



Capítulo 2: Estruturas dum Computador

SUMÁRIO:

- 2.1 Bios e BootStrap
- 2.2 Funcionamento dum Computador
 - ☞ Interrupções
- 2.2 Arquitectura de I/O
 - ☞ Hardware I/O
 - ☞ Alternativa a Interupções - Polling
 - ☞ Método Síncrono, Assíncrono,
 - ☞ Interface I/O
 - ☞ DMA
- 2.3 Estrutura de Armazenamento
 - ☞ Hierarquia de Armazenamento
- 2.4 Protecção do Hardware
 - ☞ Dual Mode

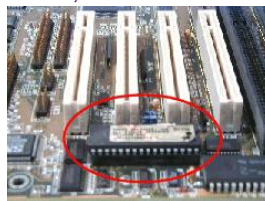


Computer Startup BIOS e BOOSTRAP

■ BIOS

- ☞ sigla para Basic Input/Output System (Sistema Básico de Entrada/Saída)
- ☞ O BIOS é o primeiro programa executado pelo computador ao ser ligado.
- ☞ Sua função é preparar a máquina para que o SO, que pode estar armazenado em diversos tipos de dispositivos (discos rígidos, disquetes, CDs, etc) possa ser carregado para memória e iniciado a sua execução.
- ☞ O BIOS é Firmware.

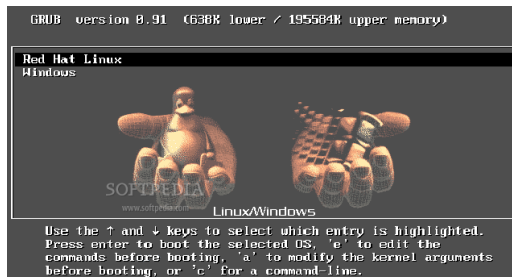
■ **Firmware** : Qualquer Software armazenado sob a forma de memória de leitura não-volatil, ROM, EPROM, EEPROM etc.



BootStrap

■ O Programa “bootstrap”

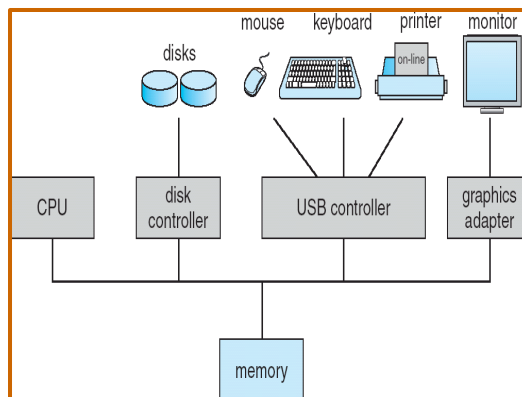
- ☞ Um programa carregado durante “power-up” ou “reboot” pelo BIOS. O BIOS não sabe nada sobre a ambiente necessário para um SO
- ☞ O BIOS apenas inicialize o hardware para um estado “conhecido”.
- ☞ O BIOS transfere controle para o programa bootstrap que inicialize todas as partes do sistema necessário para carregar o kernel do SO.
- ☞ O Bootstrap, deverá localizar e carregar o kernel do sistema operativo para memória e inicialize a sua execução ou fazer um processo mais complexo (multi-boot etc.)
- ☞ Exemplos GRUB, LILO, WINLOAD etc.



Operating System Concepts

Funcionamento dum computador

- Controladores I/O e a CPU podem executar numa forma concorrente.
- Cada controlador está encarregue dum tipo de dispositivo particular.
- Cada controlador tem um buffer local.
- A CPU movimenta dados de (para) memória principal para (a partir de) os buffers locais.
- I/O é a partir do dispositivo para o buffer local do controlador.
- O controlador informa a CPU que terminou a sua operação através dum **interrupção**.



Arquitetura .. dum computador

Operating System Concepts

2.4

Silberschatz, Galvin and Gagne ©2002



Funções desempenhadas por interrupções

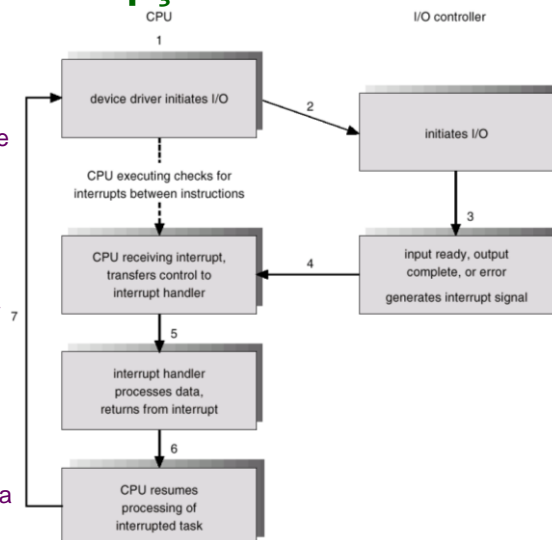
- Uma interrupção transfere o controlo para uma rotina de serviço através do interrupt vector, o qual contém os endereços de todas as rotinas de serviço à interrupção.
- A arquitectura de interrupções tem de guardar o endereço da instrução interrompida.
- As interrupções entretanto chegadas são *disabled* enquanto outra interrupção está sendo processada por forma a impedir a sua perda.
- Uma interrupção gerada por software, devido a um erro p.ex., divisão por zero, acesso inválido à memória ou a um pedido do utilizador (ctrl-c) é chamado uma “trap”.
- A rotina de Serviço (**Interrupt Service Routine -ISR**) implementa a funcionalidade da interrupção
- Um sistema operativo é interrupt driven. Explicar !!
- [Windows device manager](#)



EXEMPLO: Comunicação CPU-Dispositivo I/O via Interrupções



- Durante I/O, interrupções são feitas por vários dispositivos quando eles ficam prontos para serviço.
- Estas interrupções significam que a saída de dados terminou, ou que a entrada de dados está disponível, ou que uma falha foi detectada.
- O controlador interrompe a CPU através da emissão dum sinal na linha de pedido de interrupção;
- A CPU detecta a interrupção e despacha-a para o *interrupt handler*, que é uma rotina
- A *interrupt handler* determina a causa da interrupção, faz o processamento necessário (rotina de serviço à interrupção) e termina; ao terminar, a CPU volta ao estado anterior à interrupção



Ciclo I/O alimentado por interrupções





Tabela de eventos do processador Intel Pentium

- A maior parte das CPUs tem agora duas linhas de interrupção:

- ☞ Linha de interrupção **não-mascarável**, que é reservada para eventos tais como erros de memória irrecuperáveis
- ☞ Linha de interrupção **mascarável**, que pode ser desligada pela CPU antes da execução duma sequência de instruções críticas que não podem ser interrompidas.

- Vector de interrupções para despachar cada interrupção para o *handler* correcto

- ☞ Baseado em prioridades
- ☞ Algumas não são mascaráveis (os primeiros 32 eventos do Intel Pentium não são mascaráveis e são usadas para sinalização de erros)

| vector number | description |
|---------------|--|
| 0 | divide error |
| 1 | debug exception |
| 2 | null interrupt |
| 3 | breakpoint |
| 4 | INT0-detected overflow |
| 5 | bound range exception |
| 6 | invalid opcode |
| 7 | device not available |
| 8 | double fault |
| 9 | coprocessor segment overrun (reserved) |
| 10 | invalid task state segment |
| 11 | segment not present |
| 12 | stack fault |
| 13 | general protection |
| 14 | page fault |
| 15 | (Intel reserved, do not use) |
| 16 | floating-point error |
| 17 | alignment check |
| 18 | machine check |
| 19-31 | (Intel reserved, do not use) |
| 32-255 | maskable interrupts |

- Mecanismo de interrupções também usado para excepções (p.ex. divisão por 0, acesso a endereço de memória protegida ou inexistente)



Manipulação de interrupções

- Repare que para tratar duma interrupção
 - ☞ O sistema operativo preserva o estado da CPU
 - ☞ registadores contador de programa etc.
 - ☞ Estado do cpu é chamado o “contexto”
- O Interrupt Handler do sistema operativo determinam que acção deve ser tomada para cada tipo de interrupção.
- Comunicação CPU-Dispositivo I/O
 - ☞ *vectored* interrupt system
 - ☞ fazer lookup numa tabela
 - ☞ **Alternativa : Polling**
 - ☞ pedir ao hardware



Hardware I/O

“Every transfer is an output from one device and an input into another.”

- Como é que a CPU pode fornecer comandos e dados a um controlador para efectuar uma transferência I/O?
 - ☞ O controlador tem um ou mais registos para dados (registo **data-in**, registo **data-out**) e sinais de controlo (**registo status** e **registo control**)
 - ☞ A CPU comunica com o controlador através da leitura e escrita de “padrões de bits” nestes registos
- Duas formas de comunicação CPU-controlador:
 - ☞ Instruções I/O directas (1 byte + endereço de porto I/O)
 - ☞ Instruções I/O mapeadas em memória

| I/O address range (hexadecimal) | device |
|---------------------------------|---------------------------|
| 000-00F | DMA controller |
| 020-021 | interrupt controller |
| 040-043 | timer |
| 200-20F | game controller |
| 2F8-2FF | serial port (secondary) |
| 320-32F | hard-disk controller |
| 378-37F | parallel port |
| 3D0-3DF | graphics controller |
| 3F0-3F7 | diskette-drive controller |
| 3F8-3FF | serial port (primary) |

Localizações de alguns portos I/O de dispositivos de PCs

Hardware I/O: Comunicação CPU-Dispositivo I/O via **Polling** Espera Ocupada da CPU

A interação entre a CPU e um controlador faz-se por **aperto de mão** (*handshaking*). Assuma que são usados 2 bits (**busy** bit do registo status e **ready** bit do registo command do controlador) para coordenar a relação produtor-consumidor entre o controlador e a CPU:

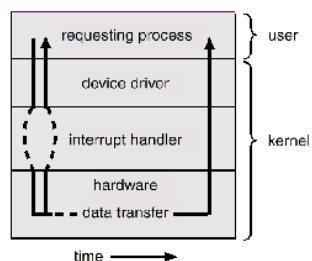
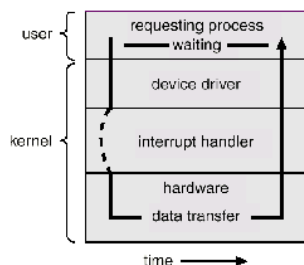
- Determines state of device
 - ☞ command-ready
 - ☞ busy
 - ☞ Error
- Busy-wait cycle to wait for I/O from device

Busy Wait

- A CPU lê repetidamente o bit **busy** até que seja 0 (**busy-wait cycle**).
 - A CPU activa o bit **write** a 1 no registo command e escreve um byte no registo data-out.
 - A CPU activa o bit **ready** a 1 no registo command.
 - Quando o controlador nota que o bit **ready** está a 1, it escreve o bit **busy** a 1.
-
- O controlador lê o registo command e vê o comando **write**. Lê o registo **data-out** para obter o byte, fazendo de seguida a I/O para o dispositivo.
 - O controlador desactiva o bit **ready** a 0, assim como o bit **error** no registo status, para indicar que a transferência foi bem sucedida, colocando depois o bit **busy** a 0 para indicar que a transferência terminou.

Método Síncrono de I/O

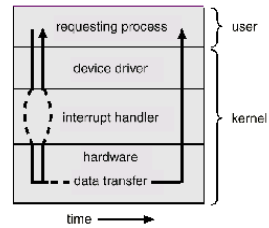
- **MÉTODO SÍNCRONO.** Após o início duma operação I/O, o controlo só retorna ao programa do utilizador depois da terminação desta operação I/O.
 - ⦿ Implementação: A instrução *wait* (se existir) coloca a CPU num estado de espera até à próxima interrupção. Alternativamente utilize-se um ciclo tipo *Wait*
 - ⦿ Desvantagens –
 - No máximo um pedido I/O é atendido de cada vez.
 - Não existe nenhum processamento I/O simultâneo.
 - Contenção para acesso à memória (caso do ciclo *Wait*).





Método Assíncrono de I/O

- **MÉTODO ASSÍNCRONO.** Após o início de I/O, o controlo retorna ao programa do utilizador sem esperar pela terminação da operação I/O.
- Vantagens – Concorrente I/O Operações. Permite Processamento **Simultâneo**.
- Desvantagens – Implementação Mais Complexa



- ↳ *Device-status table* contém uma entrada para cada dispositivo I/O, indicando o tipo, endereço e estado.
- ↳ O sistema operativo indexa a *I/O device table* para determinar o estado dum dispositivo e modificar a sua entrada na tabela por forma a reflectir a ocorrência da interrupção.

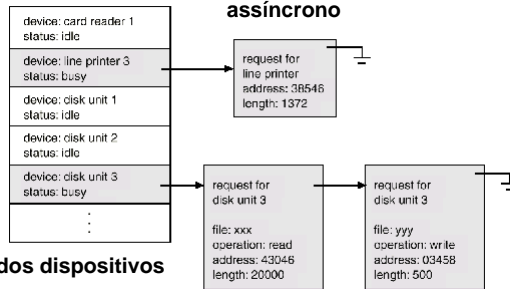
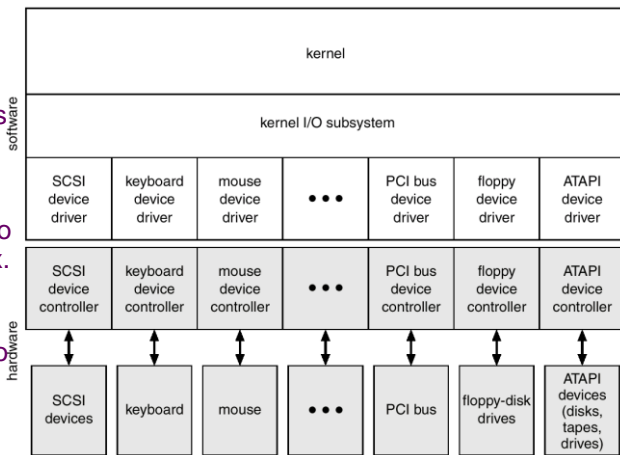


Tabela de estado dos dispositivos



Interface de I/O

- **Interface I/O do kernel:** conjunto de funções I/O independentes do hardware (p.ex. read, write, etc.)
- **Device drivers:** módulos do kernel dependentes do hardware, cada um dos quais encapsula o funcionamento específico de cada dispositivo (p.ex. rato, teclado, etc.).
- **Vantagem:** um novo periférico pode ser ligado a um computador sem que a empresa vendedora do sistema operativo tenha que fornecer o código de suporte do periférico.




Estrutura I/O do kernel



Interface de I/O: Características dos dispositivos de I/O

- Dispositivos diferenciam-se de várias maneiras:
 - ☞ **Character-stream** ou **block**.
Um dispositivo de caracteres transfere bytes um-a-um, ao passo que um dispositivo de blocos transfere um bloco de bytes como se fosse uma unidade.
 - ☞ **Sequenciais** ou de **acesso aleatório**
 - ☞ **Síncronos** ou **assíncronos**.
Transferência de dados é feita com tempos de resposta previsíveis ou não.
 - ☞ **Partilháveis** ou **dedicados**.
Há partilha concorrente por vários processos ou não.
 - ☞ **Velocidade de operação**
 - ☞ **read-write**, **read only**, ou **write only**

| aspect | variation | example |
|--------------------|---|---------------------------------------|
| data-transfer mode | character block | terminal disk |
| access method | sequential random | modem CD-ROM |
| transfer schedule | synchronous asynchronous | tape keyboard |
| sharing | dedicated sharable | tape keyboard |
| device speed | latency seek time transfer rate delay between operations | |
| I/O direction | read only write only readWrite | CD-ROM graphics controller disk |



Operating System Concepts 2.15 Silberschatz, Galvin and Gagne ©2002

Interface de I/O: dispositivos de blocos e de caracteres

- Os **dispositivos de blocos** incluem os discos rígidos
 - ☞ **Interface básica:** read, write, seek
 - ☞ **Modos de acesso:**
 - ☑ Acesso de alto-nível através de interface de sistema de ficheiros
 - ☑ Acesso de baixo-nível através array linear de blocos (*raw I/O*)
 - ☑ Acesso a *memory-mapped files* colocadas no topo de block-device drivers é possível. Uma *memory-mapped interface* fornece acesso ao disco através dum array de bytes em memória principal.
- Os **dispositivos de caracteres** (*stream character*) incluem teclados, ratos e portas série
 - ☞ **Interface básica:** get, put
 - ☞ No topo da interface, é possível construir bibliotecas para edição de linhas (p.ex. Eliminar um carácter do input stream através de backspace).






Operating System Concepts 2.16 Silberschatz, Galvin and Gagne ©2002

Interface de I/O: Dispositivos de rede

- Têm uma interface bastante diferente da interface **read-write-seek** usada pelos discos rígidos.
- Unix e Windows NT/9x/2000 usam uma interface de **sockets**
 - ☞ Há a separação entre o protocolo de rede e o funcionamento da rede
 - ☞ Inclui a funcionalidade `select` para manipular conjuntos de sockets
- Abordagens muito variadas (pipes, FIFOs, streams, queues, mailboxes)



Ethernet Adapter Cards



PCMCIA compatible Fax/Modem cards



External Fax/Modems

Operating System Concepts 2.17 Silberschatz, Galvin and Gagne ©2002

Interface de I/O: Relógios (clocks) e temporizadores (timers)

- Fornecem três funções básicas para:
 - ☞ Determinar tempo corrente
 - ☞ Determinar tempo decorrido
 - ☞ Activar um temporizador para despoletar uma operação numa dada altura
- Um *programmable interval timer* é usado para suportar a segunda e terceira funções anteriores. (Este mecanismo é usado pelo escalonador para gerar uma interrupção que preempta um processo no fim de esgotar o seu time-slice (algoritmo Round-Robin - Capitulo 5)
- A função `ioctl` (UNIX) aborda aspectos de I/O tais como relógios e temporizadores.

Operating System Concepts 2.18 Silberschatz, Galvin and Gagne ©2002



Estrutura DMA (Direct Memory Access)

- O DMA permite que certos dispositivos de hardware num computador acessem a memória do sistema para efectuar operações de I/O (leitura e escrita) independentemente da CPU.
- É usada em dispositivos I/O de alta-velocidade capazes de transmitir informação a velocidades próximas das memórias.
- O controlador de dispositivo transfere blocos de dados do *buffer* directamente para a memória principal sem intervenção da CPU.
- Só uma interrupção é gerada por bloco, em vez de uma interrupção por *byte*.
- Permite execução simultânea do CPU e operações do I/O

[Exemplo do Funcionamento dum Line Terminal](#)

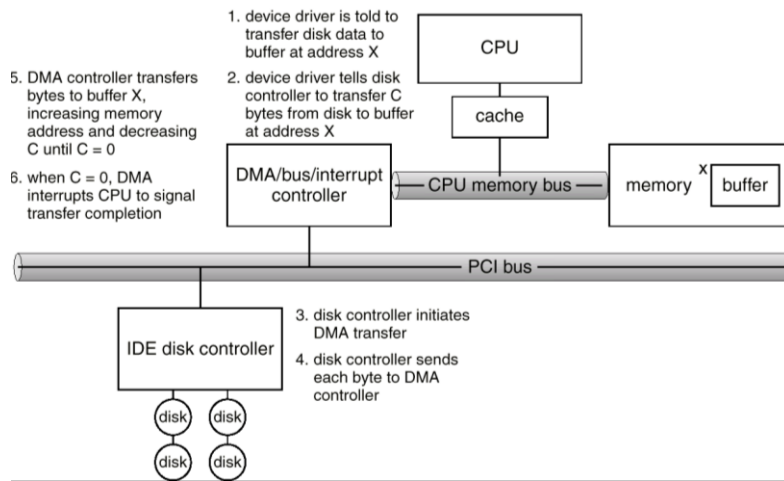


DMA Funcionamento

- Data transfer can be triggered in two ways:
 - ☞ either the software asks for data (via a function such as *read*) or
 - ☞ the hardware asynchronously pushes data to the system (for example with data acquisition devices which are always sending data).
- No Primeiro Caso - Sincrono
 1. When a process calls *read*, the driver method allocates a DMA buffer and instructs the hardware to transfer its data. The process is put to sleep.
 2. The hardware writes data to the DMA buffer and raises an interrupt when it's done.
 3. The interrupt handler gets the input data, acknowledges the interrupt, and awakens the process, which is now able to read data.



Diagram do Funcionamento DMA



DMA Refs

- *Linux Device Drivers, 2nd Edition, Alessandro Rubini & Jonathan Corbet*
<http://www.xml.com/ldd/chapter/book/ch13.html>
- *Wikipedia*
http://pt.wikipedia.org/wiki/Acesso_direto_à_memória



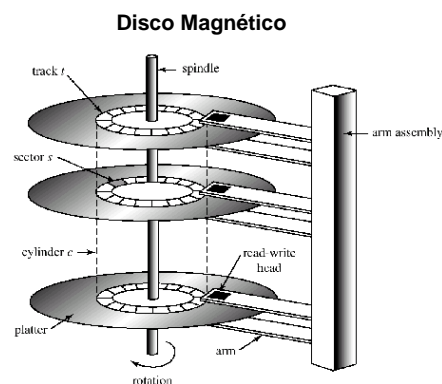
Armazenamento

- A situação ideal
 - ☞ Programas e Dados localizados na memória principal
- Não é uma situação realística
 - ☞ Memória Principal é geralmente demasiado pequena para guardar todos os programas e dados em memória
 - ☞ Memória Principal é “volatile”
 - ☞ (Pode ser realística em sistemas especiais !)
- Memória Principal é demasiado longe do CPU (Von-Neumann Bottleneck)
- Portanto sistemas computacionais fornecem várias formas de memória secundária



Meios de Armazenamento

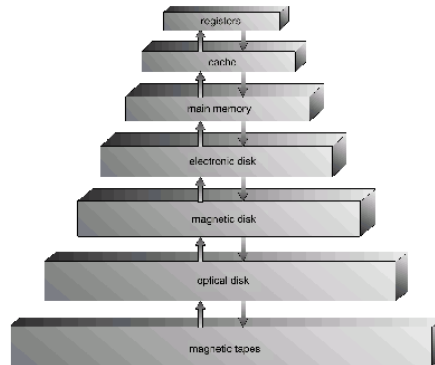
- Memória principal – meio de armazenamento que só a CPU pode aceder directamente.
- Memória secundária – extensão da memória principal que fornece uma grande capacidade de armazenamento não-volátil.
- Discos magnéticos – pratos de vidro ou de metal rígido revestidos de material magnético de gravação
 - ☞ A superfície do disco está logicamente dividida em pistas (*tracks*), as quais pro sua vez estão divididas em sectores (*sectors*).
 - ☞ O controlador do disco determina a interacção lógica entre o dispositivo e o computador.





Hierarquia de Armazenamento

- Hierarquia de armazenamento pode ser feita em termos:
 - ☞ Velocidade
 - ☞ Custo
 - ☞ Volatilidade
- *Caching* – cópia de informação num sistema de armazenamento mais rápido; a memória principal pode ser vista como a última *cache* para o armazenamento secundário.
- *Caching* – Utilização de memória de alta-velocidade para guardar os dados recentemente acedidos. Requer uma política de *gestão de cache*.



Hierarquia de memórias



Protecção de Hardware

- Operação em modo dual (*Dual-Mode*)
- Protecção I/O
- Protecção de memória
- Protecção da CPU

A partilha de recursos do sistema requer que o sistema operativo assegure que um programa incorrecto não faça com que outros programas funcionem incorrectamente.

Exemplo: SO simples de Loteamento de Tarefas
Imagine um programa incorrecto que entre num ciclo infinito a ler dados/instruções sucessivas (cartões perfurados). Poderia começar a ler os dados/instruções da próxima tarefa com resultados catastrophicos para o batch !

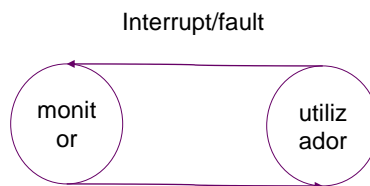
Solução: A Instrução "Load In Next Card" é privilegiado..





Operação em Dual-Mode

- Fornece suporte de hardware para diferenciar pelo menos dois modos de operação:
 1. *User mode* – execução desencadeada pelo utilizador.
 1. *Monitor mode* (ou *kernel mode* ou *system mode* ou *bit mode*) – execução desencadeada pelo sistema operativo.
- Quando uma interrupção ou excepção ocorre, o hardware comuta para o modo monitor.
- *Implementação*
um bit pode ser adicionado ao hardware do computador para indicar o modo corrente: monitor (0) ou utilizador (1) – chamado Mode Bit



Instruções privilegiadas só podem ser emitidas em modo monitor.

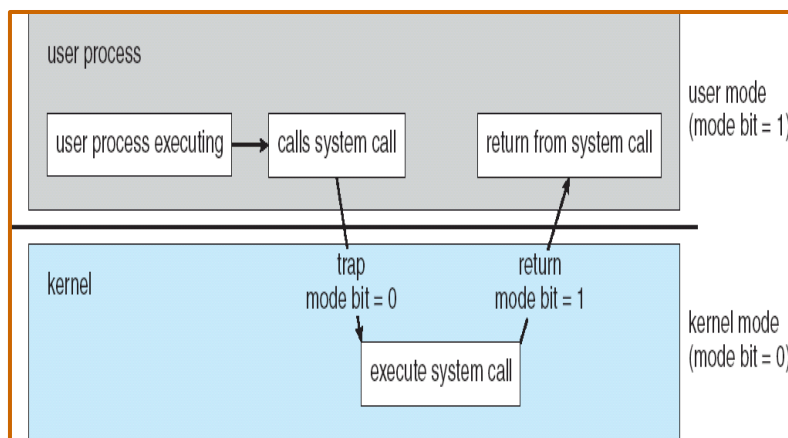
Arquitectura Intel 80486
mode bit → dual mode

SO - WINNT OS/2 Utilizem este facto – mais seguro !

[Intel 8088](#) [intel 80286](#)



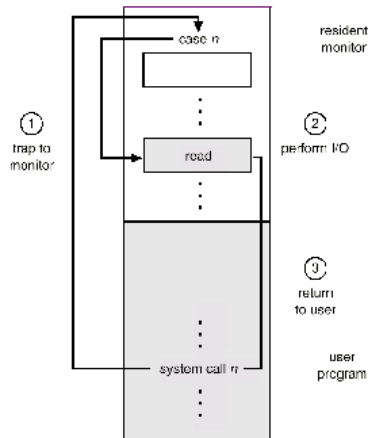
Mudança do modo





Protecção I/O

- Todas as instruções I/O são instruções privilegiadas.
- Tem de assegurar que um programa do utilizador jamais pode adquirir controlo do computador em modo monitor (ou seja, um programa do utilizador que, durante a sua execução, armazena um novo endereço no vector de interrupções).

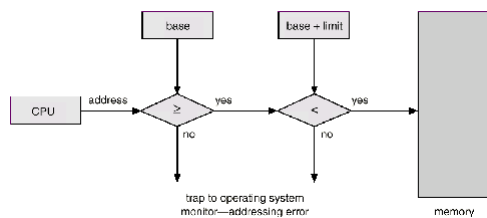
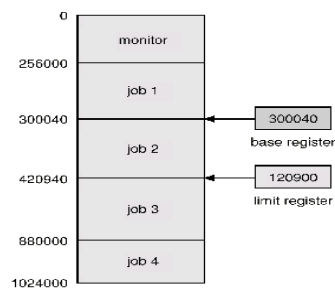


Chamada ao sistema para efectuar I/O



Protecção de Memória

- Tem de fornecer protecção de memória - pelo menos para o vector de interrupções e ISR's
- Para ter protecção de memória, adiciona-se dois registos que determinam a gama de endereços que um programa possa aceder:
 - ☞ **Registo de base** – guarda o endereço mais baixo de memória física.
 - ☞ **Registo limite** – o tamanho da gama.
- Memória fora da gama está protegida.
- Sob execução em modo monitor, o sistema operativo tem acesso livre quer à memória do monitor quer à memória do utilizador.
- As instruções de carregamento dos registos de base e de limite são instruções privilegiadas.





Protecção da CPU

- Qualquer programa não pode executar demasiado tempo !
 - Ciclo Infinito
 - Processamento Abusivo
- Assegurar que o SO MANTÊM “controlo”
- *Timer* – interrompe o computador após período especificado.
 - ☞ O timer é decrementado por cada batida (tick) do relógio.
 - ☞ Quando o timer atinge o valor 0, ocorre uma interrupção.
- O *Timer* é usado para assegurar que um SO possa manter controlo e para implementar time-sharing.
- *Timer* é também usado para calcular o tempo corrente.
- *Load-timer* é uma instrução privilegiada.

FIM DE
CAPÍTULO



Pergunta 1

- Protecting the operating system is crucial to ensuring that the computer system operates correctly. Provision of this protection is the reason behind dual-mode operation, memory protection, and the timer. To allow maximum flexibility, however, we would also like to place minimal constraints on the user.
- The following is a list of operations that are normally protected. What is the *minimal* set of instructions that must be protected?
 - a. Change to user mode.
 - b. Change to monitor mode.
 - c. Read from monitor memory.
 - d. Write into monitor memory.
 - e. Fetch an instruction from monitor memory.
 - f. Turn on timer interrupt.
 - g. Turn off timer interrupt.





Pergunta 2

- Which of the following instructions should be privileged?
 - a. Set value of timer.
 - b. Read the clock.
 - c. Clear memory.
 - d. Issue a trap instruction.
 - e. Turn off interrupts.
 - f. Modify entries in device-status table.
 - g. Switch from user to kernel mode.
 - h. Access I/O device.



Pergunta 3

- Some CPUs provide for more than two modes of operation. What are two possible uses of these multiple modes?

Answer: Multiple

modes could be used to provide a finer-grained security policy. For example, rather than distinguishing between just user and kernel mode, you could distinguish between different types of user mode. Perhaps users belonging to the same group could execute each other's code. The machine would go into a specified mode when one of these users was running code. When the machine was in this mode, a member of the group could run code belonging to anyone else in the group. Another possibility would be to provide different distinctions within kernel code. For example, a specific mode could allow USB device drivers to run. This would mean that USB devices could be serviced without having to switch to kernel mode, thereby essentially allowing USB device drivers to run in a quasi-user/kernel mode.





```
[root@Gab-4-10-2 proc]# date ; more /proc/interrupts
Thu Mar 16 15:40:34 WET 2006
CPU0
0: 1715223      XT-PIC timer
1: 3368        XT-PIC i8042
2: 0           XT-PIC cascade
8: 1           XT-PIC rtc
9: 0           XT-PIC acpi, uhci_hcd
10: 0          XT-PIC uhci_hcd
11: 33455      XT-PIC uhci_hcd, eth0, Intel 82801DB-
    ICH4
12: 141170     XT-PIC i8042
14: 35704      XT-PIC ide0
15: 141        XT-PIC ide1
```

```
[root@Gab-4-10-2 proc]# date ; more /proc/interrupts
Thu Mar 16 15:40:38 WET 2006
CPU0
0: 1718941     XT-PIC timer
1: 3374        XT-PIC i8042
2: 0           XT-PIC cascade
8: 1           XT-PIC rtc
9: 0           XT-PIC acpi, uhci_hcd
10: 0          XT-PIC uhci_hcd
11: 33618      XT-PIC uhci_hcd, eth0, Intel 82801DB-
    ICH4
12: 141170     XT-PIC i8042
14: 35733      XT-PIC ide0
15: 141        XT-PIC ide1
```

