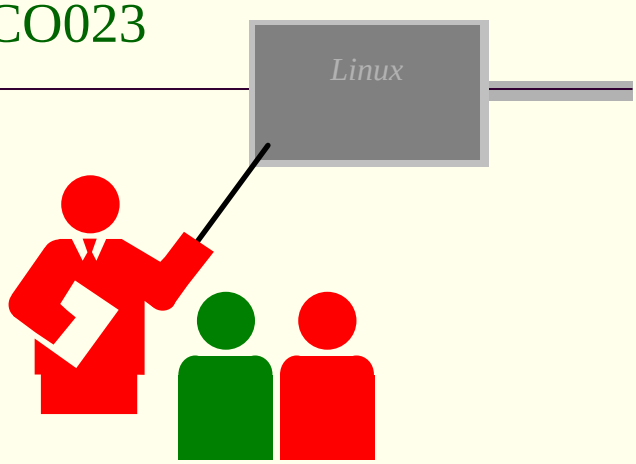
Modulo VII
Estudos de Caso: Linux e Windows

Prof. Ismael H F Santos

Ementa

- Linux
  - Linux History
  - Design Principles
  - Kernel Modules
  - Process Management
  - Scheduling
  - Memory Management
  - File Systems
  - Input and Output
  - Interprocess Communication
  - Network Structure
  - Security
- Windows XP
  - History
  - Design Principles
  - System Components
  - Environmental Subsystems
  - File system
  - Networking
  - Programmer Interface

SOP – CO023



Objectives

- To explore the history of the UNIX operating system from which Linux is derived and the principles which Linux is designed upon
- To examine the Linux process model and illustrate how Linux schedules processes and provides interprocess communication
- To look at memory management in Linux
- To explore how Linux implements file systems and manages I/O devices

History

- Linux is a modern, free operating system based on UNIX standards
- First developed as a small but self-contained kernel in 1991 by Linus Torvalds, with the major design goal of UNIX compatibility
- Its history has been one of collaboration by many users from all around the world, corresponding almost exclusively over the Internet

History (cont.)

- It has been designed to run efficiently and reliably on common PC hardware, but also runs on a variety of other platforms
- The core Linux operating system kernel is entirely original, but it can run much existing free UNIX software, resulting in an entire UNIX-compatible operating system free from proprietary code
- Many, varying **Linux Distributions** including the kernel, applications, and management tools

The Linux Kernel

- Version 0.01 (May 1991) had no networking, ran only on 80386-compatible Intel processors and on PC hardware, had extremely limited device-drive support, and supported only the Minix file system
- Linux 1.0 (March 1994) included these new features:
  - Support for UNIX's standard TCP/IP networking protocols
  - BSD-compatible socket interface for networking programming
  - Device-driver support for running IP over an Ethernet
  - Enhanced file system
  - Support for a range of SCSI controllers for high-performance disk access
  - Extra hardware support
- Version 1.2(March/95) was the final PC-only Linux kernel

Linux 2.0

- Released in June 1996, 2.0 added two major new capabilities:
  - Support for multiple architectures, including a fully 64-bit native Alpha port
  - Support for multiprocessor architectures
- Other new features included:
  - Improved memory-management code
  - Improved TCP/IP performance
  - Support for internal kernel threads, for handling dependencies between loadable modules, and for automatic loading of modules on demand
  - Standardized configuration interface

Linux 2.0 (cont.)

- Released in June 1996, 2.0 added two major new capabilities:
  - Support for multiple architectures, including a fully 64-bit native Alpha port
  - Support for multiprocessor architectures
- Other new features included:
  - Improved memory-management code
  - Improved TCP/IP performance
  - Support for internal kernel threads, for handling dependencies between loadable modules, and for automatic loading of modules on demand
  - Standardized configuration interface

Linux 2.0 (cont.)

- Available for Motorola 68000-series processors, Sun Sparc systems, and for PC and PowerMac systems
- 2.4 and 2.6 increased SMP support, added journaling file system, preemptive kernel, 64-bit memory support

The Linux System

- Linux uses many tools developed as part of Berkeley's BSD operating system, MIT's X Window System, and the Free Software Foundation's GNU project
- The min system libraries were started by the GNU project, with improvements provided by the Linux community
- Linux networking-administration tools were derived from 4.3BSD code; recent BSD derivatives such as Free BSD have borrowed code from Linux in return
- The Linux system is maintained by a loose network of developers collaborating over the Internet, with a small number of public ftp sites acting as de facto standard repositories

Linux Distributions

- Standard, precompiled sets of packages, or *distributions*, include the basic Linux system, system installation and management utilities, and ready-to-install packages of common UNIX tools
- The first distributions managed these packages by simply providing a means of unpacking all the files into the appropriate places; modern distributions include advanced package management
- Early distributions included SLS and Slackware
  - *Red Hat* and *Debian* are popular distributions from commercial and noncommercial sources, respectively

Linux Distributions (cont.)

- Early distributions included SLS and Slackware
  - *Red Hat* and *Debian* are popular distributions from commercial and noncommercial sources, respectively
- The RPM Package file format permits compatibility among the various Linux distributions

Linux Licensing

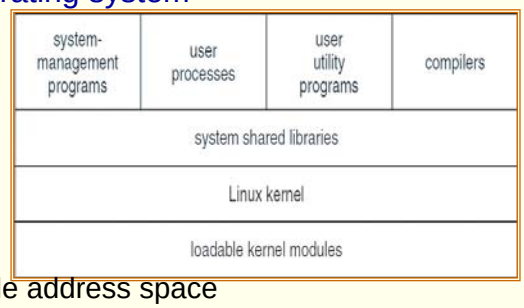
- The Linux kernel is distributed under the GNU General Public License (GPL), the terms of which are set out by the Free Software Foundation
- Anyone using Linux, or creating their own derivative of Linux, may not make the derived product proprietary; software released under the GPL may not be redistributed as a binary-only product

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

- Like most UNIX implementations, Linux is composed of three main bodies of code; the most important distinction between the kernel and all other components
- 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
- 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 (cont.)

- 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

Driver Registration

- Allows modules to tell the rest of the kernel that a new driver has become available
- The kernel maintains dynamic tables of all known drivers, and provides a set of routines to allow drivers to be added to or removed from these tables at any time
- Registration tables include the following items:
  - Device drivers
  - File systems
  - Network protocols
  - Binary format

Conflict Resolution

- A mechanism that allows different device drivers to reserve hardware resources and to protect those resources from accidental use by another driver
- The conflict resolution module aims to:
  - Prevent modules from clashing over access to hardware resources
  - Prevent *autoprobes* from interfering with existing device drivers
  - Resolve conflicts with multiple drivers trying to access the same hardware

Process Management

- UNIX process management separates the creation of processes and the running of a new program into two distinct operations.
  - The **fork** system call creates a new process
  - A new program is run after a call to **execve**
- Under UNIX, a process encompasses all the information that the operating system must maintain *t* track the context of a single execution of a single program
- Under Linux, process properties fall into three groups: the process's identity, environment, and context

Process Identity

- Process ID (PID). The unique identifier for the process; used to specify processes to the operating system when an application makes a system call to signal, modify, or wait for another process
- Credentials. Each process must have an associated user ID and one or more group IDs that determine the process's rights to access system resources and files
- Personality. Not traditionally found on UNIX systems, but under Linux each process has an associated personality identifier that can slightly modify the semantics of certain system calls
  - Used primarily by emulation libraries to request that system calls be compatible with certain specific flavors of UNIX

Process Environment

- The process's environment is inherited from its parent, and is composed of two null-terminated vectors:
  - The argument vector lists the command-line arguments used to invoke the running program; conventionally starts with the name of the program itself
  - The environment vector is a list of "NAME=VALUE" pairs that associates named environment variables with arbitrary textual values
- Passing environment variables among processes and inheriting variables by a process's children are flexible means of passing information to components of the user-mode system software

Process Environment

- The environment-variable mechanism provides a customization of the operating system that can be set on a per-process basis, rather than being configured for the system as a whole

Process Context

- The (constantly changing) state of a running program at any point in time
- The **scheduling context** is the most important part of the process context; it is the information that the scheduler needs to suspend and restart the process
- The kernel maintains **accounting** information about the resources currently being consumed by each process, and the total resources consumed by the process in its lifetime so far

Process Context (cont.)

- The **file table** is an array of pointers to kernel file structures
  - When making file I/O system calls, processes refer to files by their index into this table
- Whereas the file table lists the existing open files, the **file-system context** applies to requests to open new files
  - The current root and default directories to be used for new file searches are stored here

Process Context (Cont.)

- The **signal-handler table** defines the routine in the process's address space to be called when specific signals arrive
- The **virtual-memory context** of a process describes the full contents of the its private address space

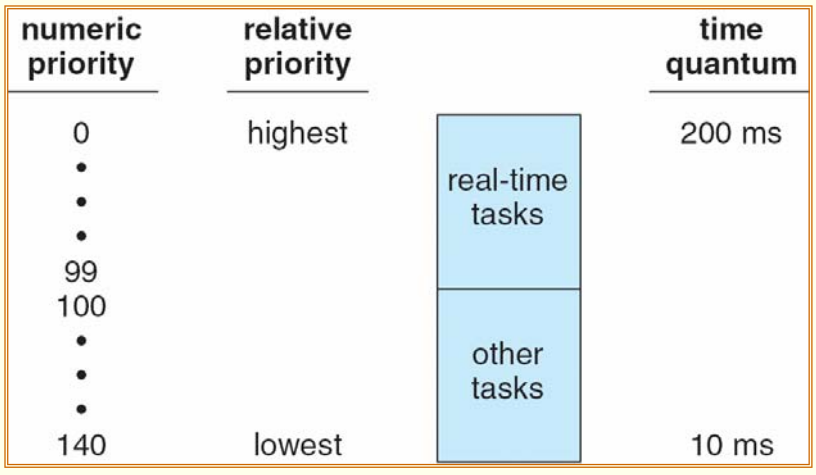
Processes and Threads

- Linux uses the same internal representation for processes and threads; a thread is simply a new process that happens to share the same address space as its parent
- A distinction is only made when a new thread is created by the **clone** system call
  - **fork** creates a new process with its own entirely new process context
  - **clone** creates a new process with its own identity, but that is allowed to share the data structures of its parent
- Using **clone** gives an application fine-grained control over exactly what is shared between two threads

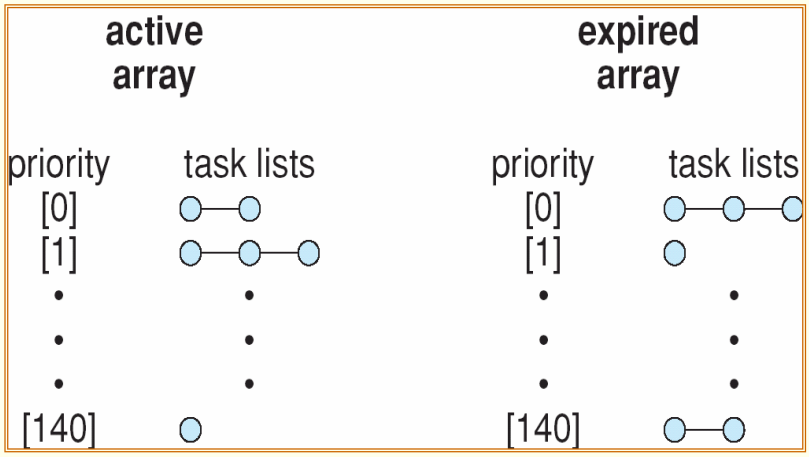
Scheduling

- The job of allocating CPU time to different tasks within an operating system
- While scheduling is normally thought of as the running and interrupting of processes, in Linux, scheduling also includes the running of the various kernel tasks
- Running kernel tasks encompasses both tasks that are requested by a running process and tasks that execute internally on behalf of a device driver
- As of 2.5, new scheduling algorithm – preemptive, priority-based
  - Real-time range
  - nice value

Relationship Between Priorities and Time-slice Length



List of Tasks Indexed by Priority



Kernel Synchronization

- A request for kernel-mode execution can occur in two ways:
  - A running program may request an operating system service, either explicitly via a system call, or implicitly, for example, when a page fault occurs
  - A device driver may deliver a hardware interrupt that causes the CPU to start executing a kernel-defined handler for that interrupt
- Kernel synchronization requires a framework that will allow the kernel's critical sections to run without interruption by another critical section

Kernel Synchronization (Cont.)

- Linux uses two techniques to protect critical sections:
  1. Normal kernel code is nonpreemptible (until 2.4)
     – when a time interrupt is received while a process is executing a kernel system service routine, the kernel's **need_resched** flag is set so that the scheduler will run once the system call has completed and control is about to be returned to user mode
  2. The second technique applies to critical sections that occur in an interrupt service routines
     – By using the processor's interrupt control hardware to disable interrupts during a critical section, the kernel guarantees that it can proceed without the risk of concurrent access of shared data structures

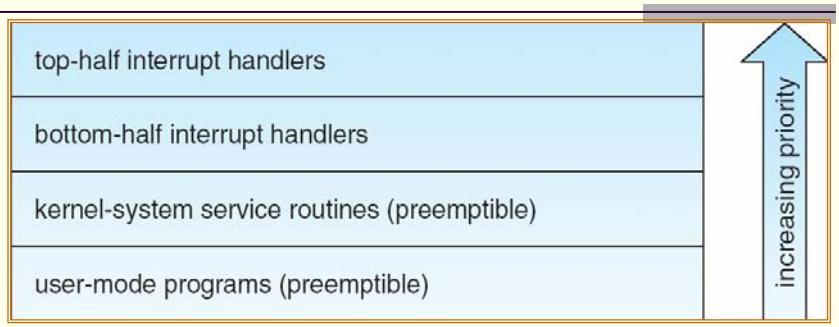
Kernel Synchronization (Cont.)

- To avoid performance penalties, Linux's kernel uses a synchronization architecture that allows long critical sections to run without having interrupts disabled for the critical section's entire duration

Kernel Synchronization (Cont.)

- Interrupt service routines are separated into a *top half* and a *bottom half.*
  - The top half is a normal interrupt service routine, and runs with recursive interrupts disabled
  - The bottom half is run, with all interrupts enabled, by a miniature scheduler that ensures that bottom halves never interrupt themselves
  - This architecture is completed by a mechanism for disabling selected bottom halves while executing normal, foreground kernel code

Interrupt Protection Levels



- Each level may be interrupted by code running at a higher level, but will never be interrupted by code running at the same or a lower level
- User processes can always be preempted by another process when a time-sharing scheduling interrupt occurs

Process Scheduling

- Linux uses two process-scheduling algorithms:
  - A time-sharing algorithm for fair preemptive scheduling between multiple processes
  - A real-time algorithm for tasks where absolute priorities are more important than fairness
- A process's scheduling class defines which algorithm to apply

Process Scheduling

- For time-sharing processes, Linux uses a prioritized, credit based algorithm
  - The crediting rule

$$\text{credits} := \frac{\text{credits}}{2} + \text{priority}$$

    factors in both the process's history and its priority
  - This crediting system automatically prioritizes interactive or I/O-bound processes

Process Scheduling (Cont.)

- Linux implements the FIFO and round-robin real-time scheduling classes; in both cases, each process has a priority in addition to its scheduling class
  - The scheduler runs the process with the highest priority; for equal-priority processes, it runs the process waiting the longest
  - FIFO processes continue to run until they either exit or block
  - A round-robin process will be preempted after a while and moved to the end of the scheduling queue, so that round-robing processes of equal priority automatically time-share between themselves

Symmetric Multiprocessing

- Linux 2.0 was the first Linux kernel to support SMP hardware; separate processes or threads can execute in parallel on separate processors
- To preserve the kernel's nonpreemptible synchronization requirements, SMP imposes the restriction, via a single kernel spinlock, that only one processor at a time may execute kernel-mode code

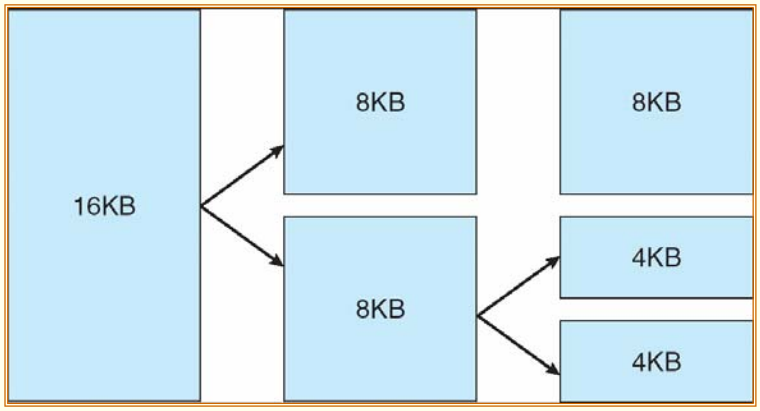
Memory Management

- Linux's physical memory-management system deals with allocating and freeing pages, groups of pages, and small blocks of memory
- It has additional mechanisms for handling virtual memory, memory mapped into the address space of running processes
- Splits memory into 3 different **zones** due to hardware characteristics

Relationship of Zones and Physical Addresses on 80x86

zone	physical memory
ZONE_DMA	< 16 MB
ZONE_NORMAL	16 .. 896 MB
ZONE_HIGHMEM	> 896 MB

Splitting of Memory in a Buddy Heap



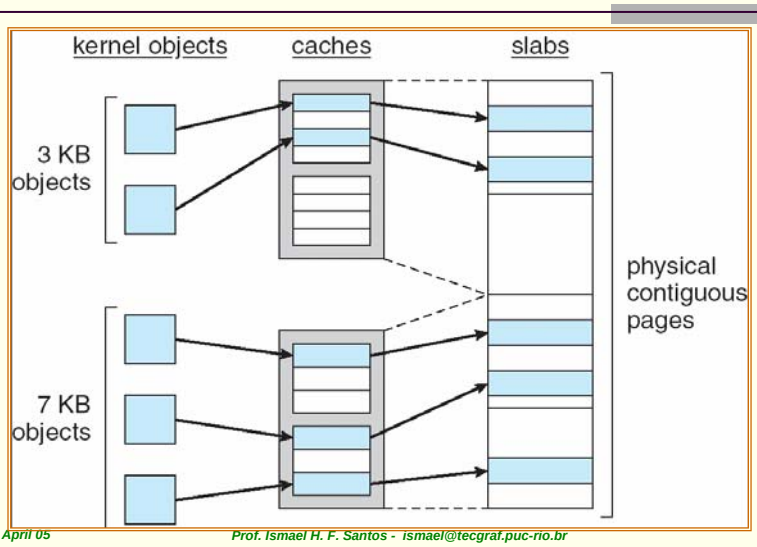
Managing Physical Memory

- The page allocator allocates and frees all physical pages; it can allocate ranges of physically-contiguous pages on request
- The allocator uses a buddy-heap algorithm to keep track of available physical pages
  - Each allocatable memory region is paired with an adjacent partner
  - Whenever two allocated partner regions are both freed up they are combined to form a larger region
  - If a small memory request cannot be satisfied by allocating an existing small free region, then a larger free region will be subdivided into two partners to satisfy the request

Managing Physical Memory

- Memory allocations in the Linux kernel occur either statically (drivers reserve a contiguous area of memory during system boot time) or dynamically (via the page allocator)
- Also uses **slab allocator** for kernel memory

Managing Physical Memory (cont.)



Virtual Memory

- The VM system maintains the address space visible to each process: It creates pages of virtual memory on demand, and manages the loading of those pages from disk or their swapping back out to disk as required
- The VM manager maintains two separate views of a process's address space:
  - A logical view describing instructions concerning the layout of the address space
    - The address space consists of a set of nonoverlapping regions, each representing a continuous, page-aligned subset of the address space
  - A physical view of each address space which is stored in the hardware page tables for the process

Virtual Memory (Cont.)

- Virtual memory regions are characterized by:
  - The backing store, which describes from where the pages for a region come; regions are usually backed by a file or by nothing (*demand-zero* memory)
  - The region's reaction to writes (page sharing or copy-on-write)
- The kernel creates a new virtual address space
  1. When a process runs a new program with the **exec** system call
  2. Upon creation of a new process by the **fork** system call

Virtual Memory (Cont.)

- On executing a new program, the process is given a new, completely empty virtual-address space; the program-loading routines populate the address space with virtual-memory regions
- Creating a new process with **fork** involves creating a complete copy of the existing process's virtual address space
  - The kernel copies the parent process's VMA descriptors, then creates a new set of page tables for the child
  - The parent's page tables are copied directly into the child's, with the reference count of each page covered being incremented

Virtual Memory (Cont.)

  - After the fork, the parent and child share the same physical pages of memory in their address spaces
- The VM paging system relocates pages of memory from physical memory out to disk when the memory is needed for something else
- The VM paging system can be divided into two sections:
  - The pageout-policy algorithm decides which pages to write out to disk, and when
  - The paging mechanism actually carries out the transfer, and pages data back into physical memory as needed

Virtual Memory (Cont.)

- The Linux kernel reserves a constant, architecture-dependent region of the virtual address space of every process for its own internal use
- This kernel virtual-memory area contains two regions:
  - A static area that contains page table references to every available physical page of memory in the system, so that there is a simple translation from physical to virtual addresses when running kernel code
  - The reminder of the reserved section is not reserved for any specific purpose; its page-table entries can be modified to point to any other areas of memory

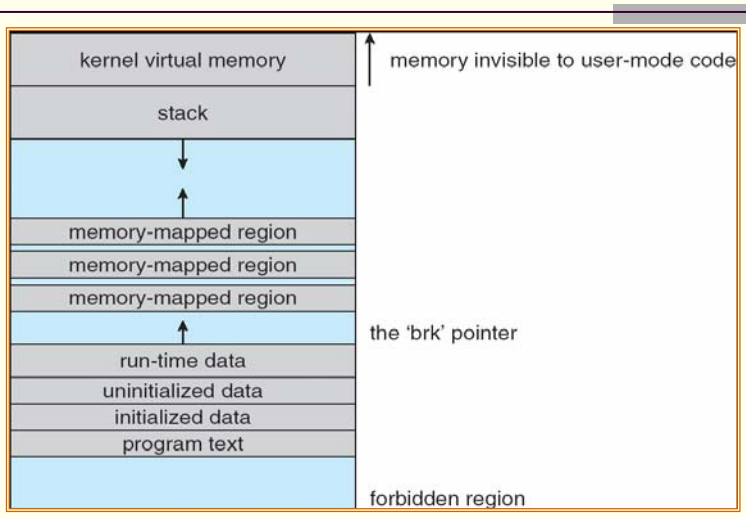
Executing and Loading User Programs

- 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

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



Static and Dynamic Linking

- A program whose necessary library functions are embedded directly in the program's executable binary file is *statically* linked to its libraries
- The main disadvantage of static linkage is that every program generated must contain copies of exactly the same common system library functions
- *Dynamic* linking is more efficient in terms of both physical memory and disk-space usage because it loads the system libraries into memory only once

File Systems

- To the user, Linux's file system appears as a hierarchical directory tree obeying UNIX semantics
- Internally, the kernel hides implementation details and manages the multiple different file systems via an abstraction layer, that is, the *virtual file system (VFS)*
- The Linux VFS is designed around object-oriented principles and is composed of two components:
  - A set of definitions that define what a file object is allowed to look like
    - The *inode-object* and the *file-object* structures represent individual files
    - the *file system object* represents an entire file system
  - A layer of software to manipulate those objects

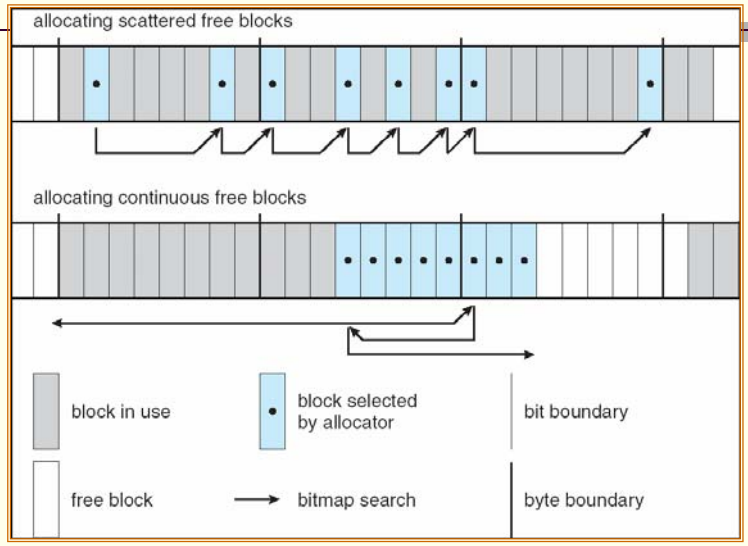
The Linux Ext2fs File System

- Ext2fs uses a mechanism similar to that of BSD Fast File System (ffs) for locating data blocks belonging to a specific file
- The main differences between ext2fs and ffs concern their disk allocation policies
  - In ffs, the disk is allocated to files in blocks of 8Kb, with blocks being subdivided into fragments of 1Kb to store small files or partially filled blocks at the end of a file

The Linux Ext2fs File System

- Ext2fs does not use fragments; it performs its allocations in smaller units
  - The default block size on ext2fs is 1Kb, although 2Kb and 4Kb blocks are also supported
- Ext2fs uses allocation policies designed to place logically adjacent blocks of a file into physically adjacent blocks on disk, so that it can submit an I/O request for several disk blocks as a single operation

Ext2fs Block-Allocation Policies



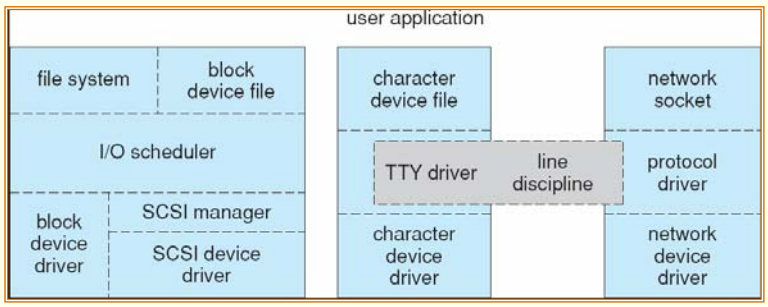
The Linux Proc File System

- The **proc** file system does not store data, rather, its contents are computed on demand according to user file I/O requests
- **proc** must implement a directory structure, and the file contents within; it must then define a unique and persistent inode number for each directory and files it contains
  - It uses this inode number to identify just what operation is required when a user tries to read from a particular file inode or perform a lookup in a particular directory inode
  - When data is read from one of these files, **proc** collects the appropriate information, formats it into text form and places it into the requesting process's read buffer

Input and Output

- The Linux device-oriented file system accesses disk storage through two caches:
  - Data is cached in the page cache, which is unified with the virtual memory system
  - Metadata is cached in the buffer cache, a separate cache indexed by the physical disk block
- Linux splits all devices into three classes:
  - *block devices* allow random access to completely independent, fixed size blocks of data
  - *character devices* include most other devices; they don't need to support the functionality of regular files
  - *network devices* are interfaced via the kernel's networking subsystem

Device-Driver Block Structure



Block Devices

- Provide the main interface to all disk devices in a system
- The *block buffer* cache serves two main purposes:
  - it acts as a pool of buffers for active I/O
  - it serves as a cache for completed I/O
- The *request manager* manages the reading and writing of buffer contents to and from a block device driver

Character Devices

- A device driver which does not offer random access to fixed blocks of data
- A character device driver must register a set of functions which implement the driver's various file I/O operations
- The kernel performs almost no preprocessing of a file read or write request to a character device, but simply passes on the request to the device
- The main exception to this rule is the special subset of character device drivers which implement terminal devices, for which the kernel maintains a standard interface

Interprocess Communication

- Like UNIX, Linux informs processes that an event has occurred via signals
- There is a limited number of signals, and they cannot carry information: Only the fact that a signal occurred is available to a process
- The Linux kernel does not use signals to communicate with processes with are running in kernel mode, rather, communication within the kernel is accomplished via scheduling states and **wait.queue** structures

Passing Data Between Processes

- The pipe mechanism allows a child process to inherit a communication channel to its parent, data written to one end of the pipe can be read a the other
- Shared memory offers an extremely fast way of communicating; any data written by one process to a shared memory region can be read immediately by any other process that has mapped that region into its address space
- To obtain synchronization, however, shared memory must be used in conjunction with another Interprocess-communication mechanism

Shared Memory Object

- The shared-memory object acts as a backing store for shared-memory regions in the same way as a file can act as backing store for a memory-mapped memory region
- Shared-memory mappings direct page faults to map in pages from a persistent shared-memory object
- Shared-memory objects remember their contents even if no processes are currently mapping them into virtual memory

Network Structure

- Networking is a key area of functionality for Linux.
  - It supports the standard Internet protocols for UNIX to UNIX communications
  - It also implements protocols native to nonUNIX operating systems, in particular, protocols used on PC networks, such as Appletalk and IPX
- Internally, networking in the Linux kernel is implemented by three layers of software:
  - The socket interface
  - Protocol drivers
  - Network device drivers

Network Structure (Cont.)

- The most important set of protocols in the Linux networking system is the internet protocol suite
  - It implements routing between different hosts anywhere on the network
  - On top of the routing protocol are built the UDP, TCP and ICMP protocols

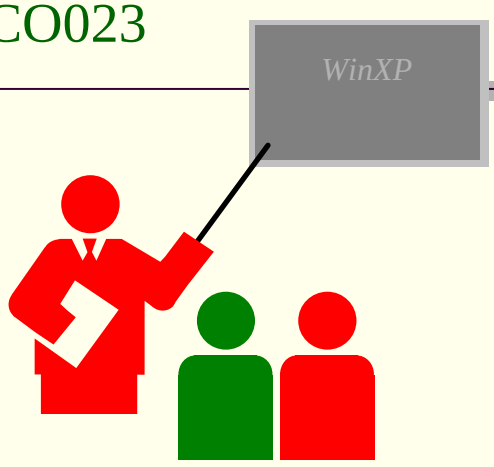
Security

- The *pluggable authentication modules (PAM)* system is available under Linux
- PAM is based on a shared library that can be used by any system component that needs to authenticate users
- Access control under UNIX systems, including Linux, is performed through the use of unique numeric identifiers (**uid** and **gid**)
- Access control is performed by assigning objects a *protections mask*, which specifies which access modes—read, write, or execute—are to be granted to processes with owner, group, or world access

Security (Cont.)

- Linux augments the standard UNIX **setuid** mechanism in two ways:
  - It implements the POSIX specification's saved *user-id* mechanism, which allows a process to repeatedly drop and reacquire its effective uid
  - It has added a process characteristic that grants just a subset of the rights of the effective uid
- Linux provides another mechanism that allows a client to selectively pass access to a single file to some server process without granting it any other privileges

SOP – CO023



Objectives

- To explore the principles upon which Windows XP is designed and the specific components involved in the system
- To understand how Windows XP can run programs designed for other operating systems
- To provide a detailed explanation of the Windows XP file system
- To illustrate the networking protocols supported in Windows XP
- To cover the interface available to system and application programmers

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

History

- In 1988, Microsoft decided to develop a “new technology” (NT) portable operating system that supported both the OS/2 and POSIX APIs
- Originally, NT was supposed to use the OS/2 API as its native environment but during development NT was changed to use the Win32 API, reflecting the popularity of Windows 3.0

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)

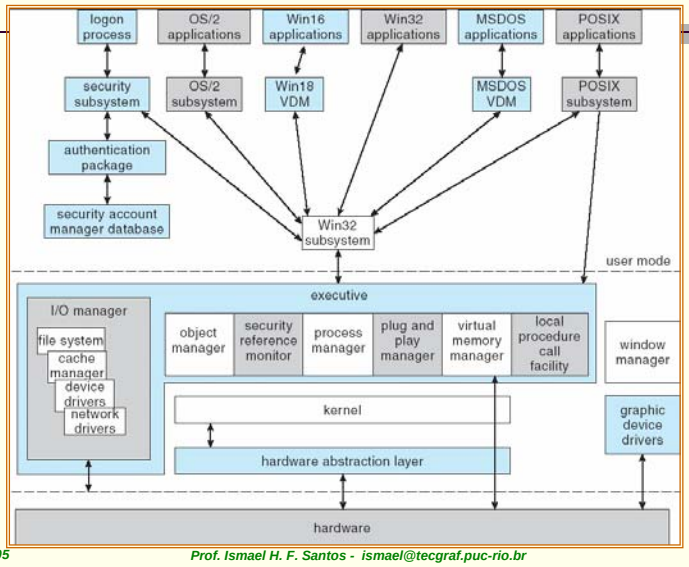
Design Principles (Cont.)

- 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 complied 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

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

Depiction of XP Architecture



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

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 — Scheduling

- The dispatcher uses a 32-level priority scheme to determine the order of thread execution
  - Priorities are divided into two classes
    - The real-time class contains threads with priorities ranging from 16 to 31
    - The variable class contains threads having priorities from 0 to 15
- Characteristics of XP's priority strategy
  - Trends to give very good response times to interactive threads that are using the mouse and windows
  - Enables I/O-bound threads to keep the I/O devices busy
  - Complete-bound threads soak up the spare CPU cycles in the background

Kernel — Scheduling (Cont.)

- Scheduling can occur when a thread enters the ready or wait state, when a thread terminates, or when an application changes a thread's priority or processor affinity
- Real-time threads are given preferential access to the CPU; but XP does not guarantee that a real-time thread will start to execute within any particular time limit
  - This is known as *soft realtime*

Windows XP Interrupt Request Levels

interrupt levels	types of interrupts
31	machine check or bus error
30	power fail
29	interprocessor notification (request another processor to act; e.g., dispatch a process or update the TLB)
28	clock (used to keep track of time)
27	profile
3–26	traditional PC IRQ hardware interrupts
2	dispatch and deferred procedure call (DPC) (kernel)
1	asynchronous procedure call (APC)
0	passive

Kernel — Trap Handling

- The kernel provides trap handling when exceptions and interrupts are generated by hardware of software
- Exceptions that cannot be handled by the trap handler are handled by the kernel's *exception dispatcher*
- The interrupt dispatcher in the kernel handles interrupts by calling either an interrupt service routine (such as in a device driver) or an internal kernel routine
- The kernel uses spin locks that reside in global memory to achieve multiprocessor mutual exclusion

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`

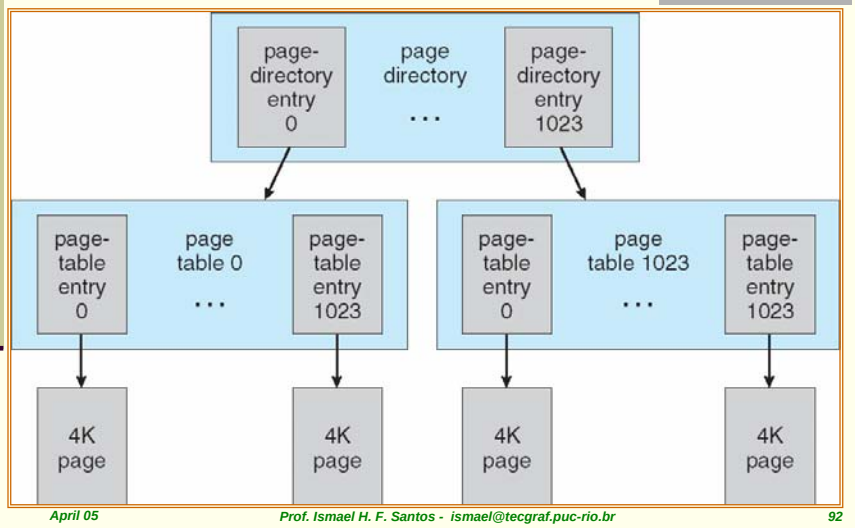
Executive — Naming Objects

- 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

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

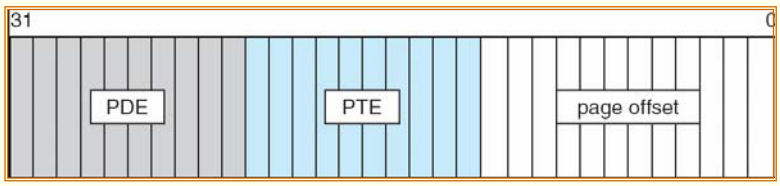
Virtual-Memory Layout



Virtual Memory Manager (Cont.)

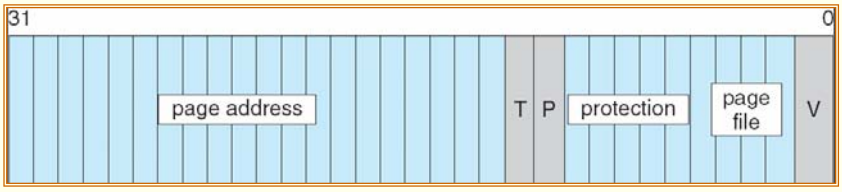
- The virtual address translation in XP uses several data structures
  - Each process has a *page directory* that contains 1024 *page directory entries* of size 4 bytes
  - Each page directory entry points to a *page table* which contains 1024 *page table entries* (PTEs) of size 4 bytes
  - Each PTE points to a 4 KB *page frame* in physical memory
- A 10-bit integer can represent all the values form 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 bye address in physical memory
- A page can be in one of six states: valid, zeroed, free standby, modified and bad

Virtual-to-Physical Address Translation



- 10 bits for page directory entry, 20 bits for page table entry, and 12 bits for byte offset in page

Page File Page-Table Entry



5 bits for page protection, 20 bits for page frame address, 4 bits to select a paging file, and 3 bits that describe the page state. V = 0

Executive — Process Manager

- Provides services for creating, deleting, and using threads and processes.
- Issues such as parent/child relationships or process hierarchies are left to the particular environmental subsystem that owns the process.

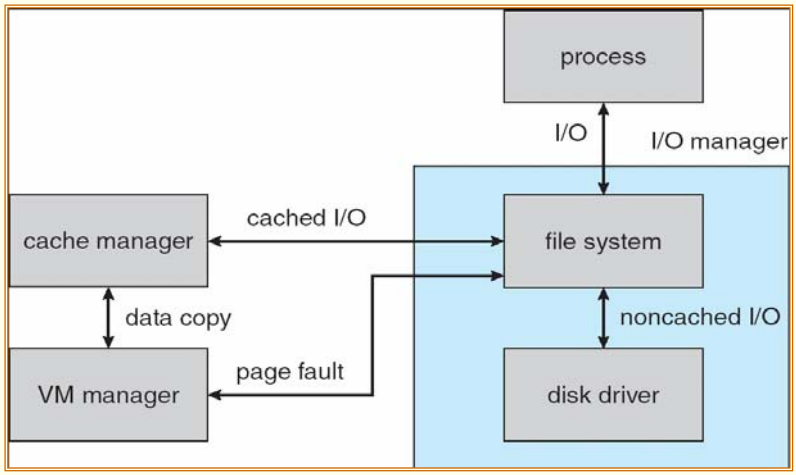
Executive — Local Procedure Call Facility

- The LPC passes requests and results between client and server processes within a single machine.
- In particular, it is used to request services from the various XP subsystems.
- When a LPC channel is created, one of three types of message passing techniques must be specified.
  - First type is suitable for small messages, up to 256 bytes; port's message queue is used as intermediate storage, and the messages are copied from one process to the other.
  - Second type avoids copying large messages by pointing to a shared memory section object created for the channel.
  - Third method, called *quick* LPC was used by graphical display portions of the Win32 subsystem.

Executive — I/O Manager

- The I/O manager is responsible for
  - file systems
  - cache management
  - device drivers
  - network drivers
- Keeps track of which installable file systems are loaded, and manages buffers for I/O requests
- Works with VM Manager to provide memory-mapped file I/O
- Controls the XP cache manager, which handles caching for the entire I/O system
- Supports both synchronous and asynchronous operations, provides time outs for drivers, and has mechanisms for one driver to call another

File I/O



Executive — Security Reference Monitor

- The object-oriented nature of XP enables the use of a uniform mechanism to perform runtime access validation and audit checks for every entity in the system
- Whenever a process opens a handle to an object, the security reference monitor checks the process's security token and the object's access control list to see whether the process has the necessary rights

Executive – Plug-and-Play Manager

- Plug-and-Play (PnP) manager is used to recognize and adapt to changes in the hardware configuration
- When new devices are added (for example, PCI or USB), the PnP manager loads the appropriate driver
- The manager also keeps track of the resources used by each device

Environmental Subsystems

- User-mode processes layered over the native XP executive services to enable XP to run programs developed for other operating system
- XP uses the Win32 subsystem as the main operating environment; Win32 is used to start all processes
  - It also provides all the keyboard, mouse and graphical display capabilities
- MS-DOS environment is provided by a Win32 application called the *virtual dos machine* (VDM), a user-mode process that is paged and dispatched like any other XP thread

Environmental Subsystems (Cont.)

- 16-Bit Windows Environment:
  - Provided by a VDM that incorporates *Windows on Windows*
  - Provides the Windows 3.1 kernel routines and sub routines for window manager and GDI functions
- The POSIX subsystem is designed to run POSIX applications following the POSIX.1 standard which is based on the UNIX model

Environmental Subsystems (Cont.)

- OS/2 subsystems runs OS/2 applications
- Logon and Security Subsystems authenticates users logging on to Windows XP systems
  - Users are required to have account names and passwords
  - The authentication package authenticates users whenever they attempt to access an object in the system
  - Windows XP uses Kerberos as the default authentication package

File System

- The fundamental structure of the XP file system (NTFS) is a *volume*
  - Created by the XP disk administrator utility
  - Based on a logical disk partition
  - May occupy a portions of a disk, an entire disk, or span across several disks
- All *metadata*, such as information about the volume, is stored in a regular file

File System (cont.)

- NTFS uses *clusters* as the underlying unit of disk allocation
  - A cluster is a number of disk sectors that is a power of two
  - Because the cluster size is smaller than for the 16-bit FAT file system, the amount of internal fragmentation is reduced
- NTFS uses logical cluster numbers (LCNs) as disk addresses
- A file in NTFS is not a simple byte stream, as in MS-DOS or UNIX, rather, it is a structured object consisting of attributes

File System — Internal Layout

- Every file in NTFS is described by one or more records in an array stored in a special file called the Master File Table (MFT)
- Each file on an NTFS volume has a unique ID called a file reference.
  - 64-bit quantity that consists of a 48-bit file number and a 16-bit sequence number
  - Can be used to perform internal consistency checks
- The NTFS name space is organized by a hierarchy of directories; the index root contains the top level of the B+ tree

File System — Recovery

- All file system data structure updates are performed inside transactions that are logged
  - Before a data structure is altered, the transaction writes a log record that contains redo and undo information
  - After the data structure has been changed, a commit record is written to the log to signify that the transaction succeeded
  - After a crash, the file system data structures can be restored to a consistent state by processing the log records

File System — Recovery (Cont.)

- This scheme does not guarantee that all the user file data can be recovered after a crash, just that the file system data structures (the metadata files) are undamaged and reflect some consistent state prior to the crash
- The log is stored in the third metadata file at the beginning of the volume
- The logging functionality is provided by the XP *log file service*

File System — Security

- Security of an NTFS volume is derived from the XP object model
- Each file object has a security descriptor attribute stored in this MFT record
- This attribute contains the access token of the owner of the file, and an access control list that states the access privileges that are granted to each user that has access to the file

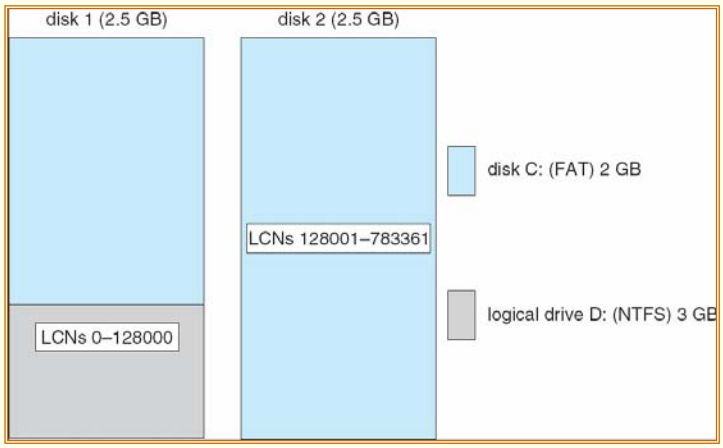
Volume Management and Fault Tolerance

- `FtDisk`, the fault tolerant disk driver for XP, provides several ways to combine multiple SCSI disk drives into one logical volume
- Logically concatenate multiple disks to form a large logical volume, a *volume set*
- Interleave multiple physical partitions in round-robin fashion to form a *stripe set* (also called RAID level 0, or "disk striping")
  - Variation: *stripe set with parity,* or RAID level 5

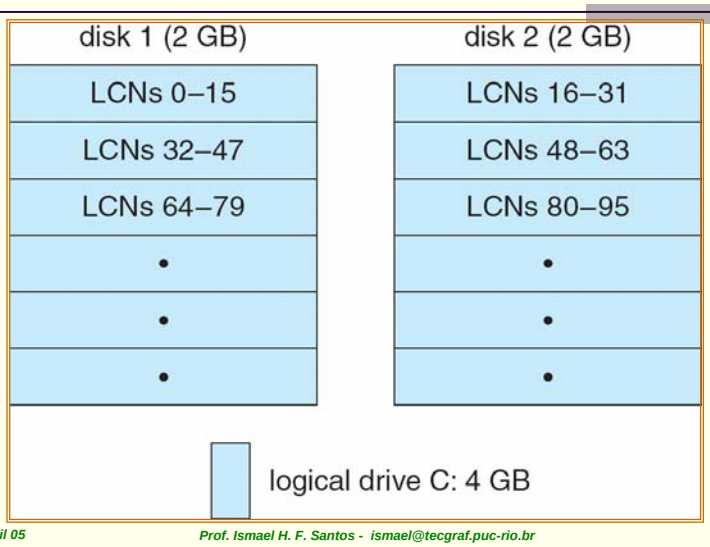
Volume Management and Fault Tolerance (cont.)

- Disk mirroring, or RAID level 1, is a robust scheme that uses a *mirror set* — two equally sized partitions on tow disks with identical data contents
- To deal with disk sectors that go bad, `FtDisk`, uses a hardware technique called *sector sparing* and NTFS uses a software technique called *cluster remapping*

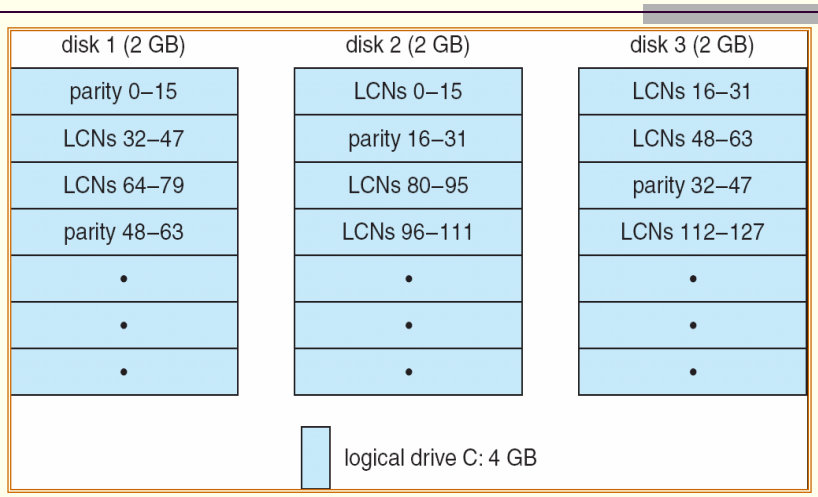
Volume Set On Two Drives



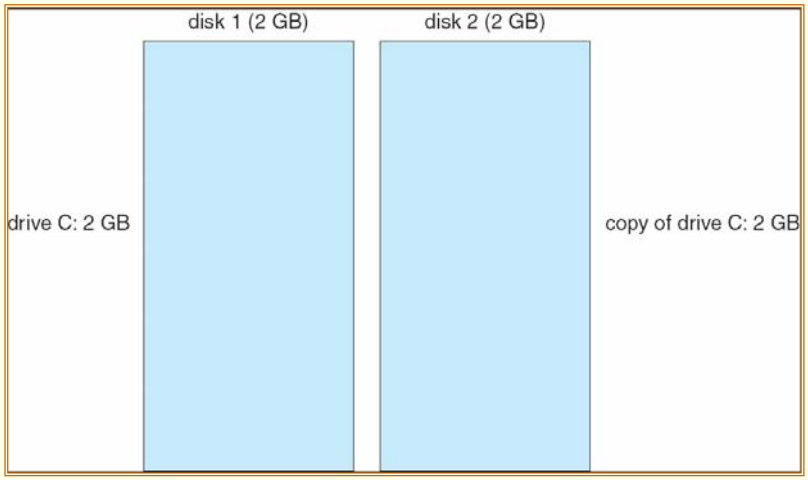
Stripe Set on Two Drives



Stripe Set With Parity on Three Drives



Mirror Set on Two Drives



File System — Compression

- To compress a file, NTFS divides the file's data into *compression units*, which are blocks of 16 contiguous clusters
- For sparse files, NTFS uses another technique to save space
  - Clusters that contain all zeros are not actually allocated or stored on disk
  - Instead, gaps are left in the sequence of virtual cluster numbers stored in the MFT entry for the file
  - When reading a file, if a gap in the virtual cluster numbers is found, NTFS just zero-fills that portion of the caller's buffer

File System — Reparse Points

- A reparse point returns an error code when accessed. The reparse data tells the I/O manager what to do next
- Reparse points can be used to provide the functionality of UNIX *mounts*
- Reparse points can also be used to access files that have been moved to offline storage

Networking

- XP supports both peer-to-peer and client/server networking; it also has facilities for network management
- To describe networking in XP, we refer to two of the internal networking interfaces:
  - NDIS (Network Device Interface Specification) — Separates network adapters from the transport protocols so that either can be changed without affecting the other
  - TDI (Transport Driver Interface) — Enables any session layer component to use any available transport mechanism
- XP implements transport protocols as drivers that can be loaded and unloaded from the system dynamically

Networking — Protocols

- The server message block (SMB) protocol is used to send I/O requests over the network. It has 4 message types:
  - Session control - File
  - Printer - Message
- The network basic Input/Output system (NetBIOS) is a hardware abstraction interface for networks
  - Used to:
    - Establish logical names on the network
    - Establish logical connections of sessions between two logical names on the network
    - Support reliable data transfer for a session via NetBIOS requests or *SMBs*

Networking — Protocols (Cont.)

- NetBEUI (NetBIOS Extended User Interface): default protocol for Windows 95 peer networking and Windows for Workgroups; used when XP wants to share resources with these networks
- XP uses the TCP/IP Internet protocol to connect to a wide variety of operating systems and hardware platforms
- PPTP (Point-to-Point Tunneling Protocol) is used to communicate between Remote Access Server modules running on XP machines that are connected over the Internet
- The XP NWLink protocol connects the NetBIOS to Novell NetWare networks

Networking — Protocols (Cont.)

- The Data Link Control protocol (DLC) is used to access IBM mainframes and HP printers that are directly connected to the network
- XP systems can communicate with Macintosh computers via the Apple Talk protocol if an XP Server on the network is running the Windows XP Services for Macintosh package

Networking — Dist. Processing Mechanisms

- XP supports distributed applications via named NetBIOS, named pipes and mailslots, Windows Sockets, Remote Procedure Calls (RPC), and Network Dynamic Data Exchange (NetDDE)
- NetBIOS applications can communicate over the network using NetBEUI, NWLink, or TCP/IP
- Named pipes are connection-oriented messaging mechanism that are named via the uniform naming convention (UNC)

Networking — Dist. Processing Mechanisms (cont.)

- Mailslots are a connectionless messaging mechanism that are used for broadcast applications, such as for finding components on the network
- Winsock, the windows sockets API, is a session-layer interface that provides a standardized interface to many transport protocols that may have different addressing schemes

Distributed Processing Mechanisms (Cont.)

- The XP RPC mechanism follows the widely-used Distributed Computing Environment standard for RPC messages, so programs written to use XP RPCs are very portable
  - RPC messages are sent using NetBIOS, or Winsock on TCP/IP networks, or named pipes on LAN Manager networks
  - XP provides the Microsoft *Interface Definition Language* to describe the remote procedure names, arguments, and results

Networking — Redirectors and Servers

- In XP , an application can use the XP I/O API to access files from a remote computer as if they were local, provided that the remote computer is running an MS-NET server
- A *redirector* is the client-side object that forwards I/O requests to remote files, where they are satisfied by a server
- For performance and security, the redirectors and servers run in kernel mode

Access to a Remote File

- The application calls the I/O manager to request that a file be opened (we assume that the file name is in the standard UNC format)
- The I/O manager builds an I/O request packet
- The I/O manager recognizes that the access is for a remote file, and calls a driver called a Multiple Universal Naming Convention Provider (MUP)

Access to a Remote File

- The MUP sends the I/O request packet asynchronously to all registered redirectors
- A redirector that can satisfy the request responds to the MUP
  - To avoid asking all the redirectors the same question in the future, the MUP uses a cache to remember with redirector can handle this file

Access to a Remote File (Cont.)

- The redirector sends the network request to the remote system
- The remote system network drivers receive the request and pass it to the server driver
- The server driver hands the request to the proper local file system driver
- The proper device driver is called to access the data
- The results are returned to the server driver, which sends the data back to the requesting redirector

Networking — Domains

- NT uses the concept of a domain to manage global access rights within groups
- A domain is a group of machines running NT server that share a common security policy and user database
- XP provides three models of setting up trust relationships
  - *One way, A trusts B*
  - *Two way, transitive, A trusts B, B trusts C so A, B, C trust each other*
  - *Crosslink – allows authentication to bypass hierarchy to cut down on authentication traffic*

Name Resolution in **TCP/IP** Networks

- On an IP network, name resolution is the process of converting a computer name to an IP address
  e.g., www.bell-labs.com resolves to 135.104.1.14
- XP provides several methods of name resolution:
  - Windows Internet Name Service (WINS)
  - broadcast name resolution
  - domain name system (DNS)
  - a host file
  - an LMHOSTS file

Name Resolution (Cont.)

- WINS consists of two or more WINS servers that maintain a dynamic database of name to IP address bindings, and client software to query the servers
- WINS uses the Dynamic Host Configuration Protocol (DHCP), which automatically updates address configurations in the WINS database, without user or administrator intervention

Programmer Interface — Access to Kernel Obj.

- A process gains access to a kernel object named XXX by calling the CreateXXX function to open a *handle* to XXX; the handle is unique to that process
- A handle can be closed by calling the CloseHandle function; the system may delete the object if the count of processes using the object drops to 0

Programmer Interface — Access to Kernel Obj. (cont.)

- XP provides three ways to share objects between processes
  - A child process inherits a handle to the object
  - One process gives the object a name when it is created and the second process opens that name
  - DuplicateHandle function:
    - Given a handle to process and the handle's value a second process can get a handle to the same object, and thus share it

Programmer Interface — Process Management

- Process is started via the CreateProcess routine which loads any dynamic link libraries that are used by the process, and creates a *primary thread*
- Additional threads can be created by the CreateThread function
- Every dynamic link library or executable file that is loaded into the address space of a process is identified by an *instance handle*

Process Management (Cont.)

- Scheduling in Win32 utilizes four priority classes:
  - IDLE_PRIORITY_CLASS (priority level 4)
  - NORMAL_PRIORITY_CLASS (level8 — typical for most processes
  - HIGH_PRIORITY_CLASS (level 13)
  - REALTIME_PRIORITY_CLASS (level 24)

Process Management (Cont.)

- To provide performance levels needed for interactive programs, XP has a special scheduling rule for processes in the NORMAL_PRIORITY_CLASS
  - XP distinguishes between the *foreground process* that is currently selected on the screen, and the *background processes* that are not currently selected
  - When a process moves into the foreground, XP increases the scheduling quantum by some factor, typically 3

Process Management (Cont.)

- The kernel dynamically adjusts the priority of a thread depending on whether it is I/O-bound or CPU-bound
- To synchronize the concurrent access to shared objects by threads, the kernel provides synchronization objects, such as semaphores and mutexes
  - In addition, threads can synchronize by using the WaitForSingleObject or WaitForMultipleObjects functions
  - Another method of synchronization in the Win32 API is the critical section

Process Management (Cont.)

- A fiber is user-mode code that gets scheduled according to a user-defined scheduling algorithm
  - Only one fiber at a time is permitted to execute, even on multiprocessor hardware
  - XP includes fibers to facilitate the porting of legacy UNIX applications that are written for a fiber execution model

Programmer Interface — Interprocess Comm.

- Win32 applications can have interprocess communication by sharing kernel objects
- An alternate means of interprocess communications is message passing, which is particularly popular for Windows GUI applications
  - One thread sends a message to another thread or to a window
  - A thread can also send data with the message

Programmer Interface — Interprocess Comm. (cont.)

- Every Win32 thread has its own input queue from which the thread receives messages
- This is more reliable than the shared input queue of 16-bit windows, because with separate queues, one stuck application cannot block input to the other applications

Programmer Interface — Memory Management

- Virtual memory:
  - VirtualAlloc reserves or commits virtual memory
  - VirtualFree decommits or releases the memory
  - These functions enable the application to determine the virtual address at which the memory is allocated
- An application can use memory by memory mapping a file into its address space
  - Multistage process
  - Two processes share memory by mapping the same file into their virtual memory

Memory Management (Cont.)

- A heap in the Win32 environment is a region of reserved address space
  - A Win 32 process is created with a 1 MB *default heap*
  - Access is synchronized to protect the heap's space allocation data structures from damage by concurrent updates by multiple threads
- Because functions that rely on global or static data typically fail to work properly in a multithreaded environment, the thread-local storage mechanism allocates global storage on a per-thread basis
  - The mechanism provides both dynamic and static methods of creating thread-local storage