
Sistemas de Operação

Apontamentos das Aulas Teóricas

Acetatos baseados nas aulas do Prof. Fernando Silva

Inês Dutra

ines@dcc.fc.up.pt (sala 1.31)

<http://www.dcc.fc.up.pt/ines/aulas/0809/SO/SO.html>

Departamento de Ciência de Computadores

Faculdade de Ciências da Universidade do Porto

2008/2009

Sistemas de Operação

- **Objetivos:** estudar os conceitos, estrutura, e mecanismos dos sistemas de operação.
- **Com que finalidade?**
 - entender como um SO funciona para melhor aproveitar suas funcionalidades durante a programação de aplicações;
 - ter uma visão crítica das limitações de um SO;
 - ter a competência para propor e implementar melhoramentos.

Sistemas de Operação – Programa

● Aulas teóricas:

- Conceitos básicos de SO: sistemas batch, buffering e spooling, multiprogramação, multi-tarefa (*time-sharing*), sistemas distribuídos e sistemas tempo-real
- Gestão de Processos: criação, escalonamento para execução, temporização, sincronização, zonas críticas, comunicação entre processos
- Gestão de Memória: mono- e multiprogramação, partições físicas, swapping, paginação, segmentação, memória virtual
- Sistema de Ficheiros: representação de ficheiros em disco, gestão do espaço em disco, mecanismos de proteção
- Periféricos de Entrada/Saída
- Impasses (deadlocks) e segurança

● Aulas práticas:

- Programação Avançada em Unix e em C: implementação de partes básicas de um SO

Bibliografia

- Silberschatz & Galvin & Gagne, [Operating System Concepts](#), 7th edition, John Wiley & Sons, 2004.
- W. Stallings, [Operating Systems: Internals and Design Principles](#), 5th edition, Prentice-Hall, 2004. (material complementar em <http://williamstallings.com/OS/OS5e.html>)
- R. Stevens & S. Rago, [Advanced Programming in the UNIX\(R\) Environment](#), 2nd edition, Addison-Wesley, 2005.
- Marc J. Rochkind, [Advanced UNIX Programming](#), 2nd edition, Addison-Wesley, 2004
- Mark Mitchell & Alex Samuel, [Advanced Linux Programming](#), New Riders, 2001.
- Warren W. Gay, [Advanced UNIX Programming](#), SAMS, 2000.

Método de Avaliação

- Aulas práticas e teóricas:
 - frequência obrigatória \Rightarrow há marcação de faltas, exceto para os oficialmente isentos.
- Trabalhos práticos: TP
 - a realizar em grupos de 2 alunos
 - 2 trabalhos obrigatórios (também para trabalhadores-estudantes)
 - valorização de 6 valores
 - nota mínima: 40% da valorização (2.4)
- Exame escrito: EX (com componente prática)
 - valorização de 14 valores
 - nota mínima em exame: 40% da valorização do exame (5.6)
 - nota mínima na parte prática do exame: 40% dessa componente (2.2)

Método de Avaliação

- Classificação final:

$$CF = (6 * NTP + 14 * EX) / 20$$

onde

$$NTP = \min(\max(EX - 20\%, TP), EX + 20\%)$$

Significado: a diferença entre a classificação obtida no trabalho prático e no exame não deverá exceder os 20%, caso contrário a classificação do trabalho prático será ajustada para esse limite.

- Nota importante: Cópias e plágios... :-)

O que é um Sistema de Operação?

- Um SO é um programa (ou conjunto de programas) que atua como intermediário entre o utilizador e o hardware do computador.
- Não há consenso sobre o que faz parte de um SO, mas é comum dizer-se que: *o SO é um programa, kernel, que está sempre em execução, considerando-se todo o resto aplicações.*
- O que fará então parte de um SO?

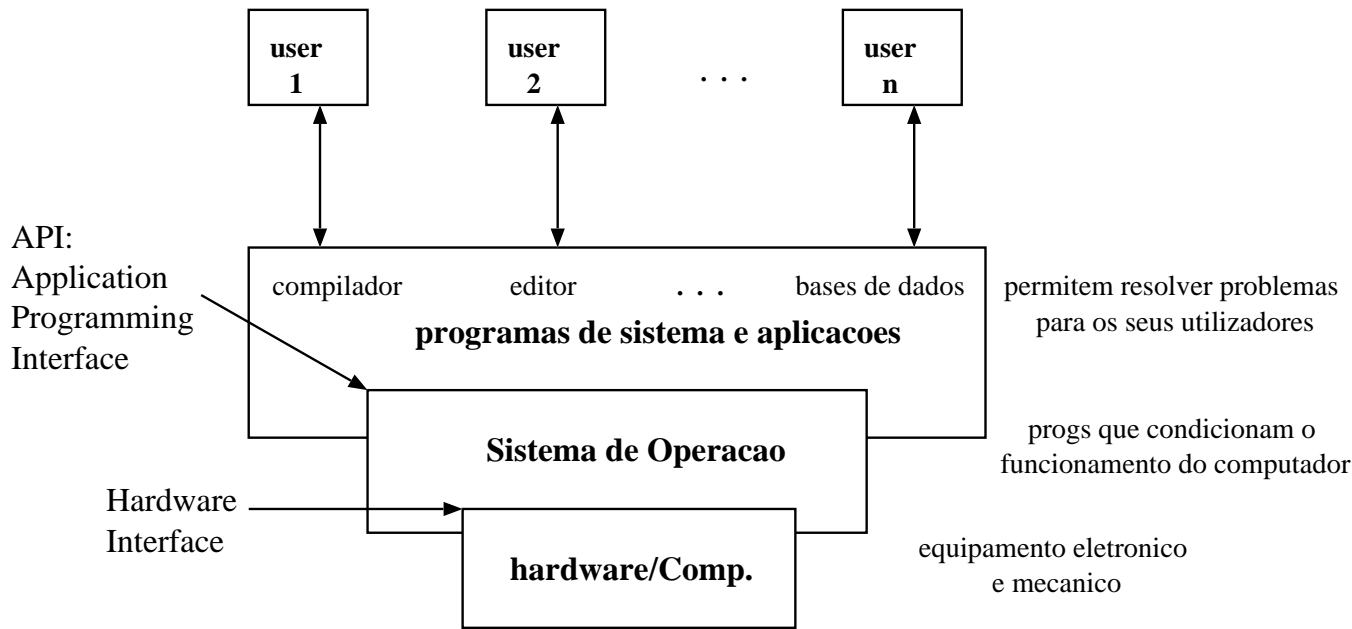
O mais certo:

- gestor de memória
- gestor de I/O
- scheduling do CPU
- gestor de tarefas (multi-tarefa/multi-programação)
- gestor de ficheiros
- gestor de comunicações (?)

E quanto a?:

- Suporte multimédia
- Interface com o utilizador (window manager)
- Internet browser

Componentes de um Sistema



problemas a resolver pelo SO:

- *como converter o que o hardware nos dá em algo que uma aplicação quer?*
- Qual é a interface ao hardware? (realidade física)
- Qual é a interface para a aplicação? (mais abstracta)
- Devemos passar mais responsabilidades para as aplicações ou para o sistema?

Funções de um SO

Um SO cumpre em geral duas funções:

- **Coordenador:**

- administra a partilha dos recursos de um computador, e.g. controla o acesso concorrente à memória, ao sistema de ficheiros e à rede.
- permite que vários utilizadores/aplicações usem o computador “em simultâneo”, de forma eficiente e justa.
- resolve pedidos conflituosos de programas e utilizadores.

- **Facilitador:**

- torna a tarefa de programação de aplicações, mais simples, rápida e menos sujeita a erros.
- esconde a complexidade do hardware dos utilizadores, fornecendo-lhes interfaces de acesso mais convenientes de usar.

Funções de um SO (cont.)

- E se não tivesse um SO?
 - processo habitual de compilação:
código fonte → compilador → código objeto → hardware
 - Como colocar o código objeto no hardware? Como escrever a resposta?
 - programar em binário. Pouco simpático e produtivo, mas foi assim com o primeiro computador!

Evolução dos SOs

- Um computador típico em 1981 vs. 2003 vs. 2006 vs. 2009:

	1981	2003	factor	2006	factor	2009	factor
CPU MHz	10	3000	300	4200	420	4200	420
Ciclos/inst.	3-4	0.5-1	3-8	0.5-1	3-8	0.5-1(*)	3-8
DRAM	128KB	512MB	4000	8192MB	64000	16384MB	128000
Disco	10MB	120GB	12000	750GB	75000	1500GB	150000
Largura banda	9600 b/s	100 Mb/s	10000	1000 Mb/s	100000	1000 Mb/s	100000
Endereçamento	16 bits	64 bits	4	64 bits	4	64 bits	4
Custo	\$25000	\$2000	0.08	\$1000	0.04	\$10.000	10 (**)

(*) para 1 processador

(**) vários processadores ou “cores”

Os primeiros sistemas (início anos 50)

- Estrutura:

- máquinas enormes acessíveis por uma consola
- um utilizador e um programa em execução de cada vez
- programador/utilizador era o operador da máquina
- fitas perfuradas e cartões

- Software:

- assembladores, compiladores
- carregadores (loaders), ligadores (linkers)
- bibliotecas de procedimentos mais comuns (libraries)
- controladores de periféricos (device-drivers)

- Seguros

- Uso ineficiente de recursos:

- baixo aproveitamento do CPU
- tempo de arranque muito significativo

Sistemas Batch Simples

- adotar um operador mais experiente
 - utilizador deixa de ter acesso ao hardware.
- agrupar programas dos utilizadores e execução gerida pelo operador.
 - melhora eficiência se os programas tiverem requisitos semelhantes
- e se um dos programas parar/falhar na sua execução?
 - operador tem de intervir.
- monitor residente: primeiro programa de um *SO rudimentar* para controlar a execução de programas:
 - (1) o monitor reside em memória e inicia execução
 - (2) transfere o controlo para outro programa
 - (3) quando o programa termina, transfere o controlo novamente para o monitor, que volta a (2).
- Dificuldades:
 1. Como é que o monitor sabe qual a natureza de uma tarefa ou qual o programa a executar?
 2. Como é que um monitor distingue: uma tarefa de outra? os dados do programa?
- Solução: através de cartões de controlo

Cartões de controlo

Cada tarefa inclui comandos para o monitor que determinam o compilador a usar, o programa e os dados para o programa:

```
$JOB
$FTN           % carrega o compilador
... --> código Fortran % é compilado e guardado
$LOAD         % carrega o prog. compilado
$RUN          % executa-o
... --> dados
$END
```

● Partes do monitor:

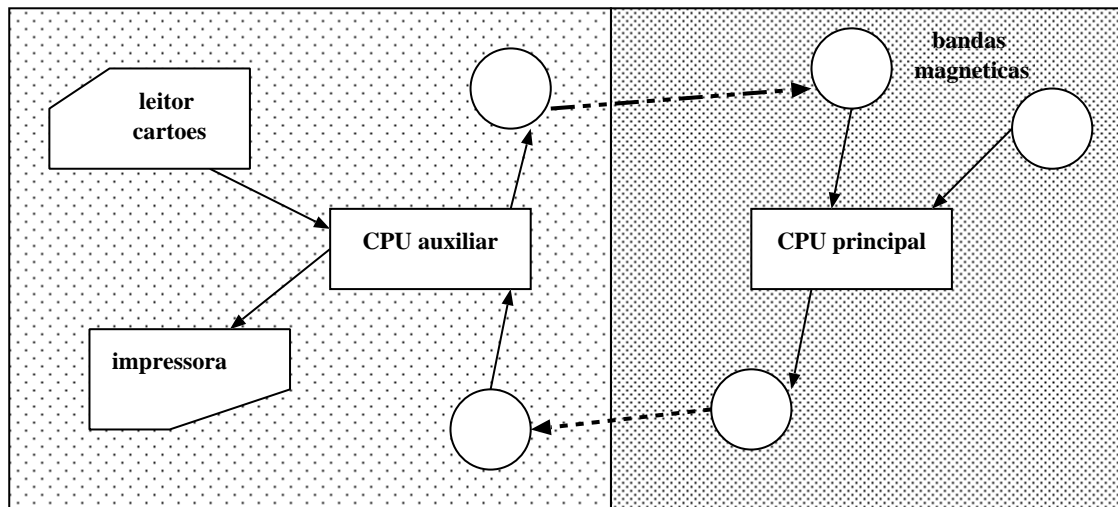
- interpretador dos cartões de controlo (comandos)
- carregador (loader) – carrega programas e aplicações para memória
- controladores de periféricos

● Problemas:

- Lento – não sobrepõe execução do CPU com I/O; leitor de cartões é lento.

● Solução: operação em off-line.

Operação off-line



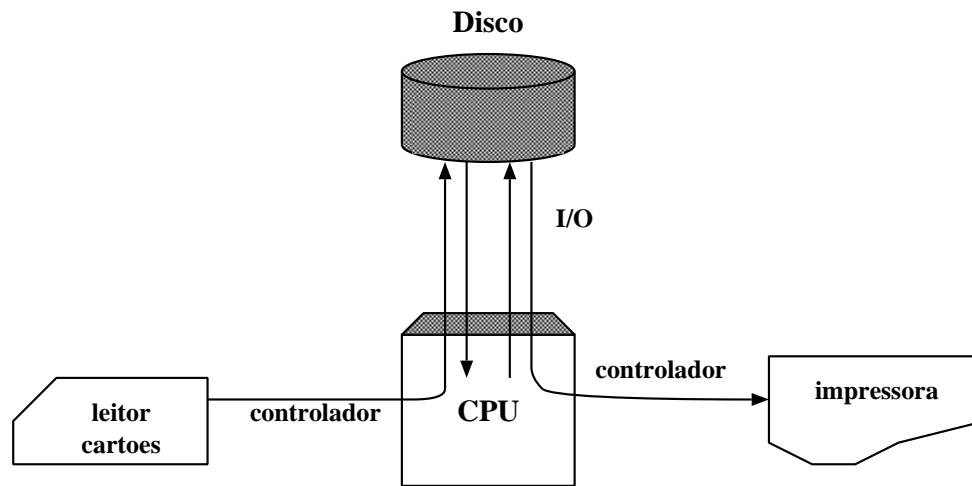
● Vantagens:

- sobreposição da execução do CPU e operações I/O, embora com 2 máquinas independentes.
- o computador não fica condicionado pela menor velocidade dos leitores de cartões e impressoras de linhas, depende da velocidade das unidades de bandas magnéticas.
- torna o sistema independente dos periféricos.

Operação off-line (cont.)

- Como sobrepor execução do CPU e operações I/O numa só máquina?
 - buffering – interpondo um buffer entre periférico e CPU.
 - Após o CPU ler do buffer os dados e começar a operar sobre estes, ambos CPU e controlador do periférico podem operar em simultâneo, estando ambos ocupados.
 - na prática, CPU é muito mais rápida e tem de esperar!

Spooling



- usa o disco como um buffer de grandes dimensões.
- sobrepõe operações de I/O de uma tarefa com a execução de outras tarefas.
Enquanto executa uma tarefa, o SO:
 - lê a próxima tarefa do leitor de cartões para o disco – fila de tarefas para execução.
 - O SO seleciona desta fila a próxima tarefa para execução – scheduling de tarefas.
 - escreve os resultados da tarefa anterior para a impressora.
- Um exemplo de spooling em SOs atuais é o envio de tarefas para a impressora.

Multiprogramação de Sistemas Batch

- **Sistemas uni-programados:**

- uma tarefa era executada de princípio a fim.
- ordem de execução das tarefas correspondia à ordem de leitura.
- o CPU não podia passar para uma nova tarefa se a actual estivesse parada à espera de I/O.

- **Sistemas multiprogramados:**

- os discos permitem acesso directo (não sequencial) e rápido, pelo que é possível pensar numa estratégia de escalonamento de tarefas (scheduling).

- a ideia é aumentar a utilização do CPU, para isso:

- ter em memória várias tarefas (programas) prontas a serem executadas.

Requer capacidade de gestão de memória pelo SO.

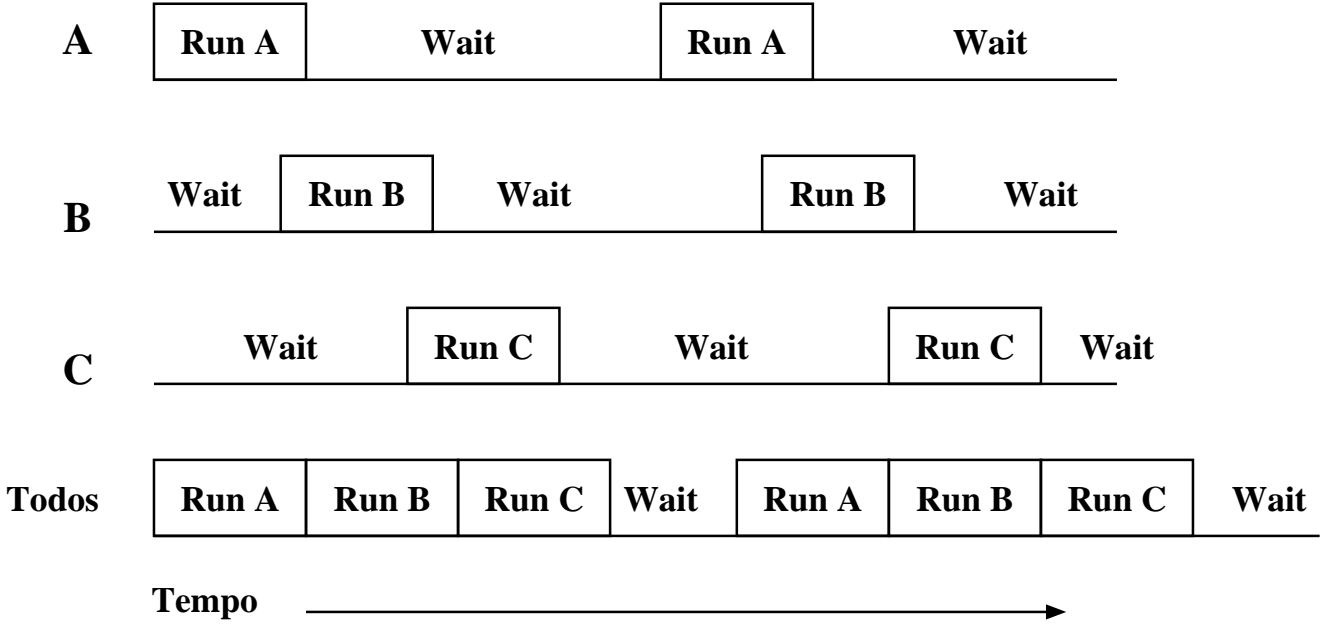
- cada tarefa é executada durante um certo tempo, caso seja interrompida o SO pega automaticamente numa nova. A tarefa interrompida será retomada logo que possa continuar a sua execução.

Que tarefa escolher? CPU scheduling.

- a execução concorrente de tarefas requer do SO a capacidade de limitar a possibilidade de interferência entre as tarefas.

Exemplo de multiprogramação

Programas



Multi-tarefa (Multitasking) ou Time-Sharing

- Sistemas Multi-tarefa:

- estende o conceito de multiprogramação.
- o CPU é partilhado por várias tarefas prontas a executar.
- cada tarefa executa durante uma fração de tempo, Δt .
- o SO comuta frequentemente entre várias tarefas em execução de forma que cria a ilusão de estar a executar todas em simultâneo.
- vai permitir a interação dos utilizadores com os respectivos programas em execução.

- Sistemas time-sharing:

- uso interativo de um computador
- muitos utilizadores a partilharem o computador.

- Multiprogramação em Batch versus Time Sharing

- Mutiprogramação em Batch: *objetivo principal é maximizar uso de CPU.*
- Time-sharing: *objetivo principal é maximizar tempo de resposta.*

Sistemas de Computador Pessoal

- computadores pessoais – sistema dedicado a um utilizador.
- periféricos I/O: teclado, rato, monitores, impressoras.
- até recentemente, não possuíam os requisitos necessários para suportar um sistema multi-tarefa.
- Sistemas operativos: MSDOS, MacOS, Windows, Windows-NT,...Windows2000, Linux.

Sistemas Paralelos (multiprocessadores)

- usar vários CPUs para executar mais tarefas por unidade de tempo.
- sistemas de memória partilhada – os CPUs partilham uma memória global e é através dela que comunicam.
- sistemas de memória distribuída – os CPUs têm a sua memória local, não há partilha. Os CPUs comunicam entre si por troca de mensagens.
- Vantagens:
 - aumenta o *throughput* – número de tarefas executadas por unidade de tempo.
 - melhora a fiabilidade.
 - multiprocessamento simétrico:
cada processador corre uma cópia idêntica do SO.

Sistemas tempo-real

- Outra forma de um SO, normalmente usado como controlador de periféricos dedicado a uma tarefa.
- Restrições de tempo, bem definidas. Ou se realiza dentro do tempo estabelecido ou o sistema falhará.
- Exemplo: controlo de um robot submarino.

Outras classificações de sistemas de operação

- monolítico (antigos sistemas: p.e. DOS, FreeBSD)
- micro-kernel: conjunto básico, pequeno de serviços (Amoeba)
- exo-kernel: exporta micro-kernels como máquinas virtuais para outras máquinas (Aegis)
- distribuído: faz migração automática de tarefas e co-escalonamento (p.e. Glunix, Mosix)