

Aspectos de um Sistema Operativo

João Paulo

Grupo de Sistemas Distribuídos
Departamento de Informática
Universidade do Minho



- Aspectos de um Sistema Operativo
 - Serviços de um sistema operativo
 - Interface com o utilizador
 - Programas de sistema
 - Chamadas ao sistema
 - Concepção e implementação de sistemas operativos
 - Estruturação de um sistema operativo
 - Máquinas virtuais



Serviços de um sistema operativo

Serviços úteis ao utilizador:

- **Interface com utilizador:** pode ser em linha de comando (CLI) ou interface gráfica (GUI)
- **Execução de programas:** carregar programas para memória, correr, terminar execução (eventualmente forçada)
- **Operações de I/O:** interacção de programas com ficheiros ou dispositivos
- **Manipulação do sistema de ficheiros:** manipulação de ficheiros e directórios, informação associada, gestão de permissões
- **Comunicação:** entre processos, na mesma máquina ou em rede; através de memória partilhada ou passagem de mensagens
- **Tratamento de erros:** no hardware ou programas; devem ser tratados adequadamente; facilidades de *debugging*



Serviços de um sistema operativo

Serviços de gestão do sistema:

- **Alocação de recursos**: aos vários utilizadores e processos; recursos como memória, tempo de CPU, ficheiros
- **Contabilização**: de como os utilizadores estão a usar recursos; e.g. espaço em disco, tempo de CPU
- **Protecção e segurança**: isolamento entre processos; controlo no acesso aos recursos; autenticação de utilizadores



Interface gráfica

- Normalmente metáfora do **desktop**:
 - rato, teclado, monitor
 - ícones que representam ficheiros, programas, acções
 - botões e menus para invocar acções; e.g. executar programa, abrir pasta
- Sistemas geralmente suportam também uma interface em linha de comandos:
 - Microsoft Windows: GUI + command shell
 - Mac OS X: Aqua GUI + unix kernel e shells
 - Linux: CLI + GUI opcional (p.ex, KDE)



Interface em linha de comando

- Permite a entrada de comandos
- Geralmente implementada por programas de sistema, que fazem uso de serviços do SO
- Com inúmeras variantes; e.g. sh, bash, csh, tcsh
- Recebe input e executa comando
- Comandos podem ser **internos** ou nomes de programas



Programas de sistema

- Software de sistema permite ter um ambiente apropriado para desenvolvimento e execução de aplicações:
 - manipulação de ficheiros
 - suporte a linguagens de programação
 - carregamento e execução de programas
 - obter informação sobre estado do sistema e histórico
 - acesso remoto e comunicação entre utilizadores
- Maioria dos utilizadores (excepto os programadores):
 - vê o sistema operativo como os programas de sistema
 - não se apercebe da existência das chamadas ao sistema



Linkers e Loaders

- Código fonte é compilado para **código objecto relocável**, que pode ser carregado em qualquer endereço de memória
- **Linker** combina ficheiros de código objecto em **executável**
 - possivelmente incluindo bibliotecas
- **Loader** carrega para memória programa em ficheiro executável
 - procede à **recolocação**: atribui endereços definitivos aos símbolos, e ajusta código e dados de acordo
- normalmente, as bibliotecas não são incluídas nos executáveis
 - são usadas **dynamically linked libraries**, que são carregadas quando necessário, e partilhada entre programas



Chamadas ao sistema

- **system call**: interface de programação para os serviços fornecidos pelo sistema operativos
- Suportadas por instruções do CPU; e.g trap, syscall
- Acedidas indirectamente via **API - application program interface**
- APIs mais comuns:
 - Win32 API para Windows
 - POSIX API para versões de UNIX, Linux, Mac OS X



Exemplo da API POSIX

Função para ler ficheiro

valor de retorno

nome da função

ssize_t

read

(int

fd,

void*

buf,

size_t

count);

parâmetros

São passados à função:

- descritor do ficheiro
- endereço do buffer onde vão ser depositados os bytes lidos
- número máximo de bytes que estamos dispostos a ler

A função devolve o número de bytes lidos ou erro

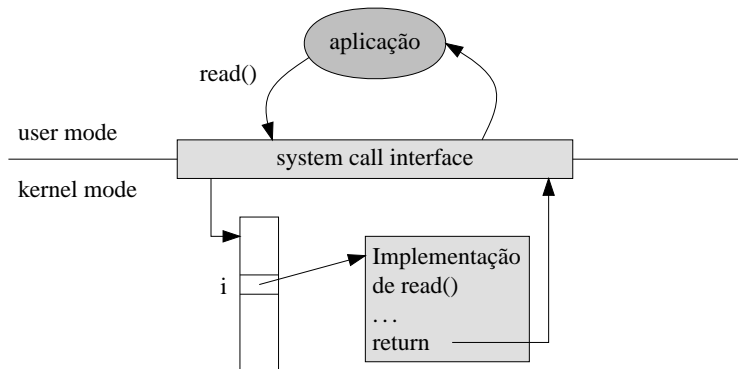


Implementação de chamadas ao sistema

- A cada system call é associado um **número**
- É mantida uma **tabela indexada** pelo número da system call
- A **system call interface** invoca a funcionalidade no kernel e devolve status e valor de retorno
- O invocador não sabe pormenores de implementação:
 - apenas sabe usar API
 - biblioteca que implementa API esconde detalhes



Relação API – system call – SO



Passagem de parâmetros às system calls

- Para além do índice da system call é necessário passar ao kernel os parâmetros da função
- Alternativas possíveis:
 - registos do CPU; mas pode haver mais parâmetros que registos
 - colocar parâmetros em tabela; passar endereço da tabela num registo; usado em Linux e Solaris
 - biblioteca faz push para o stack; kernel faz pop do stack



Exemplo: uso de chamadas ao sistema por programa

```
int main(int argc, char **argv)
{
    printf("Hello_World!\n");
    return 0;
}
```

Resultado de “strace a.out”

```
execve("./a.out", ["a.out"], [/* 28 vars */]) = 0
uname({sys="Linux", node="X1", ...})      = 0
brk(0)                                     = 0x804a000
...
write(1, "Hello_World!\n", 13)            = 13
munmap(0xb7fb3000, 4096)                  = 0
exit_group(0)                             = ?
```



Concepção e implementação de sistemas operativos

- Não existe única receita; várias alternativas com provas dadas
- Estrutura interna varia muito
- Definição de **objectivos** para:
 - **utilizador**: conveniência, facilidade de uso, confiabilidade, segurança, desempenho
 - **sistema**: facilidade de desenho e implementação, flexibilidade, confiabilidade, eficiência
- Princípio da **separação** entre política e mecanismo:
 - **política**: decisão sobre o que fazer; fim a atingir
 - **mecanismo**: como o fazer; implementar a política



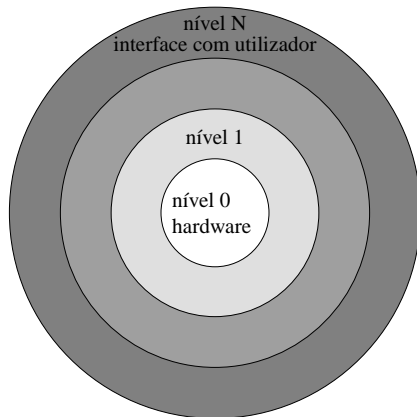
Estruturação simples

- Sem divisão em módulos
- Sem boa separação das interfaces e funcionalidades
- Máxima funcionalidade no mínimo espaço
- Exemplo: MS-DOS



Estruturação em camadas/níveis

- SO dividido por camadas/níveis, maior modularidade
- Cada camada construída usando camadas inferiores
- Dependência entre camadas e fluxo de pedidos multi-camada
- Nível 0 é o hardware; nível de topo a interface com utilizador



UNIX

Originalmente constituído por duas partes:

- Programas de sistema
- Kernel: escalonamento de processos, gestão de memória, sistema de ficheiros; muita funcionalidade num só nível

shells and commands compilers and interpreters system libraries		
system-call interface to the kernel		
signals terminal handling character I/O system terminal drivers	file system swapping block I/O system disk and tape drivers	CPU scheduling page replacement demand paging virtual memory
kernel interface to the hardware		
terminal controllers terminals	device controlers disks and tapes	memory controlers physical memory



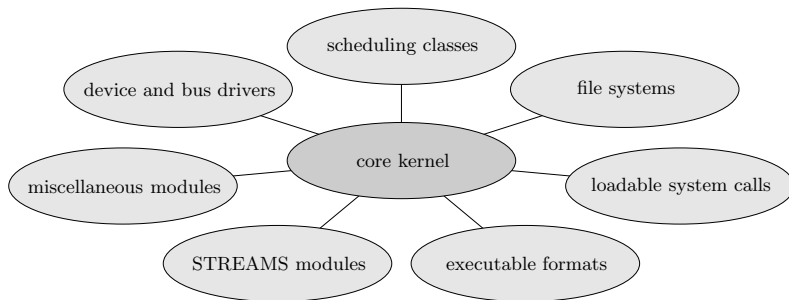
Microkernels

- Ideia: passar máximo de funcionalidade para modo utilizador
- Comunicação entre módulos por passagem de mensagens
- Benefícios:
 - fácil de estender
 - fácil portar SO para novas arquitecturas
 - maior confiabilidade (menos código no kernel)
 - mais seguro
- Inconvenientes:
 - Desempenho devido à comunicação entre kernel e modo utilizador



Estruturação por módulos

- SOs actuais usam módulos no kernel:
 - componentes separados
 - falam entre eles através de interfaces
 - módulos carregados apenas se necessários
- Mais versátil do que abordagem baseada em camadas
- Exemplo: Solaris

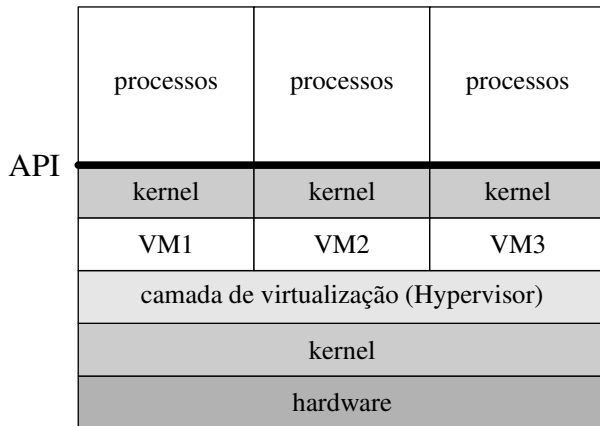


Máquinas virtuais

- Máquinas virtuais (*VM: virtual machines*) estendem a abordagem por camadas: encapsulam o hardware e sistema operativo como se fossem hardware
- Oferecem aos clientes uma interface idêntica à oferecida por determinada arquitectura de hardware
- Podem ter como clientes sistemas operativos a correr sobre o hardware virtualizado
- Os recursos físicos do computador são partilhados pelas diferentes instâncias das máquinas virtuais
- Oferecem ao cliente a ilusão de ter uma máquina só para si



Máquinas virtuais



Máquinas virtuais: isolamento

- Cada máquina virtual é isolada das outras e do hardware físico
- VMs são ideais para investigação em sistemas operativos
- Um SO experimental pode correr por cima de uma VM; quando as coisas correm mal, não é necessário reinicializar a máquina toda com o ambiente de desenvolvimento
- A implementação de VMs é difícil por ter que oferecer uma ilusão perfeita de hardware com um desempenho satisfatório



Exemplo: VMware

aplicação	aplicação	aplicação	aplicação
	Free BSD	Windows NT	Windows XP
	camada de virtualização		
Linux			
hardware			



Exemplo: Java Virtual Machine

- Outra concepção diferente de máquina virtual
- Aplicações fazem uso da **Java API**, ignorando hardware e SO
- Compilador de Java (ou outra linguagem) gera **java bytecode**
- Verificador de bytecode assegura certos invariantes:
 - permite protecção no acesso à memória sem ajuda do hardware
 - útil para hardware restricto (e.g. telemóveis)
- Runtime executa bytecode com interpretador ou compilador **JIT (just-in-time)**.

