UNIT 1 INTRODUCTION TO LINUX OPERATING SYSTEM

Structure **Page Nos.**

1.0 INTRODUCTION

Today's trends are towards development of free and platform independent software Java popularised this concept in the field of programming language and now it is being done by Linux in the operating system.

Linux started out as a Unix variant to run on an IBM PC platform but with one major difference— its source code was freely available under the auspices of Free Software Foundation (FSF). Due to this, it quickly positioned itself as an alternative to other Unix workstations such as those offered by Sun Microsystem, Compaq and Sillicon Graphics. Due to high quality designing of its kernel qualities such as stability, modularity and easy configurability — it is now dominating the corporate world significantly. For example, major banks, investment houses, retail establishments, educational institutions, etc., use it.

1.1 OBJECTIVES

After going through this unit you will be able to :

- list the features of Linux;
- list the drawbacks of Linux;
- understand the process of thread management, and
- understand the memory management features.

1.2 FEATURES OF LINUX

Linux has several strong features. In this section we will discuss and explain them.

Linux is inexpensive

The first benefit of Linux is cost. All versions of Linux may be freely downloaded from the web. If you don't want to download, prepackaged versions of Linux may be purchased online. In addition, the software may be legally shared with your friends. In addition, when the time comes to upgrade the operating system, the Linux upgrade would be free.

In addition to being inexpensive, Linux can run on the old system. Its products can run on Intel 386 microprocessors, which were popular in the late 1980s. The server has never slowed down despite increased use.

Linux is Fast

Linux runs respectably well on old computers, and it is even faster on newer, more powerful computers. This is because Linux programs are very efficient and lean. They use as few resources as possible, and unlike Windows, Linux programs use little, if any, graphics. Graphics can slow a system's response time, making it slower than it truly is. Linux may not be pretty, but it is fast.

Linux is Stable

The Linux code is well written. This both increases the speed at which Linux runs and improves the stability of the operating system. Linux is next to impossible to crash. If an application crashes, you can simply remove the program from memory to restart your computer. In older versions of Windows, a crashing program had the potential to take down the entire computer. This is one of the reasons why Linux is used on many web servers where stability is crucial. With Linux, web-hosting providers can guarantee 99.9 percent uptime.

Open-Source Software

Finally, Linux has open-source software. This means that users can read the source code and modify it as needed. This probably means little to the average user of the final version of a Linux kernel. However, during development, "beta" releases of the kernel are available to developers who will download the code and test it thoroughly. When possible, they will find any problems and correct the code. This process helps to ensure that the final release of the kernel is as well written as possible.

1.3 DRAWBACKS OF LINUX

Even though Unix and Linux operating systems are widely used on corporate servers, web sites, and large-scale networking environments, we still won't find many people using it on their desktop computers or workstations at home. There are several reasons for this.

Security

Because code is distributed with the Linux software, programmers are free to explore how the system works – for good or bad. Many security loopholes have been reported in the literature. Although most system vulnerabilities are detected before the product is released, clever programmers can still discover new ones.

Lack of Support

No system is 100 percent secure. Even Microsoft products have security vulnerabilities. However, Microsoft products do have extensive documentation and support. Microsoft releases service pack and updates frequently to fix discovered vulnerabilities.

Support and documentation for Linux can be spotty at best. A customer who downloads Linux from a server may receive only an electronic manual and access to online help pages.

Limited Software Selection Choice

People purchase computers to run software. Users of Windows computers have many software titles to choose from. Linux users have software in every category but are often limited in their choices. For example, consider Internet browsers. The two most popular browsers are Netscape Navigator and Microsoft Internet Explorer. Both are available for Linux, but only Netscape has created a Linux version of its latest browser, version 6.32. internet Explorer's most recent Linux version is 5, despite the fact that Windows XP ships with Internet Explorer version 6.

You are also limited in your choice of word processors. The most popular word processor is Microsoft Word. Chances are that every computer in your school uses Word. But Word is not available for Linux. Your best choice is **StarOffice**, a suite of applications that contains a word processor. However, StarOffice, although a very nice product, is not Word. A proficient Word user would have to learn some new skills to use StarOffice.

Limited Hardware Support

Just as not all popular software run on Linux, not all hardware products work with Linux. Red Hat and the other Linux vendors work very hard to support the more common devices. They provide drivers for hardware devices. A driver is a small program that allows the operating system to communicate with the peripheral devices. Having the correct driver is crucial. If you have a new or unusual device, you may be out of luck. For instance, many branded companies have not written a Linux compatible driver.

Complexity

Linux, like its predecessor Unix, assumes that you know what you are doing, and it assumes that you know the consequences of every command you type. In contrast, Windows XP asks you to verify everything, and then shows you a pretty animation to confirm the action.

For a beginning user, Linux can be frightening to use; entering the wrong command can have serious consequences. It dosen't help that Linux is also case sensitive, so you must enter the commands in lowercase, and be careful to use the correct case for each subcommand you use with a command. Upper — and lowercases are often different actions.

1.4 COMPONENTS OF LINUX

In this section we will take up the various component of Linux Operating System.

1.4.1 Memory Management Subsystem

Linux is made up of a number of functionally separate pieces that, together, comprise the operating system. One obvious part of Linux is the kernel itself; but even that would be useless without libraries or shells. In this section we will discusss the various components of Linux kernel.

One of the basic objectives of any operating system is to make one feel that there is a large amount of memory although it is having a small physical memory. This apparently large memory is known as virtual memory. The system divides the memory into easily handled pages (logical unit) and swaps these pages onto a hard disk as the system runs.

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 it actually has by sharing it between competing processes as they need it.

Virtual memory does more than just make your computer's memory go further. The memory management subsystem includes:

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 processes 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) addresses spaces, there are times when you need processes to share memory. For example, there could be several processes in the system running concurrently and simultaneously depending upon the number of processors residing in the system but might be using the common file, e.g., C-amplifier.

Therefore, it is better to have only one copy in physical memory and all of the processes running sharing it. Dynamic libraries are another common example of executing code shared between several processes.

Another example of shared memory is that it 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.

An Abstract Model of Virtual Memory

Before considering the methods that Linux uses to support virtual memory it is useful to consider an abstract model which is applicable to a large number of systems.

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 stored 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 through a mapping scheme using 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 of the same size. They need not be,

but if they were not the system would be very hard to administer. The size of a page varies from one system to another. Each of these pages is given a unique number; the frame number (FN). In this paged model, a virtual address comprises two parts; virtual page frame number (VPFN) and offset within the frame. Each time the processor encounters a virtual address it must extract the virtual page frame number and the offset. The processor must translate the virtual page frame number into a physical one (address of RAM) and then access the location at the correct offset into that physical page. To do this the processor uses page tables. The size of a page table varies from one process to another and the number of page tables depends upon the number of processes residing in a system.

Figure 1 shows the virtual addresses spaces of two processes, process P1 and process P2, each with its own page tables. These page tables map each processes virtual pages into physical pages in memory. This shows that process P_1's virtual page frame number 0 is mapped into memory in physical page frame number 1 and that process P_2's virtual page frame number 1 is mapped into physical page frame number 4. Entry in the theoretical page table contains the following information:

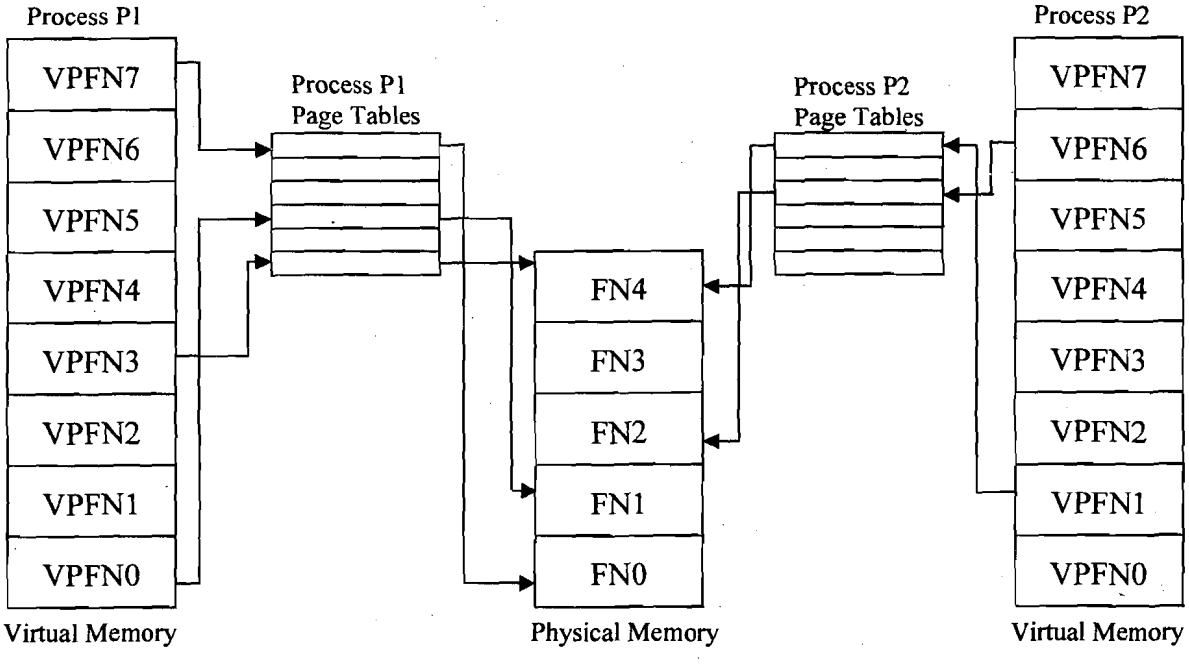

Figure 1: Abstract model of Virtual to Physical address mapping

The processor uses the virtual page frame number as an index into the processes page table to retrieve its page table entry. If the page table entry at that offset is valid, the processor takes the physical page frame number from this entry. If the entry is invalid, the process has accessed a non-existent area of its virtual memory. In this case, the processor cannot resolve the address and must pass control to the operating system so that it can fix things up.

In case the required page frame is not found, the processor generates a page fault and then the required page is brought from the hard disk.

Mapping of virtual address to physical address can be done in any order. For example, in process P_1 virtual page frame number 0 is mapped to physical page frame number 1 whereas virtual page frame number 7 is mapped to physical page frame number 0 even though it is higher in virtual memory than virtual page frame number 0. This demonstrates an interesting byproduct of virtual memory; the pages of virtual memory do not have to be present in physical memory in any particular order.

Linux shares many of the characteristics of the memory management schemes of other Unix implementations but has its own unique features. Overall, the Linux memory management scheme is quite complex. Here, we give a brief overview.

Linux Virtual Memory

Linux makes use of a three-level page table structure, consisting of the following types of tables (each individual table is the size of one page):

- **Page Directory:** An active process has a single page directory that is the size of one page. Each entry in the page directory points to one page of the page middle directory. The page directory must be in main memory for an active process.
- **Page Middle Directory:** The page middle directory may span multiple pages. Each entry in the page middle directory points to one page in the page table.
- **Page Table:** The page table may also span multiple pages. Each page table entry refers to one virtual page of the process.

To use this three-level page table structure, a virtual address in Linux is viewed as consisting of four fields. The leftmost (most significant) field is used as an index into the page directory. The next field serves as an index into the page middle directory. The third field serves as an index into the page table. The fourth field gives the offset within the selected page of memory.

Page Allocation

To enhance the efficiency of reading in and writing out pages to and from main memory, Linux defines a mechanism for dealing with contiguous blocks of pages mapped into contiguous blocks of page frames. For this purpose, the buddy system is uscd. The kernel maintains a list of contiguous page frame groups of fixed size; a group may consist of 1, 2, 4, 16, or 32 page frames. As pages are allocated and deallocated in main memory, the available groups are split and merged using the buddy algorithm.

Page Replacement Algorithm

The Linux page replacement algorithm is based on the clock algorithm in which a use bit and a modify bit are associated with each page in main memory. In the Linux scheme, the use bit is replaced with an 8-bit age variable. Each time that a page is accessed, the age variable is incremented. In the background, Linux periodically sweeps through the global page pool and decrements the age variable for each page as it rotates through all the pages in main memory. A page with an age of 0 is an "old" page that has not been referenced in some time and is the best candidate for replacement. The larger the value of age, the more frequently a page has been used in recent times and the less eligible it is for replacement. Thus, the Linux algorithm is a form of least frequently used policy.

1.4.2 Linux Process and Thread Management

Processes carry out tasks within the operating system. A program is a set of machine code instructions and data stored in an executable image on disk and is, as such, a passive entity; a process can be thought of as a computer program in running state. It is a dynamic entity, constantly changing as the machine code instructions are executed by the processor. As well as the program's instructions and data, the process also includes the program counter and all of the CPU's registers as well as the process stacks containing temporary data such as routine parameters, return addresses and saved variables. Linux is a multiprocessing operating system which can support many processes running in parallel. Processes are separate tasks each with their own rights and responsibilities and also running in their own address spaces. If one process crashes it will not cause another process in the system to crash. Each individual

process runs in its own virtual address space and is not capable of interacting with another process except through secure mechanisms to be managed by kernel.

The most precious resource in the system is the CPU, usually there is only one except in a multi-processors based system. Linux is a multiprocessing operating system, its objective is to have a process running on each CPU in the system at all times, to maximize CPU utilization. If there are more processes than CPUs (and there usually are), the rest of the processes must wait before a CPU becomes free until they can be run. In a multiprocessing system many processes are kept in memory at the same time. Whenever a process has to wait, the operating system takes the CPU away from that process and gives it to another, more deserving process. It is the scheduler which chooses which is the most appropriate process to run next and Linux uses a number of scheduling strategies to ensure fairness.

Linux supports a number of different executable file formats, ELF (Executably and linkable format) is one, Java is another and these must be managed transparently.

Data structure for Linux Processes

So that Linux can manage the processes in the system, each process is represented by a task — struct data structure (task and process are terms that Linux uses Interchangeably). The task vector is an array of pointers to every task-struct data structure in the system. This means that the maximum number of processes in the system is limited by the size of the task vector; by default it has 512 entries. As processes are created, a new task_struct is allocated from system memory and added into the task vector. To make it easy to find, the current, running process is pointed to by the current pointer.

As well as the normal type of process, Linux supports real time processes. These processes have to react very quickly to external events (hence the term "real time") and they are treated differently from normal user processes by the scheduler. Although the task-struct data structure is quite large and complex, its fields can be divided into a number of functional areas:

State: As a process executes its changes state according to its circumstances. Linux processes have the following states:

Running: The process is either running (it is the current process in the system) or it is ready to run (it is waiting to be assigned to one of the system's CPUs).

Waiting: The process is waiting for an event or for a resource. Linux differentiates between two types of waiting process; interruptible and uninterruptible. Interruptible waiting processes can be interrupted by signals whereas uninterruptible waiting processes are waiting directly on hardware conditions and cannot be interrupted under any circumstances.

Stopped: The process has been stopped, usually by receiving a signal. A process that is being debugged can be in a stopped state.

Zombie: This is a halted process which, for some reason, still has a task-struct data structure in the task vector. It is what it sounds like, a dead process.

Scheduling Information: The scheduler needs this information in order to fairly decide which process in the system most deserves to run,

Identifiers: Every process in the system has a process identifier. The process identifier is not an index into the task vector, it is simply a number. Each process also has User and group identifiers, these are used to control this processes access to the files and devices in the system.

Inter-Process Communication (IPC): Linux supports the classic Unix™ IPC mechanisms of signals, pipes and semaphores and also the System V IPC mechanisms of shared memory, semaphores and message queues to allow processes to communicate with each other and with the kernel to coordinate their activities.

Links: In a Linux system no process is independent of any other process. Every process in the system, except the initial process has a parent process. In Unix operating system the initial process is known as init. New processes are not created, they are copied, or rather cloned from previous processes. Every task-struct representing a process keeps pointers to its parent process and to its siblings (those processes with the same parent process) as well as to its own child processes.

Times and Timers: The kernel keeps track of a processes creation time as well as the CPU time that it consumes during its lifetime. Each clock tick, the kernel updates the amount of time in jiffies that the current process has spent in system and in user mode. Linux also supports process specific interval timers, processes can use system calls to set up timers to send signals to themselves when the timers expire. These timers can be single-shot or periodic timers.

File System: Processes can open and close files as they includes pointers to any files opened by this process.

Virtual memory: Most processes have some virtual memory (kernel threads and daemons do not) and the Linux kernel must track how that virtual memory is mapped onto the system's physical memory.

Processor Specific Context: A process could be thought of as the sum total of the system's current state. Whenever a process is running it is using the processor's registers, stacks and so on. This is the processes context and, when a process is suspended, all of that CPU specific context must be saved in the task_struct for the process. When a process is restarted by the scheduler its context is restored from here.

Linux Threads

A new process is created in Linux by copying the attributes of the current process. A new process can be cloned so that it shares resources, such as files, signal handlers, and virtual memory. When the two processes share the same virtual memory, they function as threads within a single process. However, no separate type of data structure is defined for a thread. Thus, Linux makes no distinction between a thread and a process.

1.4.3 File Management Subsystem

In Linux, as it is for Unix, the separate filesystems that the system may use are not accessed by device identifiers (such as a drive number or a drive name) but instead they are combined into a single hierarchical tree structure that represents the filesystem as a single entity. Linux adds each new filesystem into this single filesystem tree as they are mounted onto a mount directory, for example / mnt/cdrom. One of the most important features of Linux is its support for many different filesystems. This makes it very flexible and well able to coexist with other operating systems. The most popular filesystem for Linux is the EXT2 filesystem and this is the filesystem supported by most of the Linux distributions.

A filesystem gives the user a sensible view of files and directories held on the hard disks of the system regardless of the filesystem type or the characteristics of the underlying physical device. Linux transparently supports many different filesystems (for example MS-DOS and EXT2) and presents all of the mounted files and

filesystems as one integrated virtual filesystem. So, in general, users and processes do not need to know what sort of filesystem that any file is part of, they just use them.

The block device drivers hide the differences between the physical block device types (for example, IDE and SCSI) and, so far as each filesystem is concerned, the physical devices are just linear collections of blocks of data. The block sizes may vary between devices, for example 512 bytes is common for floppy devices whereas 1024 bytes is common for IDE devices and, again, this is hidden from the users of the system. An EXT2 filesystem looks the same no matter what device holds it.

1.4.4 Device Drivers

Device drivers make up the major part of the Linux kernel. Like other parts of the operating system, they operate in a highly privileged environment and can cause disaster if they get things wrong. Device drivers control the interaction between the operating system and the peripheral devices that they are controlling. For example, the filesystem makes use of a general block device interface when writing blocks to a disk. The driver takes care of the details and makes device specific things happen. Device drivers are specific to the controller chip that they are driving.

Check Your Progress 1

1) What are the features of Memory Management subsystem?

.....

.....

.....

2) What are the different states of a Linux operating system?

.....

.....

.....

3) What is the purpose of a file system?

.....

.....

.....

1.5 SUMMARY

Linux is an Unix like operating system, with the major difference that its source code is freely available. In this unit we have described the strong features of the operating system and also highlighted its drawbacks. Linux like any other operating system is made up of a number of functionally separate précis. The kernel is the most important component of the system. It deals Memory Manager, File Manager, Process Manager and I/O Manager. Device Drivers make up the major part of the Linux kernel. They control the interaction between the Operating system and peripheral devices. One of the main objective of Memory Management is to make available to the process a large amount of Memory although it is having a small physical memory through virtual memory concept. Although virtual memory allows processes to have separate address space it also provides shared space among many processes. The unit also describes the process and Thread Management.

1.6 SOLUTIONS/ANSWERS

1) The following are the important features:

   - Protection of a process
   - Address mapping from virtual to physical
   - Swapping of a process between Hard disk & Physical memory.
   - Providing a shared space among many processes.

2) Linux processes have the following states:

   - Running
   - Waiting
   - Stopped
   - Zombie

3) The file system provides a user a complete view of files & directories held on the hard disk of the system regardless of the file system type or characteristics of underlying physical device. Linux system supports many different file systems and presents all of the mounted files & filesystems as one integrated virtual filesystem.

1.7 FURTHER READING

Operating Systems, 4th Edition, William Stallings, PHI.