

Module I – Introduction to Operating Systems – Kernel

Prof. Ismael H F Santos

Considerações Gerais

- **Objetivo:** *Discutir os principais conceitos e os princípios básicos da Orientação a Objetos usando a linguagem C++.*
- **A quem se destina :** *Alunos e Profissionais que desejem aprofundar seus conhecimentos sobre Linguagem C++ e suas aplicações*

Bibliografia

- Sistemas Operacionais
 - Santos, F., H., Ismael; *Notas de Aula*, 2005
- *Introdução à Arquitetura de Sistemas Operacionais*
 - Francis B Machado e Luis Paulo Maia, LTC.
- *Sistemas Operacionais*
 - Toscani e outros, Serie UFRGS, no 11, Editora sagra-luzzatto
- *Fundamentos de Sistemas Operacionais*
 - Silberschatz, Abraham, Galvin, Peter, Gagne, G., LTC
- *Sistemas Operacionais Modernos*
 - Andrew S. Tanenbaum; Prentice Hall
- *Operating System Concepts: Internals and Design Principles*
 - William Stallings, Prentice Hall 1998 3ª Ed.



Agenda

- Monolithic x Micro-kernel
- Linux, Solaris and WindowsXP Kernels
- Assembly Language
- Format of Executable Programs
 - Linux
 - Win32

Monolithic
X
Micro-kernel

December 2008

Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

5

Introdução

■ Um Sistema Operacional é formado por um conjunto de rotinas que oferecem serviços aos usuários dos sistemas e suas aplicações. Este conjunto de rotinas é chamado de Núcleo de Sistema ou Kernel.

Algumas das funções do Kernel:

- *Tratamento de Interrupções*
- *Criação e Eliminação de Processos*
- *Sincronização e Comunicação entre Processos*
- *Escalonamento e controle dos Processos (Gerência de Processos)*
- *Gerência de Memória*
- *Operações de E/S (Gerência de Periféricos)*
- *Segurança do Sistema*

Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

6

Kernel Monolítico

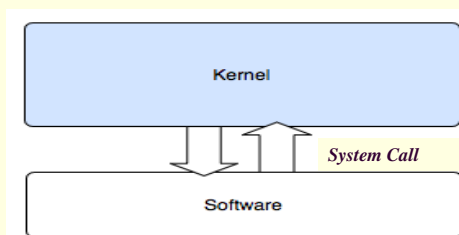
- O Kernel Monolítico foi a primeira arquitetura implementada
- Rotinas do sistema dentro do próprio Kernel
- Rotinas podem interagir livremente umas com as outras no espaço de endereçamento do Kernel

Alguns exemplos de Kernel Monolítico:

- BSD
- Linux
- MS-DOS e derivados, incluindo Windows 95, Windows 98, Windows ME
- Solaris

Kernel Monolítico - Conceitos

- Toda vez que um processo necessita de um serviço do sistema, ele o requisita através de uma *System Call*



Funções do Kernel:

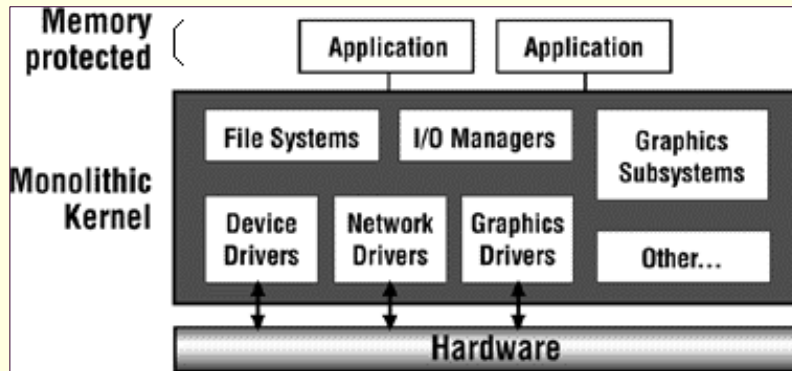
- Todas as funções do sistema
- Sistema de Arquivos
- Serviços de diretórios
- Gerência de processos
- Gerência de memória
- Mecanismos de tratamento de System Calls

■ Vantagens

- É um Kernel tradicional
- Rápida comunicação / acesso entre as componentes dentro do Kernel
- Boa performance em relação aos *Micro-Kernels*

Kernel Monolítico

Representação do Kernel Monolítico

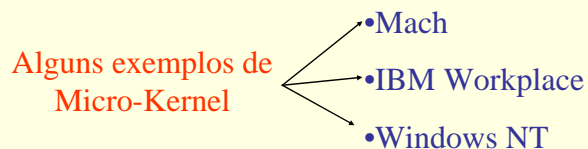


Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

9

Micro-Kernel

- O Micro-Kernel é uma evolução do Kernel Monolítico
- O Kernel do Micro-Kernel é bem pequeno se comparado ao do Monolítico
- As principais funcionalidades do sistema se encontram dentro do Kernel enquanto as secundárias se encontram fora do Micro-Kernel

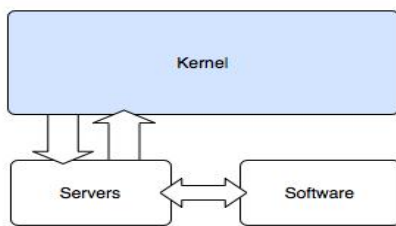


Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

10

Micro-Kernel - Conceitos

- Os processos requerem as *System Calls* através de bibliotecas do sistema que são carregadas em tempo de execução
- Os componentes do Kernel, excluindo as funções básicas, executam fora do espaço de endereçamento do Kernel



Funções no Kernel

- Apenas funções essenciais
- Gerência de memória
- Gerência de processos
- Tratamento de E/S

Vantagens

- Simplicidade
- Capacidade de extensão
- Capacidade de se adaptar facilmente a diversas plataformas (portabilidade)
- Personalizável

Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

11

Kernel Monolítico Versus Micro-Kernel

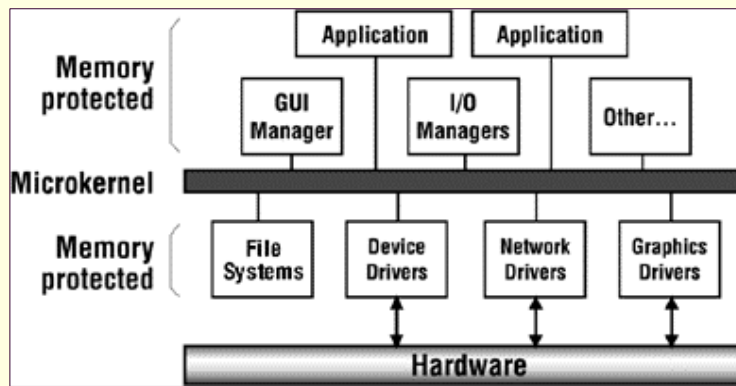
- O Micro-Kernel é tecnicamente mais avançado que o Monolítico
- A principal vantagem do Monolítico sobre o Micro-Kernel é a performance
- Juntando as vantagens do Micro-Kernel, este seria um SO mais avançado que o Monolítico. Porém, a perda de performance é uma desvantagem substancial. Isso gerou o conceito de Kernels Híbridos que aproveita a técnica avançada do Micro-Kernel e a boa performance do Monolítico

Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

12

Kernel Monolítico Versus Micro-Kernel

Representação do Micro-Kernel



Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

13

Linux



December 2008

Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

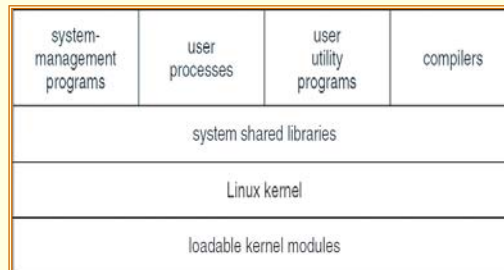
14

Linux Design Principles

- Linux is a multiuser, multitasking system with a full set of UNIX-compatible tools
- Its file system adheres to traditional UNIX semantics, and it fully implements the standard UNIX networking model
- Main design goals are speed, efficiency, and standardization
- Linux is designed to be compliant with the relevant POSIX documents; at least two Linux distributions have achieved official POSIX certification
- The Linux programming interface adheres to the SVR4 UNIX semantics, rather than to BSD behavior

Components of a Linux System

- The **kernel** is responsible for maintaining the important abstractions of the operating system
 - **Kernel code** executes in *kernel mode* with full access to all the physical resources of the computer
 - All kernel code and data structures are kept in the same single address space



Components of a Linux System (cont.)

- The **system libraries** define a standard set of functions through which applications interact with the kernel, and which implement much of the operating-system functionality that does not need the full privileges of kernel code
- The **system utilities** perform individual specialized management tasks
- **Kernel Modules**
 - Sections of kernel code that can be compiled, loaded, and unloaded independent of the rest of the kernel
 - A kernel module may typically implement a device driver, a file system, or a networking protocol

Kernel Modules

- The module interface allows third parties to write and distribute, on their own terms, device drivers or file systems that could not be distributed under the GPL
- Kernel modules allow a Linux system to be set up with a standard, minimal kernel, without any extra device drivers built in
- Three components to Linux module support:
 - module management
 - driver registration
 - conflict resolution

Module Management

- Supports loading modules into memory and letting them talk to the rest of the kernel
- Module loading is split into two separate sections:
 - Managing sections of module code in kernel memory
 - Handling symbols that modules are allowed to reference
- The module requestor manages loading requested, but currently unloaded, modules; it also regularly queries the kernel to see whether a dynamically loaded module is still in use, and will unload it when it is no longer actively needed

WindowsXP



Windows XP

- 32-bit preemptive multitasking operating system for Intel microprocessors
- Key goals for the system:
 - portability
 - security
 - POSIX compliance
 - multiprocessor support
 - extensibility
 - international support
 - compatibility with MS-DOS and MS-Windows applications.
- Uses a micro-kernel architecture
- Available in four versions, Professional, Server, Advanced Server, National Server

December 2008

Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

21

Design Principles

- Extensibility — layered architecture
 - Executive, which runs in protected mode, provides the basic system services
 - On top of the executive, several server subsystems operate in user mode
 - Modular structure allows additional environmental subsystems to be added without affecting the executive
- Portability — XP can be moved from on hardware architecture to another with relatively few changes
 - Written in C and C++
 - Processor-dependent code is isolated in a dynamic link library (DLL) called the “hardware abstraction layer” (HAL)

December 2008

Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

22

Design Principles (Cont.)

- Reliability —XP uses hardware protection for virtual memory, and software protection mechanisms for operating system resources
- Compatibility — applications that follow the IEEE 1003.1 (POSIX) standard can be compiled to run on XP without changing the source code
- Performance —XP subsystems can communicate with one another via high-performance message passing
 - Preemption of low priority threads enables the system to respond quickly to external events
 - Designed for symmetrical multiprocessing
- International support — supports different locales via the national language support (NLS) API

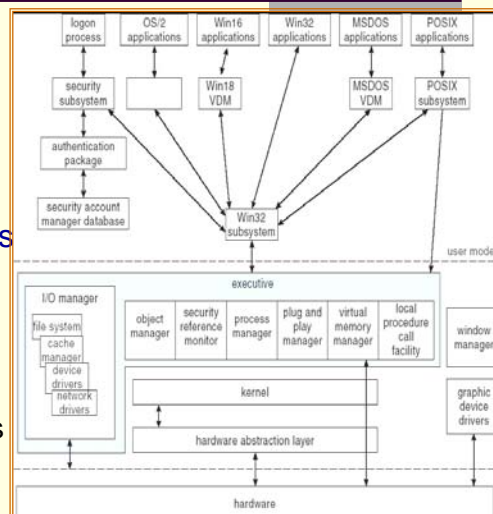
December 2008

Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

23

XP Architecture

- Layered system of modules
- Protected mode — HAL, kernel, executive
- User mode — collection of subsystems
 - Environmental subsystems emulate different operating systems
 - Protection subsystems provide security functions



December 2008

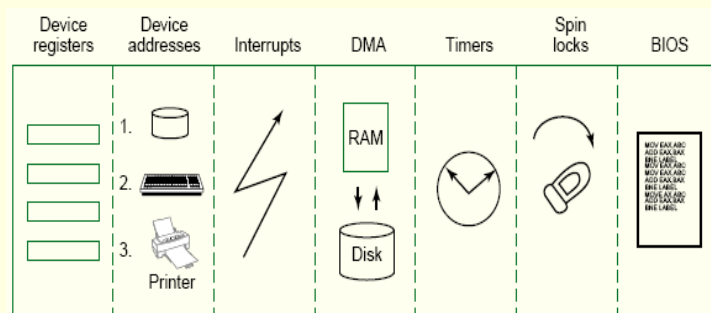
Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

24

XP Architecture

■ Hardware Abstraction Layer

- Biblioteca que engloba a parte do código do sistema dependente do hardware, como acesso à registradores e endereçamento de dispositivos, identificação de interrupções, DMA, temporização, sincronização em ambientes com múltiplos processadores e interface com a BIOS e CMOS.



December 2008

Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

25

System Components — Kernel

- Foundation for the executive and the subsystems
- Never paged out of memory; execution is never preempted
- Four main responsibilities:
 - thread scheduling
 - interrupt and exception handling
 - low-level processor synchronization
 - recovery after a power failure

December 2008

Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

26

System Components — Kernel

- Kernel is object-oriented, uses two sets of objects
 - *dispatcher objects* control dispatching and synchronization (events, mutants, mutexes, semaphores, threads and timers)
 - *control objects* (asynchronous procedure calls, interrupts, power notify, power status, process and profile objects)

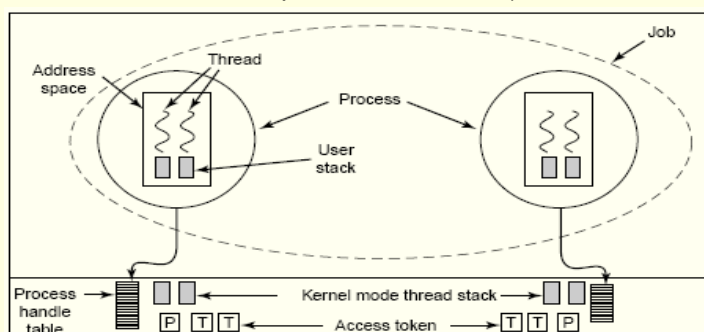
Kernel — Process and Threads

- The process has a virtual memory address space, information (such as a base priority), and an affinity for one or more processors
- Threads are the unit of execution scheduled by the kernel's dispatcher
- Each thread has its own state, including a priority, processor affinity, and accounting information
- A thread can be one of six states: *ready, standby, running, waiting, transition, and terminated*

Kernel — Process and Threads

■ Processos

- São objetos criados e eliminados pelo gerente de objetos. Cada processo possui, dentre outros recursos do sistema, um **espaço de endereçamento virtual**, uma tabela de objetos, um token de acesso e pelo menos um thread (threads adicionais podem ser criados e eliminados quando necessários).



December 2008

Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

29

Kernel — Process and Threads

■ Threads

- São implementados como objetos, sendo criados e eliminados pelo gerenciador de objetos. Todos os threads de um processo compartilham o mesmo **espaço de endereçamento virtual** e todos os recursos do processo, incluindo o token de acesso, prioridade base, tabela de objetos e seus handles.
- São implementadas em modos usuário e kernel.
- Threads em **modo kernel**, apresentam problemas de desempenho devido a necessidade de troca de modo de acesso usuário-kernel-usuário.
- Threads em **modo usuário**, denominados fibers, eliminam as trocas de contexto e de modo de acesso e, conseqüentemente, oferecem melhor desempenho.

December 2008

Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

30

Gerência do Processador

- Tipos de política de escalonamento:
 - escalonamento circular com prioridades
 - escalonamento por prioridades
- O escalonamento pode ocorrer quando:
 - a thread entra no estado de pronto ou de espera
 - a thread termina
 - a aplicação altera a prioridade da thread
- As prioridades são associadas aos processos e threads no estado de pronto.
- O Windows 2000 implementa 32 níveis de prioridades, divididos em duas faixas:
 - escalonamento de prioridade variável (1-15)
 - escalonamento de tempo real (16-31)

December 2008

Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

31

Executive — Object Manager

- XP uses objects for all its services and entities; the object manger supervises the use of all the objects
 - Generates an object *handle*
 - Checks security
 - Keeps track of which processes are using each object
- Objects are manipulated by a standard set of methods, namely `create`, `open`, `close`, `delete`, `query name`, `parse` and `security`

December 2008

Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

32

Executive — Naming Objects

- The XP executive allows any object to be given a name, which may be either permanent or temporary
- Object names are structured like file path names in MS-DOS and UNIX
- XP implements a *symbolic link object*, which is similar to *symbolic links* in UNIX that allow multiple nicknames or aliases to refer to the same file
- A process gets an object handle by creating an object by opening an existing one, by receiving a duplicated handle from another process, or by inheriting a handle from a parent process
- Each object is protected by an access control list

December 2008

Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

33

Executive — Virtual Memory Manager

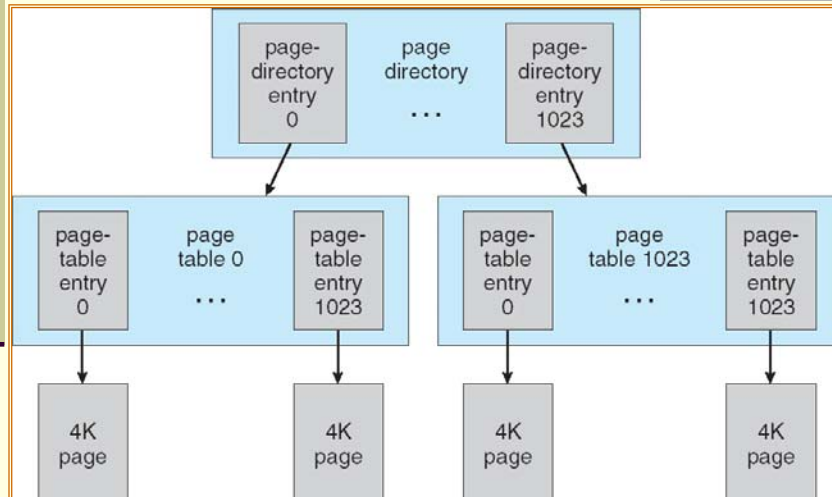
- The design of the VM manager assumes that the underlying hardware supports virtual to physical mapping a paging mechanism, transparent cache coherence on multiprocessor systems, and virtual addressing aliasing
- The VM manager in XP uses a page-based management scheme with a page size of 4 KB
- The XP VM manager uses a two step process to allocate memory
 - The first step reserves a portion of the process's address space
 - The second step commits the allocation by assigning space in the 2000 paging file

December 2008

Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

34

Virtual-Memory Layout



December 2008

Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

35

Virtual Memory Manager (Cont.)

- The virtual address translation in XP uses several data structures
 - Each process has a *page directory* that contains 1024 *page directory entries* of size 4 bytes
 - Each page directory entry points to a *page table* which contains 1024 *page table entries* (PTEs) of size 4 bytes
 - Each PTE points to a 4 KB *page frame* in physical memory
- A **10-bit integer** can represent all the values from 0 to 1023, therefore, can select any entry in the page directory, or in a page table
- This property is used when translating a virtual address pointer to a byte address in physical memory
- A page can be in one of six states: **valid, zeroed, free standby, modified and bad**

December 2008

Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

36

Solaris

December 2008 Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES 37

Kernel Solaris

- **System Call Interface** – API's do sistema permitem que os processos do usuário alcancem facilidades da kernel.
- **Execução e agendamento de processos** - a gerência processos fornece recursos para a criação, execução, gerência e a termino de processos.
- **Gerência da memória** - o sistema de memória virtual controla o mapeamento de memória física aos processos de usuário e do kernel. Os componentes específicos de hardware são alocados na camada da tradução de endereço da hardware (HAT)
- **Sistema de arquivos** - Solaris contém uma estrutura virtual de filesystems por que múltiplos filesystems podem ser configurados no kernel de Solaris ao mesmo tempo.

December 2008 Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES 38

Kernel Solaris

- O kernel de Solaris é **monolítico**, com subsistemas adicionais do kernel e serviços adicionados dinamicamente (**loadable modules**). Esta execução é facilitada pela infraestrutura runtime do linker e do kernel modular, que permite que os módulos da kernel sejam adicionados ao sistema durante a execução do mesmo sob demanda.

System Calls Scheduler Memory Mgmt Proc Mgmt VFS Framework Kernel Locking Clock & Timers Interrupt Mgmt Boot & Startup Trap Mgmt CPU Mgmt	Scheduler Classes	TS – Time Share RT – Real Time IA – Interactive Class SRM – Resource Manager Class	
	File Systems	UFS – UNIX File System NFS – Network File System PROCFS – Process File System Etc...	
	Loadable System Calls	shmsys – System V Shared Memory semsys – Semaphores msgsys – Messages Other loadable system calls ...	
	Executable Formats	ELF – SVR4 Binary Format COFF – BSD Binary Format	
	Streams Modules	pipemod – Streams Pipes ldterm – Terminal Line Disciplines Other loadable streams modules ...	
	Misc Modules	NFSSRV – NFS Server IPC – Interprocess Communication Other loadable kernel code ...	
	Device and Bus Drivers	SBus – SBus Bus Controller PCI – PCI Bus Controller sd – SCSI I/O Devices Many other devices ...	

Assembly Language



Assembly language

- Lowest level of programming (besides microcode)
- Direct processor register access utilizing architecture defined instructions
- Output of most compilers
- How is it used
 - Directly using an assembler
 - `NASM`, `ml`, `as`
 - Output by a high level compiler
 - `GCC`, `cl`

What does it look like

- Depends on the instruction set
 - IA32
 - `mov eax, 0x1`
 - PA-RISC
 - `copy %r14,%r25`
 - ARM
 - `LDR r0,[r8]`
- Instruction Sets
 - The mnemonics for the opcodes handled by the processor
 - Minimal set of “commands” that achieve a programming goal

Different Instruction Set Architectures

- **RISC - Reduced Instruction Set Computing**
 - Fixed length 32 bit instructions
 - 32 general purpose registers
 - Vendors
 - IBM (PowerPC)
 - HP (PA-RISC)
 - Apple (PowerPC)
- **CISC - Complex Instruction Set Computing**
 - Multibyte instructions
 - Multiple synonymous opcodes
 - 16 registers
 - Vendors
 - Intel (IA-32)
 - DEC [PDP-11]
 - Motorola (m68K)

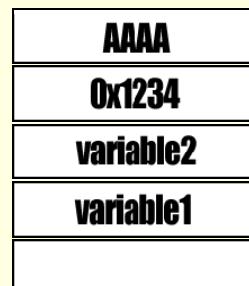
December 2008

Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

43

Registers and the Stack

- **Purpose**
 - Registers are used to store temporary data
 - Pointers
 - Computations
 - The stack is used to manage data
 - Variables
 - Data
- **Stack layout**
 - Stack is dynamic but builds as it goes
 - Addresses start at a higher address and builds to lower addresses
 - The stack is generally allocated in 4 byte chunks



December 2008

Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

44

Register sizes

- Register sizes depend on the supported architecture
 - 32 bit
 - 64 bit
- IA32
 - 16 registers 32 bits (4 bytes) each
- RISC
 - 32 general purpose registers 64 bits [8 bytes] each

IA32 Registers

- **EBP – Stack frame base pointer**
 - Points to the start of the functions stack frame
- **ESP – Stack source pointer**
 - Points to the current (top) location on the stack
- **EIP – Instruction pointer**
 - Points to the next executable instruction
- **General Purpose registers**
 - Used in general computation and control flow
 - EAX – Accumulator register
 - EBX – General data register
 - ECX – Counter register
 - EDX – General data register
 - ESI – Source index register
 - EDI – Destination index register

IA32 Registers Cont...

- **Segment registers**
 - Used to segment memory and compute addresses
 - CS – Code segment register
 - SS - Stack segment register
 - DS - Data segment register
 - ES - Extra (More data) segment register
 - FS - Third data segment register
 - GS – Fourth data segment register
- **EFLAGS**
 - CF – Carry Flag
 - SF – Signed Flag
 - ZF – Zero Flag

December 2008

Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

47

Overview of IA-32 Instruction Set

- **mov** – Moves source to destination
- **lea** – Loads effective address
- **jmp** – Jump
 - **jne** – Jump if not equal
 - **jg** – Jump if greater than
- **call** – Unconditional function call
- **ret** – Returns from a function to the caller
- **add** – Adds two values
- **sub** – subtracts two values
- **xor** – XORs two values
- **cmp** – Compares two registers

December 2008

Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

48

Calling conventions

- Calling conventions define how the callers data is arranged on the stack
- cdecl
 - Most common calling convention
 - Dynamic parameters
 - Caller unwinds stack
 - `pop ebp` and later `ret`
- fastcall
 - Higher performance
 - First two parameters are passed over registers
- stdcall
 - Common in Windows
 - Parameters are received in reverse order
 - Function unwinds stack
 - `ret 0x16`

December 2008

Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

49

Example 1

```
for(int i=0;i<5;i++)
{
    printf("Hello");
}
```



```
0000 mov ecx, 5
0003 push aHello
0009 call printf
000E loop 00000003h
0014 ...
```

```
if(x == 256)
{
    printf("Yes");
}
else
{
    printf("No");
}
```



```
0000 cmp ecx, 100h
0003 jnz 001Bh
0009 push aYes
000F call printf
0015 jmp 0027h
001B push aNo
0021 call printf
0027 ...
```

December 2008

Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

50

Example 2

```
PUSH EBP ; Pushes the contents of EBP onto the stack
MOV EBP, ESP ; Moves the address of ESP to EBP
CMP DWORD PTR [EBP+C], 111 ; Subtract what is at EBP+12 with 111
JNZ 00401054 ; If previous compare is not zero jump to
00401054
MOV EAX, DWORD PTR [EBP+10] ; Move what is at EBP+16 to EAX
CMP AX, 64 ; Subtract what we moved to EAX with 64
JNZ 00401068 ; If the comparison does not equal 0
jump to address 00401068
POP EBP ; Store the current value on the stack in EBP
RET ; Return to the caller
```

Format of Executable Programs



What are file formats?

- Files that adhere to a specific format often being executable by an operating system
- Executable files are created from source code and libraries by a compiler
- Data files can be created by anything from a text editor to an mp3 encoder

COFF

- The **Common Object File Format (COFF)** is a specification of a format for executable, object code, and shared library computer files used on Unix systems.
 - Introduced in Unix System V, and formed the basis for extended specifications such as XCOFF and ECOFF, before being largely replaced by ELF, introduced with SVR4. COFF and its variants continue to be used on some Unix-like systems and on Microsoft Windows.
- COFF was introduced in AT&T's UNIX System V for non-VAX 32-bit platforms such as the 3B20.

COFF

- Improvements over the existing AT&T `a.out` format included built-in support for symbolic debugging information, shared libraries, and an extension mechanism.
- Although `COFF` was an improvement over `a.out`, the design was too limited: there was a **limit on the maximum number of sections**, a **limit on the length of section names**, and the symbolic debugging information was **incapable of supporting newer languages** such as `C++`.
- While extended versions of `COFF` continue to be used for some Unix-like platforms, primarily in embedded systems, perhaps the most widespread use of the `COFF` format today is in **Microsoft's Portable Executable (PE) format**.
- Developed for Windows NT, the **PE format** (sometimes written as **PE/COFF**) uses a `COFF` header for object files, and as a component of the PE header for executable files.

Symbolic Debugging

- The `COFF` symbolic debugging information consists of symbolic (string) names for program functions and variables, and line number information, used for setting breakpoints and tracing execution.
- Symbolic names are stored in the `COFF` symbol table. Each symbol table entry includes a name, storage class, type, value and section number. Short names (8 characters or fewer) are stored directly in the symbol table; longer names are stored as an offset into the string table at the end of the `COFF` object.

Relative Virtual Address

- When a COFF file is generated, it is not usually known where in memory it will be loaded. The virtual address where the first byte of the file will be loaded is called **image base address**. The rest of the file is not necessarily loaded in a contiguous block, but in different sections.
- A **Relative Virtual address** is the virtual address of an object from the file once it is loaded into memory, minus the base address of the file image. If the file were to be mapped literally from disk to memory, the RVA would be the same as that of the offset into the file, but this is actually quite unusual
- Note that the RVA term is only used with objects in the image file. Once loaded into memory, the image base address is added, and ordinary VAs are used.

December 2008

Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

57

Executable Contents

- Machine code
 - Instructions the program will run
 - Memory locations
 - code addresses
 - function addresses
- Program data
 - Static variables
 - Strings
- Loader data
 - Imports
 - Exports
- Sections
 - Allows the loader to find various information
 - Not finite, executables can have user defined sections

December 2008

Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

58

Executing Programs
Linux

December 2008 Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES 59

Executing and Loading User Programs

- Initially, binary-file pages are mapped into virtual memory
 - Only when a program tries to access a given page will a page fault result in that page being loaded into physical memory
- An ELF-format binary file consists of a header followed by several page-aligned sections
 - The ELF loader works by reading the header and mapping the sections of the file into separate regions of virtual memory
- Linux maintains a table of functions for loading programs; it gives each function the opportunity to try loading the given file when an exec system call is made

December 2008 Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES 60

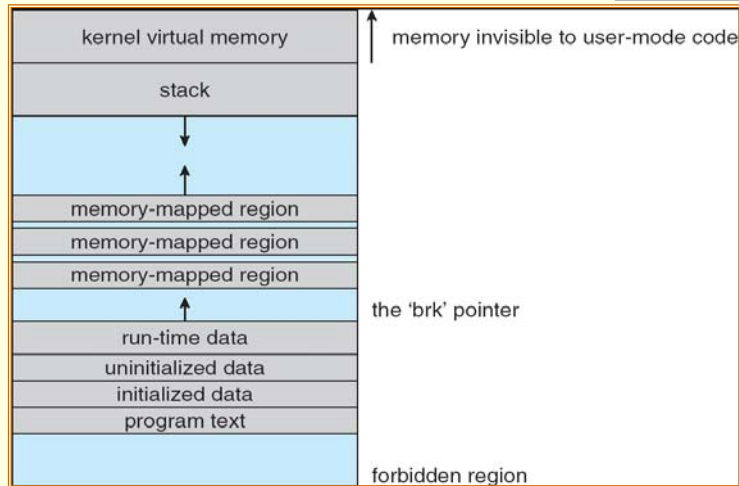
Executing and Loading User Programs

- The Executable and Linking Format (ELF, formerly called Extensible Linking Format) is a common standard file format for executables, object code, shared libraries, and core dumps. First published in the System V Application Binary Interface specification,[1] and later in the Tool Interface Standard,[2] it was quickly accepted among different vendors of Unix systems. In 1999 it was chosen as the standard binary file format for Unix and Unix-like systems on x86 by the 86open project.
- ELF is very flexible and extensible, and it is not bound to any particular processor or architecture. This has allowed it to be adopted by many different operating systems on many different platforms.

Executing and Loading User Programs

- The registration of multiple loader routines allows Linux to support both the ELF and **a.out** binary formats
- Initially, binary-file pages are mapped into virtual memory
 - Only when a program tries to access a given page will a page fault result in that page being loaded into physical memory
- An ELF-format binary file consists of a header followed by several page-aligned sections
 - The ELF loader works by reading the header and mapping the sections of the file into separate regions of virtual memory

Memory Layout for ELF Programs



December 2008

Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

63

Executable Formats

- ELF – Executable and Linker Format
 - History
 - Originally published by UNIX system laboratories as a dynamic, linkable format to be used in various UNIX platforms
 - What uses ELF
 - Linux
 - Solaris
 - Most modern BSD based unix's
 - Dissection
 - Header
 - Sections

December 2008

Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

64

ELF Header

- The header contains various information the operating system loading needs

e_ident – Contains various identification fields including Endianess, ELF version, Operating System

e_type – Identifies the object file type including relocatable

December 2008 Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

65

ELF Sections

- Each section of an ELF executable contain various information needed to execute

.bss - This section holds uninitialized data that contributes to the program's memory image. By definition, the system initializes the data with zeros when the program begins to run.

.comment - This section holds version control information.

.ctors - This section holds initialized pointers to the C++ constructor functions.

.data - This section holds initialized data that contribute to the program's memory image.

.data1 - This section holds initialized data that contribute to the program's memory image.

.debug - This section holds information for symbolic debugging. The contents are unspecified.

.dtors - This section holds initialized pointers to the C++ destructor functions.

.dynamic - This section holds dynamic linking information.

December 2008

Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

66

ELF Sections Cont...

- .dynstr** - This section holds strings needed for dynamic linking, most commonly the strings that represent the names associated with symbol table entries.
- .dysym** - This section holds the dynamic linking symbol table.
- .fini** - This section holds executable instructions that contribute to the process termination code. When a program exits normally the system arranges to execute the code in this section.
- .got** - This section holds the global offset table.
- .hash** - This section holds a symbol hash table.
- .init** - This section holds executable instructions that contribute to the process initialization code. When a program starts to run the system arranges to execute the code in this section before calling the main program entry point.
- .interp** - This section holds the pathname of a program interpreter. If the file has a loadable segment that includes the section, the section's attributes will include the SHF_ALLOC bit. Otherwise, that bit will be off.

December 2008

Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

67

ELF Sections Cont...

- .line** - This section holds line number information for symbolic debugging, which describes the correspondence between the program source and the machine code. The contents are unspecified.
- .note** - This section holds information in the "Note Section" format described below.
- .plt** - This section holds the procedure linkage table.
- .relNAME** - This section holds relocation information. By convention, "NAME" is supplied by the section to which the relocations apply. Thus a relocation section for .text normally would have the name .rel.text
- .rodata** - This section holds read-only data that typically contributes to a non-writable segment in the process image.
- .rodata1** - This section holds read-only data that typically contributes to a non-writable segment in the process image.
- .shstrtab** - This section holds section names.

December 2008

Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

68

ELF Sections Cont...

- .strtab** - This section holds strings, most commonly the strings that represent the names associated with symbol table entries.
- .symtab** - This section holds a symbol table. If the file has a loadable segment that includes the symbol table, the section's attributes will include the SHF_ALLOC bit. Otherwise the bit will be off.
- .text** - This section holds the ``text'' or executable instructions, of a program.

Executable Formats Cont...

- PE – Portable Executable
 - History
 - Microsoft migrated to the PE format with the introduction of the Windows NT 3.1 operating system. It is based of a modified form of the UNIX COFF format
 - What uses PE
 - Windows NT, 2000, Windows XP, Windows 2003, Windows CE
 - Dissection
 - DOS Stub
 - The DOS stub contains a message that the executable will not run in DOS mode
 - Optional Header (Not optional]
 - RVA
 - Relative virtual addressing
 - Sections

Optional Header

- The optional header in a PE executable contains various information regarding the executable contents needed for the OS loader

SizeOfCode – Size of the code (text) section, or the sum of all code sections if there

December 2008

Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

71

Optional Header (cont.)

- The optional header in a PE executable contains various information regarding the executable contents needed for the OS loader

SizeOfImage - Size, in bytes, of image, including all headers; must be a multiple of Section Alignment

SizeOfHeaders - Combined size of MS-DOS stub, PE Header, and section headers rounded up to a multiple of FileAlignment.

NumberOfRvaAndSizes - Number of data-dictionary entries in the remainder of the Optional Header. Each describes a location and size.

December 2008

Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

72

Sections

- The sections in a PE file contain various pieces of the executable needed to run including various RVA's and offsets

.text – Contains all executable code

.idata – Contains imported data such as dll addresses

.edata – Contains any exported data

.data – Contains initialized data like global variables and string literals

.bss – Contains un-initialized data

.rsrc – Contains all module resources

.reloc – Contains relocation data for the OS loader

Data Formats

- Different than executable formats
 - Doesn't usually contain machine code
 - Has structure but not always defined sections
- A reverser often needs to reverse how a file format functions
 - Proprietary formats are not always published
 - Reversing allows compatibility (i.e. Microsoft doc)
- Data rights management
 - Often the only way to get what you pay for is to take action

Executing Programs Windows



PE File

- Portable Executable File Format
 - PE/COFF headers
 - Data directories
 - Sections
- Demos: Dumpbin, .Net Explorer

Basic PE File Structure

- PE Header
- PE Sections
- Overlay (optional)

Imported functions needed for the application

Additional data used by the application

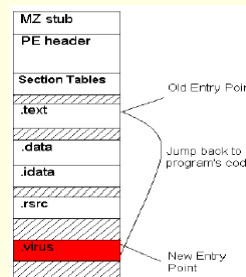
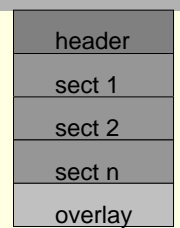


PE data needed in order to execute the application

Code and data of the application

PE Structure

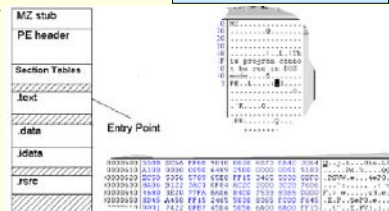
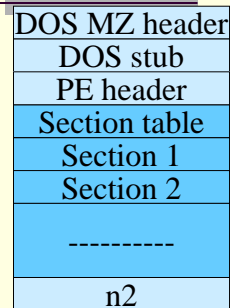
- PE header:
 - Documents "explicit" image structure
 - Vs. "implicit" structure
- PE section
 - Primary image content
 - Code, data, etc.
 - Described in header's section table
- Overlay: non-loadable data, appended to PE image
 - Certificates
 - Debug info
 - **Malware-specific payload**



Simple Win32 virus

PE file format

- **DOS MZ header**
 - So DOS can recognize program
- **DOS stub**
 - Built-in DOS executable to display "This program cannot run in MS-DOS mode"
- **PE header**
 - PE loader uses DOS MZ header to find starting offset of the PE header (skipping stub)
- **Sections**
 - Blocks of code/data organized on a logical basis



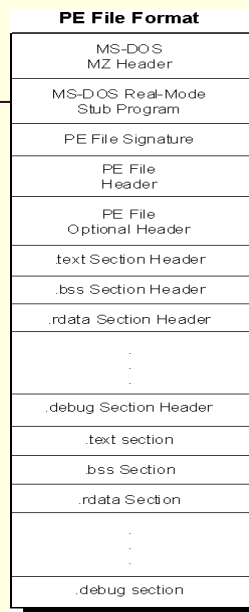
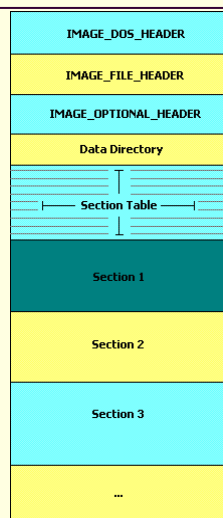
December 2008

Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

79

PE file format detailed

- **PE File Format**
 - Based on COFF Format
 - Magic Numbers, Headers, Tables, Directories, Sections
- **Simply Overlay Data with C Structures**
- **Load File as OS Loader Would Identify Entry Points (Default & Exported)**



December 2008

Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

80

Alignment

- Alignment applies to section mapping
- PE header specifies two sectional alignment values
 - *File alignment* specifies file mapped alignment
 - *Virtual alignment* specifies virtual mapped alignment

Image Mapping

- Windows loader performs “map and load” operation:
 - Map:
 1. Size the view
 2. Create view in process VA space
 3. Allocate storage
 - Load image section by section
- Our parser mimics this behavior
 - “Source representation”
 - Frequently file mapped (linker output)
 - However we may be given memory mapped image with no corresponding file image
 - “Target representation”
 - Typically virtual mapped

Mapping Translation

- Need to handle both file- and virtual-mapped images
- `clmageStream` class
 - Accepts any source representation
 - Translates to requested target representation
 - Manages all stream-related details

Section Size

- Fundamental concept when dealing with sections due to variable section alignment
 - Applies to header and sections
- 3 unique size concepts:
 - Raw size: unpadding data size
 - File size: `RoundUp(raw_size, file_align)`
 - “File cave”; persistent
 - Virtual size: `RoundUp(file_size, virtual_align)`
 - “Virtual cave”; transient
 - Be precise!
 - Always explicitly state the size type in source code

Section Size (con't)

- Interesting (and annoying) that raw section size is unavailable
 - Important if you want size of *REAL* content!
 - E.g., when parsing structures in the header
 - ... Or instructions (atoms) in a code section
- In practice, file aligned size is often treated as synonymous with raw size
- Demo:
 - Dump basic white file; identify raw, file, virtual sizes

CLR Header

- Contains CLR specific information
 - “Required runtime” version
 - Metadata location
 - Managed resources location
 - Strong name signature location
- Demo: .Net Explorer

Managed Execution CLR

December 2008

Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

87

Managed Execution

- What is the Common Language Runtime?
 - Responsible for managing the execution of code that is targeted towards the .NET platform
 - Code that requires the CLR at run-time in order to execute is referred to as *managed code*
- What is the role of the CLR?
 - The CLR exists to provide managed services to code and types
 - *Services include the class loader, IL, IL-native compilation, profiling, debugging, exception handling, serialization, security, memory management and garbage collection*
 - *Managed and unmanaged code may coexist*
 - *Low management costs until services are used*

December 2008

Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

88

Managed Execution

■ CLR Architecture

- Managed code relies on two DLLs
 - **MSCOREE.DLL**
 - The CLR, or the Runtime
 - Is an unmanaged DLL that loads managed code
 - **MSCORLIB.DLL**
 - The Base Class Library, or the runtime library
 - Is a managed DLL

December 2008

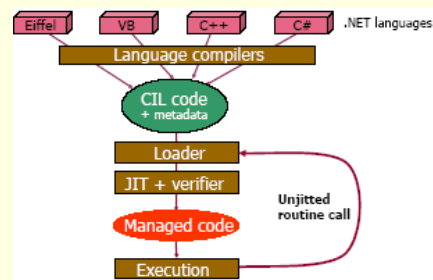
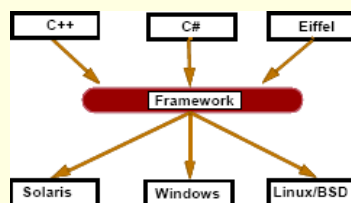
Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

89

Managed Execution

■ Where does Managed Code come from?

- Create source code using any programming language that supports the CLR
- Compile the source code using the corresponding compiler resulting in a *managed module*



December 2008

Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

90

Managed Execution

- What is a Managed Module?
 - A managed module is a standard Windows PE file that requires the CLR to execute
 - They can be loaded using the LoadLibrary system call
 - What are the constituent parts of a Managed Module?
 - PE Header
 - CLR Header
 - Metadata
 - Intermediate (IL) code

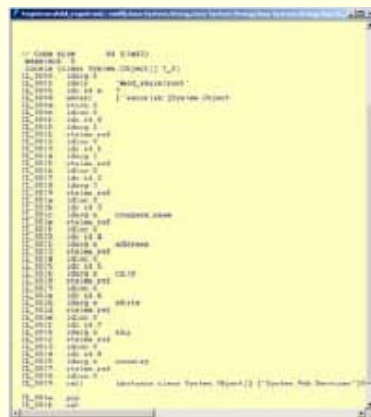
December 2008

Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

91

CLI Code: How does it look like ?

- An Intermediate Language !
- Basic instructions:
 - Loading parameters
 - Calling routines
 - Loading strings
 - Create objects
 - Push on the stack
 - Pop from the stack
 - Test & Jump



The screenshot shows a debugger window with a list of instructions. The instructions are in a compact, assembly-like format, including operations like 'ld', 'call', 'push', 'pop', and 'jmp'. The window title is 'C:\Program Files\Microsoft Visual Studio\2005\Tools\Microsoft SDK for Windows\bin\x86\Debug\CLI.exe'. The instructions are listed with their addresses and hex values.

December 2008

Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

92

Managed Execution

- The Microsoft implementation of COFF (Common Object File Format)
- COFF
 - 32-bit format for executable and object files that is portable across platforms
- Derived from Unix specification
 - Additional headers for compatibility with MS-DOS and 16-bit Windows

December 2008

Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

93

Managed Execution

- PE Header
 - Indicates the type of file: GUI, CUI, DLL plus timestamp
 - For modules containing only IL code, most information is ignored, for native CPU code contains information about that code
- CLR Header
 - Version of the CLR required
 - Flags
 - MethodDef metadata token of the module's entry point method (aka Main method)
 - Location/size of metadata, resources, strong name, etc..
- Metadata
 - Basically a set of tables
 - 2 main types of tables:
 - Tables describing defined types and members
 - Tables describing referenced types and members
- Intermediate Language (IL) Code
 - Code that the compiler produced as it compiled the source code

December 2008

Ismael H. F. Santos - ismaelh@petrobras.com.br - Petrobras/CENPES

94