Arquitetura de Computadores

MIEI – 2021/22

DI-FCT/UNL

Mecanismos Hardware Indispensáveis ao SO

Excecoes/Interrupções

Sumário

- Funções do SO
- Mecanismos Hardware Indispensáveis ao SO
- Exceções/Interrupções

Bibliografia:

Bryant, O'Hallaron Computer Systems: A Programmer's Perspective, 2nd ou 3rd Ed, secções 8.1, 8.2, 8.3

Funções do SO

- O sistema de operação (SO) é o componente do software de sistema que serve de suporte a todo software de aplicação e de sistema
- Os SOs controlam a execução de programas, a gestão de recursos, proteção e segurança
- O SO cria **uma máquina virtual com instruções de alto nível** que os programas podem usar fazendo chamadas ao sistema
- O **SO impede o acesso direto aos periféricos e ficheiros** pelos programas

Duas principais abstrações suportadas pelo SO

- **Ficheiro**
  - Sequência de bytes guardados em disco
    - A localização dos bytes é transparente para o programa que manipula o ficheiro
    - O ficheiro é conhecido pelo seu nome que é uma cadeia de caracteres

- **Processo**
  - Programa em execução
    - Executa uma computação que produz sempre os mesmos resultados independentemente de estarem em execução simultânea mais programas

Conceito de Ficheiro

- A informação é guardada de forma permanente.
- Um ficheiro é um espaço de endereçamento logicamente contíguo, contendo um conjunto de dados “inter-relacionados” e é acessível através de um **identificador único** (nome).
- Um ficheiro pode conter dados (texto, imagem, som, etc.) ou programas.
- O SO encarrega-se de gerir os ficheiros e os discos em que estes residem. A organização do disco é escondida aos programadores e utilizadores.

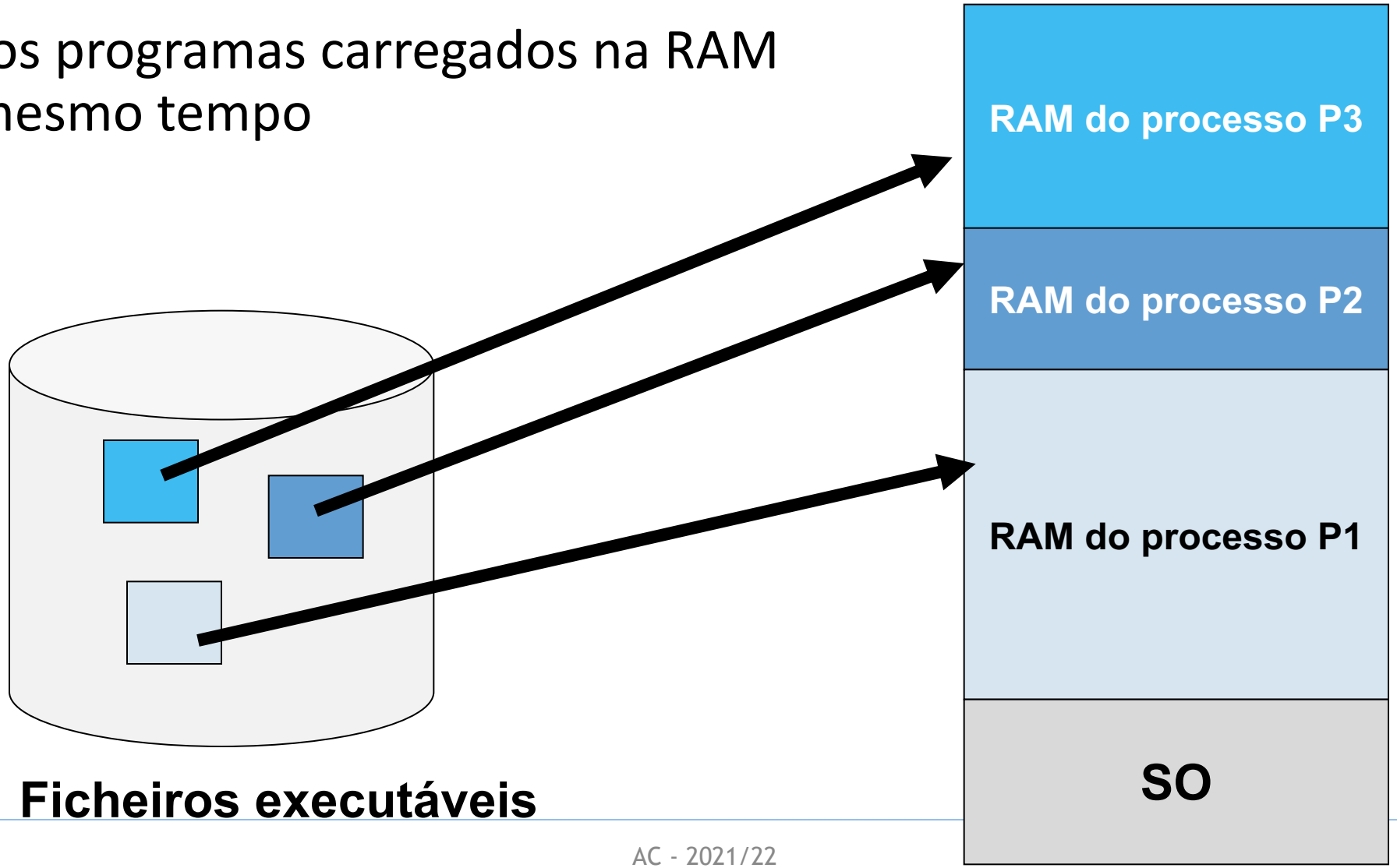
Conceito de Processo

- Um SO executa um conjunto de programas:
  - Portáteis e desktops: utilizadores têm vários programas carregados em memória
  - Servidores: múltiplos serviços numa máquina
    - o mesmo serviço pode ter vários clientes em simultâneo

- **Processo – um programa em execução;**
  - um processo executa um dado programa de forma independente de todos os outros programas em execução

Multiprogramação

- Vários programas carregados na RAM ao mesmo tempo



Processos em execução: Comando **top**

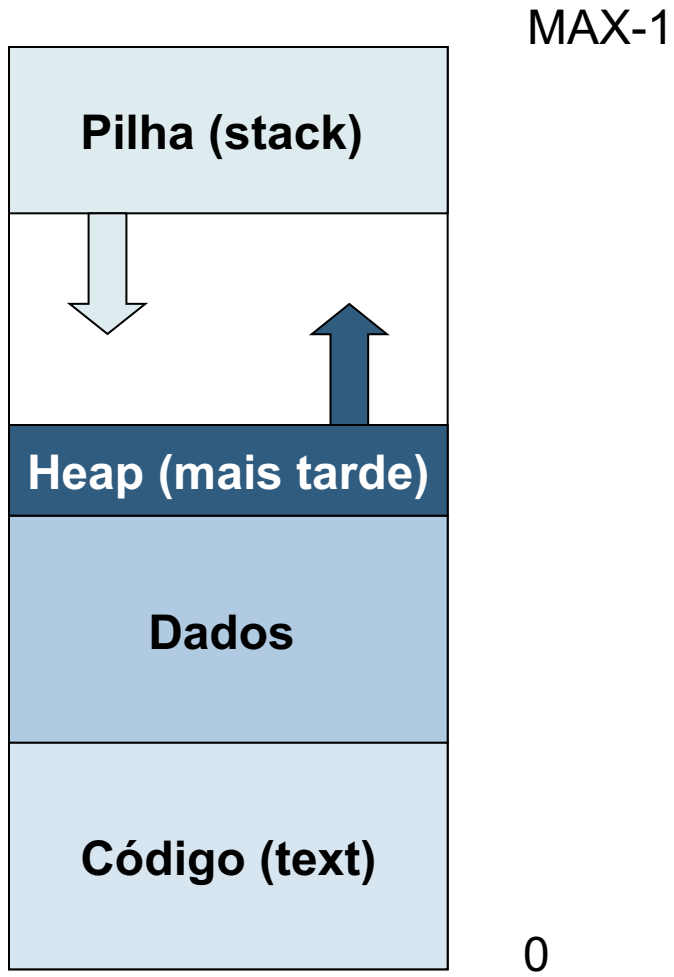
```
Processes: 434 total, 4 running, 1 stuck, 429 sleeping, 1888 threads                                                 09:46:59
Load Avg: 42.90, 17.76, 7.03  CPU usage: 44.85% user, 18.69% sys, 36.44% idle  SharedLibs: 395M resident, 82M data, 26M linkedit.
MemRegions: 52685 total, 2240M resident, 237M private, 1042M shared. PhysMem: 7993M used (1845M wired), 198M unused.
VM: 15T vsize, 3136M framework vsize, 0(0) swapins, 0(0) swapouts. Networks: packets: 25810/20M in, 25765/10M out.
Disks: 290648/4043M read, 49493/815M written.

PID  COMMAND      %CPU  TIME     #TH   #WQ   #PORT MEM    PURG   CMPRS  PGRP PPID STATE    BOOSTS        %CPU_ME %CPU_OTHRS UID  FAULTS   COW
317  mds_stores   100.0 00:08.90 25    23/11 190-  64M+   30M+   3408K- 317  1    sleeping *0[1]         0.00000 83.25298   0    85286+   185+
640  Mail         46.8  00:21.74 18/2  13/2  706+  93M-   2540K+ 17M-   640  1    running  *0[7]         0.46409 0.00000    501  70500+   1259
656  suggestd     33.7  00:06.50 10/2  9/2   223-  18M+   5028K- 5720K- 656  1    running  *3[17]        0.00000 33.62854   501  90135+   508
157  WindowServer 25.3  00:11.93 14    5     1391  182M-  6528K  8912K- 157  1    sleeping *0[1]         0.03878 0.11369    88   72839+   43695
111  mds          20.7  00:03.61 18    13    367+  23M+   0B     316K   111  1    sleeping *0[1]         105.097 15.37883   0    39700+   304
508  MRT          20.6  00:07.57 2     1     42    77M+   0B     47M    508  1    stuck     0[0]         0.00000 0.00000    501  177941+  272
210  mobileassetd 20.2  00:07.13 5     3/1   135   2696K- 0B     108K   210  1    sleeping *12[2]        0.03126 20.16799   0    40066+   251
0    kernel_task  15.2  00:15.36 209/9 0     0     330M-  0B     0B     0    0    running   0[0]         0.00000 0.00000    0    12845    2489
816  mdworker_sha 9.1   00:00.18 5     2     50+   1288K+ 0B     0B     816  1    sleeping *0[1]         0.02537 0.00000    501  9090+    166+
1    launchd      5.5   00:04.09 4     3     2048+ 16M    0B     344K-  1    0    sleeping  0[0]         0.00000 1.59018    0    12324+   577+
759  top          5.0   00:02.26 1/1   0     71+   7520K- 0B     1672K- 759  695  running  *0[1]         0.00000 0.00000    0    17685+   93
819  mdworker_sha 4.8   00:00.21 5     2     50    1304K- 0B     0B     819  1    sleeping *0[1]         0.03660 0.00000    501  10712+   166
589  Spotify Help 4.6   00:05.89 20    1     212   80M+   0B     1380K  536  536  sleeping *0[6]         0.00000 0.00000    501  76935+   3045
810  screencaptur 4.4   00:00.45 3     2     63    4388K  620K   0B     435  435  sleeping *0[311+]      0.10821 0.00000    501  19487+   819+
800  mdworker_sha 4.4   00:00.23 4     1     49    1308K- 0B     0B     800  1    sleeping *0[1]         1.68600 0.00000    501  11705+   166
786  mdworker_sha 4.4   00:00.38 4     1     57    1496K- 0B     160K-  786  1    sleeping *0[1]         0.22641 0.00000    501  20029+   183
156  cfprefsd     4.3   00:01.55 8     7     658+  1204K+ 0B     28K    156  1    sleeping *0[1503+]     0.00000 3.85629    0    2725+    84
814  mdworker_sha 4.3   00:00.22 5     2     50    1312K  0B     0B     814  1    sleeping *0[1]         0.67189 0.00000    501  10259+   166
794  mdworker_sha 3.6   00:00.34 4     1     49    1304K- 0B     28K-   794  1    sleeping *0[1]         0.33330 0.00000    501  15444+   166
799  mdworker_sha 3.4   00:00.35 4     1     49    1300K- 0B     0B     799  1    sleeping *0[1]         0.16003 0.00000    501  16373+   166
123  opendirector 3.2   00:02.13 11    10    1107+ 3580K+ 64K    12K    123  1    sleeping *0[1]         0.00000 1.50094    0    79869+   156
412  cfprefsd     3.0   00:01.62 7     6     423+  1156K- 88K    8192B  412  1    sleeping *0[1188+]     0.00000 2.69067    501  3743+    85
824  mdworker_sha 3.0   00:00.10 5     2     49+   1352K- 0B     0B     824  1    sleeping *0[1]         29.2844 0.00000    89   5885+    181+
342  com.apple.Ap 2.5   00:00.69 3     2     219   1528K  0B     12K    342  1    sleeping  0[1]         0.00000 0.00000    270  2469+    56
86   logd         2.1   00:01.41 5     4     1376+ 6348K+ 0B     1416K  86   1    sleeping *0[1]         0.00000 0.00000    0    9065+    120
805  AppleSpell   2.0   00:00.41 10    9     120-  9704K+ 204K   0B     805  1    sleeping *0[1]         37.9520 0.00000    501  9682+    339
149  notifyd      1.9   00:01.08 2     1     585+  1032K  0B     0B     149  1    sleeping *0[1]         0.00000 1.01732    0    1723+    69
692  Terminal     1.8   00:02.63 12    7     283   51M    12M    9100K  692  1    sleeping *0[10]        0.05677 0.00000    501  90726    656
96   configd      1.4   00:00.76 9     3     427+  2244K+ 0B     0B-    96   1    sleeping *0[1]         0.54281 0.00000    0    7990+    199
536  Spotify      1.3   00:05.59 44    2     481+  116M-  0B     2052K  536  1    sleeping *0[14]        0.52430 0.00000    501  63501+   3393
230  airportd     1.1   00:01.59 11    9     239+  3812K- 0B     732K-  230  1    sleeping *7+[9]        0.00000 0.53260    0    6880+    270
91   fseventsd    0.8   00:04.14 10    1     271   1856K  0B     8192B  91   1    sleeping *0[1]         0.00000 0.00000    0    105866+  106
822  mdworker_sha 0.7   00:00.10 5     2     49+   1392K+ 0B     0B     822  1    sleeping *0[1]         0.15107 0.00000    89   5951+    181+
114  diskarbitrat 0.7   00:00.23 2     1     244+  1108K+ 0B     0B     114  1    sleeping *0[1]         0.00000 0.38757    0    5499+    2104
796  com.apple.We 0.6   00:01.54 10    4     96-   84M-   56M    0B     796  1    sleeping *0[299+]      0.00000 0.00000    501  55204    8725
588  Spotify Help 0.6   00:00.79 9     1     86    24M+   0B     352K-  536  536  sleeping *0[2]         0.00000 0.00000    501  15546+   2597
623  com.apple.We 0.5   00:07.04 7     4     105   27M    540K   4716K  623  1    sleeping *50[1]        0.00000 0.00000    501  24949    262
425  trustd       0.5   00:02.69 3     2     180   3796K  660K+  36K    425  1    sleeping *4[287]       0.00000 0.52430    501  16124+   179
98   powerd       0.4   00:00.56 4     3     133   1652K+ 0B     0B     98   1    sleeping *0[1]         0.83943 0.03878    0    4203+    168
811  screencaptur 0.4   00:00.27 5     3     193-  13M-   84K+   0B     811  1    sleeping *0[96+]       0.00000 0.10821    501  7870     381
167  coreservices 0.4   00:00.24 4     1     280+  2724K+ 0B     152K   167  1    sleeping *0[1]         0.00000 0.28693    0    2258+    109
223  lsd          0.4   00:00.36 7     6     115+  1436K+ 0B     28K-   223  1    sleeping  0[74]        0.00000 0.32911    0    5122+    142
511  Spotlight    0.3   00:00.59 8     5     217+  11M+   0B     436K-  511  1    sleeping *0[5+]        0.12501 0.00000    501  11039+   484
619  com.apple.We 0.3   00:12.38 7     2     89    415M-  1288K  72M    619  1    sleeping *0[587]       0.00000 0.00000    501  170808   455
```

Ver também *task manager* do Windows (CTRL+ALT+DEL)

Processo e estado da computação

- O **estado** de uma computação inclui:
  - Os valores do *program counter* e outros registos do CPU
  - O conteúdo das várias zonas de memória usadas: dados e pilha (*stack*)
  - A interação com periféricos em curso



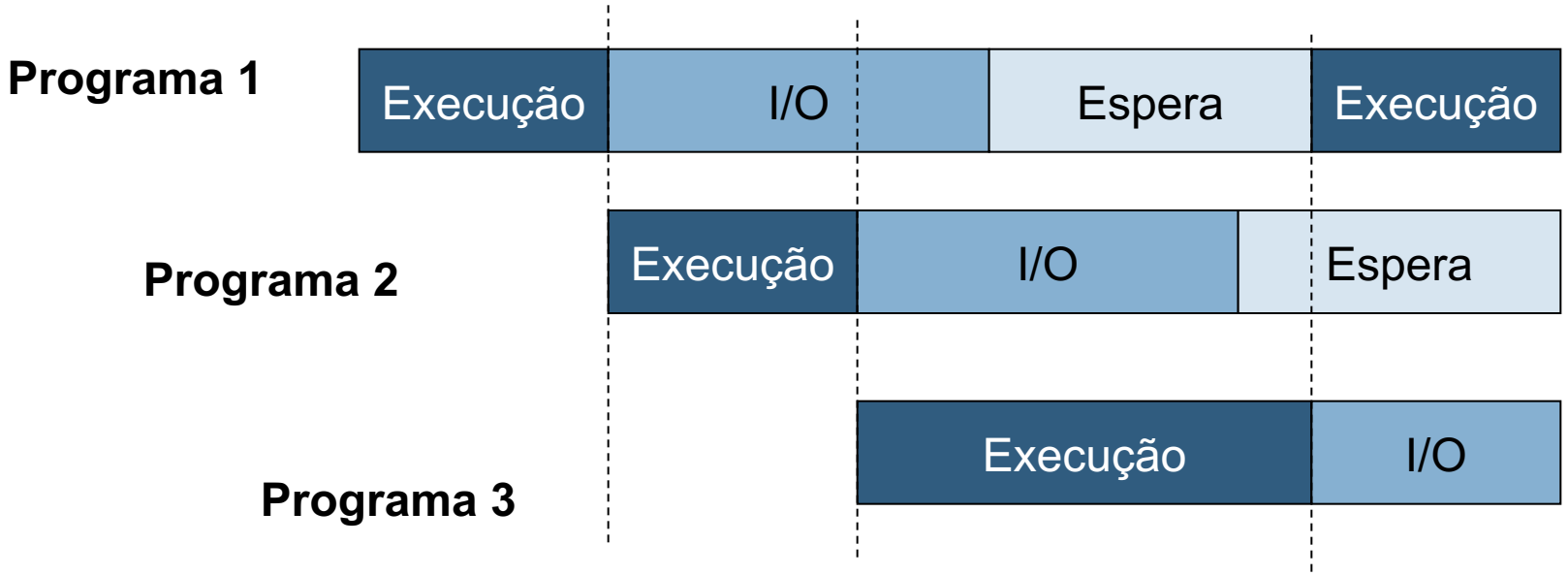
Suporte de multi-programação

- Orientado pelas operações de entrada/saída (E/S ou I/O)
  - Enquanto um processo espera que uma operação de I/O acabe, outro processo usa o CPU para executar instruções
- "**Time-slicing**" (fatias de tempo)
  - O CPU é atribuído rotativamente aos vários processos. A unidade de atribuição é uma fatia de tempo (por ex. 10 ms)

- Escalonador
  - Parte do sistema operativo
  - É ativado pelo início (ou fim) de uma operação de I/O ou pelo fim da fatia de tempo
  - Seleciona o próximo processo a correr

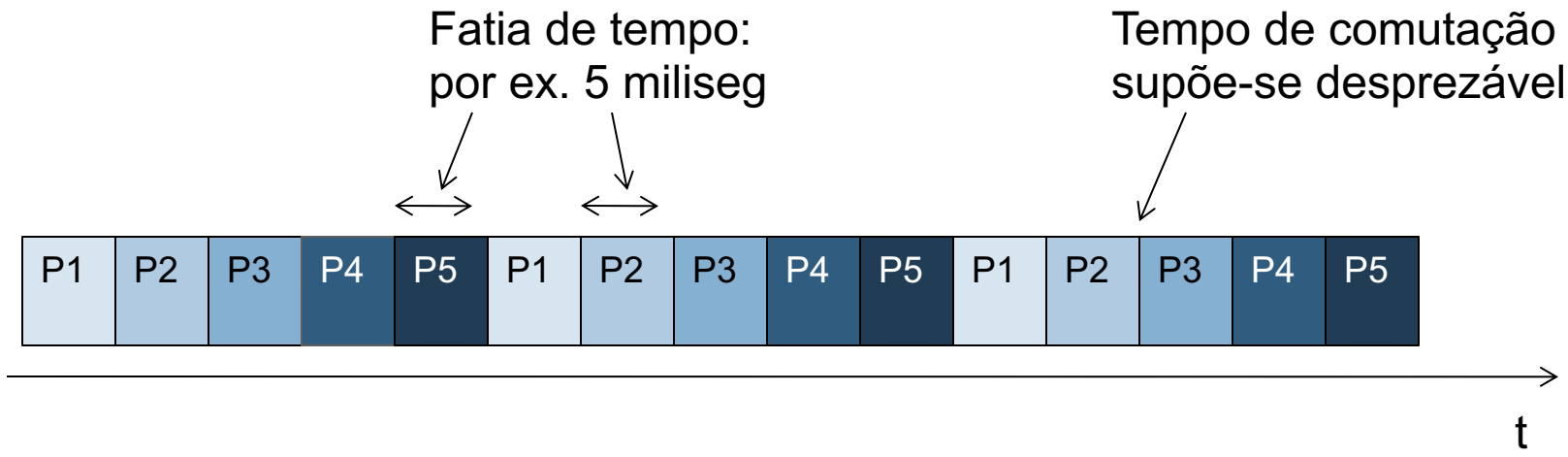
Partilha do CPU
(nos intervalos de processamento provocado por I/O)

- I/O representa uma grande percentagem do tempo de execução de um programa típico



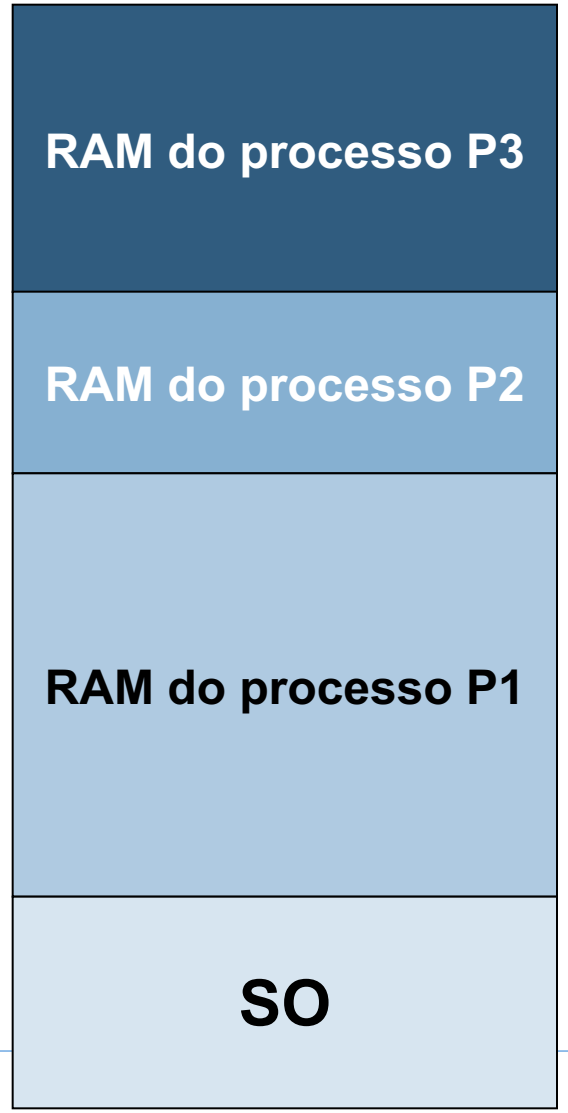
“Time-sharing” do CPU

- O CPU é atribuído aos processos de forma rotativa (round-robin)
  - Se os processos não se bloqueiam para fazer I/O, recebem a mesma percentagem do CPU



O SO assegura uma máquina virtual para cada processo

- Os resultados produzidos por um processo são os mesmos quer ele ocupe a máquina sozinho quer com outros processos
- O estado da computação (CPU) preservado nas trocas de contexto
- Um processo só consegue escrever na RAM que lhe está reservado (hardware garante)
- As interações com periféricos e ficheiros são realizadas pelo SO e portanto a sua correção é garantida



Troca de context (Context Switch)

- Quando o CPU comuta de processo (**context switch**), o SO precisa de salvar o estado da computação do processo corrente e restaurar o estado da computação do processo que vai ocupar o CPU

- O estado do processo é **guardado** pelo SO sempre que o CPU lhe é retirado
- O estado do processo é **restaurado** pelo SO sempre que o CPU volta a ser atribuído ao processo

Mecanismos Hardware Indispensáveis ao SO

- Instruções privilegiadas
- Proteção do espaço de I/O
- Proteção de memória
- Proteção do CPU
- Interrupções

Dois modos de operação do CPU

- A partilha de recursos exige que um programa incorrecto não prejudique outros programas/utilizadores do sistema

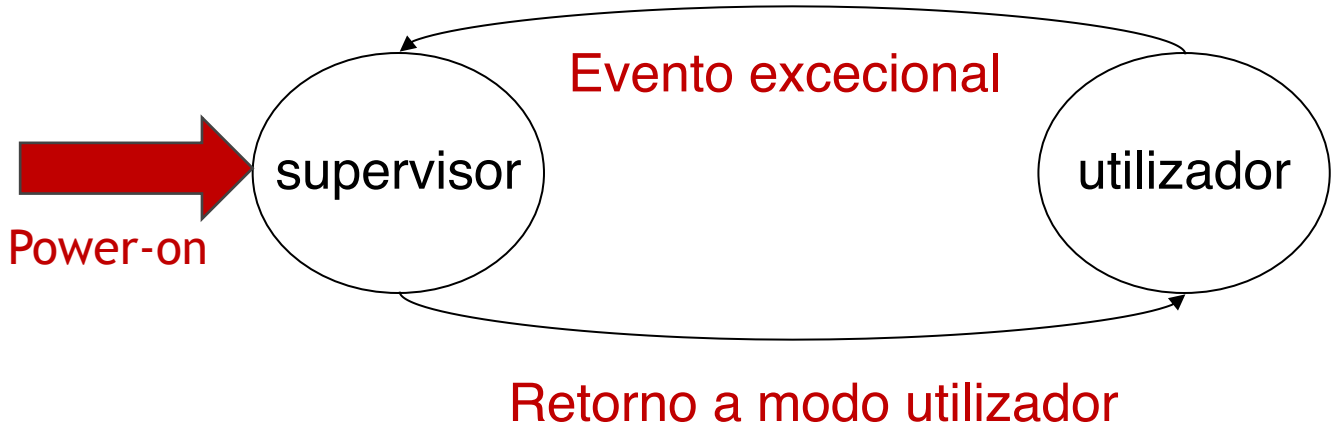
- O hardware tem de suportar pelo menos dois modos de operação do CPU:

  Modo **utilizador** – CPU está a executar por conta de um utilizador.

  Modo **sistema**/**supervisor** (**kernel** mode or **system** mode) – execução de código do SO

Modo utilizador/supervisor

- **bit de modo -** indica o modo corrente: supervisor (0) ou utilizador (1).
- Quando ocorre um evento excecional o CPU muda para modo supervisor.



***Instruções privilegiadas* só podem ser executadas em modo supervidor.**

Protecção de I/O

- Todas as instruções de I/O são **privilegiadas**.

- É preciso garantir que um programa utilizador nunca executa o seu código em modo supervisor (podia por exemplo mudar a programação dos dispositivos de entrada / saída).

Proteção de Memória

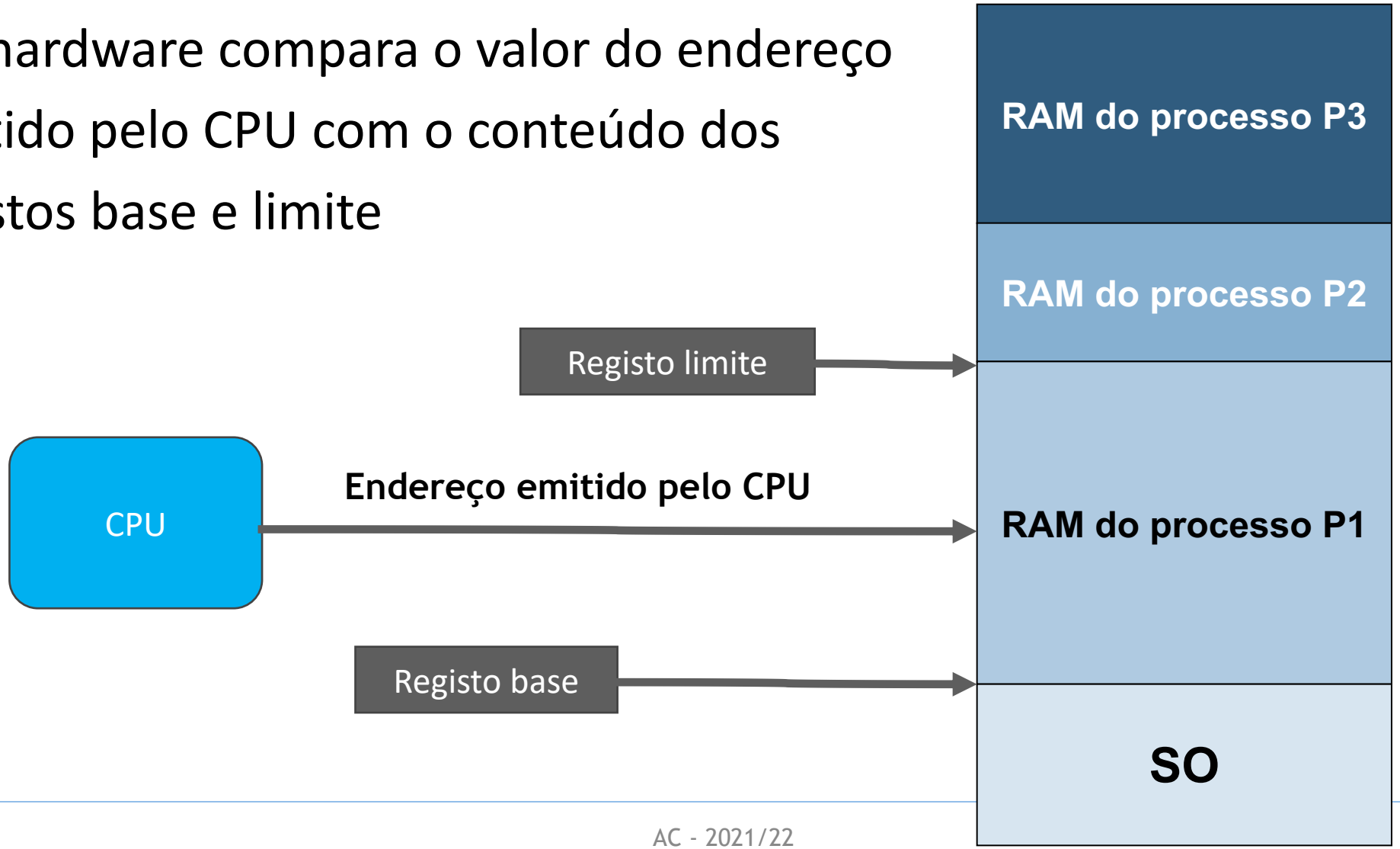
- É necessário pelo menos proteger o código do Sistema Operativo

- A solução mais simples é ter dois registos que determinam a faixa legal de endereços:
  - **Registo Base**
  - **Registo Limite**

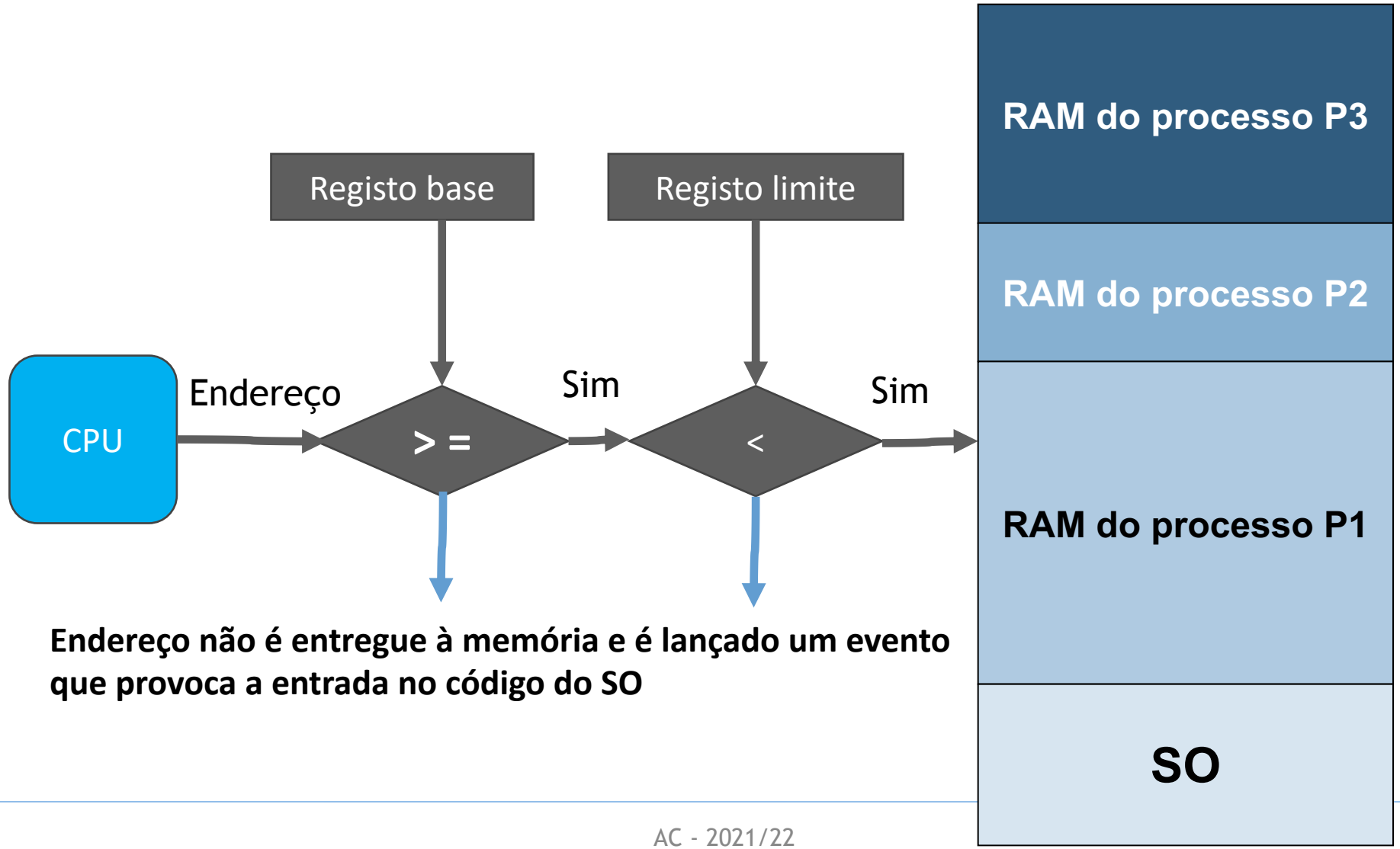
- A memória fora da zona é **protegida**.

Registos base e limite

- O hardware compara o valor do endereço emitido pelo CPU com o conteúdo dos registos base e limite



Registos base e limite



Endereço não é entregue à memória e é lançado um evento que provoca a entrada no código do SO

Protecção de memória

- Em modo supervisor o acesso à memória é total.

- As instruções para alterar os registos base e limite são **privilegiadas**.

Protecção do CPU

- **Temporizador** – interrompe o processamento ao fim de um tempo pré-determinado para assegurar que o SO nunca perde o controlo
  - Decrementado a cada “tick” do relógio.
  - Quando chega a 0 desencadeia um evento

- As instruções que manipulam o temporizador são **privilegiadas**.

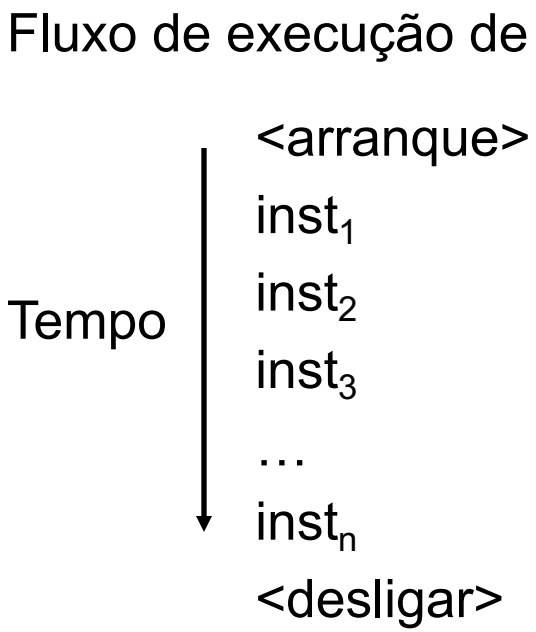
Fluxo de Execução Excecional

- Necessidade de interromper o fluxo normal de execução de instruções quando:
  - Se executa uma instrução privilegiada com o CPU em modo utilizador
  - Se faz acesso a uma posição de memória não permitida
  - O temporizador assinala o fim de um intervalo de tempo

- Noção de interrupção de programa

Execução de instruções

- Desde o arranque até desligar o CPU obtém instruções (fetch) e executa-as
  - Isto constitui o fluxo de controlo **normal** de execução de instruções

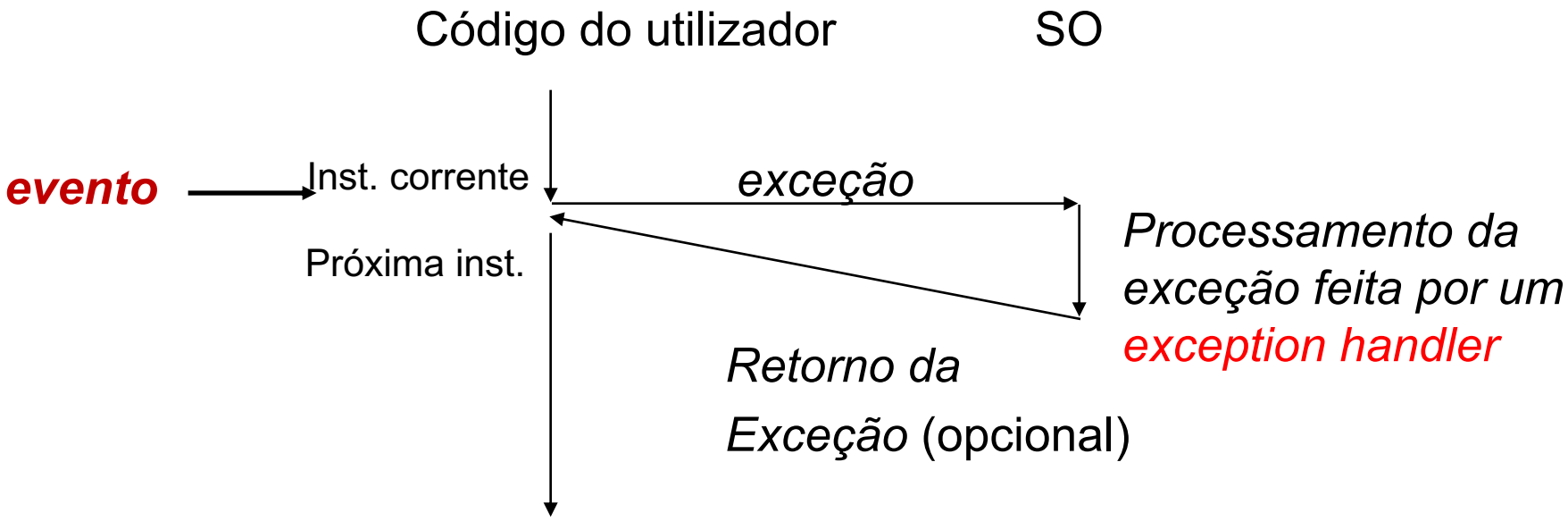


Alteração do fluxo de controlo

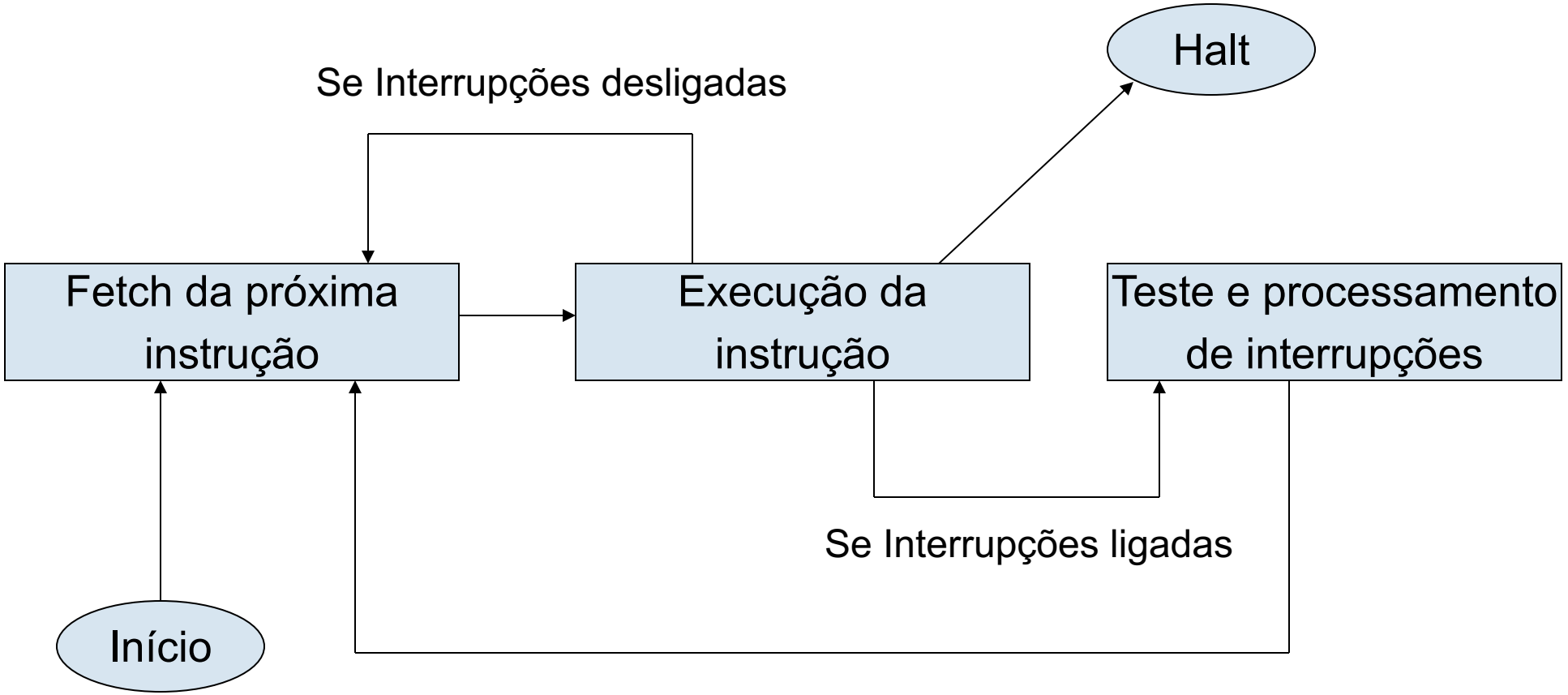
- Até agora dois mecanismos para quebrar o fluxo sequencial de instruções :
  1. Saltos incondicionais e condicionais
  2. Call e return que usam o stack.
- Correspondem a alterações de estado no programa corrente. Insuficiente para um sistema útil
- É difícil para o CPU responder a alterações exteriores ao programa corrente.
  - Dados que chegam do disco ou da rede
  - Utilizador carrega em ctl-c no teclado
  - O relógio do sistema dá um impulso
- São precisos mecanismos que forcem alterações do fluxo de controlo

Exce(p)ção

- Uma exceção transfere o controlo para o SO em resposta a um evento)



Ciclo fetch/execute com interrupções



Tipos de interrupções

- Ligados ao programa em execução
  - Overflow em operação aritmética
  - Divisão por zero
  - Execução de uma instrução privilegiada
  - Referência a uma zona de memória que não pertence ao programa

- Ligados ao hardware
  - Temporizador
  - Dispositivos de entrada/saída
  - Falha de hardware

Excepções assíncronas (Interrupções por hardware)

- Causada por eventos exteriores ao CPU
  - Colocação a 1 do pino de interrupção do CPU
  - depois do tratamento volta-se à instrução “seguinte”.

- **Exemplos**:
  - Fim de operações de entrada saída
    - ctl-c premido no teclado
    - Chegada de um pacote pela rede
    - Chegada do conteúdo de um bloco do disco

Excepções síncronas

- Resultam de eventos associados à instrução corrente:
  - Traps (ou interrupções por software)
    - Ação intencional
    - **Exemplos**: chamadas ao sistema, breakpoint no debugger
    - O control retorna à “próxima” instrução
  - Faults (falhas)
    - Não intencionais, mas potencialmente recuperáveis
    - **Exemplos**: acesso a zona de memória que é legal, mas que não está disponível (recuperável), execução de instrução privilegiada em modo utilizador (irrecuperável).
    - Pode reexecutar a instrução corrente (que provocou a falha) ou abortar o programa.

Verificação de interrupções

- O CPU verifica se há interrupções pendentes
- Se não há interrupções pendentes
  - vai fazer o “fetch” normalmente

    `IR = mem[PC]`
- Se há uma interrupção pendente,
  - suspende a execução do programa corrente e vai executar a chamada “rotina de tratamento” da interrupção (interrupt handler)

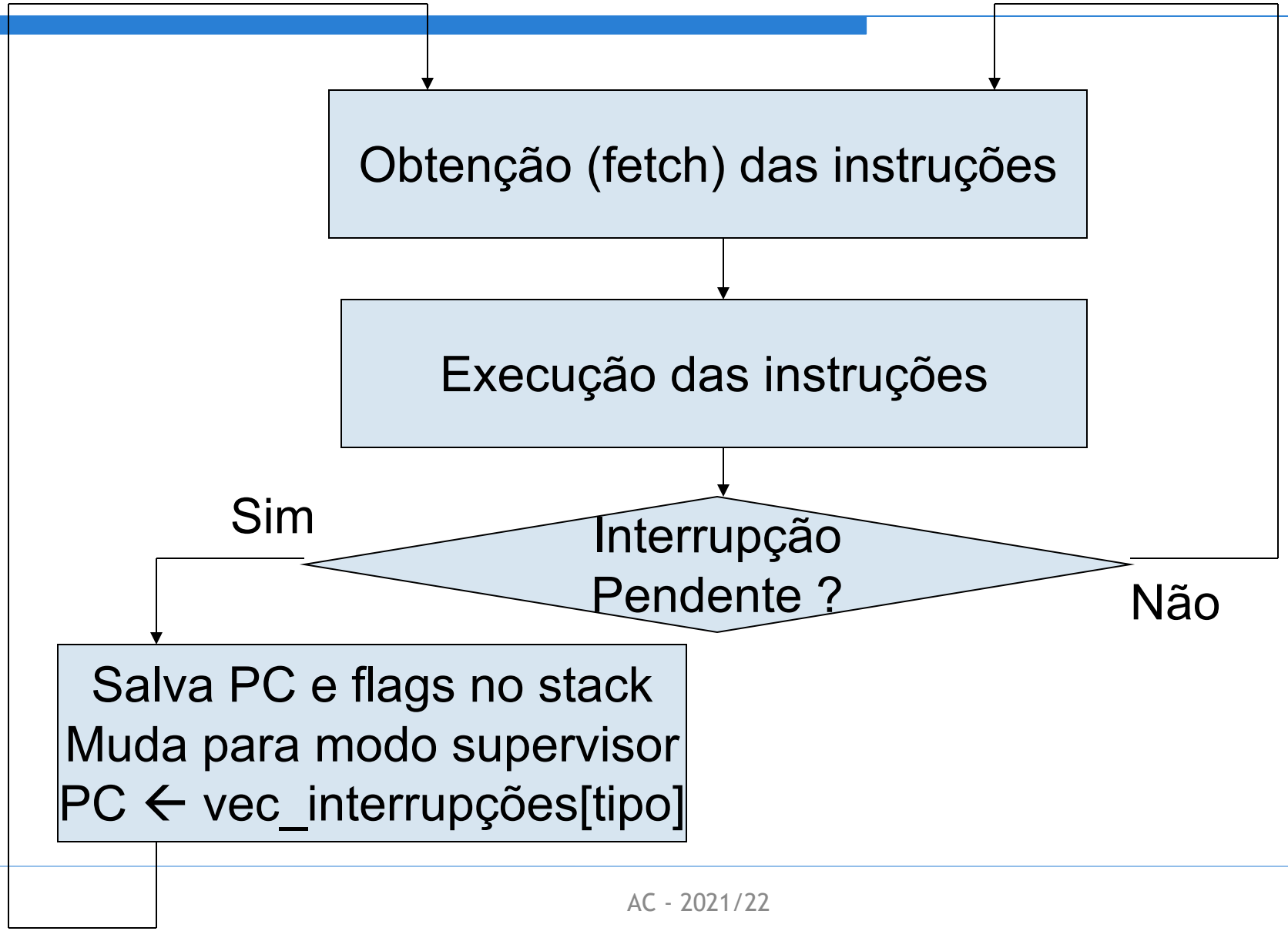
    PC ← valor dependente da causa da interrupção

- A computação interrompida será retomada mais tarde.
  - Para isso é preciso salvar o “estado do CPU”:
    - Valores do Program Counter, Flags e eventualmente Registos de uso geral

Interrupt Handler
(Rotina de tratamento de interrupções)

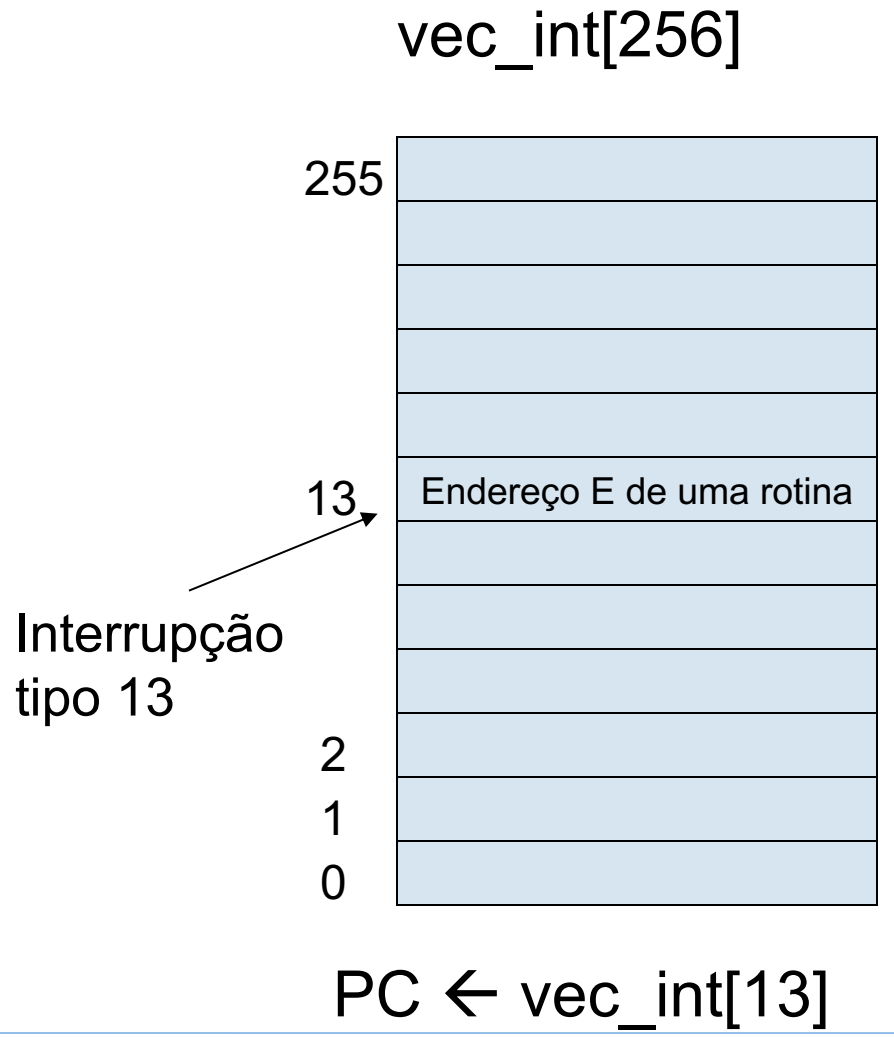
- Determina-se o motivo da interrupção e efetuam-se as ações que são necessárias
- A identificação da natureza da interrupção é feita pelo hardware.
- Muitas vezes o hardware também:
  - Salva automaticamente no “stack” o valor do PC e das “flags” PC+Flags = PSW (Processor Status Word)
  - Carrega um novo valor no PC
- O código que é invocado faz geralmente parte do sistema de operação

Interrupções



Interrupções vetorizadas

- Em muitos casos, o hardware associa um número não negativo cada causa possível de interrupção
- Esse número indexa uma tabela com endereços de funções (vetor de interrupções)



Fim da rotina de tratamento de interrupções

- Quando a rotina de tratamento de interrupção termina é executada a instrução máquina **interrupt return (iret)**

- Esta instrução
  - Desempilha o PC, o que faz retomar a execução no ponto em que a execução "normal" foi interrompida
  - Também restaura o valor das "flags". Isto permite:
    - retomar a execução no ponto onde se encontrava com as "flags" como se encontravam na altura em ocorreu a interrupção
    - O bit de modo é restaurado. O modo volta a ser "modo utilizador"

Chamadas ao sistema

- Fornecem a interface entre um programa em execução e o SO.
  - Disponíveis em “assembly”.
  - Linguagens usadas na programação de sistemas (e.g., C, C++)

Chamadas ao sistema

- Exemplo típico (UNIX, POSIX)

```
#include <errno.h>
…
retval = read(fd, buffer, nbytes);
if (retval == -1)
    switch(errno) {
         case EIO:  printf( … ); break;
         case EBADF: printf( … ); break;
          …
    }
```

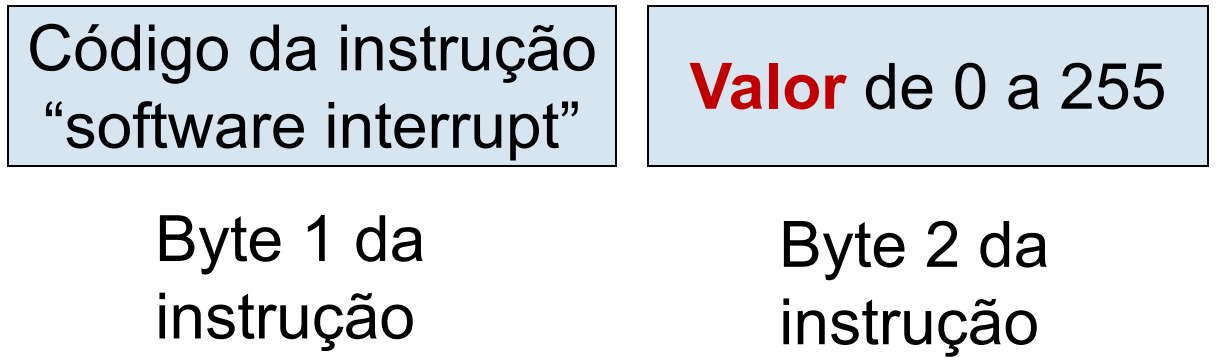
- Sistema de execução da linguagem (neste caso libc …)
  - Prepara um conjunto de parâmetros no stack ou em registos do CPU
    - Código da operação (registo) Restantes parâmetros (stack)
    - Invoca os serviços do sistema
    - Devolve os resultados

Interrupções por software

- Quando se pretende invocar os serviços do sistema é preciso invocar código que pertence ao sistema e que executa em modo supervisor

- Para “saltar para dentro” do sistema e ao mesmo tempo mudar o modo do CPU de utilizador para supervisor pode-se simular a ocorrência de uma interrupção. É o papel de uma instrução máquina chamada “software interrupt”

Interrupções por software



A execução desta instrução limita-se:

- A posicionar o indicador de que há uma interrupção pendente
- Associar a essa interrupção o tipo que aparece no 2º byte da instrução

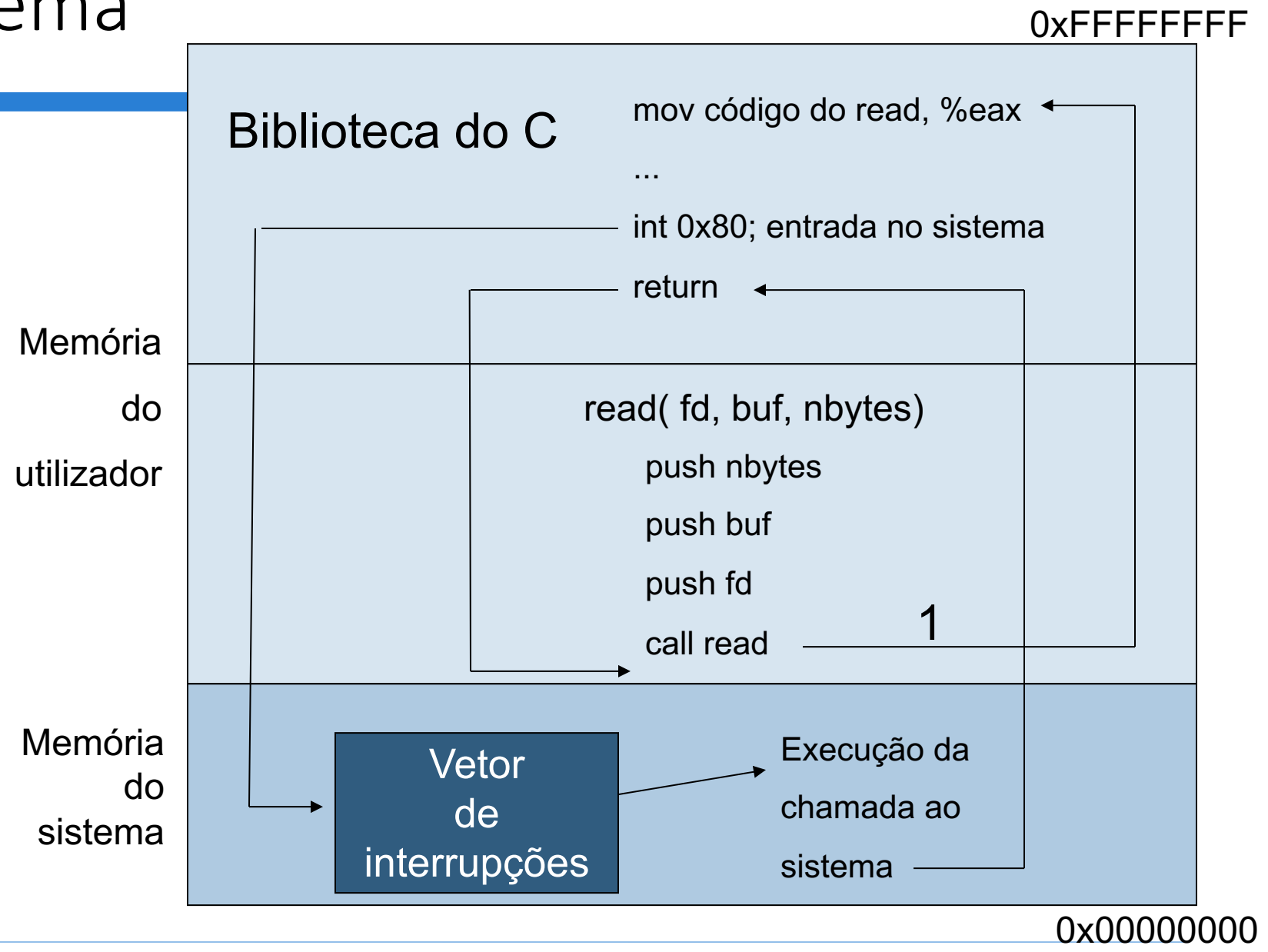
Assim o PC recebe o endereço vector_interrupções[**valor**]

Interrupções por software

- A interrupção por software é usada para fazer a chamada ao sistema
- O SO define qual é o valor associado à interrupção por software que deve ser usada
- Na posição combinada, preenche o vetor de interrupções com o endereço da sua rotina que processa as chamadas ao sistema

Chamada ao sistema

- Chamada ao sistema read (fd, buf, nbytes)



Chamada ao sistema como uma função

- A chamada ao sistema é executada como se fosse uma subrotina.

- Quando se entra no código do SO:
  - Muda-se para modo supervisor
  - Troca-se para um “stack” usado só nas chamadas
  - Verificam-se os parâmetros
  - Executa-se a chamada
  - Troca-se o “stack” para o habitual
  - Muda-se para modo utilizador
  - Retorna-se

Chamadas ao sistema no Linux

- Biblioteca standard do C na arquitectura x86

```
            movl código da chamada, %eax
            mov 1o. Parâmetro, %ebx
            ...
            int 0x80
  Cont:
```

- O início do vector de interrupções em memória (256 entradas de 8 bytes) é conhecido
- CPU em modo supervisor
  - PC ← end. handler de chamadas ao sistema no kernel
- A tabela *sys_call_table* é indexada pelo registo EAX e contem os endereços de cada uma das rotinas que suportam as chamadas ao sistema
- A rotina executada termina com a instrução **iret** que volta a colocar o CPU em modo utilizador e a execução continua no ponto a seguir à invocação

Chamada ao sistema `exit(status)`

- Usada pelo programa para libertar os recursos usados e retorna um valor que pode ser consultado.

```
movl $1, %eax   ; código do exit
movl $1, %ebx   ; status = 1
int 0x80        ; entrada no código do Linux
```