***Course Code//Semester*:-**	CSC 413/First
Course Title:-	Computer Operation II
***Course Unit*:-**	3 Units
***Course Lecturer*:-**	DR. A. S. SODIYA
Contact Hours:-	4hrs per week

COURSE DETAILS:

COURSE CONTENT:

Files system, memory management, job management and mathematical theory of jobs scheduling operating systems implementation techniques, modular and abstraction: synchronization, PSV operation, detailed studies of some operating systems.

COURSE REQUIREMENTS:

The course will be examined at the end of the semester and the examination score is 70%. Continuous Assessment Test (CAT) will be 30%, making a total of 100% for the course. The CAT consists of Mid-Semester Test, Practical, Assignment, Quiz and Presentation.

LECTURE NOTES

CHAPTER ONE - UNIX OPERATING SYSTEM

1.0 INTRODUCTION TO UNIX OPERATING SYSTEM

An operating system is the suite of programs which make the computer work.

The UNIX operating system was designed to let a number of programmers access the computer at the same time and share its resources. This real-time sharing of resources makes UNIX one of the most powerful operating systems ever.

Features of UNIX operating system:

- Multitasking capability (UNIX lets a computer do several things at once,)
- Multiuser capability (The computer can take the commands of a number of users -- determined by the design of the computer -- to run programs, access files, and print documents at the same time.)
- Portability (permit to move from one brand of computer to another with a minimum of code changes.)
- Library of application software

COMPONENT OF UNIX O/S :

The UNIX operating system is made up of *three parts*; the kernel, the shell and the programs.

1. The kernel :

The kernel of UNIX is the hub of the operating system: it allocates time and memory to programs and handles the file storage and communications in response to system calls.
An illustration of the way that the shell and the kernel work together, suppose a user types **rm myfile** (which has the effect of removing the file **myfile**). The shell searches the filestore for the file containing the program **rm**, and then requests the kernel, through system calls, to execute the program **rm** on **myfile**. When the process **rm myfile** has finished running, the shell then returns the UNIX prompt to the user, indicating that it is waiting for further commands.

2. The shell :

The shell acts as an interface between the user and the kernel. When a user logs in, the login program checks the username and password, and then starts another program called the shell. The shell is a command line interpreter (CLI) which interprets the commands the user types in and then arranges for them to be carried out.

The commands are themselves programs. When they terminate, the shell would give the user another prompt.

The adept user can customize his/her own shell, and users can use different shells on the same machine. As an illustration the shell may be customized with certain features to help the user in inputting commands, filename Completion - By typing part of the name of a command, filename or directory and pressing the [**Tab**] key, the shell will complete the rest of the name automatically. If the shell finds more than one name beginning with the letters that has been typed, it would beep, prompting the user to type a few more letters before pressing the tab key again.

3. Programs

Program consists of the tools and applications that offer additional functionality to the operating system. Typically, tools are grouped into categories for certain functions, such as word processing, business applications, or programming.

2.0 HISTORY

The history of UNIX starts back in 1969, when a small group composed of Ken Thompson, Dennis Ritchie and others started working on the "little-used PDP-7 in a corner" at Bell Lab and what was to become UNIX. For 10years,the development of UNIX proceeded at AT&T in numbered versions. The 1974 version was re-written in C, a major mile stone for the operating system's portability among different systems .The 1975 version was the first to become available outside Bell Lab. It became the basis of the first version of UNIX developed at the university of California Berkely. By 1983, Computer Research Group (CRG), UNIX System Group (USG) and a third group merge to become UNIX System Development Lab and AT&T announces UNIX System V, the first supported release. Installed base was 45,000. The goals of the group were to design an operating system to satisfy the following objectives:

- Simple and elegant

- Written in a high level language rather than assembly language
- Allow re-use of code

The group worked primarily in the high level language in developing the operating system. The first edition was released in 1971, it had an assembler for a PDP-11/20, file system, fork(), roff and ed. It was used for text processing of patent document.

In 1998, X/Open introduced the UNIX 95. In 1995, a branding programme for implementations of the Single UNIX Specification. Novell sold UnixWare business line to SCO and was Digital UNIX introduced in the same year.

In 1999, the UNIX system reaches its 30th anniversary. Linux 2.2 kernel was released. The Open Group and the IEEE commence joint development of a revision to POSIX and the Single UNIX Specification.

In 2003, The core volumes of Version 3 of the Single UNIX Specification were approved as an international standard.

In addition, while initially designed for medium-sized minicomputers, the operating system was soon moved to larger, more powerful mainframe computers. As personal computers grew in popularity, versions of UNIX found their way into these boxes, and a number of companies produce UNIX-based machines for the scientific and programming communities.

UNIX POPULARITY

Many vendors have decide to use UNIX because of the following reasons :

- UNIX is relatively easy to run. Only a very small amount of its codes are written in assembly language. UNIX is nearly the unanimous choice of operating system for computer companies started since 1985. The user benefit which results from this is that UNIX runs on a wide variety of computer systems. Many traditional vendors have made UNIX available on their systems in addition to their proprietary operating systems.

- The application program interface allows many different types of applications to be easily implemented under UNIX without writing assembly language. These applications are relatively portable across multiple vendor hardware platforms. Third party software vendors can save costs by supporting a single UNIX version of their software rather than four completely different vendor specific versions requiring four times the maintenance.
- Vendor-independent networking allows users to easily network multiple systems from many different vendors.

3.0 DESIGN ISSUES OF UNIX

UNIX is a stable, multi-user, multi-tasking system for servers, desktop and laptop. It has a graphical user interface (GUI) similar to Microsoft Windows which provides an easy to use environment.
Everything in UNIX is either a file or a process.
A *process* is an executing program identified by a unique PID (process identifier).
A file is a collection of data. They are created by users using text editors, running compilers etc.
All the files are grouped together in the directory structure.

3.1 The design of the unix operating system

Memory management Policies:

- Allocating swap space
- Freeing swap space
- Swapping
- Demand paging

MEMORY
Primary memory is a precious resource that frequently cannot contain all active processes in the system
The memory management system decides which processes should reside (at least partially) in main memory
It monitors the amount of available primary memory and may periodically write processes to a secondary device called the swap device to provide more space in primary memory

At a later time, the kernel reads the data from swap device back to main memory

UNIX Memory Management Policies

- Swapping
  - Easy to implement
  - Less system overhead
- Demand Paging
  - Greater flexibility
  - Swapping

The swap device is a block device in a configurable section of a disk

Kernel allocates contiguous space on the swap device without fragmentation

It maintains free space of the swap device in an in-core table, called map

The kernel treats each unit of the swap map as group of disk blocks

As kernel allocates and frees resources, it updates the map accordingly

Algorithm: Allocate Swap Space

```
• malloc( address_of_map, number_of_unit)
    – for (every map entry)
        • if (current map entry can fit requested units)
            – if (requested units == number of units in entry)
                » Delete entry from map
            – else
                » Adjust start address of entry
            – return original address of entry
    – return -1
```

Swapping Process Out

- Memory → Swap device
- Kernel swap out when it needs memory

1. When fork() called for allocate child process
2. When called for increase the size of process
3. When process become larger by growth of its stack
4. Previously swapped out process want to swap in but not enough memory

The kernel must gather the page addresses of data at primary memory to be swapped out Kernel copies the physical memory assigned to a process to the allocated space on the swap device.The mapping between physical memory and swap device is kept in page table entry Demand Paging Not all page of process resides in memory
When a process accesses a page that is not part of its working set, it incurs a page fault.

The kernel suspends the execution of the process until it reads the page into memory and makes it accessible to the process

3.3 INTERRUPT HANDLERS

[Macro]
system: with-enabled-interrupts *specs &rest body*

This macro should be called with a list of signal specifications, *specs*. Each element of *specs* should be a list of two elements: the first should be the Unix signal for which a handler should be established, the second should be a function to be called when the signal is received One or more signal handlers can be established in this way. with-enabled-interrupts establishes the correct signal handlers and then executes the forms in *body*. The forms are executed in an unwind-protect so that the state of the signal handlers will be restored to what it was before the with-enabled-interrupts was entered. A signal handler function specified as NIL will set the Unix signal handler to the default which is normally either to ignore the signal or to cause a core dump depending on the particular signal.

It is sometimes necessary to execute a piece a code that can not be interrupted. This macro the forms in *body* with interrupts disabled. Note that the Unix interrupts are not actually disabled, rather they are queued until after *body* has finished executing.

When executing an interrupt handler, the system disables interrupts, as if the handler was wrapped in a without-interrupts. The macro with-interrupts can be used to enable interrupts while the forms in *body* are evaluated. This is useful if *body* is going to enter a break loop or do some long computation that might need to be interrupt

For some interrupts, such as SIGTSTP (suspend the Lisp process and return to the Unix shell) it is necessary to leave Hemlock and then return to it. This macro executes the forms in *body* after exiting Hemlock. When *body* has been executed, control is returned to Hemlock.

[Function]

This function establishes *function* as the handler for *signal*.

Unless you want to establish a global signal handler, you should use the macro with-enabled-interrupts to temporarily establish a signal handler. enable-interrupt returns the old function associated with the signal.

[Function]
system: ignore-interrupt *signal*

Ignore-interrupt sets the Unix signal mechanism to ignore *signal* which means that the Lisp process will never see the signal. Ignore-interrupt returns the old function associated with the signal or *nil* if none is currently defined.

Default-interrupt can be used to tell the Unix signal mechanism to perform the default action for *signal*.

3.4 UNIX PROCESS SCHEDULING

There is the need for processes on a system to occasionally request services from the kernel. Some older operating systems had a *rendezvous* style of providing these services - the process would request a service and wait at a particular point, until a kernel task came along and serviced the request on behalf of the process.

UNIX works very differently. Rather than having kernel tasks service the requests of a process, the process itself enters *kernel space*. This means that rather than the process waiting "outside" the kernel; it enters the kernel itself (i.e. the process will start executing kernel code for itself).

When a process invokes a system call, the hardware is switched to the kernel settings. At this point, the process will be executing code from the kernel image.

The Kernel in UNIX

- Controls the execution of processes by allowing their creation, termination, communication.
- Schedules processes fairly for execution on CPU
- Allocates main memory for an executing process
- Allocates secondary memory for efficient storage and retrieval of user data
- Allows controlled peripheral device access to processes

3.4.1 Basic operations on processes in UNIX

Creation of processes in UNIX

- ➢ Establish a new process
- ➢ Assign a new unique process identifier (PID) to the new process

- Allocate memory to the process for all elements of process image, including private user address space and stack; the values can possibly come from the parent process; set up any linkages, and then, allocate space for process control block

- Create a new process control block corresponding to the above PID and add it to the process table; initialize di erent values in there such as parent PID, list of children (initialized to null), program counter (set to program entry point), system stack pointer (set to de ne the process stack boundaries)

- Initial CPU state, typically initialized to Ready or Ready, suspend Add the process id of new process to the list of children of the creating (parent) process

- r0. Initial allocation of resources

- k0. Initial priority of the process

- Accounting information and limits

- Add the process to the ready list

- Initial allocation of memory and resources must be a subset of parent's and be assigned as shared Initial priority of the process can be greater than the parent's

Management of processes in UNIX

How processes are managed after creation in UNIX

1. Suspend - Change process state to suspended
    - A process may suspend only its descendants
    - May include cascaded suspension
    - Stop the process if the process is in running state and save the state of the processor in the process control block

- If process is already in blocked state, then leave it blocked, else change its state to ready state
- If need be, call the scheduler to schedule the processor to some other process

2. Activate - Change process state to active
   - Change one of the descendant processes to ready state
   - Add the process to the ready list
3. Destroy - Remove one or more processes
   - Cascaded destruction
   - Only descendant processes may be destroyed
   - If the process to be “killed" is running, stop its execution
   - Free all the resources currently allocated to the process
   - Remove the process control block associated with the killed process
4. Change priority - Set a new priority for the process
   - Change the priority in the process control block
   - Move the process to a different queue to reflect the new priority

3.4.2 Scheduling in UNIX

Scheduler decides the process to run first by using a scheduling algorithm

3.4.2.1 Type of scheduling used in UNIX

Pre-emptibility

In UNIX, Processes in user space are *pre-emptible* - what this means is that a process may have the CPU taken away from it arbitrarily. This is how pre-emptive multitasking works: the scheduling routine will periodically suspend the currently executing process, and possibly schedule another task to run on that CPU. This means that theoretically, a process can be in a situation where it never gets the CPU back. In reality the scheduling code has an interest in fairness and will try to give the CPU to each process with a weak level of fairness, but there are no guarantees

Algorithms are:

- Shortest Remaining Time Scheduling
  - Preemptive version of shortest job next scheduling
  - Preemptive in nature (only at arrival time)
  - Highest priority to process that need least time to complete
  - Priority function P
  - Schedule for execution
  - Average waiting time calculations
- Round-Robin Scheduling
  - Preemptive in nature
  - Preemption based on time slices or time quanta
  - Time quantum between 10 and 100 milliseconds
  - All user processes treated to be at the same priority
  - Ready queue treated as a circular queue

Desirable features of a scheduling algorithm

1. Fairness: Make sure each process gets its fair share of the CPU
2. Efficiency: Keep the CPU busy 100% of the time
3. Response time: Minimize response time for interactive users
4. Turnaround: Minimize the time batch users must wait for output
5. Throughput: Maximize the number of jobs processed per hour

3.5 DEVICE MANAGEMENT

To perform useful functions, processes need access to the peripherals connected to the computer, which are controlled by the kernel through device drivers. For example, to show the user something on the screen, an application would make a request to the kernel, which would forward the request to its display driver, which is then responsible for actually plotting the character/pixel.

3.5.1 Special features of Device management in UNIX

Device drivers run as part of the kernel, either compiled in or as run-time loadable modules. The kernel architectures, Monolithic kernel does this and it have the advantage of speed and efficiency.

- **Device manager**

Device manager will be the interface between the device drivers and the both the rest of the kernel and user applications.

The device manager needs to do two things:

1. Isolate devices drivers from the kernel so that driver writers can worry about interfacing to the hardware and not about interfacing to the kernel
2. Isolate user applications from the hardware so that applications can work on the majority of devices the user might connect to their system

In most operating systems, the device manager is the only part of the kernel that programmers really see. Writing a good interface will make the difference between an efficient and reliable OS which works with a variety of devices and an OS which you spend all your own time writing and debugger drivers for.

Capabilities of device manager

1. Asynchronous I/O: that is, applications will be able to start an I/O operation and continue to run until it terminates.
2. Plug and Play: drivers will be able to be loaded and unloaded as devices are added to and removed from the system. Devices will be detected automatically on system startup, if possible.

- **Drivers**

Because we want our kernel to be plug-and-play capable, it isn't enough for drivers to be added to the kernel at compile time, as Minix and old Linux do. We must be able to load and unload them at run time. This isn't difficult: it just means we have to extend the executable file interface to kernel mode.

- **Interfaces**

Once we've detected the devices installed in the system we need to keep a record of them somewhere. The standard Unix model, employed by Minix and Linux, is to keep directory somewhere in the file system. This directory is filled with special directory entries, directory entries which don't point to any data, each of which refers to a specific device via major and minor device numbers. The major device number specifies the device type or driver to use and the minor number specifies a particular device implemented by that drivers.

3.6 Security

An important kernel design decision is the choice of the abstraction levels where the security mechanisms and policies should be implemented. Kernel security mechanisms play a critical role in supporting security at higher levels.

One approach is to use firmware and kernel support for fault tolerance (see above), and build the security policy for malicious behavior on top of that (adding features such as cryptography mechanisms where necessary), delegating some responsibility to the compiler. Approaches that delegate enforcement of security policy to the compiler and/or the application level are often called *language-based security*.

The lack of many critical security mechanisms in current mainstream operating systems impedes the implementation of adequate security policies at the application abstraction level. In fact, a common misconception in computer security is that any security policy can be implemented in an application regardless of kernel support.

4.0 ADVANTAGES OF UNIX O/S

- Unix is more flexible and can be installed on many different types of machines, including main-frame computers, supercomputers and micro-computers.

- Unix is more stable and does not go down as often as Windows does, therefore requires less administration and maintenance.

- Unix has greater built-in security and permissions features than Windows.

- Unix possesses much greater processing power than Windows.

- Unix is the leader in serving the Web. About 90% of the Internet relies on Unix operating systems running Apache, the world's most widely used Web server.

- Software upgrades from Microsoft often require the user to purchase new or more hardware or prerequisite software. That is not the case with Unix.

- The mostly free or inexpensive open-source operating systems, such as Linux and BSD, with their flexibility and control, are very attractive to (aspiring) computer wizards. Many of the smartest programmers are developing state-of-the-art software free of charge for the fast growing "open-source movement".

- Unix also inspires novel approaches to software design, such as solving problems by interconnecting simpler tools instead of creating large monolithic application programs.

CHAPTER TWO - LINUX

Introduction

What is Linux?

Linux is a UNIX-like operating system that runs on many different computers. Linux was first released in 1991 by its author Linus Torvalds at the University of Helsinki and developed by Linus Torvalds (author) and Andrew Morton. Linux is the operating system *kernel*, which comes with a *distribution* of software The Linux kernel is an operating system kernel used by a family of Unix-like operating system. It started out as a personal computer system used by individuals, and has since gained the support of several large operations such as HP, IBM, and Sun microsystem. It now used mostly as the server operating system. It's a prime example of open source development system. It's written in C

Since then it has grown tremendously in popularity as programmers around the world embraced his project of building a free operating system, adding features, and fixing problems. Linux is portable, which means you'll find versions running on name-brand or clone PCs, Apple Macintoshes, Sun workstations, or Digital Equipment Corporation Alpha-based computers. Linux also comes with source code, so you can change or customize the software to adapt to your needs. Finally, Linux is a great operating system, rich in features adopted from other versions of UNIX. The term Linux distribution is used to refer to the various operating systems that run on top of the Linux kernel. Linux is one of the most prominent examples of free/open source software. Today, the Linux kernel has received contributions from thousands of programmers.

Event Leading To the Creation

The UNIX operating system was conceived and implemented in 1960 and first released in 1970. Its portability and availability caused it to the widely adopted and modified by academic institutions and businesses. In 1983, Richard Stallman started the GNU project with the goal of creating a free UNIX like operating system. As part of the work, he wrote the GNU general public license (GPL). By the early 1990's there was almost enough available software to create a full operating system. However, the GNU kernel called HURD, failed to attract attention from developers leaving GNU incomplete. A solution seemed to appear in form of MINIX. It was released by Andrew S Tanenbaum in 1987, as an operating system, MINIX was not a superb one while source code was available, modification and retribution was restricted. This factors and lack of widely adopted free kernel made Torvalds start is project.

Processes

The concept of a *process* is fundamental to any multiprogramming operating system. A process is usually defined as an instance of a program in execution; thus, if 16 users are running vi at once, there are 16 separate processes (although they can share the same executable code). Processes are often called "tasks" in Linux source code.

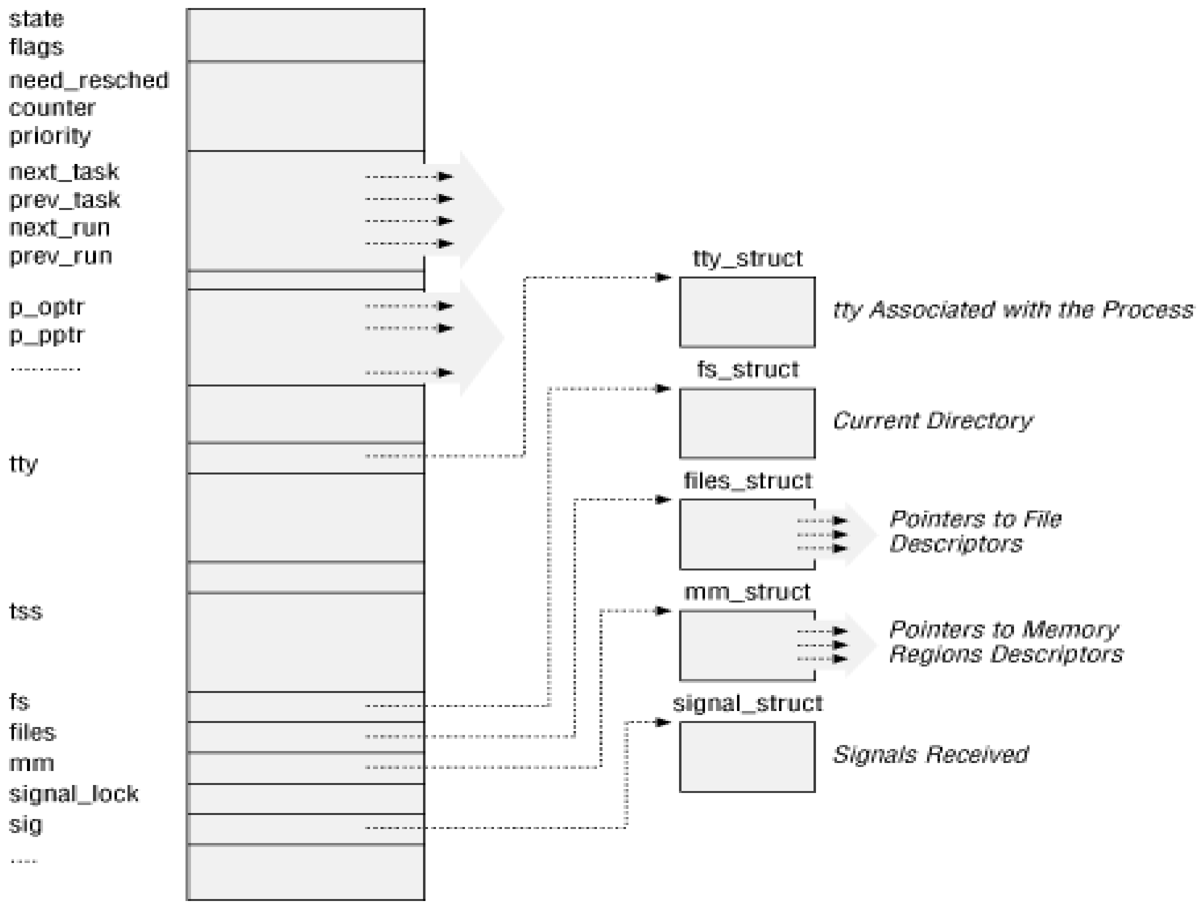
Properties of processes

- Static
- Dynamic

Process Descriptor

In order to manage processes, the kernel must have a clear picture of what each process is doing. It must know, for instance, the process's priority, whether it is running on the CPU or blocked on some event, what address space has been assigned to it, which files it is allowed to address, and so on. This is the role of the *process descriptor,* that is, of a task_struct type structure whose fields contain all the information related to a single process. As the repository of so much information, the process descriptor is rather complex. Not only does it contain many fields itself, but some contain pointers to other data structures that, in turn, contain pointers to other structures. The figure below describes the Linux process
descriptor schematically.

Figure 1 The Linux Process Descriptor



The five data structures on the right side of the figure refer to specific resources owned by the process. These resources will be covered in future chapters. This chapter will focus on two types of fields that refer to the process state and to process parent/child relationships.

Process State

As its name implies, the 'state' field of the process descriptor describes what is currently happening to the process. It consists of an array of flags, each of which describes a possible process state. In the current Linux version these states are mutually exclusive, and hence exactly one flag of state is set; the remaining flags are cleared. The following are thepossible process states:

TASK_RUNNING

The process is either executing on the CPU or waiting to be executed.

TASK_INTERRUPTIBLE

The process is suspended (sleeping) until some condition becomes true. Raising a
hardware interrupt, releasing a system resource the process is waiting for, or delivering a signal are examples of conditions that might wake up the process, that is, put its state back to TASK_RUNNING.

TASK_UNINTERRUPTIBLE

Like the previous state, except that delivering a signal to the sleeping process leaves its state unchanged. This process state is seldom used. It is valuable, however, under certain specific conditions in which a process must wait until a given event occurs without being interrupted. For instance, this state may be used when a process opens a device file and the corresponding device driver starts probing for a corresponding hardware device. The device driver must not be interrupted until the probing is complete, or the hardware device could be left in an unpredictable state.

TASK_STOPPED

Process execution has been stopped: the process enters this state after receiving a SIGSTOP, SIGTSTP, SIGTTIN, or SIGTTOU signal. When a process is being monitored by another (such as when a debugger executes a ptrace( ) system call to monitor a test program), any signal may put the process in the TASK_STOPPED state.

TASK_ZOMBIE

Process execution is terminated, but the parent process has not yet issued a wait( )- like system call (wait2( ), wait3( ), wait4( ), or waitpid( )) to return information about the dead process. Before the wait( )-like call is issued, the kernel cannot discard the data contained in the dead process descriptor because the parent could need it

Identifying A Process

Any Unix-like operating system, on the other hand, allows users to identify processes by means of a number called the *Process ID* (or *PID*). The PID is a 32-bit unsigned integer stored in the PID field of the process descriptor. PIDs are numbered sequentially: the PID of a newly created process is normally the PID of the previously created process incremented by one. However, for compatibility with traditional Unix systems developed for 16-bit hardware platforms, the maximum PID number allowed on Linux is 32767. When the kernel creates the 32768th process in the system, it must start recycling the lower unused PIDs.

Memory Management

The memory management subsystem is one of the most important parts of the operating system. Since the early days of computing, there has been a need for more memory than exists physically in a system. Strategies have been developed to overcome this limitation and the most successful of these is virtual memory. Virtual memory makes the system appear to have more memory than is physically present by sharing it among competing processes as they need it. Virtual memory does more than just make your computer's memory go farther. The memory management subsystem provides:

Large Address Spaces

The operating system makes the system appear as if it has a larger amount of memory than it actually has. The virtual memory can be many times larger than the physical memory in the system.

Protection

Each process in the system has its own virtual address space. These virtual address spaces are completely separate from each other and so a process running one application cannot affect another. Also, the hardware virtual memory mechanisms allow areas of memory to be protected against writing. This protects code and data from being overwritten by rogue applications.

Memory Mapping

Memory mapping is used to map image and data files into a process' address space. In memory mapping, the contents of a file are linked directly into the virtual address space of a process.

Fair Physical Memory Allocation

The memory management subsystem allows each running process in the system a fair share of the physical memory of the system.

Shared Virtual Memory

Although virtual memory allows processes to have separate (virtual) address spaces, there are times when you need processes to share memory. For example there could be several processes in the system running the bash command shell. Rather than have several copies of bash, one in each process's virtual address space, it is better to have only one copy in physical memory and all of the processes running bash share it. Dynamic libraries are another common example of executing code shared between several processes.

Shared memory can also be used as an Inter Process Communication (IPC) mechanism, with two or more processes exchanging information via memory common to all of them. Linux supports the Unix System V shared memory IPC.

3.1 An Abstract Model of Virtual Memory

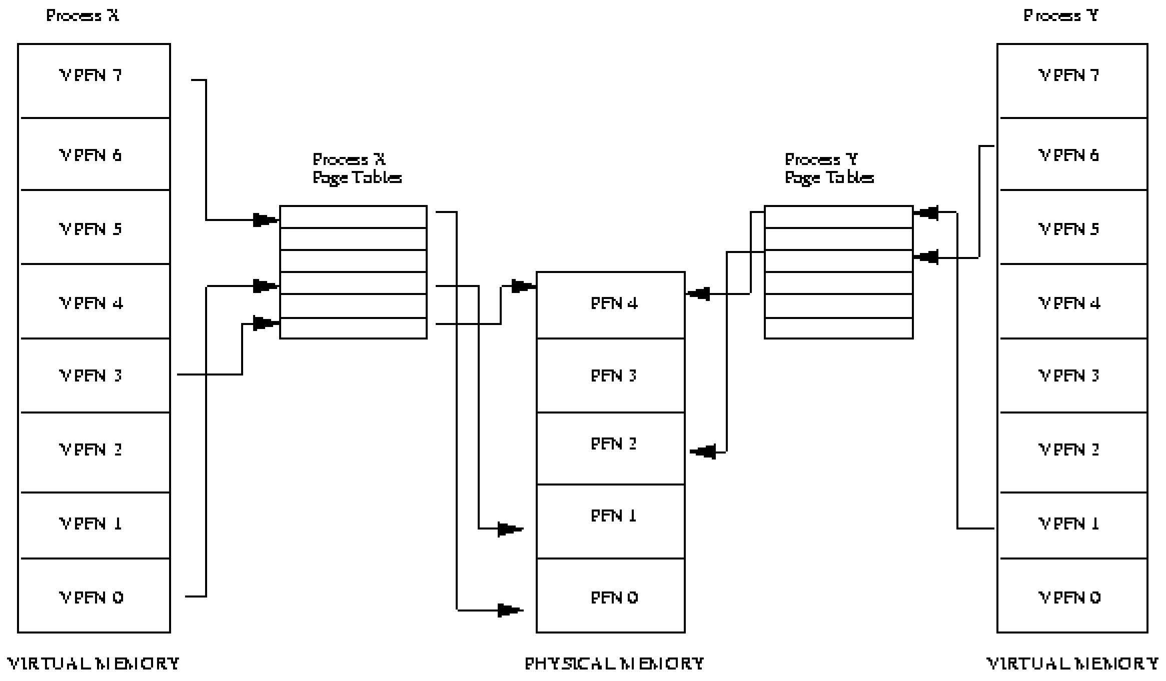


Figure 3.1: Abstract model of Virtual to Physical address mapping

Before considering the methods that Linux uses to support virtual memory it is useful to consider an abstract model that is not cluttered by too much detail.

As the processor executes a program it reads an instruction from memory and decodes it. In decoding the instruction it may need to fetch or store the contents of a location in memory. The processor then executes the instruction and moves onto the next instruction in the program. In this way the processor is always accessing memory either to fetch instructions or to fetch and store data.

In a virtual memory system all of these addresses are virtual addresses and not physical addresses. These virtual addresses are converted into physical addresses by the processor based on information held in a set of tables maintained by the operating system.

To make this translation easier, virtual and physical memory are divided into handy sized chunks called *pages*. These pages are all the same size, they need not be but if they were not, the system would be very hard to administer. Linux on Alpha AXP systems uses 8 Kbyte pages and on Intel x86 systems it uses 4 Kbyte pages. Each of these pages is given a unique number; the page frame number (PFN).

In this paged model, a virtual address is composed of two parts; an offset and a virtual page frame number. If the page size is 4 Kbytes, bits 11:0 of the virtual address contain the offset and bits 12 and above are the virtual page frame number. Each time the processor encounters a virtual address it must extract the offset and the virtual page frame number. The processor must translate the virtual page frame number into a physical one and then access the location at the correct offset into that physical page. To do this the processor uses *page tables*.

Interrupts And Exceptions

An *interrupt* is usually defined as an event that alters the sequence of instructions executed by a processor. Such events correspond to electrical signals generated by hardware circuits both inside and outside of the CPU chip.
Interrupts are often divided into *synchronous* and *asynchronous* interrupts:

• *Synchronous* interrupts are produced by the CPU control unit while executing
instructions and are called synchronous because the control unit issues them only after terminating the execution of an instruction.

• *Asynchronous* interrupts are generated by other hardware devices at arbitrary times with respect to the CPU clock signals. Intel 80x86 microprocessor manuals designate synchronous and asynchronous interrupts as *exceptions* and *interrupts*, respectively. We'll adopt this classification, although we'll
occasionally use the term "interrupt signal" to designate both types together (synchronous as well as asynchronous). Interrupts are issued by interval timers and I/O devices; for instance, the arrival of a keystroke from a user sets off an interrupt. Exceptions, on the other hand, are caused either by programming errors or by anomalous conditions that must be handled by the kernel. In

the first case, the kernel handles the exception by delivering to the current process one of the signals familiar to every Unix programmer. In the second case, the kernel performs all the steps needed to recover from the anomalous condition, such as a page fault or a request (via an int instruction) for a kernel service.

The Role of Interrupt Signals

As the name suggests, interrupt signals provide a way to divert the processor to code outside the normal flow of control. When an interrupt signal arrives, the CPU must stop what it's currently doing and switch to a new activity; it does this by saving the current value of the program counter (i.e., the content of the eip and cs registers) in the Kernel Mode stack and by placing an address related to the interrupt type into the program counter. There is a key difference between interrupt handling and process switching: the code executed by an interrupt or by an exception handler is not a process. Rather, it is a kernel control path that runs on behalf of the same process that was running when the interrupt occurred. As a kernel control path, the interrupt handler is lighter than a process (it has less context and requires less time to set up or tear down).

Interrupt handling is one of the most sensitive tasks performed by the kernel, since it must satisfy the following constraints:

• Interrupts can come at any time, when the kernel may want to finish something else it was trying to do. The kernel's goal is therefore to get the interrupt out of the way as soon as possible and defer as much processing as it can. For instance, suppose a block of data has arrived on a network line. When the hardware interrupts the kernel, it could simply mark the presence of data, give the processor back to whatever was running before, and do the rest of the processing later (like moving the data into a buffer where its recipient process can find it and restarting the process). The activities that the kernel needs to perform in response to an interrupt are thus divided into two parts: a *top half* that the kernel executes right away and a *bottom half* that is left for later. The kernel keeps a queue pointing to all the functions that represent bottom halves waiting to be executed and pulls them off the queue to execute them at particular points in processing.

• Since interrupts can come at any time, the kernel might be handling one of them while another one (of a different type) occurs. This should be allowed as much as possible since it keeps the I/O devices busy. As a result, the interrupt handlers must be coded so that the corresponding kernel control paths can be executed in a nested manner. When the last kernel control path terminates, the kernel must be able to resume execution of the interrupted process or switch to another process if the interrupt signal has caused a rescheduling activity.

• Although the kernel may accept a new interrupt signal while handling a previous one, some critical regions exist inside the kernel code where interrupts must be disabled. Such critical regions must be limited as much as possible since, according to the previous requirement, the kernel, and in particular the interrupt handlers, should run most of the time with the interrupts enabled.

Interrupts and Exceptions

The Intel documentation classifies interrupts and exceptions as follows:
• Interrupts:

Maskable interrupts

Sent to the INTR pin of the microprocessor. They can be disabled by clearing the IF flag of the eflags register. All IRQs issued by I/O devices give rise to maskable
interrupts.

Nonmaskable interrupts

Sent to the NMI (Nonmaskable Interrupts) pin of the microprocessor. They are not
disabled by clearing the IF flag. Only a few critical events, such as hardware failures,
give rise to nonmaskable interrupts.

• Exceptions:

Processor-detected exceptions

Generated when the CPU detects an anomalous condition while executing an instruction. These are further divided into three groups, depending on the value of the eip register that is saved on the Kernel Mode stack when the CPU control unit raises the exception:

Faults

The saved value of eip is the address of the instruction that caused the fault, and hence that instruction can be resumed when the exception handler terminates. Resuming the same instruction is necessary whenever the handler is able to correct the anomalous condition that caused the exception.

Traps

The saved value of eip is the address of the instruction that should be executed after the one that caused the trap. A trap is triggered only when there is no need to re-execute the instruction that was terminated. The main use of traps is for debugging purposes: the role of the interrupt signal in this case is to notify the debugger that a specific instruction has been executed (for instance, a breakpoint has been reached within a program). Once the user has examined the data provided by the debugger, she may ask that execution of the debugged program resume starting from the next instruction.

Aborts

A serious error occurred; the control unit is in trouble, and it may be unable to store a meaningful value in the eip register. Aborts are caused by hardware failures or by invalid values in system tables. The interrupt signal sent by the control unit is an emergency signal used to switch control to the corresponding abort exception handler. This handler has no choice but to force the affected process to terminate.

Programmed exceptions

Occur at the request of the programmer. They are triggered by int or int3 instructions; the ‘into’ (check for overflow) and ’bound’ (check on address bound) instructions also give rise to a programmed exception when the condition they are checking is not true. Programmed exceptions are handled by the control unit as traps; they are often called *software interrupts*. Such exceptions have two common uses: to implement system calls, and to notify a debugger of a specific event.
Linux uses two types of descriptors:

Interrupt gates & trap gates.
Trap gate: Trap gates are used for activating exception handlers.

Interrupt gate: Cannot be accessed by user mode progs

The Linux Booting Process

In most cases, the Linux kernel is loaded from a hard disk, and a two-stage boot loader is required. The most commonly used Linux boot loader on Intel systems is named LILO (Linux Loader); corresponding programs exist for other architectures. LILO may be installed either on the MBR, replacing the small program that loads the boot sector of the active partition, or in the boot sector of a (usually active) disk partition. In both cases, the final result is the same: when the loader is executed at boot time, the user may choose which operating system to load. The LILO boot loader is broken into two parts, since otherwise it would be too large to fit into
the MBR. The MBR or the partition boot sector includes a small boot loader, which is loaded into RAM starting from address 0x00007c00 by the BIOS. This small program moves itself to the address 0x0009a000, sets up the Real Mode stack (ranging from 0x0009b000 to 0x0009a200), and loads the second part of the LILO boot loader into RAM starting from address 0x0009b000. In turn, this latter program reads a map of available operating systems from disk

and offers the user a prompt so she can choose one of them. Finally, after the user has chosen the kernel to be loaded (or let a time-out elapse so that LILO chooses a default), the boot loader may either copy the boot sector of the corresponding partition into RAM and execute it or directly copy the kernel image into RAM. Assuming that a Linux kernel image must be booted, the LILO boot loader, which relies on BIOS routines, performs essentially the same operations as the boot loader integrated into the kernel image described in the previous section about floppy disks. The loader displays the "Loading Linux" message; then it copies the integrated boot loader of the kernel image to address 0x00090000, the setup( ) code to address 0x00090200, and the rest of the kernel image to address 0x00010000 or 0x00100000. Then it jumps to the setup( ) code.

<u>The setup( ) functions</u>

1. Invokes a BIOS procedure to find out the amount of RAM available in the system.

2. Sets the keyboard repeat delay and rate. (When the user keeps a key pressed past a certain amount of time, the keyboard device sends the corresponding keycode over and over to the CPU.)

3. Initializes the video adapter card.

4. Reinitializes the disk controller and determines the hard disk parameters.

5. Checks for an IBM Micro Channel bus (MCA).

6. Checks for a PS/2 pointing device (bus mouse).

7. Checks for Advanced Power Management (APM) BIOS support.

8. If the kernel image was loaded low in RAM (at physical address 0x00010000), moves it to physical address 0x00001000. Conversely, if the kernel image was loaded high in RAM, does not move it. This step is necessary because, in order to be able to store the kernel image on a floppy disk and to save time while booting, the kernel image stored on disk is compressed, and

the decompression routine needs some free space to use as a temporary buffer following the kernel image in RAM.

9. Sets up a provisional Interrupt Descriptor Table (IDT) and a provisional Global
Descriptor Table (GDT).

10. Resets the floating point unit (FPU), if any.

11. Reprograms the Programmable Interrupt Controller (PIC) and maps the 16 hardware interrupts (IRQ lines) to the range of vectors from 32 to 47. The kernel must perform this step because the BIOS erroneously maps the hardware interrupts in the range from to 15, which is already used for CPU exceptions (see Section 4.2.3 in Chapter 4).

12. Switches the CPU from Real Mode to Protected Mode by setting the PE bit in the cr0 status register. The provisional kernel page tables contained in swapper_pg_dir and pg0 identically map the linear addresses to the same physical addresses. Therefore, the transition from Real Mode to Protected Mode goes smoothly.

13. Jumps to the startup_32( ) assembly language function.

The startup_32( ) Functions

There are two different startup_32( ) functions; the one we refer to here is coded in the *arch/i386/boot/compressed/head.S* file. After setup( ) terminates, the function has been moved either to physical address 0x00100000 or to physical address 0x00001000, depending on whether the kernel image was loaded high or low in RAM.

This function performs the following operations:

1. Initializes the segmentation registers and a provisional stack.

2. Fills the area of uninitialized data of the kernel identified by the _edata and _end symbols with zeros.

3. Invokes the decompress_kernel( ) function to decompress the kernel image. The "Uncompressing Linux . . . " message is displayed first. After the kernel image has been decompressed, the "O K, booting the kernel." message is shown. If the kernel image was loaded low, the decompressed kernel is placed at physical address 0x00100000. Otherwise, if the kernel image was loaded high, the decompressed kernel is placed in a temporary buffer located after the compressed image. The decompressed image is then moved into its final position, which starts at physical address 0x00100000.

4. Jumps to physical address 0x00100000. The decompressed kernel image begins with another startup_32( ) function included in the *arch/i386/kernel/head.S* file. Using the same name for both the functions does not create any problems (besides confusing our readers), since both functions are executed by jumping to their initial physical addresses.

The second startup_32( ) function essentially sets up the execution environment for the first Linux process (process 0). The function performs the following operations:

1. Initializes the segmentation registers with their final values.

2. Sets up the Kernel Mode stack for process.

3. Invokes setup_idt( ) to fill the IDT with null interrupt handlers.

4. Puts the system parameters obtained from the BIOS and the parameters passed to the operating system into the first page frame.

5. Identifies the model of the processor.

6. Loads the gdtr and idtr registers with the addresses of the GDT and IDT tables.

7. Jumps to the start_kernel( ) function.

A.5 Modern Age: The start_kernel( ) Function

The start_kernel( ) function completes the initialization of the Linux kernel. Nearly every kernel component is initialized by this function; we mention just a few of them:

• The page tables are initialized by invoking the paging_init( ) function.
• The page descriptors are initialized by the mem_init( ) function
• The final initialization of the IDT is performed by invoking trap_init( ) and init_IRQ( ).
• The slab allocator is initialized by the kmem_cache_init( ) and
kmem_cache_sizes_init( ) functions.
• The system date and time are initialized by the time_init( ) function (see
• The kernel thread for process 1 is created by invoking the kernel_thread( ) function. In turn, this kernel thread creates the other kernel threads and executes the */sbin/init* program.

Device Management(Managing I/O Devices)

The aim of this section is to illustrate the overall organization of device drivers in Linux.

I/O ARCHITECTURE

In order to make a computer work properly, data paths must be provided that let information flow between CPU(s), RAM, and the score of I/O devices that can be connected nowadays to a personal computer. These data paths, which are denoted collectively as the *bus*, act as the primary communication channel inside the computer. Several types of buses, such as the ISA, EISA, PCI, and MCA, are currently in use. In this section we'll discuss the functional characteristics common to all PC architectures, without giving details about a specific bus type.

In fact, what is commonly denoted as bus consists of three specialized buses:

Data bus

A group of lines that transfers data in parallel. The Pentium has a 64-bit-wide data bus.

Address bus

A group of lines that transmits an address in parallel. The Pentium has a 32-bit-wide address bus.

Control bus

A group of lines that transmits control information to the connected circuits. The
Pentium makes use of control lines to specify, for instance, whether the bus is used to allow data transfers between a processor and the RAM or alternatively between a processor and an I/O device. Control lines also determine whether a read or a write transfer must be performed. When the bus connects the CPU to an I/O device, it is called an *I/O bus*. In this case, Intel 80x86 microprocessors use 16 out of the 32 address lines to address I/O devices and 8, 16, or 32 out of the 64 data lines to transfer data. The I/O bus, in turn, is connected to each I/O Understanding the Linux Kernel 344 device by means of a hierarchy of hardware components including up to three elements: I/O ports, interfaces, and device controllers. architecture.

I/O Ports

Each device connected to the I/O bus has its own set of I/O addresses, which are usually called *I/O ports*. In the IBM PC architecture, the I/O address space provides up to 65,536 8-bit
I/O ports. Two consecutive 8-bit ports may be regarded as a single 16-bit port, which must start on an even address. Similarly, two consecutive 16-bit ports may be regarded as a single 32-bit port, which must start on an address that is a multiple of 4. Four special assembly language instructions called in, ins, out, and outs allow the CPU to read from and write into an I/O port.

While executing one of these instructions, the CPU makes use of the address bus to select the required I/O port and of the data bus to transfer data between a CPU register and the port. I/O ports may also be mapped into addresses of the physical address space: the processor is
then able to communicate with an I/O device by issuing assembly language instructions that operate directly on memory (for instance, mov, and, or, and so on). Modern hardware devices tend to prefer mapped I/O, since it is faster and can be combined with DMA.

An important objective for system designers is to offer a unified approach to I/O
programming without sacrificing performance. Toward that end, the I/O ports of each device are structured into a set of specialized registers. The CPU writes into
the *control register* the commands to be sent to the device and reads from the *status register* a value that represents the internal state of the device. The CPU also fetches data from the device by reading bytes from the *input register* and pushes data to the device by writing bytes into the *output register*.

Associating Files with I/O Devices

UNIX-like operating systems are based on the notion of a *file*, which is just an information container structured as a sequence of bytes. According to this approach, I/O devices are treated as files; thus, the same system calls used to interact with regular files on disk can be used to directly interact with I/O devices. As an example, the same write( ) system call may be used to write data into a regular file, or to send it to a printer by writing to the */dev/lp0* device file. Let's now examine in more detail how this schema is carried out.

Device Files

Device files are used to represent most of the I/O devices supported by Linux. Besides its name, each device file has three main attributes:

Type

Either *block* or *character*.

Major number

A number ranging from 1 to 255 that identifies the device type. Usually, all device
files having the same major number and the same type share the same set of file
operations, since they are handled by the same device driver.

Minor number

A number that identifies a specific device among a group of devices that share the
same major number. The mknod( ) system call is used to create device files. It receives the name of the device file, its type, and the major and minor numbers as parameters. The last two parameters are merged in a 16-bit dev_t number: the eight most significant bits identify the major number, while the remaining ones identify the minor number. The MAJOR and MINOR macros extract the two values from the 16-bit number, while the MKDEV macro merges a major and minor number into a 16-bit number. Actually, dev_t is the data type specifically used by application programs; the kernel uses the kdev_t data type. In Linux 2.2 both types reduce to an unsigned short integer, but kdev_t will become a complete device file descriptor in some future Linux version.
Device files are usually included in the */dev* directory. The following illustrates the attributes of some device files. Notice how the same major number may be used to identify both a character and a block device.

Name	**Type**	**Major**	**Minor**	**Description**
/dev/fd0	block	2	0	Floppy disk
/dev/hda	block	3	0	First IDE disk
/dev/hda2	block	3	2	Second primary partition of first IDE disk
/dev/hdb	block	3	64	Second IDE disk

/dev/hdb3	block	3	67	Third primary partition of second IDE disk
/dev/ttyp0	char	3	0	Terminal
/dev/console	char	5	1	Console
/dev/lp1	char	6	1	Parallel printer
/dev/ttyS0	char	4	64	First serial port
/dev/rtc	char	10	135	Real time clock
/dev/null	char	1	3	Null device (black hole)

Usually, a device file is associated with a hardware device, like a hard disk (for instance, */dev/hda*), or with some physical or logical portion of a hardware device, like a disk partition (for instance, */dev/hda2*). In some cases, however, a device file is not associated to any real hardware device, but represents a fictitious logical device. For instance, */dev/null* is a device file corresponding to a "black hole": all data written into it are simply discarded, and the file appears always empty. As far as the kernel is concerned, the name of the device file is irrelevant. If you created a device file named */tmp/disk* of type "block" with major number 3 and minor number 0, it would be equivalent to the */dev/hda* device file shown in the table. On the other hand, device filenames may be significant for some application programs. As an example, a communication program might assume that the first serial port is associated with the */dev/ttyS0* device file. But usually most application programs can be configured to interact with arbitrarily named device files.

File System Management

The Second Extended File system (Ext2) is native to Linux and is used on virtually every Linux system, Furthermore, Ext2 illustrates a lot of good practices in its support for modern file system features with fast performance.

General Characteristics Each Unix-like operating system makes use of its own file system. Although all such file systems comply with the POSIX interface, each of them is implemented in a different way.

The first versions of Linux were based on the Minix filesystem. As Linux matured, the *Extended Filesystem (Ext FS)* was introduced; it included several significant extensions but offered unsatisfactory performance. The *Second Extended Filesystem (Ext2)* was introduced in 1994: besides including several new features, it is quite efficient and robust and has become the most widely used Linux filesystem.

The following features contribute to the efficiency of Ext2:

• When creating an Ext2 filesystem, the system administrator may choose the optimal block size (from 1024 to 4096 bytes), depending on the expected average file length.

For instance, a 1024 block size is preferable when the average file length is smaller
than a few thousand bytes because this leads to less internal fragmentation—that is,
less of a mismatch between the file length and the portion of the disk that stores it. On the other hand, larger block sizes are usually preferable for files greater than a few thousand bytes because this leads to fewer disk transfers, thus reducing
system overhead.

• When creating an Ext2 filesystem, the system administrator may choose how many inodes to allow for a partition of a given size, depending on the expected number of files to be stored on it. This maximizes the effectively usable disk space.

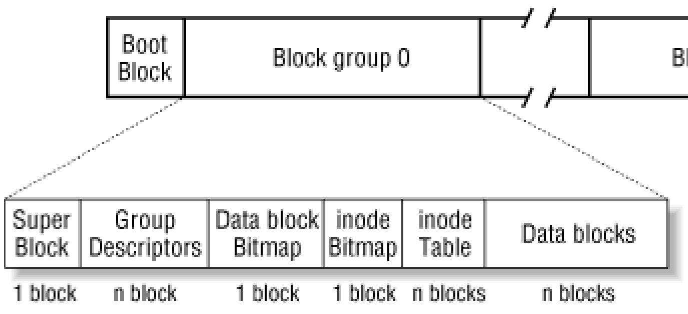
• The file system partitions disk blocks into groups. Each group includes data blocks and inodes stored in adjacent tracks. Thanks to this structure, files stored in a single block group can be accessed with a lower average disk seek time.

• The filesystem *preallocates* disk data blocks to regular files before they are actually used. Thus, when the file increases in size, several blocks are already reserved at physically adjacent positions, reducing file fragmentation.

• Fast symbolic links are supported. If the pathname of the symbolic link has 60 bytes or less, it is stored in the inode and can thus be translated without reading a data block.

Disk Data Structures

Figure 2 Layouts of an Ext2 partition and of an Ext2 block group



The first block in any Ext2 partition is never managed by the Ext2 filesystem, since it is reserved for the partition boot sector. The rest of the Ext2 partition is split into *block groups* , each of which has the layout shown in Figure 2. As you will notice from the figure, some data structures must fit in exactly one block while others may require more than one block. All the block groups in the filesystem have the same size and are stored sequentially, so the kernel can derive the location of a block group in a disk simply from its integer index. Block groups reduce file fragmentation, since the kernel tries to keep the data blocks belonging to a file in the same block group if possible. Each block in a block group contains one of the following pieces of information:

• A copy of the filesystem's superblock
• A copy of the group of block group descriptors
• A data block bitmap
• A group of inodes
• An inode bitmap

• A chunk of data belonging to a file; that is, a data block

If a block does not contain any meaningful information, it is said to be free.

Superblock

An Ext2 disk superblock is stored in an ext2_super_block structure. The __u8, __u16, and __u32 data types denote unsigned numbers of length 8,
16, and 32 bits respectively, while the __s8, __s16, __s32 data types denote signed numbers of length 8, 16, and 32 bits. The s_inodes_count field stores the number of inodes, while the s_blocks_count field stores the number of blocks in the Ext2 filesystem. The s_log_block_size field expresses the block size as a power of 2, using 1024 bytes as the unit. Thus, denotes 1024-byte blocks, 1 denotes 2048-byte blocks, and so on. These_log_frag_size field is currently equal to s_log_block_size, since block fragmentation is not yet implemented. The s_blocks_per_group, s_frags_per_group, and s_inodes_per_group fields store the
number of blocks, fragments, and inodes in each block group, respectively. Some disk blocks are reserved to the superuser (or to some other user or group of users
selected by the s_def_resuid and s_def_resgid fields). These blocks allow the system administrator to continue to use the filesystem even when no more free blocks are available for normal users. The s_mnt_count, s_max_mnt_count, s_lastcheck, and s_checkinterval fields set up the Ext2 filesystem to be checked automatically at boot time. These fields cause */sbin/e2fsck* to run after a predefined number of mount operations has been performed, or when a predefined amount of time has elapsed since the last consistency check. (Both kinds of checks can be
used together.) The consistency check is also enforced at boot time if the filesystem has not been cleanly unmounted (for instance, after a system crash) or when the kernel discovers some errors in it. The s_state field stores the value if the filesystem is mounted or was not cleanly unmounted, 1 if it was cleanly unmounted, and 2 if it contains errors.

Group Descriptor And Bitmap

Each block group has its own group descriptor, an ext2_group_desc structure

The bg_free_blocks_count, bg_free_inodes_count, and bg_used_dirs_count fields are used when allocating new inodes and data blocks. These fields determine the most suitable block in which to allocate each data structure. The bitmaps are sequences of bits, where the value specifies that the corresponding inode or data block is free and the value 1 specifies that it is used. Since each bitmap must be stored inside a single block and since the block size can be 1024, 2048, or 4096 bytes, a single bitmap describes the state of 8192, 16,384, or 32,768 blocks.

CHAPTER THREE – SOLARIS OPERATING SYSTEM

HISTORY

The history of Solaris, a Unix-based operating system developed by Sun Microsystems, displays that company's ability to be innovative and flexible. Solaris was introduced in the year 1987 out of an alliance between AT&T and Sun Microsystems to combine the leading Unix versions (BSD, XENIX, and System V) into one operating system.In 1991, Sun replaced it's existing Unix operating system (SunOS 4) with one based on SVR4. This new OS, Solaris 2, contained many new advances, including use of the Open Windows graphical user interface, NIS+, Open Network Computing (ONC) functionality, and was specially tuned for symmetric multiprocessing.

This kicked off Solaris' history of constant innovation, with new versions of Solaris being released almost annually over the next fifteen years. Sun was constantly striving to stay ahead of the curve, while at the same time adapting Solaris to the existing, constantly evolving wider computing world. The catalogue of innovations in the Solaris OS are too numerous to list here, but a few milestones are worth mentioning.

- Solar 2.5.1 in 1996 added CDE, the NFSv3 file system and NFS/TCP, expanded user and group IDs to 32 bits, and included support for the Macintosh PowerPC platform.
- Solaris 2.6 in 1997 introduced WebNFS file system, Kerberos 5 security encryption, and large file support to increase Solaris' internet performance.
- Solaris 2.7 in 1998 (renamed just Solaris 7) included many new advances, such as native support for file system meta-data logging (UFS logging). It was also the first 64-bit release, which dramatically increased its performance, capacity, and scalability.
- Solaris 8 in 2000 took it a step further was the first OS to combine datacenter and dot-com requirements, offering support for IPv6 and IPSEC, Multipath I/O, and IPMP.
- Solaris 9 in 2002 saw the writing on the wall of the server market, dropped OpenWindows in favour of Linux compatibility, and added a Resource Manager, the Solaris Volume Manager, extended file attributes, and the iPlanet Directory Server.

- Solaris 10, the current version, was released to the public in 2005 free of charge and with a host of new developments. The latest advances in the computing world are constantly being incorporated in new versions of Solaris 10 released every few months.

To mention just a few, Solaris features more and more compatibility with Linux and IBM systems, has introduced the Java Desktop System based on GNOME, added Dynamic Tracing (Dtrace), NFSv4, and later the ZFS file system in 2006.

Also in 2006, Sun set up the OpenSolaris Project. Within the first year, the OpenSolaris community had grown to 14,000 members with 29 user groups globally, working on 31 active projects. Although displaying a deep commitment to open-source ideals, it also provides Sun with thousands of developers essentially working for free.

SOLARIS PROCESSES

The process is one of the fundamental abstractions of Unix. Every object in Unix is represented as either a **file** or a **process**(with the introduction of the /proc structure, there has been an effort to represent even processes as files)**.** Processes are usually created with **fork**or a less resource alternative such as **fork1 or vfork.fork**duplicates the entire process context, while fork1 only duplicates the context of the calling thread. This can be useful for example, when **exec**will be called shortly.
Solaris like other UNIX systems, provide two modes of operation: **user mode** and **kernel (or system mode).** Kernel mode is a more privileged mode of operation. Processes can be executed in either mode, but user processes usually operate in user mode.

SOLARIS PROCESS SCHEDULING

In Solaris, highest priorities are scheduled first. Kernel thread scheduling information can be revealed with ps –elcL**.** A process can exist in one of the following states:

- Running
- Sleeping
- Ready

✓ **KERNEL THREADS MODEL**

The kernel threads model consist of the following objects:

- **Kernel threads –** this is what is scheduled/executed on a processor
- **User threads –** the user-level thread state within a process
- **Process -** the object that tracks the execution environment of a program
- **Lightweight process (lwp) –** Execution context for a user tread. It associates a user thread with a kernel thread.

In Solaris 10 kernel, kernel services and tasks are executed as kernel threads. When a user thread is created, the associated lwp and kernel threads are also created and linked to the user thread.

KERNEL THREADS MODEL

An application's parallelism is the degree of parallel execution achieved.This is limited by the number of processors available in the hardware configuration. Concurrency is the maximum achievable parallelism in a theoretical machine that has an unlimited number of processors.

Threads are frequently used to increase an application's concurrency. A thread represents a relatively independent set of instructions within a program. A thread is a control point within a process. It shares global resources within the context of the process (address space, open files, user credentials, quotas, etc). Threads also have private resources (program counter, stack, register context, etc).

The main benefit of threads (as compared to multiple processes) is that the context switches are much cheaper than those required to change current processes. Even within a single-processor environment, multiple threads are advantageous because one thread may be able to progress even though another thread is blocked while waiting for a resource. Inter-process communication also takes considerably less time for threads than for processes, since global data can be shared instantly.

The kernel threads model consist of the following objects:

- **Kernel threads –** this is what is scheduled/executed on a processor

- **User threads –** the user-level thread state within a process
- **Process -** the object that tracks the execution environment of a program
- **Lightweight process (lwp) –** Execution context for a user tread. It associates a user thread with a kernel thread.

In Solaris 10 kernel, kernel services and tasks are executed as kernel threads. When a user thread is created, the associated lwp and kernel threads are also created and linked to the user thread.

Kernel Threads

A kernel thread is the entity that is scheduled by the kernel. If no lightweight process is attached, it is also known as a system thread. It uses kernel text and global data, but has its own kernel stack, as well as a data structure to hold scheduling and synchronization information.

Kernel threads can be independently scheduled on CPUs. Context switching between kernel threads is very fast because memory mappings do not have to be flushed.

Lightweight Processes

A lightweight process can be considered as the swappable portion of a kernel thread.Another way to look at a lightweight process is to think of them as "virtual CPUs" which perform the processing for applications. Application threads are attached to available lightweight processes, which are attached to a kernel thread, which is scheduled on the system's CPU dispatch queue. LWPs can make system calls and can block while waiting for resources. All LWPs in a process share a common address space. IPC (inter-process communication) facilities exist for coordinating access to shared resources.

By default, one LWP is assigned to each process; additional LWPs are created if all the process's LWPs are sleeping and there are additional user threads that libthread can schedule. The programmer can specify that threads are bound to LWPs.

User Threads

User threads are scheduled on their LWPs via a scheduler in libthread. This scheduler does implement priorities, but does not implement time slicing. If time slicing is desired, it must be programmed in. Locking issues must also be carefully considered by the programmer in order to prevent several threads from blocking on a single resource.

Each thread has the following characteristics:

- Has its own stack.
- Shares the process address space.
- Executes independently (and perhaps concurrently with other threads).
- Completely invisible from outside the process.
- Cannot be controlled from the command line.
- No system protection between threads in a process; the programmer is responsible for interactions.
- Can share information between threads without IPC overhead.

✓ **PRIORITY MODEL**

The Solaris kernel is fully **preemptible.** This means that all threads, including the threads that support the kernel's own activities can be deferred to allow a higher-priority thread to run.

Solaris recognizes 170 different priorities, 0-169. Within these priorities fall a number of different scheduling classes:

- **TS (Timeshare):** This is the default class for processes and their associated kernel threads. Priorities falling within this class range 0-59 and are dynamically adjusted in an attempt to allocate processor resources evenly.
- **IA (Interactive):** This is an enhanced version of the TS class that applies to the in-focus window in the GUI. Its intent is to give extra resources to processes associated with that specific window. Like TS, IA's range is 0-59.

- **FSS (Fair-share scheduler):** This class is share-based rather than priority-based. Threads managed by FSS are scheduled based on their associated shares and the processor's utilization. FSS also has a range 0-59.
- **FX (Fixed-priority):** The priorities for threads associated with this class are fixed (in other words, they do not vary dynamically over the lifetime of the thread). FX also has a range 0-59.
- **SYS (system):** The SYS class is used to schedule kernel threads. Threads in this class are "bound" threads, which mean that they run until they block or complete. Priorities for SYS threads are in the 60-99 range.
- **RT (Real-time):** Threads in the RT class are fixed-priority, with a fixed time quantum. Their priorities range 100-159, so an RT thread will preempt a system thread. Of these, FSS and FX were implemented in Solaris 9.

Fair Share Scheduler

The default Timesharing (TS) scheduling class in Solaris attempts to allow each process on the system to have relatively equal CPU access. The nice command allows some management of process priority, but the new Fair Share Scheduler (FSS) allows more flexible process priority management that integrates with the project framework. Each project is allocated a certain number of CPU shares via the project. CPU-shares resource control and each project is allocated CPU time based on its CPU-shares value divided by the sum of the CPU-shares values for all active projects. Anything with a zero CPU-shares value will not be granted CPU time until all projects with non-zero CPU-shares are done with the CPU. The maximum number of shares that can be assigned to any one project is 65535.
FSS can be assigned to processor sets, resulting in more sensitive control of priorities on a server than raw processor sets.

The Fair Share Scheduler should not be combined with the TS, FX (fixed-priority) or IA (interactive) scheduling classes on the same CPU or processor set. All of these scheduling classes use priorities in the same range, so unexpected behavior can result from combining FSS with any of these. (There is no problem, however, with running TS and IA on the same processor set.)

Time Slicing for FSS

In FSS, the time quantum is the length of time that a thread is allowed to run before it has to release the processor. The QUANTUM is reported in ms. (The output of the above command displays the resolution in the RES parameter. The default is 1000 slices per second.

Fixed Priority Scheduling

FX scheduler sets policy scheduling for processes used by applications and users. These processes are fixed. For example, priocnt1 and dispadminare two utilities that control the Fixed-Priority Scheduling. The FX class is the same priority as the FSS, IA, and TS classes.

THE SOLARIS BOOTUP AND SHUTDOWN

The Solaris Boot process is made up of four phases and is illustrated in the figure below:

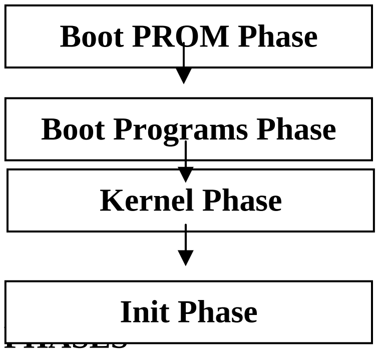


FIG. 1 SOLARIS BOOTUP PHASES

Boot PROM Phase: The hardware tests and initializes itself

Boot Programs Phase: The initial boot programs are loaded into the memory.

Kernel Phase: The kernel loads itself and its modules into memory and then unloads the boot programs from memory.

Init Phase: The init process is started by the kernel. The **init**process then executes the run control scripts.

Phase 1: The Boot PROM Phase

During this phase of the boot up, the system first powers up and checks itself. On the PROM chip is a program known as the monitor program. This program is used for initial system tests and diagnostics. It tests the system's memory, CPU and mother board. It does not test all devices attached to the server, only the server's main components.

If a third-party device is attached to anSBus controller, the device driver is then loaded from a firmware chip on the device (some manufacturers don't include device drivers on the hardware itself). If the open boot variable **diag-level** is set to **max** and the variable **diag-switch**is set to **true**the system will perform extensive diagnostics during the power on self test. The banner information looks like the figure below:

Sun blade 100 (UltraSPARC-IIe) Keyboard present OpenBoot 4.0, 128MB memory installed, Serial #50632835. Ethernet Address 0:3:ba:2:c2:3d, Host ID: 8323c12b.

FIG. 2 Output from the banner command.

After the power on self test is complete, the boot process stops at the **O.K** prompt or continues to boot the Solaris operating system. This depends on the value of the OpenBoot**auto-boot**? variable:

- If the auto-boot variable is set to true, the system boots the device specified in the **boot-device** variable. The default boot device OpenBoot value on most system is the **disk or disk:a.** A second boot device **(net)** can be also be specified. If for some reason the first boot device does not work, the second boot device is tried.
- If the **auto-boot?**variable is set to **false** the system stops at the **OK** prompt.

Phase 2: Boot Program Phase

This phase starts when the system has checked itself and starts to load the **bootblk** program from the boot device. The **bootblk**program is a smallsection of code on the first sector of the first track of the first drive of the hard drive or tape device. When bootblk runs, it shows a message like

Fcode UFS Reader 1.12 00/07/17 15:48:16

Bootblkhas only one function. It loads the**ufsboot** program into the memory and then dies. When the **Fcode UFS Reader …bootblk**has done its work. The following message should now appear:

Loading: /platform/SUNW,Sun-Blade-100/ufsboot
Loading :/platform/sun4u/ufsboot

The ufsboot program loads the kernel into memory. After the program is loaded into memory, the ufsboot program dies.

It is important that a system administrator understand what is happening with the **ufsboot**program and the **bootblk** program. If the system messages shown above do not appear, the server may be dead or something may be wrong with these two programs, which will then need to be reloaded or repaired.

Phase 3: Kernel Phase

This phase starts when the initial boot programs **bootblk** and **ufsboot** have been loaded and the kernel is now starting to load. The kernel can be thought as the core program that defines the Solaris operating. The kernel uses the **ufsboot**program to read kernel modules into memory. A kernel module can be thought of as a dynamic piece of software code. Only the modules that are needed are loaded into the kernel. This makes the kernel faster and more efficient than if it always had to load all its modules into memory. After enough modules are loaded into memory, the **ufsboot** program dies.

When the front slash symbol (/) starts to swirl, the kernel is starting to load. The SunOS Release is now also shown. This indicates that the boot device is booting and working. If there are any further problems with the boot process, they will most likely be caused by an error in a run control script.

Phase 4: The Init Phase

The init phase starts after the kernel has loaded itself and its modules into memory. The **sched**process is the first process to be loaded. It has a PID (Process Identification Number) of zero (0), as shown with the **ps–ef**command. The**sched** process is responsible for the scheduling policy and priority of processes. After **sched** starts up, the process called **init** is started, with a PID of one (1). The innit process reads a text file /etc/innittab. Among other things, this file defines the default run level and controls how the init process calls up and executes run control scripts.

MEMORY MANAGEMENT

- **The process Memory Usage**

  The /usr/proc/bin/pmap command is available in Solaris 2.6 and above. It can help pin down which process is memory hog. /usr/proc/bin/pmap –x PID prints out details of memory use by a process. Summary statistics regarding process size can be found in the RSS column of ps – ly or top. dbx, the debugging utility in the SunPro package, has extensive memory leak detection built in. The source code will need to be compiled with the –g flag by the appropriate SunPro compiler. Ipcs –mb shows memory statistics for shared memory. This may be useful when attempting to size memory to fit expected traffic.

- **Swap Space**

  The Solaris virtual memory system combines physical memory with available swap space via **swapfs**. If insufficient total virtual memory space is provided, new processes will be unable to open.
- **Paging**

  Solaris uses both common types of paging in its virtual memory system. These types are:
  - **Swapping**(swaps out all memory associated with a user process) and
  - **Download paging** (swaps out the not recently used pages)

  Which method is used is determined by comparing the amount of available memory with several key parameters
- **Solaris 8 Paging**

  Solaris 8 uses a different algorithm for removing from memory. This new architecture is known as the cyclical page cache. The cyclical page cache uses a file system free list to cache file system data only. Other memory objects are managed on a separate free list

SECURITY

File Integrity and Secure Execution

System administrators can detect possible attacks on their systems by monitoring for changes to file information. In the Solaris 10 OS, binaries are digitally signed, so administrators can track changes easily, and all patches or enhancements are embedded with digital signatures, eliminating the false positives associated with upgrading or patching file integrity-checking software.

User and Process Rights Management

In traditional UNIX platform-based operating systems, applications and users often need administrative access to perform their jobs. However, most implementations offer just one level of higher privilege: root or superuser. This means that any user or application given root access has the ability to make major changes to the operating system—and is typically the target of hacking attempts. The Solaris 10 OS offers unique User Rights Management (also known as role-based access control, or RBAC) and Process Rights Management (also known as privileges)

Network Service Protection

The Solaris 10 OS ships with Solaris IP Filter firewall software preinstalled. This integrated firewall can reduce the number of network services that are exposed to attack and provides protection against maliciously crafted networking packets. Starting in Solaris 10 8/07, the IP Filter firewall can also filter traffic flowing between Solaris Containers when it is configured in the Global Zone. In addition, TCP Wrappers are integrated into the Solaris 10 OS, limiting access to service-based allowed domains.

Cryptographic Services and Encrypted Communication

For high-performance, system-wide cryptographic routines, the Solaris Cryptographic Framework adds a standards-based, common API that provides a single point of administration and uniform access to both software and hardware-accelerated, cryptographic functions. The pluggable Solaris Cryptographic Framework can balance loads across accelerators, increasing encrypted network traffic throughput, and it is available to applications written to use Public Key Cryptography Standards (PKCS) #11, Sun Java Enterprise System, NSS, OpenSSL, and Java Cryptographic Extension software.

Flexible Enterprise Authentication

The Solaris 10 OS delivers a number of flexible authentication features. At the foundation of Solaris is support for Pluggable Authentication Mechanism (PAM), which make it possible to add authentication services to Solaris dynamically. Sun and third-party vendors provide many PAM modules and customers can create their own to meet specific security needs.

Repeatable Security Hardening and Monitoring

New features in the Solaris 10 OS make it easier than ever to minimize and harden a system. The Reduced Networking Metacluster install option creates a minimized Solaris OS image, ready for administrators to add functionality and services in direct support of their system's purpose.

Mandatory Access Control and Labeling

If your system requirements include privacy, increased accountability, and reduced risk of security violations, then Solaris Trusted Extensions is for you. A standard part of Solaris, true multi-level security is available for the first time in a commercial-grade operating system that runs all your existing applications and is supported on over 1,200 x64/x86 and SPARC platforms.

WEAKNESS AND STRENGHT

A security weakness in Solaris Trusted Extensions Policy configuration may allow a remote unprivileged user who has authorized or unauthorized access to the X server, to leverage an additional vulnerability which could lead to arbitrary code execution as a local privileged or unprivileged user.

Sun has acknowledged a weakness in Pidgin on Solaris, which can be exploited by malicious people to cause a DoS (Denial of Service).

CONCLUSION

The development of the Solaris OS demonstrates Sun Microsystems' ability to be on the cutting edge of the computing world without losing touch with the current computing environment. Sun regularly releases new versions of Solaris incorporating the latest development in computer technology, yet also included more cross-platform compatibility and incorporating the advances of other systems. The OpenSolaris project is the ultimate display of these twin strengths-Sun has tapped into the creative energy of developers across the world and receives instant feedback about what their audience wants and needs. If all software companies took a lesson from Sun, imagine how exciting and responsive the industry could be.

CHAPTER 4 : MS-DOS

1.0 INTRODUCTION

MS DOS is an acronym that stands for **M**icro**S**oft **D**isk **O**perating **S**ystem. It is often referred to as **DOS**. It is an old operating system for x86-based personal computers, purchased by Microsoft that manages everything on your computer: hardware, memory, files. It is an operating system that existed prior to Windows.

MS-DOS was the most commonly used member of the DOS family of operating systems, and was the main operating system for personal computers during the 1980s up to mid 1990s. It was preceded by M-DOS (also called MIDAS), designed and copyrighted by Microsoft in 1979. MSDOS was written for the Intel 8086 family of microprocessors, particularly the IBM PC and compatibles. It was gradually replaced on consumer desktop computers by operating systems offering a graphical user interface (GUI), in particular by various generations of the Microsoft Windows operating system. MS-DOS developed out of **QDOS** (**Q**uick and **D**irty **O**perating **S**ystem), also known as 86-DOS. DOS, as with any operating system, controls computer activity. It manages operations such as data flow, display, data entry amongst other various elements that make up a system.

The role of DOS is to interpret commands that the user enters via the keyboard.

These commands allow the following tasks to be executed:

- file and folder management
- disk upgrades
- hardware configuration
- memory optimization
- program execution

These commands are typed after the prompt, in the case of MS-DOS (Microsoft DOS, the most well known): the drive letter followed by a backslash, for example: A:\ or C:\. And after them, the enter key. The files that make up DOS involves:

IO.SYS : This is a program to handle input/output to your peripheral devices. It stays in memory when you run applications programs

MSDOS.SYS: This is a program for application programs to use. It contains special subprograms to make many commonly needed operations easy for programmers. COMMAND.COM : This program accepts the commands you enter and runs the right program. CONFIG.SYS: Configures the hardware environment Mouse , Printer, Keyboard , Country codes (time, date, currency),other devices and system commands AUTOEXEC.BAT: Programs/commands to be run at system start Batch file (automatically executing set of programs/commands) IO.SYS and MSDOS are loaded into the PC memory by a special program called a boot record each time you start up DOS . The command used to initialize new disks with DOS,FORMAT/S puts this on the disk along with IO.SYS and MSDOS.SYS

2.0 HISTORY

MS-DOS (Microsoft Disk Operating System) is a single-user, single-tasking computer operating system that uses a command line interface. In spite of its very small size and relative simplicity, it is one of the most successful operating systems that have been developed to date.

A Quick and Dirty History

When IBM launched its revolutionary personal computer, the IBM PC, in August 1981, it came complete with a 16-bit operating system from Microsoft, MS-DOS 1.0. This was Microsoft's first operating system, and it also became the first widely used operating system for the IBM PC and its clones. MS-DOS 1.0 was actually a renamed version of QDOS (Quick and Dirty Operating System), which Microsoft bought from a Seattle company, appropriately named Seattle Computer Products, in July 1981. QDOS had been developed as a clone of the CP/M eight-bit operating system in order to provide compatibility with the popular business applications of the day such as WordStar and dBase. CP/M (Control Program for Microcomputers) was written by Gary Kildall of Digital Research several years earlier and had become the first operating system for microcomputers in general use.

QDOS was written by Tim Paterson, a Seattle Computer Products employee, for the new Intel 16-bit 8086 CPU (central processing unit), and the first version was shipped in August, 1980. Although it was completed in a mere six weeks, QDOS was sufficiently different from CP/M to be considered legal. Paterson was later hired by Microsoft. Microsoft initially kept the IBM deal a secret from Seattle Computer Products. And in what was to become another extremely fortuitous move, Bill Gates, the not uncontroversial cofounder of Microsoft, persuaded IBM to let his company retain marketing rights for the operating system separately from the IBM PC project. Microsoft renamed it PC-DOS (the IBM version) and MS-DOS (the Microsoft version). The two versions were initially nearly identical, but they eventually diverged.

The acronym DOS was not new even then. It had originally been used by IBM in the 1960sin the name of an operating system (i.e., DOS/360) for its System/360 computer. At that time the use of disks for storing the operating system and data was considered cutting edge technology. Until its acquisition of QDOS, Microsoft had been mainly a vendor of computer programming languages. Gates and co-founder Paul Allen had written Microsoft BASIC and were selling it on disks and tape mostly to PC hobbyists.

MS-DOS soared in popularity with the surge in the PC market. Revenue from its sales fuelled Microsoft's phenomenal growth, and MS-DOS was the key to company's rapid emergence as the dominant firm in the software industry. This product continued to be the largest single contributor to Microsoft's income well after it had become more famous for Windows. Subsequent versions of MS-DOS featured improved performance and additional functions, not a few of which were copied from other operating systems. For example, version 1.25, released in 1982, added support for double-sided disks, thereby eliminating the need to manually turn the disks over to access the reverse side.

Version 2.0, released the next year, added support for directories, for IBM's then huge 10MB hard disk drive (HDD) and for 360KB, 5.25-inch floppy disks. This was followed by version 2.11 later in the same year, which added support for foreign and extended characters. Version 3.0 launched in 1984, added support for 1.2MB floppy disks and 32MB HDDs. This was soon followed by version 3.1, which added support for networks. Additions and improvements in subsequent versions included support for multiple HDD partitions, for disk compression and for larger partitions as well as an improved diskchecking utility, enhanced memory management, a disk defragmenter and an improved text editor.

The final major version was 7.0, which was released in 1995 as part of Microsoft Windows 95. It featured close integration with that operating system, including support for long filenames and the removal of numerous utilities, some of which were on the Windows 95 CDROM. It was revised in 1997 with version 7.1, which added support for the FAT32 file system on HDDs.

3.0 OPERATING SYSTEM FUNCTIONS

3.1 SCHEDULING

Scheduling is a key concept in computer multitasking, multiprocessing operating system and real-time operating system designs. **Scheduling** refers to the way processes are assigned to run on the available CPUs, since there are typically many more processes running than there are available CPUs. This assignment is carried out by software's known as a **scheduler** and **dispatcher**.

Objectives of a scheduler

- CPU utilization - to keep the CPU as busy as possible.
- Throughput - number of processes that complete their execution per time unit.
- Turnaround - total time between submission of a process and its completion.
- Waiting time - amount of time a process has been waiting in the ready queue.
- Response time - amount of time it takes from when a request was submitted ntil the first response is produced.
- Fairness - Equal CPU time to each thread.

But MS-DOS is non-multitasking, and as such did not feature a scheduler. MS-DOS was not designed to be a multi-user or multitasking operating system, but many attempts were made to retrofit these capabilities. Since it does not perform scheduling functions, when you run a sub

process synchronously on MS-DOS, make sure the program terminates and does not try to read keyboard input. If the
program does not terminate on its own, you will be unable to terminate it, because MS-DOS provides no general way to terminate a process. Pressing “ctrl C” or `C-<BREAK>' might sometimes help in these cases.
Group C Page 6

3.2 MEMORY MANAGEMENT

MS-DOS Memory Management Functions

 Provide students with a brief overview of memory management in the MS-DOS
operating system. Mention that to run a second job, the user must close or pause the first file before opening the second.

 Point out that the Memory Manager uses a first-fit memory allocation scheme in early DOS versions because it is the most efficient strategy in a single-user environment.

 Discuss briefly the two forms of main memory, ROM and RAM. MS-DOS provides three memory management functions- allocate, deallocate, and resize (modify). For most programs, these three memory allocation calls are not used.When DOS executes a program, it gives all of the available memory, from the start of that program to the end of RAM, to the executing process. Any attempt to allocate memory without first giving unused memory back to the system will produce an “insufficient memory” error.

ALLOCATE MEMORY

Function (ah): 48h
Entry parameters: bx- Requested block size (in paragraphs)
Exit parameters: If no error (carry clear):
ax:0 points at allocated memory block

If an error (carry set):
bx- maximum possible allocation size
ax- error code (7 or 8)
This call is used to allocate a block of memory. On entry into DOS, bx contains the size of the requested block in paragraphs (groups of 16 bytes). On exit, assuming no error, the ax register contains the segment address of the start of the allocated block. If an error occurs, the block is not allocated and the ax register is returned containing the error code.
If the allocation request failed due to insufficient memory, the bx register is returned containing the maximum number of paragraphs actually available.
Group C Page 7

DEALLOCATE MEMORY

Function (ah): 49h
Entry parameters: es:0- Segment address of block to be deallocated
Exit parameters: If the carry is set, ax contains the error code (7,9)
This call is used to deallocate memory allocated via function 48h above. The es register cannot contain an arbitrary memory address. It must contain a value returned by the allocate memory function. You cannot use this call to deallocate a portion of an allocated block. The modify allocation function is used for that operation.

MODIFY MEMORY ALLOCATION

Function (ah): 4Ah
Entry parameters: es:0- address of block to modify allocation size
bx- size of new block
Exit parameters: If the carry is set, then
ax contains the error code 7, 8, or 9
bx contains the maximum size possible (if error 8)
This call is used to change the size of an allocated block. On entry, es must contain the segment address of the allocated block returned by the memory allocation function. Bx must contain the

new size of this block in paragraphs. While you can almost always reduce the size of a block, you cannot normally increase the size of a block if other blocks have been allocated after the block being modified. Keep this in mind when using this function.
Group C Page 8

3.3 FILE SYSTEM

Before we go any further, it would be a good idea to look at the DOS file system. The **file system** lets us store information in named **files**. You can call a file anything you like which might help you remember what it contains as long as you follow certain basic rules:

1. File names can be up to 8 characters long. You can use letters and digits but only a few punctuation marks (! $ % # ~ @ - ( ) _ { }). You can't exceed 8 characters or use spaces or characters like * or ? or +. Names are case-insensitive, i.e. it doesn't matter whether you use capitals or lowercase letters; "A" and "a" are treated as the same thing.
2. File names can also have an **extension** of up to three characters which describes the type of file. There are some standard extensions, but you don't have to use them.
Examples include COM and EXE for executable programs, TXT for text files, BAK
for backup copies of files, or CPP for C++ program files. The extension is separated by a dot from the rest of the filename.
For example, a file called FILENAME.EXT has an 8-character name (FILENAME) followed by a three-character extension (.EXT). You could also refer to it as filename.txt since case doesn't matter, but I'm going to use names in capitals for emphasis throughout this document. Files are stored in **directories**; a directory is actually just a special type of file which holds a list of the files within it. Since a directory is a file, you can have directories within directories. Directory names also follow the same naming rules as other files, but although they can have
an extension they aren't normally given one (just an 8-character name). The system keeps track of your **current directory**, and if you just refer to a file using a name
like FILENAME.EXT it's assumed you mean a file of that name in the current directory. You can specify a **pathname** to identify a file which includes the directory name as well; the directory is separated from the rest of the name by a backslash ("\"). For example, a file called

LETTER1.TXT in a directory called LETTERS can be referred to as LETTERS\LETTER1.TXT (assuming that the current directory contains the LETTERS directory as a subdirectory). If LETTERS contains a subdirectory called PERSONAL, which in turn contains a file called DEARJOHN.TXT, you would refer to this file as Group C Page 9

LETTERS\PERSONAL\DEARJOHN.TXT (i.e. look in the LETTERS directory for PERSONAL\DEARJOHN.TXT, which in turn involves looking in the PERSONAL subdirectory for the file DEARJOHN.TXT).

Every disk has a **root directory** which is the main directory that everything else is part of. The root directory is called "\", so you can use **absolute pathnames** which don't depend on what your current directory is. A name like \LETTERS\LETTER1.TXT always refers to the same file regardless of which directory you happen to be working in at the time; the "\" at the beginning means "start looking in the root directory", so \LETTERS\LETTER1.TXT means "look in the root directory of the disk for a subdirectory called LETTERS, then look in this
subdirectory for a file called LETTER1.TXT". Leaving out the "\" at the beginning makes this a **relative pathname** whose meaning is relative to the current directory at the time. If you want to refer to a file on another disk, you can put a letter identifying the disk at the beginning of the name separated from the rest of the name by a colon (":"). For example,

A:\LETTER1.TXT refers to a file called LETTER1.TXT in the root directory of drive A. DOS keeps track of the current directory on each disk separately, so a relative pathname like A:LETTER1.TXT refers to a file called LETTER1.TXT in the currently-selected directory on drive A. For convenience, all directories (except root directories) contain two special names: "." refers to the directory itself, and ".." refers to the parent directory (i.e. the directory that contains this one). For example, if the current directory is \LETTERS\PERSONAL, the name ".." refers to the directory \LETTERS, "..\BUSINESS" refers to \LETTERS\BUSINESS, and "..\.." refers to the root directory "\".

Group C Page 10

3.4 PROCESS MANAGEMENT

MS-DOS boot process

1. The BIOS, having completed its test and setup functions, loads the boot code found in the master boot record and then transfers control of the system to it. At that point, the master boot record code is executed. If the boot device is a floppy disk, the process skips to step 7 below.

2. The next step in the process is the master boot code examining the master partition table. It first must determine if there is an extended DOS partition, then it must determine if there is a bootable partition specified in the partition table.

3. If the master boot code locates an extended partition on the disk, it loads the extended partition table that describes the first logical volume in the extended partition. This extended partition table is examined to see if it points to another extended partition table. If it does, this second table is examined for information about the *second* logical volume in the extended partition. Logical volumes in the extended partition have their extended partition table chained together one to the next. This process continues until all of the extended partitions have been loaded and recognized by the system.

4. Once the extended partition information (if any) has been loaded, the boot code attempts to start the primary partition that is marked active, referred to as the boot partition. If no boot partitions are marked active, then the boot process will terminate with an error. The error message is often the same as that which occurs if the BIOS could not locate a boot device, generally shown on screen as "No boot device", but also can show up as "NO ROM BASIC - SYSTEM HALTED". If there is a primary partition marked active and there is an installed operating system, the boot code will boot it. The rest of the steps presume this example is of an MS- DOS primary partition.
Group C Page 11

5. At this stage, the master or volume boot sector is loaded into memory and tested, and the boot code that it contains is given control of the remainder of the boot

process. 6. The boot code examines the disk structures to ensure that everything is correct. If not, the boot process will end in an error here.

7. During the next step, the boot code searches the root directory of the device being booted for the operating system files that contain the operating system. For MS-DOS, these are the files "IO.SYS", "MSDOS.SYS" and "COMMAND.COM".

8. If no operating system files are found, the boot program will display an error
message similar to "Non-system disk or disk error - Replace and press any key when ready". Keep in mind that this message does not means that the system was never booted. It means that the BIOS examined the floppy disk for example and just rejected it because it couldn't boot an operating system. The volume boot code was indeed loaded and executed, as that is what posts the message when it can't find the operating system files.

9. In the final stages of the boot process, presuming that the operating system files are found, the boot program will load those operating system files into memory and transfer control to them. In MS-DOS, the first is IO.SYS and its code is executed.

10. SYS will then execute MSDOS.SYS. Then the more complete operating system code loads and initializes the rest of the operating system structures beginning with the command interpreter COMMAND.COM and then the execution of the CONFIG.SYS and AUTOEXEC.BAT files. At this point the operating system code itself has control of the computer.

MS-DOS Process Management

This relates to the Operating System’s activity of managing the processor in the system, i.e. allocating and de-allocating of the processor to the process. The Operating System decides.
Group C Page 12
the same based on the priority of the process depending if there exists any and certain predefined algorithms. Although MS-DOS is a single tasking operating system, this does not mean there can only be one program at a time in memory. However we can still load several programs into

memory at one time under DOS. The only catch is that DOS only provides the ability for them to run one at a time in a very specific fashion. Unless the processes are cooperating, their execution profile follows a very strict pattern. That's why Dos exhibit Serial Multi tasking.
Users often wish to perform more than one activity at a time (load a remote file while editing a program) and uni programming does not allow this. So DOS put in things like memory resident programs that invoked asynchronously, but still have separation problems. One key problem with DOS is that there is no memory protection - one program may write the memory of another program, causing weird bugs.

Child Processes in DOS

In Dos we have one process and one thread.

When a DOS application is running, it can load and executing some other programs using the DOS EXEC function. Under normal circumstances, when an application (the parent) runs a second program (the child), the child process executes to completion and then returns to the parent. This is very much like a procedure call, except it is a little more difficult to pass parameters between the two. Group C Page 13

3.5 INPUTS/OUTPUT IN MSDOS

The core of MS-DOS is a device-independent input/output (I/O) handler, represented on a system disk by the hidden file MSDOS.SYS. It accepts requests from application programs to do high-level I/O, such as sequential or random access of named disk files, or communication with character devices such as the console. The handler processes these requests and converts them to a very low level form that can be handled by the I/O system. Because MSDOS.SYS is hardware independent, it is nearly identical in all MS-DOS versions provided by
manufacturers with their equipment. The I/O system is totally device dependent and is represented on the disk by the hidden file IO.SYS. It is normally written by hardware manufacturers (who know their equipment best, anyway) with the notable exception of IBM,

whose I/O system was written to IBM's specifications by Microsoft. The tasks required of the I/O system, such as outputting a single
byte to a character device or reading a contiguous group of physical disk sectors into memory, are as simple as possible.

Departing From the Windows for MS-DOS I/O Model

The I/O system used in Windows for MS-DOS is based on, and limited by, capabilities of the underlying MS-DOS operating system. The device drivers at the core of this I/O system are commonly:

 Written in assembly language--they're not portable: they can only run on Intel 80x86 family processors.

 Monolithic in design, not layered--similar code for common tasks is repeated in each device driver for a given class of devices. Monolithic design also makes mixing and matching different file systems and device drivers impossible.

 Not designed for pre-emptive multitasking use--Windows has to perform many nasty tricks to allow even non-pre-emptive multitasking with non-multitasking MS-DOS and its non-multitasking device drivers.

 Incompatible with multiprocessor platforms--MS-DOS is inherently a singleprocessor OS, so MS-DOS device drivers are utterly incapable of synchronizing access to shared resources in a multiprocessor situation.
Group C Page 14

3.6 SECURITY

The security syntax set in MSDOS is masked on the command base and the password identifier which cannot be altered by someone who is not authorized. Password can be set for some programs starting from MSDOS like QBASIC.

The command – line is not prone to virus. Also it MSDOS cannot be used by someone who does not know the commands limiting the risk of harming the system. Group C Page 15

4.0 DESIGN ISSUES

MS-DOS Design Criteria

The primary design requirement of MS-DOS was CP/M-80 translation compatibility, meaning that, if an 8080 or Z80 program for CP/M were translated for the 8086 according to Intel's published rules, that program would execute properly under MS-DOS. Making CP/M-80 translation compatibility a requirement served to promote rapid development of 8086 software, which, naturally, Seattle Computer was interested in. There was partial success: those software developers who chose to translate their CP/M-80 programs found that they did indeed run under MS-DOS, often on the first try. Unfortunately, many of the software developers Seattle Computer talked to in the earlier days preferred to simply ignore MSDOS. Until the IBM Personal Computer was announced, these developers felt that CP/M-86 would be *the* operating system of 8086/8088 computers. Other concerns crucial to the design of MS-DOS were speed and efficiency. Efficiency
primarily means making as much disk space as possible available for storing data by minimizing waste and overhead. The problem of speed was attacked three ways: by minimizing the number of disk transfers, making the needed disk transfers happen as quickly as possible, and reducing the DOS's "compute time," considered overhead by an application program. The entire file structure and disk interface were developed for the greatest speed and efficiency. The last design requirement was that MS-DOS be written in assembly language. While this characteristic does help meet the need for speed and efficiency, the reason for including it is much more basic. The only 8086 software-development tools available to Seattle Computer at that time were an assembler that ran on the Z80 under CP/M and a monitor/debugger that fit into a 2K-byte EPROM (erasable programmable read-only memory). Both of these tools had been developed in house. Group C Page 16

5.0 IMPLEMENTATION

MSDOS is a single-user operating system. MS-DOS employs a command line interface and a batch scripting facility via its command interpreter, command.com (all operations to be carried out must be written through commands); without the knowledge of those commands, the user cannot execute a job. Despite its command-line interface, it is easy to learn. The commands of MSDOS are not case sensitive. MSDOS is supported and embedded in other operating system like *Microsoft windows, MacOs;* it can be launched in windows by:

(i) Start -> All programs -> accessories -> command prompt or
(ii) Start -> run -> type **cmd** -> ok
The MSDOS contains five files (IO.SYS, MSDOS.SYS, COMMAND.COM, CONFIG.SYS and AUTOEXEC.BAT). The commands in MSDOS can be categorized in two forms:
INTERNAL and **EXTERNAL** commands.
Group C Page 17

Internal commands are executed without loading a separate program file. Command.com is responsible for the execution of internal commands and the special batch commands. Internal commands are part of the command processor always available to be used. External command exists as executable files handled by separate programs from the DOS diskette. Each program is a member of the .COM or .EXE family. An external command can only be used when the disk containing the program is in drive. They can be used for peripheral devices like printer. A file's full name is called FILESPEC. The FILESPEC for a disk file has four parts: ***drive name, directory name, file name and extension.*** Characters like. " / \ [ ] : | < > + = ; , cannot be used in naming files because they mean other things in MSDOS. When you start up DOS, the current directory is automatically the root directory. The CD will change the current directory. Other folders in the root directory are called sub – directories. A file in any directory can be accessed by typing:

cd ROOT DIRECTORY\SUB-DIRECTORY (Level 1)\ SUB-DIRECTORY (level2)...\filename.ext. CLASSIFYING MS-DOS COMMANDS

Either internal or external command will be specified for each command of MSDOS in this text. MS-DOS commands fall roughly into three categories;
Group C Page 18

1. **Environment Commands:** These report on or affect the operating system environment. Examples are CLS (clear screen), TIME, DATE, VER (display MS-DOS version number), and HELP.

 BREAK (internal): Used from the DOS prompt or in a batch file or in the
CONFIG.SYS file to set (or display) whether or not DOS should check for a Ctrl +
Break key combination.
BREAK =on|off

 CLS (internal) – clears screen: Clears (erases) the screen. **CLS**

 DATE AND TIME (internal): Displays and/or sets the system date. **DATE mm-dd-yy** or **DATE**

 GRAPHICS (external): Provides a way to print contents of a graphics screen display.

GRAPHICS [printer type][profile] [/B][/R][/LCD][/PB:(id)] [/C][/F][/P(port)]

 MODE (external): Sets mode of operation for devices or communications.
MODE n
MODE LPT#[:][n][,][m][,][P][retry]
MODE [n],m[,T]
MODE (displaytype,linetotal)
MODE COMn[:]baud[,][parity][,][databits][,][stopbits][,][retry]
MODE LPT#[:]=COMn [retry]
MODE CON[RATE=(number)][DELAY=(number)]
MODE (device) CODEPAGE PREPARE=(codepage) [d:][path]filename

MODE (device) CODEPAGE PREPARE=(codepage list) [d:][path]filename
MODE (device) CODEPAGE SELECT=(codepage)
MODE (device) CODEPAGE [/STATUS]
MODE (device) CODEPAGE REFRESH

 VER (internal): Displays the DOS version number. **VER**

 SELECT (external): Formats a disk and installs country-specific information and keyboard codes (starting with DOS Version 6, this command is no longer available).

SELECT [d:] [d:][path] [country code][keyboard code]

2. **Directory and File Commands:** These manipulate files. Examples are COPY, DEL (delete), TYPE (display file to screen) and, DIR (directory - or list all files in current directory). It can be further divided into three (3);

(a) **Commands for disk maintenance**

 BACKUP (external): Makes a backup copy of one or more files. (In DOS Version 6, this program is stored on the DOS supplemental disk.)
Group C Page 19

BACKUP d:[path][filename] d:[/S][/M][/A][/F:(size)] [/P][/D:date] [/T:time] [/L:[path]filename]

 RESTORE (external): Restores to standard disk storage format files previously stored using the BACKUP command.

RESTORE d: [d:][path]filename [/P][/S][/B:mm-dd-yy] [/A:mm-ddyy][/ E:hh:mm:ss] [/L:hh:mm:ss] [/M][/N][/D]

 RECOVER (external): Resolves sector problems on a file or a disk. (Beginning with DOS Version 6, RECOVER is no longer available).

RECOVER [d:][path]filename or **RECOVER d:**

 VERIFY (internal): Turns on the verify mode; the program checks all copying
operations to assure that files are copied correctly.
VERIFY on|off

 FORMAT (external): Formats a disk to accept DOS files. **FORMAT d:[/1][/4][/8][/F:(size)] [/N:(sectors)] [/T:(tracks)][/B|/S][/C][/V:(label)]**
[/Q][/U][/V]

 SYS (external): Transfers the operating system files to another disk.
SYS [source] d:

 CHKDSK (external): Checks a disk and provides a file and memory status report.
CHKDSK [d:][path][filename] [/F][/V]

 DISKCOPY (external): Makes an exact copy of a diskette.
DISKCOPY [d:] [d:][/1][/V][/M]

 DISKCOMP (external): Compares the contents of two diskettes.
DISKCOMP [d:] [d:][/1][/8]

 LABEL (external): Creates or changes or deletes a volume label for a disk.
LABEL [d:][volume label]

 VOL (internal): Displays a disk's volume label.
VOL [d:] (b) **Commands for directory control**

 DIR (internal): Displays directory of files and directories stored on disk.

DIR [d:][path][filename] [/A:(attributes)] [/O:(order)]
[/B][/C][/CH][/L][/S][/P][/W]

 ASSIGN (external): Redirects disk drive requests to a different drive.

ASSIGN A:=B: [...] /sta

 MKDIR (internal): Creates a new subdirectory.

MKDIR (MD) [d:]path

Group C Page 20

 CHDIR (internal): Displays working (current) directory and/or changes to a different directory.

CHDIR (CD) [d:]path or **CHDIR (CD)[..]**

 RMDIR (internal): Removes a subdirectory.

RMDIR (RD) [d:]path

 TREE (external): Displays directory paths and (optionally) files in each subdirectory.

TREE **[d:][path] [/A][/F]**

 PATH (external): Sets or displays directories that will be searched for programs not in the current directory.

PATH; or **PATH [d:]path[;][d:]path[...]**

 JOIN (external): Allows access to the directory structure and files of a drive through a directory on a different drive.

JOIN d: [d:path] or **JOIN d: [/D]**

 SUBST (external): Substitutes a virtual drive letter for a path designation.

SUBST d: d:path or **SUBST d: /D**

(c) **Commands for file control:**

 COPY (internal): copies source file to target file and appends file.

COPY [d:][path]source [d:][path][target] [/V]

Or COPY [d:][path]filename+[d:][path]filename[...][d:][path][filename] [/V]

 COMP (external): Compares two groups of files to find information that does not match.

COMP [d:][path][filename] [d:][path][filename]

[/A][/C][/D][/L][/N:(number)]

 RNAME (internal): Changes the filename under which a file is stored.

RENAME (REN) [d:][path]filename [d:][path]filename

 ERASE/DELETE (internal): Deletes (erases) files from disk.

DEL (ERASE) [d:][path]filename [/P]

 TYPE (internal): Displays the contents of a file.

[d:][path]filename

 PRINT (external): Queues and prints data files.

PRINT [/B:(buffersize)] [/D:(device)] [/M:(maxtick)] [/Q:(value] [/S:(timeslice)] [/U:(busytick)] [/C][/P][/T] [d:][path][filename] [...]

Group C Page 21

 ATTRIB (external) : Sets or displays the read-only, archive, system, and hidden attributes of a file or directory.

ATTRIB [d:][path]filename [/S]

ATTRIB [+R|-R] [+A|-A] [+S|-S] [+H|-H] [d:][path]filename [/S]

3. **Utilities:** These perform some useful function. Examples are FORMAT (format a diskette) and EDIT (invoke MS-DOS text editor).

Group C Page 22

6.0 STRENGTH AND WEAKNESS

Strengths of Microsoft Disk Operating System

(a) **It has a good User interface**
MS-DOS employs a command line interface and a batch scripting facility via its command interpreter, command.com. MS-DOS was designed so users could easily substitute a different command line interpreter

(b) **MS-DOS compatibility with other Microsoft operating systems** users also desired a graphical user interface. Many programs running under MS-DOS tried to fill the void by creating their own graphical interface, such as Microsoft Word for DOS, XTree, and the Norton Shell. However, this required duplication of effort and did not provide much consistency in interface design (even between product lines). Non-Microsoft efforts to provide a consistent interface.

(c) **It has command line interpreter which is the most efficient way to manage files and run a computer program**. It can be used to run program like JAVA and C++, MYSQL.

(d) **Software Base**
There is a huge software base for developing software in DOS, which is another major strength.

Weakness of MS-DOS
(a) **MSDOS** is a single user operating system.
(b) It does not support features like multi-tasking and multi-processing.
(c) DOS doesn't have built-in capability for scheduling or multithreading.
(d) You must also install interrupt handlers directly into the software application, and API calls tend to be through software interrupts rather than some other more direct procedural method instead.

(e) Equipment vendors supporting DOS tend to follow an approach of either providing raw spec sheets for their equipment or writing a pre-compiled binary object library that has to be linked into your software using a specific compiler.

Group C Page 23

CONCLUSION

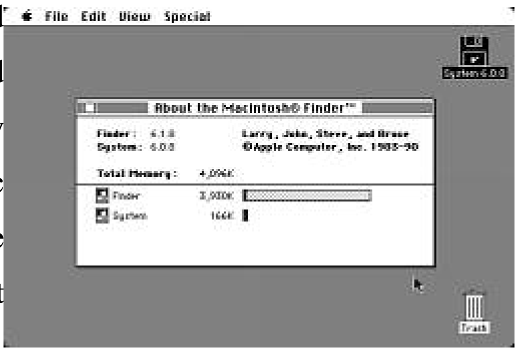
MSDOS is a powerful operating system. Though it is an old operating system yet not outdated and versions of it are being released. There is a huge software base for developing software in DOS, which is another major strength**.** DOS controls the computer's hardware and provides an environment for programs to run. Everything you can do with a GUI can be done at the DOS prompt. It can be used to execute specific programs directly from the command prompt like sql queries, C++, C# and Java programs because it interface between with computer hardware and software effectively.

CHAPTER 5: MACINTOSH OPERATING SYSTEM

INTRODUCTION

Mac OS is a Graphical User Interface based operating system designed for Apple's Macintosh Computer Mac OS was named by the company Apple as "Mac System Software" in the beginning, a specially designed operating system only for 68000 first Motorola processors. With own Macintosh hardware, Mac OS takes up a special role in the world of desktop systems.

The first version was "System 1" and appeared bundled with the Mac in 1984. The classic desktop is designed as a single user operating system and almost completely hides the full path to files and directories. The graphic representation is reduced to the essence. Overall the interface is very easy to use and does not need the right mouse button for user interaction.



About the Macintosh® Finder™

Finder: 6.1.8
System: 6.0.8
Larry, John, Steve, and Bruce
©Apple Computer, Inc. 1983-90
Total Memory: 4,096K
Finder 3,930K
System 166K

Starting with System 3.0, the used filesystem, Hierachical File System was used officially, which does not different between uppercase and lowercase letters.

System 5.0 was the first release to run several programs with the integrated MultiFinder at the same time. In 1988, system 6.0 came onto the market. It requires 1 MB RAM and can address up to 8 MB. The file system can organize hard disks up to 2 GByte with 65,536 files. Optionally applications run with the multi Finder in cooperative multitasking. For word processing are programs such as WriteNow, MacWrite II, and Microsoft Word 4.0 available In May 1991, system 7 came into existence. The new operating system needed 2 MB RAM, optionally it can be switched to 32-bit depending from the used hardware. New is the direct support of networks with file exchange, AppleScript as scripting language and display of colors. Balloons provide help for the user to use the interface. The TrueType fonts are scalable to any size.

The System Software 7.5 appeared in 1994 and requires at least 4 MB RAM. It was running both on 68000-Macs and Power Macintosh. In September 1996, the update System 7.5.5 includes all available bug fixes, Open Transport 1.1.2, current Ethernet driver and support for storage drive volumes up to 4 GB. With release 7.6 the company Apple changed the name from Mac Software

System to Mac Operating System in 1997. Mac OS 8 by Apple appeared in July 1997. As minimum requirements are specified a 68040 or PowerPC processor, 32 MB RAM and 120 MB of free disk space. The CTRL key is used to display a specific context menu for different actions. It makes it easier to copy files. **Mac OS 8.1**, Informations are stored more efficiently on the file system. The file system can handle up to 2 billion files with a current file size of up to 2 GB.

Mac OS 8.5 further optimized the stability and speed of the operating system, AppleScript is now up to 5 times faster than the previous version. The graphical display is accelerated by new QuickDraw routines. Copying files has become faster and increase the disk throughput. A tool for system maintaining detects and fixes errors on the file system automated. Following applications are included in current version: Finder 8.5 QuickTime Pro 3, Open Transport 2, Internet Explorer 4.01, Outlook Express 4.01, e.t.c.

The operating system Mac OS 9 has been developed under the name Sonata and released to the 23. October 1999. The installation requires 32 MB RAM with virtual memory. The free disk storage should be 150 up to 400 MB depending on the installation type. 50 new features are added in comparison to the previous version. This includes support for multiple users with password and access management for files and settings. The login is available through authentication by voice. MAC OS X

The operating system core Darwin is open source, Mac OS X works with preemptive multi-tasking and includes beside the new Graphical user interface (GUI) Aqua the classic GUI from Mac OS 9.

Mac OS X 10.0 came out in March 2001. To install are 128 MB RAM (256 MB RAM starting from Mac OS X 10.3.9) and 1.5 GB hard disk space (3.0 GByte starting from Mac OS X 10.2) provided. Mac OS X 10.5 requires at least 512 MB RAM and 9 GByte of free disk space.

- 32-bit or 64-bit processing

Field of Application
- digital photography
- 2-D and 3-D animations
- video processing
- audio processing

Structure Information
- supports QuickTime
- graphical user interaction with the finder
- graphical representation by Quickdraw

Considerable performance and comfort improvements were carried out in version **Mac OS X 10.1**. The surface reacts quicker at user interaction, the system start was accelerated and the OpenGL performance increased noticeable.

Mac OS X 10.3 has now a GUI in metallic scheme and the optimized Finder. The use and access in heterogeneous networks was further simplified. 12 million MacOS X user were counted in October 2004.

According to Apple **Mac OS X 10.4** brings more than 200 new features. Features are the fast, system-wide and index-based search function named Spotlight, the Dashboard for easy access to small programms (Widgets), the Automator for the simplified composition of Applescripts for the automation of tasks. The Web browser Safari in version 2.0 now contained. Further novelty is the delivery at a DVD medium, an installation of CD-ROM is no longer possible.

First since the 10th January 2006 is MacOS X 10.4.4 next to the PowerPC version available for Intel based Macs. On the 6. June 2005 Steve jobs announced the switch to Intel processors. As further details became known that Apple had developed Mac OS X since 2000 internally also for the Intel platform.

Apple released the successor **MacOS X 10.5**, Leopard at the 26 october,2007. With more than 300 innovations MacOS offers the user an enhanced user interface with virtual desktops, a fast file preview and Dock with 3D effect.ose snapshot. The security of the operating system and applications is improved by 11 enhancements. The first update with bug fixes was released with Mac OS X 10.5.1 by Apple on November 15th, 2007. It contains general bug fixes for the operating system to improve stability, better compatibility and safety.
Mac OS X 10.5.2 cames with 125 bug fixes and smaller optimizations on January 24th, 2008.

Mac OS X 10.6 is a Mac computer with Intel Core 2 Duo processor with at least 1 GB memory and 5 GB free space ahead. This operating system no longer exists as PowerPC execution. Apple placed the focus development on performance and stability. It supports up to 16 TByte memory, it is optimized for multi core processors, and is a pure 64-bit operating system. With the technology OpenCL graphics processor can speed up in specific applications calculation

PROCESS MANAGEMENT

Mac OS implemented non-preemptive multitasking. Although the scheduling algorithm was simple in the absence of preemption, it supported process priorities.
A process could only be created by another process, except the initial process, which was created by the operating system as the “shell” process upon booting. The shell process ran the *Desktop Manager* application by default. The system’s process management API included calls for creating, terminating, suspending, and resuming.

Technical History of Apple’s Operating Systems processes.

Terminating a process also resulted in the termination of all its descendants. Examples of MAC process-management system calls included the following.
• make_process
• kill_process

- activate_process
- suspend_process
- info_process
- setpriority_process
- yield_process
- sched_class

System-level exceptions resulted in the termination of a process—a side effect of the execution of default exception handlers. Processes could install custom exception handlers, which were invoked with detailed exception context. Examples of Lisa exception-management system calls included the following.

- enable_excep
- disable_excep
- declare_excep_hdl
- signal_excep

Interprocess Communication

By default, a process was not allowed to access the logical address space of another process. Interprocess communication was possible through multiple mechanisms such as events, shared files, and shared memory. *Events* were structured messages consisting of a system-attached header and a sender-provided data block, transmitted between processes over named *channels*. A process could listen on a channel, waiting for messages to arrive. Alternatively, a process could register an exception handler and arrange for an exception to be generated upon message arrival.

Mac OS X Internals

Examples of mac event-channel management system calls included the following.

- make_event_chn
- kill_event_chn

- open_event_chn
- close_event_chn
- wait_event_chn

CHAPTER SIX: WINDOWS OPERATING SYSTEM

INTRODUCTION

Microsoft Windows is a series of software operating systems and graphical user interfaces produced by Microsoft. Microsoft first introduced an operating environment named Windows in November 1985 as an add-on to MS-DOS in response to the growing interest in graphical user interfaces (GUIs). Microsoft Windows came to dominate the world's personal computer market, overtaking Mac OS, which had been introduced previously. At the 2004 IDC Directions conference, it was stated that Windows had approximately 90% of the client operating system market. The most recent client version of Windows is Windows Vista; the most recent server version is Windows Server 2008. Vista's successor, Windows 7 (currently in public beta) is slated to be released between July 1, 2009 and June 30, 2010.
The term **Windows** collectively describes any or all of several generations of Microsoft operating system products. These products are generally categorized as follows:

✓ History of Microsoft Windows

Microsoft has taken two parallel routes in its operating systems. One route has been for the home user and the other has been for the professional IT user. The dual routes have generally led to home versions having greater multimedia support and less functionality in networking and security, and professional versions having inferior multimedia support and better networking and security.

The first version of Microsoft Windows, version 1.0, released in November 1985, lacked a degree of functionality and achieved little popularity, and was to compete with Apple's own operating system. Windows 1.0 is not a complete operating system; rather, it extends MS-DOS. Microsoft Windows version 2.0 was released in November, 1987 and was slightly more popular than its predecessor. Windows 2.03 (release date January 1988) had changed the OS from tiled windows to overlapping windows. The result of this change led to Apple Computer filing a suit against Microsoft alleging infringement on Apple's copyrights.

A Windows for Workgroups 3.11 desktop

Microsoft Windows version 3.0, released in 1990, was the first Microsoft Windows version to achieve broad commercial success, selling 2 million copies in the first six months. It featured improvements to the user interface and to multitasking capabilities. It received a facelift in Windows 3.1, made generally available on March 1, 1992. Windows 3.1 support ended on December 31, 2001.
In July 1993, Microsoft released Windows NT based on a new kernel. NT was considered to be the professional OS and was the first Windows version to utilize preemptive multitasking.[citation needed]. Windows NT would later be retooled to also function as a home operating system, with Windows XP.
On August 24, 1995, Microsoft released Windows 95, a new, and major, consumer version that made further changes to the user interface, and also used preemptive multitasking. Windows 95 was designed to replace not only Windows 3.1, but also Windows for Workgroups, and MS-DOS. It was also the first Windows operating system to use Plug and Play capabilities. The changes Windows 95 brought to the desktop were revolutionary, as opposed to evolutionary, such as those in Windows 98 and Windows Me. Mainstream support for Windows 95 ended on December 31, 2000 and extended support for Windows 95 ended on December 31, 2001.

The next in the consumer line was Microsoft Windows 98 released on June 25, 1998. It was substantially criticized for its slowness and for its unreliability compared with Windows 95, but many of its basic problems were later rectified with the release of Windows 98 Second Edition in 1999. Mainstream support for Windows 98 ended on June 30, 2002 and extended support for Windows 98 ended on July 11, 2006.

As part of its "professional" line, Microsoft released Windows 2000 in February 2000. The consumer version following Windows 98 was Windows Me (Windows Millennium Edition). Released in September 2000, Windows Me implemented a number of new technologies for Microsoft: most notably publicized was "Universal Plug and Play."

In October 2001, Microsoft released Windows XP, a version built on the Windows NT kernel that also retained the consumer-oriented usability of Windows 95 and its successors. This new version was widely praised in computer magazines. It shipped in two distinct editions, "Home" and "Professional", the former lacking many of the superior security and networking features of the Professional edition. Additionally, the first "Media Center" edition was released in 2002,[20] with an emphasis on support for DVD and TV functionality including program recording and a remote control. Mainstream support for Windows XP ended on April 14, 2009. Extended support will continue until April 8, 2014.

In April 2003, Windows Server 2003 was introduced, replacing the Windows 2000 line of server products with a number of new features and a strong focus on security; this was followed in December 2005 by Windows Server 2003 R2.
On January 30, 2007 Microsoft released Windows Vista. It contains a number of new features, from a redesigned shell and user interface to significant technical changes, with a particular focus on security features. It is available in a number of different editions, and has been subject to some criticism.

- ✓ **Early versions**

Windows 1.0, Windows 2.0, and Windows 2.1x

The history of Windows dates back to September 1981, when the project named "Interface Manager" was started. It was announced in November 1983 (after the Apple Lisa, but before the Macintosh) under the name "Windows", but Windows 1.0 was not released until November 1985. The shell of Windows 1.0 was a program known as the MS-DOS Executive. Other supplied programs are Calculator, Calendar, Cardfile, Clipboard viewer, Clock, Control Panel, Notepad, Paint, Reversi, Terminal, and Write. Windows 1.0 does not allow overlapping windows, due to Apple Computer owning this feature. Instead all windows are tiled. Only dialog boxes can appear over other windows.

Windows 2.0 was released in October 1987 and featured several improvements to the user interface and memory management. Windows 2.0 allowed application windows to overlap each other and also introduced more sophisticated keyboard-shortcuts. It could also make use of expanded memory.

Windows 2.1 was released in two different flavors: Windows/386 employed the 386 virtual 8086 mode to multitask several DOS programs, and the paged memory model to emulate expanded memory using available extended memory. Windows/286 (which, despite its name, would run on the 8086) still ran in real mode, but could make use of the high memory area.

The early versions of Windows were often thought of as simply graphical user interfaces, mostly because they ran on top of MS-DOS and used it for file system services. However, even the earliest 16-bit Windows versions already assumed many typical operating system functions; notably, having their own executable file format and providing their own device drivers (timer, graphics, printer, mouse, keyboard and sound) for applications. Unlike MS-DOS, Windows allowed users to execute multiple graphical applications at the same time, through cooperative multitasking.

Windows implemented an elaborate, segment-based, software virtual memory scheme, which allowed it to run applications larger than available memory: code segments and resources were swapped in and thrown away when memory became scarce, and data segments moved in memory when a given application had relinquished processor control, typically waiting for user input.

Windows 3.0 and Windows 3.1x

Windows 3.0 (1990) and Windows 3.1 (1992) improved the design, mostly because of virtual memory and loadable virtual device drivers (VxDs) which allowed them to share arbitrary devices between multitasked DOS windows. Also, Windows applications could now run in protected mode (when Windows was running in Standard or 386 Enhanced Mode), which gave them access to several megabytes of memory and removed the obligation to participate in the software virtual memory scheme. They still ran inside the same address space, where the segmented memory provided a degree of protection, and multi-tasked cooperatively. For

Windows 3.0, Microsoft also rewrote critical operations from C into assembly, making this release faster and less memory-hungry than its predecessors. With the introduction of the Windows for Workgroups 3.11, Windows was able to bypass DOS for file management operations using 32-bit file access.

Windows 95, Windows 98, and Windows Me

Windows 95 featured a new user interface, supported long file names, could automatically detect and configure installed hardware (plug and play), natively ran 32-bit applications, and featured several technological improvements that increased its stability over Windows 3.1. Windows 95 uses pre-emptive multitasking and runs each 32-bit application in a separate address space. This makes it harder for a single buggy application to crash the whole system. It was still not a secure multi-user operating system like Windows NT as a strict separation between applications was not enforced by the kernel. The API was a subset of the Win32 API supported by Windows NT, notably lacking support for Unicode and functions related to security. Windows 95 was now bundled together with MS-DOS 7.0, however its role was mostly delegated to that of a boot loader.

There were several releases of Windows 95; the first in 1995, with Service Pack 1 following in December which included Internet Explorer 2.0. Subsequent versions were only available with the purchase of a new computer and were called OEM Service Releases. OSR1 was equivalent to Windows 95 with SP1. OSR2 (also called Windows 95 B) included support for FAT32 and UDMA and shipped with Internet Explorer 3.0. OSR 2.1 included basic support for USB and OSR 2.5 (also called Windows 95 C) shipped with Internet Explorer 4.0.

Microsoft's next OS was Windows 98, which had two versions; the first in 1998 and the second, named Windows 98 Second Edition, in 1999.

In 2000, Microsoft released Windows Me (Me standing for Millennium Edition), which used the same core as Windows 98 but adopted some aspects of Windows 2000 and removed the "boot in DOS mode" option. It also added a new feature called System Restore, allowing the user to set the computer's settings back to an earlier date. Me is also the last DOS-based Windows release which does not include Microsoft Product Activation.

Windows NT family

The NT family of Windows systems was fashioned and marketed for higher reliability business use, and was unencumbered by any Microsoft DOS patrimony. The first release was MS Windows NT 3.1 (1993, numbered "3.1" to match the consumer Windows version, which was followed by NT 3.5 (1994), NT 3.51 (1995), NT 4.0 (1996), and Windows 2000 (2000). 2000 is the last NT-based Windows release which does not include Microsoft Product Activation. NT 4.0 was the first in this line to implement the "Windows 95" user interface (and the first to include Windows 95's built-in 32-bit runtimes). Microsoft then moved to combine their consumer and business operating systems with Windows XP, coming in both home and professional versions (and later niche market versions for tablet PCs and media centers); they also diverged release schedules for server operating systems. Windows Server 2003, released a year and a half after Windows XP, brought Windows Server up to date with MS Windows XP. After a lengthy development process, Windows Vista was released toward the end of 2006, and its server counterpart, Windows Server 2008 was released in early 2008. In 2009, Windows 7 and Windows Server 2008 R2 entered beta. Microsoft plans to release Windows 7 in late 2009 or early 2010.
Windows CE, Microsoft's offering in the mobile and embedded markets, is also a true 32-bit operating system that offers various services for all sub-operating workstations.

Windows CE

Windows CE (officially known as Windows Embedded), is an edition of Windows that runs on minimalistic computers, like satellite navigation systems, and uncommonly mobile phones. Windows Embedded runs as CE, rather than NT, which is why it should not be mistaken for Windows XP Embedded, which is NT. Windows CE was used in the Sega Dreamcast along with Sega's own proprietary OS for the console.

Windows Lifecycle Policy

Microsoft has stopped releasing updates and hotfixes for many old Windows operating systems, including all versions of Windows 9x, and earlier versions of Windows NT. Windows versions prior to XP are no longer supported, with the exception of Windows 2000, which is currently in the Extended Support Period, that will end on July 13, 2010. No new updates are created for unsupported versions of Windows.

DESIGN ISSUES

WINDOWS RESOURCE MANAGEMENT

This has do with the management of all the resources in the component of the Operating System e. g the Memory, the Central Processing Unit, I/O devices and other components.

- ✓ **PROCESS MANAGEMENT**

Processor

A circuit designed to automatically perform lists of logical and arithmetic operations.
Unlike microprocessors, processors may be designed from discrete components rather than be a monolithic integrated circuit.

A process is running program containing one or more threads. A process encapsulates the protected memory and environment for its threads.

Processes and Threads

In addition to being a preemptive multitasking operating system, Windows NT is also multithreaded, meaning that more than one thread of execution (or ***thread***) can execute in a single task at once.

A process comprises:

- A private memory address space in which the process's code and data are stored.
- An access token against which Windows NT makes security checks.
- System resources such as files and windows (represented as object handles).
- At least one thread to execute the code.

A thread comprises:

- A processor state including the current instruction pointer.
- A stack for use when running in user mode.
- A stack for use when running in kernel mode.

Since processes (not threads) own the access token, system resource handles, and address space, threads do NOT have their own address spaces nor do they have their own access token or system resource handles. Therefore, all of the threads in a process SHARE the same memory, access token, and system resources (including quota limits) on a "per-process" rather than a "per-thread" basis. In a multithreaded program, the programmer is responsible for making sure that the different threads don't interfere with each other by using these shared resources in a way that conflicts with another thread's use of the same resource. (As you might suspect, this can get a little tricky.)

Why Use Multithreading?

Multithreading provides a way to have more than one thread executing in the same process while allowing every thread access to the same memory address space. This allows very fast communication among threads. Threads are also easier to create than processes since they don't require a separate address space.

Inside Windows NT, processes and threads are represented as objects that are created, maintained, and destroyed by the Process Manager. These Process Manager process and thread objects contain simpler kernel process and thread objects.

Some typical examples of the use of multiple threads are using a background thread to print a document in a word processor and to recalculate a spreadsheet. When a new thread is created to do these tasks, the main thread can continue responding to user input. A single-threaded application can't respond to user input until it's done printing or recalculating or whatever.

On a uniprocessor platform, the use of multiple threads allows a user to continue using a program even while another thread is doing some lengthy procedure. But only one thread executes at a time.

On a multiprocessor platform, more than one processor may be running different threads in the same process. This has the potential for very significantly speeding up the execution of your program.

Sharing A Single Address Space--Synchronizing Access To Data

Running each process in its own address space had the advantage of reliability since no process can modify another process's memory. However, **all of a process's threads run in the same address space and have unrestricted access to all of the same resources, including memory.** While this makes it easy to share data among threads, it also makes it easy for threads to step on each other. As mentioned before, multithreaded programs must be specially programmed to ensure that threads don't step on each other.

A section of code in Windows operating system that modifies data structures shared by multiple threads is called a ***critical section***. It is important than when a critical section is running in one thread that no other thread be able to access that data structure. Synchronization is necessary to ensure that only one thread can execute in a critical section at a time. This synchronization is accomplished through the use of some type of Windows synchronization object. Programs use Windows synchronization objects rather than writing their own synchronization both to save coding effort and for efficiency: when you wait on a Windows synchronization object, you do NOT use any CPU time testing the object to see when it's ready.

Windows provides a variety of different types of synchronization objects that programs can use to coordinate threads' access to shared data structures. Synchronization objects remember their states and can be set and tested in one uninterruptable step. They also cause the thread to be suspended while waiting on an object and to automatically restart when the other thread signals that it's done.

During the initialization of a program, the program creates a synchronization object for each data structure or object that will be shared among threads.

EVERY critical section will have the following structure:

1. Wait on the synchronization object before accessing the data structure. The Windows waiting API insures that your thread is suspended until the synchronization object becomes unlocked. As soon as the synchronization object becomes unlocked, Windows sets the synchronization object to "locked" and restarts your thread.
2. Access the data structure. (This is the critical section.)
3. Unlock the synchronization object so that the data can be accessed by other threads.

The first step is critical because if it's omitted then any thread can access the data structure while you're accessing. The last step is also critical--it it's omitted, then no thread will be able to access the data even after you're done.

Using this technique on every critical section insures that only one thread can access the data at a time.

The Life Cycle Of A Thread

Each thread has a ***dispatcher state*** that changes throughout its lifetime.

The most important dispatcher states are:

- Running: only one thread per processor can be running at any time.
- Ready: threads that are in the Ready state may be scheduled for execution the next time the kernel dispatches a thread. Which Ready thread executes is determined by their priorities.
- Waiting: threads that are waiting for some event to occur before they become Ready are said to be waiting. Examples of events include waiting for I/O, waiting for a message, and waiting for a synchronization object to become unlocked.

SCHEDULLING

The Kernel's Dispatcher

The kernel's dispatcher performs scheduling and context switching.

Thread scheduling is the act of determining which thread runs on each processor at a given time.

Context switching is the act of saving one thread's volatile state (CPU register contents) and restoring another thread's state so it can continue running where it previously left off.

How Thread Priorities Affect Scheduling

The kernel's dispatcher schedules threads to run based a 32-level priority scheme. Windows guarantees that the threads that are ready that have the highest priority will be running at any given time. (That's one thread on a single-processor system.) Threads with a priority of 31 will be run before any others, while threads with a priority of 0 will run only if no other threads are ready. The range of priorities is divided in half with the upper 16 reserved for real-time threads and the lower 16 reserved for variable priority threads.

Real-time threads run at the same priority for their entire lifetime. They are commonly used to monitor or control systems that require action to be taken at very precise intervals. These threads run at higher priorities than all variable priority threads, which means that they must be used sparingly.

Variable priority threads are assigned a base priority when they are created. (A thread's base priority is determined by the process to which the thread belongs.) The priority of such threads can be adjusted dynamically by the kernel's dispatcher. A thread's dynamic priority can vary up to two priority levels above or below its base priority.

The dispatcher maintains a ***priority queue*** of ready tasks. When prompted to reschedule, it changes the state of the highest priority task to Standby. When the conditions are right, a context switch is performed to begin the thread's execution and the thread goes into the Ready state.

Lower priority threads will always be preempted when a higher priory thread enters the ready state. This is true even if the lower priority thread has time remaining in its quantum, or if the lower priority thread is running on a different processor.

Performance Tuning

In order to get the computer system to perform as users expect, Windows changes the priorities of threads over time.

Each process has a base priority. Threads in a process can alter their base priority by up to two levels up or down.

Depending on the type of work the thread is doing, Windows may also adjust the thread's dynamic priority upwards from its base priority. For instance:

- Threads that are waiting for input get a priority boost, as do threads in the foreground process. This makes the system responsive to the user.
- Threads get a priority boost after completing a voluntary wait.
- All threads periodically get a priority boost to prevent lower priority threads from holding locks on shared resources that are needed by higher priority threads.
- Compute-bound threads get their priorities lowered.

Scheduling On Multiprocessor Systems

A ***multiprocessing*** operating system is one that can run on computer systems that contain more than one processor. Windows is a ***symmetric multiprocessing (SMP)*** system, meaning that it assumes that all of the processors are equal and that they all have access to the same physical memory. Therefore, Windows can run any thread on any available processor regardless of what process, user or Executive, owns the thread.

There are also asymmetric multiprocessing (ASMP) systems in which processors are different from each other--they may address different physical memory spaces, or they may have other differences. These operating systems only run certain processes on certain processors--for instance, the kernel might always execute on a particular processor.

The design of Windows supports ***processor affinity***, whereby a process or thread can specify that it is to run on a particular set of processors, but this facility isn't supported in the first release.

Windows uses the same rules for scheduling on a multiprocessor system as it does on a single processor system, so at any given time the threads that are ready and have the highest priorities are actually running.

Using Task Manager

The *Task Manager* utility shows the applications and processes that are currently running on your computer, as well as CPU and memory usage information. To access Task Manager, press

CtrḷAḷtDelete and click the Task Manager button. Alternatively, right-click an empty area in the Taskbar and select Task Manager from the pop-up menu.
The Task Manager dialog box has four main tabs: Applications, Processes, Performance, and Networking. These options are covered in the following subsections.

Managing Application Tasks

The Applications tab of the Task Manager dialog box, shown in **Figure iii**, lists all of the applications that are currently running on the computer. For each task, you will see the name of the task and the current status (running, not responding, or stopped).
To close an application, select it and click the End Task button at the bottom of the dialog box. To make the application window active, select it and click the Switch To button. If you want to start an application that isn't running, click the New Task button and specify the location and name of the program you wish to start.

- Processor Scheduling, which allows you to optimize the processor time for running programs
  or background services
- Memory Usage, which allows you to optimize memory for programs or system cache
- Virtual Memory, which is used to configure the paging file

Stopping Processes

Process	**Description**
System Idle Process	A process that runs when the processor is not executing any other threads
smss.exe	Session Manager subsystem
csrss.exe	Client-server runtime server service
mmc.exe	Microsoft Management Console program (used to track resources used by MMC snap-ins such as System Monitor)
explorer.exe	Windows Explorer interface
Ntvdm.exe	MS-DOS and Windows 16-bit application support

Managing Process Priority

You can manage process priority through the Task Manager utility or through the start command-line utility. To change the priority of a process that is already running, use the Processes tab of Task Manager. Right-click the process you want to manage and select Set Priority from the pop-up menu. You can select from RealTime, High, AboveNormal, Normal, BelowNormal, and Low priorities.

Options for the start Command-Line Utility

Option Description

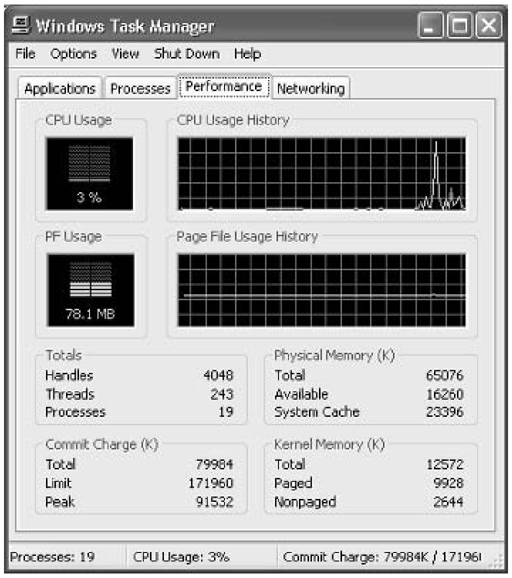
- low Starts an application in the Idle priority class.
- normal Starts an application in the Normal priority class.
- high Starts an application in the High priority class.
- realtime Starts an application in the RealTime priority class.
- abovenormal Starts an application in the AboveNormal priority class.
- belownormal Starts an application in the BelowNormal priority class.
- min Starts the application in a minimized window.
- max Starts the application in a maximized window.
- separate Starts a Windows 16-bit application in a separate memory space. By default Windows 16-bit applications run in a shared memory space, NTVDM, or NT Virtual DOS Machine.
- shared Starts a DOS or Windows 16-bit application in a shared memory space.

Managing Performance Tasks

The Performance tab shows the following information:

- CPU Usage, in real time and in a history graph
- Page File Usage, in real time and in a history graph
- Totals for handles, threads, and processes
- Physical Memory statistics
- Commit Charge memory statistics

- Kernel Memory statistics



Totals

Handles	4048
Threads	243
Processes	19

Physical Memory (K)

Total	65076
Available	16260
System Cache	23396

Commit Charge (K)

Total	79984
Limit	171960
Peak	91532

Kernel Memory (K)

Total	12572
Paged	9928
Nonpaged	2644

Processes: 19 CPU Usage: 3% Commit Charge: 79984K / 17196

A task manager windows

Scheduling Tasks

Windows includes a *Task Scheduler* utility that allows you to schedule tasks to occur at specified intervals. You can set any of your Windows programs to run automatically at a specific time and at a set interval, such as daily, weekly, or monthly. For example, you might schedule your Windows Backup program to run daily at 2:00 a.m.

Tuning and Upgrading the Processor

If you suspect that you have a processor bottleneck, you can try the following solutions:

- Use applications that are less processor-intensive.
- Upgrade your processor.
- If your computer supports multiple processors, add one. Windows can support up to two processors, which will help if you use multithreaded applications. You can also use processor affinity to help manage processor-intensive applications.

Monitoring and Optimizing the Processor

Processor bottlenecks can develop when the threads of a process require more processing cycles than are currently available. In this case, the process will wait in a processor queue and system responsiveness will be slower than if process requests could be immediately served.
The most common causes of processor bottlenecks are processor-intensive applications and other subsystem components that generate excessive processor interrupts (for example, disk or network subsystems).
In a workstation environment, processors are usually not the source of bottlenecks. You should still monitor this subsystem to make sure that processor utilization is at an efficient level.

MEMORY MANAGEMENT

The memory manager implements virtual memory, provides a core set of services such as memory mapped files, copy-on-write memory, large memory support, and underlying support for the cache manager.

Memory management in Microsoft Windows operating systems has evolved into a rich and sophisticated architecture, capable of scaling from the tiny embedded platforms (where Windows executes from ROM) all the way up to the multi-terabyte NUMA configurations, taking full advantage of all capabilities of existing and future hardware designs.

With each release of Windows, memory management supports many new features and capabilities. Advances in algorithms and techniques yield a rich and sophisticated code base, which is maintained as a single code base for all platforms and SKUs.

Memory management improvements in Windows Vista focused on areas such as dynamic system address space, enhanced NUMA and large system/page support, advanced video model support, I/O and section access, and robustness and diagnosability.

Among other things, a multiprogramming operating system kernel must be responsible for managing all system memory which is currently in use by programs. This ensures that a program does not interfere with memory already used by another program. Since programs time share, each program must have independent access to memory.

Cooperative memory management, used by many early operating systems assumes that all programs make voluntary use of the kernel's memory manager, and do not exceed their allocated memory.

Memory protection enables the kernel to limit a process' access to the computer's memory. Various methods of memory protection exist, including memory_segmentation and paging. All methods require some level of hardware support (such as the 80286 MMU) which doesn't exist in all computers.

Virtual memory

The use of virtual memory addressing (such as paging or segmentation) means that the kernel can choose what memory each program may use at any given time, allowing the operating system to use the same memory locations for multiple tasks.

If a program tries to access memory that isn't in its current range of accessible memory, but nonetheless has been allocated to it, the kernel will be interrupted in the same way as it would if the program were to exceed its allocated memory. When the kernel detects a page fault it will generally adjust the virtual memory range of the program which triggered it, granting it access to the memory requested. This gives the kernel discretionary power over where a particular application's memory is stored, or even whether or not it has actually been allocated yet.

In modern operating systems, application memory which is accessed less frequently can be temporarily stored on disk or other media to make that space available for use by other programs. This is called swapping, as an area of memory can be used by multiple programs, and what that memory area contains can be swapped or exchanged on demand.

✓ ERROR HANDLING

Existing operating systems include error handling tools that identify and log errors that occur during the operation of the operating systems and provide a user with the ability to view the logged errors. Although most applications running in the operating system's environment provide error messages when errors occur, these error messages are often technical in nature and may not be easily understood by the user. Therefore, the error handling tools not only identify and log these errors but, in addition, they describe the errors to the user in an easily comprehensible format. For example, the Windows NT® operating system by Microsoft Corporation includes an event log that collects error messages from applications, device drivers and the operating system itself to a common location that the user can access. The Windows NT® operating system also includes an event viewer that permits the user to view the error messages in the order of their occurrences.

For existing error handling tools, the client application must have advance knowledge of the possible types of service failures. Otherwise, the client application would not be able to provide meaningful information regarding the nature of the error. This poses problems in a client-server environment because the client application and applications of the service providers are loosely coupled and developed independently, and it is not practical to change an application as its underlying service provider evolves. It is, therefore, difficult for a client application to obtain meaningful information for an error message within a client/server environment.

The present invention resolves the above problems by providing a mechanism for a client application that identifies the source of a service error and obtains detailed error information at the time the error occurs. In particular, the mechanism accesses the error message information specific to the service that encounters the error and, then, reports that information as part of its error handling operation. The present invention also functions in a plurality of nested subsystems in which a subsystem that detects a particular problem is identified among a group of subsystems that are called in a nested manner. Accordingly, the present invention automatically handles the operation of reporting events whose source subsystem is different from that of the reporting subsystem without requiring the reporting subsystem to have advance knowledge of the possible errors that could be encountered.

Using Event Viewer

You can use the *Event Viewer* utility to track information about your computer's hardware and software, as well as to monitor security events. Every Windows XP computer will show three types of log files (depending on your configuration you may also have other log files):

System log

Tracks events related to the Windows XP operating system. This log is useful in troubleshooting Windows XP problems.

Security log

Tracks events related to Windows XP auditing. By default auditing is not enabled.

If you enable security auditing you can select what to track and whether you will track security success and/or failure events.

Application log

Tracks events related to applications that are running on your computer. For example, you might see that your e-mail program recorded an error. These errors are useful in troubleshooting application problems or for developers to fix application problems.

INTERRUPT

Interrupts are central to operating systems as they provide an efficient way for the operating system to interact and react to its environment. The alternative is to have the operating system "watch" the various sources of input for events (polling) that require action -- not a good use of CPU resources. Interrupt-based programming is directly supported by most CPUs. Interrupts provide a computer with a way of automatically running specific code in response to events. Even very basic computers support hardware interrupts, and allow the programmer to specify code which may be run when that event takes place.

When an interrupt is received the computer's hardware automatically suspends whatever program is currently running, saves its status, and runs computer code previously associated with the interrupt. This is analogous to placing a bookmark in a book when someone is interrupted by a phone call and then taking the call. In Windows operating systems interrupts are handled by the operating system's kernel. Interrupts may come from either the computer's hardware or from the running program.

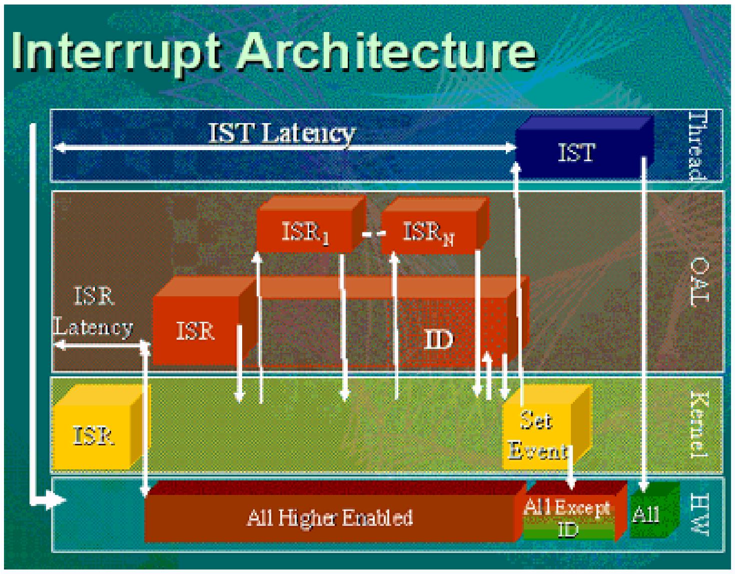
kernel is the core process of a preemptive operating system, consisting of a multitasking scheduler and the basic security services. Depending on the operating system, other services such as virtual memory drivers may be built into the kernel. The kernel is responsible for managing the scheduling of threads and processes.

When a hardware device triggers an interrupt the operating system's kernel decides how to deal with this event, generally by running some processing code. How much code gets run depends on the priority of the interrupt (for example: a person usually responds to a smoke detector alarm before answering the phone). The processing of hardware interrupts is a task that is usually delegated to software called device drivers, which may be either part of the operating system's kernel, part of another program, or both. Device drivers may then relay information to a running program by various means.

A program may also trigger an interrupt to the Windows operating system. If a program wishes to access hardware for example, it may interrupt the operating system's kernel, which causes control to be passed back to the kernel. The kernel will then process the request. If a program wishes additional resources (or wishes to shed resources) such as memory, it will trigger an interrupt to get the kernel's attention.

Interrupt Architecture

The first step in exploring the interrupt architecture of Microsoft® Windows® is defining an overall model of the hardware, kernel, OAL and thread interactions during an interrupt. The following diagram is an overall picture of these different levels of responsibility and the transitions that cause changes of state.



The diagram represents the major transitions during an interrupt with time increasing to the right of the diagram. The bottom most layer of the diagram is the hardware and the state of the interrupt controller. The next layer is the kernel interactions during interrupt servicing. The OAL describes the board support package (BSP) responsibilities. The top most layer represents the application or driver thread interactions needed to service an interrupt. The diagram represents the interactions during a single interrupt; representing the new ability of Windows to have shared interrupts. The activity starts with an interrupt represented by the line at the left most section of the chart. An exception is generated causing the kernel ISR vector to be loaded onto the processor. The kernel ISR interacts with the hardware disabling all equal and lower priority interrupts on all processors except for the ARM and Strong ARM architectures. The kernel then vectors to the OAL ISR that has been registered for that particular interrupt. The OAL ISR then can either directly handle the interrupt or can use NKCallIntChain to walk a list of installed ISRs. The main ISR or any of the installed ISRs then performs any work and returns the mapped interrupt called SYSINTR for that device. If the ISR determines that its associated device is not causing the interrupt the ISR returns SYSINTR_CHAIN, which causes NKCallIntChain( ) to walk the ISR list to the next interrupt in the chain. The ISRs are called in the order that they were installed creating a priority on the calling list.

Microsoft has advanced the Windows interrupt architecture. The ability of the OS to deal with shared interrupts greatly extends the ability of Windows to support many interrupt architectures.

Knowledge of this interrupt architecture greatly speeds up the investigation times into driver and latency issues. A Windows model of the operating system interaction is the key to this understanding. Shared interrupts have greatly increased the openness of Windows supporting the platform provider and application developer scenarios that are pervasive from company to company or within companies. Understanding latency sources will help in diagnosis of driver and real-time issues. The interrupt structure in Windows is well defined and understandable.

SECURITY

Security has been a hot topic with Windows for many years, and even Microsoft itself has been the victim of security breaches.[citation needed] Consumer versions of Windows were originally designed for ease-of-use on a single-user PC without a network connection, and did not have security features built in from the outset.

Windows NT and its successors are designed for security (including on a network) and multi-user PCs, but were not initially designed with Internet security in mind as much since, when it was first developed in the early 1990s, Internet use was less prevalent.

These design issues combined with flawed code (such as buffer overflows) and the popularity of Windows means that it is a frequent target of computer worm and virus writers.

Microsoft releases security patches through its Windows Update service approximately once a month (usually the second Tuesday of the month), although critical updates are made available at shorter intervals when necessary. In Windows 2000 (SP3 and later), Windows XP and Windows Server 2003, updates can be automatically downloaded and installed if the user selects to do so.

Use these steps to develop a strong password:

- Think of a sentence that you can remember. This will be the basis of your strong password or pass phrase. Use a memorable sentence, such as "My son Aiden is three years old."
- Check if the computer or online system supports the pass phrase directly. If you can use a pass phrase (with spaces between characters) on your computer or online system, do so.

- If the computer or online system does not support pass phrases, convert it to a password. Take the first letter of each word of the sentence that you've created to create a new, nonsensical word. Using the example above, you'd get: "msaityo".

- Add complexity by mixing uppercase and lowercase letters and numbers. It is valuable to use some letter swapping or misspellings as well. For instance, in the pass phrase above, consider misspelling Aiden's name, or substituting the word "three" for the number 3. There are many possible substitutions, and the longer the sentence, the more complex your password can be. Your pass phrase might become "My SoN Ayd3N is 3 yeeRs old." If the computer or online system will not support a pass phrase, use the same technique on the shorter password. This might yield a password like "MsAy3yo".
- Finally, substitute some special characters. You can use symbols that look like letters, combine words (remove spaces) and other ways to make the password more complex. Using these tricks, we create a pass phrase of "MySoN 8N i$ 3 yeeR$ old" or a password (using the first letter of each word) "M$8ni3y0".
- Test your new password with Password Checker Password Checker is a non-recording feature on this Web site that helps determine your password's strength as you type.

✓ **<u>4 steps to protect your computer</u>**

Step 1. Keep your firewall turned on

What is a firewall?

A firewall helps protect your computer from hackers who might try to delete information, crash your computer, or even steal your passwords or credit card numbers. Make sure your firewall is always turned on.

Step 2. Keep your operating system up-to-date

What are operating system updates?

High priority updates are critical to the security and reliability of your computer. They offer the latest protection against malicious online activities. Microsoft provides new updates, as necessary, on the second Tuesday of the month.

Step 3. Use updated antivirus software

What is antivirus software?

Viruses and spyware are two kinds of usually malicious software that you need to protect your computer against. You need antivirus technology to help prevent viruses, and you need to keep it regularly updated.

Step 4. Use updated antispyware technology

What is antispyware software?

Viruses and spyware are two kinds of usually malicious software that you need to protect your computer against. You need antispyware technology to help prevent spyware, and you need to keep it regularly updated.

Some threats to the security of Windows are;

I. ***Worms***
II. ***Spyware***
III. ***Viruses***

- ✓ **Security in Microsoft Windows**

While the Windows 9x series offered the option of having profiles for multiple users, they had no concept of access privileges, and did not allow concurrent access; and so were not true multi-user operating systems. In addition, they implemented only partial memory protection. They were accordingly widely criticised for lack of security.

The Windows NT series of operating systems, by contrast, are true multi-user, and implement absolute memory protection. However, a lot of the advantages of being a true multi-user operating system were nullified by the fact that, prior to Windows Vista, the first user account created during the setup process was an administrator account, which was also the default for new accounts. Though Windows XP did have limited accounts, the majority of home users did not change to an account type with fewer rights – partially due to the number of programs which unnecessarily required administrator rights – and so most home users ran as administrator all the time.

Windows Vista changes this by introducing a privilege elevation system called User Account Control. When logging in as a standard user, a logon session is created and a token containing only the most basic privileges is assigned. In this way, the new logon session is incapable of making changes that would affect the entire system.

When logging in as a user in the Administrators group, two separate tokens are assigned. The first token contains all privileges typically awarded to an administrator, and the second is a restricted token similar to what a standard user would receive. User applications, including the Windows Shell, are then started with the restricted token, resulting in a reduced privilege environment even under an Administrator account. When an application requests higher privileges or "Run as administrator" is clicked, UAC will prompt for confirmation and, if consent is given (including administrator credentials if the account requesting the elevation is not a member of the administrators group), start the process using the unrestricted token.

Windows Defender

On January 6, 2005, Microsoft released a beta version of Microsoft AntiSpyware, based upon the previously released Giant AntiSpyware. On February 14, 2006, Microsoft AntiSpyware became Windows Defender with the release of beta 2. Windows Defender is a freeware program designed to protect against spyware and other unwanted software. Windows XP and Windows Server 2003 users who have genuine copies of Microsoft Windows can freely download the program from Microsoft's web site, and Windows Defender ships as part of Windows Vista.

File Permissions

At windows version from window NT3 have been based on a file system permission system referred to as AGLP (Account, Global, Local, and Permission). In essence, where file permissions are applied to the file and folder in the form of a local group, which then has other global group.

CHAPTER NINE – OPERATING SYSTEM SECURITY

The fact that an operating system is computer software makes it prone to error just as any human creation. Programmers make mistakes, and inefficient code is often implemented into programs even after testing. Some developers perform more thorough testing and generally produce more efficient software. Therefore, some operating systems are more error prone while others are more secure.

1.1 Security in Computer

The branch of computer technology known as information security as applied to computers and networks is the computer security. The objective of computer security includes protection of information and property from theft, corruption, or natural disaster, while allowing the information and property to remain accessible and productive to its intended users. The term computer system security means the collective processes and mechanisms by which sensitive and valuable information and services are protected from publication, tampering or collapse by unauthorized activities or untrustworthy individuals and unplanned events respectively.

The technologies of computer security are based on logic. As security is not necessarily the primary goal of most computer applications, designing a program with security in mind often imposes restrictions on that program's behavior.
Security in this regard could be referred to as the protection mechanism used to safeguard information in the computer. Security has many facets and the three of the more important ones are

1. The threats
2. The nature of intruders
3. Accidental data loss

1. **Threats**: From security perspective, computer systems have three general goals, with corresponding threats.

(i) Data Confidentiality: Concerned with having secret data remain secret meaning if the owner of some data has decided that these data are available to certain people and no others, the system should guarantee that release of the data to unauthorized people does not occur.

(ii) Data Integrity: Means that unauthorized users should not be able to modify any data without the owner's permission. If a system cannot guarantee that data deposited in it remain unchanged until the owner decides to change them, it is not worth much as an information system.

(iii) System Availability: Means that nobody can disturb the system to make it usable. If a computer is an internet server, sending a flood of requests to it may cripple it by eating up all of its CPU time just examining and discarding incoming requests. Another aspect of security problem in operating system is Privacy that is protecting individuals from misuse of information about them.

(2) **The nature of intruders:** In the security literature, people who are nosing around places where they have no business being are called ***intruders*** or sometimes called ***adversaries.*** Intruders act in two different ways

1 ***Passive Intruders****:* just want to read files they are not authorised to read.

2 ***Active Intruders***: are more malicious; they want to make unauthorised changes to data.

When designing a system to be secure against intruders, it is important to keep in mind the kind of intruder one is trying to protect against. Some categories are

(a) Casual prying by nontechnical users.

(b) Snooping by insiders.

(c) Determined attempts to make money.

(d) Commercial or military espionage.

Another category of security pest that has manifested itself in recent times is the ***Virus***. The term virus is a piece of code that replicates itself and usually does some damage.

(3) **Accidental data loss:** In addition to threats caused by malicious intruders, valuable data can be lost by accident. Some of the accidental data losses are

(1) ***Acts of God***: fires, flood, earthquakes, wars, etc.

(2) ***Hardware or software errors***: CPU malfunction, unreadable disks or tapes, telecommunication errors, program bugs.

(3) ***Human errors:*** incorrect data entry, wrong tape or disk mounted, wrong program run, lost disk or tape, or some other mistake.Most of these can be dealt with by maintaining adequate backups, preferably far away from the original data.

1.2 Operating system design, security and complexity

An operating system is a software component that acts as the core of a computer system. It performs various functions and is essentially the interface that connects your computer and its supported components. Various operating systems are in use today to satisfy the ever changing customer demands. Nevertheless the most widespread operating systems are: **Microsoft Windows, Linux/Unix and Macintosh**. **Windows** are mostly used as personal computers, **Linux/Unix** are mainly open source while **Apple Macs** are often used for graphic designs or other specialist applications. Irrespective of their application or use, all operating systems up to date have been subject to security compromises or likewise failures. It is a fact that the majority of hacking tools, viruses, worms or Trojan horses are written for Windows, but this is merely due to the fact that Windows occupy almost 90% of the global market.

The security of the operating system is therefore a necessity for the overall system security. Today most commercially developed operating systems provide security through authentication of the users, maintenance of access control mechanisms, separation of kernel and user spaces and providing trusted applications to modify or manage system resources. However the above security features are inadequate to protect the operating system from attacks in today's environment.

SECURITY ISSUES IN THE LINUX OPERATING SYSTEMS

While companies like Microsoft and Apple own the commercial software market, a variety of non profit organizations or intellectual individuals contribute constantly to the proliferation of the ***open source software***. In terms of operating systems, Linux is the example par excellence of all free ***open source systems***, Introduced by **Linux Torvalds**, at the time a student. Linux was the first fully functional operating system that was offered for free under the ***open source*** agreement to the public. Following this event, with the participation of thousand of admirers across the globe, a myriad of ***open source*** Linux based operation systems flooded the cyber world.

There are different reasons that motivate thousands of people around the globe to participate in open source projects and release software to the public. Intellectual gratification, pleasure of creativity or of solving complex and challenging tasks, are of some the driving forces in this domain. Whatever the reasons, the benefits of using open source are manifold. Open-source software powers many of the web sites on the Internet, corporate computer, servers used for research and development, it can be found in digital video recorders , telephones, personal digital assistants (PDAs) watches, networking hardware, MP3 players and automobiles

Nevertheless, open source software does not come without issues or disadvantages. Even though the codes are available to thousands of eyes for scrutiny, there is no guarantee for security or optimal performance. Although that is ok for simple home applications it is not the case for enterprise, commercial or critical applications. Also because of lack of standardization and complex licensing issues, open source software is prone to misuse or abuse. Standardization is hard to achieve, because open source creators are completely free in their choice of design, implementation or adherence to existing standards. Usually standardization is enforced by market forces and industry regulators; however in the case of open source software both these factors do not exercise enough pressure to drive the process. Version proliferation is another major open source issue. As a matter of fact, this matter does not concern only open source software but commercial software as well. Nevertheless, the effect on the open source software is more evident. Developing many versions of a program in a short period not only confuses users but also requires a steep learning curve. This is true in particular in the case of constant changes of graphical user interfaces and navigation concepts from one version to the next one. At the same time, constant introductions of new versions of a software package do not affect in positive way its reputation since the user might believe this is a sign of instability.

Another issue that affects ICT today is that of the implementation of open source software. Because, open source is developed by the co-operation of different individuals, it is hard to establish a proper working relationship with the person in charge (if there is any). Furthermore, technical support, documentation issues, no access to advice, are some of the problems that a company or individual that uses open source software might face. Hardware and software compatibilities influence also the processes ever since most of hardware manufacturers do not expose their trade secrets, therefore not allowing access to their codes to open source developers. Open source developers have no other choice but to design and release their own code (hardware drivers) therefore contributing to the complexity and lack of standardization.

Future prognosis on the Linux operating system

The open source phenomenon is definitely influencing in a positive way ICT and probably the trend will not change in the future. Open source projects are available to "millions of eyes" for scrutiny, improvement or testing. Nevertheless, it is likely that in the future will continue to experience the same issues mentioned above with some improvements in the area of standardization. Some open source will definitely transform in commercial software provided that they have matured enough and captured a significant market share. Red Hat Linux for example, is a typical example of how open source software becomes commercial under the right circumstances.

SECURITY ISSUES IN THE WINDOWS OPERATING SYSTEM

Security is the main problem that Windows operating systems are facing since their introduction. Lack of vision from its developers regarding security is probably the main reason behind this issue. The first windows were designed to be simple and productive but not very secure. Although new operating systems versions were introduced within the last 5-10 years, the same issues with security persisted. In my opinion, because of market pressure and product development circles, it was almost impossible for Microsoft to totally change their operating system approach. Instead they continued to build on top of each previous model. Unfortunately, their operating systems are still vulnerable affecting significantly ICT applications worldwide.

To understand the impact of operating system vulnerabilities on ICT suffice to look at the case of “SQL slammer’, a worm released on the web in 2003 (Forte, 2003). Slammer, also known as “SQL hell”, is a worm that affected Microsoft Windows operating systems in January 2003 affecting within ten minutes 75 thousands machines worldwide. Slammer exploited vulnerability in SQL server and desktop engine slowing down communications and affecting businesses financially worldwide. Following this incident several modified slammer versions were released online. This is not an isolated episode that demonstrates the lack of security vision and poor operating system design. Viruses and worms like Melissa, code red, sasser, nimda, donut, spida or slapper have also impacted information communication telecommunications globally. In 2007 Computer Economics, a well known research company conducted a research on the impact of the malware globally estimating a $ 13 billion in financial losses in 2006 only. It is quite obvious how ICT and global communications are affected by malware which attributes its success to operating system vulnerabilities.

Microsoft has been able to take care of malware attacks by releasing patches or services packs. Although, it looks like this is the right approach to this problem it does eradicate the problem and provide temporarily relief from threats. We need a holistic approach to deal with the root of the problem not with it consequences. As a matter of fact operating system patches manage to avoid the threat temporarily, since what they usually do is a mere change of names or locations of important operating system files used by malware. In other occasions Microsoft has even discontinued shipping certain programs with its operating systems as a security measure against malware, therefore not dealing with the main problems: design and security.

In 2007, Microsoft released officially to the public Windows Vista and in October 2009 Windows seven. Although, the graphical user interfaces look impressive, both operating systems are still vulnerable to malware or system hacking. A very simple example is the 'the sticky key backdoor', one of Vista's vulnerabilities. Vinoo Thomas, a McAfee researcher, in 2007 released a blog online informing the public about the Sticky Keys vulnerability. Vista apparently does allow the modification of sethc.exe file (located at: C:/windows/system32/sethc.exe) and no integrity checks are performed before execution. **Authentication** can simply be bypassed by replacing this file with cmd.exe using a live CD like Backtrack or direct logging and entering windows explorer (Vinoo, 2007).

Moreover, Vista activation mechanism has been broken almost one year after its official release. The same security scenario applies to Windows seven. This operating system can be bypassed in the same way as Windows Vista (using the installation disk and entering recovery mode). Furthermore online news of a zero day attack is spreading around. Certainly, this poor security performance of Microsoft Windows, the most used operating system worldwide, does not sound promising for ICT and its future.

Future prognosis on the windows operating system

Operating system design is a factor that will influence ICT in the times to come. The main reason is security. If we take in consideration the fact that digital globalization is facilitating the distribution of malware and the number of internet users is rising exponentially, we should expect more sophisticated attacks on the windows operating systems and ICT. Under these circumstances, we should review the windows operating system design strategy, focusing on security and build reliable operating system.

SECURITY ISSUES IN THE UNIX OPERATING SYSTEMS

There are a number of elements that have lead to the popularity of the UNIX operating system in the world today. The most notable factors are its portability among hardware platforms and the interactive programming environment that it offers to users. In fact, these elements have had much to do with the successful evolution of the UNIX system in the commercial market place. As the UNIX system expands further into industry and government, the need to handle UNIX system security will no doubt become imperative. For example, the US government is committing several million dollars a year for the UNIX system and its supported hardware. The security requirements for the government are tremendous, and one can only guess at the future needs of security in industry.
In this paper, we will cover some of the more fundamental security risks in the UNIX system. Discussed are common causes of UNIX system compromise in such areas as file protection, password security, networking and hacker violations. In our conclusion, we will comment upon ongoing effects in UNIX system security and their direct influence on the portability of the UNIX operating system.

In the UNIX operating system environment, files and directories are organized in a tree structure with specific access modes. The setting of these modes through permission bits (as octal digits), is the basis of UNIX system security. Permission bits determine how users can access files and the type of access they are allowed. There are three user access modes for all UNIX system files and directories: the owner, the group, and others. Access to read, write and execute within each of the user types is also controlled by permission bits. Flexibility in file security is convenient, but it has been criticized as an area of System security compromise.

FILE AND DIRECTORY SECURITY

In the UNIX operating system environment, files and directories are organized in a tree structure with specific access modes. The setting of these modes, through permission bits (as octal digits), is the basis of UNIX system security. Permission bits determine how users can access files and the type of access they are allowed. There are three user access modes for all UNIX system files and directories: the owner, the group, and others. Access to read, write and execute within each of the user types is also controlled by permission bits (Figure 1). Flexibility in file security is convenient, but it has been criticized as an area of system security compromise.

```
        Permission modes

    OWNER               GROUP              OTHERS

    ----------------------------------------------------------

    rwx         :          rwx         :       rwx

    ----------------------------------------------------------

    r=read  w=write  x=execute



    -rw--w-r-x 1 bob csc532 70 Apr 23 20:10 file

    drwx------ 2 sam A1 2 May 01 12:01 directory
```

FIGURE 1. File and directory modes: File shows Bob as the owner, with read and writes permission. Group has write permission, while others has read and execute permission. The directory gives a secure directory not readable, writeable, or executable by Group and Others.

Since the file protection mechanism is so important in the UNIX operating system, it stands to reason that the proper setting of permission bits is required for overall security. Aside from user ignorance, the most common area of file compromise has to do with the default setting of permission bits at file creation. In some systems the default is octal 644, meaning that only the file owner can write and read to a file, while all others can only read it. (3) In many "open" environments this may be acceptable. However, in cases where sensitive data is present, the access for reading by others should be turned off. The file utility umask does in fact satisfy this requirement. A suggested setting, umask 027, would enable all permission for the file owner, disable write permission to the group, and disable permissions for all others (octal 750). By inserting this umask command in a user .profile or .login file, the default will be overwritten by the new settings at file creation. The CHMOD utility can be used to modify permission settings on files and directories. Issuing the following command,

```
chmod u+rwd,g+rw,g-w,u-rwx file
```

will provide the file with the same protection as the umask above (octal 750). Permission bits can be relaxed with chmod at a later time, but at least initially, the file structure can be made secure using a restrictive umask. By responsible application of such utilities as umask and chmod, users can enhance file system security. The Unix system, however, restricts the security defined by the user to only owner, group and others. Thus, the owner of the file cannot designate file access to specific users. As Kowack and Healy have pointed out, "The granularity of control that (file security) mechanisms is often insufficient in practice (...) it is not possible to grant one user write protection to a directory while granting another read permission to the same directory. (4) A useful file security file security extension to the Unix system might be Multics style access control lists. With access mode vulnerabilities in mind, users should pay close attention to files and directories under their control, and correct permissions whenever possible. Even with the design limitations in mode granularity, following a safe approach will ensure a more secure Unix system file structure.

DIRECTORIES

Directory protection is commonly overlooked component of file security in the Unix system. Many system administrators and users are unaware of the fact, that "publicly writable directories provide the most opportunities for compromising the Unix system security" (6). Administrators tend to make these "open" for users to move around and access public files and utilities. This can be disastrous, since files and other subdirectories within writable directories can be moved out and replaced with different versions, even if contained files are unreadable or unwritable to others. When this happens, an unscrupulous user or a "password breaker" may supplant a Trojan horse of a commonly used system utility' *For example:*

Imagine that the /bin directory is publicly writable. The perpetrator could first remove the old version (with rm utility) and then include his own fake su to read the password of users who execute this utility.

Although writable directories can destroy system integrity, readable ones can be just as damaging. Sometimes files and directories are configured to permit read access by other. This subtle convenience can lead to unauthorized disclosure of sensitive data: a serious matter when valuable information is lost to a business competitor. As a general rule, therefore, read and write access should be removed from all but system administrative directories. Execute permission will allow access to needed files; however, users might explicitly name the file they wish to use. This adds some protection to unreadable and unwritable directories. So, programs like lp file.x in an unreadable directory /ddr will print the contents of file.x, while ls/ddr would not list the contents of that directory.

USER AUTHENTICATION

Another area is the user authentication. In the UNIX system, authentication is accomplished by personal passwords. Though passwords offer an additional level of security beyond physical constraints, they lend themselves to the greatest area of computer system compromise. Lack of user awareness and responsibility contributes largely to this form of computer insecurity. This is true
of many computer facilities where password identification, authentication and authorization are required for the access of resources, and the Unix operating system is no exception. Password information in many time-sharing systems are kept in restricted files that are not ordinarily readable by users. The UNIX system differs in this respect, since it allows all users to have read access to the /etc/passwd file where encrypted passwords and other user information are stored.

DATA ENCRYPTION

Although the Unix system implements a one-way encryption method, and in most systems a modified version of the data encryption standard (DES), password breaking methods are known. Among these methods, brute-force attacks are generally the least effective, yet techniques involving the use of heuristics (good guesses and knowledge about passwords) tend to be successful. For example, the /etc/passwd file contains such useful information as the login name and comments fields. Login names are especially rewarding to the "password breaker" since many users will use login variants for passwords (backward spelling, the appending of a single digit etc.). The comment field often contains items such as surname, given name, address, telephone number, project name and so on. To quote Morris and Grampp in their landmark paper on Unix system security: The authors made a survey of several dozen local machines, using as trial passwords a collection of the 20 most common female first names, each followed by a single digit. The total number of passwords tried was therefore 200. At least one of these 200 passwords turned out to be a valid password on every machine surveyed. If an intruder knows something about the people using a machine, a whole new set of candidates is available. Family and friend's names, auto registration numbers, hobbies, and pets are particularly productive categories to try interactively in the unlikely event that a purely mechanical scan of the password file turns out to be disappointing.

Thus, given a persistent system violator, there is strong evidence, that he will find some information about users in the /etc/passwd file. With this in mind, it is obvious that a password file should be unreadable to everyone except those in charge of system administration.

SECURITY ISSUES IN THE MACINTOSH OPERATING SYSTEM

It's been called one of the safest operating systems of all times, but the Mac's OS X Tiger may not be as safe as it seems. Mac's OS X Tiger has become a favorite among Mac users for its bells and whistles and its UNIX based architecture. From a power user to newbie, Tiger provides both comfort and security for all OS X users. Some of the flaws found in its security is as follows:

- *FAILING TO USE ITS SOFTWARE UPDATE*

Regularly updating Tiger's software is one of the easiest ways to keep your computer protected from the latest exploits and malicious Internet content. In January of this year, a couple of computer guru's published "The Month of Apple Bugs"(MoAB) -- a website dedicated to pointing out 31 of OS X's vulnerabilities and security flaws. After reviewing the website, Apple acted promptly and has since released several updates addressing the critical bugs. With software updates turned off, there's a good chance the computer could fall victim to one of MoAB's exploits.

- *MINDLESSLY SURFING WITH SAFARI*

Although much safer than Microsoft's Internet Explorer, Tiger's default
web browser Safari is not immune to security flaws. To obtain the safest Internet browsing experience, a few of Safari's features should be modified:

Make sure all "AutoFill" options are disabled, and always use "Private
Browsing" on each of the computer's accounts. Although surfing without "Private Browsing" enabled could save you some time, in the long run you're simply opening yourself up to greater security risks.
To be ultra conservative with web browsing, one can disallow all cookies and remove all existing cookies via the "Show Cookies" dialog. (Keeping in mind that some websites require cookies for complete functionality). By not accepting cookies, one may be limiting web browsing experience, so one is torn in between securing his/her system or enjoying the web experience.

❖ *INCORRECTLY CONFIGURING SECURITY PREFERENCES*

Tiger's security panel features a handful of security preferences which permit users to select varying levels of security based upon their particular usage requirements. Obviously, however, when configuring your security preference it's important to understand what each option does, and the benefits of a particular setting: One of the most important and often overlooked preferences is whether to permit automatic login. Requiring a password to wake the computer is imperative in preventing unauthorized access to unattended computers. "Disabling automatic login is necessary for any level of security. If you enable automatic login, an intruder can automatically log in without having to authenticate. Even if you automatically log in with a very restricted user account, this makes it much easier to perform malicious actions on the computer."
In addition to requiring a user to login after the computer has been asleep, it is also important to require an additional login wherever an important system wide preference is being changed. In order to prevent faulty administration, either from a malicious user or just from an unwitting friend who accidentally makes a system change, it is important to require an extra step of authentication when altering system preferences. After all, we can all make mistakes when toggling options, but by requiring an extra authentication step whenever a system preference is changed, you can make sure that many of these types of errors never occur.

- *LEAVING UNUSED HARDWARE DEVICES ENABLED*

Most of us are no longer worried by our Internet connections, but instead connect to the Internet through multiple styles of broadband connections. For example, instead of being tied down to Ethernet cables at home, users are taking advantage of wireless connections using their laptops from anywhere, and connecting their Bluetooth devices to their computer for extra support. While having different types of broadband connections is great for the user, it's awful for the security of the computer. To protect your computer, you should make sure to "disable any unused hardware devices listed in Network preferences. Enabled, unused devices (such as AirPort and Bluetooth) are a security risk." While we're not suggesting that you do away with these great feature altogether, we are saying that when they're not in use, you should turn them off.

- *TROJAN HORSE ALERT*

Recently, a new variant of the Hell Raiser Trojan Horse, which was identified as OSX/HellRTS.D, has been discovered. Experts have analyzed this new variant, and it is detected in the latest MacScan spyware definitions update as HellRaiser Trojan Horse 4.2. MacScan has detected previous variants of this trojan horse since 2005. HellRaiser is a trojan horse that allows complete control of a computer by a remote attacker, giving the attacker the ability to transfer files to and from the infected computer, pop up chat messages on the infected system, display pictures, speak messages, and even remotely restart or shut down the infected machine. The attacker can search through the files on the infected computer, choosing exactly what they want to steal, view the contents of the clipboard, or even watch the user's actions on the infected computer.

In order to become infected, a user must run the server component of the Trojan horse, which can be disguised as an innocent file. The attacker then uses the client component of the Trojan horse to take control of the infected system.

SECURITY ISSUES IN SOLARIS OPERATING SYSTEM

Solaris or Oracle Solaris as it is now known is a UNIX-based operating system introduced by Sun Microsystems in 1992 as the successor to SunOS. The prominent flaw in Solaris operating system is the multiple security vulnerabilities in PostgreSQL Shipped with Solaris 10 which allows the Elevation of Privileges or Denial of Service (DoS)

Multiple Security vulnerabilities affecting the PostgreSQL software shipped with Solaris 10 may allow a local or remote user who has access to the PostgreSQL server to cause a Denial of Service (DoS) to the PostgreSQL instance or the server it runs on (due to excessive resource consumption), or to gain elevated privileges on the server.

Regular Expression Denial-of-Service

(CVE-2007-4772, CVE-2007-6067, CVE-2007-4769):

Three separate issues in the regular expression libraries used by PostgreSQL allow malicious user to initiate a denial-of-service by passing certain regular expressions in SQL queries.

First, users could create infinite loops using some specific regular expressions.

Second, certain complex regular expressions could consume some excessive amounts of memory.

Third, out-of-range backref numbers could be used to crash the backend.

All of these issues have been patched.

DBLink privilege Escalation (CVE-2007-6601):

DBLink functions combined with local trust or ident authentication could be used by malicious user to gain superuser privileges.

This issue has been fixed and does not affect users who have not installed DBLink (an optional module), or who are using password authentication for local access. This same problem was addressed in the previous release cycle.

These issues can occur in the following releases

SPARC (Scalable Processor Architecture) Platform

Solaris 10 PostgreSQL 8.1

Without patch 123590-08

Solaris 10 PostgreSQL 8.2

Without patch 136998-02

X86 Platform

Solaris 10 PostgreSQL 8.1
Without patch 123591-08
Solaris 10 PostgreSQL 8.2
Without patch 136999-02

Solaris 8 and 9 do not ship with PostGreSQL and are not impacted by this issue.
A user exploiting this vulnerability must have an account on the PostgreSQL server.
This issue affects PostGreSQL versions 7.4x prior to 7.4.19, 8.0.x prior to 8.0.15, 8.1.x prior to 8.1.11 and 8.2.x prior to 8.2.6.

CONCLUSION

Information and Communication Technologies (ICT) will provide benefits to our society for years to come. The proliferation of these technologies or their decline will be affected amongst all by security issues on the area of operating system design and security, open source issues, and design complexity. Therefore, designing better operating systems, improving on their security, are some of the challenges for the future. If we take in consideration the fact that digital globalization is facilitating the distribution of malware and the number of internet users is rising exponentially, we should expect more sophisticated attacks on the windows operating systems and ICT. Under these circumstances, we should review the windows operating system design strategy, focusing on security and build reliable operating system.

The open source phenomenon is definitely influencing in a positive way ICT and probably the trend will not change in the future. Open source projects are available to “millions of eyes” for scrutiny, improvement or testing. Nevertheless, it is likely that in the future will continue to experience the same issues mentioned above with some improvements in the area of standardization.

Some open source will definitely transform in commercial software provided that
they have matured enough and captured a significant market share. Red Hat
Linux for example, is a typical example of how open source software becomes
commercial under the right circumstances.

CHAPTER TEN:

DISTRIBUTED OPERATING SYSTEM

An operating system is a program that controls the resources of a computer and provides its user with an interface or a virtual machine that's more convenient to use than bear machine.

To begin with, we use the term distributed system to mean a distributed operating system as opposed to a database system or some distributed application system such as a banking system, another name for a distributed operating system is DIS-**CENTRALIZED OPERATING SYSTEM.**
Example of a centralized (not distributed) operating system are; MS-DOS, UNIX, and CP/M.

A distributed operating system is the one that look to its user like an ordinary centralized operating system but runs on multiple, independent, central processing unit (CPU). The key concept in distributed operating system is the TRANPARENCY. What determine a distributed operating system are the software and not the hardware.

In distributed system, the error can be made to tolerate both hardware and software error but it is the software error and not the hardware that cleans the error when it occurs.

Network OS is used to manage Networked computer systems and create, maintain and transfer files in that Network.
Distributed OS is also similar to Networked OS but in addition to it the platform on which it is running should have high configuration such as more capacity RAM, High speed Processor. The main difference between the DOS and the NOS is the transparent issue: Transparency:

- How aware are users of the fact that multiple computers are being used?

Types of Distributed Operating Systems

- Network Operating Systems
- Distributed Operating Systems

Network-Operating Systems

- Users are aware where resources are located
- Network OS is built on top of centralized OS.
- Handles interfacing and coordination between local OSs.
- Users are aware of multiplicity of machines.

Distributed-Operating Systems

- Designed to control and optimize operations and resources in distributed system.

  - Users are not aware of multiplicity of machines
  - Access to remote resources similar to access to local resources
  - Data Migration – transfer data by transferring entire file, or transferring only those portions of the file necessary for the immediate task
  - Computation Migration – transfer the computation, rather than the data, across the system
- Computation speedup – sub processes can run concurrently on different sites
  - Process Migration – execute an entire process, or parts of it, at different sites
  - Load balancing – distribute processes across network to even the workload
  - Hardware preference – process execution may require specialized processor
  - Software preference – required software may be available at only a particular site
  - Data access – run process remotely, rather than transfer all data locally

EXAMPLES OF DISTRIBUTED OPERATING SYSTEM

- The Cambridge Distributed Computing System
- Amoeba
- The V Kernel
- The Eden Project

There are so many types of distributed operating system but these are chosen based on three criteria which are:

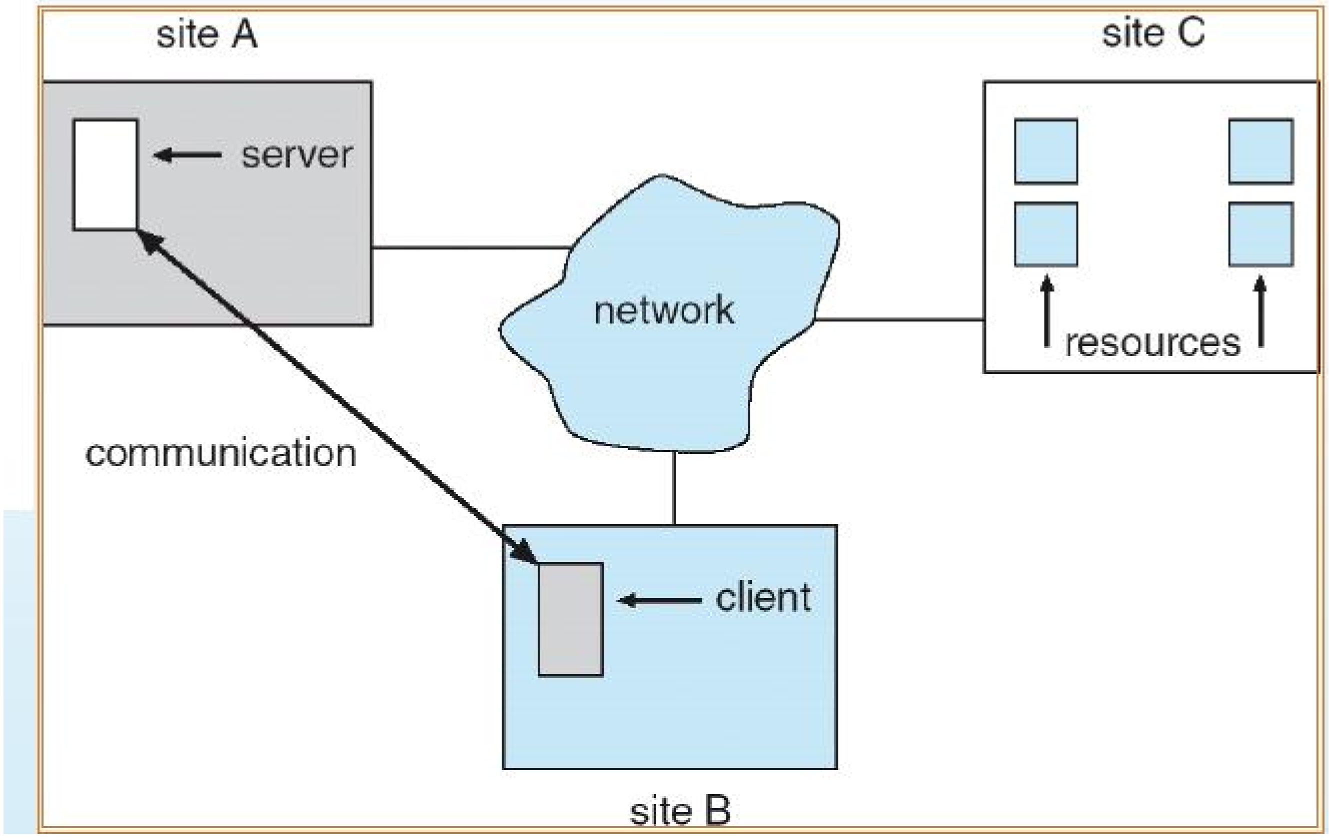
First, we only chose systems that were designed from scratch as-distributed systems (systems that gradually evolved by connecting together existing centralized systems or are multiprocessor versions of UNIX were excluded).

Second, we only chose systems that have actually been implemented; paper designs did not count.

Third, we only chose systems about which a reasonable amount of information was available.

NOTE: if a user can tell which computer he/she is using, then he/she is not using a distributed system. The user of a true distributed operating system should not know (or care) on which machine (or machines) their programs are running, where their files are stored and so on.

A Distributed System



TRANSPARENCY

• Goal motivated by the desire to hide all irrelevant system-dependent details from the user, whenever possible.

• It is more important in distributed systems due to higher implementation complexities.

• **Shielding the system**-dependent information from the users is a trade-off between simplicity and effectiveness.

• **Access transparency** - Local and remote system entities must remain indistinguishable when viewed through the user interface. The distributed operating system maintains this perception through the exposure of a single access mechanism for a system entity, regardless of that entity being local or remote to the user. Transparency dictates that any differences in methods of accessing any particular system entity—either local or remote—must be both invisible to, and undetectable by the user.

• **Location transparency** - Location transparency comprises two distinct sub-aspects of transparency, Naming transparency and User mobility. Naming transparency requires that nothing in the physical or logical references to any system entity should expose any indication of the entities location, or its local or remote relationship to the user. User mobility requires the consistent referencing of system entities, regardless of the system location from which the reference originates. Transparency dictates that the relative location of a system entity—either local or remote—must be both invisible to, and undetectable by the user.

• **Migration transparency** - Logical resources and physical processes migrated by the system, from one location to another in an attempt to maximize efficiency, reliability, availability, security, or whatever reason, should do so automatically controlled solely by the system. There are a myriad of possible reasons for migration; in any such event, the entire process of migration before, during, and after should occur without user knowledge or interaction. Transparency dictates that both the need for, and the execution of any system entity migration must be both invisible to, and undetectable by the user.

• **Concurrency transparency** - The distributed operating system allows for simultaneous use of system resources by multiple users and processes, which are kept completely unaware of the concurrent usage. Transparency dictates that both the necessity for concurrency and the multiplexed usage of system resources must be both invisible to, and undetectable by the user.

• **Replication transparency** - A system's elements or components may need to be copied to strategic remote points in the system in an effort to possibly increase efficiencies through better proximity, or provide for improved reliability through the duplication of a back-up. This duplication of a system entity and its subsequent movement to a remote system location may occur for any number of possible reasons; in any event, the entire process before, during, and after should occur without user knowledge or interaction. Transparency dictates that the necessity and execution of replication, as well as the existence of replicated entities throughout the system must be both invisible to, and undetectable by the user.

• **Parallelism transparency** - Arguably the most difficult aspect of transparency, and described by Tanenbaum as the "Holy grail" for distributed system designers. A system's parallel execution of a task among various processes throughout the system should occur without any required user knowledge or interaction. Transparency dictates that both the need for, and the execution of parallel processing must be both invisible to, and undetectable by the user.

• **Failure transparency** - In the event of a partial system failure, the system is responsible for the automatic, rapid, and accurate detection and orchestration of a remedy. These measures should exhibit minimal user imposition, and should initiate and execute without user knowledge or interaction. Transparency dictates that users and processes be exposed to absolute minimal imposition as a result of partial system failure; and any system-employed techniques of detection and recovery must be both invisible to, and undetectable by the user.

• **Performance transparency** - In any event where parts of the system experience significant delay or load imbalance, the system is responsible for the automatic, rapid, and accurate detection and orchestration of a remedy. These measures should exhibit minimal user imposition, and should initiate and execute without user knowledge or interaction. While reasonable and predictable performances are important goals in these situations, there should be no expressed or implied concepts of fairness or equality among affected users or processes. Transparency dictates that users and processes be exposed to absolute minimal imposition as a result of performance delay or load imbalance; and any system-employed techniques of detection and recovery must be both invisible to, and undetectable by the user.

• **Size transparency** - A system's geographic reach, number of nodes, level of node capability, or any changes therein should exists without any required user knowledge or interaction. Transparency dictates that system and node composition, quality, or changes to either must be both invisible to, and undetectable by the user.

• **Revision transparency** - System occasionally have need for system-software version changes and changes to internal implementation of system infrastructure. While a user may ultimately become aware of, or discover the availability of new system functions or services, their implementation should in no way be the prompt for this discovery. Transparency dictates that the implementation of system-software version changes and changes to internal system infrastructure must be both invisible to, and undetectable by the user; except as revealed by administrators of the system.

SCHEDULLING TECHNIQUES

The hierarchical model provides a general model for resource control but does not provide any specific guidance on how to do scheduling. If each process uses an entire processor (i.e., no multiprogramming), and each process is independent of all the others, any process can be assigned to any processor at random. However, if it is common that several processes are working together and must communicate frequently with each other, as in UNIX pipelines or in cascaded (nested) remote procedure calls, then it is desirable to make sure that the whole group runs at once. Let us assume that each processor can handle up to N processes. If there are plenty of machines and N is reasonably large, the problem is not finding a free machine (i.e., a free slot in some process table), but something more subtle. The basic difficulty can be illustrated by an example in which processes A and B run on one machine and processes C and D run on another. Each machine is time shared in, say, 100-millisecond time slices, with A and C running in the even slices, and B and D running in the odd ones. Suppose that A sends many messages or makes many remote procedure calls to D. During time slice 0, A starts up and immediately calls D, which unfortunately is not running because it is now C's turn. After 100 milliseconds, process switching takes place, and D gets A's message, carries out the work, and quickly replies. Because B is now running, it will be another 100 milliseconds before A gets the reply and can proceed. The net result is one message exchange every 200 milliseconds. What is needed is a way to ensure that processes that communicate frequently run simultaneously. Although it is difficult to determine dynamically the inter-process communication patterns, in many cases a group of related processes will be started off together.

PROCESS MANAGEMENT

Process management provides policies and mechanisms for effective and efficient sharing of a system's distributed processing resources between that system's distributed processes. These policies and mechanisms support operations involving the allocation and de-allocation of processes and ports to processors, as well as provisions to run, suspend, migrate, halt, or resume execution of processes. While these distributed operating system resources and the operations on them can be either local or remote with respect to each other, the distributed operating system must still maintain complete state of and synchronization over all processes in the system; and do so in a manner completely consistent from the user's unified system perspective.

As an example, load balancing is a common process management function. One consideration of load balancing is which process should be moved. The kernel may have several mechanisms, one of which might be priority-based choice. This mechanism in the kernel defines *what can be done*; in this case, choose a process based on some priority. The system management components would have policies implementing the decision making for this context. One of these policies would define what priority means, and how it is to be used to choose a process in this instance.

RESOURCES MANAGEMENT

Resource management in a distributed system differs from that in a centralized system in a fundamental way. Centralized systems always have tables that give complete and up-to-date status information about all the resources being managed; distributed systems do not. For example, the process manager in a traditional centralized operating system normally uses a “process table” with one entry per potential process. When a new process has to be started, it is simple enough to scan the whole table to see whether a slot is free. A distributed operating system, on the other hand, has a much harder job of finding out whether a processor is free, especially if the system designers have rejected the idea of having any central tables at all, for reasons of reliability. Furthermore, even if there is a central table, recent events on outlying processors may have made some table entries obsolete without the table manager knowing it. The problem of managing resources without having accurate global state information is very difficult.

PROCESSOR ALLOCATION

One of the key resources to be managed in a distributed system is the set of available processors. One approach that has been proposed for keeping tabs on a collection of processors is to organize them in a logical hierarchy independent of the physical structure of the network, as in MICROS. This approach organizes the machines like people in corporate, military, academic, and other real-world hierarchies. Some of the machines are workers and others are managers. For each group of k workers, one manager machine (the "department head") is assigned the task of keeping track of who is busy and who is idle. If the system is large, there will be an unwieldy number of department heads; so some machines will function as "deans," riding herd on k department heads. If there are many deans, they too can be organized hierarchically, with a "big cheese" keeping tabs on k deans. This hierarchy can be extended ad infinitum, with the number of levels needed growing logarithmically with the number of workers. Since each processor need only maintain communication with one superior and k subordinates, the information stream is manageable. An obvious question is, "What happens when a department head, or worse yet, a big cheese, stops functioning (crashes)?" One answer is to promote one of the direct subordinates of the faulty manager to fill in for the boss. The choice of which one can either be made by the subordinates themselves, by the deceased's peers, or in a more autocratic system, by the sick manager's boss. To avoid having a single (vulnerable) manager at the top of the tree, one can truncate the tree at the top and have a committee as the ultimate authority. When a member of the ruling committee malfunctions, the remaining members promote someone one level down as a replacement. Although this scheme is not completely distributed, it is feasible and works well in practice. In particular, the system is self repairing, and can survive occasional crashes of both workers and managers without any long-term effects. In MICROS, the processors are monoprogrammed, so if a job requiring S processes suddenly appears, the system must allocate S processors for it. Jobs can be created at any level of the hierarchy. The strategy used is for each manager to keep track of approximately how many workers below it are available (possibly several levels below it). If it thinks that a sufficient number are available, it reserves some number R of them, where R 2 S, because the estimate of available workers may not be exact and some machines may be down. If the manager receiving the request thinks that it has too few processors available, it passes the request upward in the tree to its boss. If the boss cannot handle it either, the request continues propagating upward until it reaches a level that has enough available workers at its disposal. At that point, the manager splits the request into parts and parcels them out among the managers below it, which then do the same thing until the wave of scheduling requests hits bottom. At the bottom level, the processors are marked as “busy,” and the actual number of processors allocated is reported back up the tree. To make this strategy work well, R must be large enough so that the probability is high that enough workers will be found to handle the whole job. Otherwise, the request will have to move up one level in the tree and start all over, wasting considerable time and computing power. On the other hand, if R is too large, too many processors will be allocated, wasting computing capacity until word gets back to the top and they can be released. The whole situation is greatly complicated by the fact that requests for processors can be generated randomly anywhere in the system, so at any instant, multiple requests are likely to be in various stages of the allocation algorithm, potentially giving rise to out-of-date estimates of available workers, race conditions, deadlocks, and more.

Failure Recovery

Failure Detection

Detecting hardware failure is difficult. To detect a link failure, a handshaking protocol can be used. Assume Site A and Site B has established a link. At fixed intervals, each site will exchange an I-am-up message indicating that they are up and running. If Site A does not receive a message within the fixed interval, it assumes either (a) the other site is not up or (b) the message was lost. Then, Site A can now send an “Are-you-up?” message to Site B. If Site A does not receive a reply, it can repeat the message or try an alternate route to Site B.
If Site A does not ultimately receive a reply from Site B, it concludes some type of failure has occurred in site B. Such failure could be that:
Types of failures:

- Site B is down
- The direct link between A and B is down
- The alternate link from A to B is down
- The message has been lost.

However, Site A cannot determine exactly **why** the failure has occurred

Reconfiguration

When Site A determines a failure has occurred, it must reconfigure the system:

1. If the link from A to B has failed, this must be broadcast to every site in the system
2. If a site has failed, every other site must also be notified indicating that the services offered by the failed site are no longer available.

When the link or the site becomes available again, this information must again be broadcast to all other sites.

Distributed Deadlock Detection

There are two kinds of potential deadlocks which are:

- resource deadlocks
- communication deadlocks

Resource deadlocks are traditional deadlocks, in which all of some set of processes are blocked waiting for resources held by other blocked processes. For example, if A holds X and B holds Y, and A wants Y and B wants X, a deadlock will result. In principle, this problem is the same in Centralized and distributed systems, but it is harder to detect in the latter because there are no centralized tables.

The other kind of deadlock that can occur in a distributed system is a communication deadlock. Suppose A is waiting for a message from B and B is waiting for C and C is waiting for A. Then we have a deadlock. Chandy et al. [1983] present an algorithm for detecting (but not preventing) communication deadlocks. Very crudely summarized, they assume that each process that is blocked waiting for a message knows which process or processes might send the message. When a process logically blocks, they assume that it does not really block but instead sends a query message to each of the processes that might send it a real (data) message. If one of these processes is blocked, it sends query messages to the processes it is waiting for. If certain messages eventually come back to the original process, it can conclude that a deadlock exists. In effect, the algorithm is looking for a knot in a directed graph.

Redundancy Techniques

All the redundancy techniques that have emerged take advantage of the existence of multiple processors by duplicating critical processes on two or more machines. A particularly simple, but effective, technique is to provide every process with a backup process on a different processor. All processes communicate by message passing.

Whenever anyone sends a message to a process, it also sends the same message to the backup process. The system ensures that neither the primary nor the backup can continue running until it has been verified that both have correctly received the message. Thus, if one process crashes because of any hardware fault, the other one can continue. Furthermore, the remaining process can then clone itself, making a new backup to maintain the fault tolerance in the future. One disadvantage of duplicating every process is the extra processors required, but another, more subtle problem is that, if processes exchange messages at a high rate, a considerable amount of CPU time may go into keeping the processes synchronized at each message exchanged. If a process crashes, recovery is done by sending the most recent checkpoint to an idle processor and telling it to start running. The recorder process then spoon feeds it all the messages that the original process received between the checkpoint and the crash. Messages sent by the newly restarted process are discarded. Once the new process has worked its way up to the point of crash, it begins sending and receiving messages normally, without help from the recording process.

STRENGTH AND WEAKNESS OF DISTRIBUTED OPERATING SYSTEM

STRENGTH

- The main goal of distributed system is the enormous rate of technological change in micro processor technology.

- Micro processors have become powerful and cheap compared with mainframes and minicomputer, so it has become attractive to think about designing large system that composes of many processors.
- Relative simplicity of software: each software has a dedicated function.
- Incremental growth.
- Reliability and availability.

WEAKNESS

- Unless one is very careful, it is easy for the communication protocol overhead to become a major source of in efficiency.
- With distributed systems, a high degree of fault tolerance is often, at least, an implicit goal.
- A more fundamental problem in distributed system is the lack of global state information.
- It is hard to schedule the processor optimally if you are not sure how many are up at the moment.

CONCLUSION

Distributed operating systems are still in an early phase of development, with many unanswered questions and relatively little agreement among workers in the field about how things should be done. Many experimental systems use the client-server model with some form of remote procedure call as the communication base, but there are also systems built on the connection model. Relatively little has been done on distributed naming, protection, and resource management, other than building straightforward name servers and process servers.
Fault tolerance is an up-and-coming area, with work progressing in redundancy techniques and atomic actions. Finally, a considerable amount of work has gone into the construction of file servers, print servers, and various other servers, but here too there is much work to be done. The only conclusion that we draw is that distributed operating systems will be an interesting and fruitful area of research for a number of years to come.

REFERNCES AND FURTHER READINGS

Stallings W. (2011).”Operating Systems: Internals and Design Principles”, 7th Edition, published by Prentice Hall.

Deitel. H. M. (2008). “Operating Systems”, Fourth Edition, published Pearson Education, Inc.

Tanenbaum A. S. (2007). “Modern Operating Systems”, 3rd Edition, published by Prentice Hall.

EXERCISES

QUESTION 1

ai. Mention the two main goals of memory management.
ii. How is memory management achieved in UNIX operating system?
b. Describe the symbols that are used to enforce access right on both files and directories in UNIX.
c.i. What is page fault?
ii. How is it handled?

QUESTION 2

a. Mention five historical features of LINUX.
b.i. Mention the function that completes the initialization of the LINUX kernel.
ii. Mention 5 types of kernel components that are initialized by this function.
c. i. What is interrupt?
ii. Explain the 2 types of interrupt
iii. Describe how interrupt is handled in LINUX operating system.

QUESTION 3

a. Solaris kernel is fully pre-emptible. Explain the six different priorities that are recognised in Solaris.
b. Describe the Solaris Run levels
c. Using appropriate diagram describe the Solaris booting phases.

QUESTION 4

a. Mention and explain the 9 main attributes that are used to evaluate operating systems.
b. Hence use the attributes to evaluate LINUX, WINDOWS AND SOLARIS.
c. Describe the intended future direction of WINDOWS operating system.

QUESTION 5

a. Explain the following concepts as they relate to operating system. Also mention at least one operating system that implements each of these concepts.

i) Multi threading ii) SMP iii) Processor affinity iv) Task scheduler. v) Pipe

b.i. What is a critical section in WINDOWS operating system?

ii. Describe the structure of every critical section.

c.i. Which architectural component is responsible for hardware error handling in WINDOWS?

ii. State four key functions of this architecture.

<u>QUESTION 6</u>

a. Explain the 3 main goals of computer security

b. Identify and discuss 4 types of attacks on operating systems.

c. Describe UNIX password security.