Sistemas Operativos

Cap. II

Funcionamento

Interfaces e Arquitectura

Prof. José Rogado

jose.rogado@ulusofona.pt

Universidade Lusófona

Funcionamento, Interfaces e Arquitectura

- Arranque
- Interrupções e Excepções
- Modos de Funcionamento
- Serviços do Sistema Operativo
- Interfaces Utilizador
- System Calls
- Tipos de System Calls
- Programas Sistema
- Design e Implementação do Sistema Operativo
- Arquitectura de Sistemas Operativos
- Noção de Máquina Virtual

- Objectivos
  - Explicar o funcionamento básico do de um SO
  - Descrever os serviços e interfaces que o SO fornece aos utilizadores, aplicações e outros sistemas
  - Apresentar as várias arquitecturas de um sistema operativo

Arranque do Computador

Designado **por Bootstrap, é** composto pelos seguintes passos

- Uma sequência inicial de instruções simples é executada a partir de memória não volátil (flash, EEPROM, ..), que acede aos primeiros blocos do disco e traz para memória um programa com mais funcionalidades (*Boot Primário*)
- Este carrega seguidamente *Boot Secundário* que está armazenado numa zona específica do disco magnético (ex.: GRUB)
- Este programa, capaz de aceder ao SGF do disco e de dialogar com o utilizador, carrega o ficheiro com o código executável do sistema operativo em memória
- Começa então a inicialização do sistema operativo:
  - É criada a primeira *kernel task* que começa por detectar as características do hardware CPU, memória, periféricos e carregar os respectivos gestores.
  - É criado o primeiro processo utilizador *(init process)* que realiza a inicialização do serviços sistema de acordo com um conjunto de scripts e ficheiros de configuração
- O sistema Operativo uma vez inicializado, fica à espera de pedidos de serviços

Funcionamento Baseado em Eventos

- Quando nada acontece, o SO está num ciclo de espera
  - *System Idle Process*
- O funcionamento de um SO é "*event driven*", reagindo a:
  - **Interrupções**: geradas por hardware que necessita de intervenção
    - Timer. Operação de E/S, interacção com utilizador
  - **Excepções:** *Software Interrupts* ou *Traps* gerados por software
    - Erros de execução ou pedidos de serviços explícitos ao SO
- O Sistema Operativo utiliza mecanismos de protecção contra os possíveis erros das aplicações
  - Processos que entram em ciclo sem fim
  - Processos que tentam aceder uns os outros ou ao próprio SO provocando erros

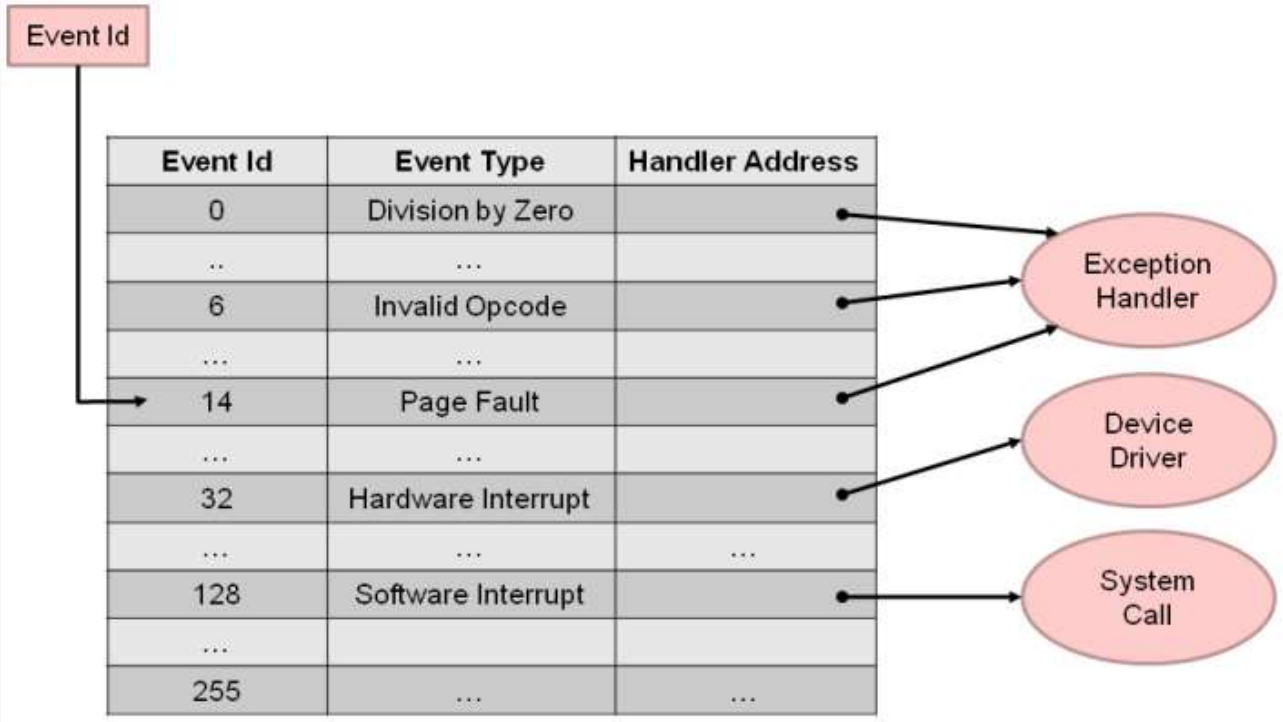
Conceito de Interrupção

- Uma interrupção é um evento despoletado por um sinal eléctrico que transfere o controle para uma rotina de serviço associada
- A correspondência entre o tipo de sinal e a rotina é feita através de um *vector de interrupções* que contém os endereços de todas as rotinas de serviço para um dados sistema.
- O mecanismo de interrupção deve salvaguardar o contexto de execução e estado do processador no ciclo em que esta ocorreu.
- Existe uma hierarquia de serviço associadas às interrupções.
  - Durante a execução do tratamento de interrupção, todos as interrupções de prioridade inferior são desactivadas
- Uma interrupção também pode ser gerada por software, quer por um erro ou pela execução de uma instrução do processador.
  - É nesse caso geralmente designada por *excepção*
- O Sistema Operativo é comandado por ***interrupções***.*e* ***excepções***

Gestão de Interrupções

- Quando ocorrem interrupções ou excepções, o SO realiza um conjunto de acções essenciais para a coerência do sistema:
- Guarda o estado do CPU salvaguardando os valores dos registos e do Program Counter em memória
  - *System Stack* (pilha sistema).
- Determina o tipo de interrupção que ocorreu através de um mecanismo hardware que dirige o controle para o endereço armazenado num endereço pré-determinado
  - *Vector de Interrupções (ou Interrupt Vector Table)*
- Vários serviços podem estar associados a um mesmo nível, o que obriga a que a origem seja determinada através da consulta dos periféricos associados a esse nível.
  - *Polling*
- A cada tipo de interrupção estão associados segmentos de códigos distintos, que realizam os serviços associados a cada tipo de interrupção
  - *Interrupt Handlers*

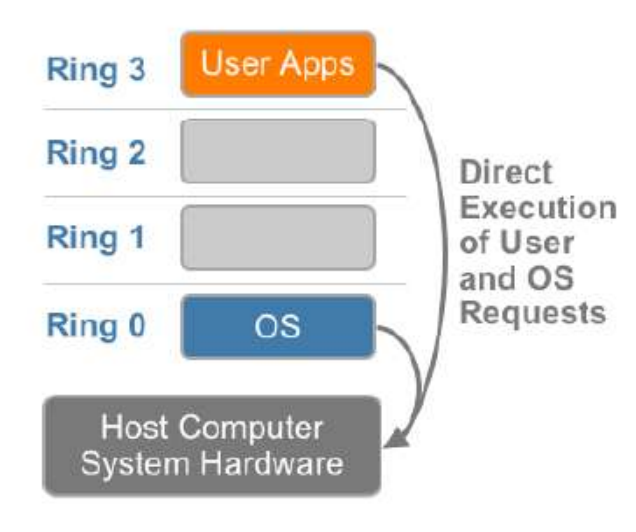
Vector de Interrupções



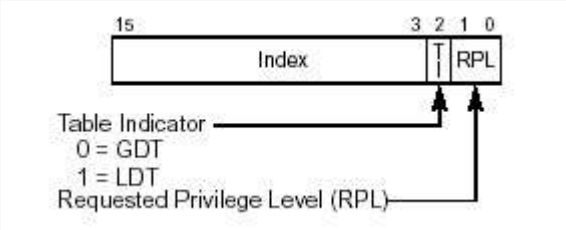
- O identificador de evento é utilizado para endereçar uma tabela, designada por Vector de Interrupções, que contém os endereços das rotinas de tratamento de cada tipo de evento (*Event Handler*)

Protecção do Nucelo

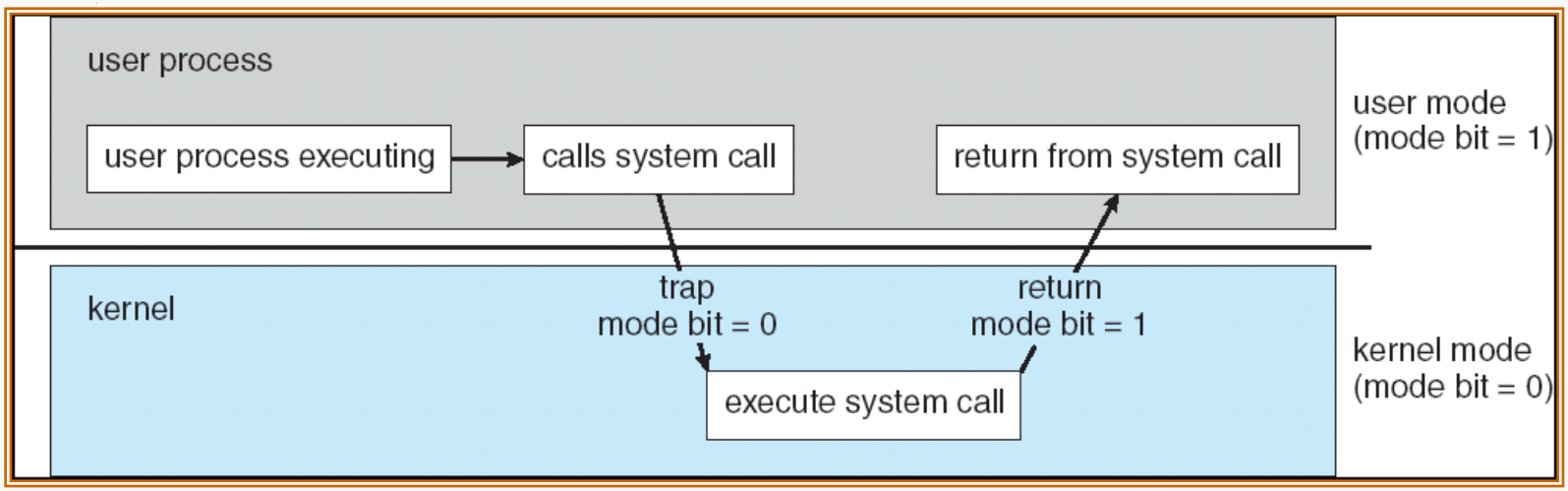
- O núcleo do SO tem de estar protegido das aplicações (falhas, vírus, etc…)
- Para isso, o SO utiliza pelo menos dois níveis de privilégio distintos
  - Funcionamento em **Dual Mode**
  - Modo **Utilizador** e modo **Supervisor**
- Privilégio implementado em Hardware:
  - Distingue a execução de código em modo utilizador e modo supervisor (**bit de modo**)
  - Fornece instruções **privilegiadas**, que só podem ser executadas em modo **Supervisor**
- Na arquitectura Intel o modo de execução é designado por **Current Privilege Level** (CPL)



Implementação do Dual-Mode na arquitectura Intel (CPL)



Transição de Modo



- A comutação automática para o modo **Supervisor**, é despoletada pelos seguintes eventos:
  - Interrupções hardware, excepções ou traps
  - Acessos a memória fora do espaço atribuído à aplicação
  - Acessos a periféricos
  - Execução de instruções privilegiadas em modo utilizador
- Os *system calls* são sempre realizados em modo Supervisor
- A intervalos de tempo regulares (timer), o processador transita para o modo supervisor para evitar ciclos infinitos das aplicações e utilização indevida de recursos

Protecção entre Aplicações

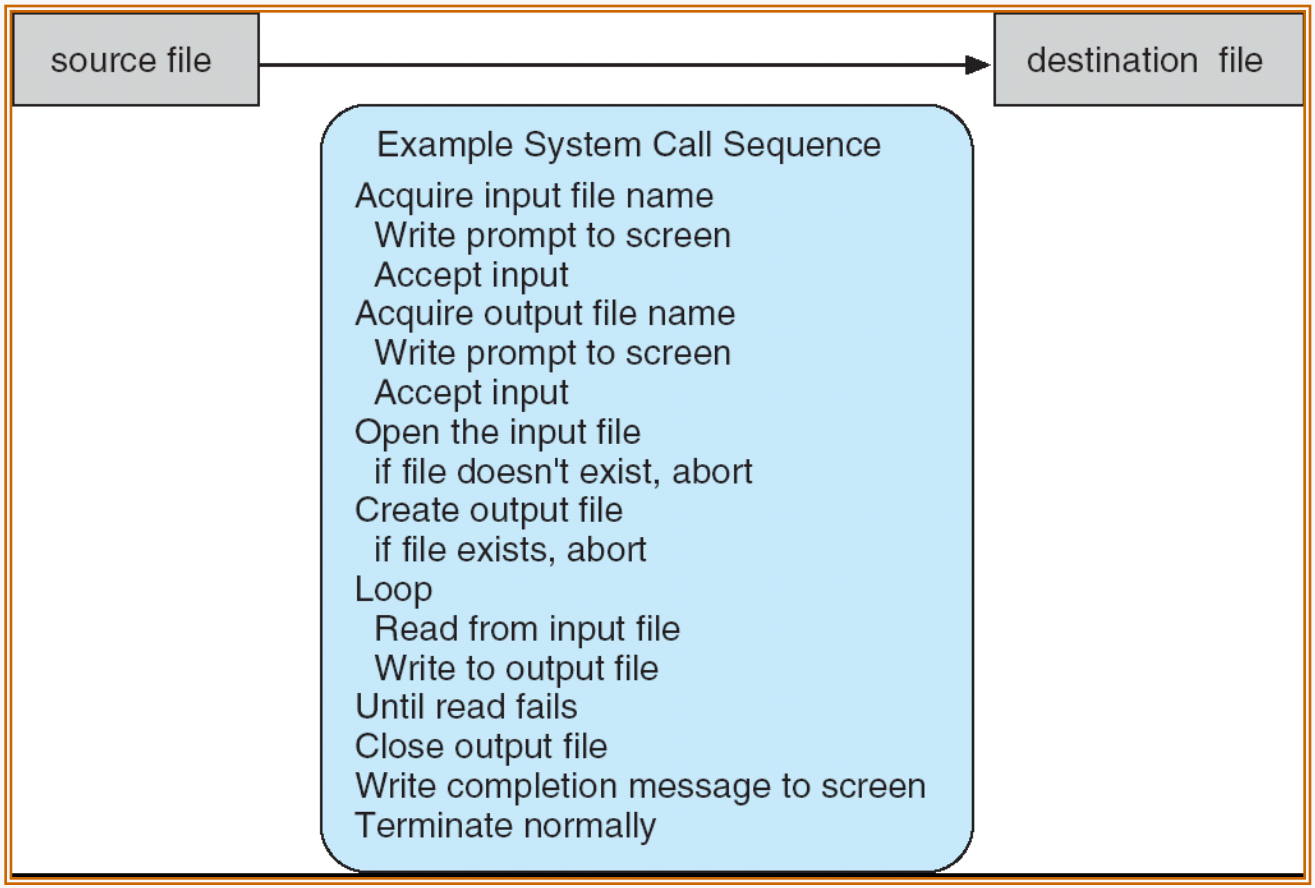
- A protecção entre aplicações é criada pela noção de espaço de endereçamento
- Cada aplicação é executada dentro de uma zona de memória exclusiva que contém:
  - O código da aplicação (pode ser partilhado em modo leitura)
  - Os dados da aplicação
  - A zona de variáveis dinâmicas (pilha)
- A noção de espaço de endereçamento é criada pela utilização de um dispositivo hardware
  - MMU - Memory Management Unit
  - Cada aplicação tem um perfil distinto da MMU
  - O kernel tem o seu próprio perfil que é automaticamente activado pelo bit de modo
- Cada **Processo** tem o seu espaço de endereçamento próprio
  - Veremos mais detalhes no Capítulo 7 sobre a Gestão da Memória

System Calls

- Interface de programação fornecida pelo SO
- Normalmente acessível em linguagens de alto nível (C, C++, Java, C#)
- Normalmente as aplicações utilizam uma **Application Program Interface (API)** que encapsula o acesso directo aos system calls
- As APIs mais utilizadas são a Win32 API para Windows, a POSIX API para praticamente todas as versões de UNIX, e a Java API para a Java Virtual Machine (JVM).
- Motivos para utilizar APIs em vez dos system calls directamente
  - Portabilidade – independência da plataforma
  - Esconder complexidade inerente aos system calls
  - Acréscimo de funcionalidades que optimizam o desempenho
- O acesso aos system calls está implementada em bibliotecas que são carregadas com as aplicações

Exemplo de Utilização

- Sequência de System Calls para copiar o conteúdo de um ficheiro para outro



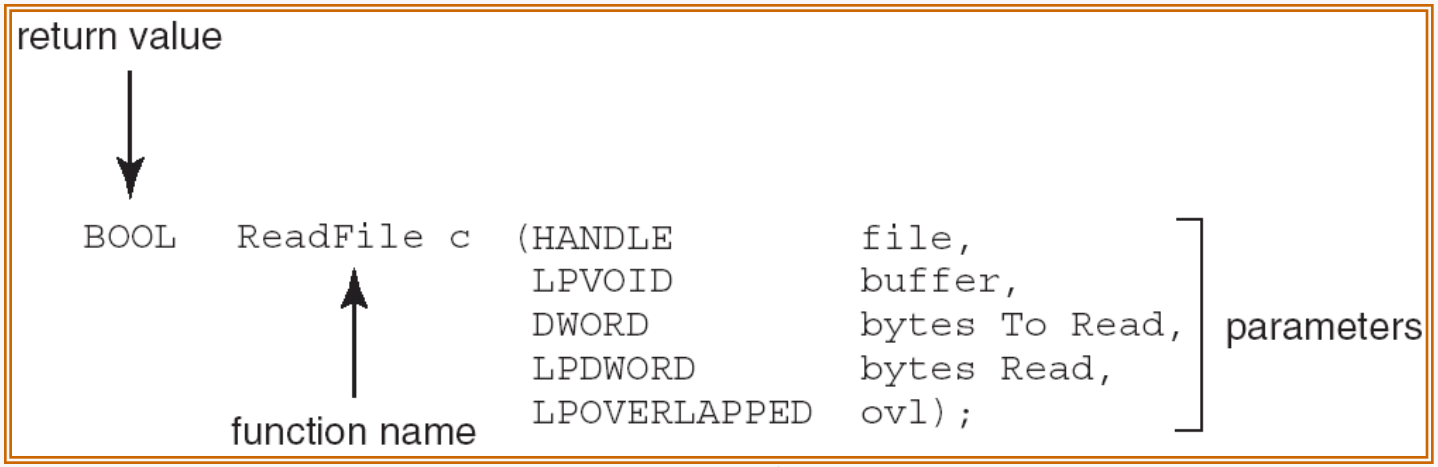
Exemplo de Programa

```
main(int argc, char *argv[])
{
    int source_fd, dest_fd;
    int count;
    char buffer[1024];

    if (argc != 3){
        fprintf(stderr, "Usage %s <source> <dest>\n", argv[0]);
        exit(1);
    }
    if ((source_fd = open(argv[1], O_RDONLY)) == -1) {
        fprintf(stderr, "Cannot open file %s\n", argv[1]);
        exit(1);
    }
    if ((dest_fd = open(argv[2], O_CREAT|O_RDWR, S_IREAD|S_IWRITE)) == -1) {
         fprintf(stderr, "Cannot create file %s\n", argv[2]);
         exit (1);
    }
    while ((count = read(source_fd, buffer, sizeof(buffer))) > 0) {
        fprintf(stderr, "Read %d bytes from fd %d\n", count, source_fd);
        if (write(dest_fd, buffer, count) != count)
            fprintf(stderr, "Could not write to file %s\n", argv[2]);
        else
            fprintf(stderr, "Wrote %d bytes to fd %d\n", count, dest_fd);
    }
    close(source_fd);
    close(dest_fd);
}
```

Utilização de System Calls para realizar a cópia de um ficheiro

Exemplo da API - Windows Standard

- A função ReadFile() da Win32 API
  - Uma função para ler o conteúdo de um ficheiro



- Descrição dos parâmetros de ReadFile()
  - HANDLE file - o descriptor do ficheiro a ler
  - LPVOID buffer - um buffer onde os dados irão ser colocados
  - DWORD bytesToRead - o número de bytes a ler
  - LPDWORD bytesRead - o número de bytes lidos na última leitura
  - LPOVERLAPPED ovl - indica se dever ser usada leitura entrlaçada

Exemplo da API UNIX Standard

NAME

read - read from a file descriptor

SYNOPSIS

```
#include <sys/types.h>
#include <unistd.h>

int read(int fd, char *buf, size_t count);
```

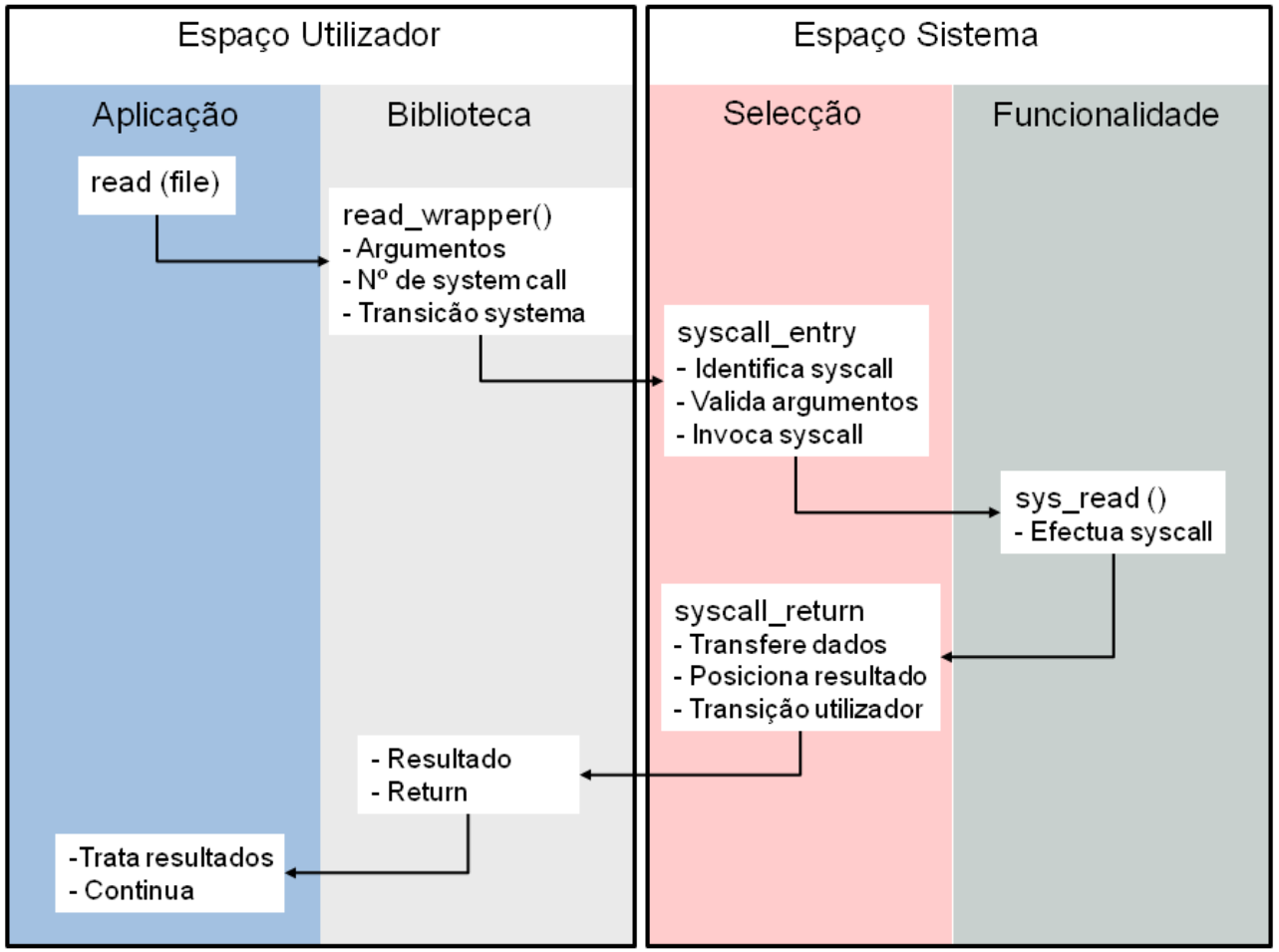
DESCRIPTION

read reads up to *count* bytes from file descriptor *fd* into the buffer starting at *buf*.

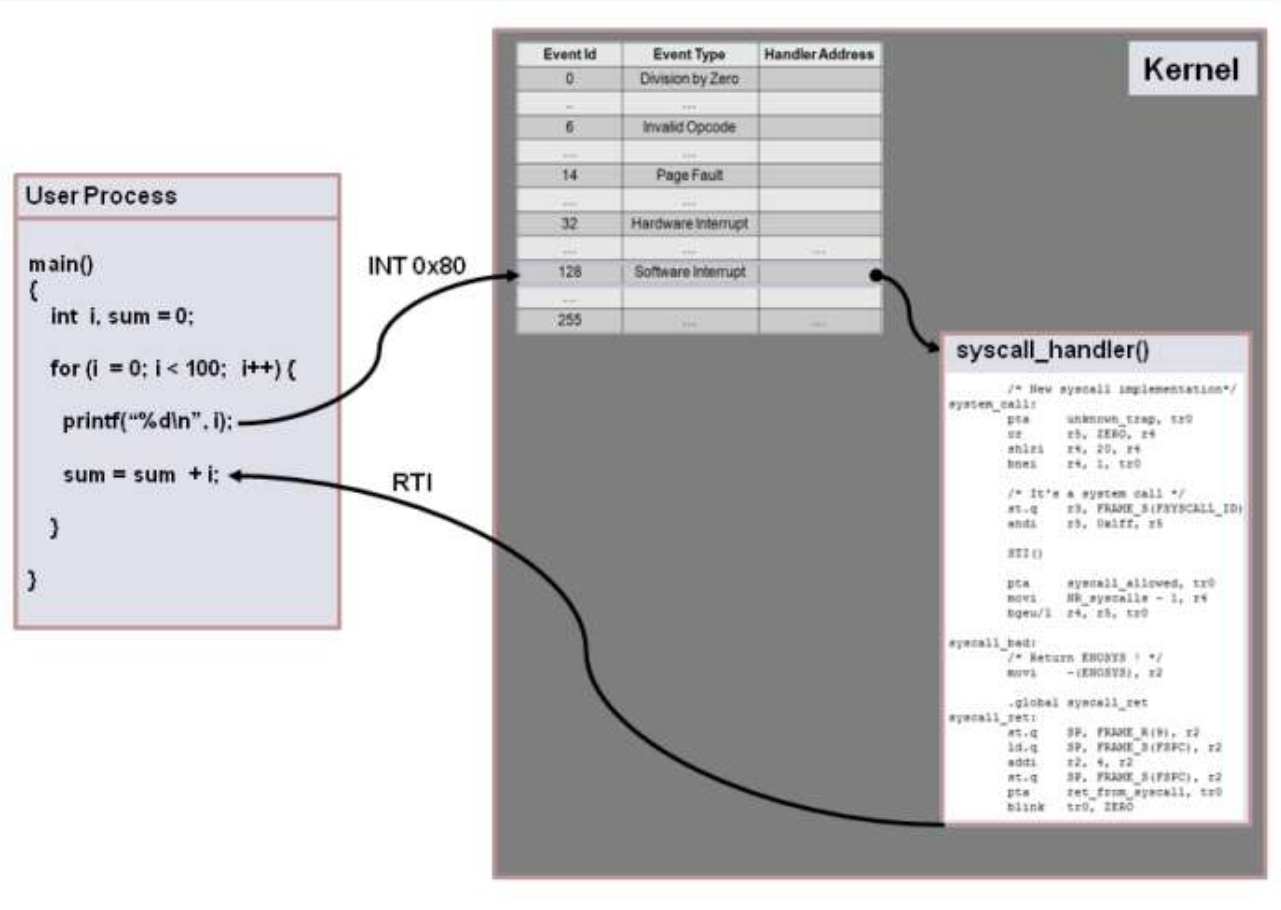
RETURN VALUE

On success, the number of bytes read are returned (zero indicates end of file). On error, -1 is returned, and *errno* is set appropriately.

Transição para os Syscalls em Linux



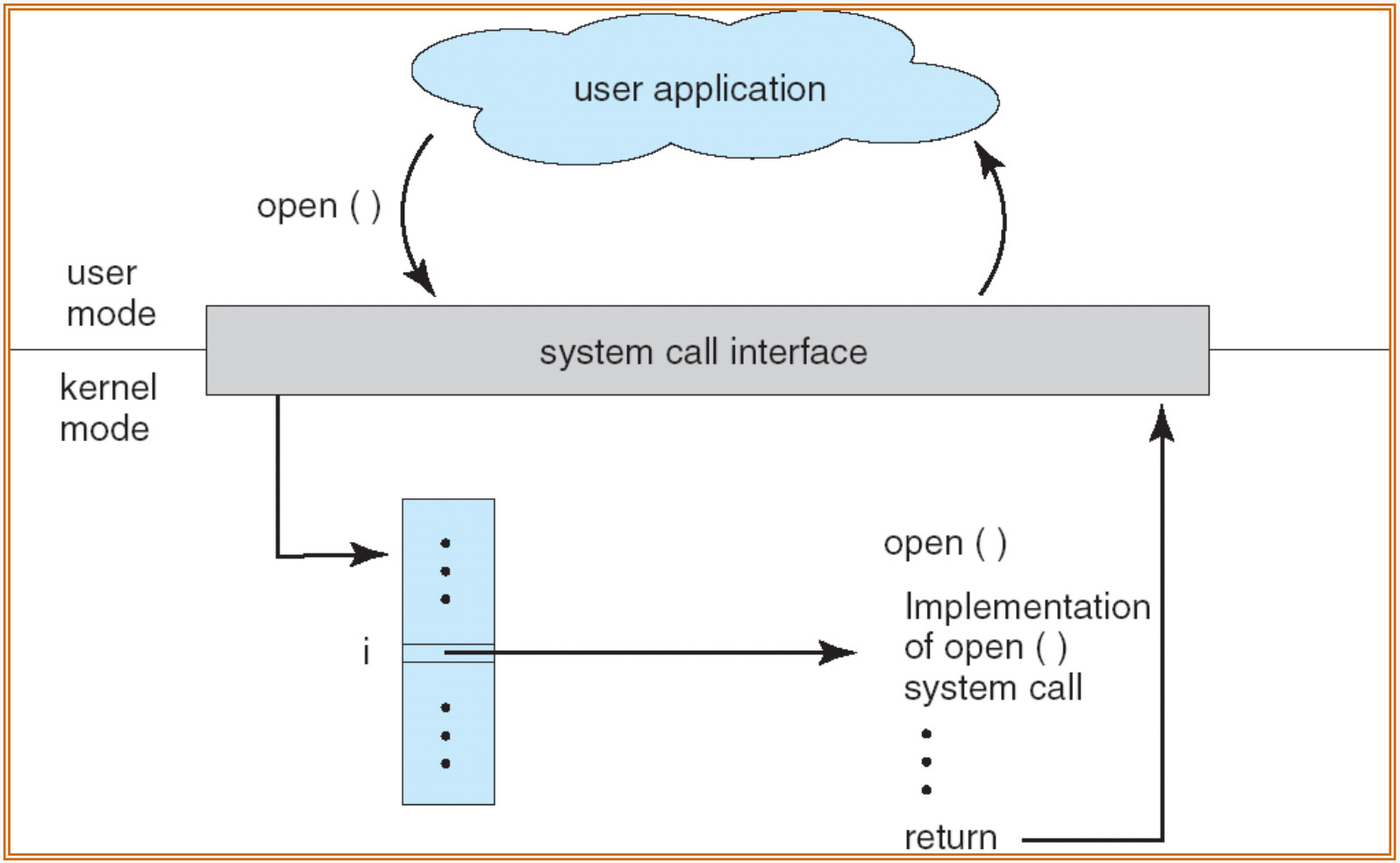
Detalhe da Transição em Linux i386



Implementação dos System Calls

- A cada system call é associado um número inteiro
  - A interface mantém uma tabela com o endereço de cada *system call handler* que é indexada pelo número do *system call*
- Através desta tabela, o respectivo *handler* é invocado no kernel
  - Os parâmetros do system call são transferidos para o kernel
  - Uma vez executado, o resultado e os parâmetros de retorno são transferidos para o programa utilizador, como se tivesse havido uma invocação de uma função normal
- A aplicação que invoca o *system call* não precisa de saber como este é implementado
  - Só precisa de obedecer à sintaxe da API (assinatura do método) e esperar pelos resultados da invocação
  - Precisa de conhecer a semântica associada ao *system call*
  - Os detalhes da interface sistema são escondidos pela API
    - São geridos por uma biblioteca fornecida pelo sistema – camada funcional que é incluída na aplicação no momento do carregamento do ficheiro executável

Mecanismo dos System Calls



Linux System Calls Numbers

```
/*
 * This file contains the system call numbers.
 */

#define __NR_restart_syscall      0
#define __NR_exit                 1
#define __NR_fork                 2
#define __NR_read                 3
#define __NR_write                4
#define __NR_open                 5
#define __NR_close                6
#define __NR_waitpid              7
#define __NR_creat                8
#define __NR_link                 9
#define __NR_unlink              10
#define __NR_execve              11
#define __NR_chdir               12
#define __NR_time                13
#define __NR_mknod               14
#define __NR_chmod               15
#define __NR_lchown              16
#define __NR_break               17
#define __NR_oldstat             18
#define __NR_lseek               19
#define __NR_getpid              20
#define __NR_mount               21
#define __NR_umount              22
#define __NR_setuid              23
#define __NR_getuid              24
#define __NR_stime               25
#define __NR_ptrace              26
#define __NR_alarm               27
#define __NR_oldfstat            28
#define __NR_pause               29
#define __NR_utime               30
#define __NR_stty                31
#define __NR_gtty                32
#define __NR_access              33
#define __NR_nice                34
#define __NR_ftime               35
#define __NR_sync                36
#define __NR_kill                37
#define __NR_rename              38
#define __NR_mkdir               39
#define __NR_rmdir               40
#define __NR_dup                 41
#define __NR_pipe                42
```

Arquitectura x86 (32 bits)

https://w3challs.com/syscalls/?arch=x86

Arquitectura ia64 (64 bits)

https://w3challs.com/syscalls/?arch=x86_64

Linux Syscall Table

```
.data
ENTRY(sys_call_table)
        .long sys_restart_syscall       /* 0 - old "setup()" system call, used for restarting */
        .long sys_exit
        .long sys_fork
        .long sys_read
        .long sys_write
        .long sys_open          /* 5 */
        .long sys_close
        .long sys_waitpid
        .long sys_creat
        .long sys_link
        .long sys_unlink        /* 10 */
        .long sys_execve
        .long sys_chdir
        .long sys_time
        .long sys_mknod
        .long sys_chmod         /* 15 */
        .long sys_lchown16
        .long sys_ni_syscall    /* old break syscall holder */
        .long sys_stat
        .long sys_lseek
        .long sys_getpid        /* 20 */
        .long sys_mount
        .long sys_oldumount
        .long sys_setuid16
        .long sys_getuid16
        .long sys_stime         /* 25 */
        .long sys_ptrace
        .long sys_alarm
        .long sys_fstat
        .long sys_pause
        .long sys_utime         /* 30 */
        .long sys_ni_syscall    /* old stty syscall holder */
        .long sys_ni_syscall    /* old gtty syscall holder */
        .long sys_access
        .long sys_nice
        .long sys_ni_syscall    /* 35 - old ftime syscall holder */
        .long sys_sync
        .long sys_kill
        .long sys_rename
        .long sys_mkdir
        .long sys_rmdir         /* 40 */
        .long sys_dup
        .long sys_pipe
```

As rotinas são invocada pela instrução:

```
call *ia32_sys_call_table(,%rax,8)
```

No ficheiro:

```
arch/x86/include/asm/unistd_32.h
```

Windows Syscall Table

- Uma tabela contendo os nomes dos System Calls e os respectivos valores para cada uma das versões do sistema:
  - http://netlab.ulusofona.pt/so/praticas/winsyscalls.h
  - Cada elemento da tabela é do tipo:

```
struct SCENTRY {
    char* name;
    int   x[17];
}
```

- Utilização
  - Nome: syscalls[syscall_number].syscall_name
  - Valor: syscalls[syscall_number].x[system_version]
- O mecanismo de entrada no *kernel* é semelhante ao do Linux
  - Usa a instrução int $0x2e para comutar para o modo de privilégio máximo
  - Mais detalhes em:

http://www.codeguru.com/Cpp/W-P/system/devicedriverdevelopment/article.php/c8035

Passagem de Parâmetros em Linux

- Na arquitectura Intel x86, os parâmetros são passados nos registos
  - Registo **eax** contém o número do syscall
  - Registos **ebx**, **ecx**, **edx**, **esi** e **edi** contêm os primeiros 5 argumentos
  - No caso de haver mais, são passados por referência, sendo o endereço da sua localização passado por registo.
  - O valor de retorno do *syscall* é enviado através do registo **eax**.
- Todos os argumentos passados para o kernel têm de ser validados por este
  - Evitar utilização de recursos não autorizados ou não existentes
  - Evitar acessos a zonas de memória não pertencentes ao espaço de endereçamento do processo
  - Evitar acessos indevidos a seu próprio espaço
    - P.ex.: escrever em zonas *read only*
- Os argumentos são copiados por funções em C e Assembly
  - http://lxr.free-electrons.com/source/arch/x86/lib/usercopy_32.c?v=3.2

Invocação Directa de Syscalls (x86)

- Programa em Assembler que invoca os system calls **write()** e **exit()** através da instrução **`int 0x80`**

```
1           .data
2   msg:
3           .string  "Hello World\n"
4   end:
5           .long 0                         # null terminated string
6
7           .text
8   .globl _start
9   _start:
10          movl     $1, %ebx               # stdout
11          movl     $msg, %ecx             # message
12          movl     $(end-msg), %edx       # length
13          movl     $4, %eax               # NR_write = 4
14          int      $0x80                  # call kernel
15
16          movl     $0, %ebx               # exit value
17          movl     $1, %eax               # NR_exit = 1
18          int      $0x80                  # call kernel
```

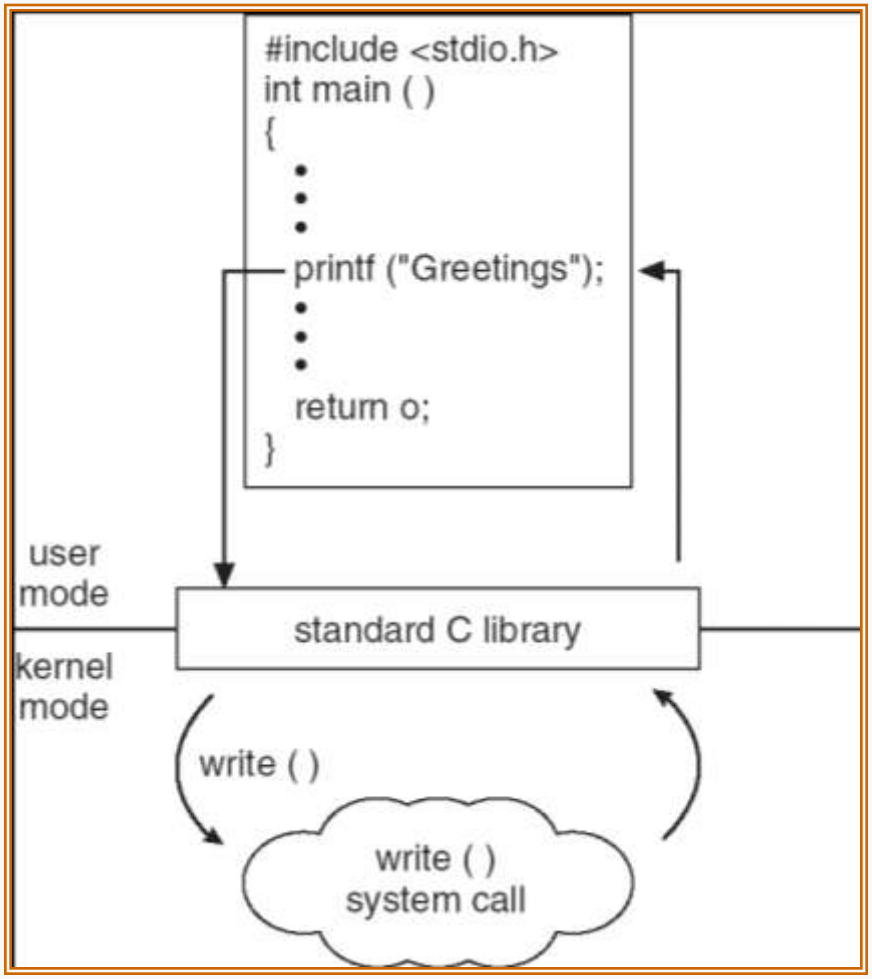
Invocação Directa de Syscalls (ia64)

- A invocação dos system calls na arquitectura 64 bits é feita através da instrução **`syscall`**
- Utiliza os registos rdi, rsi, rdx, r10, r8 e r9 para os argumentos e o rax para o número do system call.

```
1           .data
2   msg:
3           .asciz  "Hello, World\n"
4   end:
5           .long 0
6
7           .text
8   .global _start
9
10  _start:
11          # write(1, msg, sizeof(msg))
12          mov     $1, %rdi                # file handle 1 is stdout
13          mov     $msg, %rsi              # address of string to output
14          mov     $(end-msg), %rdx        # number of bytes
15          mov     $1, %rax                # system call 1 is write
16          syscall                         # call kernel to write
17
18          # exit(0)
19          xor     %rdi, %rdi              # we want return code 0
20          mov     $60, %rax               # system call 60 is exit
21          syscall                         # call kernel to exit
```

Invocação através da Libc

- Programa em C que invoca a função de biblioteca printf(), que por sua vez chama o system call **write()**



Utilização de sysenter/sysexit

- Na arquitectura Intel a partir do Pentium II, existem duas instruções permitem a realização de system calls mais rapidamente
  - sysenter permite a entrada no sistema sem passar por uma interrupção software
  - sysexit permite a saída do kernel pelo mesmo mecanismo
  - O kernel linux utiliza estas instruções preferencialmente a partir da versão 2.6
- A invocação destas instruções faz-se pela invocação directa de código *assembler* colocado pelo kernel numa página específica de todos os processos (*virtual dynamic shared object* - vdso)
  - O ponto de entrada é designado por *kernel_vsyscall*
  - Endereço pode variar por distribuição e formato executável
- Mais detalhes em:

  manugarg.googlepages.com/systemcallinlinux2_6.html

  www.codeguru.com/cpp/w-p/system/devicedriverdevelopment/article.php/c8223

Sequência de Invocação

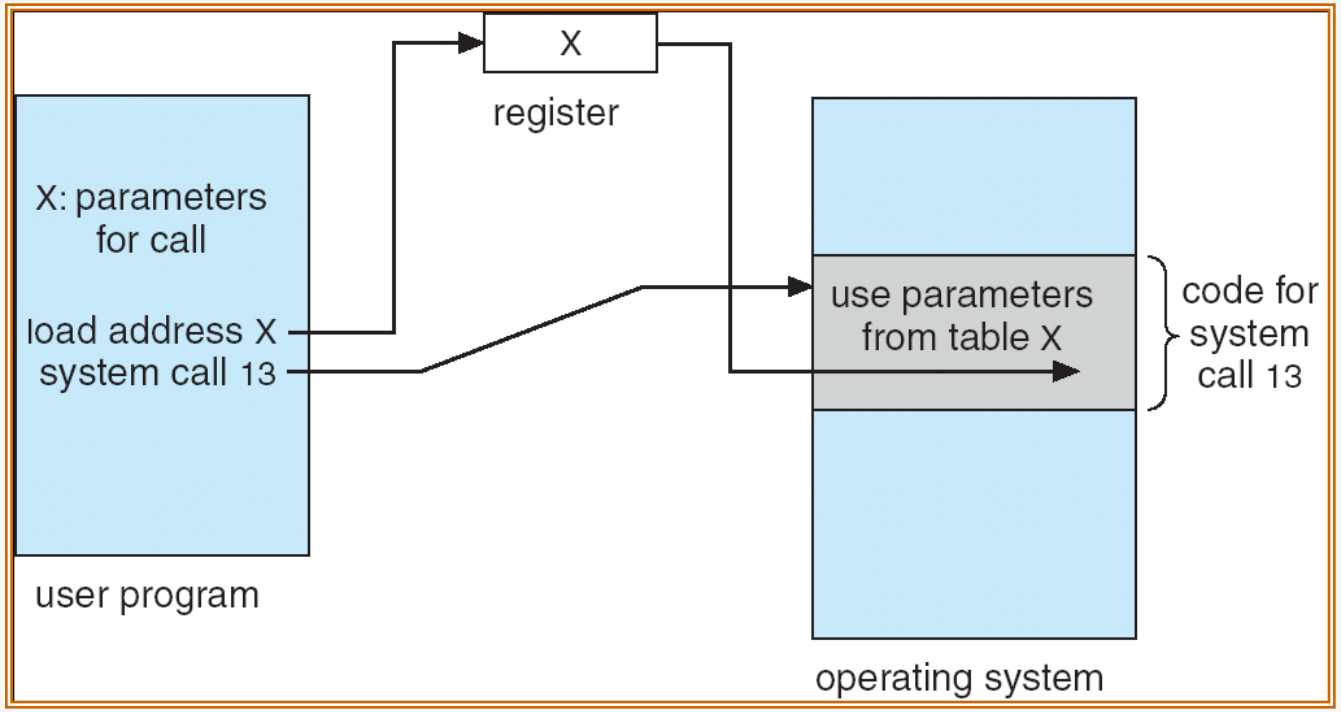
- Programa em Assembler que invoca os system calls **write()** e **exit()** através das instruções sysenter/sysexit

```
    .data
msg:
    .string "Hello World\n"
end:
    .long 0                         # null terminated string

    .text
.globl main
main:
    movl    $1, %ebx                # stdout
    movl    $msg, %ecx              # message
    movl    $(end-msg), %edx        # length
    movl    $4, %eax                # NR_write = 4
    call    *%gs:0x10               # call VSDO entry point

    movl    $0, %ebx                # exit value
    movl    $1, %eax                # NR_exit = 1
    call    *%gs:0x10               # call VDSO entry point
```

Passagem de Parâmetros nos System Call

- É necessário trocar mais informação do que a identificação do *system call*
  - Parâmetros de entrada e saída
  - O tipo exacto e quantidade de parâmetros a passar e receber varia com o SC e o OS
- Três métodos genéricos são utilizados para passar os parâmetros para o SO
  - Mais simples: passar os parâmetros nos registos do processador
    - Método utilizado no Linux
    - Em alguns casos pode haver mais parâmetros do que regs.
  - Os parâmetros são guardados num bloco ou tabela em memória e o endereço do bloco é passado como parâmetro num registo
    - Método utilizado no Windows e Solaris
  - Os parâmetros podem ser *empurrados (pushed)* para a pilha pela aplicação, e *puxados (popped)* pelo SO
- Estes dois últimos métodos não limitam o número e comprimento dos parâmetros passados
  - Sempre que há passagem de dados por referência estes têm de ser copiados para o kernel (in) ou para o processo utilizador (out)
  - Caso das operações de I/O que realizam transferências de dados importantes (leitura ou escritura de ficheiros ou rede)

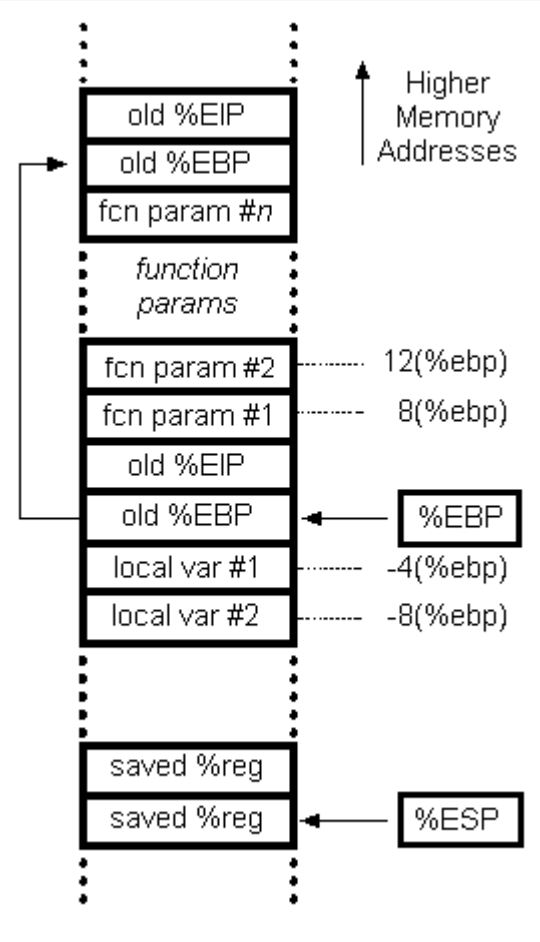
Passagem de Parâmetros por Referência



Passagem de Parâmetros na Pilha

Arquitectura Intel 32 bits (IA32)

Offset	Conteúdo da Pilha
16(%ebp)	3º argumento da função
12(%ebp)	2º argumento da função
8(%ebp)	1º argumento da função
4(%ebp)	Endereço de retorno (EIP)
0(%ebp)	Anterior EBP
-4(%ebp)	Primeira variável local
-8(%ebp)	Segunda variável local
-12(%ebp)	Terceira variável local



http://www.unixwiz.net/techtips/win32-callconv-asm.html

Uma implementação simples da libc (x86)

- Exemplo de implementação simples da função de invocação do syscall `write(fd, buffer, count)` a partir de um programa em C:

```
    .text
.globl _my_write
    .type   _my_write, @function
_my_write:
    pushl   %ebp            /* Save previous stack frame */
    movl    %esp, %ebp      /* Initialize current stack frame */
    pushl   %ebx            /* save used regs */
    pushl   %ecx
    pushl   %edx
    movl    8(%ebp), %ebx   /* file descriptor */
    movl    12(%ebp), %ecx  /* buffer */
    movl    16(%ebp), %edx  /* size */
    movl    $4, %eax        /* syscall_write */
    int     $0x80           /* jump to kernel sysentry */
    popl    %edx            /* restore used regs */
    popl    %ecx
    popl    %ebx
    movl    %ebp, %esp
    popl    %ebp
    ret
```

Salvaguarda do *stack frame* anterior e inicialização do *stack frame* corrente

Salvaguarda dos *registos* utilizados

Passagem dos argumentos da função invocada para os registos adequados

Carregamento do nº de syscall e passagem para o kernel

Restauração do contexto do contexto anterior à invocação e retorno ao programa principal

Uma implementação simples da libc (ia64)

- A implementação na arquitectura 64 bits é muito mais simples, pois a convenção de passagem de parâmetros do compilador é muito parecida com a dos system calls:

```
/*
 * This interface uses the gcc calling convention for ia64:
 * Left to right args are passed in registers rdi, rsi, rdx, rcx, r8, r9...
 * The ia64 syscall argument convention is rdi, rsi, rdx, r10, r8, and r9,
 * with the sycall number in rax.
 * Since the following system calls only use three args, the registers are already
 * set for the syscalls and there is almost nothing to be done.
 */
        .text
        .globl _my_write
        .type   _my_write, @function

_my_write:


        movq    $1, %rax        /* syscall_write */
        syscall                 /* jump to kernel sysentry */
        ret

        .globl _my_read
        .type   _my_read, @function

_my_read:

        movq    $0, %rax        /* syscall_read */
        syscall                 /* jump to kernel sysentry */
        ret
```

Carregamento do nº de syscall em rax e passagem para o kernel

Referências de Código do Kernel

Para aprofundar estes conceitos aconselha-se a análise de alguns excertos de código do kernel disponíveis em: http://lxr.linux.no/

- Definição dos números de System Calls
  - http://lxr.free-electrons.com/source/arch/x86/include/asm/unistd_32.h?v=3.2
- Tabela de System Calls
  - http://lxr.free-electrons.com/source/arch/x86/ia32/ia32entry.S?v=3.2#L505
- Pontos de entrada no Kernel
  - int $80 system_call
    - http://lxr.free-electrons.com/source/arch/x86/ia32/ia32entry.S?v=3.2#L380
  - sysentry system_call
    - http://lxr.free-electrons.com/source/arch/x86/ia32/ia32entry.S?v=3.2#L98
- Criação do Virtual Dynamic Shared Object (VDSO)
  - http://lxr.free-electrons.com/source/arch/x86/vdso/vdso32-setup.c?v=3.2
- Teste e cópia de argumentos dos System Calls (in/out)
  - http://lxr.free-electrons.com/source/arch/x86/lib/usercopy_32.c?v=3.2

Tipos de System Calls

- Controle de Processos
  - fork(), exec(), wait(), exit(), kill(), …
- Gestão de Ficheiros
  - open(), creat(), read(), write(), seek(), …
- IPC
  - pipe(), dup(), shmat(), shmget(), …
- Gestão de Periféricos
  - mount(), umount(), ioctl(), read(), write(), …
- Informação e manutenção
  - stat(), time(), chmod(), …
- Comunicações
  - socket(), bind(), connect(), send(), receive(), …
- Debug
  - ptrace()

Serviços do Sistema Operativo

- Os serviços mais evidentes do SO são as funções directamente relacionadas com o utilizador:
- Interface Utilizador – quase todos os SOs fornecem um tipo de UI
  - Command-Line Interpreter (CLI)
  - Graphics User Interface (GUI)
  - Batch
- Execução de programas – o sistema carrega um programa em memória, executa-o e termina-o, reportando eventuais erros.
- Operações de I/O - Um programa em execução requer operações de I/O relativamente a ficheiros ou periféricos.
- Operações relativas ao Sistema de Ficheiros – a grande maioria de operações das aplicações têm a ver com
  - ficheiros: ler, escrever, apagar, mover
  - directórios listar, procurar, gestão de acessos

Serviços do Sistema Operativo (Cont.)

- Comunicações – troca e/ou partilha de dados entre aplicações residentes no mesmo computador ou em computadores ligados por rede
  - Podem ser feitas por memória partilhada ou através de mensagens (pacotes movimentados pelo SO através de protocolos)
- Detecção de Erros – o SO tem de estar constantemente a par de todos os erros, a tentar reportá-los ou mesmo corrigi-los.
  - Podem ocorrer no CPU ou na memória, periféricos ou na maioria dos casos, em programas utilizador
  - Para cada tipo de erro, o SO toma a acção apropriada para garantir a continuidade da execução do sistema como um todo
  - Facilidades de debug podem aumentar as capacidades dos utilizadores o programadores utilizarem o sistema correctamente

Interface Utilizador - CLI

Command Line Interpreter permite a execução directa de comandos

- A implementação pode ser realizada no kernel ou num programa sistema
  - Podem ter inúmeras variantes
    - Os mais conhecidos são os “**shells**”
  - Princípio: lêem um comando da consola do utilizador e executam a acção correspondente
    - Os comandos podem fazer parte do próprio interpretador
    - Ou podem ser realizados em programas separados
      - Neste último caso adicionar novos comandos não requer a modificação do interpretador
      - É o caso do shell Unix e MS-DOS

Windows CLI


C:\WINDOWS\system32\cmd.exe
C:\Documents and Settings\Jose Rogado\My Documents>dir
Volume in drive C has no label.
Volume Serial Number is 5F27-105E
Directory of C:\Documents and Settings\Jose Rogado\My Documents
03-08-2005 11:54 <DIR> .
03-08-2005 11:54 <DIR> ..
15-07-2005 18:28 <DIR> Altova Projects
17-06-2005 15:36 <DIR> Bluetooth Exchange Folder
06-10-2005 11:49 <DIR> Docs
27-06-2005 09:18 <DIR> Ensino
12-08-2005 19:36 <DIR> Luv
04-07-2005 11:30 <DIR> Mail
07-08-2005 01:55 <DIR> My Albums
03-10-2005 16:31 <DIR> My Downloads
19-06-2005 01:14 <DIR> My eBooks
01-07-2005 14:30 <DIR> My Music
31-08-2005 23:44 <DIR> My Pictures
13-07-2005 23:00 <DIR> My Skype Pictures
06-07-2005 12:57 <DIR> My Skype Received Files
31-07-2005 20:41 <DIR> My Videos
28-08-2005 13:20 <DIR> Sao
0 File(s) 0 bytes
17 Dir(s) 46.107.586.560 bytes free
C:\Documents and Settings\Jose Rogado\My Documents>ipconfig
Windows IP Configuration
Ethernet adapter Wireless Network Connection:
Connection-specific DNS Suffix . : homenet.pt
IP Address. . . . . : 192.168.1.6
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . : 192.168.1.1
Ethernet adapter Local Area Connection:
Media State . . . . . : Media disconnected
C:\Documents and Settings\Jose Rogado\My Documents>
cmd.exe

Unix CLI

bash

```
jrogado@jqr-dsktp:~/system$ pwd
/home/jrogado/system
jrogado@jqr-dsktp:~/system$ cd ..
jrogado@jqr-dsktp:~$ cd ..
jrogado@jqr-dsktp:/home$ ls
jrogado
jrogado@jqr-dsktp:/home$ cd -
/home/jrogado
jrogado@jqr-dsktp:~$ ls
ArchitectureLectures.pdf  Mail     system     The Linux Kernel HOWTO.pdf
ConsumoAguas.sxc          network  test       traffic-test
Desktop                   Ola.sxw  test.c     TransfParqueExpo
Downloads                 Photos   test-page  TransfParqueExpoColor
jrogado@jqr-dsktp:~$ ls -l
total 3516
-rw-------  1 jrogado jrogado 3173520 2005-10-15 14:14 ArchitectureLectures.pdf
-rw-r--r--  1 jrogado jrogado    5279 2005-08-22 13:00 ConsumoAguas.sxc
drwxr-xr-x  3 jrogado jrogado    4096 2005-10-13 02:13 Desktop
drwxr-xr-x  4 jrogado jrogado    4096 2005-08-08 23:26 Downloads
drwx------  7 jrogado jrogado    4096 2005-08-05 12:20 Mail
drwxr-xr-x  2 jrogado jrogado    4096 2005-10-15 16:15 network
-rw-r--r--  1 jrogado jrogado    7812 2005-08-09 00:06 Ola.sxw
drwxr-xr-x  2 jrogado jrogado    4096 2005-08-22 00:51 Photos
drwxr-xr-x  2 jrogado jrogado    4096 2005-10-16 17:27 system
-rwxr-xr-x  1 jrogado jrogado   11515 2005-10-01 17:36 test
-rw-r--r--  1 jrogado jrogado      31 2005-10-01 17:36 test.c
-rw-r--r--  1 jrogado jrogado    4396 2005-08-11 13:58 test-page
-rw-r--r--  1 jrogado jrogado  120750 2005-09-30 10:55 The Linux Kernel HOWTO.pdf
-rw-------  1 root    root        954 2005-10-13 23:18 traffic-test
-rw-r--r--  1 jrogado jrogado   77907 2005-08-22 13:16 TransfParqueExpo
-rw-r--r--  1 jrogado jrogado  132393 2005-08-22 13:25 TransfParqueExpoColor
jrogado@jqr-dsktp:~$ cat test.c
main()
{
        printf("Hello\n");
}
jrogado@jqr-dsktp:~$
```

Scripting

- Os CLIs também possibilitam a execução de uma sequência de comandos ou instruções, designado por script
  - Constituem uma linguagem de programação específica do sistema
- É a forma mais utilizada para realizar tarefas de configuração ou manutenção do sistema
  - No decorrer do Bootstrap, são executados centenas de scripts que permitem o arranque da totalidade dos serviços associados ao Sistema Operativo
- Existem inúmeras linguagens de scripting
  - cmd: Windows, contidos em ficheiros .bat (batch)
  - Shell: Unix, contidos em ficheiros .sh (shell scripts)
  - Python: também uma linguagem de programação, independente da plataforma

Exemplo de batch

```
title Script de Teste
@echo off

:start
cls
echo O que pretende fazer?
echo.
echo 0. Sair
echo 1. Ir para a pasta indicada e listar o conteudo
echo 2. Listar o conteudo da pasta corrente
echo.

set /p choice="Indique a sua escolha: "
if %choice%==0 exit
if %choice%==1 goto changedir
if %choice%==2 goto curdir

echo Escolha invalida: %choice%
echo.
pause
cls
goto start
:changedir
set /p directory="Indique a nova pasta: "
echo Changing to directory %directory%
cd %directory%
:curdir
echo.
dir
echo.
pause
goto start
```

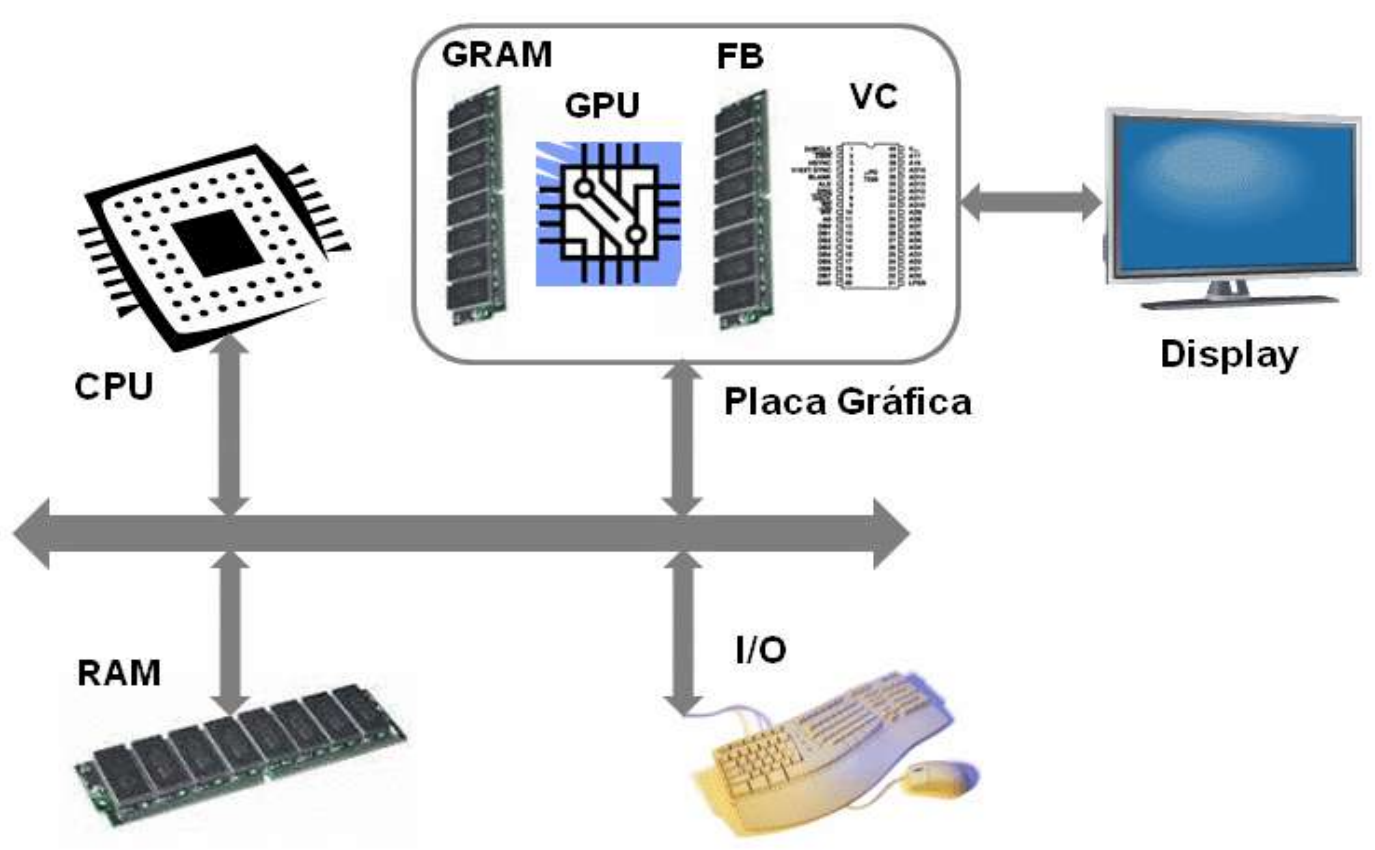
Exemplo de Shell script

```
# Copia todos os ficheiros C da pasta corrente
# para a pasta passada em argumento

case "$1" in
    "")
        echo Usage: $0 {dir}
        exit 1
        ;;
    *)
        ;;
esac

if [ -d $1 ]
    then
    for i in *.c
        do
        echo "Copying file" $i to $1
        cp $i $1
        done
     else
    echo $1 does not exist
    exit 1
fi
exit 0
```


Interface Utilizador - GUI

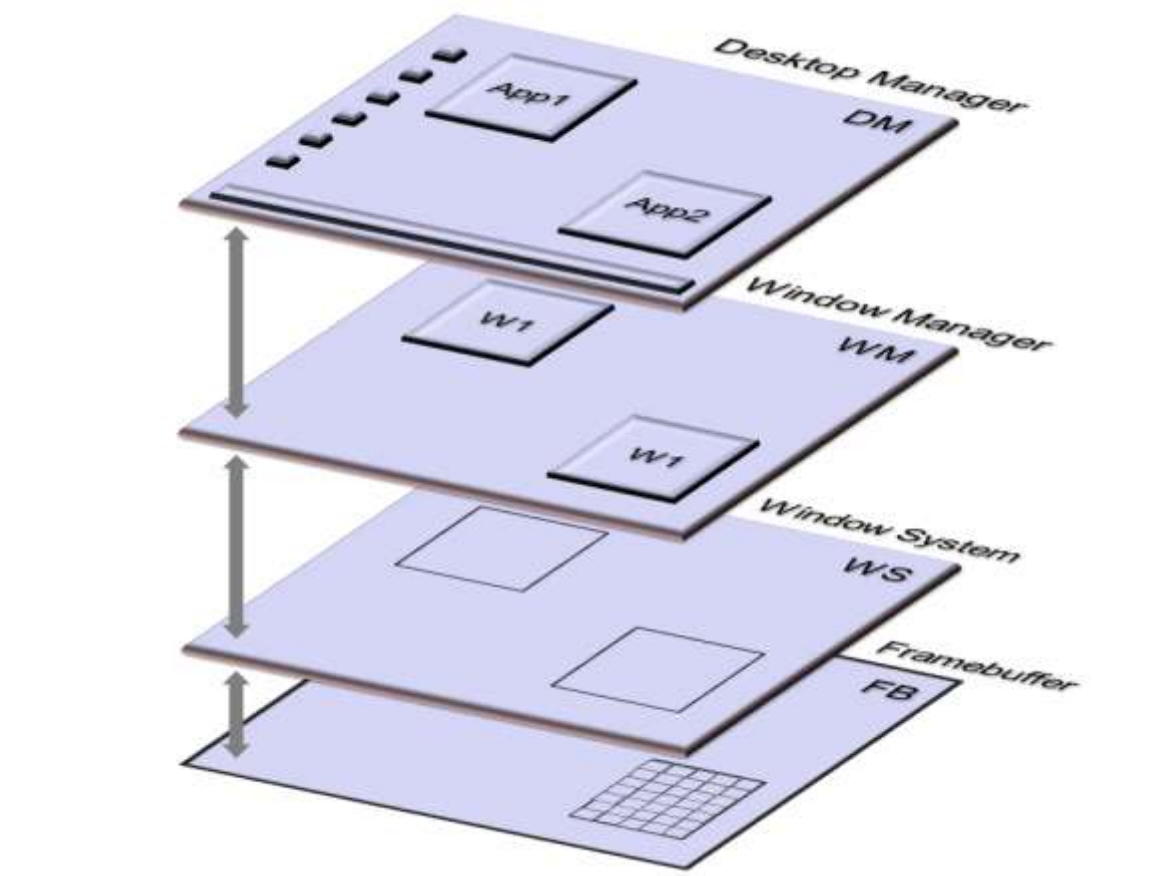
- **Graphical User Interface**
  - Mais intuitiva e com maior grau de usabilidade
  - Tornou o computador utilizável pelo público em geral
- É uma interface mais adaptada aos computadores de uso pessoal
  - Utiliza um rato, um teclado e um monitor com sistema de janelas
  - **Icons** representam ficheiros, programas e acções
  - A movimentação do rato e acção dos botões podem ser programadas de acordo com o *look and feel* das aplicações
    - Obter informação, escolher opções, executar comandos, abrir directórios ou pastas
  - Criada na Xerox PARC cerca em 1973
    - Generalizada com os Apple McIntosh
- A maioria dos sistemas actuais inclui interfaces CLI e GUI
  - Microsoft Windows é um GUI com o CLI cmd
  - Apple Mac OS X usa o “Aqua” GUI interface por cima de um kernel de tipo UNIX, com vários shells disponíveis
  - Linux tem um CLI com várias GUIs opcionais (Gnome, KDE, Cinnamon, …)

Placa Gráfica

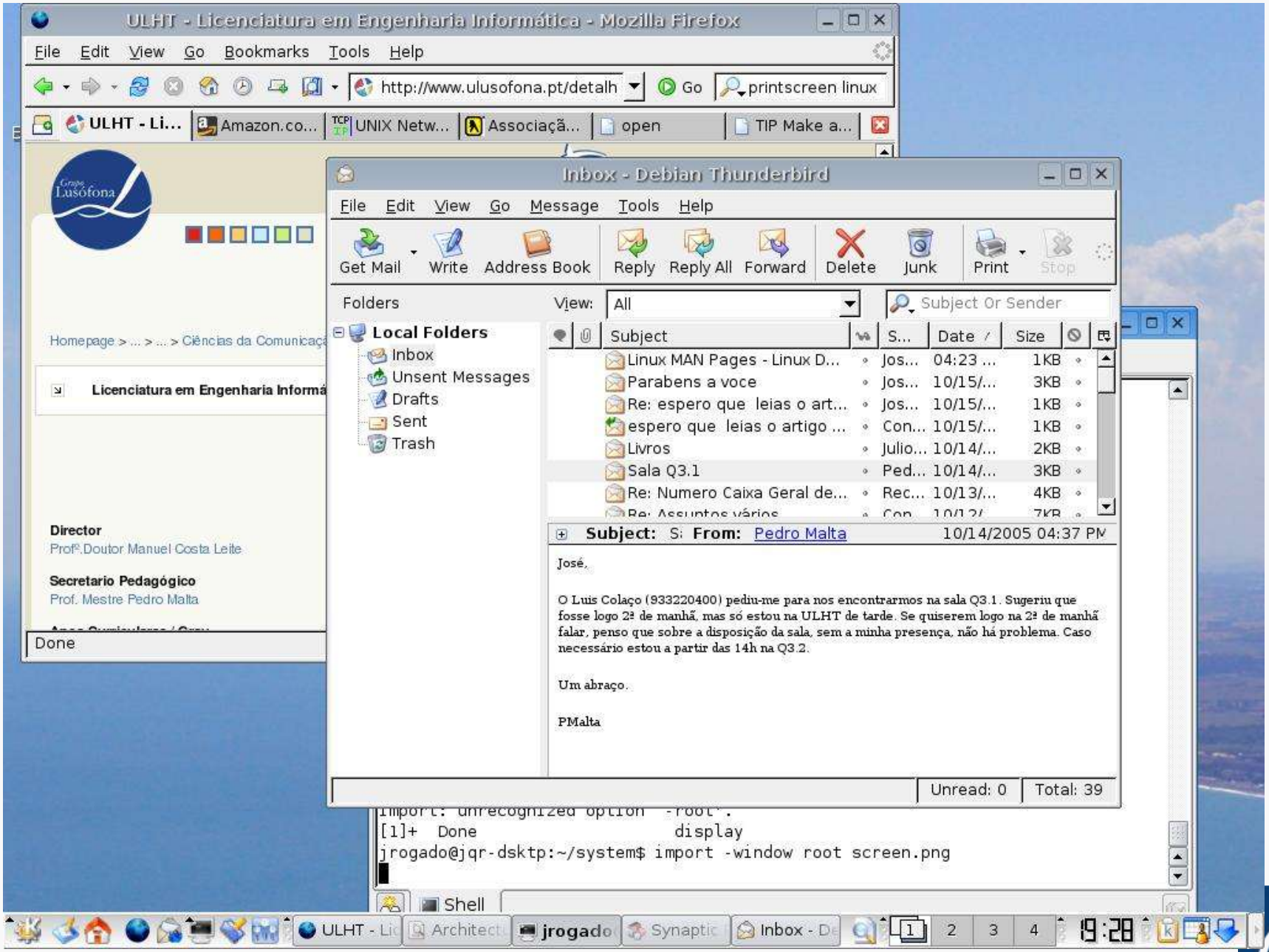


- De forma simplificada os diversos componentes de uma placa gráfica, são:
  - GRAM: Graphic RAM
  - GPU: Graphical Processing Unit
  - FB: Frame Buffer
  - VC: Video Controller

Componentes Software de um GUI



Linux KDE

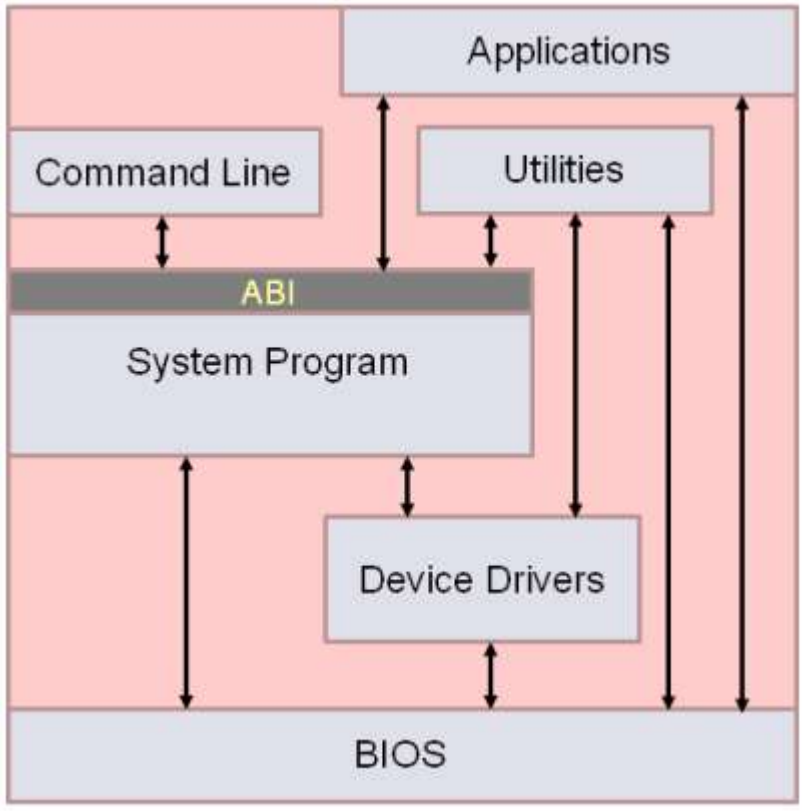

ULHT - Licenciatura em Engenharia Informática - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://www.ulusofona.pt/detalh
printscreen linux
Inbox - Debian Thunderbird
File Edit View Go Message Tools Help
Get Mail Write Address Book Reply Reply All Forward Delete Junk Print Stop
Folders View: All
Subject Or Sender
Local Folders
Inbox
Unsent Messages
Drafts
Sent
Trash
Subject S... Date Size
Linux MAN Pages - Linux D... Jos... 04:23 ... 1KB
Parabens a voce Jos... 10/15/... 3KB
Re: espero que leias o art... Jos... 10/15/... 1KB
espero que leias o artigo ... Con... 10/15/... 1KB
Livros Julio... 10/14/... 2KB
Sala Q3.1 Ped... 10/14/... 3KB
Re: Numero Caixa Geral de... Rec... 10/13/... 4KB
Subject: From: Pedro Malta 10/14/2005 04:37 PM
José,
O Luis Colaço (933220400) pediu-me para nos encontrarmos na sala Q3.1. Sugeriu que fosse logo 2ª de manhã, mas só estou na ULHT de tarde. Se quiserem logo na 2ª de manhã falar, penso que sobre a disposição da sala, sem a minha presença, não há problema. Caso necessário estou a partir das 14h na Q3.2.
Um abraço.
PMalta
Unread: 0 Total: 39
Homepage > ... > ... > Ciências da Comunicaç
Licenciatura em Engenharia Informá
Director
Profº.Doutor Manuel Costa Leite
Secretario Pedagógico
Prof. Mestre Pedro Malta
Done
[1]+ Done display
jrogado@jqr-dsktp:~/system$ import -window root screen.png
Shell
ULHT - Li
Architect
jrogado
Synaptic
Inbox - De
1 2 3 4
19:28

Arquitectura e Implementação de SOs

- A arquitectura interna dos Sistemas Operativos pode variar bastante de uma versão para outra
- Para a definir deve-se começar por estabelecer os objectivos e as especificações
- Estes dependem fortemente dos objectivos e do tipo de hardware
- Diferentes pontos de vista:
  - Utilizadores: deve ser de fácil compreensão e utilização, fiável, seguro e rápido
  - Conceptores: deve ser fácil de desenhar, implementar e manter, deve ser flexível, fiável, eficiente e não ter erros (bugs)
- Muitas vezes estes pontos de vista não são fáceis de conciliar
  - A funcionalidade, simplicidade e usabilidade são difíceis de conseguir simultaneamente
- É importante separar conceitos:
  - Política: o que fazer?
  - Mecanismo: como fazer?
  - A separação garante flexibilidade em futuras mudanças

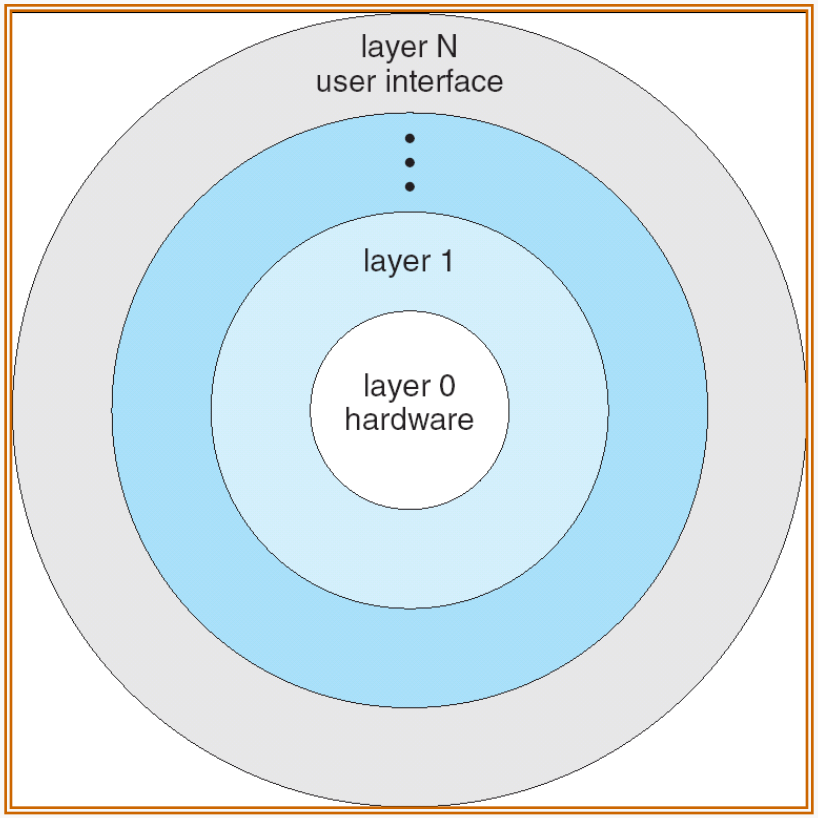
Arquitectura Simples: MS-DOS

- Concebido para fornecer o máximo de funcionalidades num mínimo de espaço
- Não é um sistema modular
  - Não tem interfaces e níveis de funcionalidade bem definidos
- As aplicações acedem directamente aos gestores de periféricos ou ao BIOS
- O BIOS representa um primeiro nível de abstracção e portabilidade



Arquitectura em Camadas

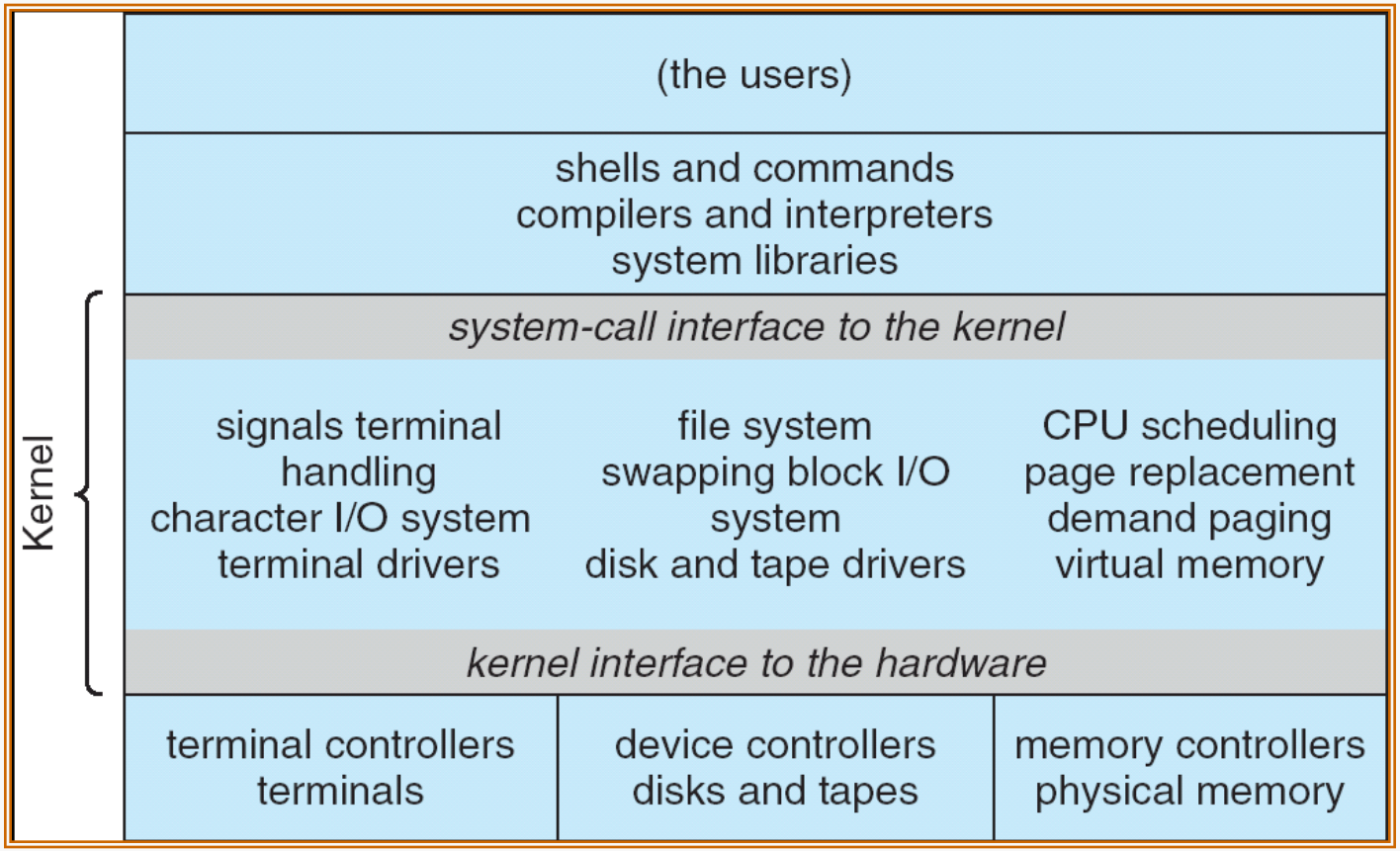
- O SO é dividido em várias camadas ou níveis, cada um construído em cima do anterior (ex: sistemas de *Mainframes* – VMS, Multics)
  - O nível mais baixo é o hardware
  - O mais elevado é a interface utilizador
- O princípio da modularidade implica que os níveis sejam escolhidos de forma a que cada um só utilize serviços dos níveis inferiores



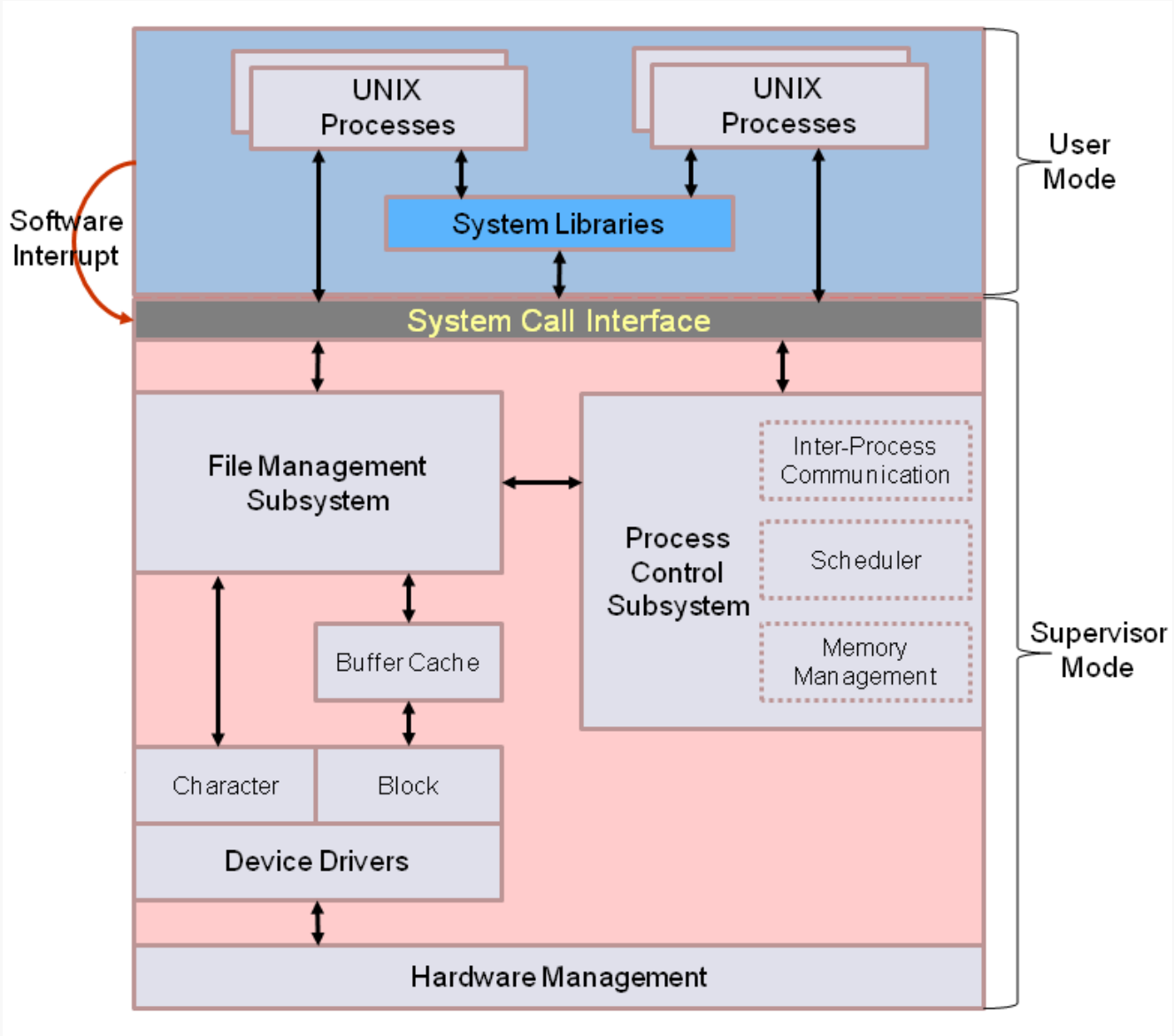
Arquitectura de tipo Unix

- O sistema Unix original era limitado pelas funcionalidades do hardware
  - Minicomputador Digital PDP-11 de 16 bits e memória segmentada
- Mas rapidamente começou a ser utilizado em hardware mais evoluído
- É consistido por **dois** níveis distintos:
  - Os programas sistema e as aplicações
  - O núcleo (*kernel*)
    - É consistido por toda a funcionalidade que está abaixo da interface dos *system calls*
    - Fornece a gestão de processos, de memória e de ficheiros, os protocolos de rede, assim como as funções de mais baixo nível
    - Muitas funcionalidades para um só nível
  - As versões iniciais eram pouco modulares

Arquitectura Original do Sistema UNIX

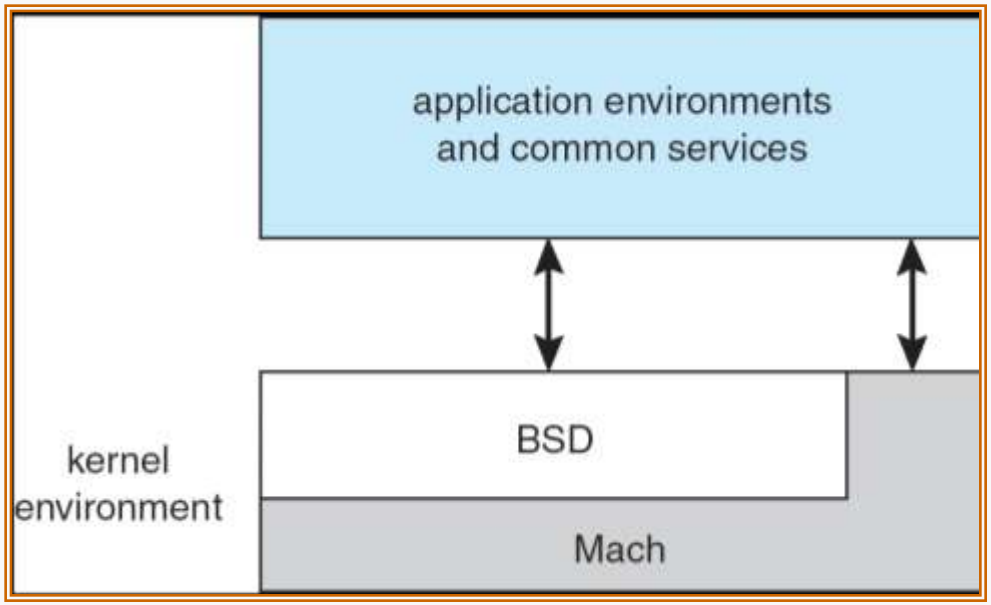


Componentes do Sistema Unix



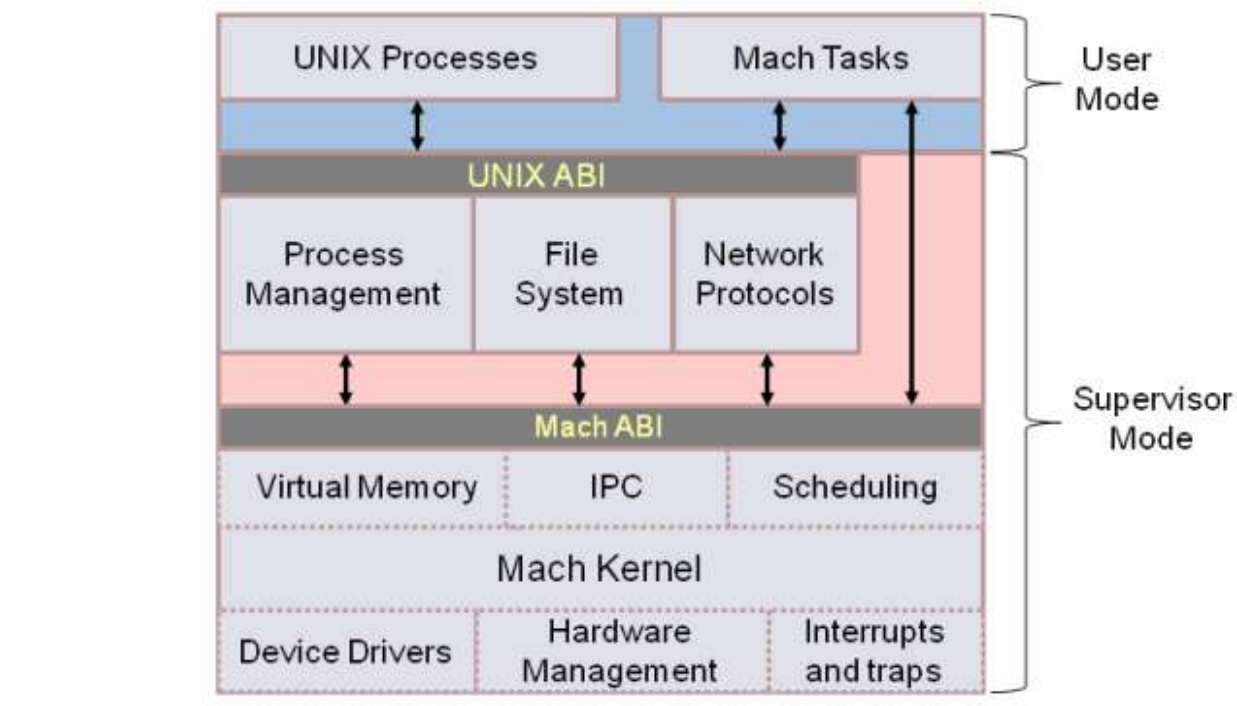
Adaptado de Bach, M., "The Design of the Unix Operating System", 1986, Ed. Prentice Hall

Evolução da Arquitectura Unix



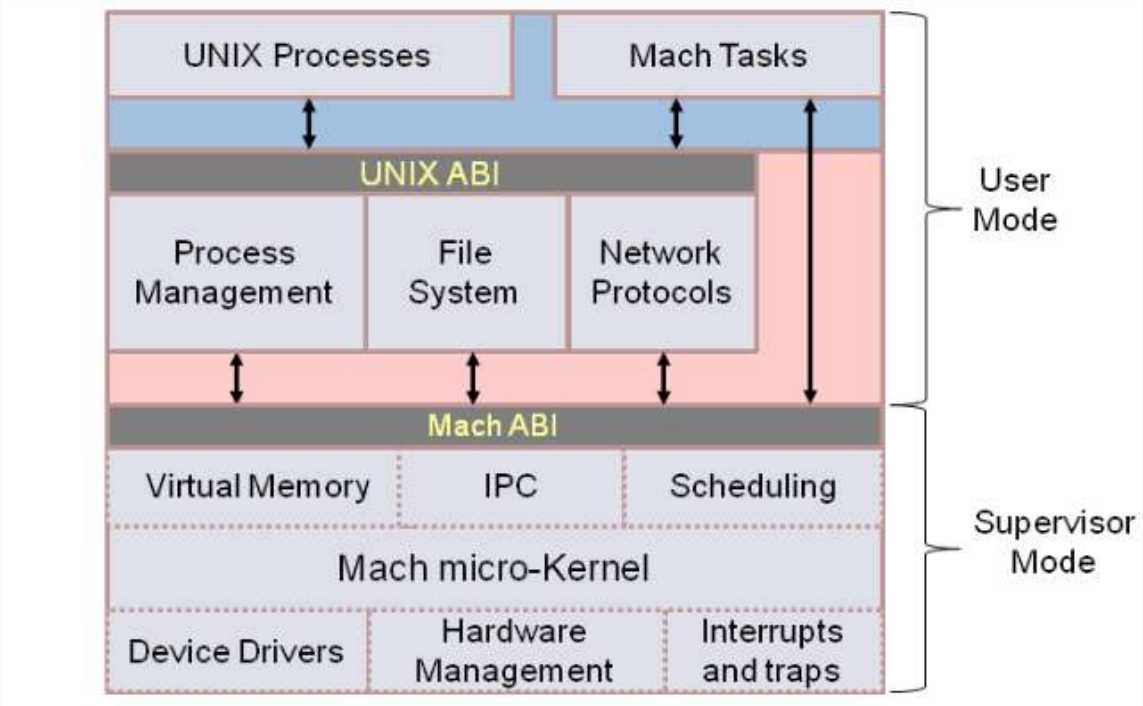
- Uma das evoluções mais radicais na arquitectura do sistema Unix foi realizada a partir de 1985 na *Carneghie-Mellon University* com o projecto Mach.
  - O núcleo foi modificado a partir do sistema BSD, tendo as funções de mais baixo nível sido separadas num módulo designado por micro-núcleo Mach 2.5.
- Os serviços do sistema estão separados do núcleo através de uma interface de mensagens internas ao sistema
  - Sistema de Ficheiros, Processos, Protocolos, etc..
- Este projecto deu origem a uma nova geração de sistemas
  - OSF/1 MK com núcleo Mach 3.0 => Digital Unix
  - Mac OS/X

Detalhe da Arquitectura Mach 2.5



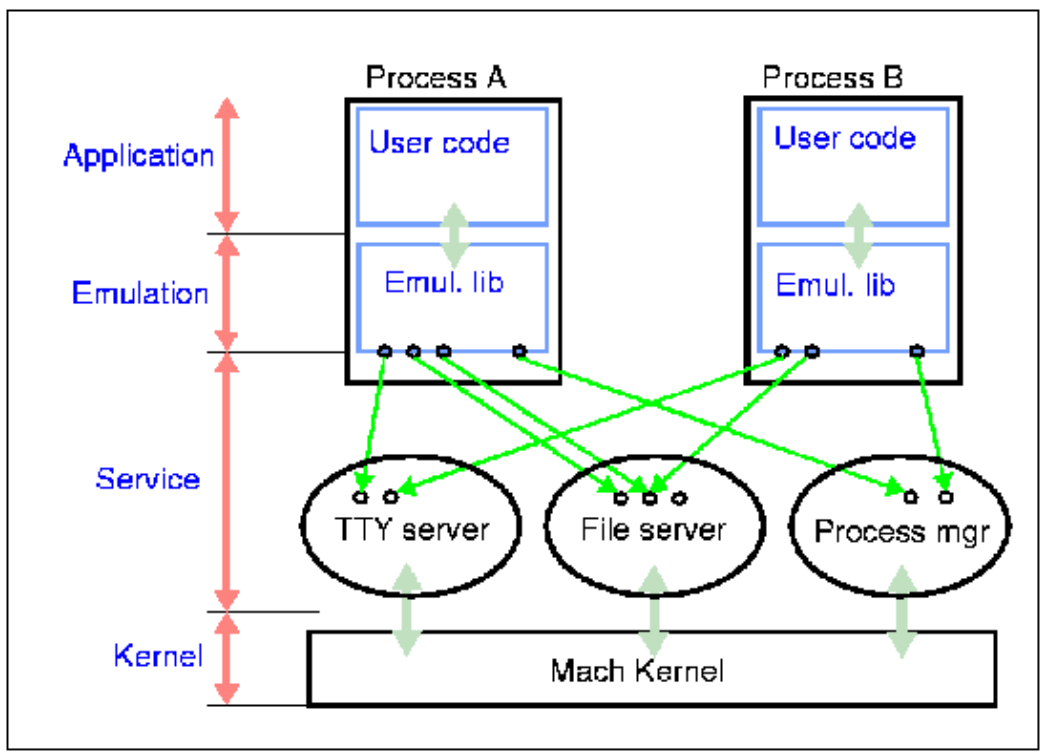
- O sistema fornece dois tipos de Application Binary Interfaces (ABIs):
  - Uma de tipo Unix clássico permitindo a compatibilidade com as aplicações existentes
  - Uma outra dando acesso às funcionalidades específicas do núcleo Mach que permitem implementar aplicações com conceitos diferentes

Evolução: conceito de Micro-Núcleo



- O Sistema Mach 3.0 baseia-se no conceito de micro-kernel e torna-se ainda mais modular
  - As serviços do sistema são implementadas em servidores que se executar em modo utilizador
  - Só os componentes básicos correm em modo supervisor*: o* ***µ-kernel***

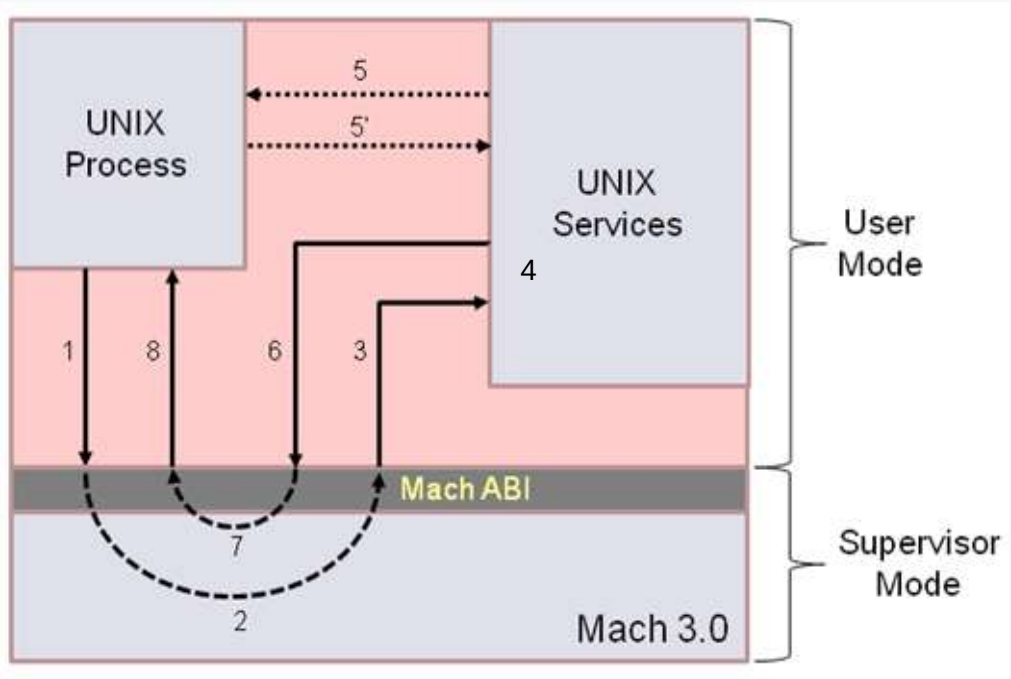
Comunicação por Mensagens



- A comunicação entre as aplicações e os servidores faz-se por mensagens
- Utilização de um mecanismo de comunicação entre processos
  - MACH Inter Process Communication (IPC)

Um *system call* UNIX em Mach 3.0

1. Invocação do *system call* através de um *software interrupt* no contexto da aplicação.
2. Transição para o *µ-Kernel*, que detecta a invocação de uma funcionalidade UNIX.
3. Envio de uma mensagem do *µ-Kernel* ao(s) servidor(es) UNIX, com os parâmetros do *system call* invocado.
4. Recepção da mensagem pelo servidor, identificação do *system call* e execução da funcionalidade invocada.
5. Possível interacção entre o servidor e a aplicação, caso haja transferências de dados por referência (caso p. ex.: do read ou do write)
6. Envio da mensagem de retorno do servidor ao *µ-Kernel* contendo os resultados do *system call*.
7. Recepção da mensagem pelo *µ-Kernel* e criação do contexto para retornar os resultados à aplicação
8. Retorno do *software interrupt* para a aplicação que invocou o *system call*.



Crítica da Arquitectura Micro-Núcleo

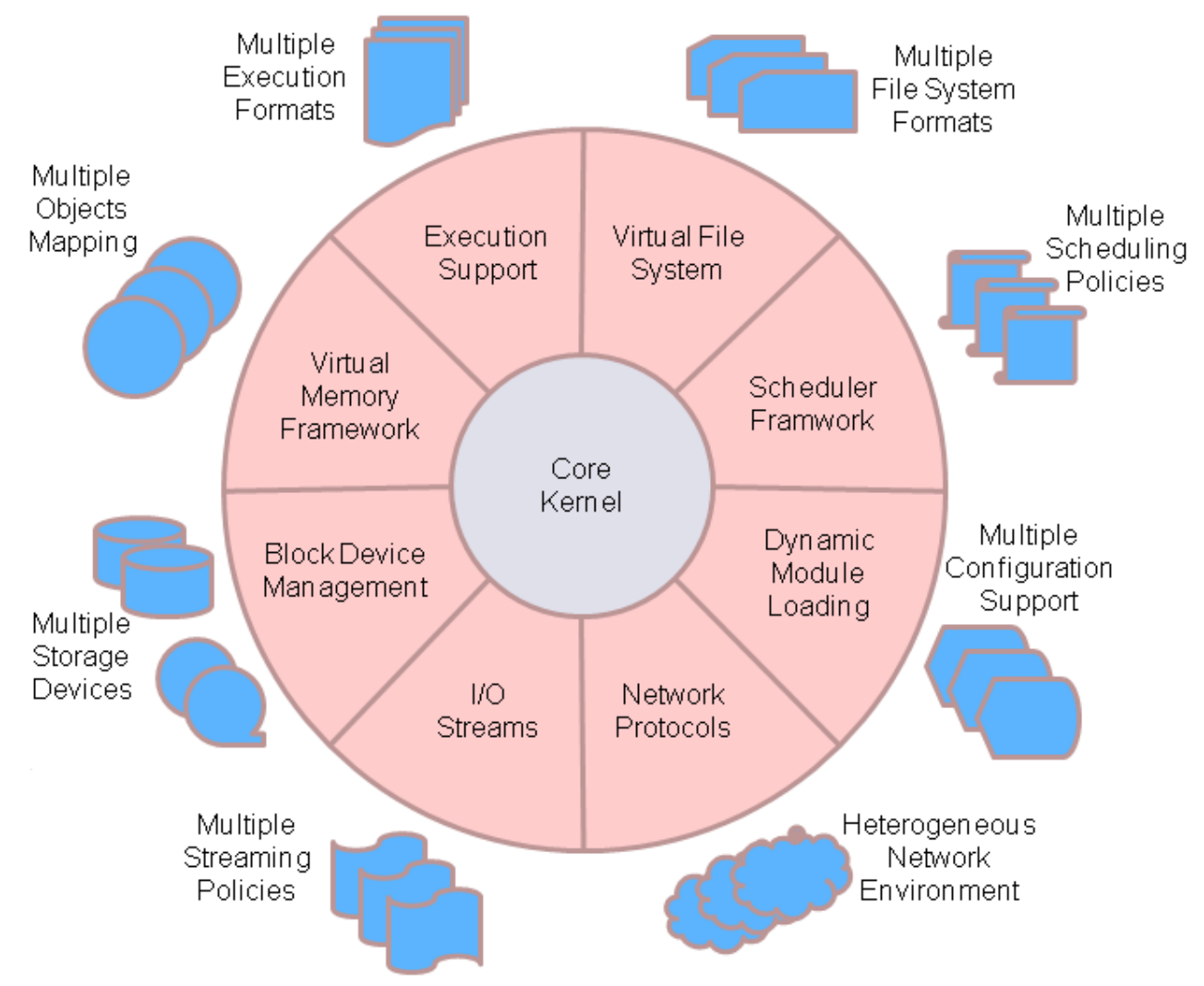
- Benefícios:
  - Mais fácil modificar e manter o micro-núcleo
  - Mais fácil suportar novas arquitecturas hardware
    - Todas as dependências estão concentradas no micro-núcleo
  - Mais fiável e seguro
    - Menos código em modo "*supervisor*"
- Desvantagens:
  - Pior desempenho na comunicação entre serviços e entre estes e o núcleo
  - Versões posteriores voltam a repor os servidores em modo *supervisor*
- Referências
  - Mach 3.0
    - http://www.cs.cmu.edu/afs/cs/project/mach/public/www/mach.html
  - Chorus OS
    - http://www.experimentalstuff.com/Technologies/ChorusOS/index.html
  - Micro-núcleo Minix 3
    - http://www.minix3.org/doc/herder.html

Herança: Arquitectura Modular

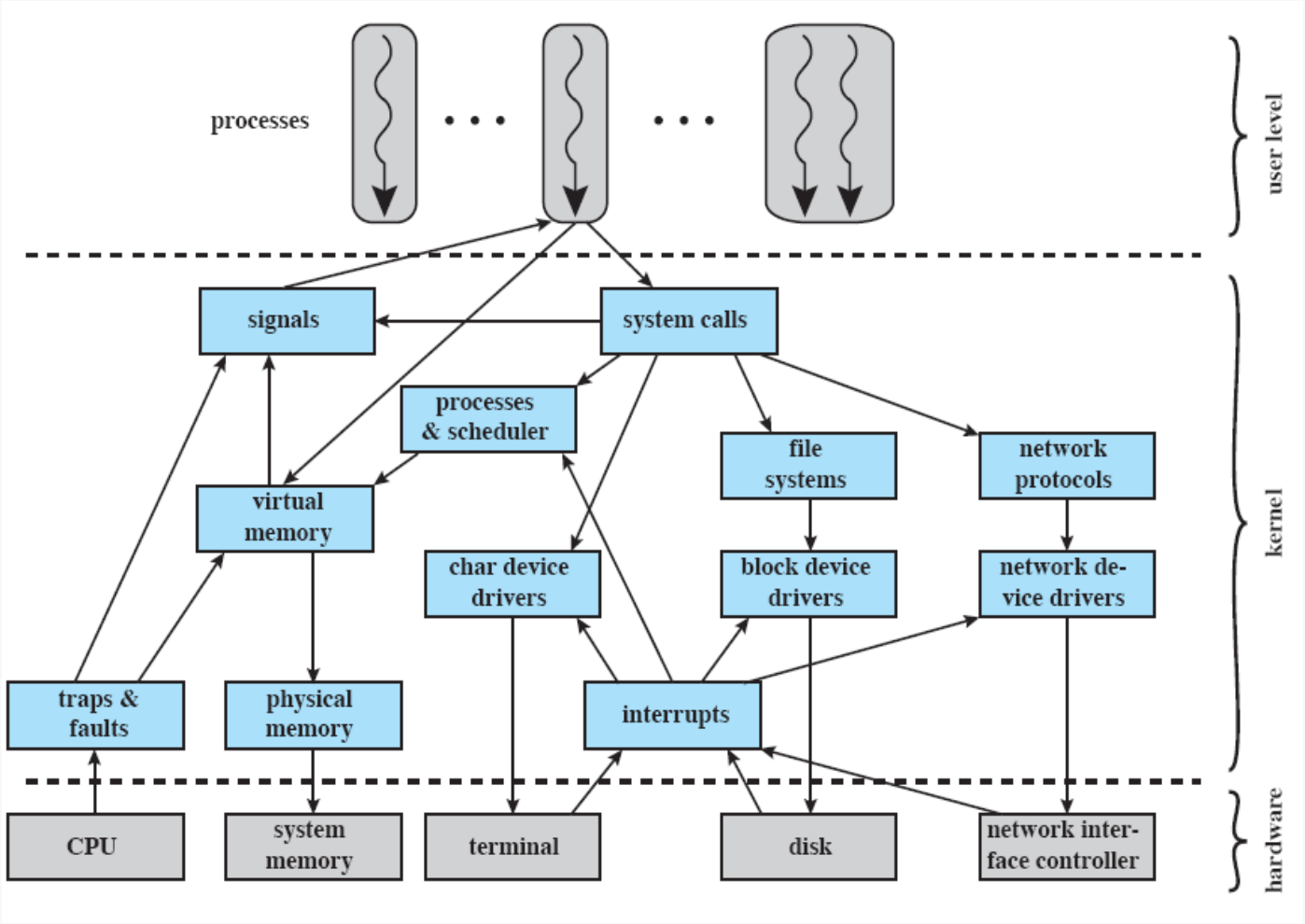
- A maioria dos sistemas operativos modernos são baseados em arquitecturas modulares
  - Utilizam uma aproximação orientada aos objectos
  - Os principais componente funcionais são independentes e comunicam por interfaces bem definidas
  - Os componentes podem ser carregados dinamicamente no núcleo
- Aproximação inspirada no conceito de micro-núcleo
  - Separação em camadas com níveis de abstracção distintos
  - Isolamento das funcionalidades básicas numa camada independente dos serviços de mais alto nível
  - Implementação dos serviços sistema em módulos distintos, com possibilidade de configuração dinâmica
  - Manutenção dos serviços em modo supervisor para garantia de desempenho
- Arquitecturas actuais, flexíveis e dinâmicas
  - Ex: Mach OS, Linux, Windows

Arquitectura Unix Modular

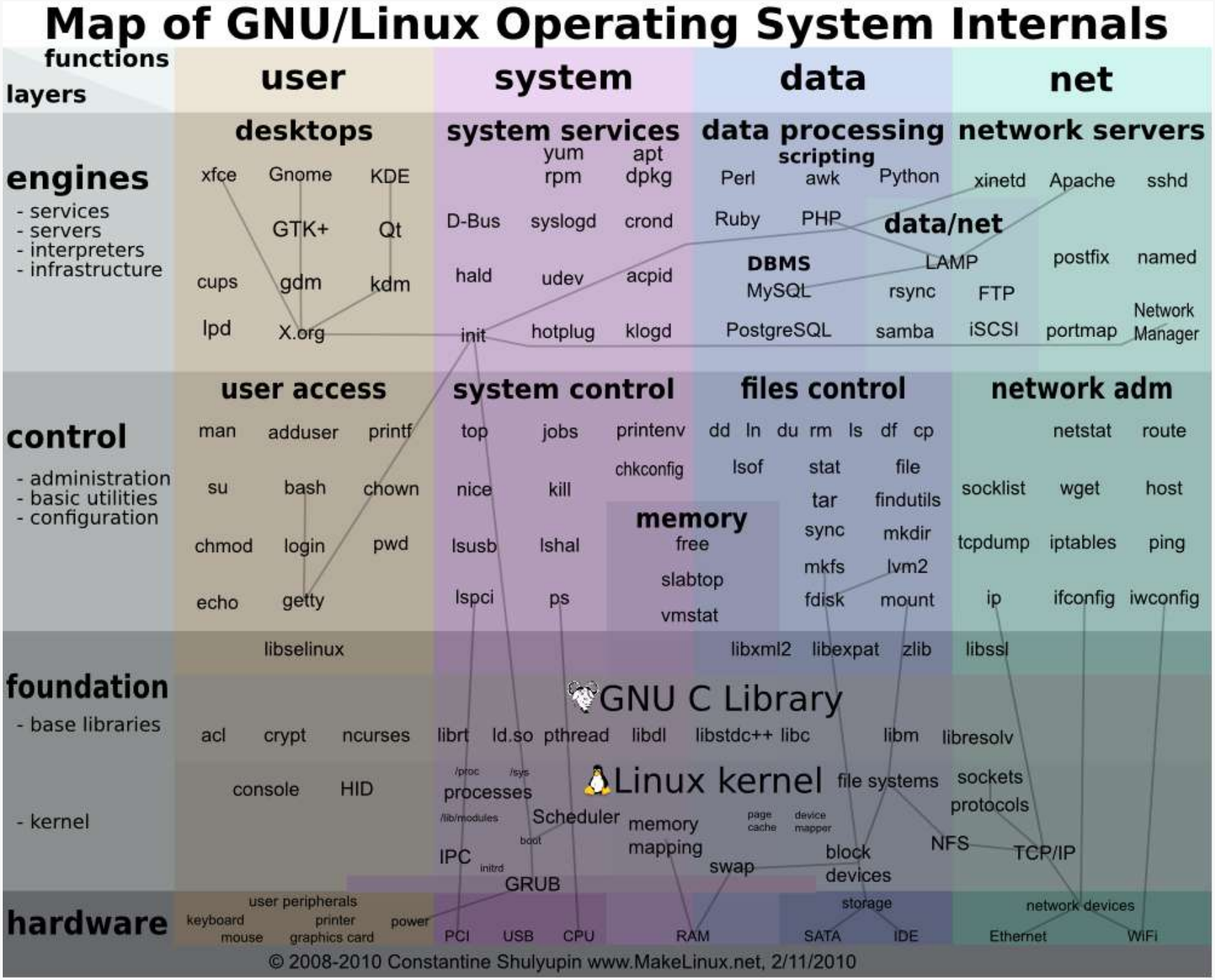
Adaptado de Vahalia, U., "Unix Internals: the new Frontiers", 1995, Ed. Prentice Hall



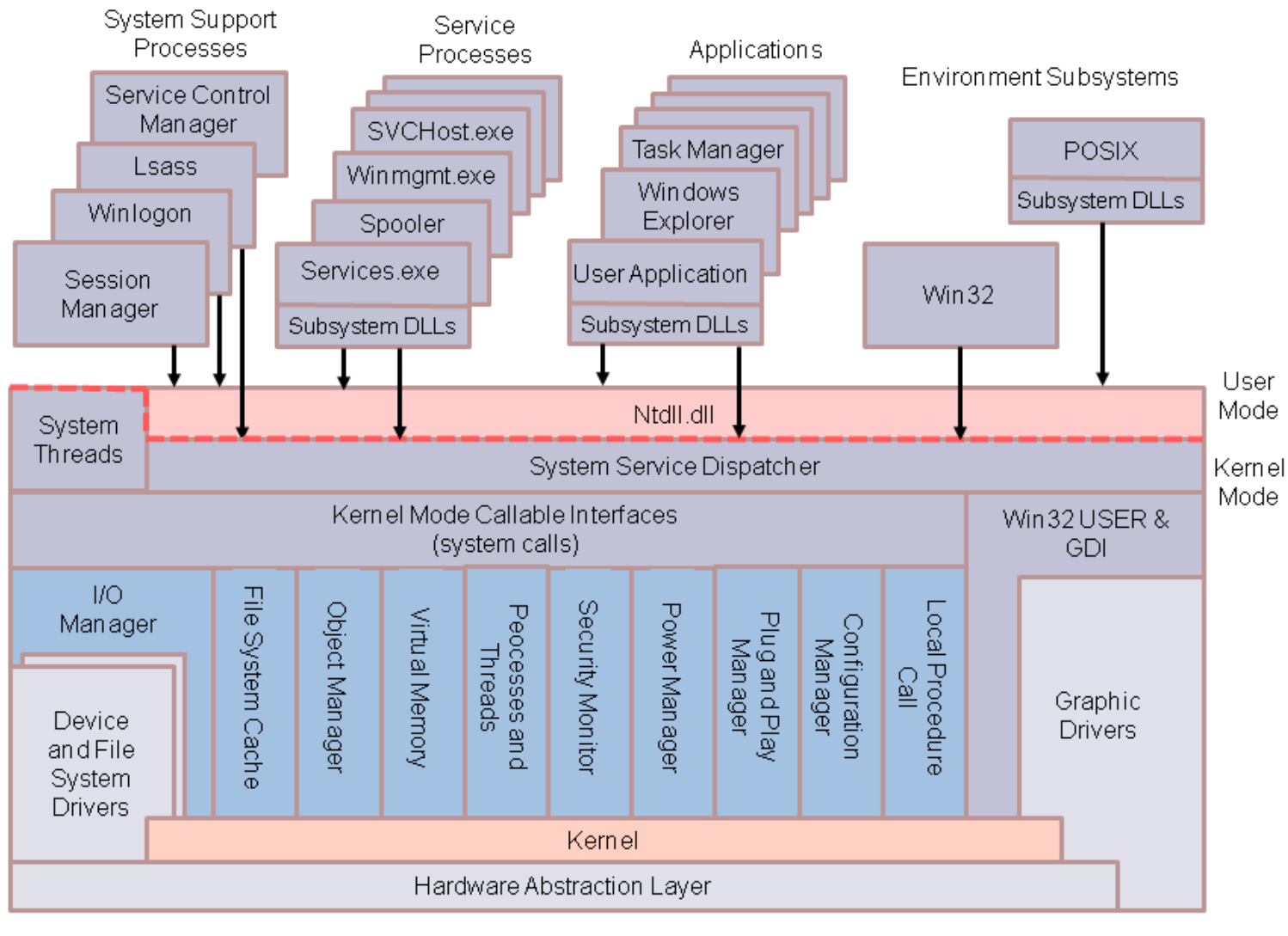
Arquitectura Genérica Linux



Source: Stallings, W., "Operating Systems: Internals and Design Principles"


Map of GNU/Linux Operating System Internals
functions
layers
user
system
data
net
desktops
system services
data processing
network servers
engines
- services
- servers
- interpreters
- infrastructure
xfce
Gnome
KDE
GTK+
Qt
cups
gdm
kdm
lpd
X.org
yum
apt
rpm
dpkg
D-Bus
syslogd
crond
hald
udev
acpid
init
hotplug
klogd
scripting
Perl
awk
Python
Ruby
PHP
data/net
DBMS
MySQL
PostgreSQL
LAMP
rsync
samba
xinetd
Apache
sshd
FTP
iSCSI
postfix
named
portmap
Network Manager
control
- administration
- basic utilities
- configuration
user access
system control
files control
network adm
man
adduser
printf
su
bash
chown
chmod
login
pwd
echo
getty
top
jobs
printenv
chkconfig
nice
kill
lsusb
lshal
lspci
ps
memory
free
slabtop
vmstat
dd
ln
du
rm
ls
df
cp
lsof
stat
file
tar
findutils
sync
mkdir
mkfs
lvm2
fdisk
mount
netstat
route
socklist
wget
host
tcpdump
iptables
ping
ip
ifconfig
iwconfig
libselinux
libxml2
libexpat
zlib
libssl
foundation
- base libraries
- kernel
GNU C Library
acl
crypt
ncurses
librt
ld.so
pthread
libdl
libstdc++
libc
libm
libresolv
console
HID
/proc
/sys
processes
Linux kernel
file systems
sockets
protocols
/lib/modules
Scheduler
memory mapping
page cache
device mapper
boot
IPC
initrd
swap
block devices
NFS
TCP/IP
GRUB
hardware
user peripherals
keyboard
mouse
printer
graphics card
power
PCI
USB
CPU
RAM
storage
SATA
IDE
network devices
Ethernet
WiFi
© 2008-2010 Constantine Shulyupin www.MakeLinux.net, 2/11/2010

Arquitectura Genérica NT



Russinovich, M., Solomon, D. Ionescu, A.,"Windows Internals", 6/e 2012, Ed. Microsoft Press, ISBN-10: 0735648735

Detalhe Arquitectura NT

Process Management
Memory Management
File Management
Device Mgmt Infrastructure

Process
T T T

Process
T

Process
T T T T T

Libraries

Subsystem

Subsystem

Subsystem

User

Supervisor

NT Executive

NT Kernel

Hardware Abstraction Layer

I/O Subsystem

Processor(s)

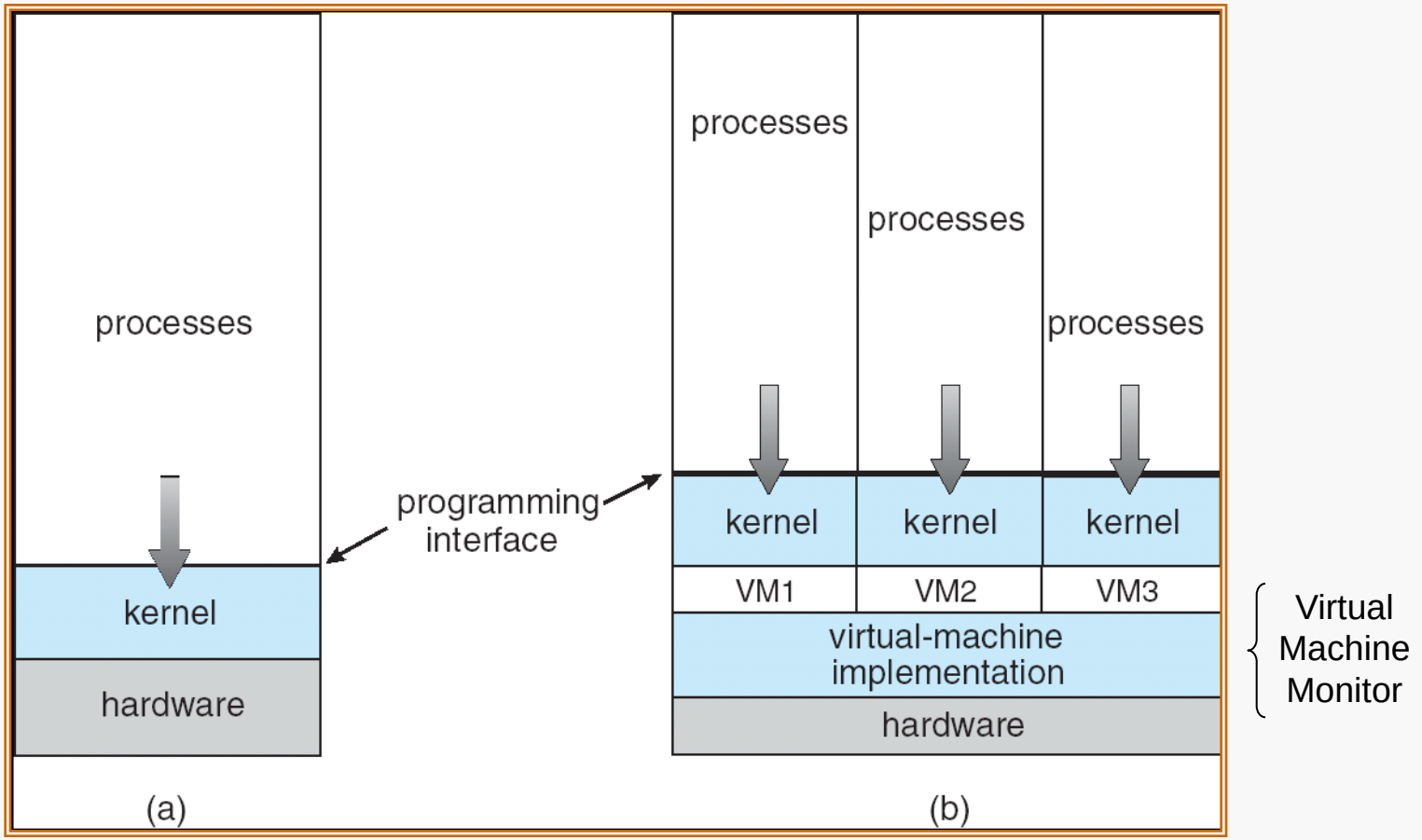
Main Memory

Devices

Noção de Máquina Virtual

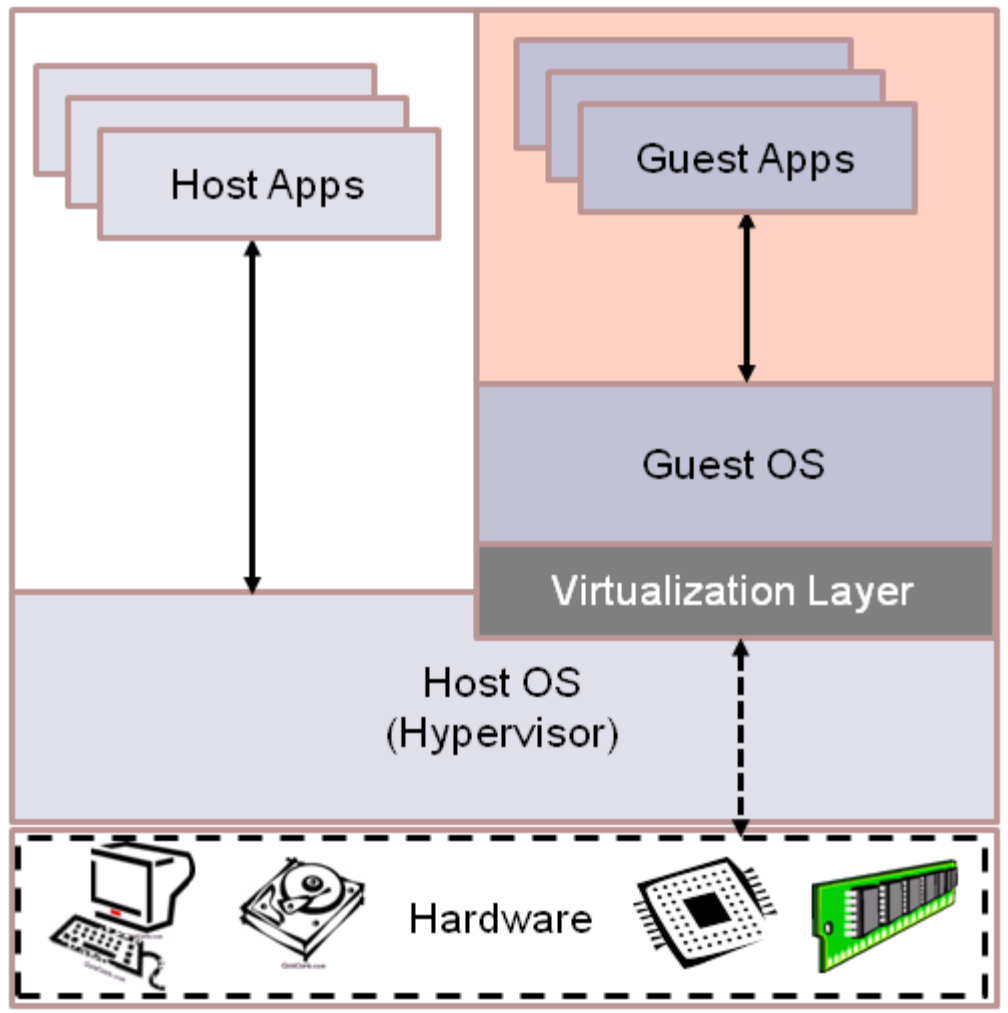
- A noção de *máquina virtual* leva o conceito das níveis de abstracção até às últimas consequências
  - Considera a camada do Sistema Operativo como um tipo de hardware com características específicas
- Uma máquina virtual fornece uma interface *idêntica* à do hardware que simula
- Os recursos do computador real são partilhados de forma a criar a noção de máquina virtual
  - Ex: O sistema operativo cria a ilusão de que múltiplos sistemas operativos se estão a executar simultaneamente
    - Cada um com o seu próprio processador e a sua memória virtual
- Um sistema baseado em máquina virtual é uma excelente plataforma para investigação e desenvolvimento de sistemas operativos
  - O desenvolvimento é feito na máquina virtual, de forma a não interromper o funcionamento do sistema
- O primeiro OS com implementação de máquina virtual foi o IBM VM
  - VM/370 - 1972

Arquitectura da Máquina Virtual

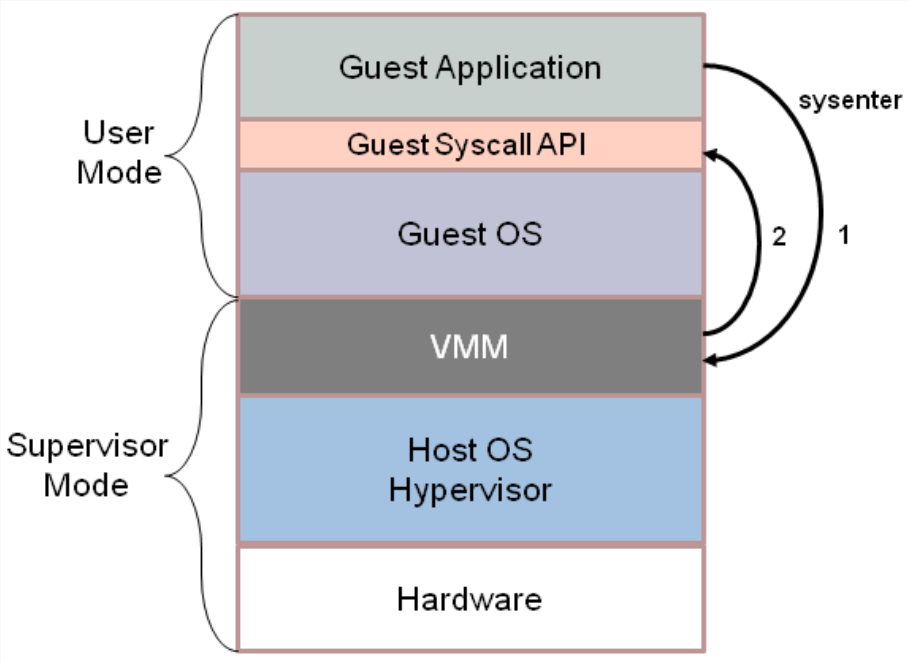


(a) Sistema Normal (b) Máquina Virtual

Detalhe

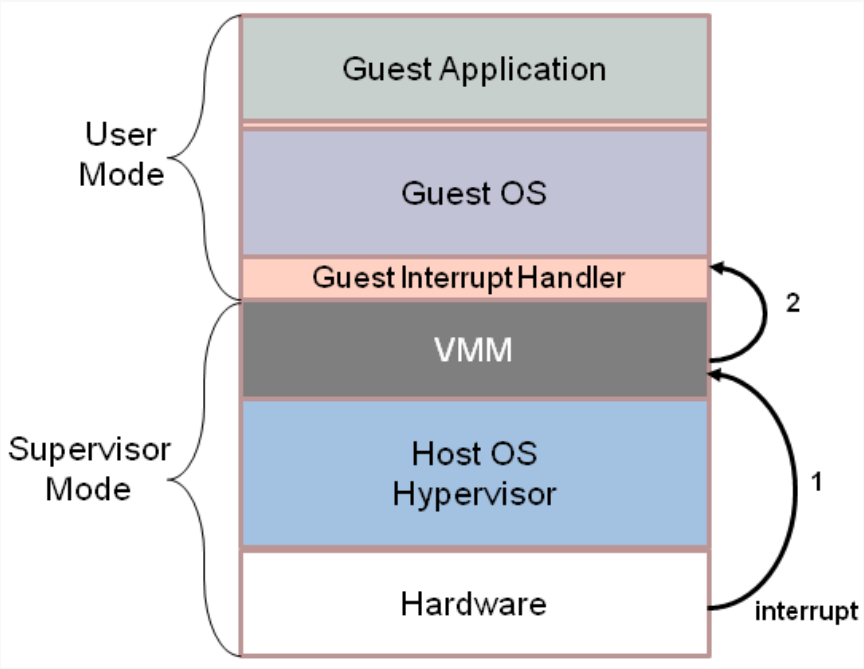

Host Apps
Guest Apps
Guest OS
Virtualization Layer
Host OS
(Hypervisor)
Hardware

Gestão dos System Calls



- Quando uma aplicação efectua um system call a transição dá-se para o Host OS e não para o Guest
- Tem de haver um mecanismo de redirecção para o Guest OS

Gestão de Interrupções

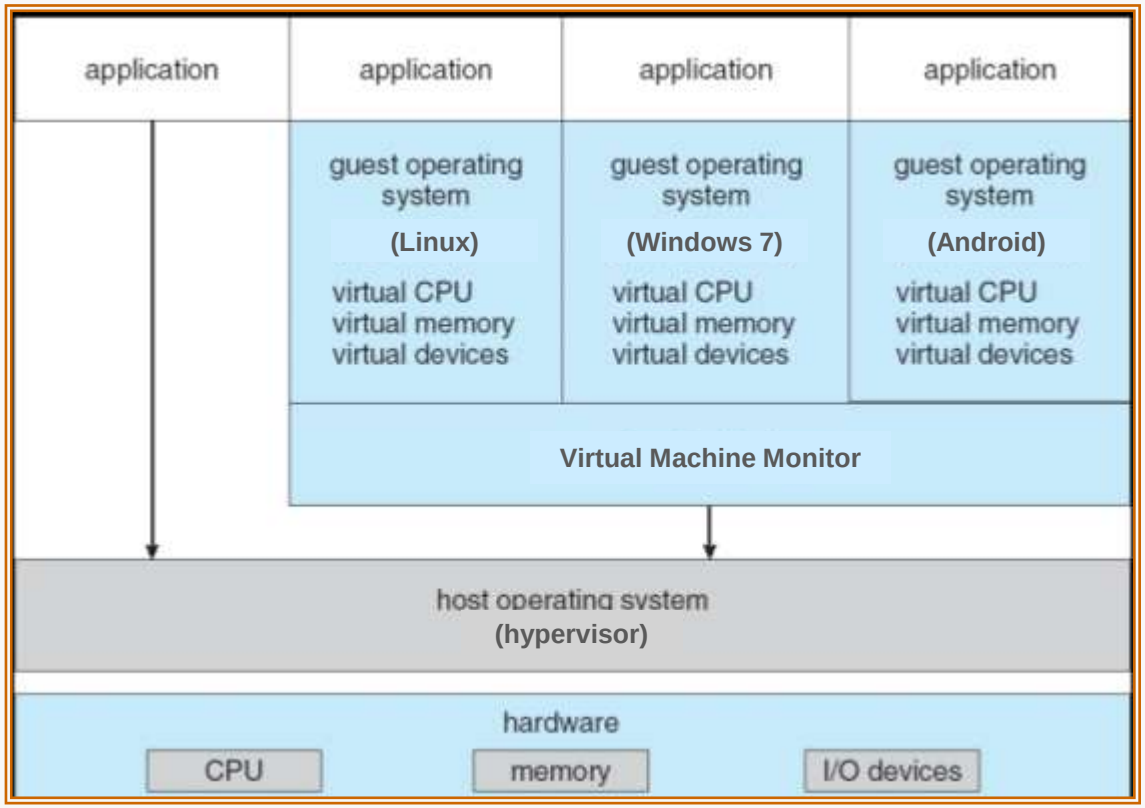


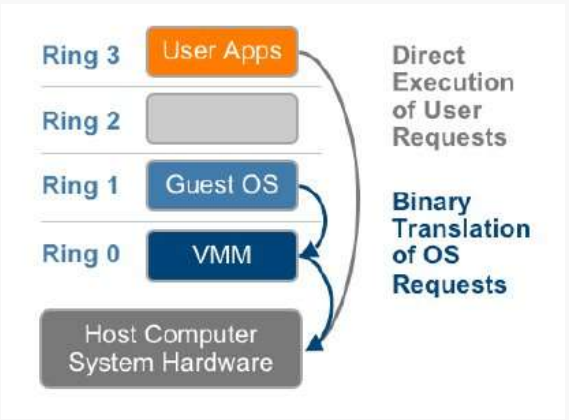
- Quando o hardware gera uma interrupção, é o Host OS que realiza o serviço e não para o Guest
- Tem de haver um mecanismo de redirecção para o Guest OS

Possíveis Implementações

- O conceito é difícil de implementar devido à dificuldade de simular o comportamento exacto da máquina real
  - O que acontece quando o sistema operativo hóspede acede a instruções privilegiadas só executáveis em modo supervisor?
  - Utilização de um VMM (Virtual Machine Monitor) para realizar a simulação
  - Várias técnicas de implementação são possíveis:
- Paravirtualização: consiste em modificar o Guest de forma a substituir todas as instruções privilegiadas por chamadas a funções específicas do VMM
  - Implica modificação do sistema Guest, o que a torna pouco utilizável, embora permita melhor desempenho
- Full Virtualization: consiste em interceptar as instruções privilegiadas do Guest e substituí-las dinamicamente por funções do VMM
  - Não necessita modificação do sistema Guest, mas tem pior desempenho

Exemplo: Arquitectura VMware



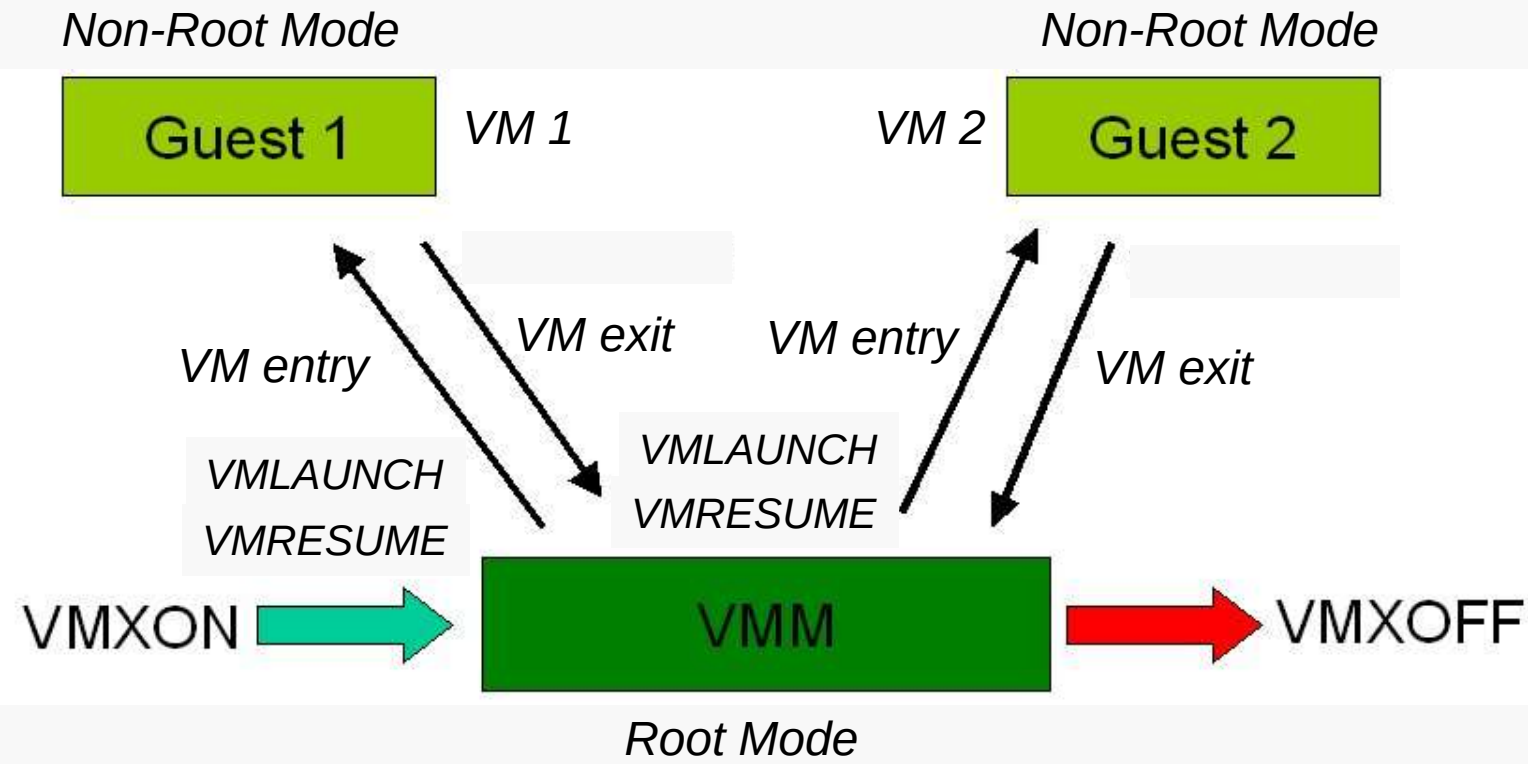


- *Full Virtualization*, implementada pela VMware, Virtual Box, Virtual PC, etc..
- Executa o sistema operativo Guest num nível de privilégio intermédio (p.ex.: 1), intercepta todas as instruções que não são permitidas nesse nível, e substituí-las por código equivalente no VMM

Extensões de Suporte às VM

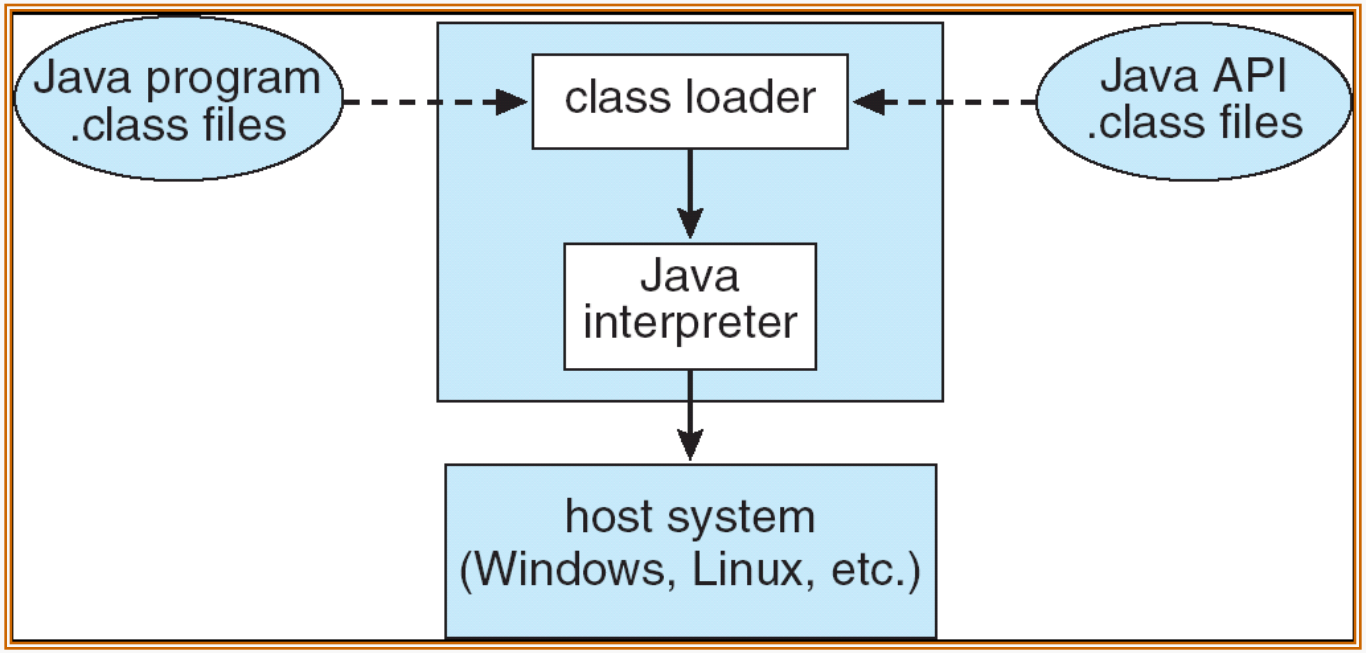
- Hoje em dia os principais processadores (Intel, AMD) fornecem extensões hardware para suporte da virtualização
- Arquitectura Intel Virtual: Technology Extensions (VT-x ou VT-i)
  - Dois novos modos de execução
    - Root para execução do VMM
    - Non-root para execução das VMs
  - Ambos suportam todos os níveis de privilégio do processador
  - A passagem de um contexto para outro (*VM Enter)* é provocada por certas instruções específicas (*VMENTER*, *VMRESUME*), ou pela execução de instruções não autorizadas (*VM Exit)*
  - Ver : www.intel.com/technology/itj/2006/v10i3/1-hardware/1-abstract.htm
- Arquitectura AMD: AMD-V ou SVM
  - Implementa funcionalidades semelhantes
- Intel® Virtualization Technology for Directed I/O (VT-d)
  - Permite a virtualização directa dos periféricos

Mudanças de Contexto VMM/VM



- A entrada na VM (*VM enter*) é activada pelas instruções *VMLAUNCH* (início) ou VMRESUME (continuação) a partir do VMM
- A saída da VM é (VM exit) é provocada pela execução de instruções não autorizadas no modo *Non-Root*, provocando a entrada no VMM
- Assim o código de cada OS não necessita de ser modificado e pode correr no seu nível de privilégio habitual

A Máquina Virtual Java



- Criação de uma abstracção de processador e Sistema Operativo
  - As aplicações são compiladas para uma linguagem máquina virtual
    - byte-code
  - Uma máquina virtual Java interpreta o código máquina como se fosse um processador virtual
- A máquina Java é implementada em cima de um OS tradicional

Referências Adicionais

- Como complemento deste slides os alunos poderão consultar e estudar os elementos seguintes:
  - Livro sobre programação avançada em C (incluindo System Calls)
    - http://www.cs.cf.ac.uk/Dave/C
  - Mapa Interactivo do Sistema Linux
    - http://www.makelinux.net/kernel_map
  - Livro “Windows Internals” de M. Russinovitch (Ed. 6 disponível on-line)
    - Windows-Internals Part1 (6thEdition)
  - Site e livro sobre o Mac OS X
    - http://osxbook.com

Fim da 2ª Parte