Conceitos de Sistemas Informáticos

Módulo de

Sistemas Operativos

Francisco Soares de Moura
fsm@di.uminho.pt

Grupo de Sistemas Distribuídos
http://gsd.di.uminho.pt

Programa para hoje

- Apresentação
  - whoamI + GSD
  - Actividade docente, de investigação e prestação de serviços
- Seguida de uma pequena conversa sobre SOs, como exercício de "aquecimento"

É uma cadeira de ***Engenharia***

- É preciso
  - Perceber como funcionam os sistemas informáticos
  - Perceber os compromissos
    - ⇒ usar a "massa cinzenta"
  - Sujar as mãos na "massa"
    => programar, configurar, avaliar
  - Tomar decisões... (aplicar a teoria, justificando)

Atenção!

- Esta cadeira (módulo) engana:
  - Não chega saber usar um SO
  - Nem estamos aqui (só) para aprender a usar um sistema operativo: Linux ou Windows XP, ...
- O objectivo é
  - Perceber "*how things work*"

Más notícias?

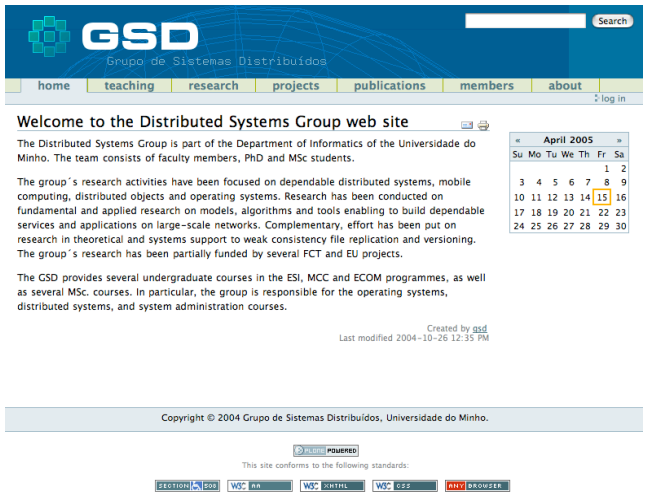
- Não é uma disciplina de "marrar"
  - Não comece a estudar perto do exame
  - Tente perceber o que está em jogo
  - Tente ser "engenheira/o"

    --> resolva o problema!

  - Prepare-se para justificar sempre o que faz ou propõe

Equipa Docente

- Aulas Teóricas
  - Francisco Soares de Moura (fsm@di.uminho.pt)
- Aulas Práticas
  - José Pedro Oliveira (jpo@di.uminho.pt)
- Horário de atendimento (fsm):
  - 2ª-feira, 12H00-13H00 ?
  - 3ª-feira, 9H45-10H45?



GSD

- Mestrado + ensino não graduado (excluindo "alfabetização"):
  - CSI → SO1 → SO II → SOD1 → SOD2

    → AS1 → AS2 / SED
- Investigação (ver página do grupo):
  - Sistemas Distribuídos de "Confiança" (robustos, tolerantes a faltas, seguros, ...)
  - Sistemas de Larga Escala, Mobilidade, Objectos em Sistemas Distribuídos, ...
- Prestação de Serviços
  - Administração de Sistemas e Redes (e.g. gestão de Labs, serviços de rede, redes wireless, segurança, etc.)

Programa

- Introdução, conceitos de programação de sistemas e noções de gestão de periféricos
- Gestão de processos e noções de programação concorrente
- Gestão de memória (+ recap compile/link/load)
- Gestão de ficheiros e noções de sistemas distribuídos

Bibliografia recomendada

- Sebenta de Sistemas Operativos ()
- 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

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

Transparências

- (Progressivamente) disponíveis em:

  http://gsd.di.uminho.pt/
- Baseadas nas transparências originais correspondentes aos livros recomendados
- Servem apenas como "âncora" ao estudo

Avaliação

- Veja a página de CSI
- SO vale 1/3 da cotação do Exame final
- As perguntas cobrem a matéria teórica e prática

Programa

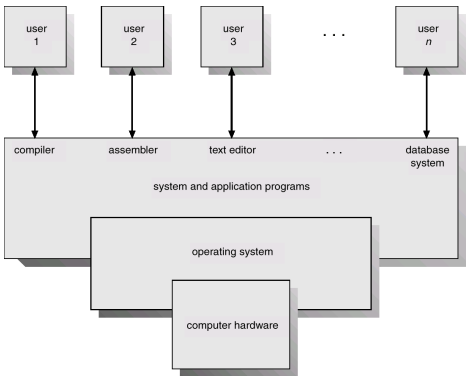
- Introdução, conceitos de programação de sistemas e noções de gestão de periféricos
- Gestão de processos e noções de programação concorrente
- Gestão de memória
- Gestão de periféricos

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

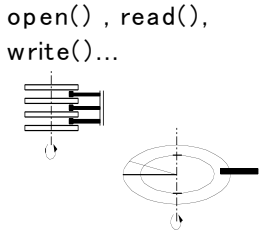


Portanto...

- 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 uma *máquina virtual*
- Gestor de recursos



Objectivos (1)

- Conveniência
  - SO esconde os detalhes do hardware

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

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

    e.g. ficheiros vs. blocos em disco

Na prática...

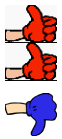
- É o Sistema Operativo quem define a "personalidade " de um computador
- Como se comporta o mesmo computador (hardware) após ter arrancado
  - MSDOS?
  - Windows 95?
  - Windows XP?
  - Linux, Knopix...?

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)

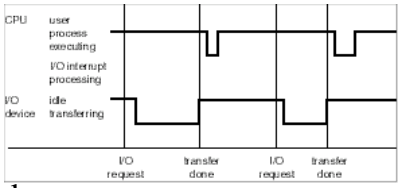
Embrião de um sistema operativo?

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

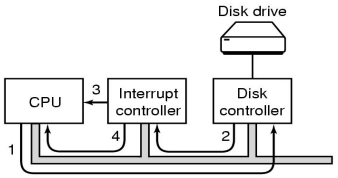
Exemplo: Polling IO

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

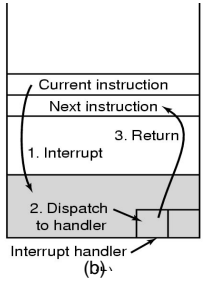
    (Equivalente a Já acabaste? Já acabaste? Já acabaste? Já acabaste? Já acabaste? Já acabaste? Já acabaste? Já acabaste? Já acabaste? Já acabaste? Já acabaste? Já acabaste? Já acabaste? Já acabaste?...)
  - OK, regressa de disk_io()

Resulta em *desperdício de tempo de CPU*

Exemplo: Interrupt-driven IO



(a)



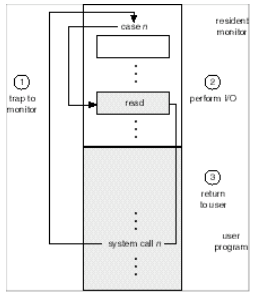
(b)

(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, trata-se rapidamente o evento, e continua a execução

Soluções (software)

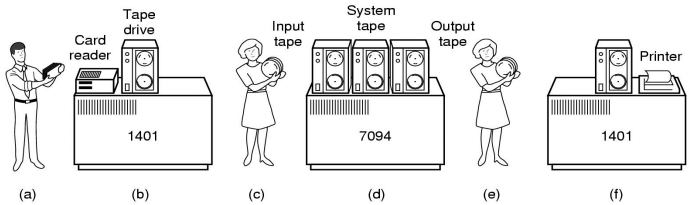
- Chamadas ao Sistema
- Virtualização de periféricos
- Multiprogramação



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?

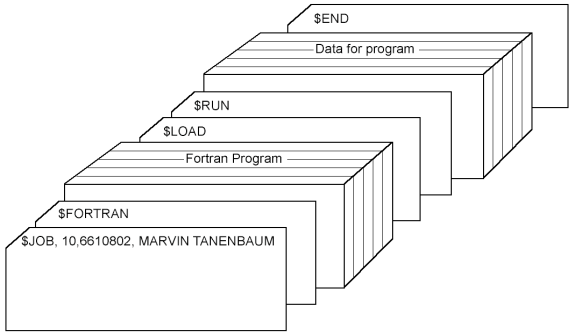
Primeiros sistemas de batch



Processador auxiliar faz IO de periféricos lentos (virtuais)

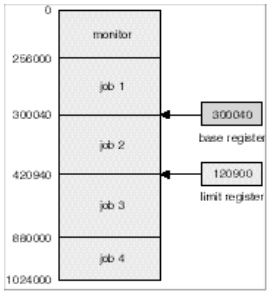
- Carregar cartões no 1401, que os copia para banda magnética
- Colocar banda no 7094 e executar os programas
- Recolher banda com resultados e colocá-la no 1401, que os envia para a impressora

Exemplo de um "job"

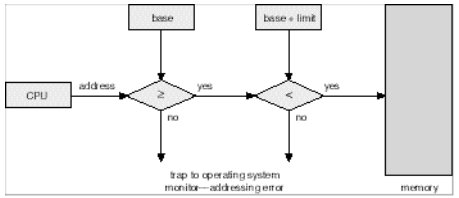


Multiprogramação

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

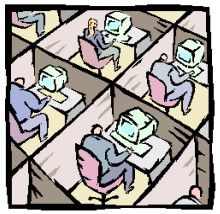


Protecção de memória



- 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

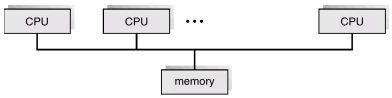
- Terminais (consolas) ligados ao computador central permitem que os utilizadores voltem a interagir directamente
- Sistema Operativo reparte o tempo de CPU pelos vários programas prontos a executar

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 tempo de resposta). Para isso necessitaria de paralelizar a aplicação, 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 lpr

- 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)!

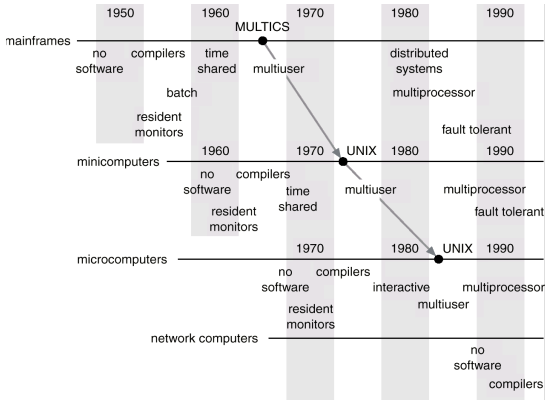
- Actualmente a virtualização é *HOT TOPIC* (vmware, …)

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

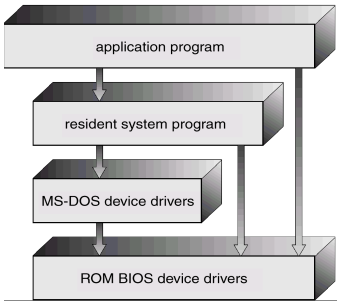


Arquitectura de Sistemas Operativos

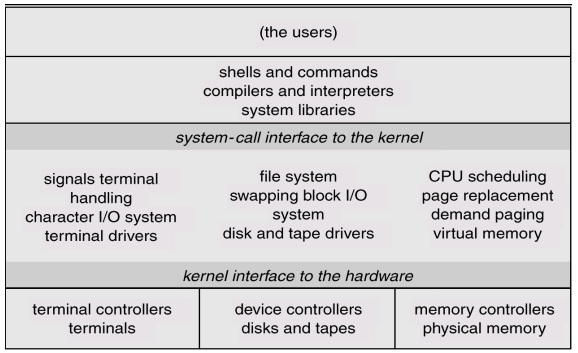
- Alguns exemplos
  - Sistemas monolíticos
  - Sistemas em camadas, hierárquicos
  - Modelo cliente-servidor
  - Máquinas virtuais

  …

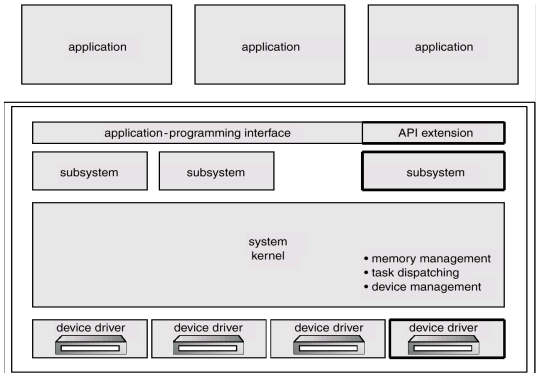
MS-DOS Layer Structure



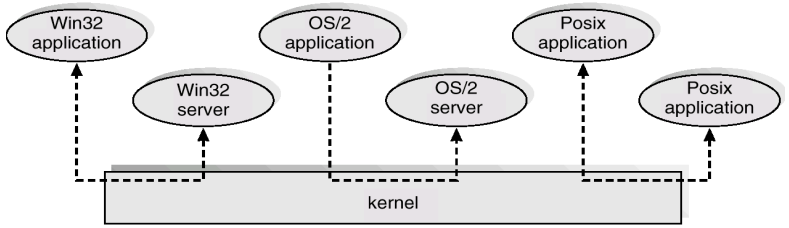
UNIX System Structure



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 executável:
  - Resultado da compilação, ligação, (re)colocação em memória
  - Normalmente dependerá de módulos externos, libs
- Processo em execução:
  - 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!)

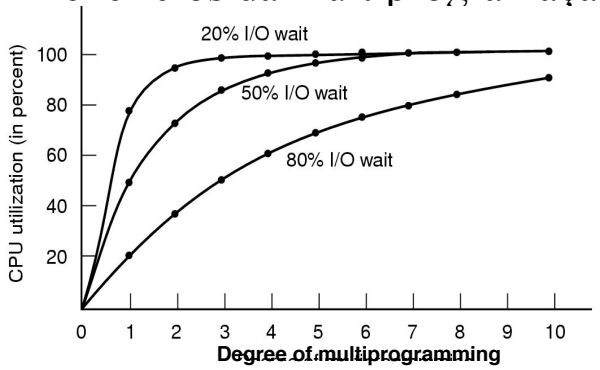
Programa

- Introdução
- Gestão de processos e 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



Paralelismo versus concorrência

- Execução paralela => hardware
  - Vários computadores, eg. cluster
  - Multiprocessamento, eg. um processo em cada CPU
  - CPU a executar instruções em paralelo com a operação de disco (que se manifesta através de uma interrupção, com prioridade superior à actividade no CPU)

Concorrência

- Criada pelo SO ao repartir tempo de CPU pelas várias actividades, em resultado de esperas passivas ou desafectação forçada
- Também conhecida por pseudo-paralelismo

Em geral...

- Seja num ambiente de paralelismo real ou simulado pelo SO
- Existem várias "actividades" em execução "paralela"
- Essas actividades não são independentes, há interacção entre elas
- Facto que levanta algumas questões...

Papel do SO

- O sistema operativo tem a responsabilidade de
  - Fornecer mecanismos que permitam a criação e interacção entre processos
  - Gerir a execução concorrente (ou em paralelo), de acordo com as políticas definidas pelo administrador de sistemas

Cooperação e Competição

- Em geral, um conjunto de actividades tem 2 tipos de interacção
  - Cooperam entre si para atingir um resultado comum
    - Processo inicia transferência do disco e aguarda que esta termine
    - O disco interrompe e a rotina de tratamento avisa o processo
  - Competem por recursos partilhados (CPU, espaço livre, etc.)
    - Há necessidade de forçar os processos a esperar até que o recurso fique disponível

Sincronização

- Em ambos os casos, estamos perante uma questão de sincronização:
  - Cooperação (espera até que evento seja assinalado)
  - Competição (espera até que recurso esteja disponível)
- Sincronizar é atrasar deliberadamente um processo até que determinado evento surja
  - Convém que a espera seja passiva.

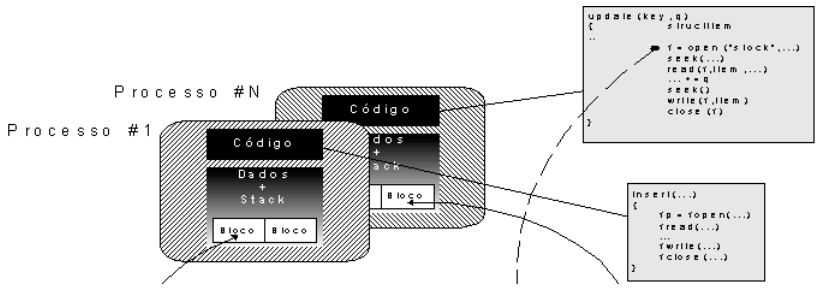
Comunicação

- Para poder haver interacção tem de haver alguma forma de comunicação
  - Processo podem requisitar ao SO um segmento partilhado, podem escrever/ler ficheiros comuns, enviar/receber "mensagens", etc.
  - Threads do mesmo processo podem comunicar através de variáveis globais
- A comunicação pode ser tão simples como o assinalar a ocorrência de um evento (de sincronização), ou pode transportar dados

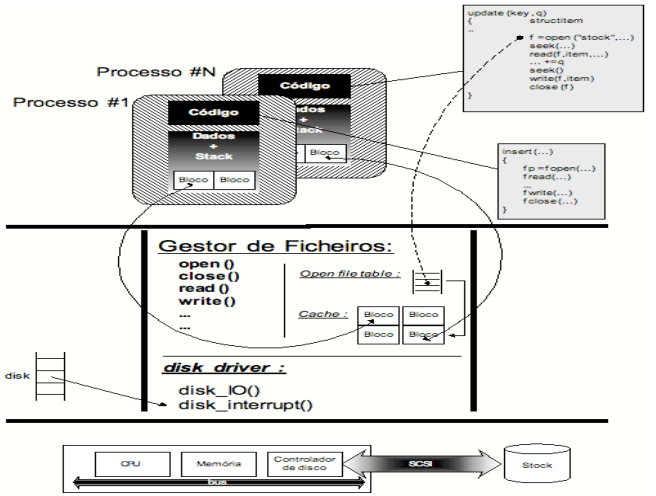
Exemplos

- Acesso a ficheiros partilhados
- Mirroring de discos
- Impressão

Acesso a ficheiros partilhados



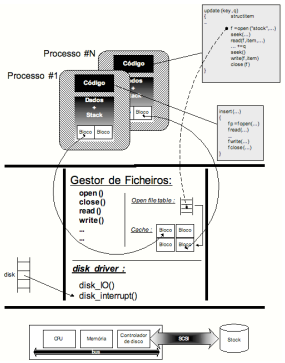
Dois programas a acederem simultaneamente ao mesmo ficheiro ou base de dados



"race conditions"

- Assegure-se que percebeu
  - Como surgem a competição
    - 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



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.

Mais exemplos de concorrência

- Inspirados na vida real
  - Diálogo cliente/bar(wo)men
  - Acesso ao WC
  - Acesso a um parque de estacionamento ou sala de cinema sem marcação de lugar
- São exemplos, respectivamente, de
  - Sincronização
  - Exclusão mútua (caso particular em que capacidade = 1)
  - Controlo de capacidade

Sincronização

BARMAN

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

CLIENTE

```
while (n_copos_balcao == 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).

=> Exclusão mútua

Regiões críticas

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

Regiões críticas

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

* Poderá ser interessante para os Sistemas Operativos, em circunstâncias bem definidas. Quais?

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:
  - Semáforos
  - Monitores
  - Sleep / Wakeup
  - Mensagens
  - Contagem de eventos

Semáforos

- Imagine uma caixa com bolas, rebuçados, pedras ...
- E as operações seguintes:

P:

Se há bola(s) na caixa, retiro uma e continuo, senão aguardo (passivamente) que alguém deposite uma.

V:

Devolvo a bola à caixa; se há alguém bloqueado à espera, acordo-a