

## Entradas/Saídas

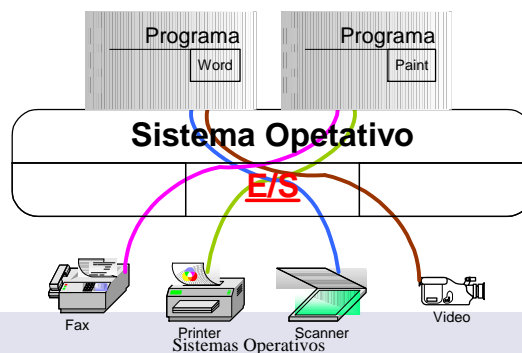
Modelo Conceptual  
Gestor de Periféricos  
Unix E/S  
Windows 2000 E/S

8/28/2003

Sistemas Operativos

## Introdução

- Parte do SO que permite a um programa interagir com o ambiente que o rodeia.
- Cria canais virtuais entre os programas e os periféricos.



8/28/2003

Sistemas Operativos

## Periféricos

- O número de periféricos tem vindo a aumentar significativamente
  - Teclado, rato, microfone, joystick, leitor de cartões, mesa gráfica, scanner, câmara de vídeo, altifalantes, impressora, display, DVD, disco, disquete, etc.
  - Redes de computadores com diferentes meios físicos: ethernet, wi-fi, gprs, umts, wimax
- Diferentes características dos periféricos
  - operações aceites: só leitura, só escrita, ambas
  - unidade de transferência de informação: carácter, cadeias de caracteres, blocos de dimensão fixa, etc.
  - velocidade de transferência: de alguns bytes por segundo até aos milhões de bytes por segundo
  - representação de dados: ASCII, EBCDIC, UNICODE, *Little Endian*, *Big Endian*, etc.
  - acção em caso de erro: parar e dar erro, alertar operador, insistir, etc.

## Objectivos das Entradas/Saídas

- Estender o núcleo do sistema operativo permitindo a inclusão de periféricos
- Modelo de programação das Entradas/saídas uniforme, evitando que os programadores de aplicações tenham de lidar como detalhe dos diferentes periféricos
- Modelos de programação dos gestores de periféricos que simplifique a programação, a torne mais robusta e promova a reutilização

## Modelo Computacional das E/S

### Objectivos

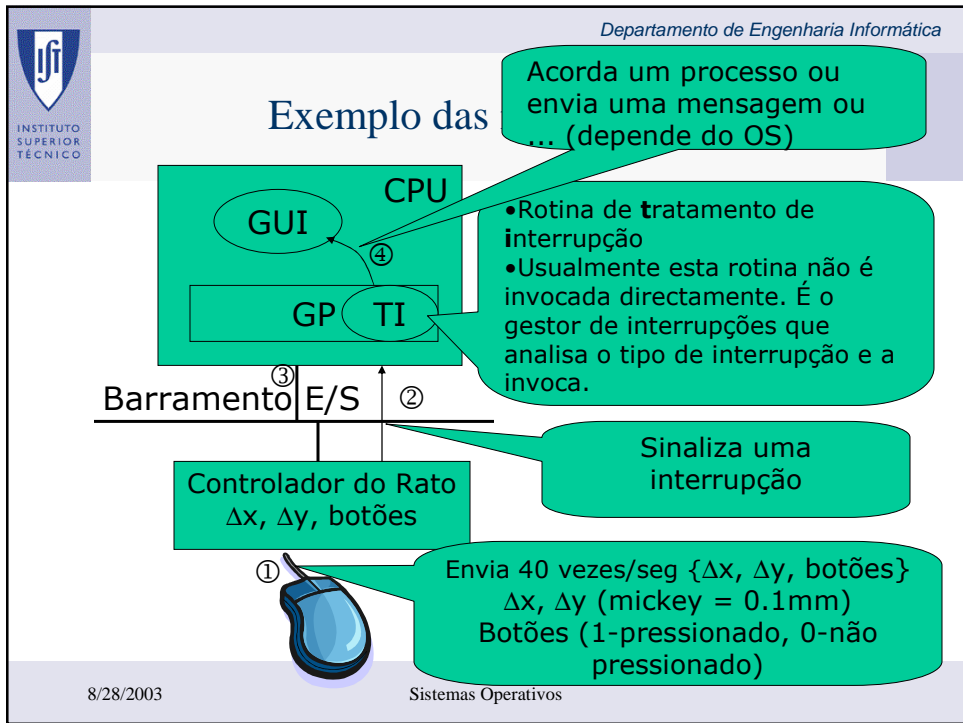
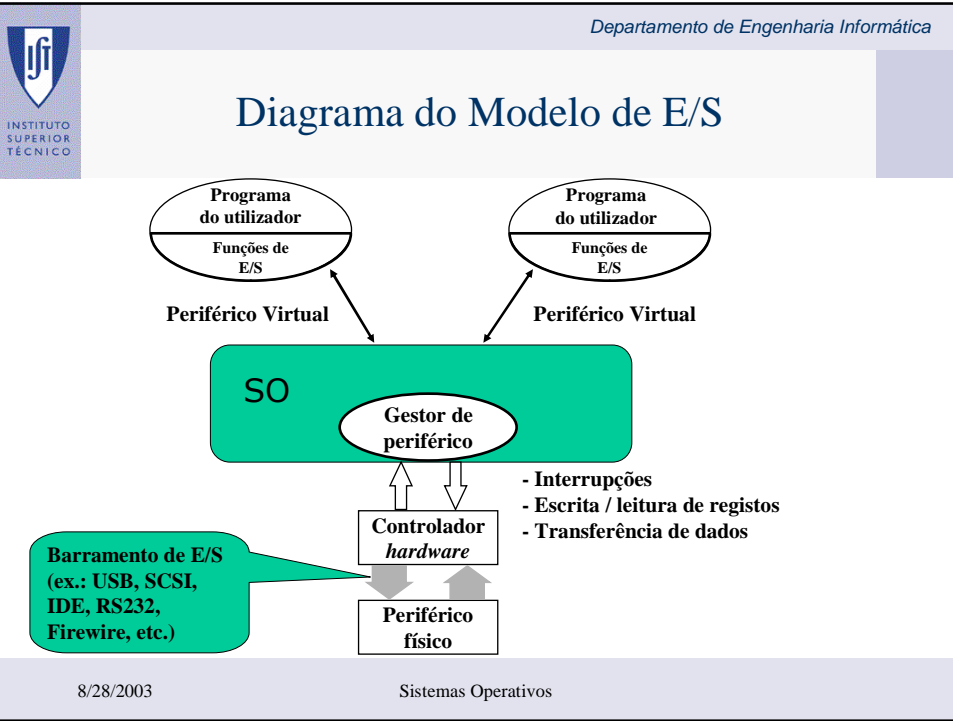
- Uniformização da designação.
- Uniformização do acesso.
- Independência do periférico do SO
  - não deve ser necessário redesenhar o núcleo para cada novo tipo de periférico.
- Redirecção das E/S.
- Adaptação automática a novos periféricos

### Soluções

- Conceito de periférico virtual
- Decomposição da interacção com os periféricos.
- Uniformização entre os mecanismos de comunicação e E/S
- Carregamento dinâmico de módulos no núcleo.

## Entidades do Modelo de E/S

- **Periféricos virtuais:**
  - entidades abstractas sobre as quais se realizam todas as operações de E/S - *files* (Multics, UNIX, Windows), *data sets* (IBM)
- **Funções de E/S:**
  - conjunto reduzido e uniforme de funções necessárias à interacção com qualquer periférico.
- **Gestores de periféricos (GP) (*device drivers* ou *device handlers*):**
  - componente do modelo (processo autónomo ou parte do programa do sistema operativo) que efectua a interacção real com o periférico



## Modelo Computacional

- O periférico virtual é idêntico a um canal de comunicação que é:
  - explicitamente estabelecido
  - sobre o qual são trocadas informações entre o processo utilizador e o gestor de periférico
- Associação ao periférico virtual:
  - estabelecimento de um canal de comunicação entre o processo e o gestor de periférico:
    - IdCanal = AbrirCanal ( Nome )
    - FecharCanal ( IdCanal )
- Designação dos periféricos:
  - varia consoante os sistemas
  - em Windows (NT/2000/XP) os nomes são mantidos por um gestor de nomes autónomo
  - em UNIX são identificados como ficheiros (especiais, tipo caracter ou bloco)

8/28/2003

Sistemas Operativos

## Modelo Computacional(cont.)

- função de E/S:
  - suporte de vários tipos de sincronização associados à operação
  - suporte de dois tipos de interface para a transferência de dados:
    - sequências de caracteres de dimensão variável (*byte stream*)
    - blocos de dados de tamanho fixo

NOME	PARÂMETROS	DESCRIÇÃO
Ler	(Fd, Tampão, bytes)	Lê do dispositivo para um tampão de memória
Escrever	(Fd, Tampão, bytes)	Escreve um tampão para o dispositivo
Posicionar	(Fd, Posição)	Posiciona o cursor de leitura ou escrita
LerAtributos	(Fd, Tampão)	Lê atributos do dispositivo
EscreverAtributos	(Fd, Atributos)	Modifica os atributos
LerDirectório	(Fd, Tampão)	Lê um directório para um tampão de memória
InfoPendente	(Fd)	Verifica se existe algo para fazer
HouveMudança	(dispositivo)	Verifica se o dispositivo mudou (ex. mudança de disquete)
Despeja	(Fd)	Assegura que todos os tampões internos são escritos
Cancela	(Fd)	Cancela a última operação pedida

8/28/2003

Sistemas Operativos

## Modelo Computacional(cont.)

- sincronização:
  - escrita: o processo cliente fica bloqueado até que os dados tenham sido transferidos para o periférico ou para estruturas de dados internas ao sistema operativo (cache)
  - leitura: o processo cliente fica normalmente bloqueado até que os dados pedidos lhe tenham sido transferidos; no entanto, há sistemas que permitem evitar tal bloqueio e efectuar leituras assíncronas à posteriori

## Utilização dos Periféricos

- As aplicações podem, se tiverem privilégios para tal, usar directamente os periféricos, abrindo o respectivo canal e efectuando as leituras e escritas necessárias
- Contudo, muitos periféricos são partilhados (ex.: discos, redes) razão pela qual não são utilizados directamente pelas aplicações mas acedidos através de outros módulos do sistema operativo

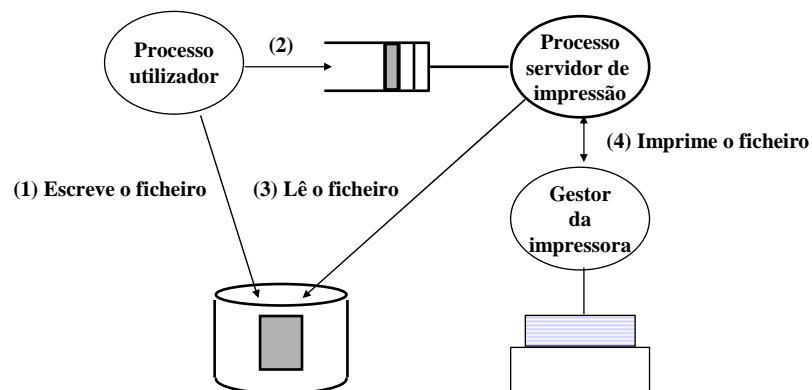
## Partilha de Periféricos

- Se o processo necessitar de utilização exclusiva de um periférico tal implica a abri-lo em exclusão mútua reservando a sua utilização:
  - bandas magnéticas
- Os processos usam os periféricos de memória de massa para guardar informação mas a sua utilização não obriga a abri-los porque são partilhados através do sistema de ficheiros
  - Discos, memory pen
- Alguns periféricos são usados em exclusividade durante um certo período de tempo por um processo, mas a optimização da gestão do periférico faz com que sejam controlados por um servidor como qual os restantes processos interactuam
  - impressoras

8/28/2003

Sistemas Operativos

## Partilha Indirecta de Periféricos de Utilização Exclusiva (*spooling*)



8/28/2003

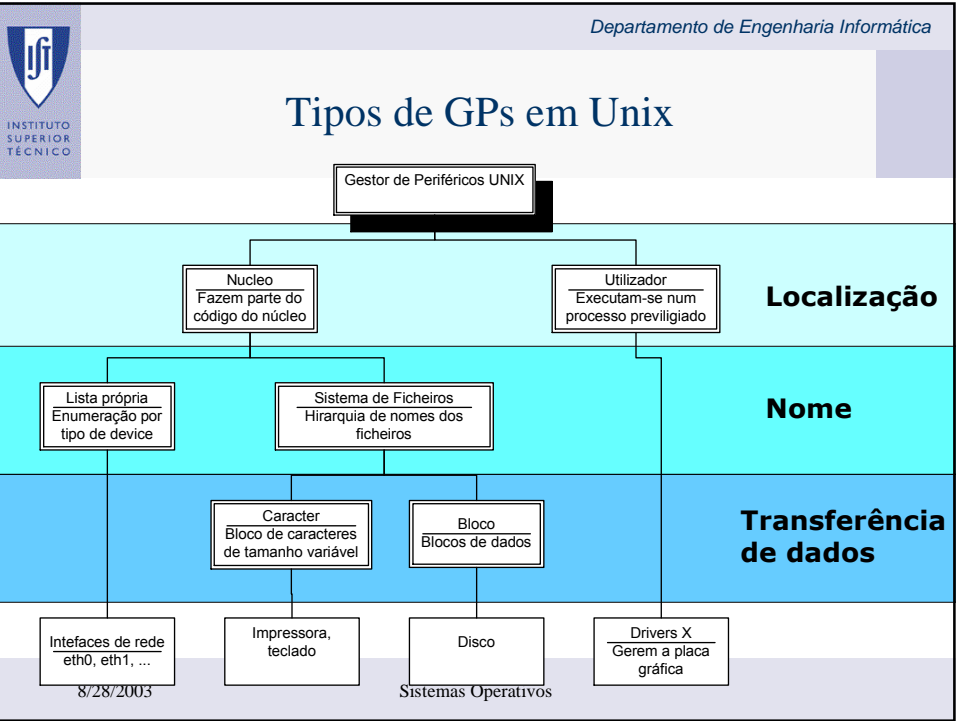
Sistemas Operativos

## Unix E/S

## Classificação - Localização

- A maioria dos GPs em Unix são módulos que fazem parte do núcleo.
  - São exceção os gestores de periférico da interface gráfica X
- Os gestores integrados no núcleo
  - Podem seguir o modelo computacional dos ficheiros, onde se designam como ficheiros e são manipulados essencialmente pelas mesmas funções
  - Podem ter um modelo próprio, estão neste caso os periféricos de controlo das redes





Departamento de Engenharia Informática

**Classificação – Espaço de Nomes**

- A maioria dos periféricos geridos por gestores no núcleo têm o seu nome no directório /dev
- As placas de rede têm um espaço de nomes próprio

```
# ifconfig
eth0

# ls -l /dev
crw-rw-rw- 1 root  root   5,  0 Aug 30  2001 tty
crw--w---- 1 root  root   4,  0 Mar 15 2002 tty0
crw----- 1 root  root   4,  1 Oct  6 14:42 tty1
crw--w---- 1 root  tty    4, 10 Aug 30  2001 tty10
crw--w---- 1 root  tty    4, 11 Aug 30  2001 tty11
...
brw-rw---- 1 root  floppy  2,  0 Aug 30  2001 fd0
brw-rw---- 1 root  floppy  2,  4 Aug 30  2001 fd0CompaQ
brw-rw---- 1 root  floppy  2,  4 Aug 30  2001 fd0d360
brw-rw---- 1 root  floppy  2, 12 Aug 30  2001 fd0D360
...
```

8/28/2003

Sistemas Operativos

## Periféricos no “/dev” (I)

- A comunicação faz-se através das mesmas primitivas utilizadas nos sistemas de ficheiros.
- Tal permite a redirecção fácil das E/S de um periférico para um ficheiro, e vice versa.

```
fd = open("/dev/tty0", O_RDWR);  
write(fd,&buffer,size);  
read(fd,&buffer,size);
```

## Periféricos no /dev (II)

- Cada gestor de periféricos pode gerir mais do que um periférico.
  - O gestor de periféricos para um terminal é igual para todos.
- Cada periférico é identificado por dois números
  - *Major number* – Identifica o gestor do periférico
  - *Minor number* – Identifica o periférico no domínio dos periféricos geridos pelo mesmo gestor.
- O nome do periférico só é importante para as aplicações não para o núcleo

## Periféricos no /dev (III)

```
#ls -l /dev
crw-rw-rw- 1 root  root    5,  0 Aug 30  2001 tty
crw--w---- 1 root  root    4,  0 Mar 15  2002 tty0
crw----- 1 root  root    4,  1 Oct  6 14:42 tty1
crw--w---- 1 root  tty     4, 10 Aug 30  2001 tty10
crw--w---- 1 root  tty     4, 11 Aug 30  2001 tty11
...
brw-rw---- 1 root  floppy  2,  0 Aug 30  2001 fd0
brw-rw---- 1 root  floppy  2,  4 Aug 30  2001 fd0CompaQ
brw-rw---- 1 root  floppy  2,  4 Aug 30  2001 fd0d360
brw-rw---- 1 root  floppy  2,  4 Aug 30  2001 fd0D360
...
```

### Major Numbers

- Majors diferentes = gestores diferentes.
- Majors iguais = mesmo gestor

### Minor Numbers

- Minors diferentes = periféricos diferentes
- Minors iguais e majors diferentes = periféricos diferentes
- Majors e Minors iguais = mesmo periférico

8/28/2003

Sistemas Operativos

## Periféricos no /dev (IV)

- Os periféricos podem ser divididos em
  - Tipo bloco.
  - Tipo caracter.
- Os periféricos de tipo bloco permitem a transferência de blocos de dados entre a memória e o periférico.
  - Discos, disquetes, etc.
- Os periféricos de tipo caracter transferem caracter a caracter.
  - Teclado, Écran, Impressora, rato, etc.

8/28/2003

Sistemas Operativos

## Instalação de um GP

1. Inserir o GP na tabelas do Kernel

```
root# insmod ./new-driver.o
```

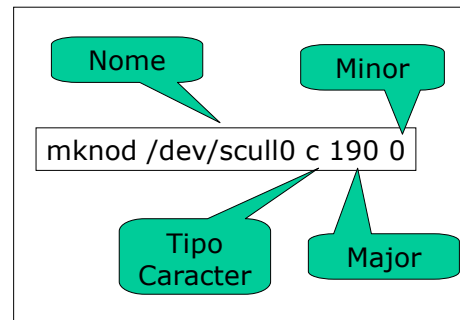
2. Obter o major em que ficou

```
#root ls /proc/devices
Character devices:
 1 mem
 2 pty
 3 tty
 4 ttyS
 5 cua
190 new-driver

Block devices:
 2 fd
 3 ide0
22 ide1
```

3. Criação de i-node

- Associa um nome a um dispositivo



8/28/2003

Sistemas Operativos

## Periféricos no /dev (V)

Modo carácter

```
#ls -l /dev
crw-rw-rw- 1 root  root   5,  0 Aug 30  2001 tty
crw--w---- 1 root  root   4,  0 Mar 15  2002 tty0
crw----- 1 root  root   4,  1 Oct  6 14:42 tty1
crw--w---- 1 root  tty    4, 10 Aug 30  2001 tty10
crw--w---- 1 root  tty    4, 11 Aug 30  2001 tty11
...
brw-rw---- 1 root  floppy  2,  0 Aug 30  2001 fd0
brw-rw---- 1 root  floppy  2,  4 Aug 30  2001 fd0CompaQ
brw-rw---- 1 root  floppy  2,  4 Aug 30  2001 fd0d360
brw-rw---- 1 root  floppy  2, 12 Aug 30  2001 fd0D360
...
```

Modo bloco

8/28/2003

## GPs de rede

- Não são acedidos directamente pelo utilizador.
- O utilizador acede a uma camada do protocolo (ex. Sockets INET) que passa a outra (ex.: TCP) e a outra (ex.: IP) e por fim ao GP.
- Todas as camadas estão no núcleo pelo que as funções do GP são acedidas directamente.

## Gestores de Periféricos fora do Núcleo

- O único caso notável que se executa num processo autónomo é a classe de gestores de placas gráficas utilizados pelo sistema X.
  - Estes gestores executam-se no mesmo processo que o servidor X.
  - O servidor X é um processo com privilégios de super-utilizador
  - A interface destes gestores é definida pelo servidor X, não pelo sistema operativo.

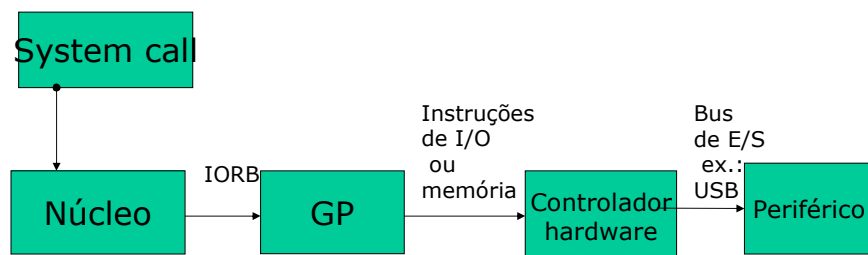
## Programação de GPs

Unix E/S  
Windows 2000 E/S

8/28/2003

Sistemas Operativos

## Arquitectura de comunicação no GP



8/28/2003

Sistemas Operativos

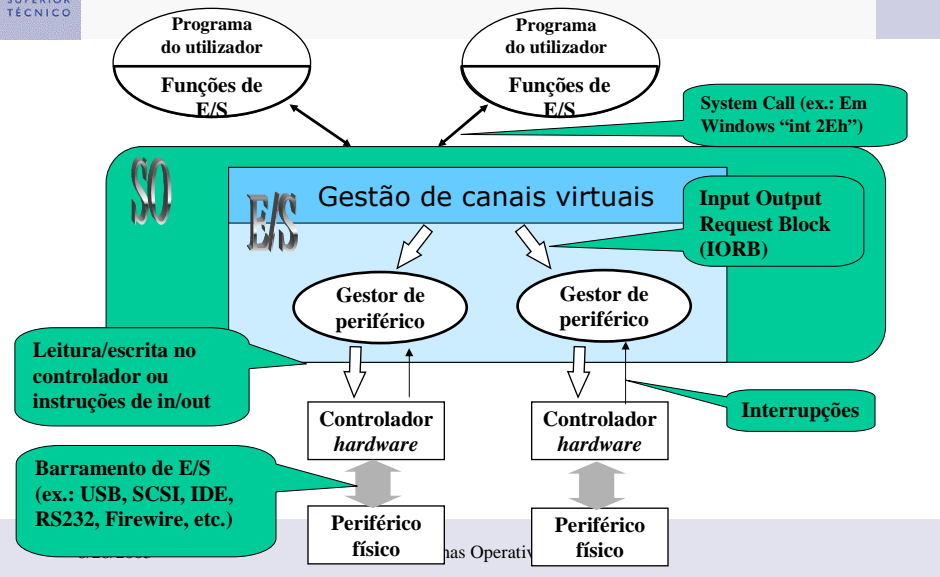
## Input/Output Request Block (IORB)

- os IORB são gerados pelas funções de E/S após a validação possível da coerência dos parâmetros:
  - validade do modo de operação pedido (leitura, escrita, etc.)
  - validade do modo de transmissão (cadeias de caracteres, blocos, etc.)
- um IORB é uma mensagem normalizada que transmite informação para um gestor de periférico:
  - o gestor está normalmente à espera dos IORB e, quando os recebe, desencadeia a operação pedida introduzindo a sequência de controlo adequada no controlador do periférico
  - a partilha de um periférico por vários utilizadores (externos ou módulos do núcleo) pode-se realizar através de uma fila de mensagens onde o gestor recebe os diversos IORB

8/28/2003

Sistemas Operativos

## Arquitectura de E/S



## Funções do GP

- esperar por um IORB
- validar os parâmetros dependentes do periférico
- programar o controlador do periférico de acordo com o pedido expresso no IORB
- esperar pela conclusão da operação e analisar ou processar o resultado da mesma:
  - normalmente a conclusão é assinalada por uma interrupção
  - no entanto, as rotinas de interrupção devem executar um conjunto reduzido de instruções durante o mais curto espaço de tempo possível
- tratamento das condições de erro
- se necessário transferir dados entre tampões próprios e o espaço de endereçamento do processo do utilizador
- avisar o processo utilizador da conclusão da operação

8/28/2003

Sistemas Operativos

## Programação de um GP

- **Armazenamento temporário de dados:**
  - conjunto fixo ou variável de tampões tendo em conta um fluxo médio expectável ou a optimização de transferências
  - regulação de fluxo
- **Interacção com a gestão de memória:**
  - fixação de páginas dos processos utilizadores na memória principal
  - utilização de endereços reais em vez de endereços virtuais
- **Rotina de interrupção:**
  - determinar a causa da interrupção
  - executar as operações necessárias
  - assinalar ao gestor o respectivo acontecimento (normalmente usando mecanismos especiais e não semáforos)

8/28/2003

Sistemas Operativos



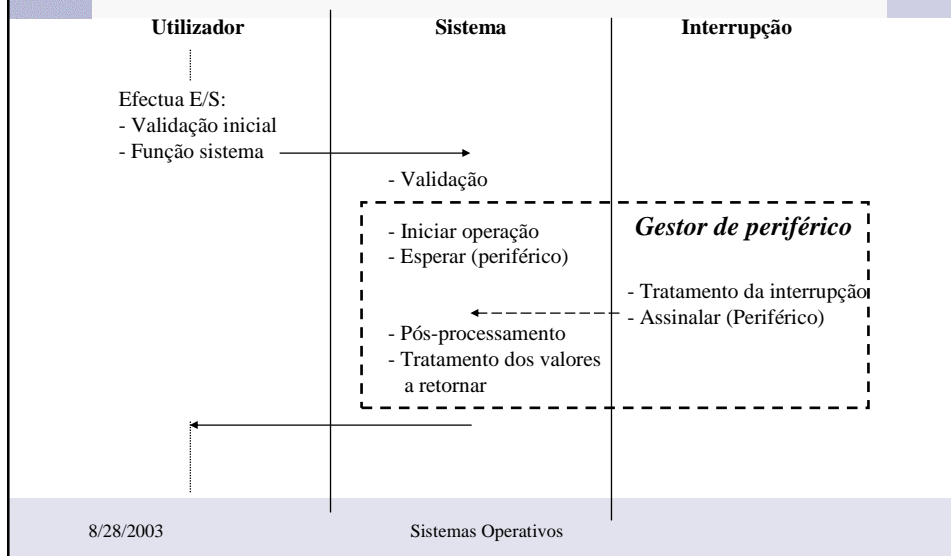
## GPs no núcleo ou fora do núcleo

- **Processo independente:**
  - maior flexibilidade (teoricamente).
  - necessidade de privilégios especiais que permitam ultrapassar as protecções habituais (alta prioridade, controlo de interrupções, gestão de memória, etc.).
  - isolamento de espaços de endereçamento entre os gestores e o núcleo do sistema operativo.
  - Ex.: servidor X na maioria dos sistemas Unix.
- **Integrado no núcleo :**
  - poupa o tempo gasto na comutação entre processos.
  - maior simplicidade e redução das operações do núcleo do sistema operativo.
  - Caso mais comum.

8/28/2003

Sistemas Operativos

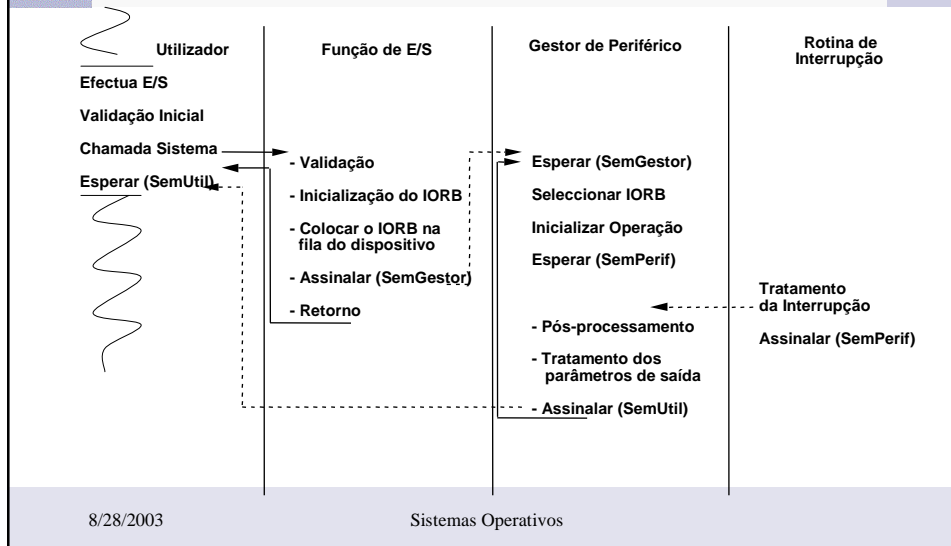
## GP Integrado no Núcleo



8/28/2003

Sistemas Operativos

## GP como Tarefa Independente



## Acesso do SO aos controladores

- Portabilidade do sistema operativo
  - Virtualização do hardware
  - Acesso ao Hardware (ex.: registos, portos, controladores, etc.) através de uma camada reduzida usualmente escrita em assembly.
- Em Windows chama-se HAL (Hardware Abstraction Layer)
- Em Linux, não tem nome mas existem um conjunto de funções que são de utilização recomendável e que efectuem essa abstracção.
- Estas camadas dificultam a utilização de potencialidades específicas dos processadores.

## Acesso do processador ao controlador

- **Instruções específicas de entradas saídas**
  - Alguns processadores possuem instruções específicas para acesso ao espaço de endereçamento de E/S
    - ex.: “in R1,300h” lê para R1 o valor no endereço 300h do espaço de endereçamento de E/S
  - Este acesso é ortogonal ao acesso a memória pelo que não interfere com os mecanismos de memória virtual.
  - Estas instruções não são normalmente geradas pelos compiladores de linguagens de alto nível pelo que é necessário programá-las em assembly.
- **E/S mapeadas em memória**
  - Memória do controlador mapeada no espaço de endereçamento físico do processador.
  - Nos sistemas operativos com memória virtual são necessárias funções que asseguram que determinados endereços virtuais correspondem a endereços físicos.
  - É necessário possuir mecanismos que indiquem que a memória mapeada nestes endereços não é passível de ser colocada em cache.
  - Mesmo em processadores com instruções específicas de entradas saídas este é o método mais utilizado para comunicação com os controladores.

8/28/2003

Sistemas Operativos

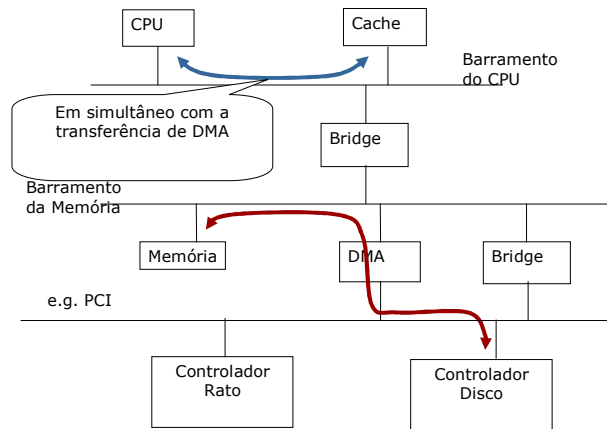
## Modos de transferência de dados

- **E/S programáticas**
  - O processador executa um conjunto de instruções do gestor de periféricos que escrevem/lêem para/do o periférico.
- **Direct Memory Access (DMA)**
  - Unidade de hardware especializada que transfere blocos de dados da memória para a memória do periférico.
  - Presente na maioria dos computadores.
  - Permite libertar o processador para computação interna e com acesso à cache, mas usualmente não permite o acesso à memória.
- **Processador de periféricos**
  - Processador especializado que partilha a memória com o processador principal e gere o acesso aos controladores.
  - Muito pouco comum (ex.: CDC6600)

8/28/2003

Sistemas Operativos

## Arquitectura do Computador com DMA



8/28/2003

Sistemas Operativos

## Unix E/S

Estruturas do núcleo

- /dev
- Rede

Sincronização

8/28/2003

Sistemas Operativos

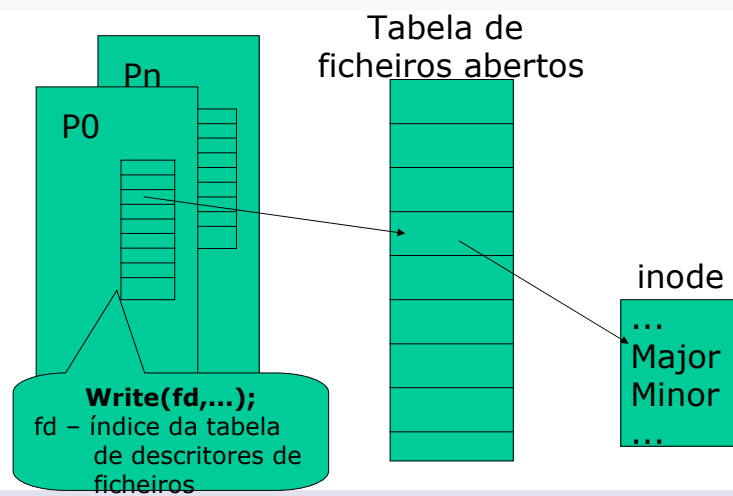
## Periféricos no /dev

- Realização de uma operação num periférico
  - Localização do periférico
  - Agulhagem da operação/interrupção
- Criação de um periférico

8/28/2003

Sistemas Operativos

## Localização do Major/Minor a partir do descritor de ficheiros



8/28/2003

Sistemas Operativos

## Localização do Major/Minor a partir do descritor de ficheiros

- A abertura de um periférico é feita com o comando open tal como para um ficheiro
  - Facilita o redirecionamento
- O método de abertura é semelhante
  - Depois de aberto a tabela de ficheiros abertos fica com uma nova entrada que aponta para um i-node.
- O i-node contém o major e o minor do periférico (veremos à frente como criar esse i-node especial).

8/28/2003

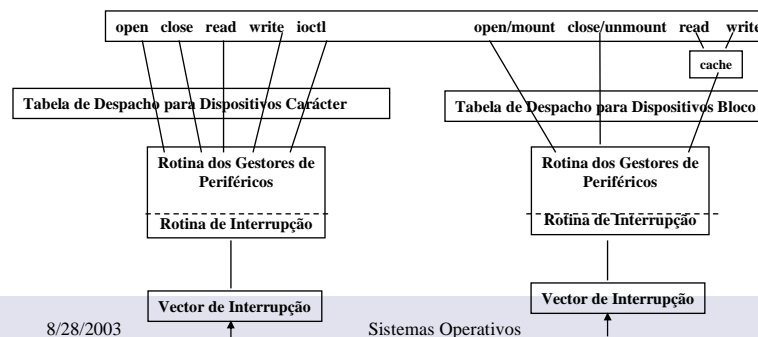
Sistemas Operativos

## Agulhagem das funções

**Tabela de Despacho**

Major N	open	close	read	write	ioctl
0	conopen	conclose	conread	conwrite	conioctl
1	syopen	nulldev	syread	sywrite	syioctl

**Arquitetura Interna das Entradas/Saídas**



8/28/2003

Sistemas Operativos

## Agulhagem das funções

- Existe uma tabela com todos os gestores de periféricos.
- Cada GP tem as suas funções específicas.
- Essas funções são apontadas pela tabela e indexadas pelo major.
- Com o major obtido no passo anterior é possível chamar a função respectiva e passar-lhe o minor.

## Criação de um GP

- Carregamento do módulo com o GP
- Registo das funções
- Registo da interrupção
- Reserva de memória

Operações  
realizadas pelo  
código do GP

## Carregamento de um módulo

- O linux suporta carregamento dinâmico de módulos:

```
#define MODULE
#include <linux/module.h>
int init_module(void) { printk("<1>Hello, world\n"); return 0; }
void cleanup_module(void) { printk("<1>Goodbye cruel world\n"); }
```

- Para carregar um módulo:

```
root# gcc -c hello.c
root# insmod ./hello.o
Hello, world
root# rmmod hello
Goodbye cruel world
root#
```

## Insmod

- Obtém a tabela de símbolos do Kernel para resolver as chamadas do módulo ao kernel (ex.. printk)
- Chama a função `sys_init_module` que:
  - Reserva memória para o módulo
  - Copia o código para a memória reservada
  - Resolve as chamadas ao kernel
  - Chama a rotina de inicialização do módulo (`int init_module(void)`)



## Registo das funções

```

struct file_operations scull_fops = {
    llseek:    scull_llseek,
    read:      scull_read,
    write:     scull_write,
    ioctl:     scull_ioctl,
    open:      scull_open,
    release:   scull_release,
};

int init_module(void) {
    ...
    /* Register your major, and accept a dynamic number. */
    result = register_chrdev(scull_major, "scull", &scull_fops);
    if (result < 0) {
        printk(KERN_WARNING "scull: can't get major %d\n", scull_major);
        return result;
    }
    if (scull_major == 0) scull_major = result; /* dynamic */
}

```

Funções definidas anteriormente

Chamada pelo sys\_init\_module

## Registo de interrupções

```

if (short_irq >= 0) {
    result = request_irq(short_irq, short_interrupt,
                        SA_INTERRUPT, "short", NULL);

    if (result) {
        printk(KERN_INFO "short: can't get assigned irq %i\n", short_irq);
        short_irq = -1;
    } else { /* actually enable it -- assume this *is* a parallel port
    */
        outb(0x10, short_base+2);
    }
}

```

ls /proc/interrupts

## Memória E/S (PCI/ISA mem)

- Dependendo da arquitectura o acesso ao *hardware* dos periféricos pode ser efectuado por:
  - Instruções de in e out específicas
  - Memória paginada
  - Memória não paginada
- Para garantir independência da arquitectura o acesso é feito através de funções específicas.

## Memória E/S (PCI/ISA mem)

```
if (check_mem_region(mem_addr, mem_size)) {  
    printk("drivername: memory already in use\n");  
    return -EBUSY;  
}  
request_mem_region(mem_addr, mem_size, "drivername");  
[...]  
unsigned readb(address);  
unsigned readw(address);  
unsigned readl(address);  
void writeb(unsigned value, address);  
void writew(unsigned value, address);  
void writel(unsigned value, address);  
[...]  
release_mem_region(mem_addr, mem_size);
```

## Sincronização I

- Activação/Desactivação de interrupções
  - Sti() / Cli()
  - Não aconselhável porque reduz o desempenho e em multiprocessadores não funciona
- Spinlock

```
spinlock_t my_lock = SPIN_LOCK_UNLOCKED;  
spin_lock_init(&my_lock);  
spin_unlock(spinlock_t *lock);  
[...] /* Outras variantes */
```

## Sincronização II

- Variáveis de sincronização

- Bits

```
/* try to set lock */  
while (test_and_set_bit(nr, addr) != 0)  
    wait_for_a_while();  
/* do your work */  
/* release lock, and check... */  
if (test_and_clear_bit(nr, addr) == 0)  
    something_went_wrong(); /*already released: error*/
```

- Inteiros

```
int atomic_inc_and_test(atomic_t *v);  
int atomic_dec_and_test(atomic_t *v);  
int atomic_add_and_test(int i, atomic_t *v);  
int atomic_sub_and_test(int i, atomic_t *v);
```