A

Seminar report

On

LINUX OPERATING SYSTEM

Submitted in partial fulfillment of the requirement for the award of degree of Bachelor of Technology in Computer Science

SUBMITTED TO:
www.studymafia.org

SUBMITTED BY:
www.studymafia.org

Acknowledgement

I would like to thank respected Mr…….. and Mr. ……..for giving me such a wonderful opportunity to expand my knowledge for my own branch and giving me guidelines to present a seminar report. It helped me a lot to realize of what we study for.

Secondly, I would like to thank my parents who patiently helped me as i went through my work and helped to modify and eliminate some of the irrelevant or un-necessary stuffs.

Thirdly, I would like to thank my friends who helped me to make my work more organized and well-stacked till the end.

Next, I would thank Microsoft for developing such a wonderful tool like MS Word. It helped my work a lot to remain error-free.

Last but clearly not the least, I would thank The Almighty for giving me strength to complete my report on time.

Preface

I have made this report file on the topic **LINUX OPERATING SYSTEM**; I have tried my best to elucidate all the relevant detail to the topic to be included in the report. While in the beginning I have tried to give a general view about this topic.

My efforts and wholehearted co-corporation of each and everyone has ended on a successful note. I express my sincere gratitude to …………..who assisting me throughout the preparation of this topic. I thank him for providing me the reinforcement, confidence and most importantly the track for the topic whenever I needed it.

CONTENTS

✓**INTRODUCTION TO OPERATING SYSTEM**

✓**LINUX COMPONENTS**

✓**PROCESS MANAGEMENT**

✓**PROCESS SCHEDULING**

✓**CONCLUSION**

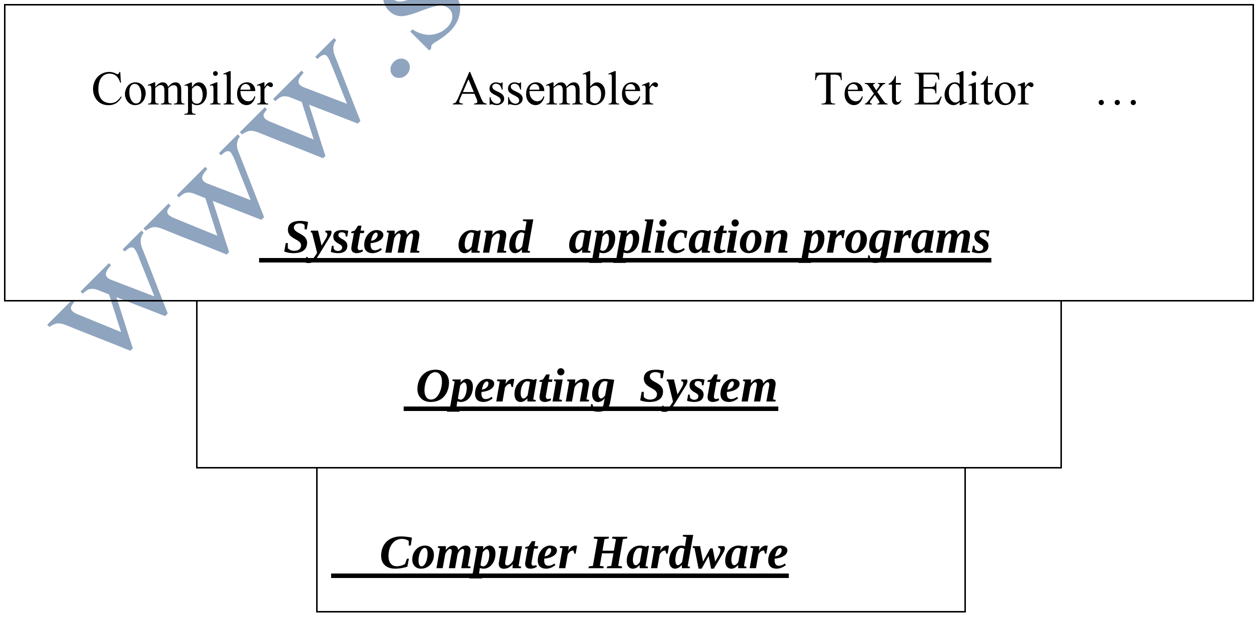
✓**REFERENCES**

INTRODUCTION

An Operating System is a program that acts as an Intermediatery between the user of a computer and the computer hardware. The purpose of an operating system is to provide an environment in which a user can execute programs in a convenient and efficient manner. An Operating System is a program that manages the computer hardware. It also provides a basis for application programs.

An Operating System is an important part of almost every computer system. An amazing aspect of operating system is how varied it is useful in accomplishing the tasks.

Mainframe Operating System are designed primarily to optimize utilization of hardware. Personal Operating System Support complex games, business applications and everything in Between. Handheld computer Operating System are designed to provide an environment in which a user can easily interface with computer to execute programs.



LINUX OPERATING SYSTEM

LINUX is modern , free operating system based on UNIX standards. It has been designed to run efficiently and reliably on common PC hardware, it run also on a variety of other platforms. It provides a programming interface and user interface compatible with standard UNIX systems.

A Complete Linux system includes many Components that were independently of Linux.
The core Linux operating-system Kernel is entirely original, But it allows much existing free UNIX software to run, resulting in an entire UNIX-compatible operating system free from proprietary code.

The Linux kernel is implemented as a traditional monolithic kernel for performance reasons, but it is modular enough in design to allow most drivers to be dynamically loaded and unloaded at a run time.

Linux is a multi-user system ,providing protection between processes and running multiple processes according to a time-sharing scheduler.

Inter-process communication is supported for message queues ,semaphores and shared memory and BSD's socket interface.

Multiple networking protocols can be accessed simultaneously through the socket interface.

To the user, the file system appears as a hierarchical Directory tree that obeys UNIX semantics. Linux uses an abstraction layer to manage multiple different file system.

Device-oriented network and virtual file systems are supported. Device-oriented file-systems access disk storage through two caches.

The memory management system uses page sharing and copy-on-write to minimize the duplication of data shared by different processes. Pages are loaded on demand when they are first referenced, and are paged back out to backing store according to an LFU algorithm if physical memory needs to be reclaimed.

LINUX COMPONENTS

The Linux System is composed of three main bodies of code

- ✓ Kernel
- ✓ System Libraries
- ✓ System Utilities

KERNEL:- The Kernel is responsible for maintaining all the important abstractions of the Operating System.

SYSTEM LIBRARIES:- The System Libraries define a set of Standard set of functions through which applications can interact with kernel and that implement much of the operating system Functionality that does not need the full privileges of kernel code.

SYSTEM UTILITIES:- The System Utilities are programs that Perform individual, specialized management tasks,including Virtual Memory and processes.

System Management Programs	User Processes	User Utility Programs	Compilers
<u>System Shared Libraries</u>			
<u>Linux Kernel</u>			
<u>Loadable Kernel Modules</u>			

The Linux Kernel forms the core of the Linux Operating System.
It provides all the functionality necessary to run processes and it provides system services to give arbitrated and protected access to the hardware resources .

The System Libraries provide many types of functionality .
At the simplest level, they allow applications to make kernel system service requests.
Making a system call involves transferring control from unprivileged user mode to privileged Kernel mode.

The System Libraries may also provide more complex versions of the basic system calls. The libraries provide routines that do not correspond to system calls at all, such as sorting algorithms ,mathematical functions and string manipulation routines. The Linux System includes a wide variety of user-mode programs- boot System utilities and User utilities. The System utilities include all the programs necessary to initialize the system, to load kernel modules.

PROCESS MANAGEMENT

A process is the basic context within which all-user requested Activity is serviced within operating system. To be compatible with other UNIX systems, Linux must use a process model similar to those of other versions of UNIX. Linux operates differently from UNIX in a few key places.

1: THE FORK/EXEV PROCESS MODEL
2: PROCESSES AND THREADS

1: ***THE FORK/EXEV PROCESS MODEL*** :-

The basic principle of UNIX process management is to separate two distinct operations: the creation of processes and the running of the new program. A new process is created by the FORK system call, and a new program is run after a call to EXEV.

These are two distinctly separate functions. A new process may be created with FORK without a new program being run the new sub-process simply continues to execute exactly the same program that first , parent process was running.

This model has the advantage of great simplicity. Rather than having to specify every detail of the environment of a new program in the system call that runs that program. If a parent process wishes to modify the environment in which a new program is to be run, it can Fork and then still running the original program in a child process , make any system call it requires to modify that child process before finally executing the new program.

Broadly , **Process Properties** fall into three groups :

1: *PROCESS IDENTITY*
2: PROCESS ENVIRONMENT
3: PROCESS CONTEXT

1**: Process Identity :-** A process identity consists mainly of the following items:

- ✓ **Process ID(PID) :** Each process has a unique identifier. PIDs are used to specify processes to the operating system when an application makes a system call to signal, modify , or wait for another process. Additional identifier associate the process with a process group and login session.
- ✓ **Credentials:-** Each process must have an associated user-id and one or more group IDs that determine the rights of a process to access system resources and files.
- ✓ **Personality :-** Process personality are not traditionally found on UNIX systems , but under Linux each process has an associated personality identifier that can modify slightly the semantics of certain system calls.

The process group and session identifiers can be changed if the process wants to start a new group or session. Its credentials can be changed, subject to appropriate security checks.

2**: PROCESS ENVIRONMENT:-** A process environment is inherited from its parent and is composed of two null-terminated vectors:

- ✓ **The Argument Vector**
- ✓ **The Environment Vector**

The Argument Vector simply lists the command-line arguments used to invoke the running program and conventionally starts with the name of the program itself.

The Environment Vector is a list of "NAME=VALUE" pairs that associates named environment variables with arbitrary textual values. The environment is not held in kernel memory, but is stored in the process' own user-mode address space as the first datum at the top of the process' stack.

The passing of environment variables from one process to the next and inheriting of these variables by the children of the process ,provide flexible ways to pass information to components of the user-mode system software.

3**: PROCESS CONTEXT :-** The process identity and environment properties are usually setup when a process is created and not changed until that process exists. Process context is the state of the running program at any one time , it changes constantly.

Scheduling Context :-

The most important part of the process context is its scheduling context: the information that the scheduler needs to suspend and restart the process. This information includes saved copies of all the process' registers. Floating point registers are stored separately and are restored only when needed , so that processes that do not use floating-point arithmetic do not incur the overhead of saving that state.

A key part of the scheduling context is the process' kernel stack: a separate area of kernel memory reserved for use exclusively by kernel-mode code.

- ✓ **Accounting:** The kernel maintains information about the resources currently being consumed by each process and the total resources consumed by the process in its entire life time so far.

- ✓ **File Table :** The file table is an array of pointers to kernel file structures. While making file-I/O system calls , processes refer to files by their index into this table.

- ✓ **File-System Context** : Whereas the file table lists the existing open files, the file-system context applies to requests to open new files.

- ✓ **Signal – Handler Table :** UNIX systems can deliver asynchronous signals to a process in response to various external events. The signal-handler table defines the routine in the process' address space to be called when the specific signals arrive.
- ✓ **Virtual – Memory Context :** The Virtual-memory context describes the full contents of a process' private address space.

PROCESSES AND THREADS

Most modern operating systems support both processes and threads. Processes represent the execution of single programs, whereas threads represent separate , concurrent execution contexts within a single process running a single program.

Any two separate processes have their own independent address spaces, even if they are using shared memory to share some of the contents of their virtual memory. Threads can be implemented in several ways. A thread may be implemented in the operating system's kernel as an object owned by a process or it may be fully independent entity. It cannot be implemented in the kernel at all – threads may be implemented purely within application or library code with the help of kernel-supplied timer interruptions.

The Linux kernel deals simply with the difference between processes and threads : it uses exactly the same internal representation for each. A thread is just a new process that happens to share the same address space as its parent. The distinction between a process and a thread is made only when a new thread is created by the CLONE system call. Whereas fork creates a new process that has its own entirely new process context , clone creates a new process that has its own identity but is allowed to share the data structure of its parent.

This distinction can be accomplished because Linux does not hold process' entire context with in the main process data structure rather it holds contexts with in independent sub contexts. The clone system call accepts an argument that tells it which sub contexts to copy, and which to share, when it creates the new process.

The POSIX working groups have defined a programming interface , specified in the POSIX.1c standard , to allow applications to run multiple threads. The Linux system libraries support two separate mechanisms that implement this single standard in different ways.
It also allows multiple threads to be executing kernel system calls simultaneously.

PROCESS SCHEDULING

Scheduling is the job of allocating CPU time to perform different tasks within an operating system. Normally, we think of scheduling as being the running and interrupting of processes , but another aspect of scheduling is also important to Linux : the running of the various kernel tasks. Kernel tasks that are requested by a running process and tasks that execute internally on behalf of a device driver.

1***: Kernel Synchronization***
2: Process Scheduling
3: Symmetric Multiprocessing

✓ **KERNEL SYNCHRONIZATION :-** The way that the kernel schedules its own operations is fundamentally different from the way that it does process scheduling. A request for kernel mode execution can occur in two ways. A running program may request an operating system service, either explicitly via a system call, or explicitly- for example when a page fault occurs. Alternately a device driver may deliver a hardware interrupt that causes the CPU to start executing a kernel-defined handler for that interrupt.

The first part of Linux's solution to this problem lies in making normal code non-preemptible. Usually when a timer interrupt is received by the kernel, it invokes the process scheduler, potentially so that it can suspend execution of the currently running process and resume running another one.

Once a piece of kernel code starts running, it can gurantee that it will be the only kernel code running until one of the following action occurs:

- ✓ An Interrupt
- ✓ A Page Fault
- ✓ A Kernel code call to the scheduler itself

Interrupts are a problem only if they contain critical sections themselves. Timer interrupts never directly cause a process re-scheduler; they just request that a reschedule be performed later, so any incoming interrupts can not effect the execution order of non-interrupt kernel code. Once the interrupt service is finished, execution will simply return to the same kernel code that was running when the interrupt was taken.

Page faults are a potential problem; if a Kernel routine tries to read/write to user memory, it may incur a page-fault that requires disk I/O completed and the running process will be suspended until the I/O completes. When the process becomes run able again, it will continue to execute in kernel mode, continuing at the instruction after the call to the scheduler.

Linux implements this architecture by separating interrupt service routines into two sections: the top half and the bottom half. The Top Half is a normal interrupt service routine, and runs with recursive interrupts disabled, interrupts of a higher priority may interrupt the routine, but interrupts of the same or lower priority are disabled. The Bottom Half of a service routine is run, with all interrupts enabled, by a miniature scheduler that ensures that bottom halves never interrupts themselves. The Bottom Half scheduler is invoked automatically whenever an interrupt service routine exists.

Top Half Interrupt handlers
Bottom Half Interrupt handlers
Kernel-system routine(not preemptible)
User-Mode Programs(Preemptible)

✓ **<u>PROCESS SCHEDULING :-</u>**

Once the kernel has reached a rescheduling point – either a rescheduling interrupt has occurred or a running kernel process has blocked waiting for some wakeup signals-it must decide what process to run next. Linux has two separate process scheduling algorithms.

1: TIME SHARING Algorithms for preemptive scheduling
2: TIME SHARING Algorithms for real-time tasks

Part of every process identity is a scheduling class, that defines which of these algorithms to apply to the process. The scheduling classes used by Linux are defined in the POSIX standard's extensions for real-time computing.

For time-sharing processes, Linux uses a prioritized, credit-based algorithm. Each process possesses a certain number of scheduling credits, when a new task must be chosen to run, the process with the most credits is selected. Every time that a time interrupts occurs, the currently running process loses one credit, when its credits reaches zero, it is suspended and another process is chosen. If no run able processes have any credits, then Linux performs a re-crediting operation, adding credits to every process in the system, rather than to just the run-able ones, according to the following rule:

Credits = (Credits /2) + Priority

This algorithm tends to mix two factors, the process' history and priority. The use of the process priority in calculating new credits allows the priority of a process to be fine-tuned. Background batch jobs can be given a low priority, hence will receive a smaller percentage of the CPU time than will smaller jobs that have higher priorities.

Linux's real-time scheduling is simpler still. Linux implements the two real-time scheduling classes required by POSIX.1b: First come, First serve (FCFS) and Round-robin. Linux's real-time scheduling is soft-rather than hard-real time.

✓ SYMMETRIC MULTIPROCESSING :-

The Linux 2.0 kernel was the first stable Linux kernel to support Symmetric Multiprocessor (SMP) hardware. Separate processes or threads can execute in parallel on separate processors. The implementation of SMP in this kernel imposes the restriction that only one processor at a time can be executing kernel-mode code. SMP uses a single kernel spin lock to enforce this rule. This spin lock does not pose a problem for computation-bound tasks, but tasks that involve a lot of kernel activity can become seriously bottlenecked.

The development kernel 2.3 seems to have completely removed the single-kernel spin lock.

CONCLUSION

The Latest LINUX Operating System therefore a recently developed system which has proved the more convenient and efficient operating system from the processing point of view and from the addition of latest soft-wares point of view. It has been considered as consisting of an advance technology than WIN-98 and Windows-XP.

REFERENCES

www.google.com
www.wikipedia.org
www.studymafia.org