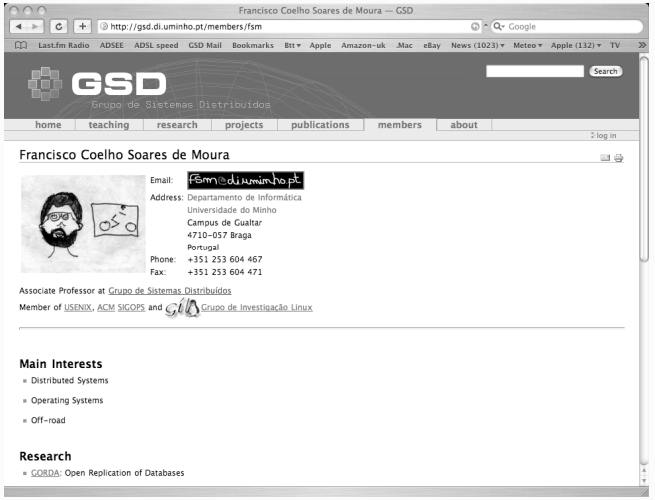
Sistemas Operativos

LEBIO

Grupo de Sistemas Distribuídos

http://gsd.di.uminho.pt



AVISOS!!!!

- Programa para hoje:
  - Apresentação da disciplina
  - Recapitulação de conceitos básicos:
    - Como funciona um computador
    - O que é um programa
  - Seguido de um pequeno “aquecimento” aos SO

As más notícias

- É uma cadeira de engenharia: temos de
  - Perceber os compromissos => usar a “massa cinzenta”
  - “Sujar as mãos” => programar, configurar, debug...
- Muitos alunos encravam em deficiências passadas, por exemplo dificuldade de raciocinar, concepção de algoritmos, programação, como funciona um computador...
- Esta cadeira engana muito: usar/”manusear” um SO não chega

As boas notícias

- Basta um pouco de estudo e bom senso para fazer a cadeira
- No final do semestre deverá ter ultrapassado grande parte das deficiências anteriores!

Organização das aulas

- Componente teórica:
  - "systems programming": ligação hw/so/sw
  - Interessa perceber os "porquês", não interessa decorar...
- Componente prática em Linux
- Parceria com a disciplina de Sistemas Distribuídos

Equipa Docente

- Responsável pela disciplina + aulas teóricas:
  - Francisco Soares de Moura (fsm@di.uminho.pt)
- Aulas práticas:
  - José Pedro Oliveira (jpo@di.uminho.pt)

SO e SD

- As disciplinas de SO e SD são vistas como complementares:
  - Nos conteúdos programáticos
  - Na equipa docente
  - Eventualmente na avaliação: trabalho prático comum

Programa

- Recapitulação de conceitos de programação de sistemas
- Noções de programação concorrente
- Gestão de processos, memória, ficheiros e periféricos
- Mãos na massa:
  - Aulas práticas em Linux
  - Operação+administração de sistemas linux
  - Programação de "baixo nível": C, syscalls, libs…

Bibliografia recomendada

- Sebenta de Sistemas Operativos (em construção…)
- A. Silberschatz et al., *Applied Operating System Concepts*, John Wiley & Sons, 2000.

OU

- A. S. Tanenbaum, *Modern Operating Systems*, 2nd edition Prentice Hall, 2001.

Bibliografia recomendada

- fsm 2004, ***Vou fazer Sistemas Operativos***
- www.google.com
  - Introduction to operating systems
  - Introduction to linux
  - How computers work

  …
- www.gildot.org, www.slashdot.org …

Bibliografia Adicional

- R. Stevens, *Advanced Programming in the Unix Environment*, Addison Wesley, 1990.
- Diversos artigos sobre sistemas operativos, a disponibilizar na página da cadeira ou a pesquisar na Internet.
- Manuais do sistema operativo, FAQs, código fonte do Linux, …

Transparências

- (Progressivamente) disponíveis em:

  http://gsd.di.uminho.pt/SOI/5305O3.html

- Baseadas nas transparências originais correspondentes aos livros recomendados
- Servem apenas como "âncora" ao estudo

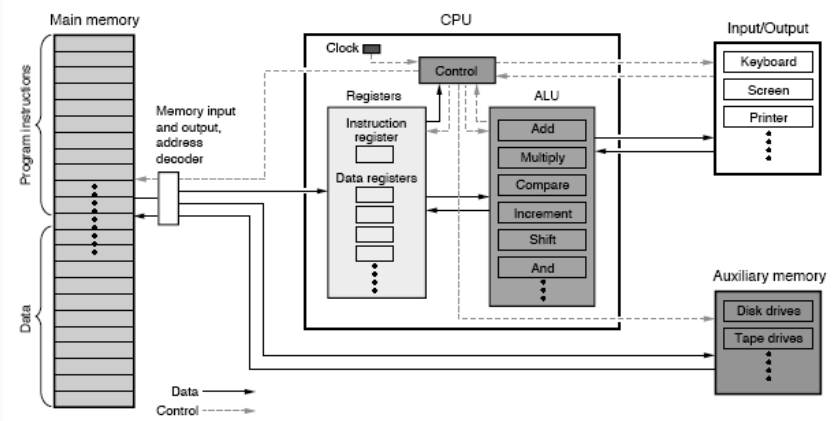
Avaliação

- 3 componentes, percentagens a definir brevemente
  - Exame final + prática em Linux + trabalho prático
- Exame cobre matéria teórica e prática
  - Código, pequenos programas
  - Valoriza-se a capacidade de raciocínio e a concepção de algoritmos (por oposição à utilização de "padrões" de soluções)
- Ninguém faz a disciplina apenas com a parte teórica
- Dificilmente a fará só com a prática

Programa

- Introdução
- Gestão de processos
- Noções de programação concorrente
- Gestão de memória
- Gestão de ficheiros
- Gestão de periféricos

O que é um computador?



O que é uma instrução (máquina) ?

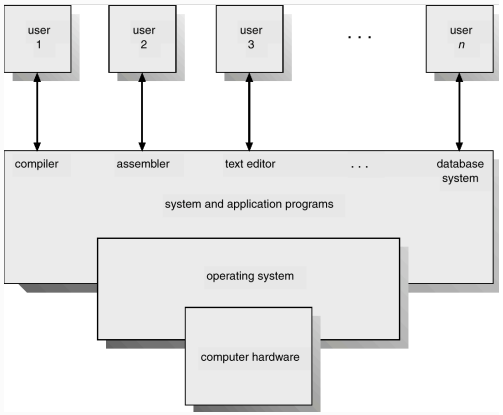
- LOAD A addr
- LOAD B addr
- SAVE A addr
- SAVE B addr
- LOADI A, val
- LOADI B, val
- ADD A, B
- SUB A, B
- MUL A, B
- CMP A, B
- JMP addr
- JE addr
- JNE addr
- JG addr
- ...

Para que serve um computador?

- Para facilitar a vida aos utilizadores
- Para executar programas (aplicações)

O que é um Sistema Operativo?

- Programa que actua como **intermediário** entre os utilizadores e o hardware

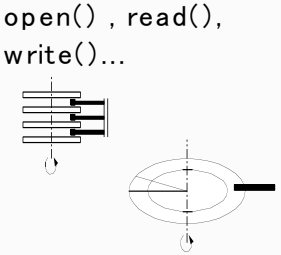


Portanto...

- SO deve colocar o hardware à disposição dos programas e utilizadores, mas de uma forma
  - **conveniente,**
  - **protegida,**
  - **eficiente**,
  - **justa,**
  - …

O Sistema Operativo pode ser visto como..

- Extensão da máquina, fornecedor de *máquina virtual*
- Gestor de recursos



Objectivos (1)

- Conveniência
  - SO esconde os detalhes do hardware

    e.g. dimensão e organização da memór
  - Simula máquina virtual com valor acrescentado

    e.g. cada *processo* executa numa "máquina" protegi
  - Fornece API mais fácil de usar do que o hardware

    e.g. ficheiros vs. blocos em disc

Na prática...

- É o Sistema Operativo quem define a "personalidade " de um computador
- Como se comporta o mesmo computador (hardware) após ter arrancado
  - MSDOS?
  - Windows 95?
  - WindowsXP?
  - Linux, Knoppix...?

Objectivos (2)

- Eficiência
  - SO controla a alocação de recursos
    - Se 3 programas usarem a impressora ao mesmo tempo → sai lixo?
    - Programa em ciclo infinito → computador bloqueia?
    - Processo corrompe a memória dos outros → programas morrem?
  - Multiplexação:
    - Tempo: cada processo usa o recurso à vez (impressora, CPU)
    - Espaço: recurso é partilhado (memória central, disco)

Objectivos (3)

- Recapitulemos então os objectivos gerais de um SO
  - Conveniência
  - Eficiência
- Os nossos critérios de avaliação serão portanto…
  -  Dá jeito?
  -  É eficiente ou aumenta a eficiência geral do sistema?
  -  Nem uma nem outra?

Evolução

- Sistemas de Computação
  - 1ª geração (1945/1955) – Válvulas e placas programáveis
  - 2ª geração (1955/1965) – Transistores e sistemas "batch"
  - 3ª geração (1965/1980) – ICs, Time-Sharing
  - 4ª geração (1980/) – PCs, Workstations, Servidores
  - ?? – PDAs, smartphones, GRID…

No início era assim...

- Acesso livre ao computador
  - Utilizador podia fazer tudo
  - Utilizador tinha de fazer tudo...
- Eficiência era baixa
  - Elevado tempo de preparação
  - Tempo "desperdiçado" com debug

E para aumentar a eficiência...

- Introduziu-se um operador especializado
  - Utilizador entrega fita perfurada ou cartões
  - Operador carrega o programa, executa-o e devolve os resultados
- Ganhou-se em eficiência, perdeu-se em conveniência
  - Operador é especialista em operação, não em programação
  - Pode haver escalonamento (i.e. alteração da ordem de execução)
  - Utilizador deixou de interagir com o seu programa

Melhor do que um operador...

- Só com um programa!
  - Controla a operação do computador
  - Encadeia "jobs", operador apenas carrega e descarrega
- Utilizadores devem usar rotinas de IO do sistema (embora ainda possam escrever as suas)

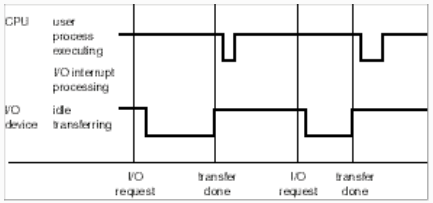
Embrião de um sistema operativo

Mas havia o risco de...

- Se perder eficiência devido a erros de programação
  - Ciclos infinitos
  - Erros na leitura ou escrita de periféricos
  - Programa do utilizador destruir o "programa de controle"
  - Espera por periféricos lentos

Soluções (hardware)

- Interrupções



- Relógio de Tempo Virtual
- Instruções privilegiadas, 2 ou mais modos de execução
- Protecção de memória

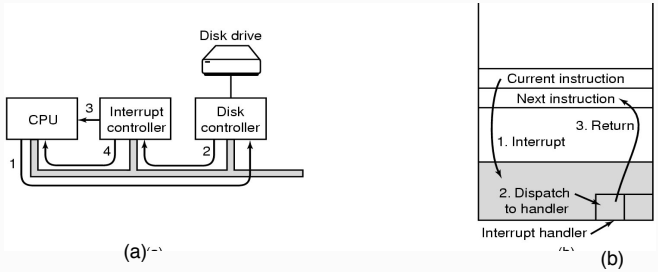
Exemplo: Polling IO

- Disk_IO()
  - Carrega o controlador de disco com parâmetros adequados (pista, sector, endereço de memória, direcção...)
  - While (NOT IO_done) /* do nothing*/

    (Equivalente a Já acabaste? Já acabaste? Já acabaste? Já acabaste? Já acabaste? Já acabaste? Já acabaste? Já acabaste? Já acabaste? Já acabaste? Já acabaste? Já acabaste? Já acabaste? Já acabaste? Já acabaste?...)
  - OK, regressa de disk_io()

Resulta em *desperdício de tempo de CPU*

Exemplo: Interrupt-driven IO

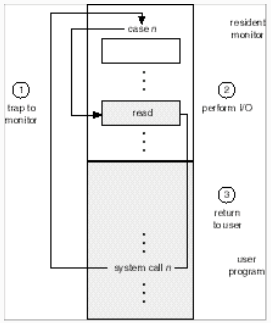


(a) OS inicia operação de IO e prepara-se para receber a interrupção

(b) No fim da operação de IO, o programa em execução é interrompido momentaneamente, trata-se o evento, e continua a execução

Soluções (software)

- Chamadas ao Sistema
- Virtualização de periféricos
- Multiprogramação

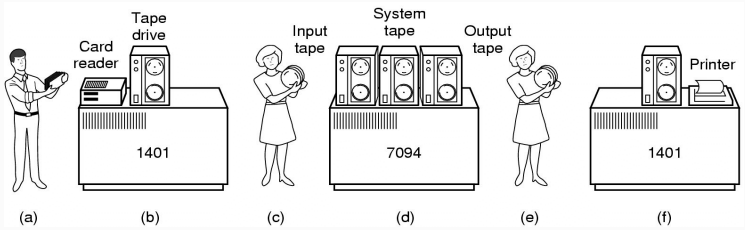


Antes de continuar…

- Assegure-se que percebeu os conceitos anteriores, e que entendeu os problemas que as soluções indicadas procuram resolver…

- Por exemplo,
  - sabe mesmo o que são e para que servem os 2 modos de execução?
  - modo de execução é hardware ou software?
  - e multiprogramação? Multiprocessamento?
  - o que é o tempo virtual?

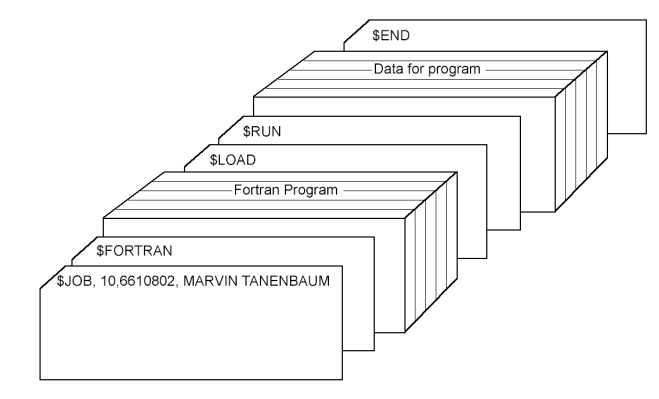
Primeiros sistemas de batch



Processador auxiliar faz IO de periféricos lentos (virtuais)

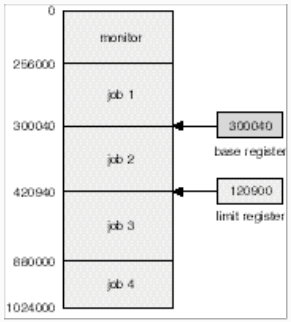
- Carregar cartões no 1401, que os copia para banda magnética
- Colocar banda no 7094 e executar os programas
- Recolher banda com resultados e colocá-la no 1401, que os envia para a impressora

Exemplo de um "job"

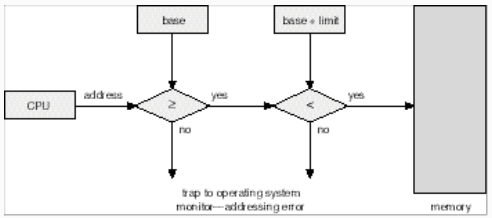


Multiprogramação

Vários jobs são carregados para memória central, e o tempo de CPU é repartido por eles.

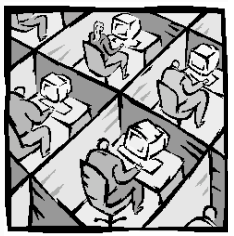


Protecção de memória



- Note que estes testes têm de ser feitos sempre que há um acesso à memória...
- 2, 3 ou mesmo 4 vezes por instrução?

E a conveniência?



- Teve de esperar pelos sistemas de

  Time-Sharing

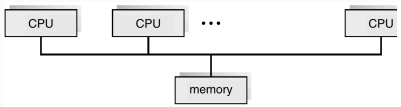
- Terminais (consolas) ligados ao computador central permitem que os utilizadores voltem a interagir directamente
- Sistema Operativo reparte o tempo de CPU pelos vários programas prontos a executar

E desde aí?

- Com o computador pessoal volta tudo ao início...
  - Control Program for Microcomputers
  - Monoprogramação, baixa eficiência...
- Mas...
  - É muito conveniente para o utilizador
  - É barato, logo eficiência não é a prioridade

Multiprocessamento (1)

- Vantagens
  - *throughput*
  - economia
  - *graceful degradation*
  - ...



Exemplo: com 2 CPUs

- A ideia é executar o dobro da carga no mesmo intervalo de tempo (i.e. maior throughput)
- não é executar um programa mais depressa (i.e. baixar temp de resposta). Para isso necessitaria de paralelizar a aplicaçã dividi-la em vários processos

Multiprocessamento (2)

- Arquitectura
  - Simétrico
    - Qualquer CPU pode executar código do SO, mas
      - cuidado com *race conditions*, (e.g. tabela de blocos de memória livres)
      - hardware mais sofisticado (e.g disco interrompe todos os CPUs?)
  - Assimétrico
    - Periféricos associados a um só CPU, o que executa o SO
      - Não há *races,* mas os outros CPUs podem estar parados porque esse não "despacha" depressa,
      - nesse caso o *throughput* diminui

Sistemas Distribuídos (1)

- Nos anos 80 apareceram as redes locais para partilha de
  - recursos caros (e.g. impressoras) ou
  - inconvenientes de replicar (e.g. sistemas de ficheiros)
  - redirecionamento de IO

Exemplo: cat fich.txt | rsh print_server l

- Questões
  - protocolos de comunicação, modelo cliente-servidor?
  - como saber o estado de recursos remotos?

Sistemas Distribuídos (2)

- Actualmente
  - passou-se dos *network aware OSs* para sistemas que estão vocacionados para o trabalho em rede
  - as aplicações podem localizar e aceder recursos remotos de uma forma transparente
- E chegou-se à Web...

E ainda...

- SOs para *mainframes*:
  - IBM MVS, IBM VM/CMS.
  - desenvolvidos nos anos 60 e ainda em operação (z/VM)!

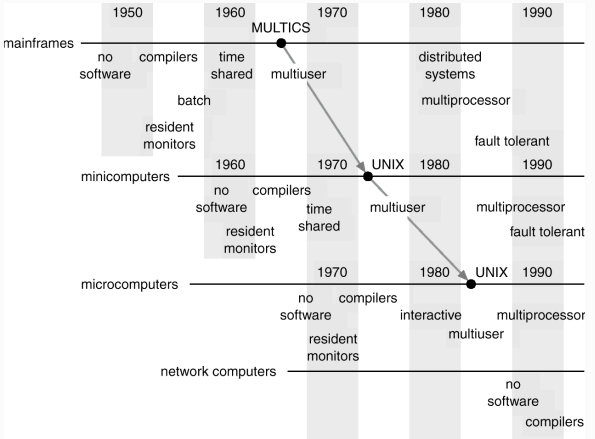
– Actualmente a virtualização é *HOT TOPIC* (vmware, …)

E ainda...

- SO de Tempo Real
  - controlo de processos industriais, sistemas de vôo, automóveis, máquinas de lavar, etc.
  - SO normais não conseguem dar garantias de tempo de resposta.

- SOs para computadores "restritos":
  - smartcards, PDAs, telemóveis, sensores...

Evolução de conceitos de SO

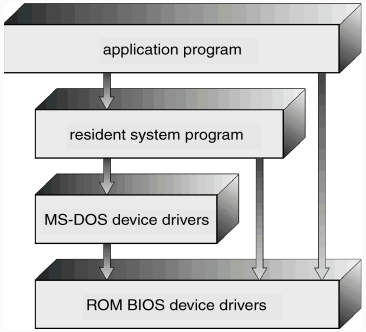


Arquitectura de Sistemas Operativos

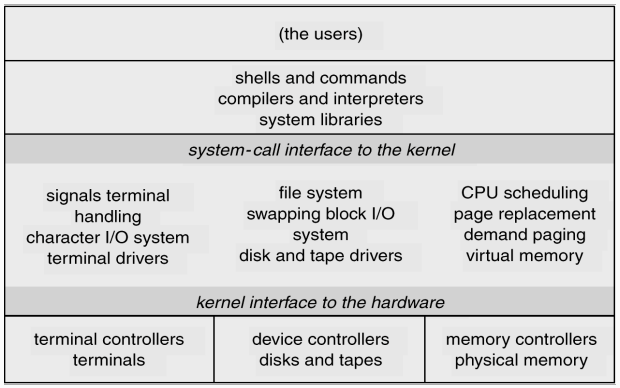
- Alguns exemplos
  - Sistemas monolíticos
  - Sistemas em camadas, hierárquicos
  - Modelo cliente-servidor
  - Máquinas virtuais

  …

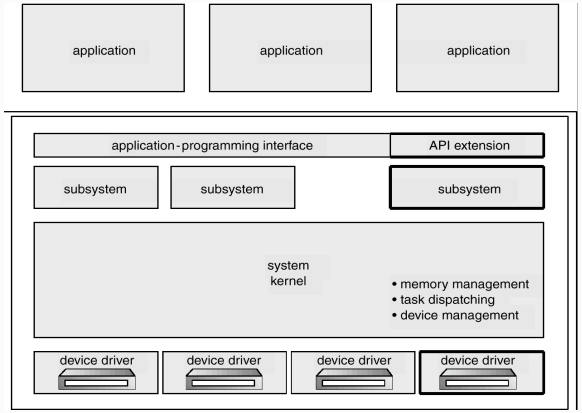
MS-DOS Layer Structure



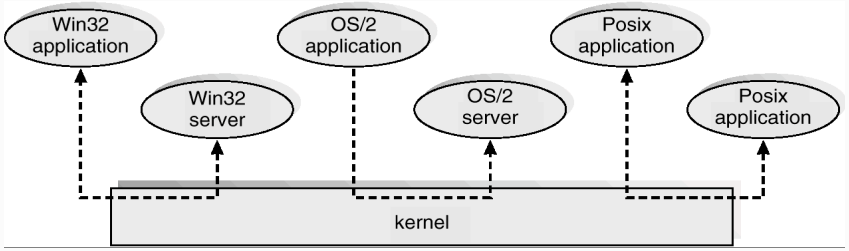
UNIX System Structure



OS/2 Layer Structure



Windows NT (cliente-servidor)



E nas nossas aulas?

- O nosso SO é bastante modular
  - Módulo de gestão de processos
  - Módulo de gestão de memória
  - Módulo de gestão de periféricos
  - Módulo de gestão de ficheiros (2º semestre)
- Mas há hierarquia / interdependência:
  - e.g. Memória virtual / memória real / disco / processos

Agora que já sabemos

- Para que serve um sistema operativo
- Quais os objectivos de um sistema operativo
- E começamos a saber:
  - como é um sistema operativo → estrutura interna, algoritmos, ...
  - e os porquês de ser assim

    → que benefícios/objectivos se pretendem alcançar com determinadas estratégias

    → em que circunstâncias não se pode fazer melhor

Convinha garantir que...

- Sabemos de facto
  - "Como é" um programa (e porquê?)
  - "Como é" um computador (e porquê?)
- Ou seja,
  - perceber as razões para o hardware e software de sistemas serem como são

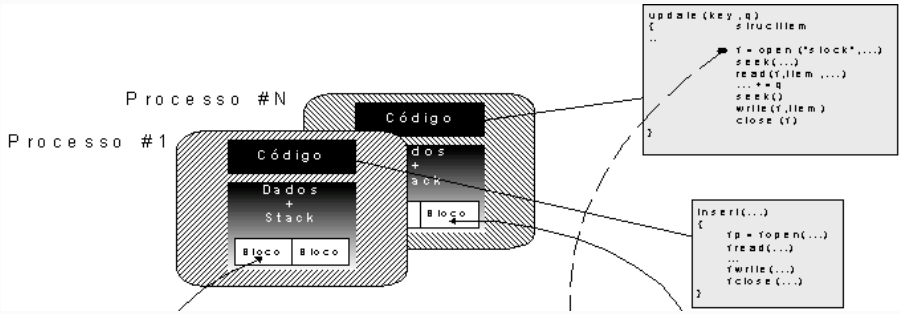
O que é/como é um programa/processo?

- Programa <u>executável</u>:
  - Resultado da compilação, ligação, (re)colocação em memória
  - Normalmente dependerá de módulos externos, libs
- Processo <u>em execução:</u>
  - código já (re)colocado em memória central + dados +stack
  - Estruturas de gestão:
    - Processo: contexto, recursos HW e SO em uso (registos, ficheiros abertos...
    - Utilizador (uid, gid, account...)

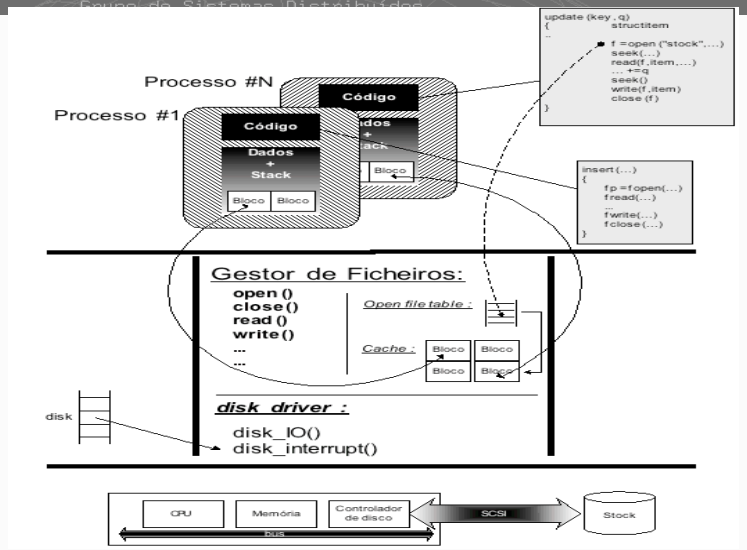
O que é/como é um computador?

- CPU
  - Registos (PC, SP, BP, CS, DS...) → "contexto volátil"
  - Instruções privilegiadas → só podem ser executadas em modo "protegido"; a forma de um programa do utilizador solicitar serviços ao SO é através das chamadas ao sistema (syscalls)
- Memória (mas o que é um endereço? E modos de endereçamento?)
- Periféricos + formas de dialogar com eles
- Interrupções (já agora, recordemos traps e excepções!)

The big picture



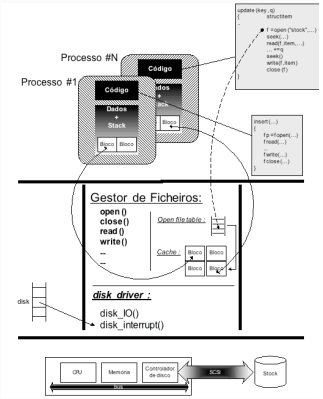
Dois programas a acederem simultaneamente ao mesmo fichei ou base de dados



The big picture revisited

- Assegure-se que percebeu
  - Como surgem as "race conditions"
    - entre processos
    - dentro do SO
  - Vantagens/desvantagens do uso de caches

Note que estamos a falar de caches por software, cópias de dados em memória mas acessíveis em contextos diferentes



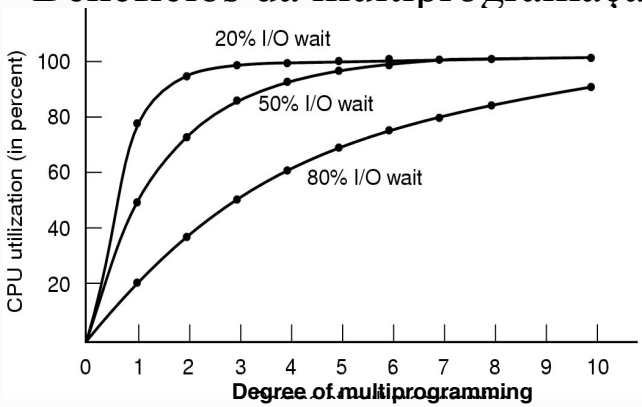
Programa

- Introdução
- Gestão de processos
- Noções de programação concorrente
- Gestão de memória
- Gestão de periféricos

Porquê criar vários processos?

- Porque dá jeito... + conveniência
  - Estruturação dos programas
  - Para não estar à espera (spooling, background...)
  - Multiplas actividades / janelas
- Porque é melhor + eficiência
  - Múltiplos CPUs
  - Aumenta a utilização de recursos (e.g multiprogramação)

Benefícios da multiprogramação



Processos

- Processo: um programa em execução, tem actividade própria
- Programa: entidade *estática*, Processo: entidade *dinâmica*
- Duas invocações do mesmo programa resultam em dois processos diferentes (e.g. vários utilizadores a usarem cada um a sua shell, o vi, browser, etc.)

Processos

- O contexto de execução de um processo (i.e. o seu estado) compreende:
  - código
  - dados (variáveis globais, *heap*, *stack*)
  - estado do processador (registos)
  - ficheiros abertos,
  - tempo de CPU consumido, …

Exemplo de informação sobre um processo

Process management	**Memory management**	**File management**
Registers	Pointer to text segment	Root directory
Program counter	Pointer to data segment	Working directory
Program status word	Pointer to stack segment	File descriptors
Stack pointer		User ID
Process state		Group ID
Priority		
Scheduling parameters		
Process ID		
Parent process		
Process group		
Signals		
Time when process started		
CPU time used		
Children's CPU time		
Time of next alarm		

Processos

- O SO deverá ser capaz de:
  - Criar, suspender e reiniciar a execução de processos
  - Suportar a comunicação entre processos
- O próprio SO tem muitos processos "do sistema"

Processos

- Para poderem executar os seus programas, os processos requerem tempo de CPU, memória, utilização de dispositivos…
- Por outras palavras, os processos

  COMPETEM POR RECURSOS
- E cabe ao sistema operativo fazer o escalonamento dos processos, i.e. atribuir os recursos pela ordem correspondente às políticas de escalonamento

Políticas de escalonamento

- Qual a melhor?
- E a resposta é...
  - Depende!
    - De quem responde, utilizador ou administrador?
- É preciso definir OBJECTIVOS

Objectivos

- Conveniência
  - Justiça
  - Redução dos tempos de resposta
  - Previsibilidade

    ...
- Eficiência
  - Débito (*throughput*), transacções por segundo, ...
  - Maximização da utilização de CPU e outros recursos
  - Favorecer processos “bem comportados”, etc.

Critérios de escalonamento

- IO-bound ou CPU-bound
- Interactivo ou não (batch, background)
- Urgência de resposta (e.g. tempo real)
- Comportamento recente (utilização de memória, CPU)
- Necessidade de periféricos especiais
- PAGOU para ir à frente dos outros...

Estados de um processo (i)

Processos em Unix

- Para criar um novo processo:
  - **fork**: cria um novo processo (a chamada ao sistema retorna "duas vezes", uma para o pai e outra para o filho)
  - A partir daqui, ambos executam o mesmo programa
- Para executar outro programa
  - **exec**: substitui o programa do processo corrente por um novo programa
- Para terminar a execução
  - **exit**

Compare o **exec** com a invocaçã de uma função: são muito diferentes

Relação entre processos

- Possibilidades na execução dos filhos:
  - Pai e filho executam concorrentemente
  - Pai aguarda pelo fim da execução do filho para continuar
- Possibilidades no espaço de endereçamento:
  - O do filho é uma duplicação do do pai
  - O do filho é um programa diferente desde a criação

fork/exec

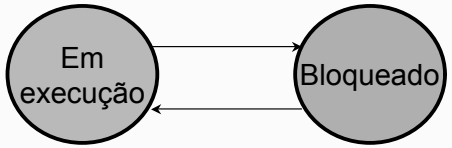
```
pid = fork()

if (pid == 0) {
        /* Sou o filho */
        exec( novo programa )
else {
        /* Sou o pai
           A Identificação do meu filho é colocada na variavel  pid
        */
}
```

fork'ing e exec'ing

- O padrão fork/exec é muito frequente (e.g. shell)
- Optimizações (a rever no capítulo de gestão de memória):
  - **copy on write**
  - Variante: **vfork**, não duplica o espaço de endereçamento; ambos os processos partilham o espaço de endereçamento e o pai é bloqueado até o filho terminar ou invocar o exec

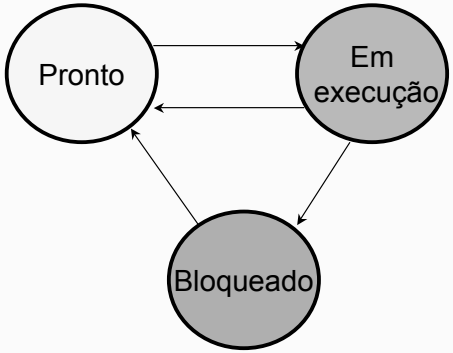
Estados de um processo (ii)

Podemos para já admitir que durante a sua "vida" os processos passam por 2 estados:



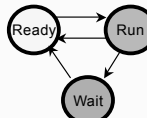
Estados de um processo (iii)

- Na prática, há mais processos não bloqueados do que CPUs
- Surge uma fila de espera com processos **Prontos a executar**
- Processos em execução podem ser desafectados

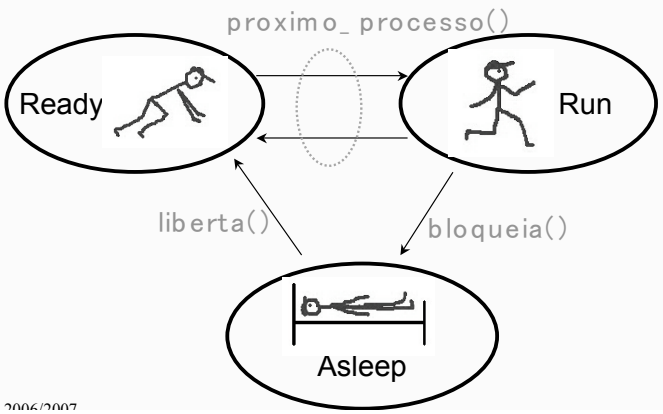


Estados de um processo (iv)

- Em execução
  - Foi-lhe atribuído o/um CPU, executa o programa correspondente
- Bloqueado
  - O processo está logicamente impedido de prosseguir, e.g. porque lhe falta um recurso ou espera por evento
  - Do ponto de vista do SO, é uma transição VOLUNTÁRIA!
- Pronto a executar, aguarda escalonamento

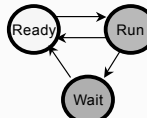


Primitivas de despacho (i)



Primitivas de despacho (ii)

- Bloqueia(evento)
  - Coloca processo corrente na fila de processos parados à espera deste "evento"
  - Invoca próximo_processo()
- Liberta(evento) ou liberta(processo,evento)
  - Se o outro processo não está à espera de mais nenhum evento, então coloca-o na lista de processos prontos a executar
  - Nesta altura pode invocar ou não próximo_processo()



Primitivas de despacho (iii)

- Proximo_processo()
  - Selecciona um dos processos existentes na lista de processos prontos a executar, de acordo com a política de escalonamento
  - Executa a comutação de contexto
    - Salvaguarda contexto volátil do processo corrente
    - Carrega contexto do processo escolhido e regressa (executa o return)

Como o Stack Pointer foi mudado, "regressa" para o processo escolhido!

Principais decisões

- Qual o próximo processo?
- Quando começa a executar?
- Durante quanto tempo?

- Por outras palavras,

Há desafectação forçada ou não

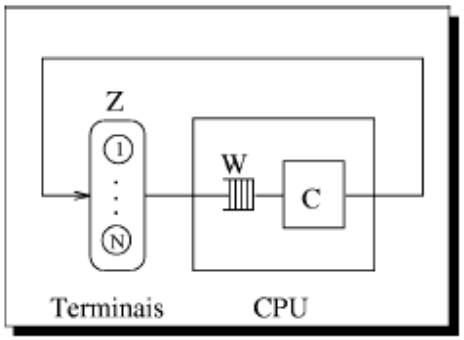
Escalonamento de processos

- Quando, uma vez atribuído a um processo, o CPU nunca lhe é retirado então diz-se que o escalonamento é **cooperativo** (non-preemptive).
  - Exemplos: Windows 3.1, co-rotinas, thread_yield()
- Quando o CPU pode ser retirado a um processo ao fim do quantum ou porque surgiu outro de maior prioridade diz-se que o escalonamento é com **desafectação forçada** (preemptive)

Escalonamento de processos

- Escalonamento **cooperativo** (non-preemptive).
  - "poor man´s approach to multitasking" ?
  - Sensível às variações de carga
- Escalonamento com **desafectação forçada**
  - Sistema "responde" melhor
  - Mas a comutação de contexto tem overhead

Modelo de sistema interactivo



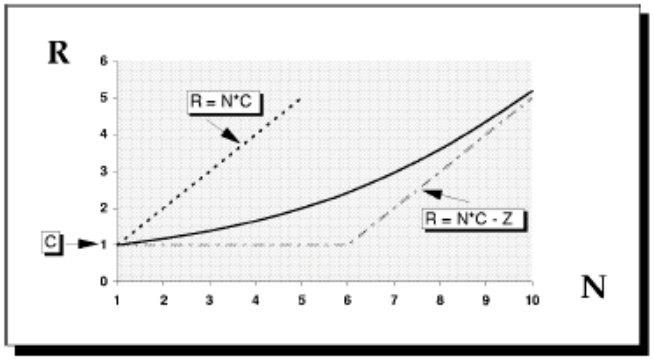
Z = Think time

C = Service time

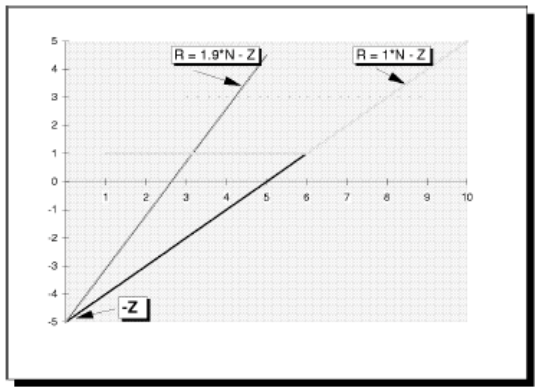
W = Wait time

N = Number of users

Tempo de Resposta (carga homogénea)



Tempo de Resposta (carga heterogénea)



- Assuma-se agora que uma em cada 10 interacções é muito longa, 10 vezes maior
- Veja-se a degradação de tempos de resposta

Tempo de Resposta (carga heterogénea)

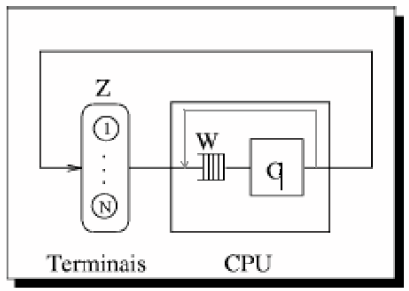
- Para evitar que as interações longas monopolizem o CPU e aumentem o tempo de resposta das restantes deve usar-se desafectação forçada.
- Neste caso deve atribuir-se um quantum (ou time slice) para permitir a troca rápida de processos:
  - Interacções curtas terminam dentro dessa fatia de tempo, logo não são afectadas pela política de desafectação.
  - Interacções longas executam durante um quantum e a seguir o processo correspondente regressa ao estado de Pronto a Executar, dando a vez a outros processos. Mais tarde ser-lhe-á atribuído nova fatia de tempo, e sucessivamente até a interacção terminar.

Duração da fatia de tempo

- Maioria das interacções deve "caber" num quantum

  $R = W + C$
- Se precisar de 2 passagens pelo CPU, $T_{Resposta}$ é quase o dobro!

  $R = W + q + W + c'$



Escalonamento de processos

- Escalonadores de longo-prazo (segundos, minutos) e de curto-prazo (milisegundos)
- Processo CPU-bound: processo que faz pouco I/O mas que requer muito processamento
- Processo I/O-bound: processo que está frequentemente à espera de I/O.

Escalonamento de processos

- Os processos prontos são seriados numa fila (*ready list*)
- A lista é uma lista ligada de apontadores para PCB's
- A lista poderá estar ordenada por prioridades de forma a dar um tratamento preferencial aos processos com maior prioridade

Escalonamento de processos

- Quando um processo é escalonado, é retirado da *ready list* e posto a executar
- O processo pode "perder" o CPU por várias razões:
  - Aparece um processo com maior prioridade
  - Pedido de I/O (passa ao estado de bloqueado)
  - O *quantum* expira (passa ao estado de pronto)

Escalonamento de processos

- Pretende-se maximizar a utilização do CPU tendo em atenção outras coisas importantes:
  - Tempo de resposta para aplicações interactivas
  - Utilização de dispositivos de I/O
  - Justiça na distribuição do tempo de CPU

Escalonamento de processos

- A decisão de escalonar um processo pode ser tomada em diversas alturas:
  - Qdo um processo passa de a-executar a bloqueado
  - Qdo um processo passa de a-executar a pronto
  - Qdo se completa uma operação de I/O
  - Qdo um processo termina

Escalonamento de processos

- Diferentes algoritmos de escalonamento favorecem optimizações diferentes:
  - Tempo de resposta
  - Máxima utilização do CPU utilization

Escalonamento de processos

- Alguns algoritmos de escalonamento:
  - FCFS (First Come, First Served)
  - SJF (Shortest Job First)
  - SRTF (Shortest Remaining Time First)
  - Preemptive Priority Scheduling
  - RR (Round Robin)

First Come, First Served (FCFS)

- A *ready list* é uma fila FIFO
- O processos são colocados no fim da fila e selecionado o da frente
- Método cooperativo
- Nada apropriado para ambientes interactivos

FCFS

- Tempo de espera com grandes flutuações dependendo da ordem de chegada e das características dos processos
- Sujeito ao “efeito de comboio”
- Uma vantagem óbvia do FCFS é sua simplicidade de implementação
- Parece haver vantagens em escalonar os processos mais curtos à frente…

SJF (Shortest Job First)

- A ideia é escalonar sempre o processo mais curto primeiro
- Possibilidades:
  - Desafectação forçada (SRTF) - interrompe o processo em execução se aparecer um mais curto
  - Cooperativo – aguardar pela terminação do processo em execução mesmo na presença de um processo recente mais curto

SJF

- Não se consegue adivinhar o tempo de processamento dos processos
- Apenas se podem fazer estimativas
- Usa uma combinação de tempos reais e suas estimativas para fazer futuras previsões.

Preemptive Priority

- Associa uma *prioridade* (geralmente um inteiro) a cada processo.
- A *ready queue* é uma fila seriada por prioridades.
- Escalona sempre o processo na frente da fila.
- Se aparece um processo com maior prioridade do que o que está a executar faz a troca dos processos

Preemptive Priority

- Problema: starvation
- Uma solução: envelhecimento – aumenta a prioridade dos processos pouco a pouco de forma a que inevitavelmente executem e terminem.

RR (Round Robin)

- Dá a cada processo um intervalo de tempo fixo de CPU de cada vez
- Quando um processo esgota o seu quanto retira-o do CPU e volta a colocá-lo no fim da fila.
- Ignorando os overheads do escalonamento, cada um dos *n* processos CPU-bound terá (1/*n*) do tempo disponível de CPU

RR

- Se o quantum for (muito) grande o RR tende a comportar-se como o FCFS
- Se o quantum for (muito) pequeno então o overhead de mudanças de contexto tende a dominar degradando os níveis de utilização de CPU
- Tem um tempo de resposta melhor que o SJF (o quantum "é" normalmente o SJ)

Avaliação de algoritmos

- **Modelo determinístico**
  - Definição da carga tipo: ordem de chegada dos processos, tempos de execução, distribuição CPU/IO, etc e avaliação analítica do desempenho do algoritmo.
  - Vantagem: simples
  - Desvantagem: o ajuste dos resultados depende directamente dos dados de entrada. São necessários vários cenários para se poder generalizar os resultados.

Avaliação de algoritmos

- **Teoria de filas de espera**
  - Definição de um modelo matemático do sistema e avaliar segundo a teoria das filas de espera.
  - Desvantagem: para que o modelo seja tractável é normalmente necessário fazer muitas simplificações que nem sempre são razoáveis na prática.

Avaliação de algoritmos

- **Simulação**
  - Escrever/adaptar/usar um programa que modele o sistema (dispositivos de IO, CPU, etc. em software) e analisar o desempenho do algoritmo de escalonamento através de medidas directas de utilização e tempo de resposta
  - Tentar obter traços do comportamento de sistemas reais
  - Desvantagem: tempos de execução longos

Programa

- Introdução
- Gestão de processos
- Noções de programação concorrente
- Gestão de memória
- Gestão de periféricos

Programação Concorrente

- A possibilidade de execução "simultânea" leva ao acesso em concorrência a recursos partilhados.
- O acesso concorrente pode ser feito a zonas de endereçamento partilhadas ou a (genericamente) ficheiros.
- O acesso concorrente pode facilmente resultar na incoerência dos dados partilhados.

Programação Concorrente

- Para garantir a **coerência** dos dados é necessário que os processos **cooperem** e acedam **ordenadamente** aos recursos partilhados.
- O SO fornece um conjunto de mecanismos que permitem aos processos **sincronizarem-se** e controlarem a ordem de acesso aos recursos partilhados

Exemplo: BA

BARMAN

```
while (n_copos ==
   MAX_BALCAO)
   /* aguarda por vaga no
   balcão*/;
n_copos = n_copos + 1;
pousar_copo_no_balcao();
….
```

CLIENTE

```
while (n_copos == 0)
   /* aguarda por copo
   cheio */;
n_copos = n_copos – 1;
tirar_copo_do_balcao();
…
```


Regiões críticas

- Para um dado recurso partilhado, cada processo "declara" as regiões do seu código que acedem ao recurso como **regiões críticas**.
- A execução de uma região crítica (relativa a um recurso partilhado X) por parte de um processo está dependente do processo receber garantias de que nenhum outro processo executará a sua região crítica (relativa tb. a X).

Regiões críticas

- Com mais rigor, deverá ser assegurado que:
  - Não podem estar dois processos a executar as suas regiõe críticas.
  - Todo o processo que o pretenda deverá inevitavelmente poder executar a sua região crítica.

Regiões críticas

- Formas "pouco interessantes" de implementar regiões críticas:
  - Inibição de interrupções
  - Variáveis de guarda
  - Alternância estrita
  - Algoritmo de Peterson
  - Test-And-Set

Exemplo: BA c/ alternância estrita

BARMAN

```
while (1) {
   while (vez != BARMAN)
   /* aguarda vez para por copo
   no balcão*/;
   pousar_copo_no_balcao();
   vez = CLIENTE;
}
```

CLIENTE

```
while (1) {
   while (vez != CLIENTE)
   /* aguarda vez para beber
   */;
   tirar_copo_do_balcao();
   vez = BARMAN;
}
```


Exemplo: Algoritmo de Peterson

```
int vez;
int interessado[2];
entrar_regiao_critica(int processo)
{
    int outro;
    outro = 1 – processo;
    interessado[processo] = 1;
    vez = processo;
    while (vez == processo && interessado[outro])
      /* espera que o outro saia da região crítica */;
}
```

```
sair_regiao_critica(int processo)
{
    interessado[processo] = 0;
}
```

Atenção à inversão de prioridades

Regiões críticas

- Formas interessantes de implementar regiões críticas – primitivas de comunicação entre processos:
  - Sleep / Wakeup
  - Semáforos
  - Contagem de eventos
  - Monitores
  - Mensagens

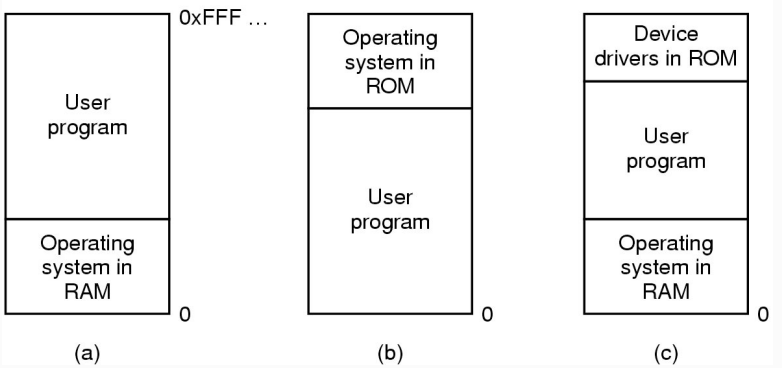
Programa

- Introdução
- Gestão de processos
- Noções de programação concorrente
- Gestão de memória
- Gestão de periféricos

Gestão de Memória

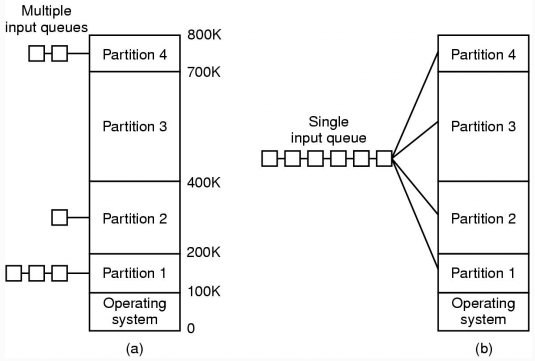
- Idealmente a memória seria:
  - grande
  - rápida
  - não volátil
- Hierarquia da memória
  - Pouca memória rápida, cara – cache
  - Velocidade média, custo aceitável – memória principal
  - Gigabytes de memória lenta, discos baratos
- O gestor de memória gere esta hierarquia da memória

Monoprogramação



Três formas de organizar a memória com SO e apenas um processo

Multiprogramação c/ partições fixas



Filas separadas para cada partição Fila única

Partições

- Em princípio, a fila única será mais eficiente porque não mantém processos à espera de serem carregados para memória quando há partições disponíveis, mas...
- Embora para efeitos de protecção baste mudar os registos que marcam os limites inferior e superior da partição,
- Os endereços terão de mudar se o programa for "swapped out" "swapped in" para outra partição.

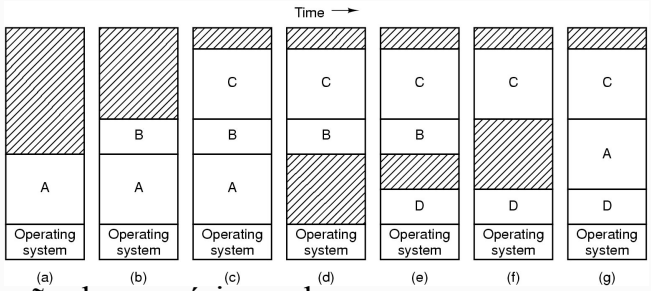
Recolocação e Protecção

- Incerteza sobre o endereço de carregamento do programa
  - Endereços de variáveis e funções não pode ser absoluto
  - Um processo não se pode sobrepor a outro processo
- Solução: uso de valores de base e limite
  - Endereços adicionados à base para obter endereços físicos
  - Endereços superiores ao limite são erros

Partições de dimensão variável

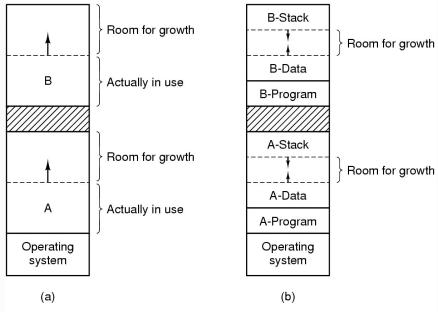
- Havendo suporte de hardware para recolocação dinâmica, pode-se passar para um número (variável) de partições de memória com dimensão variável
- A dimensão da partição é estabelecida quando o programa é carregado
- Conduz a algoritmos de "alocação": first-fit, best-fit, worst-fit, buddy-system...

Swapping



- A alocação de memória muda com:
  - Processos que são carregados
  - Processos que são libertados

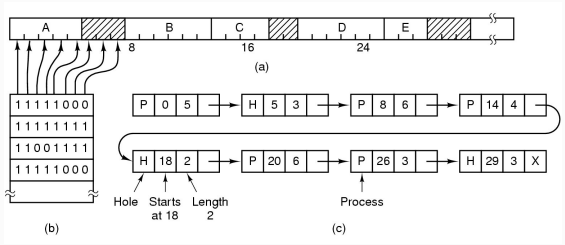
Desperdício de espaço



- Alocação para segmento de dados crescente
- Alocação para segmentos de dados e stack crescentes

Gestão de memória

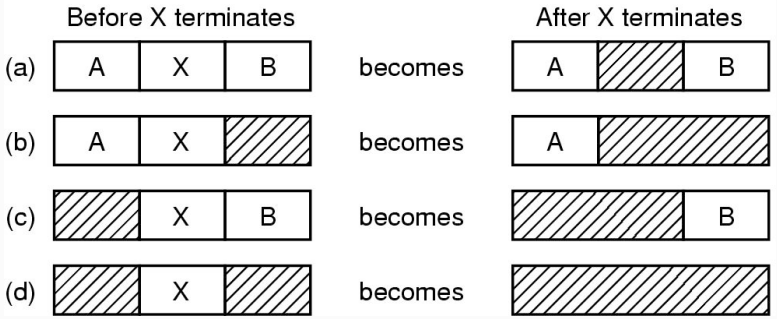
- Zona de memória com 5 processos e 3 espaços livres



(b) Bitmap correspondente

(c) Mesma informação de uma lista ligada

Gestão de memória listas ligadas

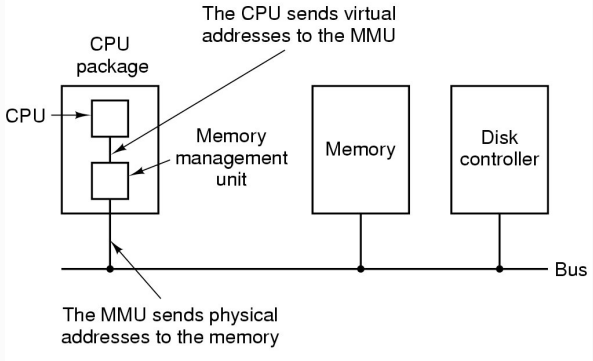


É complicado juntar espaços livres!

Alguns problemas com partições

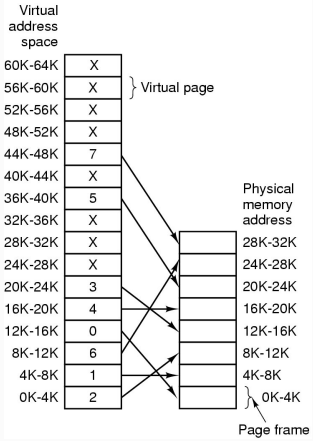
- Dimensão máxima dos programas diminui; Pode obrigar a overlays
- Desperdício com fragmentação interna e/ou externa
- Desperdício devido à dispersão de referências, estática e dinâmica
- Desperdício porque não consegue partilhar (porque não sabe onde começa/acaba o código)
- Desperdício de CPU devido a algoritmos de gestão complicados
- Protecção "tudo ou nada"; não distingue código, dados e stack

Memória Virtual

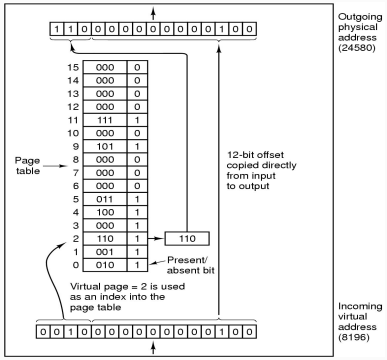


Memória Virtual

A relação entre endereços virtuais e físicos é dada por uma tabela

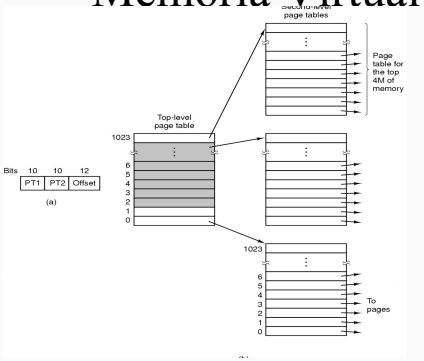


Memória Virtual



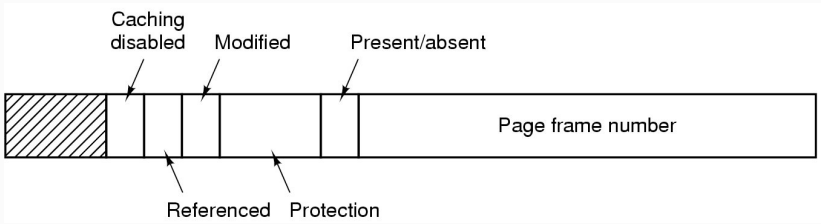
Operação da MMU com 16 páginas de 4 KB

Memória Virtual



- Endereço de 32 bit c/ 2 campos para tabelas de páginas
- Tabelas de páginas de 2 níveis

Memória Virtual



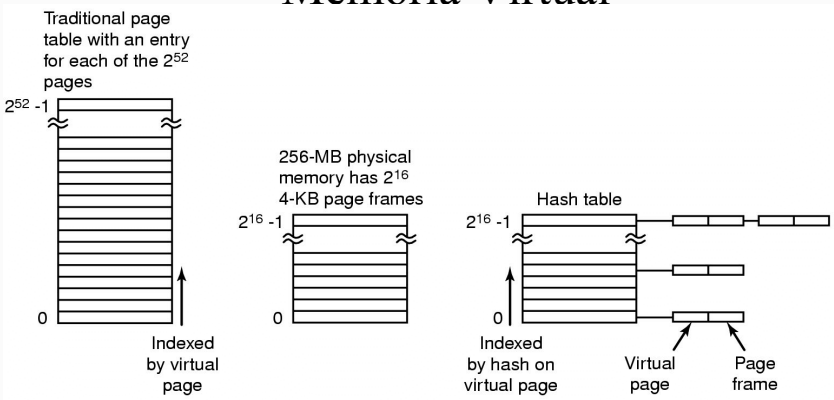
Entrada típica da tabela de páginas

Memória Virtual

- TLBs – Translation Lookaside Buffers – para melhorar o des

Valid	Virtual page	Modified	Protection	Page frame
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

Memória Virtual



- Comparação entre tabelas tradicionais e **tabelas invertidas**

Rejeição de páginas

- Um *page fault* leva:
  - A decidir que página em memória rejeitar
  - A criar espaço para uma nova página
- Uma página modificada tem que ser escrita
  - Uma não modificada é logo libertada
- Convém não rejeitar uma página frequentemente usada
  - Pois provavelmente terá de ser carregada a seguir

Rejeição de páginas

- Rejeitar a página que será usada mais tarde
  - Inexequível
- Aproximado por estimativa
  - Histórico de execuções anteriores do processo
  - Também isto é impraticável

Rejeição de páginas FIFO

- Mantém uma lista das páginas em memória
  - Segundo a ordem em que foram carregadas
- A página no topo da lista é rejeitada
- Desvantagem
  - A página há mais tempo em memória poderá ser a mais usada

Rejeição de páginas LRU

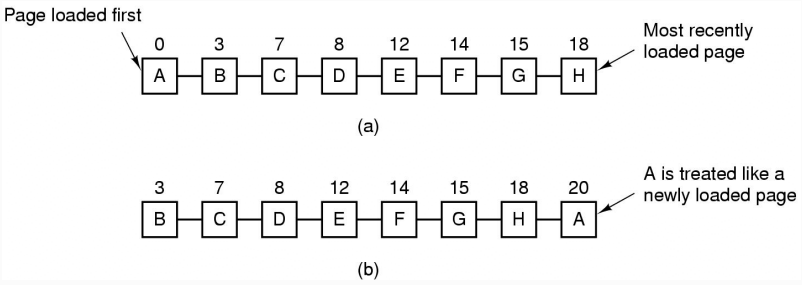
- Assume que as páginas usadas recentemente serão usadas em breve
  - Rejeita a página que não foi usada há mais tempo
- Tem que gerir uma lista de páginas
  - Ordenada pela mais recente
  - Actualizada em todos os acessos à memória!
- A alternativamente seria manter um contador em cada entrada da tabela de páginas
  - Escolhe a página com o menor valor
  - Periodicamente teria de colocar o contador a zero

Rejeição de páginas NRU

- Cada página tem 1 bit de acesso e 1 de escrita
- As páginas são assim classificadas:
  1. Não acedida, não modificada
  2. Não acedida, modificada
  3. Acedida, não modificada
  4. Acedida, modificada

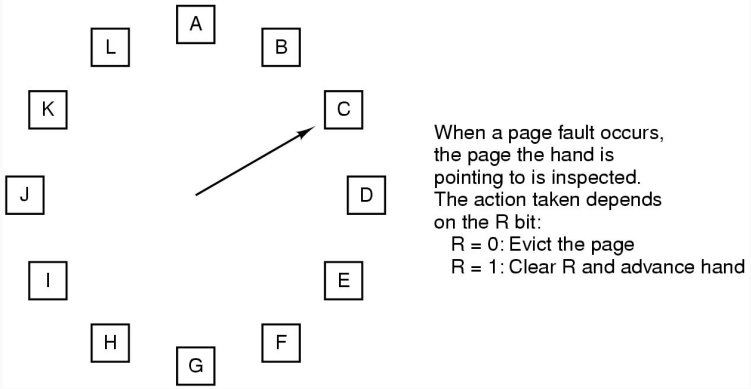
NRU remove a página com menor "ranking"

Segunda Oportunidade

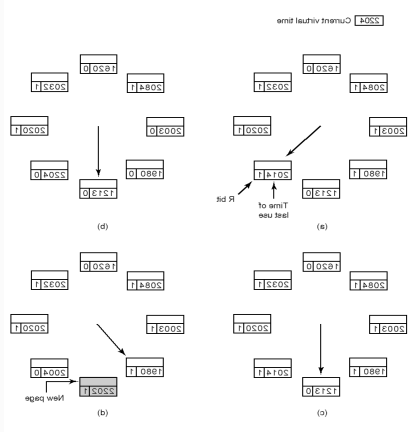


- Ordem FIFO
- Se a página mais antiga tiver sido acedida, não é rejeitada
- É limpo o bit de acesso e é colocada no fim da fila

Rejeição de páginas: Relógio



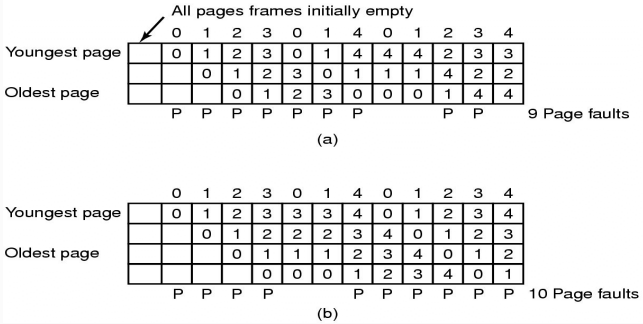
WorkingSetClock



Rejeição de páginas

Algoritmo	Características
Óptimo	Inexequível. Padrão para comparação.
NRU (não usado recentemente)	Aproximação grosseira.
FIFO	Leva à rejeição de páginas importantes.
Segunda Oportunidade	Melhoramento do FIFO.
Relógio	Solução realista.
LRU (menos recentemente usado)	Muito bom. Implementação exacta difícil.
NFU (menos frequentemente usado)	Aproximação grosseira do LRU.
Aging (envelhecimento)	Aproximação boa e eficiente do LRU.
Working set	Implementação ineficiente.
WSClock	Aproximação boa e eficiente.

Anomalia de Beladv

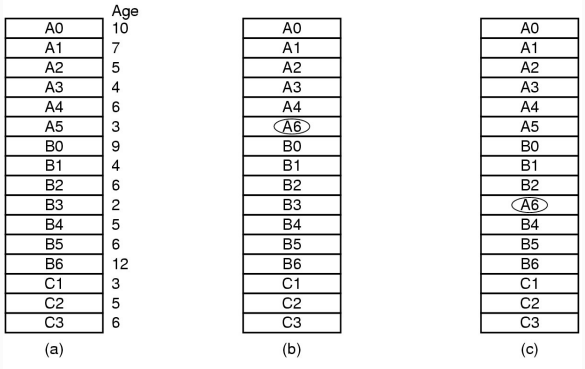


- FIFO com 3 page frames
- FIFO com 4 page frames
- Os P's indicam ocorrência de page faults

Sistemas Paginados

- Aspectos de Concepção de
  - Alocação local e global
  - Controlo de carga / thrashing
  - Tamanho das página

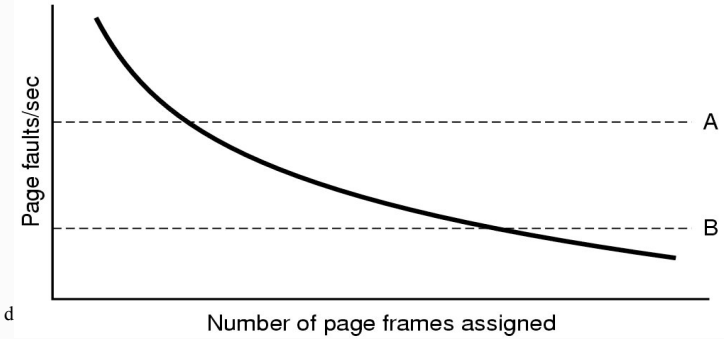
Alocacão Local e Global



Config. original Subst. local Subst. global

Alocação Local e Global

- Alocação de páginas Local e Global



Controlo de carga / thrashing

- Apesar de um bom desenho, pode ainda ocorrer thrashing
- Quando a Frequência de Page Faults indica que:
  - Alguns processos precisam de mais memória
  - Mas nenhum pode ceder da que tem
- Solução :
  Reduzir o número de processos que competem por memória
  - Passar um ou mais processos para disco e dividir as páginas que lhes estavam atribuídas
  - Rever o grau de multiprogramação

Tamanho das páginas

Páginas pequenas

- Vantagens
  - Menos fragmentação interna
  - Melhor adequação a várias estruturas de dados e código
  - Menos partes de programas não usados em memória
- Desvantagens
  - Mais páginas, tabelas de páginas maiores

Tamanho das páginas

- Overhead devido às tabelas e à fragmentação interna

$$overhead = \frac{s \cdot e}{p} + \frac{p}{2}$$

Espaço da tabela de páginas

Fragmentação interna

- Em que
  - s = tamanho médio dos processos em bytes
  - p = tamanho das páginas
  - e = entrada na tabela de páginas

Valor óptimo quando

$$p = \sqrt{2se}$$

Aspectos de Implementação

O SO intervem 4 vezes na paginação:

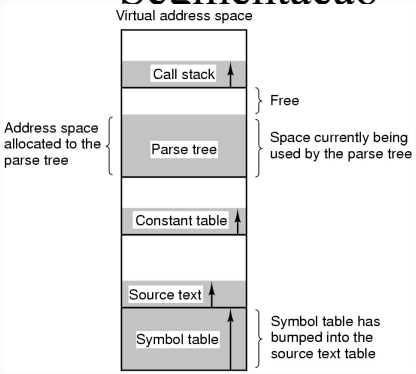
1. Criação do processo
   - Determinar o tamanho do programa
   - Criar a tabela de páginas
2. Execução do processo
   - Re-inicializar a MMU para o novo processo
   - Limpar a TLB
3. Na Page Fault
   - Determinar o endereço virtual causador da page fault
   - Colocar a página em memória (criar espaço na pool)
4. Fim da execução do processo
   - Libertar a tabela de páginas e as páginas associadas

Aspectos de Implementação

Tratamento da Page Fault

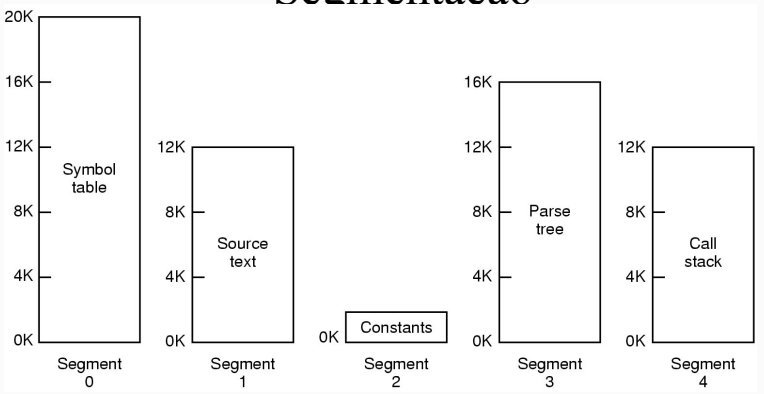
- O hardware interrompe o kernel
- São salvaguardados os registos
- SO determina a página virtual necessária
- SO valida endereço e procura page frame
- Se a página foi alterada, escreve-a para disco

Segmentacão



- Espaço de endereçamento único com tabelas crescentes
- Uma tabelas pode sobrepor-se a outra

Segmentacão

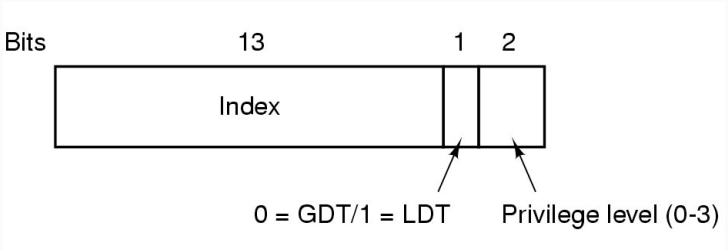


- Cada tabela pode crescer ou encolher independentemente

Segmentação vs Paginação

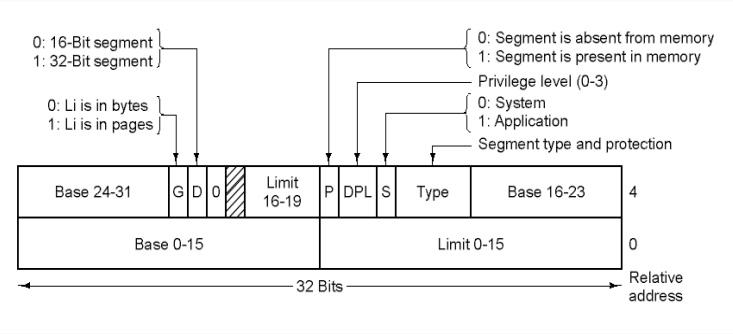
	Paginação	Segmentação
Transparente para o programador	Sim	Não
Número de espaços de endereçamento	1	Vários
O espaço de endereçamento pode ultrapassar o tamanho da memória física	Sim	Sim
O código e dados podem ser distintos e protegidos separadamente	Não	Sim
Tabelas de tamanho variável podem são geridas facilmente	Não	Sim
A partilha de código é facilitada	Não	Sim

Segmentação com Paginação: Pentium



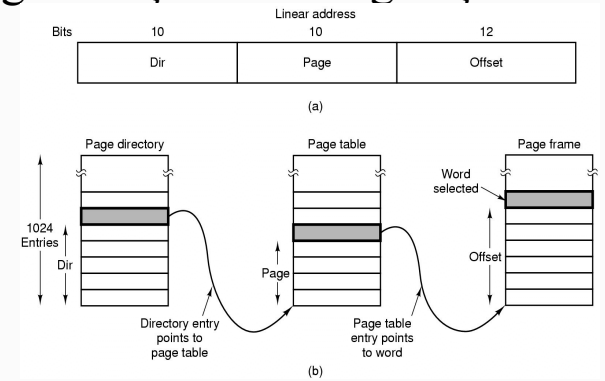
Um ***selector*** no Pentium

Segmentação com Paginação: Pentium



Descritor de segmento de código

Segmentação com Paginação: Pentium



Mapeamento de um endereço linear para o endereço físico

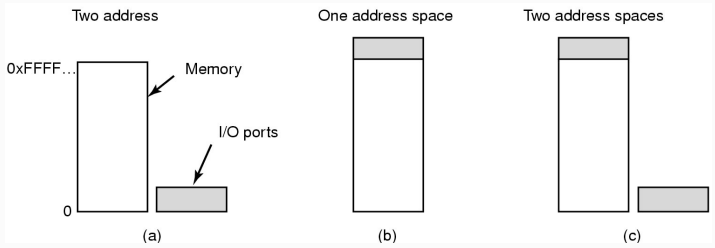
Programa

- Introdução
- Gestão de processos
- Noções de programação concorrente
- Gestão de memória
- Gestão de periféricos

Hardware de E/S

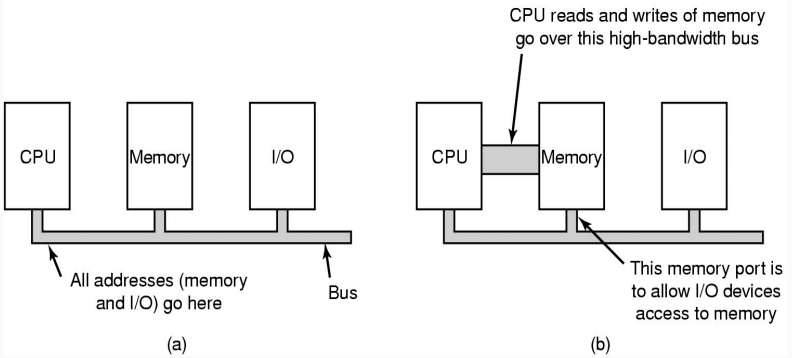
Device	Data rate
Keyboard	10 bytes/sec
Mouse	100 bytes/sec
56K modem	7 KB/sec
Telephone channel	8 KB/sec
Dual ISDN lines	16 KB/sec
Laser printer	100 KB/sec
Scanner	400 KB/sec
Classic Ethernet	1.25 MB/sec
USB (Universal Serial Bus)	1.5 MB/sec
Digital camcorder	4 MB/sec
IDE disk	5 MB/sec
40x CD-ROM	6 MB/sec
Fast Ethernet	12.5 MB/sec
ISA bus	16.7 MB/sec
EIDE (ATA-2) disk	16.7 MB/sec
FireWire (IEEE 1394)	50 MB/sec
XGA Monitor	60 MB/sec
SONET OC-12 network	78 MB/sec
SCSI Ultra 2 disk	80 MB/sec
Gigabit Ethernet	125 MB/sec
Ultrium tape	320 MB/sec
PCI bus	528 MB/sec
Sun Gigaplane XB backplane	20 GB/sec

E/S mapeado em memória



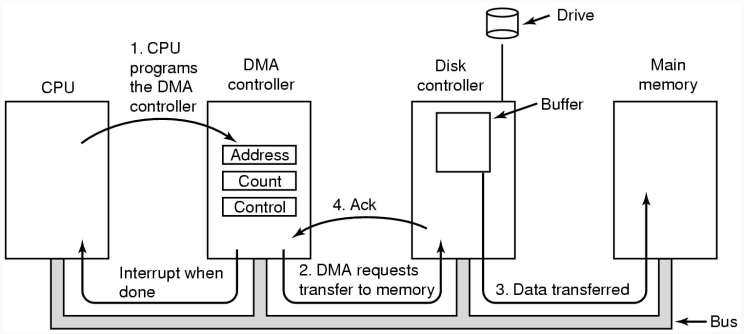
- Portas de E/S e memória separados
- E/S mapeado em memória
- Híbrido

E/S mapeado em memória

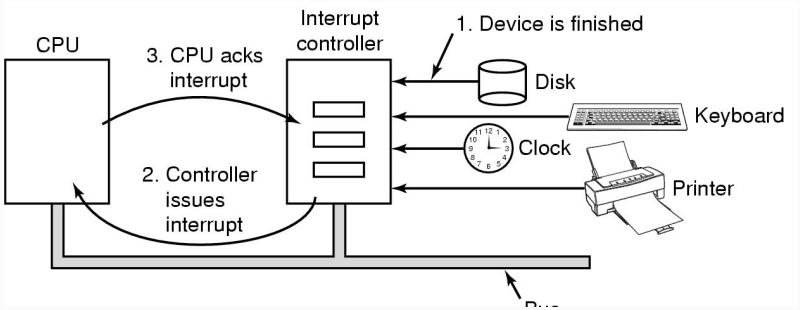


Arquitectura de barramento único e duplo

Memória de Acesso Directo (DMA)



Interrupções



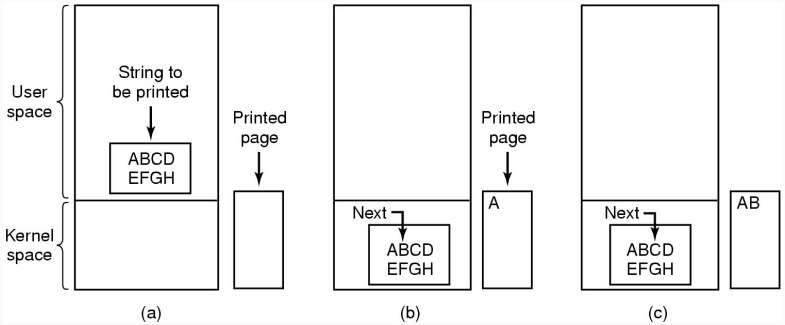
Características do Software de E/S

- Independência de dispositivos
  - programs podem aceder a qualquer dispositivo através de abstracções, sem que o tenham de especificar à priori.
- Uniformização de nomes
  - nomes de ficheiros e dispositivos independentes da máquina
- Tratamento de erros
  - Tão perto do hardware quanto possível

Objectivos do Software de E/S

- Transferências síncronas e assíncronas
  - chamadas bloqueantes vs. interrupções
- Armazenamento (buffering)
  - Armazenamento e cache a vários níveis
- Dispositivos partilhados vs. dedicados

E/S programado



Fases na impressão de uma string

E/S programado

```
copy_from_user(buffer, p, count);              /* p is the kernel bufer */
for (i = 0; i < count; i++) {                  /* loop on every character */
      while (*printer_status_reg != READY) ;   /* loop until ready */
      *printer_data_register = p[i];           /* output one character */
}
return_to_user( );
```

Escrita de uma string para a impressora usando E/S programad

E/S por interrupções

```
copy_from_user(buffer, p, count);
enable_interrupts( );
while (*printer_status_reg != READY) ;
*printer_data_register = p[0];
scheduler( );
```

(a)

```
if (count == 0) {
    unblock_user( );
} else {
    *printer_data_register = p[i];
    count = count – 1;
     i = i + 1;
}
acknowledge_interrupt( );
return_from_interrupt( );
```

(b)

Escrita de uma string para a impressora usando E/S por interrupções

E/S com DMA

```
copy_from_user(buffer, p, count);
set_up_DMA_controller( );
scheduler( );
```

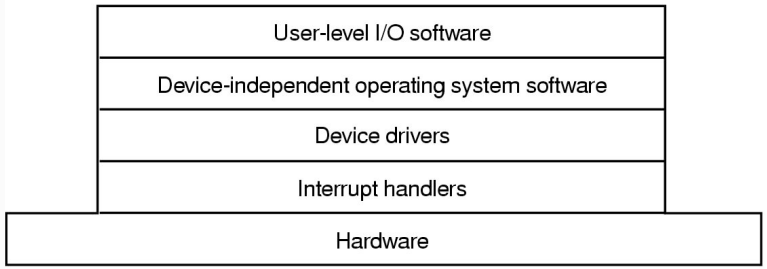
(a)

```
acknowledge_interrupt( );
unblock_user( );
return_from_interrupt( );
```

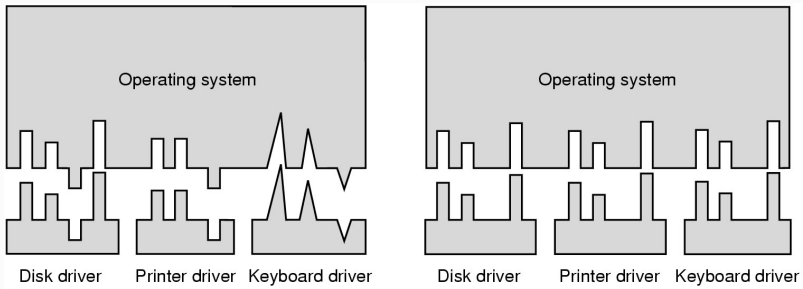
(b)

Operação de E/S com DMA

Níveis do software de E/S



E/S independente do dispositivo



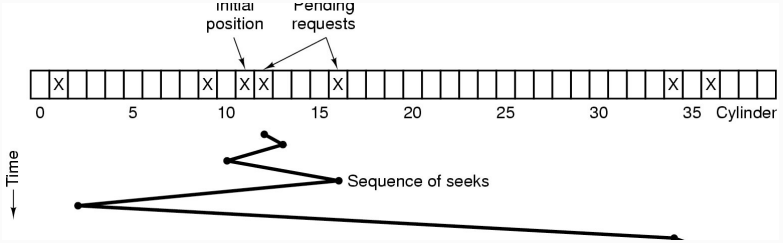
Sem interface standard

Com interface standard

Discos

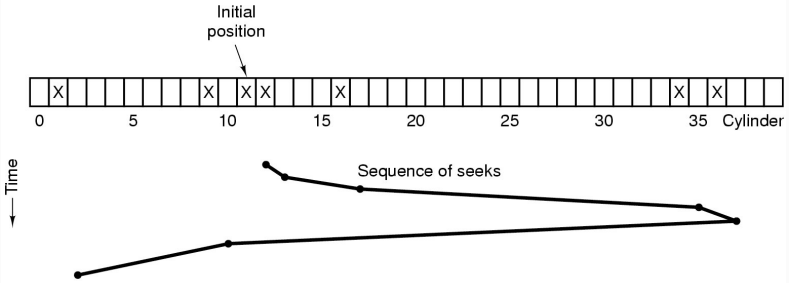
- O tempo necessário para ler um bloco é determinado por três factores:
  - Tempo de procura
  - Tempo de rotação do disco
  - Tempo de transferência
- O tempo de procura é dominante

Escalonamento de pedidos a disco



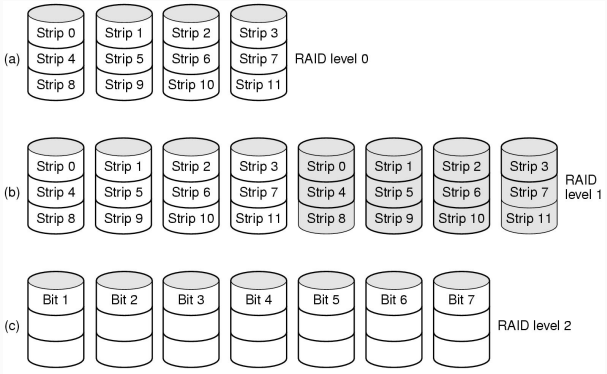
Shortest Seek First (SSF)

Escalonamento de pedidos a disco

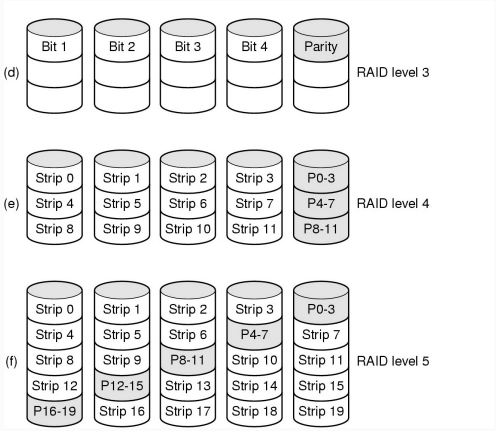


Elevador

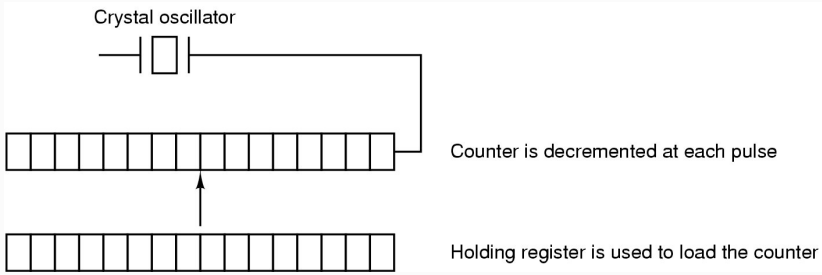
Sistemas RAID



Sistemas RAID

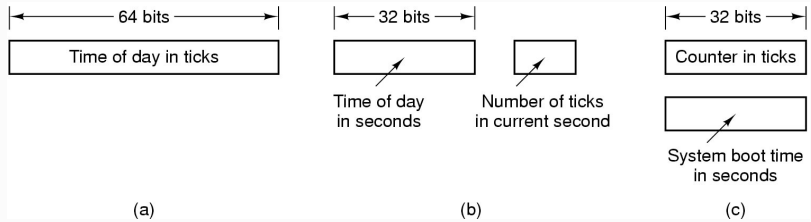


Relógios



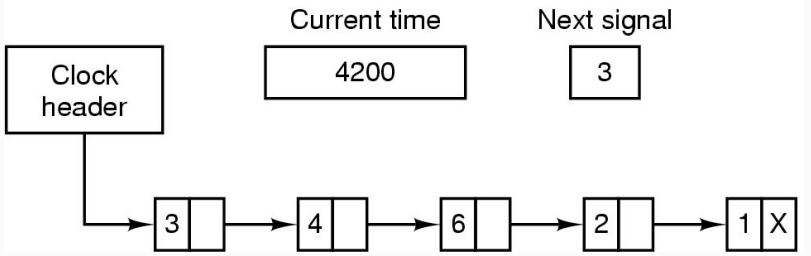
Um relógio programável

Relógios



Três formas de manter a hora actual

Relógios



Simulação de vários contadores com um único relógio