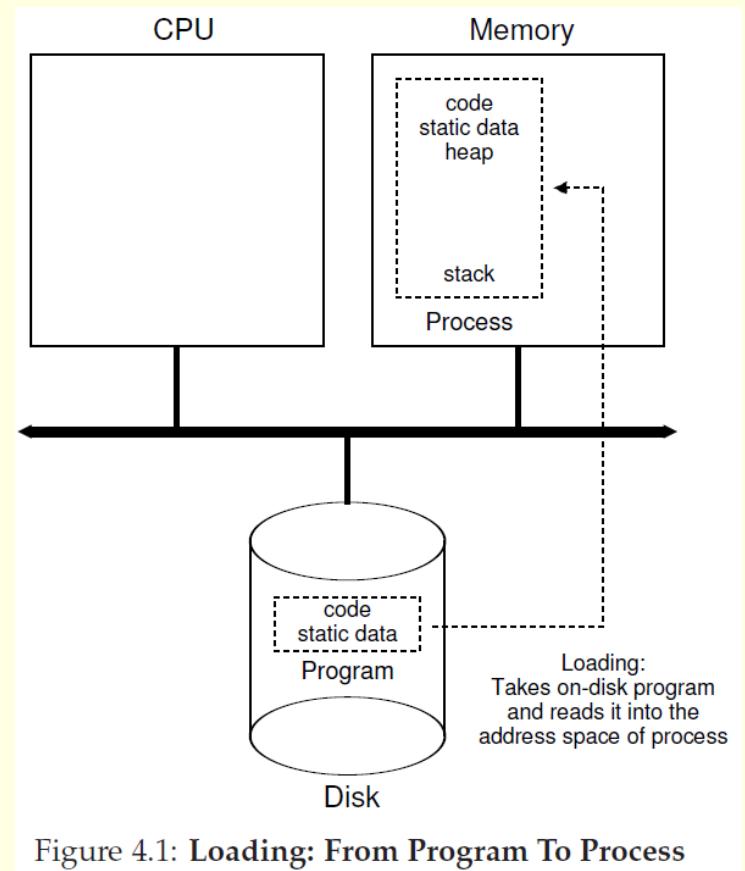


Figure 4.1: Loading: From Program To Process

SO: Gestão de Memória

□ Espaço de Endereçamento (1)

- **Definição:** o EE é o conjunto de posições de memória que (potencialmente) podem, ao longo da “vida” (execução) de um processo, ser acedidas.
- Cada “tipo” de SO define o mapa de EE que suporta
 - O mapa do Linux é diferente do Windows.
 - O mapa da versão Linux 32-bits é parecido com o da versão 64-bits, mas a **posição e dimensões** das áreas do mapa são **muitíssimo** diferentes (mas a versão 64 consegue executar programas 32, desde que... ☺)

SO: Gestão de Memória

- Espaço de Endereçamento (2)
 - Cada “tipo” de SO define ... (continuação)
 - Os utilitários de desenvolvimento (compiladores, assemblers, linkers, etc.) têm de produzir um mapa que obedeça à especificação do SO onde vai ser corrido o programa.
- Mapas de EE modernos
 - Espaço linear
 - Contíguo – não está partido em regiões diferentes (segmentos), acedidas via diferentes registos de endereçamento (registos de segmento) – mas é um único espaço “logicamente” dividido em zonas distintas: código, dados, stack, ...

SO: Gestão de Memória

□ Espaço de Endereçamento:

- Demo Linux 32 e 64-bits (“intro” ao Lab-02 ☺)

```
... includes ...

int main(int argc, char *argv[]) {

    printf("location of code : %p (hex) %u (dec)\n", main, (unsigned int)main);

    char *ptr = malloc(100e6);
    printf("location of heap : %p (hex) %u (dec)\n", ptr, (unsigned int)ptr);

    int x = 3;
    printf("location of stack: %p (hex) %u (dec)\n", &x, (unsigned int)&x);

    return 0;
}
```

SO: Gestão de Memória

□ Espaço de Endereçamento

- Linux 32-bits

```
fso@fso-32VM:~/FSO$ ./a.out
location of code : 0x804849b (hex) 134513819 (dec)
location of heap : 0xb1dc3008 (hex) 2983997448 (dec)
location of stack: 0xbfc978e4 (hex) 3217651940 (dec)
```

- Linux 64-bits

```
fso@fso-64VM:~/FSO$ ./a.out
location of code : 0x4005d6 (hex) 4195798 (dec)
location of heap : 0x7fce3f098010 (hex) 140523797577744 (dec)
location of stack: 0x7ffeb039acec (hex) 140731854990572 (dec)
```

SO: Gestão de Memória

□ Espaço de Endereçamento: para pensar...

$$2^{64} = 18E = 18446744073709551616$$

? ->
140731854990572 (~128T) ->
(top of stack)

$$2^{32} = 4G = 4294967296$$



<- ?
<- 3217651940
(top of stack)

$$2^{48} = 256T \rightarrow 2^{47} = 128T$$

Contrariamente a muita “sabedoria enlatada da Internet”, as arquitecturas Intel/AMD de 64-bits actuais não usam (ainda) 64, mas apenas 48 bits, de endereçamento, em que metade (47 bits) é para o espaço dos processos e a outra metade para o SO...

SO: Gestão de Memória

□ Ficheiros executáveis

- Linux 32-bits

```
fso@fso-32VM:~/FSO$ ls -l a.out
-rwxrwxr-x 1 fso fso 7432 Ago 23 17:02 a.out
```

- Linux 64-bits

```
fso@fso-64VM:~/FSO$ ls -l a.out
-rwxrwxr-x 1 fso fso 8712 Ago 23 11:27 a.out
```

□ Questões:

- Como é que ficheiros tão pequenos (~ 8KB) dão origem a EE tão grandes? (~ 3GB em 32-bits e 128TB!!! em 64-bits)
- 128TB num laptop com 32GB de RAM?! (é... o meu tem ☺; mas isso até nem interessa muito porque o “computador” é a VM e tem 2GB)

SO: Gestão de Memória

□ Espreitando um executável-32 (feito em AC ☺)

```
fso@fso-32VM:~/FSO$ objdump -h a.out
a.out:      file format elf32-i386

Sections:
Idx Name      Size    VMA          LMA          File off  Algn
 0 .interp    00000013  08048154  08048154  00000154  2**0
              CONTENTS, ALLOC, LOAD, READONLY, DATA
...
 13 .text     00000212  080483a0  080483a0  000003a0  2**4
              CONTENTS, ALLOC, LOAD, READONLY, CODE
 14 .fini    00000014  080485b4  080485b4  000005b4  2**2
              CONTENTS, ALLOC, LOAD, READONLY, CODE
 15 .rodata   0000007e  080485c8  080485c8  000005c8  2**2
              CONTENTS, ALLOC, LOAD, READONLY, DATA
...
 24 .data     00000008  0804a01c  0804a01c  0000101c  2**2
              CONTENTS, ALLOC, LOAD, DATA
 25 .bss      00000004  0804a024  0804a024  00001024  2**0
              ALLOC
```

SO: Gestão de Memória

□ Regiões de Memória – interpretando o **objdump**

- Nome e dimensão (bytes) da região (ex: text é código)
- ALLOC: SO tem de alocar espaço em memória RAM
- LOAD: SO tem de carregar bytes do ficheiro para RAM
- READONLY: proteger; se o processo tentar escrever nessa zona... bummm! (segmentation fault)
- CODE: código executável do programa (processo)
- DATA: dados do processo
- Nota: `.rodata` tem, p. ex., as strings constantes. `.data`, tem as variáveis globais inicializadas, e `.bss` as não-inicializadas

SO: Gestão de Memória

□ Gestão de Memória

- A memória central é “repartida” pelos processos, pelo SO e pelos periféricos.
- O SO é responsável por:
 - Manter informação sobre que partes da memória estão livres e quais estão ocupadas, e por quem.
 - Isolar e proteger o EE dos processos e o espaço do próprio SO de forma a que:
 - um processo não possa aceder ao EE de outros ...
 - ... nem ao espaço do SO
 - Atribuir (“alocar”) e libertar memória de acordo com as necessidades dos processos (e outros... o próprio SO e periféricos).

SO: Gestão de Memória

□ Gestão de Memória

- O EE de um processo típico não está integralmente implementado na RAM, digamos que tem “faltas” (é esparso).
 - Quando a execução de um processo referencia um endereço que “cai numa zona em falta”, o SO intervém: se a operação é permitida (p.ex., o stack precisa de crescer) mais RAM é alocada; senão, **bummm!** segmentation fault ☹

Esta e outras questões serão estudadas (bastante ☺) mais adiante...

Fundamentos de Sistemas de Operação

Unix Windows NT Netware Mac OS DOS/V/S Vax/VMS
Linux Solaris HP/UX AIX Mach Chorus

Gestão de Memória:
Fim da Parte I