1

LINUX OPERATING SYSTEM INTRODUCTION

Unit Structure

1.0 Objective

1. To presents a general picture of what is inside a Unix kernel and how Linux competes against other well-known Unix systems.
2. To learn various kernel types.
3. To gain knowledge on GRUB
4. To learn about Inter process communication

1.1 Introduction

Linux is a member of the large family of Unix-like operating systems. A relative newcomer experiencing sudden spectacular popularity starting in the late 1990s, Linux joins such well-known commercial Unix operating systems as System V Release 4 (SVR4) developed by AT&T, which is now owned by Novell; the 4.4 BSD release from the University of California at Berkeley (4.4BSD), Digital Unix from Digital Equipment Corporation (now Compaq); AIX from IBM; HP-UX from Hewlett-Packard; and Solaris from Sun Microsystems.

Linux was initially developed by Linus Torvalds in 1991 as an operating system for IBM compatible personal computers based on the Intel 80386 microprocessor. Linus remains deeply involved with improving Linux, keeping it up to date with various hardware developments and coordinating the activity of hundreds of Linux developers around the world. Over the years, developers have worked to make Linux available on other architectures, including Alpha, SPARC, Motorola MC680x0, PowerPC, and IBM System/390.

1.2 Linux Versus Other Unix-Like Kernels

One of the more appealing benefits to Linux is that it isn't a commercial operating system: its source code under the GNU Public License is open and available to anyone to study.

The GNU project is coordinated by the Free Software Foundation, Inc. (http://www.gnu.org/); its aim is to implement a whole operating system freely usable by everyone. The availability of a GNU C compiler has been essential for the success of the Linux project. Technically speaking, Linux is a true Unix kernel, although it is not a full Unix operating system, because it does not include all the applications such as filesystem utilities, windowing systems and graphical desktops, system administrator commands, text editors, compilers, and so on. However, since most of these programs are freely available under the GNU General Public License, they can be installed into one of the filesystems supported by Linux. Since Linux is a kernel, many Linux users prefer to rely on commercial distributions, available on CD-ROM, to get the code included in a standard Unix system. Alternatively, the code may be obtained from several different FTP sites.

The Linux source code is usually installed in the /usr/src/linux directory. In the rest of this book, all file pathnames will refer implicitly to that directory. 1.1 Linux Versus Other Unix-Like Kernels The various Unix-like systems on the market, some of which have a long history and may show signs of archaic practices, differ in many important respects. All commercial variants were derived from either SVR4 or 4.4BSD; all of them tend to agree on some common standards like IEEE's POSIX (Portable Operating Systems based on Unix) and X/Open's CAE (Common Applications Environment).

Understanding the Linux Kernel 7 The current standards specify only an application programming interface (API)—that is, a well-defined environment in which user programs should run. Therefore, the standards do not impose any restriction on internal design choices of a compliant kernel. As a matter of fact, several non-Unix operating systems like Windows NT are POSIX-compliant. In order to define a common user interface, Unix-like kernels often share fundamental design ideas and features. In this respect, Linux is comparable with the other Unix-like operating systems. The 2.2 version of the Linux kernel aims to be compliant with the IEEE POSIX standard. This, of course, means that most existing Unix programs can be compiled and executed on a Linux system with very little effort or even without the need for patches to the source code. Moreover, Linux includes all the features of a modern Unix operating system, like virtual memory, a virtual filesystem, lightweight processes, reliable signals, SVR4 inter process communications, support for Symmetric Multiprocessor (SMP) systems, and so on. By itself, the Linux kernel is not very innovative. When Linus Torvalds wrote the first kernel, he referred to some classical books on Unix internals, like Maurice Bach's The Design of the Unix Operating System (Prentice Hall, 1986). Actually, Linux still has some bias toward the Unix baseline described in Bach's book (i.e., SVR4).

1.3 Types of Kernels

The microkernel and monolithic kernels are two types of kernels in the operating system. The kernel is the main part of the OS. As a result, the kernel's important code is stored in different memory spaces. The kernel is a crucial component because it maintains the proper functioning of the complete system. It manages hardware and processes, files handling, and several other functions.

Microkernel

The microkernel is a type of kernel that permits the customization of the OS. It is privileged and provides low-level address space management as well as **Inter-Process Communication (IPC)**. Furthermore, OS functions like the virtual memory manager, file system, and CPU scheduler are built on top of the microkernel. Every service has its address space to make them secure. Moreover, every application has its address space. As a result, there is protection between applications, OS Services, and the kernel.

When an application requests a service from the OS services, the OS services communicate with one another in order to provide the requested service to the application. Inter-Process Communication (IPC) can assist in establishing this communication. Overall, microkernel-based operating systems offer a high level of extensibility. It is also possible to customize the operating system's services to meet the needs of the application.

Monolithic Kernel

The monolithic kernel manages the system's resources between the system application and the system hardware. Unlike the microkernel, user and kernel services are run in the same address space. It increases the kernel size and also increases the size of the OS.

The monolithic kernel offers CPU scheduling, device management, file management, memory management, process management, and other OS services via the system calls. All of these components, including file management and memory management, are located within the kernel. The user and kernel services use the same address space, resulting in a fast-executing operating system. One drawback of this kernel is that if anyone process or service of the system fails, the complete system crashes. The entire operating system must be modified to add a new service to a monolithic kernel.

Exo Kernel

It is the type of kernel which follows end-to-end principle. It has fewest hardware abstractions as possible. It allocates physical resources to applications.

Example:

Nemesis, ExOS etc.

However, Linux doesn't stick to any particular variant. Instead, it tries to adopt good features and design choices of several different Unix kernels. Here is an assessment of how Linux competes against some well-known commercial Unix kernels:

- **The Linux kernel is monolithic**: It is a large, complex do-it-yourself program, composed of several logically different components. In this, it is quite conventional; most commercial Unix variants are monolithic. A notable exception is CarnegieMellon's Mach 3.0, which follows a microkernel approach.
- Traditional Unix kernels are compiled and linked statically. Most modern kernels can dynamically load and unload some portions of the kernel code (typically, device drivers), which are usually called modules. Linux's support for modules is very good, since it is able to automatically load and unload modules on demand. Among the main commercial Unix variants, only the SVR4.2 kernel has a similar feature.
- **Kernel threading**: Some modern Unix kernels, like Solaris 2.x and SVR4.2/MP, are organized as a set of kernel threads. A kernel thread is an execution context that can be independently scheduled; it may be associated with a user program, or it may run only some kernel functions. Context switches between kernel threads are usually much less expensive than context switches between ordinary processes, since the former usually operate on a common address space. Linux uses kernel threads in a very limited way to execute a few kernel functions periodically; since Linux kernel threads cannot execute user programs, they do not represent the basic execution context abstraction. (That's the topic of the next item.)
- **Multithreaded application support**: Most modern operating systems have some kind of support for multithreaded applications, that is, user programs that are well designed in terms of many relatively independent execution flows sharing a large portion of the application data structures. A multithreaded user application could be composed of many lightweight processes (LWP), or processes that can operate on a common Understanding the Linux Kernel 8 address space, common physical memory pages, common opened files, and so on. Linux defines its own version of lightweight processes, which is different from the types used on other systems such as SVR4 and Solaris. While all the commercial Unix variants of LWP are based on kernel threads, Linux regards lightweight processes as the basic execution context and handles them via the nonstandard clone() system call.
- **Linux is a nonprimitive kernel**: This means that Linux cannot arbitrarily interleave execution flows while they are in privileged mode. Several sections of kernel code assume they can run and modify data structures without fear of being interrupted and having another thread alter those data structures. Usually, fully pre-emptive kernels are associated with special real-time operating systems. Currently, among conventional, general-purpose Unix systems, only

Solaris 2.x and Mach 3.0 are fully pre-emptive kernels. SVR4.2/MP introduces some fixed pre-emption points as a method to get limited pre-emption capability.

- **Multiprocessor support**: Several Unix kernel variants take advantage of multiprocessor systems. Linux 2.2 offers an evolving kind of support for symmetric multiprocessing (SMP), which means not only that the system can use multiple processors but also that any processor can handle any task; there is no discrimination among them. However, Linux 2.2 does not make optimal use of SMP. Several kernel activities that could be executed concurrently—like filesystem handling and networking—must now be executed sequentially.
- **Filesystem**: Linux's standard filesystem lacks some advanced features, such as journaling. However, more advanced filesystems for Linux are available, although not included in the Linux source code; among them, IBM AIX's Journaling File System (JFS), and Silicon Graphics Irix's XFS filesystem. Thanks to a powerful objectoriented Virtual File System technology (inspired by Solaris and SVR4), porting a foreign filesystem to Linux is a relatively easy task.
- **STREAMS**: Linux has no analog to the STREAMS I/O subsystem introduced in SVR4, although it is included nowadays in most Unix kernels, and it has become the preferred interface for writing device drivers, terminal drivers, and network protocols.

Several features make Linux a wonderfully unique operating system. Commercial Unix kernels often introduce new features in order to gain a larger slice of the market, but these features are not necessarily useful, stable, or productive. As a matter of fact, modern Unix kernels tend to be quite bloated. By contrast, Linux doesn't suffer from the restrictions and the conditioning imposed by the market, hence it can freely evolve according to the ideas of its designers (mainly Linus Torvalds).

Specifically, Linux offers the following advantages over its commercial competitors: Linux is free. You can install a complete Unix system at no expense other than the hardware (of course). Understanding the Linux Kernel 9 Linux is fully customizable in all its components. Thanks to the General Public License (GPL), you are allowed to freely read and modify the source code of the kernel and of all system programs. Several commercial companies have started to support their products under Linux, most of which aren't distributed under a GNU Public License. Therefore, you may not be allowed to read or modify their source code. Linux runs on low-end, cheap hardware platforms. You can even build a network server using an old Intel 80386 system with 4 MB of RAM. Linux is powerful. Linux systems are very fast, since they fully exploit the features of the hardware components.

The main Linux target is efficiency, and indeed many design choices of commercial variants, like the STREAMS I/O subsystem, have been rejected by Linus because of their implied performance penalty. Linux has a high standard for source code quality. Linux systems are usually very stable; they have a very low failure rate and system maintenance time. The Linux kernel can be very small and compact. Indeed, it is possible to fit both a kernel image and full root filesystem, including all fundamental system programs, on just one 1.4 MB floppy disk! As far as we know, none of the commercial Unix variants is able to boot from a single floppy disk. Linux is highly compatible with many common operating systems. It lets you directly mount filesystems for all versions of MS-DOS and MS Windows, SVR4, OS/2, Mac OS, Solaris, SunOS, NeXTSTEP, many BSD variants, and so on. Linux is also able to operate with many network layers like Ethernet, Fiber Distributed Data Interface (FDDI), High Performance Parallel Interface (HIPPI), IBM's Token Ring, AT&T WaveLAN, DEC RoamAbout DS, and so forth. By using suitable libraries, Linux systems are even able to directly run programs written for other operating systems.

For example, Linux is able to execute applications written for MSDOS, MS Windows, SVR3 and R4, 4.4BSD, SCO Unix, XENIX, and others on the Intel 80x86 platform. Linux is well supported. Believe it or not, it may be a lot easier to get patches and updates for Linux than for any proprietary operating system! The answer to a problem often comes back within a few hours after sending a message to some newsgroup or mailing list. Moreover, drivers for Linux are usually available a few weeks after new hardware products have been introduced on the market. By contrast, hardware manufacturers release device drivers for only a few commercial operating systems, usually the Microsoft ones. Understanding the Linux Kernel 10 Therefore, all commercial Unix variants run on a restricted subset of hardware components. With an estimated installed base of more than 12 million and growing, people who are used to certain creature features that are standard under other operating systems are starting to expect the same from Linux. As such, the demand on Linux developers is also increasing. Luckily, though, Linux has evolved under the close direction of Linus over the years, to accommodate the needs of the masses.

1.4 GRUB in Linux

The GRUB (Grand Unified Bootloader) is a bootloader available from the GNU project. A bootloader is very important as it is impossible to start an operating system without it. It is the first program which starts when the program is switched on. The bootloader transfers the control to the operating system kernel.

GRUB Features

GRUB is the default bootloader for many of the Linux distributions. This is because it is better than many of the previous versions of the bootloaders. Some of its features are:

- GRUB supports LBA (Logical Block Addressing Mode) which puts the addressing conversion used to find files into the firmware of the hard drive
- GRUB provides maximum flexibility in loading the operating systems with required options using a command based, pre-operating system environment.
- The booting options such as kernel parameters can be modified using the GRUB command line.
- There is no need to specify the physical location of the Linux kernel for GRUB. It only required the hard disk number, the partition number and file name of the kernel.
- GRUB can boot almost any operating system using the direct and chain loading boot methods.

GRUB Installation Process

GRUB automatically becomes the default loader after it is installed. The following steps are followed to install GRUB:

- It is important to use the latest GRUB package available to install GRUB. Or the GRUB package from the installation CD-ROM is used.
- The root shell prompt is opened and the command **/sbin/grub-install** is run after the GRUB package is installed. The in the command is the location where the GRUB stage 1 boot loader should be installed.
- After all this is done, the GRUB graphical boot loader menu appears before the kernel loads into memory when the system boots.

GRUB Boot Process

The boot process using GRUB requires the GRUB to load itself into memory. This is done in the following steps:

- The stage 1 boot loader is loaded into the memory by the BIOS. This boot loader is also known as the primary boot loader. It exists on 512 bytes or less of disk space within the master boot record. The primary boot loader can load the stage 1.5 or stage 2 boot loader if required.

- The stage 1.5 boot loader is loaded into the memory by the stage 1 boot loader if required. This may be necessary in some cases as some hardware require a middle step before moving on to the stage 2 loader.
- The secondary boot loader is also known as the stage 2 boot loader and it can be loaded into the memory by the primary boot loader. Display of the GRUB menu and command environment are functions performed by the secondary boot loader. This allows the user to look at system parameters and select the operating system to boot.

- The operating system or kernel is loaded into the memory by the secondary boot loader. After that, the control of the machine is transferred to the operating system.

GRUB Interfaces

There are three interfaces in GRUB which all provide different levels of functionality. The Linux kernel can be booted by the users with the help of these interfaces. Details about the interfaces are:

Menu Interface

The GRUB is configured by the installation program in the menu interface. It is the default interface available. It contains a list of the operating systems or kernels which is ordered by name. A specific operating system or kernel can be selected using the arrow keys and it can be booted using the enter key.

Menu Entry Editor Interface

The e key in the boot loader menu is used to access the menu entry editor. All the GRUB commands for the particular menu entry are displayed there and these commands may be altered before loading the operating system.

Command Line Interface

This interface is the most basic GRUB interface, but it grants the most control to the user. Using the command line interface, any command can be executed by typing it and then pressing enter. This interface also features some advanced shell features.

GRUB vs GRUB2

The default menu for GRUB2 looks very similar to GRUB but there are some changes made in this.

- Grub has two configuration files namely menu. lst and grub. conf whereas, Grub2 has only one main configuration file namely grub.cfg and it looks very close to a full scripting language. And this configuration file is overwritten by certain Grub 2 package updates, whenever a kernel is added or removed, or when the user runs update-grub. For any configuration changes, we need to run update-grub to make the changes effective.
- In Grub, it is really hard for the normal user to modify the configuration. But Grub2 is more user-friendly, Grub-mkconfig will automatically changes the configuration.
- In Grub, partition number starts from 0, whereas in Grub2 it starts with 1. The first device is still identified with hd0. These changes can be altered if needed by making some changes to device.map file of the '/etc/grub' folder.
- Grub uses physical and logical addresses to address the disk, it can't even read from new partitions whereas, Grub2 uses UUID to identify a disk thus is more reliable. It supports LVM and RAID devices.

- In today's Linux Distros including (Ubuntu 16.04 and RHEL 7), GRUB2 will now directly show a login prompt and no menu is displayed now.
- If you want to see the menu during boot you need to hold down SHIFT key. Even sometimes by pressing ESC you can also display the menu.
- Users have also now choice of creating custom files in which they can place their own menu entries. You can make use of a file called 40_custom which is available in '/etc/grub.d' folder.
- Even users can now change the menu display settings. This is done through a file called grub located in /etc/default folder.

1.5 Inter Process Communication (IPC)

A process can be of two types:

- Independent process.
- Co-operating process.

An independent process is not affected by the execution of other processes while a co-operating process can be affected by other executing processes. Though one can think that those processes, which are running independently, will execute very efficiently, in reality, there are many situations when co-operative nature can be utilized for increasing computational speed, convenience, and modularity. Inter-process communication (IPC) is a mechanism that allows processes to communicate with each other and synchronize their actions. The communication between these processes can be seen as a method of co-operation between them. Processes can communicate with each other through both:

1. Shared Memory
2. Message passing

Figure 1 below shows a basic structure of communication between processes via the shared memory method and via the message passing method.

An operating system can implement both methods of communication. First, we will discuss the shared memory methods of communication and then message passing. Communication between processes using shared memory requires processes to share some variable, and it completely depends on how the programmer will implement it. One way of communication using shared memory can be imagined like this: Suppose process1 and process2 are executing simultaneously, and they share some resources or use some information from another process. Process1 generates information about certain computations or resources being used and keeps it as a record in shared memory. When process2 needs to use the shared information, it will check in the record stored in shared memory and take note of the information generated by process1 and act accordingly. Processes can use shared memory for extracting information as a record from another process as well as for delivering any specific information to other processes.

Let's discuss an example of communication between processes using the shared memory method.

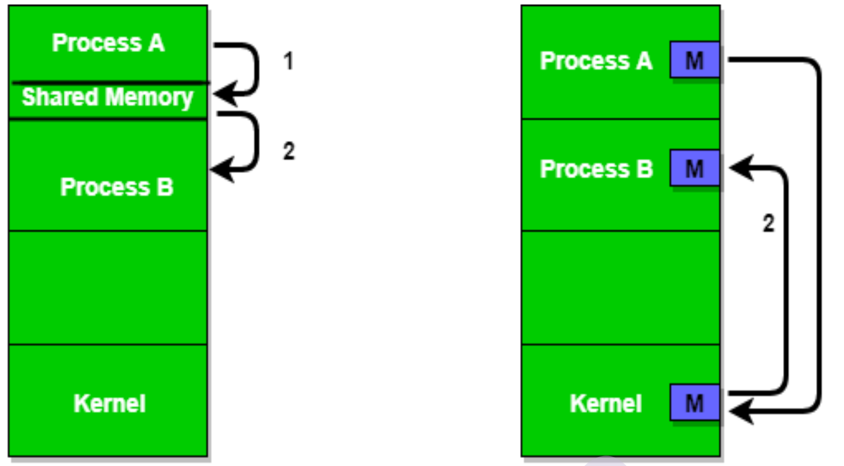
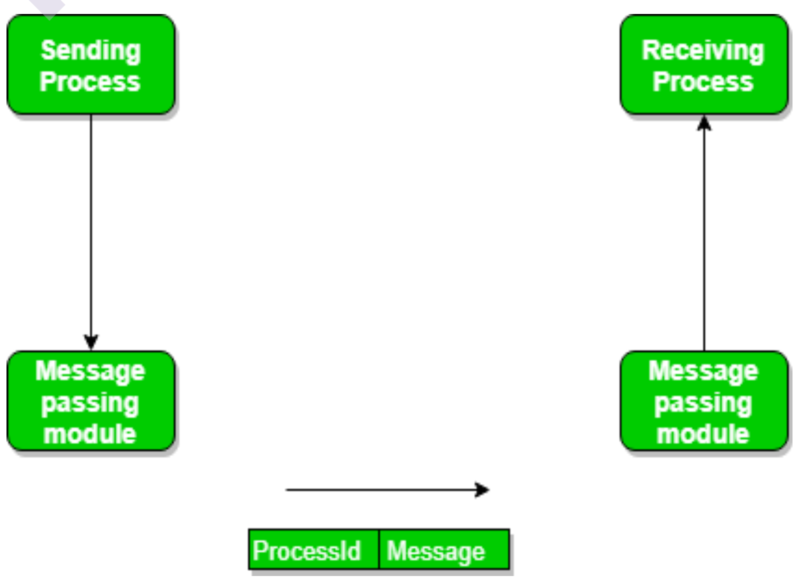


Figure 1 - Shared Memory and Message Passing

ii) Messaging Passing Method

Now, We will start our discussion of the communication between processes via message passing. In this method, processes communicate with each other without using any kind of shared memory. If two processes p1 and p2 want to communicate with each other, they proceed as follows:

- Establish a communication link (if a link already exists, no need to establish it again.)
- Start exchanging messages using basic primitives. We need at least two primitives:
  – **send** (message, destination) or **send**(message)
  – **receive** (message, host) or **receive**(message)



The message size can be of fixed size or of variable size. If it is of fixed size, it is easy for an OS designer but complicated for a programmer and if it is of variable size then it is easy for a programmer but complicated for the OS designer. A standard message can have two parts: **header and body.**

The **header part** is used for storing message type, destination id, source id, message length, and control information. The control information contains information like what to do if runs out of buffer space, sequence number, priority. Generally, message is sent using FIFO style.

Process Scheduling in Linux Scheduling is the action of assigning *resources* to perform *tasks.* We will mainly focus on scheduling where our *resource* is a processor or multiple processors, and the *task* will be a thread or a process that needs to be executed. The act of scheduling is carried out by a process called **scheduler.**

The scheduler goals are to

- Maximize *throughput* (number of tasks done per time unit)
- Minimize *wait time* (amount of time passed since the process was ready until it started to execute)
- Minimize *response time* (amount of time passed since the process was ready until it finished executing)
- Maximize *fairness* (distributing resources fairly for each task)

Process Types in Linux

Linux has two types of processes

- Real-time Processes
- Conventional Processes

Real-time processes are required to 'obey' response time constraints without any regard to the system's load. In different words, real-time processes are **urgent and cannot be delayed** no matter the circumstances. An example of a real-time process in Linux is the migration process which is responsible for distributing processes across CPU cores (a.k.a load balancing).

Conventional processes don't have strict response time constraints and they can suffer from delays in case the system is 'busy'. Each process type has a different scheduling algorithm, and as long as there are ready-to-run real-time processes they will run and make the conventional processes wait.

Real-Time Scheduling

There are two scheduling policies when it comes to real-time scheduling, SCHED_RR and SCHED_FIFO. The policy affects how much runtime a process will get and how is the **runqueue** is operating. The ready-to-run processes I have mentioned are stored in a queue called runqueue. The scheduler is picking processes to run from this runqueue based on the policy.

SCHED_FIFO

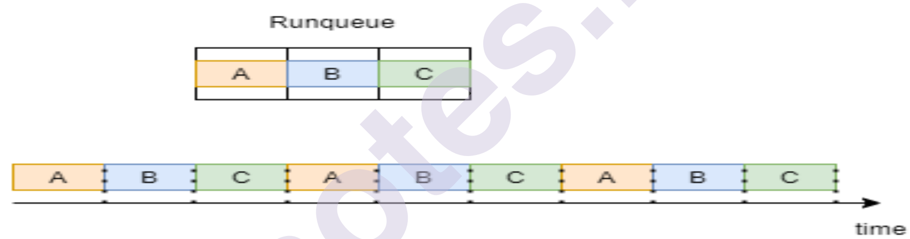
In this policy the scheduler will choose a process based on the arrival time (FIFO = First In First Out). A process with a scheduling policy of SCHED_FIFO can 'give up' the CPU under a few circumstances:

- Process is waiting, for example for an IO operation. When the process is back to 'ready' state it will go back to the end of the runqueue.
- Process yielded the CPU, with the system call *sched_yield.* The process will immediately go back to the end of the runqueue.

SCHED_RR

RR = Round Robin

In this scheduling policy, every process in the runqueue gets a time slice (quantum) and executes in his turn (based on priority) in a cyclic fashion. Let's consider an example where we have 3 processes in our runqueue, A B C, all of them have the policy of SCHED_RR. As shown in the drawing below, each process gets a time slice and executes in his turn. when all processes ran 1 time, they repeat the same execution order.



Real-Time Scheduling Summary

A real-time process can be scheduled in two different policies, SCHED_FIFO and SCHED_RR. The policy affects how the runqueue is working and how much time each process is getting for execution.

Conventional Scheduling

CFS — Completely Fair Scheduler is the scheduling algorithm of conventional processes since version 2.6.23 of Linux.So CFS is focusing mainly on one metric — it wants to be fair as much as possible, meaning that he gives every process gets an even time slice of the CPU. **Note that**, processes with higher priority might still get bigger time slices. In order for us to understand how CFS works, we will have to get familiar with a new term — virtual runtime (vruntime).

Virtual Runtime

Virtual runtime of a process is the amount of time spent by actually executing, not including any form of waiting. As we mentioned, CFS tries to be as fair as possible.

To accomplish that, CFS will schedule the process with the minimum virtual time that is ready to run. CFS maintains variables holding the maximum and minimum virtual runtime for reasons we will understand soon.

CFS — Completely Fair Scheduler

Before talking about how the algorithm works, let's understand what data structure this algorithm is using. CFS uses a red-black tree which is a balanced binary search tree — meaning that insertion, deletion, and look-up are performed in O(logN) where N is the number of processes.

The key in this tree is the **virtual runtime** of a process. New processes or processes that got back to the ready state from waiting are inserted into the tree with a key vruntime=min_vruntime. This is extremely important in order to prevent starvation of older processes in the tree. Moving on to the algorithm, at first, the algorithm sets itself a time limit — sched_latency.

In this time limit, it will try to execute already processes — N. This means that each process will get a time slice of the time limit divided by the number of processes — Q_i = sched_latency/N.

When a process finishes its time-slice (Q_i), the algorithm picks the process with the least virtual runtime in the tree to execute next.

1.6 Let us Sum Up

- Linux is a member of the large family of Unix-like operating systems
- The microkernel and monolithic kernels are two types of kernels in the operating system. The kernel is the main part of the OS.
- The GRUB (Grand Unified Bootloader) is a bootloader available from the GNU project. A bootloader is very important as it is impossible to start an operating system without it.
- Inter-process communication (IPC) is a mechanism that allows processes to communicate with each other and synchronize their actions.
- Process Scheduling in Linux Scheduling is the action of assigning *resources* to perform *tasks.*

1.7 List of References

- Linux Pocket Guide is a book written by Daniel J. Barrett
- The Complete Reference is a book written by Richard Petersen
- Linux Kernel Development is a book written by Robert Love
- The Linux Programming Interface is a book written by Michael Kerrisk

1.8 Bibliography

- Linux Kernel Development is a book written by Robert Love
- The Linux Programming Interface is a book written by Michael Kerrisk

1.9 Unit End Exercise

1. Explain Linux Kernel.
2. What are the types of Kernels?
3. Explain Operating system booting process.
4. Mention the difference between GRUB-I, GRUB-II.
5. Explain Inter Processes communication.
6. Explain various scheduling.

❄❄❄❄❄❄❄

2

MEMORY MANAGEMENT AND VIRTUAL MEMORY IN LINUX

Unit Structure

2.0 OBJECTIVES

After going through this unit, you will be able to:

- Understand the fundamentals of operating system
- Acquaint with the operating system mechanism in handling and managing the process and threads along with their communication
- Understand the mechanisms and conceptualize the components involved in designing the memory management in contemporary operating system

2.1 INTRODUCTION

Memory is a valuable resource that must be maintained properly. While today's average home computer has two thousand times the memory of the IBM 7094 (the world's largest computer in the early 1960s), applications and the data they are expected to process have also increased dramatically. "Programs and their data expand to fill the memory available to contain them," says Parkinson's Law. We'll look at how operating systems handle and manage memory in this chapter.

Every programmer dreams of having an endlessly huge, infinitely fast memory that is also nonvolatile, meaning it does not lose its contents if the power goes off. Why not ask for it to be reasonably priced while we're at it? Unfortunately, technology is unable to transform such fantasies into reality. As a result, most computers contain a memory hierarchy, with a tiny amount of extremely fast, expensive volatile cache memory, hundreds of megabytes of medium-speed, medium-price volatile main memory (RAM), and tens or hundreds of gigabytes of slow, inexpensive nonvolatile disc storage. The operating system's role is to coordinate how these memories are utilized.

The memory manager is the element of the operating system that handles the memory hierarchy. Its role is to keep track of which bits of memory are in use and which are not, to allocate memory to processes when they require it and deallocate it when they are finished, and to manage swapping between main memory and disc when main memory is insufficient to accommodate all processes. It's in the kernel on most systems (excluding MINIX 3).

In this chapter, we'll look at a variety of memory management strategies, ranging from the most basic to the most complex. We'll start at the beginning, looking at the most basic memory management system available, and work our way up to more complex systems.

2.2 BASIC MEMORY MANAGEMENT

There are two types of memory management systems: those that swap processes between main memory and disc during execution (swapping and paging) and those that don't. Keep in mind that swapping and paging are largely artifacts of a lack of main memory that can hold all apps and data at the same time. If primary memory grows to the point that there is genuinely enough of it, arguments for one memory management technique or another may become obsolete.

On the other hand, as previously said, software appears to increase at the same rate as memory, so effective memory management may be required at all times. Many institutions in the 1980s used a 4 MB VAX to run a timesharing system with dozens of (mostly satisfied) users. For a single-user Windows XP machine, Microsoft now recommends at least 128 MB. The trend toward multimedia places even greater demands on memory, therefore good memory management will be required for at least the next decade.

2.2.1. MONOPROGRAMMING WITHOUT SWAPPING OR PAGING

The simplest memory management approach is to execute only one application at a time, with that program and the operating system sharing memory. Figure 2.1 depicts three variations on this topic. The operating system may be in RAM (Random Access Memory) at the bottom of memory, as shown in figure 2.1(a), or in ROM (Read-Only Memory) at the top of memory, as shown in figure 2.1(b), or the device drivers may be in ROM at the top of memory, with the rest of the system in RAM down below, as shown in figure 2.1(c). The first model was once common on mainframes and minicomputers, but it is now rarely seen. On some palmtop computers and embedded systems, the second model is used. Early on, the third model was employed. Personal computers (e.g., those running MS-DOS), where the BIOS is the piece of the system stored in the ROM (Basic Input Output System).

Only one process can execute at a time when the system is organized this way. The operating system copies the desired application from disc to memory and executes it as soon as the user forms a command. The operating system shows a prompt character and waits for a new command when the process is complete. It loads a new program into memory, overwriting the old one, when it receives the command.

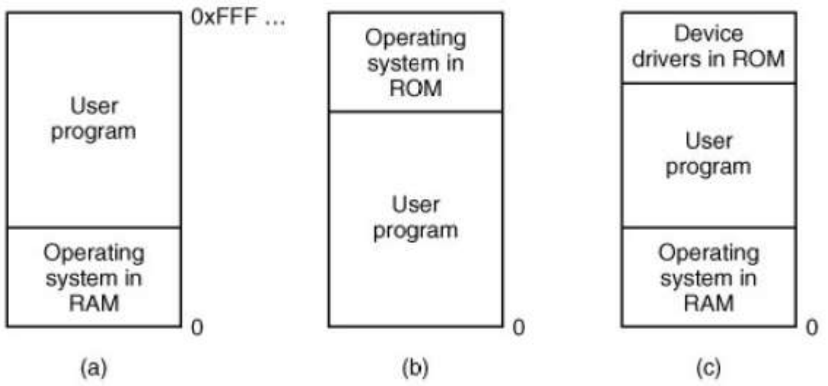


Figure 2.1 With one operating system and one user process, there are three easy ways to organize memory. Other options are also available.

2.2.2. MULTIPROGRAMMING WITH FIXED PARTITIONS

Monoprogramming is rarely implemented these days, with the exception of very small embedded devices. Multiple processes can run at the same time in most modern systems. When many processes are operating at the same time, one can use the CPU while the other is waiting for I/O to complete. As a result, multiprogramming improves CPU usage. Although network servers can always execute several processes (for distinct clients) at the same time, most client (i.e., desktop) systems now have this capability as well.

The simplest method for achieving multiprogramming is to divide memory into n (potentially uneven) segments. This partitioning can be done manually, for example, when the machine is booted.

When a job comes in, it can be placed in the input queue for the smallest partition that can accommodate it. Because the partitions in this system are fixed, any space in a partition that is not used by a work is wasted while that process is running. This system of fixed partitions and independent input queues is depicted in figure 2.2(a).

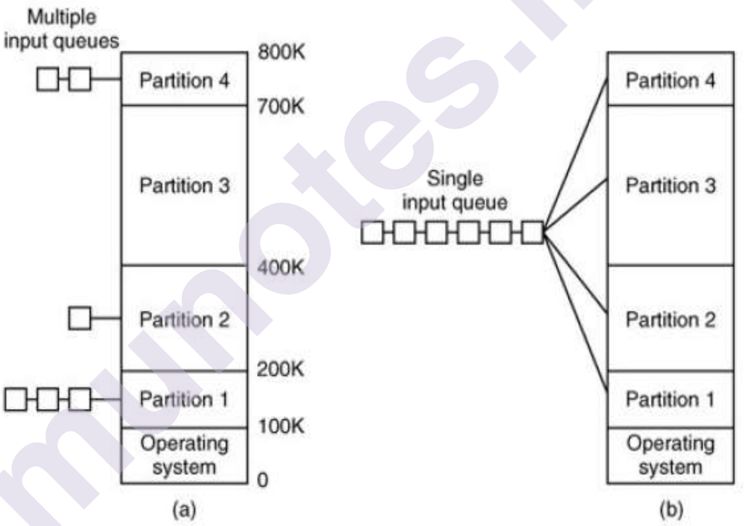


Figure 2.2: (a) Memory partitions are fixed, and each partition has its own input queue.

(b) Memory partitions are fixed, and each partition has a single input queue.

When the large partition of queue is vacant but the queue for a small partition is filled, as is the situation for partitions 1 and 3 in figure 2.2 (a), the disadvantage of dividing the incoming jobs into different queues becomes obvious. Even if there is plenty of memory available, little jobs must wait to get into memory. Maintaining a single queue, as shown in figure 2.2 (b), is an alternate arrangement. Whenever a partition becomes available, the work closest to the top of the queue that fits in it could be loaded and executed.

As wasting a large partition on a tiny project is undesirable, another technique is to examine the whole input queue whenever a partition becomes available and select the largest job that fits. It's worth noting that the latter approach considers little tasks as unworthy of a full partition, whereas it's normally preferable to provide the best service to the smallest jobs (often interactive activities), not the worst.

Having at least one little partition around is one way out. Small jobs will be able to run on this partition without the need for a huge partition. Another option is to establish a rule that no job that is eligible to run may be skipped over more than k times. It receives one point for each time it is skipped over. It cannot be skipped once it has accumulated k points.

2.2.3. RELOCATION AND PROTECTION

Multiprogramming brings two key issues that must be addressed: relocation and privacy. Separate jobs will be run at different addresses, as shown in Figure 2.2. When a program is linked, the linker needs to know where in memory the program will start.

Assume an example that the first instruction is a call to a procedure located at absolute address 100 in the binary file generated by the linker. This program will jump to absolute address 100, which is inside the operating system, if it is loaded in partition 1 (at address 100K). A call to 100K + 100 is all that is required. If the program is loaded into partition 2, the call to 200K + 100, and so on, must be made. This issue is called as the relocation problem.

One option is to change the instructions while the program is being loaded into memory. 100K is added to each address in program put into partition 1, 200K is added to addresses in program loaded into partition 2, and so on. To conduct this type of relocation during loading, the linker must include a list or bitmap in the binary program that specifies which program words are addresses to be relocated and which are opcodes, constants, or other elements that must not be relocated.

The problem of protection is not solved by relocating during loading. A malicious application can create a new instruction and jump to it at any time. There is no mechanism to prevent a program from creating an instruction that reads or writes any word in memory since program in this system use absolute memory addresses rather than addresses relative to a register. Allowing processes to read and write memory belonging to other users is very undesirable in multiuser systems.

Equipping the machine with two unique hardware registers, known as the base and limit registers, is an alternate solution to both the relocation and protection difficulties. When a process is scheduled, the start address of its partition is loaded into the base register, and the length of the partition is placed into the limit register. Before being transmitted to memory, every memory address is automatically supplemented with the contents of the base register. Thus, if the base register has the value 100K, a CALL 100 instruction becomes a CALL 100K + 100 instructions without changing the instruction itself. The limit register is also verified to ensure that addresses do not attempt to target memory outside the current partition. The base and limit registers are protected by hardware to prevent user program from changing them.

The necessity to do an addition and a comparison on each memory reference is a disadvantage of this technique. Although comparisons are quick, addition takes a long time due to carry propagation time unless specific addition circuits are employed.

2.3 SWAPPING

Organizing memory into fixed segments is straightforward and effective using a batch system. When a work reaches the front of the queue, it is loaded into a partition. It is retained in memory until it is completed. There's no reason to use anything more elaborate if you can keep enough jobs in memory to keep the CPU active all the time.

The situation is different with timesharing systems or graphics-oriented personal computers. When main memory is insufficient to accommodate all of the presently running processes, extra processes must be stored on disc and brought in to execute dynamically.

Depending on the hardware available, there are two broad techniques to memory management. Swapping is the most basic technique, which involves bringing in each process in its whole, executing it for a bit, and then putting it back on the disc. Virtual memory, on the other hand, allows applications to run even if they are only partially in main memory.

In figure 2.3, the operation of a switching system is depicted. Only process A is initially stored in memory. Processes B and C are then either created or swapped in from memory. A is swapped off to storage in figure 2.3(d). Then D arrives, and B exits. Finally, A arrives. Because A is now in a different location, the addresses it contains must be redirected, either by software when it is swapped in or (most likely) by hardware during program execution.

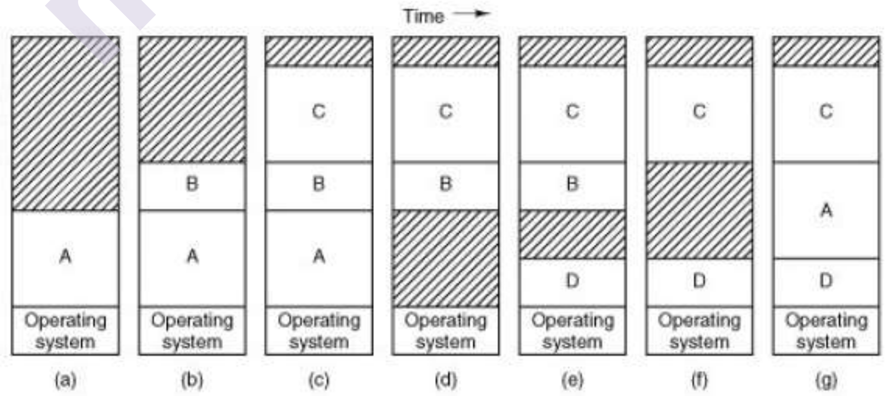


Figure 2.3 As processes enter and exit memory, the memory allocation changes. The shaded areas are memory that hasn't been used yet.

When swapping generates many memory holes, they can be merged into one large one by shifting all the processes as far down as possible. This concept is known as memory compaction. It is frequently avoided since it consumes a significant amount of CPU time.

One thing worth mentioning is the amount of memory that should be assigned to a process when it is created or swapped in. When processes are established with a constant size, the allocation is straightforward: the operating system allocates exactly what is required, no more and no less.

If, on the other hand, processes' data segments can grow by dynamically allocating memory from a heap, as many programming languages allow, a problem arises whenever a process attempts to grow. If there is a hole adjacent to the process, it can be allocated and the process permitted to develop into it. If, on the other hand, the expanding process is adjacent to another process, it will either have to be transferred to a memory hole large enough for it, or one or more processes will have to be swapped out to make room. If a process can't grow in memory and the swap area on the disc is full, it'll have to wait or die.

If most processes are expected to grow as they run, allocating a little more memory whenever a process is swapped in or moved is probably a good idea to reduce the overhead involved with moving or swapping processes that no longer fit in their assigned memory. When switching processes to disc, however, only the memory that is really in use should be changed; exchanging the extra RAM is wasteful. Figure 2.4(a) shows a memory arrangement in which two processes have been given space for expansion.

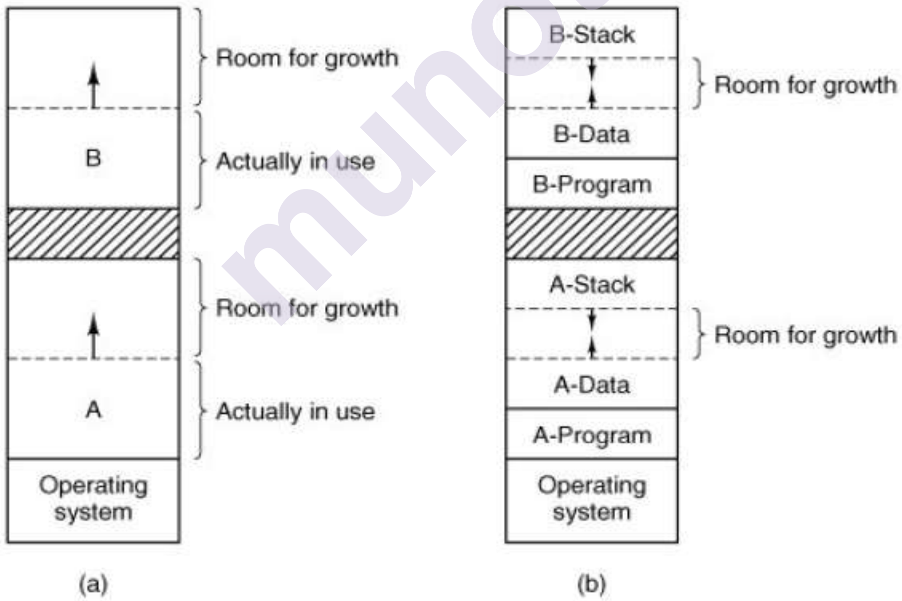


Figure 2.4: (a) Making space for an expanding data portion.
(b) Setting aside space for a rising stack and data segment.

If processes can have two expanding segments, for example, the data segment as a heap for dynamically created and released variables and a stack segment for typical local variables and return addresses, an alternate design, shown in figure 2.4(b), emerges. We can see that each process has a stack at the top of its allotted memory that is increasing downward, as well as a data segment just beyond the program text that is rising upward in this diagram. Either segment can make use of the shared memory between them. If it runs out, the process must be transferred to a hole with enough space, swapped out of memory until a larger hole can be created, or destroyed.

2.3.1. MEMORY MANAGEMENT WITH BITMAPS

When memory is dynamically allotted, the operating system is responsible for managing it. Bitmaps and free lists are the two most common techniques to keep track of memory use.

A bitmap divides memory into allocation units, which can be many kilobytes in size. To each of the allocation unit a bit in the bitmap corresponds to 0 if the unit is free and 1 if it is occupied (or vice versa). Figure 2.5 depicts a section of memory and the bitmap that corresponds.

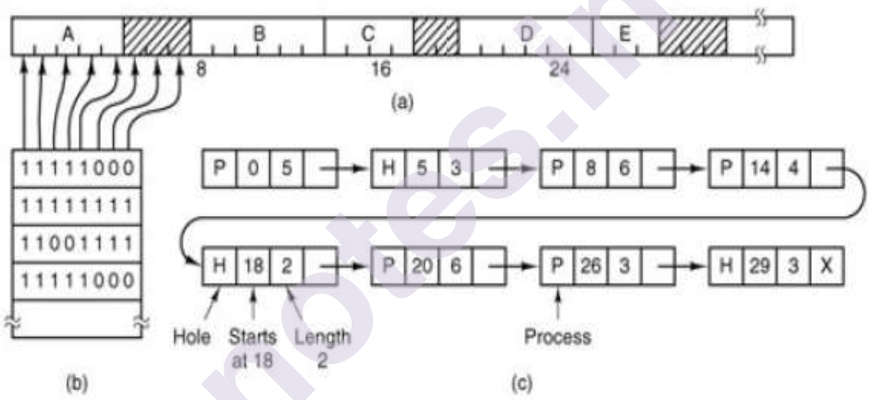


Figure 2.5: (a) A section of memory that contains five processes and three holes. The memory allocation units are indicated by tick marks. Shaded areas (0 in the bitmap) are unrestricted.
(b) Its related bitmap.
(c) The same data as in a list.

The size of the allocation unit is a crucial design consideration. The greater the bitmap, the smaller the allocation unit. However, even with a 4-byte allocation unit, 32 bits of memory will only require 1 bit of the map. Because a memory of 32n bits uses n map bits, the bitmap will only take up 1/33 of the memory. The bitmap will be smaller if the allocation unit is large, but if the process size is not an exact multiple of the allocation unit, significant memory will be wasted in the last unit of the process.

Because the size of a bitmap depends only on the size of memory and the size of the allocation unit, it is a straightforward approach to keep track of memory words in a set amount of memory. The main problem is that when a k-unit process is brought into memory, the memory management must scan the bitmap for a sequence of k consecutive 0 bits in the map. Because the run may transcend word boundaries in the map, searching a bitmap for a run of a particular length is a long process; this is an argument against bitmaps.

2.3.2 MEMORY MANAGEMENT WITH LINKED LISTS

Maintaining a linked list of allocated and free memory segments, where a segment is either a process or a gap between two processes, is another technique to keep track of memory. Figure 2.5(c) depicts the memory of figure 2.5(a) as a linked list of segments. Each list entry describes a hole (H) or process (P), as well as the location at which it begins, the length, and a pointer to the next entry.

The segment list is kept ordered by address in this example. The advantage of sorting this method is that updating the list is simple when a process ends or is replaced. Except when it's at the very top or very bottom of memory, a terminating process usually has two neighbors. These could be holes or processes, resulting in the four combinations indicated in figure 2.6. Updating the list in figure 2.6(a) necessitates replacing a P with an H. Two entries are combined into one in figure 2.6(b) and figure 2.6(c), making the list one entry shorter.

Three entries are combined in figure 2.6(d), and two items are eliminated from the list. Because the terminating process's process table slot will usually point to the process's list entry, it may be more convenient to have the list as a double-linked list rather than the single-linked list shown in figure 2.5(c). This format makes it easy to locate the prior entry and determine whether or not a merge is possible.

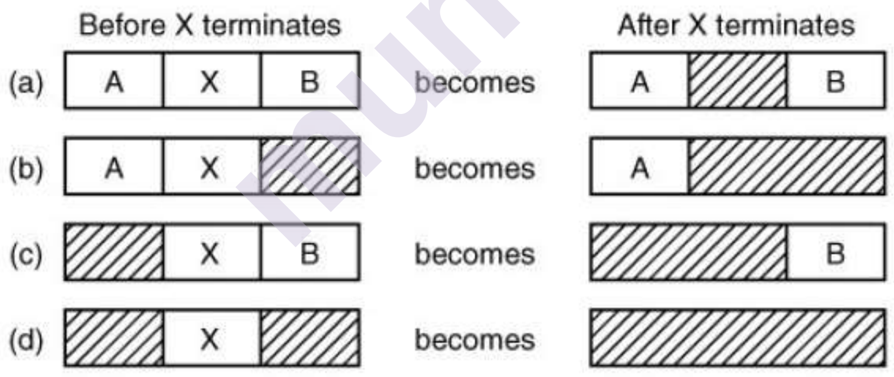


Figure 2.6 For the terminating process, X, there are four neighboring combinations.

Several strategies can be used to allocate memory for a newly generated process (or an old process being swapped in from disc) when the processes and holes are kept on a list sorted by address. The memory management, we suppose, knows how much memory to allocate. First fit is the simplest algorithm. The process manager checks the list of segments until it locates a large enough hole. Except in the statistically unusual situation of a precise fit, the hole is then split into two portions, one for the process and one for the unused memory. Because it searches as little as possible, first fit is a fast algorithm.

Next fit is a small variant of first fit. It functions similarly to initial fit, with the exception that it retains track of its location whenever it finds a suitable hole. When it's called to discover a hole again, it starts searching the list from where it left off the last time, rather than starting from the beginning, as first fit does. According to Bays (1977) simulations, following fit performs somewhat worse than first fit. Best fit is another well-known method. Best fit scans the entire list for the tiniest hole that is suitable. Rather than splitting up a large hole that may be needed later, best fit looks for a hole that is near to the actual size required.

Consider figure 2.5 as an example of initial fit and best fit. If a block of size 2 is required, the hole will be allocated at 5, but the hole will be allocated at 18. Because it must search the complete list every time it is invoked, best fit is slower than first fit. It also resulted in more wasted memory than first fit or next fit because it tends to fill memory with tiny, useless holes. On average, the first fit produces larger holes.

To get around the difficulty of breaking up nearly precise matches into a process and a little hole, consider the worst fit approach, which is to always select the largest available hole, ensuring that the hole broken off is large enough to be useful. Worst fit has also been proved to be a bad concept through simulation. By keeping distinct lists for processes and holes, all four algorithms can be made faster. As a result, they can focus all of their attention on holes rather than processes. Because a freed segment must be deleted from the process list and entered into the hole list, the additional complexity and slowdown when deallocating memory is an unavoidable cost of this allocation speedup. If separate lists for processes and holes are kept, the hole list can be sorted by size to find the best fit faster. When best fit examines a list of holes from smallest to largest, it recognizes that the hole that fits is the smallest one that will perform the task, resulting in the best fit. As with the single list technique, no additional searching is required. First fit and best fit are equally fast with a hole list organized by size, and next fit is meaningless. A slight optimization is achievable when the holes are kept on separate lists from the processes. The holes themselves can be utilized instead of a distinct set of data structures for keeping the hole list, as shown in figure 2.5(c). Each hole's first word may represent the hole size, while the second word could be a link to the next item. Figure 2.5(c) which requires three words and one bit (P/H) are no longer required.

2.4 VIRTUAL MEMORY

People were initially confronted with programs that were too large to fit in the available memory many years ago. The most common technique was to divide the programs into sections known as overlays. Overlay 0 would be the first to run. It would then request for another overlay when it was finished. Some overlay systems were extremely complicated, allowing many overlays to be stored in memory at the same time. The overlays were stored on disc and dynamically swapped in and out of memory by the operating system as needed.

Although the system did the actual work of shifting overlays in and out, the programmer was responsible for deciding how to partition the program into sections. It took a long time and was tedious to break down enormous programs into small, modular bits. It didn't take long for someone to come up with a means to automate the entire process.

Virtual memory is the name given to the method that was invented (Fotheringham, 1961). Virtual memory works on the premise that the total size of the program, data, and stack may exceed the amount of physical memory accessible. The operating system keeps the bits of the program in main memory that is currently in use, and the remainder on the disc.

2.4.1 PAGING

Paging is a memory management strategy that does away with the need for contiguous physical memory allocation. This approach allows a process's physical address space to be non-contiguous.

- Logical Address or Virtual Address (Represented in bits): The CPU generates an address.
- The set of all logical addresses generated by a program (expressed in words or bytes) is known as the logical address space or virtual address space.
- Physical Address (in bits): An address that is physically existent on the memory unit.
- The set of all physical addresses that correspond to the logical addresses (expressed in words or bytes) is known as the Physical Address Space.

Example:

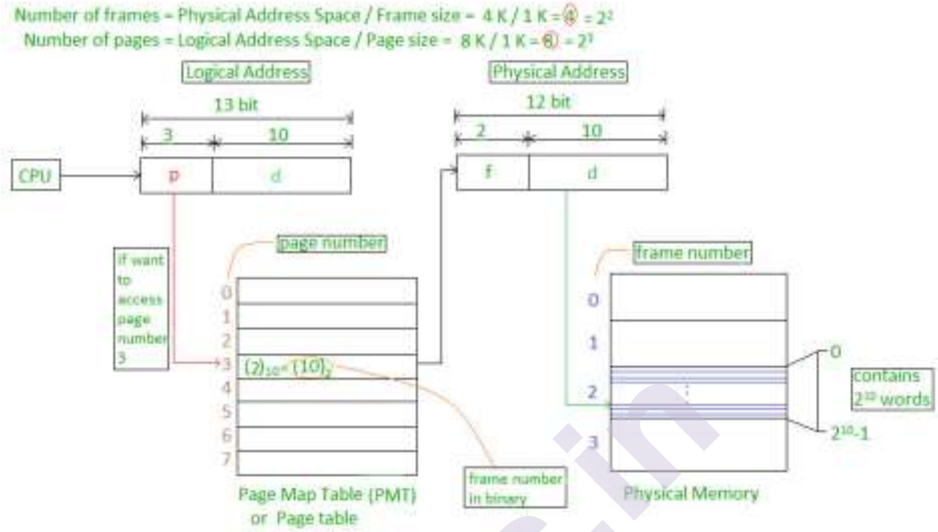
- Logical Address Space = 2^{31} words = 2 G words (1 G = 2^{30}) if Logical Address is 31 bits.
- Logical Address = $\log_2 2^{27}$ = 27 bits if Logical Address Space = 128 M words = $2^7 * 2^{20}$ words.
- Physical Address Space = 2^{22} words = 4 M words (1 M = 2^{20}) if Physical Address is 22 bits.
- Physical Address = $\log_2 2^{24}$ = 24 bits if Physical Address Space = 16 M words = $2^4 * 2^{20}$ words.

The memory management unit (MMU), which is a hardware component, performs the mapping from virtual to physical address, which is known as the paging mechanism.

- The Physical Address Space is organized into frames, which are fixed-size pieces of data.
- The Logical Address Space is also divided into pages, which are fixed-size blocks.
- Page Dimensions = Frame Dimensions

Consider the following scenario:

- When the Physical Address is 12 bits, the Physical Address Space is 4 kilobytes.
- If the logical address is 13 bits, the address space is 8 K words.
- 1 K words Equals page size = frame size (assumption)



The address generated by the CPU is categorized into two parts.

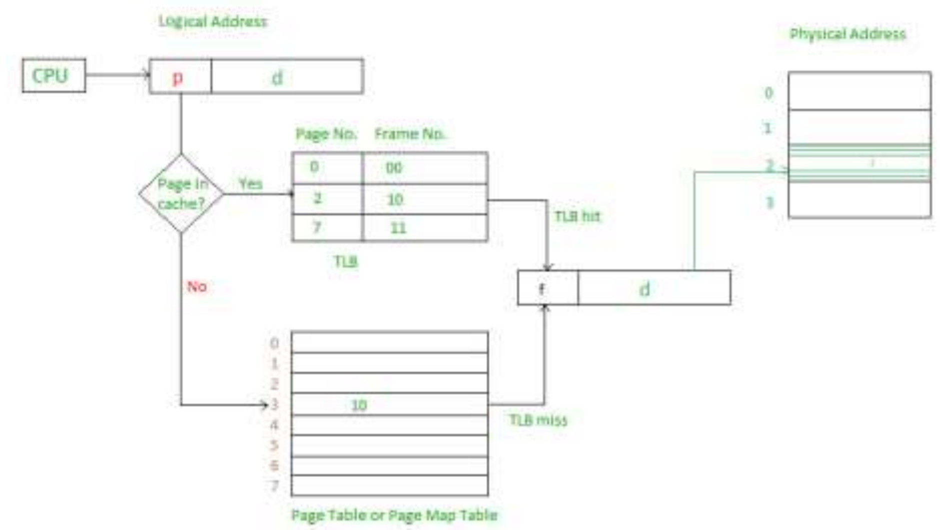
- Page Number (p): It indicates the number of bits required to indicate pages in Logical Address Space
- Page offset (d): It indicates the number of bits necessary to represent a certain word in a page, the size of a page in Logical Address Space, the word number of a page, or the offset of a page.

The physical address is separated into two parts.

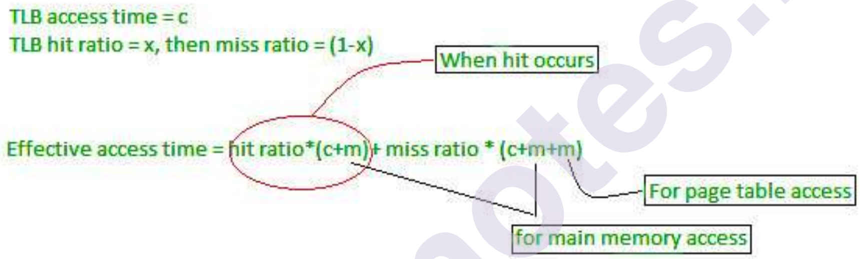
- Frame number (f): The number of bits necessary to represent a frame of Physical Address Space.
- Frame offset (d): The number of bits necessary to represent a certain word in a frame, or the physical address space frame size, or the word number of a frame, or the frame offset.

Dedicated registers can be used to implement the page table in hardware. However, using a register for the page table is only useful if the page table is tiny. We can employ TLB (translation Look-aside buffer), a particular, small, fast look-up hardware cache, if the page table has a significant number of entries.

- The TLB is a high-speed, associative memory.
- TLB entries are made up of two parts: a tag and a value.
- When this memory is accessed, an item is compared to all tags at the same time. If the object is located, the value associated with it is returned.



Time to access main memory = m
Effective access time = m (for page table) + m (for main memory) if page tables are kept in main memory (for particular page in page table)



2.4.2 PAGE TABLE

The Page Table is a data structure that is used by the virtual memory system in the operating system of a computer to record the mapping between physical and logical addresses. With the help of the page table, the logical address created by the CPU is converted into a physical address. As a result, the page table primarily supplies the relevant frame number (frame base address) where that page is stored in main memory. Figure 2.7 represents the paging model of physical and logical memory.

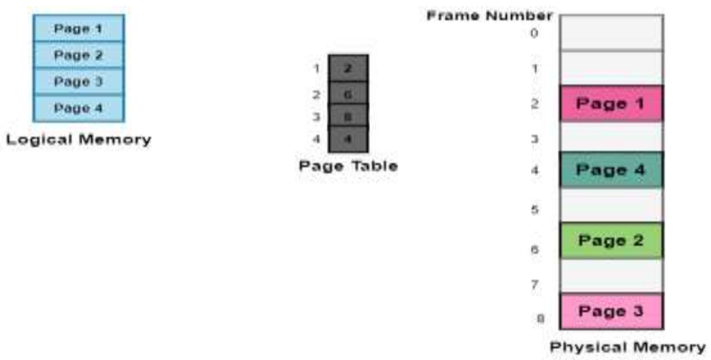


Figure 2.7 Physical and logical memory paging model

Characteristics of the Page Table

The following are some of the features of the Page Table:

- It's saved in the system's main memory.
- In general, the number of entries in the page table equals the number of pages divided by the procedure.
- PTBR stands for page table base register, and it is used to store the base address for the current process's page table.
- Each process has its own table of contents.

Techniques used for structuring the Page Table

The following are some of the most popular approaches for structuring the Page table:

[1] Hierarchical Paging

[2] Hashed Page Tables

[3] Inverted Page Tables

[1] Hierarchical Paging

It is also known as multilevel paging. If the page table is too large to accommodate in a single place, we may need to create a hierarchy with multiple levels. The logical address space is divided into multiple page tables in this sort of paging. One of the simplest ways is hierarchical paging, which may be accomplished using a two-level page table or a three-level page table.

- **Two-level page Table**
  Consider a system with a 32-bit logical address space and a 1 KB page size, which is partitioned into:
- The page number is made up of 22 bits.
- Page Offset is a 10-bit value.

  As we page the Page table, the page number is further separated into
- The page number is made up of 12 bits.
- Page Offset is a 10-bit value.

  As a result, the logical address is:

Page Number		Page Offset
P1	P2	d
12	10	10

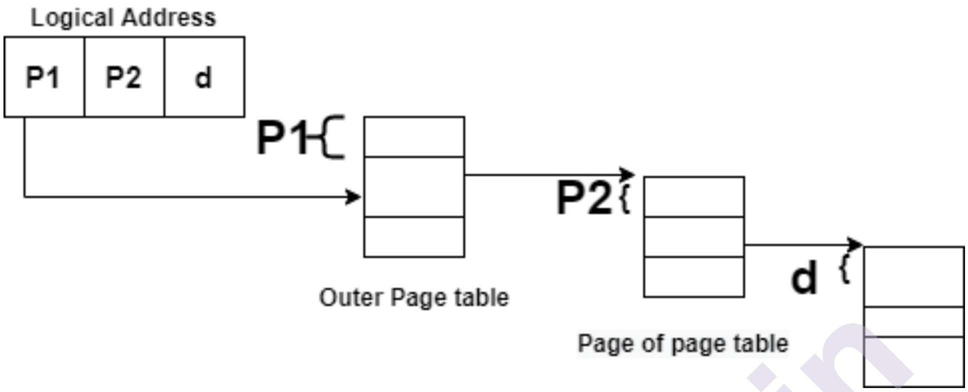
In the diagram above,

- P1 is the Outer Page table's index.

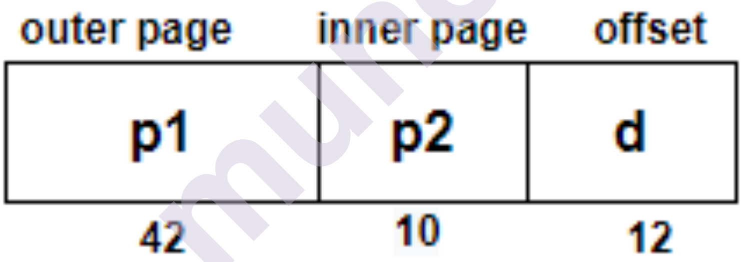
- The displacement within the page of the Inner page Table is indicated by P2.

Forward-mapped Page Table is so named because address translation operates from the outer page table inward.

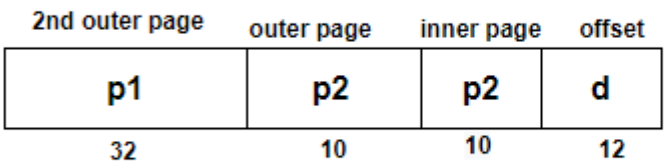
The Address Translation Scheme for a Two-Level Page Table is shown in the diagram below.



- **Three-level page Table**
  A two-level paging technique is not suited for a system with a 64-bit logical address space. Let's pretend the page size is 4KB in this example. If we apply the two-page level method in this situation, the addresses will appear as in the image below.



To prevent creating such a big table, divide the outer page table, which will result in a three-level page table:



[2] Hashed Page Tables

This method is used to deal with address spaces larger than 32 bits. The number is hashed into a page table on this virtual page. This Page table consists primarily of a chain of elements that hashes to the same elements.

The following are the main components of each element:

- The number of the virtual page
- The mapped page frame's value.
- A pointer to the linked list's next element.

The Hashed Page Table's address translation technique is shown in the diagram below:

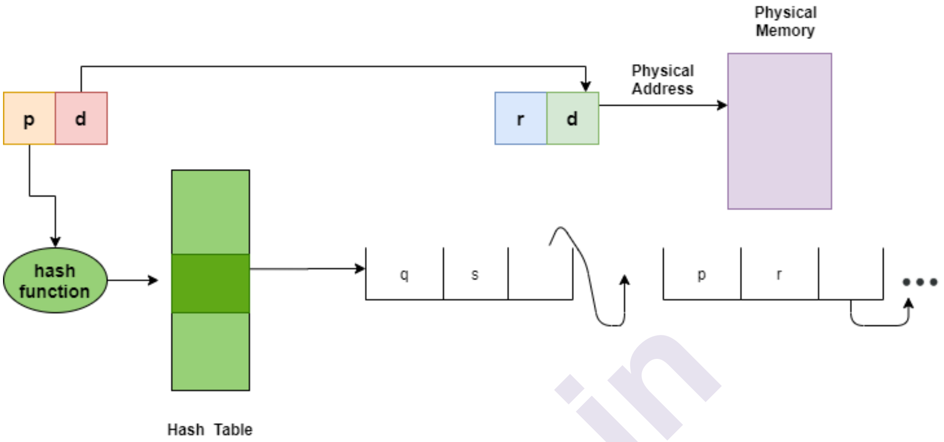


Figure 2.8 Hashed Page Table

In this chain, the Virtual Page numbers are compared for a match; if a match is discovered, the matching physical frame is extracted. Clustered page tables are often used in this approach for 64-bit address space.

- **Clustered Page Tables**
  These are similar to hashed tables, but instead of one page, each item links to many pages (i.e. 16). Typically utilized in sparse address spaces where memory references are dispersed and non-contiguous.

[3] Inverted Page Tables

The Inverted Page table is a data structure that combines a page table and a frame table into one. Each virtual page number and real memory page has their own entries. The virtual address of the page stored in that real memory location, as well as information about the process that owns the page, make up the majority of the entry. While this strategy reduces the amount of memory required to store each page table, it also increases the time required to search the table whenever a page reference is encountered.

The address translation scheme of the Inverted Page Table is shown in the diagram below:

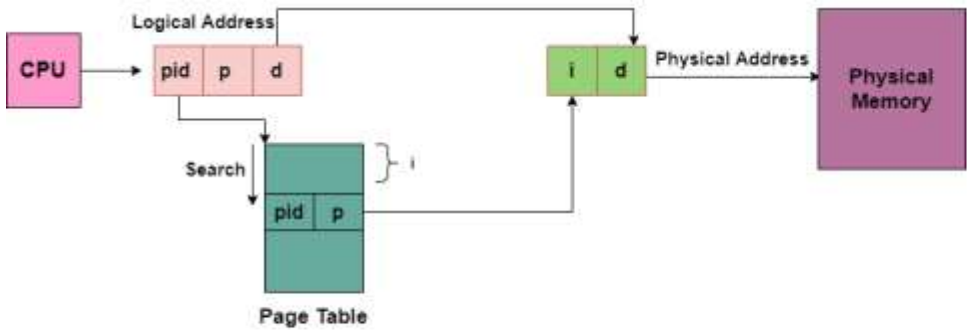


Figure 2.9 Inverted page table's address translation scheme

Because numerous processes may have the identical logical addresses, we must keep track of the process id of each entry. After running through the hash function, numerous entries can map to the same index in the page table. As a result, chaining is utilized to deal with this.

2.4.3 Translation Look aside Buffers

- **Drawbacks of Paging:**
  1] The size of a Page table might be quite large, resulting in a waste of main memory.
  2] Reading a single word from the main memory will take longer on the CPU.

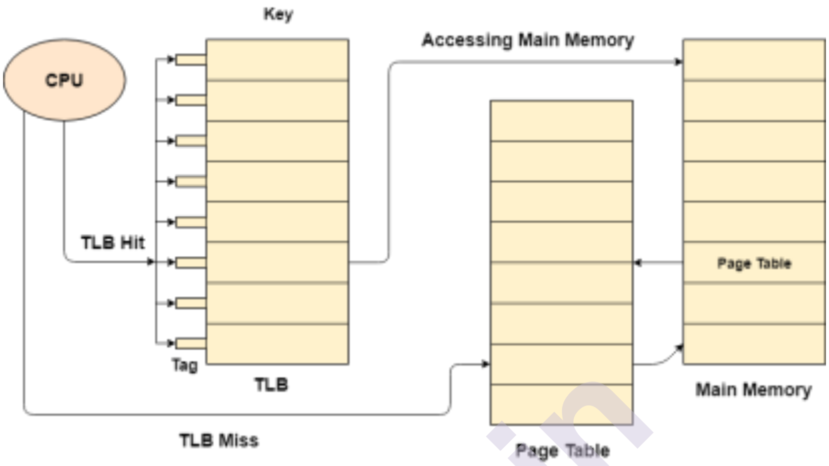
- **How to decrease the page size table**
  1] The size of the page table can be reduced by raising the page size, however this will result in internal fragmentation and page waste.
  2] Another option is to use multilevel paging, but this increases the effective access time and is therefore not a viable option.

- **How to decrease the effective access time**
  1] The CPU can utilize a register with the page table stored inside it to reduce the time it takes to access the page table, but the registers are not inexpensive and are small in comparison to the page table size, so this is not a realistic solution.
  2] To solve these numerous paging flaws, we must seek out a memory that is less expensive than the register and faster than the main memory, allowing the CPU to focus on accessing the actual word rather than repeatedly accessing the page table.

- **Locality of reference**
  The notion of locality of reference in operating systems asserts that, rather than loading the complete process in main memory, the OS can load only the number of pages in main memory that are regularly accessed by the CPU, as well as the page table entries that correspond to those many pages.

- **Translation look aside buffer (TLB)**
  A translation look aside buffer is a memory cache that can be utilized to reduce the time it takes to repeatedly access the page

table. It's a memory cache that's closer to the CPU; therefore it takes the CPU less time to access TLB than it does to access main memory. To put it another way, TLB is faster and smaller than main memory, but it is also cheaper and larger than the register. TLB adheres to the principle of locality of reference, which means it only stores the entries of the many pages that the CPU accesses regularly.



In translation look aside the buffers; there are tags and keys that are used to map data. When the requested entry is located in the translation look aside buffer, it is referred to as a TLB hit. When this occurs, the CPU just accesses the actual location in main memory. If the item isn't located in the TLB (TLB miss), the CPU must first read the page table in main memory, then the actual frame in main memory. As a result, the effective access time in the case of a TLB hit will be less than in the case of a TLB miss. As a result, the effective access time can be calculated as follows:

$EAT = P (t + m) + (1 - p) (t + k.m + m)$

Where,

'p' is the TLB hit rate,

't' is the time taken to access TLB,

'm' indicates the time taken to access main memory k = 1, if the single level paging has been implemented.

We can deduce from the formula that

1] If the TLB hit rate is increased, the effective access time will be reduced.

2] In the case of multilevel paging, the effective access time will be increased.

2.5 PAGE REPLACEMENT ALGORITHMS

When a new page needs to be loaded into the main memory, the Page Replacement Algorithm determines which page to remove, also known as swap out. When a requested page is not present in the main memory and the available space is insufficient to allocate to the requested page, Page Replacement occurs.

When the page chosen for replacement is paged out and referenced again, it must read in from disc, which necessitates I/O completion. The quality of the page replacement method is determined by this process: the less time spent waiting for page-ins, the better.

A page replacement algorithm tries to determine which pages should be replaced in order to reduce the frequency of page misses. There are numerous page replacement algorithms to choose from. These algorithms are tested by executing them on a specific memory reference string and counting the number of page faults. The method for that circumstance is better if there are less page faults. When a process requests a page and that page is found in main memory, it is referred to as a page hit; otherwise, it is referred to as a page miss or a page fault.

There are several page replacement algorithms in operating system as indicated in the diagram 2.10 below

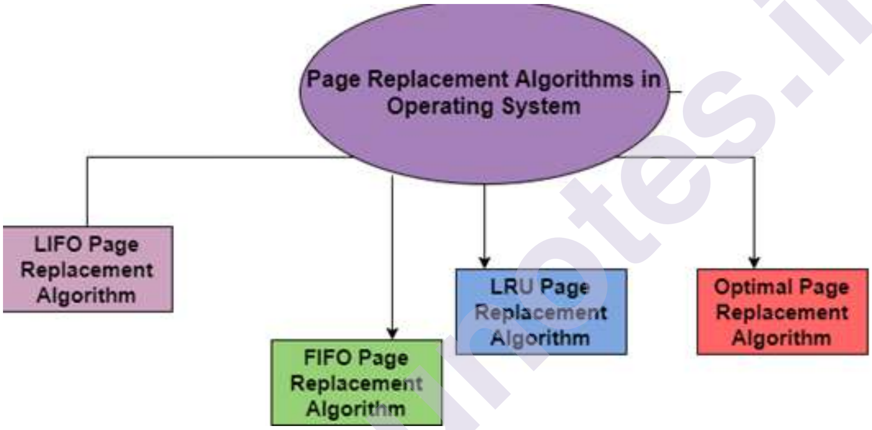


Figure 2.10 Various page replacement algorithms

2.5.1 First In First Out (FIFO)

This is the most basic page replacement method. The OS maintains a queue in this algorithm, with the oldest page at the front and the most recent page at the back, to keep track of all the pages in memory.

When a page needs to be replaced, the FIFO algorithm replaces the page at the front of the queue, which is the page that has been in memory the longest.

- **EXAMPLE:**

  Consider the following page reference string of size 12: 1, 2, 3, 4, 5, 1, 3, 1, 6, 3, 2, 3 with frame size 4 (i.e. maximum 4 pages in a frame).

1	2	3	4	5	1	3	1	6	3	2	3

1	1	1	1	5	5	5	5	5	5	2	2
	2	2	2	2	1	1	1	1	1	1	1
		3	3	3	3	3	3	6	6	6	6
			4	4	4	4	4	4	3	3	3

M	M	M	M	M	M	H	H	M	M	M	H

M = Miss
H = Hit

Total Page Fault=9

All four spaces are initially empty, therefore when 1, 2, 3, and 4 arrive, they are assigned to the empty spots in the order of their arrival. This is a page fault because the numbers 1, 2, 3, and 4 are not in memory.

When page 5 arrives, it is not available in memory, so a page fault occurs, and the oldest page in memory, 1, is replaced.

Because 1 is not in memory when it arrives, a page fault occurs, and it replaces the oldest page in memory, i.e. 2.

When 3,1 arrives, it is already in the memory, i.e., Page Hit, therefore there is no need to update it.

When page 3 arrives, it is not in memory, therefore a page fault occurs, and the oldest page in memory, 4, is replaced.

When page 2 arrives, it is not in memory, therefore a page fault occurs, and the oldest page in memory, 5, is replaced.

When 3 arrives, it is already in the memory, i.e., Page Hit, therefore there is no need to update it.

Page Fault Ratio = 9/12, i.e. total miss/total cases possible

- **Advantages**

  1] Simple and straightforward to implement.

  2] Low overhead.

- **Disadvantages**

  1] Poor performance.

  2] It doesn't take into account how often you use it or when you last used it; it just changes the oldest page.

  3] Belady's Anomaly affects this algorithm (i.e. more page faults when we increase the number of page frames).

2.5.2 Least Recently Used (LRU)

The Least Recently Used page replacement algorithm maintains track of how many times a page has been used in a short period of time. It is based on the assumption that the pages that have been widely utilized in the past will also be heavily used in the future.

When page replacement occurs in LRU, the page that has not been utilized for the longest period is replaced.

- **EXAMPLE:**

1	2	3	4	5	1	3	1	6	3	2	3

1	1	1	1	5	5	5	5	5	5	2	2
	2	2	2	2	1	1	1	1	1	1	1
		3	3	3	3	3	3	3	3	3	3
			4	4	4	4	4	6	6	6	6

M	M	M	M	M	M	H	H	M	H	M	H

M = Miss
H = Hit

Total Page Fault: 8

All four spaces are initially empty, therefore when 1, 2, 3, and 4 arrive; they are assigned to the empty spots in the order of their arrival. This is a page fault because the numbers 1, 2, 3, and 4 are not in memory.

Because 5 is not in memory when it arrives, a page fault occurs, and it replaces 1 as the least recently utilized page.

When 1 arrives, it is not in memory, therefore a page fault occurs, and it takes the place of 2.

When 3,1 arrives, it is already in the memory, i.e., Page Hit, therefore there is no need to update it.

When 6 arrives, it is not found in memory, causing a page fault, and it takes the place of 4.

When 2 arrives, it is not found in memory, causing a page fault, and it takes the place of 5.

When 3 arrives, it is already in the memory, i.e., Page Hit, therefore there is no need to update it.

Page Fault Ratio = 8/12

- **Advantages**

  1] Efficient

  2] Not affected by Belady's Anomaly.

- **Disadvantages**

  1] Implementation is difficult.

  2] Expensive.

  3] Hardware support is required.

2.5.3 Optimal Page Replacement

The best page replacement algorithm is the Optimal Page Replacement algorithm, which produces the fewest page faults. This algorithm is also known as OPT that stands for clairvoyant replacement algorithm, or

Belady's optimal page replacement policy. This algorithm replaces pages that will not be used for the longest period of time in the future, i.e., pages in the memory that will be referred to the farthest in the future.

This approach was first proposed a long time ago and is difficult to execute since it necessitates future knowledge about program behavior. Using the page reference information obtained on the first run, however, it is possible to implement optimal page replacement on the second run.

- **EXAMPLE:**

1	2	3	4	5	1	3	1	6	3	2	3

1	1	1	1	1	1	1	1	6	6	6	6
	2	2	2	2	2	2	2	2	2	2	2
		3	3	3	3	3	3	3	3	3	3
			4	5	5	5	5	5	5	5	5

M	M	M	M	M	H	H	H	M	H	H	H

M = Miss
H = Hit

Total Page Fault=6

All four spaces are initially empty, therefore when 1, 2, 3, and 4 arrive, they are assigned to the empty spots in the order of their arrival. This is a page fault because the numbers 1, 2, 3, and 4 are not in memory.

When 5 arrives, it is not in memory, causing a page fault, and it substitutes 4, which will be utilized the most in the future among 1, 2, 3, and 4.

When 1,3,1 arrives, they are already in the memory, i.e., Page Hit, therefore there is no need to update them.

When 6 arrives, it is not found in memory, causing a page fault, and it takes the place of 1.

When 3, 2, 3 appears, it is already in the memory, i.e., Page Hit, thus there is no need to update it.

Page Fault Ratio = 6/12

- **Advantages**

1] Implementation is simple.
2] The data structures are simple.
3] Extremely effective.

- **Disadvantages**

1] Future knowledge of the program is required.
2] Time-consuming.

2.5.4 Last In First Out (LIFO)

The FIFO principle is comparable to how this method works. The newest page, which is the last to arrive in the primary memory, gets replaced in this way. This algorithm uses the stack to keep track of all the pages.

The last items entered are the first to be eliminated in the LIFO technique of data processing. FIFO (First In, First Out) is the contrary of LIFO, in which objects are deleted in the order they were entered.

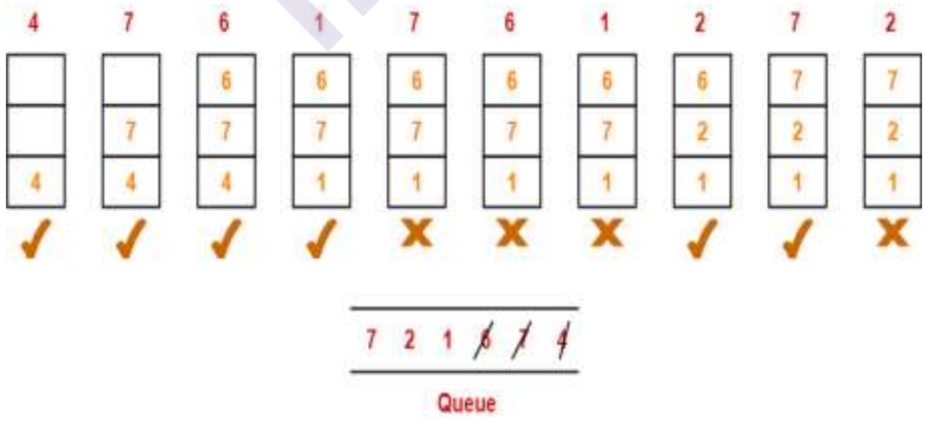
Imagine stacking a deck of cards by laying one card on top of the other, starting at the bottom, to help understand LIFO. You begin removing cards from the top of the deck after it has been entirely stacked. Because the last cards to be placed on the deck are the first to be removed, this procedure is an example of the LIFO approach.

When pulling data from an array or data buffer, computers sometimes employ the LIFO approach. The LIFO method is used when a computer needs to access the most recent data entered. The FIFO approach is utilized when data must be retrieved in the order it was entered.

2.5.5 PRACTICE PROBLEMS BASED ON PAGE REPLACEMENT ALGORITHMS

Problem-01: In main memory, a system uses three page frames to store process pages. It employs a FIFO (First in, First Out) page replacement policy. Assume that all of the page frames are blank at first. What is the total number of page faults that will be generated while processing the following page reference string-

4, 7, 6, 1, 7, 6, 1, 2, 7, 2
Calculate the hit and miss ratios as well.
Solution:
Number of total references = 10



From this figure,
Total number of page fault occurred = 6

Calculating Hit ratio-

Total number of page hits

= Total number of references – Total number of page misses or page faults
= 10 – 6
= 4

Thus, Hit ratio
= Total number of page hits / Total number of references
= 4 / 10
= 0.4 or 40%

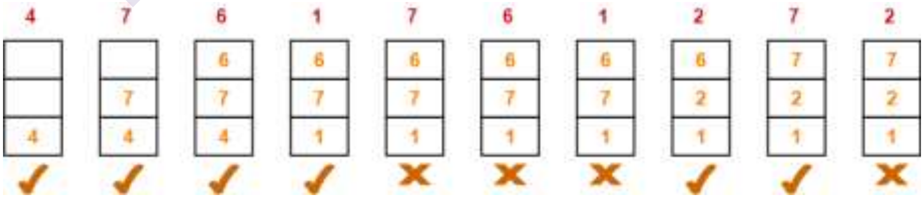
Calculating Miss ratio-

Total number of page misses or page faults = 6
Thus, Miss ratio
= Total number of page misses / Total number of references
= 6 / 10
= 0.6 or 60%

Alternatively,
Miss ratio
= 1 – Hit ratio
= 1 – 0.4
= 0.6 or 60%

Problem-02: In main memory, a system uses three page frames to store process pages. It replaces pages based on the Least Recently Used (LRU) policy. Assume that all of the page frames are blank at first. What is the total number of page faults that will be generated while processing the following page reference string-

4 , 7, 6, 1, 7, 6, 1, 2, 7, 2
Calculate the hit and miss ratios as well.
Solution:
Number of total references = 10



From this figure,
Total number of page fault occurred = 6
Solving same as above-

- Hit ratio = 0.4 or 40%
- Miss ratio = 0.6 or 60%

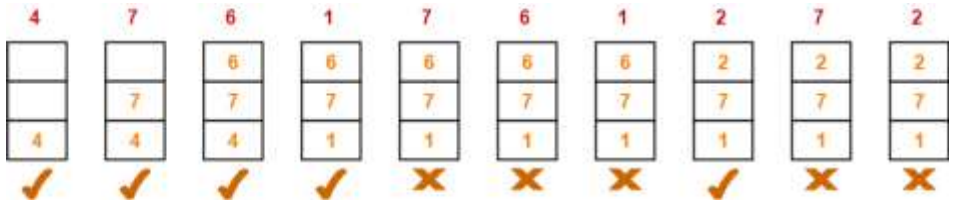
Problem-03: In main memory, a system uses three page frames to store process pages. The Optimal page replacement policy is used. Assume that all of the page frames are blank at first. What is the total number of page faults that will be generated while processing the following page reference string-

4, 7, 6, 1, 7, 6, 1, 2, 7, 2

Calculate the hit and miss ratios as well.

Solution:

Number of total references = 10



From this figure,

Total number of page fault occurred = 5

Solving same as above-

- Hit ratio = 0.5 or 50%
- Miss ratio = 0.5 or 50%

2.6 DESIGN ISSUES FOR PAGING SYSTEMS

The difficulties that operating system designers must address in order to acquire optimal performance from a paging system will be discussed in the following sections.

2.6.1 The working set model

Processes are started with none of their pages in memory in the purest form of paging. The CPU receives a page fault as soon as it attempts to acquire the first instruction, requiring the operating system to bring in the page holding the first instruction. Other page faults, such as those affecting global variables and the stack, frequently appear rapidly. After a while, the process has most of the pages it requires and begins to run smoothly with minimal page errors. Demand paging is the name given to a system in which pages are loaded only when they are needed, rather than in advance.

Of course, writing a test program that reads all the pages in a huge address space in a methodical manner, creating so many page faults that there isn't enough memory to keep them all is simple. Thankfully, most procedures do not operate in this manner. They have a locality of reference, which means that during any phase of execution, the process only refers to a small portion of the total number of pages. For example, each pass of a multipass compiler only looks at a fraction of the pages, and a different fraction at that.

The working set refers to the set of pages that a process is currently using. The process will operate without creating many errors if the full working set is in memory until it progresses to the next execution phase (e.g., the next pass of the compiler). Because executing an instruction takes a few nanoseconds and reading a page from the disc takes around 10 milliseconds, if the available memory is insufficient to hold the complete working set, the operation will generate multiple page faults and run slowly. It will take a long time to complete at a rate of one or two instructions per 10 milliseconds. Thrashing is a term used to describe a program that causes page faults every few instructions.

Processes are regularly relocated to disc (i.e., all of their pages are deleted from memory) in a multiprogramming system to give other processes a chance at the CPU. When a process is brought back in, the question of what to do emerges. Nothing needs to be done from a technical standpoint. Until its working set is loaded, the process will only create page faults. The issue is that having 20, 100, or even 1000 page faults every time a process is loaded is slow and wastes a lot of CPU time, because processing a page fault takes the operating system a few milliseconds of CPU time, not to mention a lot of disc I/O.

As a result, many paging systems attempt to keep track of each process' working set and verify that it is in memory before allowing it to operate. The working set model is the name for this method. It is intended to drastically minimize the number of page faults. Prepaging is the process of loading pages before allowing processes to run. It's worth noting that the working set evolves with time.

The operating system must keep track of which pages are in the working set in order to implement the working set concept. The aging algorithm is one technique to keep track of this data. The clock algorithm's performance can be improved by using information about the working set.

2.6.2 Local versus Global Allocation Policies

Several strategies for selecting a page to replace when a defect occurs have been discussed in the preceding sections. The allocation of memory among the competing runnable processes is a major challenge involved with this decision.

Consider figure 2.11 (a). The set of runnable processes in this diagram is made up of three processes: A, B, and C. Assume A receives a page fault. Should the page replacement method look for the least recently used page using only the six pages now allocated to A, or all the pages in memory? When only A's pages are examined, the page with the lowest age value is A5, resulting in the condition depicted in Figure 2.11. (b).

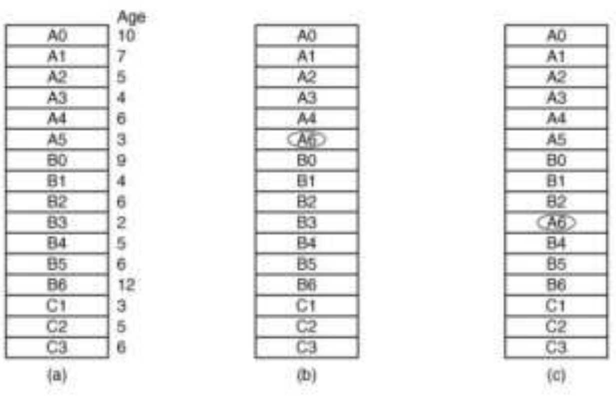


Figure 2.11: Local Vs. Global page replacement

(a) Original Configuration
(b) Local page replacement
(c) Global page replacement

If, on the other hand, the page with the lowest age value is eliminated regardless of whatever page it is, page B3 is chosen, and we will have the scenario as shown in the figure 2.11(c). The algorithm shown in figure 2.11 (b) is a local page replacement algorithm, whereas the algorithm shown in figure 2.11 (c) is a global algorithm. Local algorithms effectively correspond to dedicating a fixed portion of memory to each process. Page frames are dynamically allocated among the runnable processes via global algorithms. As a result, the amount of page frames allocated to each activity changes over time.

Global algorithms perform better in general, especially when the working set size varies throughout the course of an operation. Even if there are lots of free page frames, thrashing will occur if a local algorithm is employed and the working set expands. Local algorithms waste memory as the working set shrinks. If you utilize a global approach, the system will have to decide how many page frames to give each process on a regular basis. One method is to keep an eye on the working set size as indicated by the ageing bits, but this does not guarantee that thrashing will not occur. The working set can grow or shrink in microseconds, but the ageing bits are a rough estimate based on a number of clock ticks.

2.6.3 Page size

The page size is frequently a setting that the operating system can set. Even if the hardware supports 512-byte pages, the operating system can treat pages 0 and 1, 2 and 3, 4 and 5, and so on as 1-KB pages by allocating two consecutive 512-byte page frames for them.

The ideal page size is determined by balancing numerous competing elements. As a result, there is no one-size-fits-all solution. To begin, there are two arguments in favor of a tiny page size. A text, data, or stack segment picked at random will not fill an integral number of pages. Half of the final page will be blank on average. That page's extra space is being squandered. Internal fragmentation is the term for this type of waste. Internal fragmentation will waste np/ 2 bytes with n segments in memory and a page size of p bytes. This supports the idea of a tiny page size.

When we consider a program that consists of eight 4 KB sequential phases, another rationale for a modest page size emerges. With a 32-KB page size, the program must always be given 32 KB. It only takes 16 KB with a 16-KB page size. It only takes 4 KB at any time with a page size of 4 KB or less. A big page size, on average, will result in more unneeded program being stored in memory than a small page size. Small pages, on the other hand, imply that applications will require a high number of pages, necessitating the use of a large page table.

When the CPU switches from one process to another on some machines, the page table must be loaded into hardware registers. A small page size on these machines means that the time required to load the page registers increases as the page size decreases. Furthermore, as the page size lowers, the page table takes up more space.

2.6.4 Virtual Memory Interface

Our entire discussion has been based on the assumption that virtual memory is transparent to processes and programmers. That is, all they perceive on a machine with a smaller physical memory is a big virtual address space. That is true for many systems, but in certain advanced systems, programmers have some control over the memory map and can use it in unconventional ways to improve program behavior. We'll take a look at a couple of them in this section.

Allowing two or more processes to share the same memory is one rationale for providing programmers control over their memory map. If programmers can identify memory areas, it may be conceivable for one process to give the name of a memory region to another so that the latter can map it in. High bandwidth sharing is feasible when two (or more) processes share the same pages: one process writes to the shared memory while the other reads from it.

A high-performance message forwarding system can also be implemented via page sharing. Normally, data is duplicated from one address space to another when messages are passed, which costs a lot of money. A message can be passed by having the sending process unmap the page(s) containing the message and the receiving process map them in if processes can manage their page map. Instead of copying all of the data, only the page names must be copied. Distributed shared memory is yet another advanced memory management technology. The concept is to allow different processes on a network to share a set of pages, maybe as a single shared linear address space, but not necessary. A page fault occurs when a process refers to a page that is not currently mapped in. The page fault handler, which may be in kernel or user space, then locates the machine that is holding the page and sends it a message instructing it to unmap the page and transfer it over the network. The page is mapped in and the faulting instruction is restarted when it comes.

2.7 SEGMENTATION

Segments are used to break down a procedure. Segments are the sections into which a program is separated that are not always all the same size. Another method of splitting accessible memory is segmentation. It's a new memory management technique that generally supports the user's perspective on memory. The logical address space consists primarily of segments. Each section is given a name as well as a length.

A procedure is divided into segments in generally. Segmentation splits or segments the memory in the same way as paging does. However, there is a distinction: paging splits the memory into fixed segments, whereas segmentation divides the memory into variable segments, which are then loaded into logical memory space. A program is essentially a grouping of segments. A segment is a logical unit that includes things like: main program, procedure, function, method, object, local and global variable, symbol table, common block, stack, arrays.

2.7.1 Types of segmentation

The following are the several forms of segmentation:

1] **Virtual Memory Segmentation:** In this sort of segmentation, each process is divided into n divisions, but they are not segmented all at once.

2] **Simple Segmentation:** With this type, each process is divided into n divisions and all of them are segmented at the same time, but during runtime, and they can be non-contiguous (that is they may be scattered in the memory).

2.7.2 Characteristics of segmentation

The following are some characteristics of the segmentation technique:

1] Variable-size partitioning is used in the Segmentation scheme.

2] Segments are the conventional name for supplementary memory partitions.

3] The length of modules determines the partition size.

4] Secondary memory and main memory are thus partitioned into unequal-sized sections using this technique.

2.7.3 Need of segmentation

The separation of the user's image of memory and the real physical memory is one of the major downsides of memory management in the operating system. Paging is a technique that allows these two memories to be separated.

The user's perspective is essentially mapped to physical storage. This mapping also allows for the separation of physical and logical memory. The operating system may partition the same function into multiple pages, which may or may not be loaded into memory at the same time. The operating system is also unconcerned about the user's perspective on the process. The system's efficiency suffers as a result of this strategy. Because it breaks the process into chunks, segmentation is a better technique.

2.7.4 User's view of a program

The user's perspective on segmentation is depicted in the figure 2.12 below

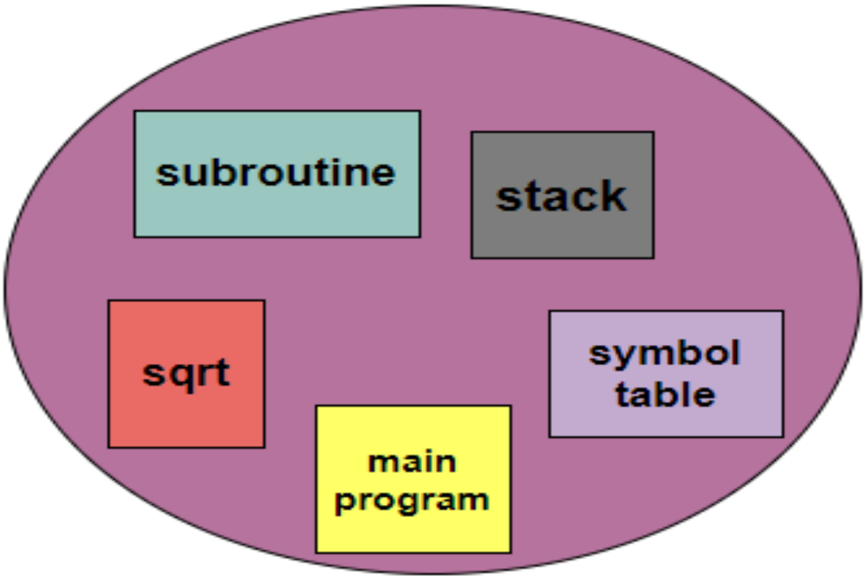


Figure 2.12 User's view of segmentation

Basic Method

A segmented computer system has a logical address space that can be divided into several parts. And the segment's size is changeable, meaning it can expand or shrink. As we previously stated, each segment has a name and length throughout execution. And the address primarily specifies the segment's name as well as its displacement inside the segment. As a result, the user provides each address using two values: segment name and offset. Because implementation segments are numbered rather than named, they are referred to as segment number rather than name.

As a result, the logical address is made up of two tuples:
$\langle$segment-number, offset$\rangle$
where,
Segment Number(s): A Segment Number is a number that represents the number of bits needed to represent a segment.
Offset(d): The amount of bits necessary to express the size of the segment is represented by segment offset.

2.7.5 Segmentation Architecture

Segment Table

The term "Segment Table" refers to a table that is used to store data from all process segments. In this method, there isn't a straightforward link between logical and physical addresses. The segment table is used to convert a two-dimensional logical address to a one-dimensional physical address. This table is primarily stored in the main memory as a distinct segment. The base address of the segment table is stored in a table known as the Segment table base register (STBR)

Each entry in the segment table has the following information:

1] Segment Base/Base Address: The segment base primarily comprises the starting physical address in the memory where the segments are stored.

2] Segment Limit: The segment limit is mostly used to define the segment's length.

Segment Table Base Register (STBR): The STBR register is used to point to the memory location of the segment table.

Segment Table Length Register (STLR): The number of segments used by a program is indicated by this register. If s<STLR, the segment number s is allowed.

	Limit	Base
Segment 0	1400	1400
Segment 1	400	6200
Segment 2	1100	4400
Segment 3	1300	4800

Segment Table

2.7.6 Segmentation Hardware

The segmentation hardware is depicted in the figure 2.13 below

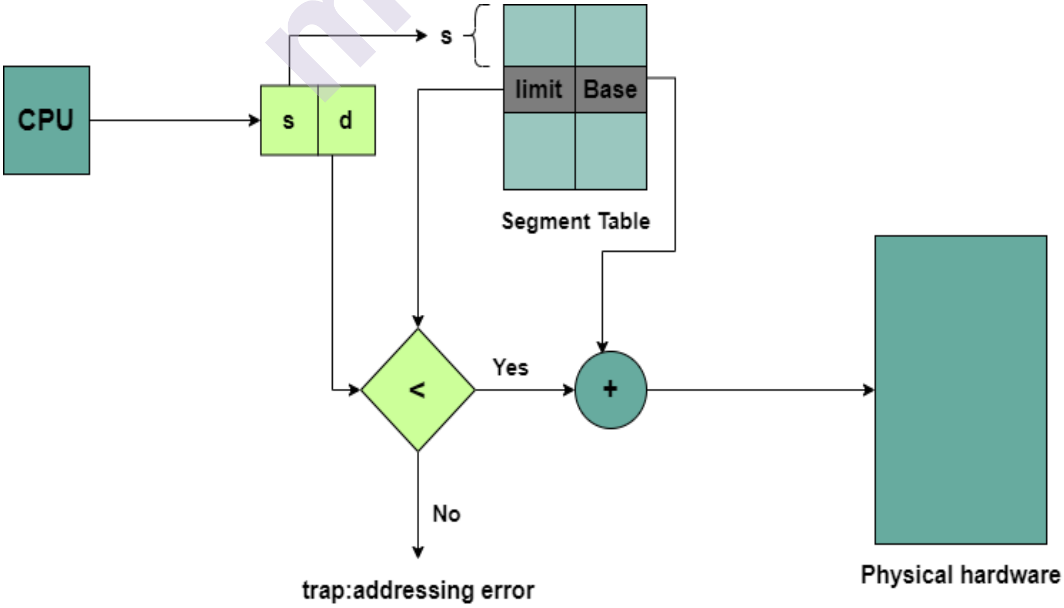


Figure 2.13 Segmentation hardware

The two portions of the logical address generated by the CPU are:

- **Segment Number(s):** It's a key to the segment table.
- **Offset(d):** It must be between '0' and also it should be in a 'segment limit'. If the Offset is greater than the segment limit, the trap is generated.

As a result, correct offset + segment base = physical memory address and a segment table is essentially a collection of base-limit register pairs.

2.7.7 Advantages of Segmentation

The following are some of the benefits of using the segmentation technique:

- The segment table is mostly utilized in the Segmentation technique to keep track of segments. In addition, the segment table takes significantly less space than the paging table.
- Internal Fragmentation does not exist.
- In general, segmentation allows us to separate a program into modules for easier viewing.
- The size of the segments varies.

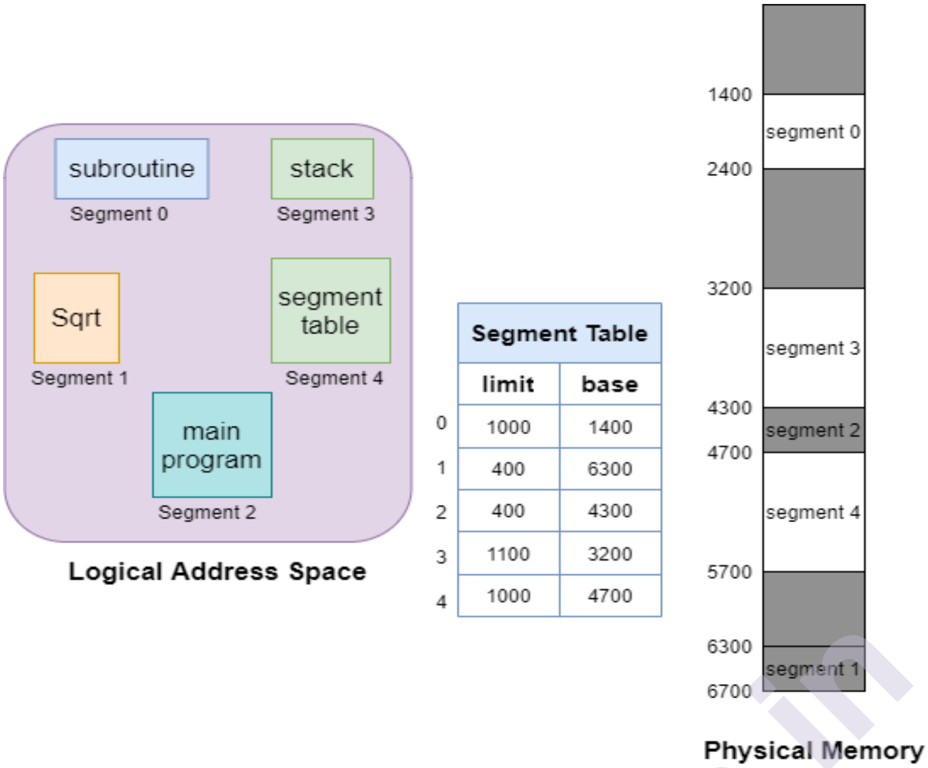
2.7.8 Disadvantages of Segmentation

The following are some of the technique's drawbacks:

- The overhead of maintaining a segment table for each process is significant.
- This method is quite costly.
- Because two memory visits are now necessary, the time it takes to fetch the instruction increases.
- In segmentation, segments are of different sizes and so are not ideal for exchanging.
- As the open space is divided down into smaller bits, and processes are loaded and withdrawn from the main memory, this strategy leads to external fragmentation, resulting in a lot of memory waste.

2.7.9 Example of Segmentation

The segmentation example is shown below, with five segments numbered from 0 to 4. As illustrated, these portions will be stored in physical memory. Each segment has its own entry in the segment table, which provides the segment's beginning entry address in physical memory (referred to as the base) as well as the segment's length (denoted as limit).



Segment 2 starts at position 4300 and is 400 bytes long. As a result, a reference to byte 53 of segment 2 is mapped to position 4300 (4300+53=4353) in this scenario. 3200 (the base of segment 3) + 852=4052 is mapped to a reference to segment 3, byte 85. Because this segment is 1000 bytes long, a reference to byte 1222 of segment 0 would result in a trap to the OS.

2.8 CASE STUDY: LINUX MEMORY MANAGEMENT

- Memory management is one of the most difficult tasks performed by the Linux kernel. It is linked to a number of topics and issues.

- One of the most significant aspects of the operating system is the memory management subsystem. There has always been a demand for more memory than is physically available in a system since the dawn of computing. Virtual memory is the most successful of the strategies explored to overcome this issue. Virtual memory gives the impression that the system has more memory than it actually has by spreading it across competing processes as needed.

- Virtual memory does more than merely extend the memory of your machine. The memory management subsystem allows you to manage your memory.

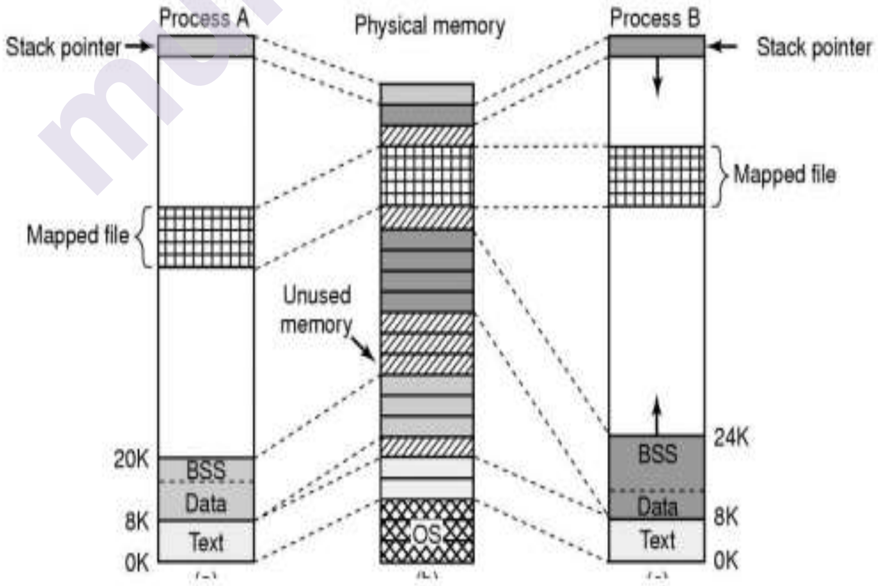
- Large Address Spaces: The operating system gives the impression that the system has more memory than it actually has. The virtual memory in the system can be many times greater than the physical memory.

- Security: Each system process has its own virtual address space. Because these virtual address spaces are fully distinct, a process executing one program will not influence another. Furthermore, the hardware virtual memory techniques enable for the protection of memory sectors against writing. This prevents unauthorized apps from overwriting code and data.

- Memory Mapping: Memory mapping is a technique for mapping picture and data files into the address space of a process. The contents of a file are linked directly into the virtual address space of a process through memory mapping.

- Equitable Physical Memory Allocation: The memory management subsystem allots a fair share of the system's physical memory to each operating process.

- Virtual Memory Sharing: Although virtual memory allows program to have their own (virtual) address space, there are situations when they must share memory. For example, the bash command shell could be used by multiple processes in the system. Rather than having multiple copies of bash, one for each process' virtual address space, it is preferable to have only one copy in physical memory, which all bash-running processes share. Another typical example of executing code shared by several processes is dynamic libraries. Shared memory can also be utilized as an Inter Process Communication (IPC) mechanism, allowing two or more processes to exchange data via shared memory. Unix TM System V shared memory IPC is supported by Linux.



- The virtual and physical memory is separated into pages, which are fixed length blocks of memory.
- The theoretical page table contains the following information for each entry.

- Valid flag: This determines whether or not the page table entry is valid.
- The physical page frame number.

Information on access control: This section explains how to use the page. Is it possible to write to it? Is there any executable code in it?

Linux Memory Management System Calls

System call	Description
s = brk(addr)	Change data segment size
a = mmap(addr, len, prot, flags, fd, offset)	Map a file in
s = unmap(addr, len)	Unmap a file

Physical memory management

Linux separates memory into three zones:

- ZONE DMA - these are pages that can be used to perform DMA operations.
- ZONE NORMAL - pages that are mapped on a regular basis.
- ZONE HIGHMEM - non-permanently mapped pages with high-memory addresses.

2.9 SUMMARY

We looked at memory management in this chapter. The simplest systems, we discovered, do not swap or page at all. When a program is loaded into memory, it stays there until it is completed. This is how most embedded systems function, with the programming maybe even stored in ROM. Some operating systems only allow one process in memory at a time, whilst others allow multiprogramming.

Swapping is the next stage. The system can handle more processes than it has memory for when swapping is used. Processes that don't have enough memory are moved to the disc. A bitmap or a hole list can be used to keep track of free space in memory and on disc.

Virtual memory is commonly found in more modern computers. Each process address space is partitioned into uniformly sized chunks called pages, which can be inserted into any available page frame in memory in their most basic form. There has been a slew of page replacement algorithms proposed. Second chances and ageing are two of the more well-known examples. Choosing an algorithm isn't enough to make paging systems operate successfully; other considerations include establishing the working set, memory allocation policy, and page size.

Segmentation makes it easier to link and share data structures that change in size during execution. It also makes it easier to provide varying levels of protection to different areas. To create a two-dimensional virtual memory, segmentation and paging are sometimes coupled. Segmentation and paging are supported by the Intel Pentium.

2.10 LIST OF REFERENCES

1] An Introduction to Operating Systems: Concepts and Practice (GNU/Linux), 4th edition, Pramod Chandra P. Bhatt, Prentice-Hall of India Pvt. Ltd, 2014

2] Operating System Concepts with Java Eight Edition, Avi Silberschatz, Peter Baer Galvin, Greg Gagne, John Wiley & Sons, Inc., 2009, http://codex.cs.yale.edu/avi/os book/OS8/os8j

3] UNIX and Linux System Administration Handbook, Fourth Edition, Evi Nemeth, Garth Snyder, Tren Hein, Ben Whaley, Pearson Education, Inc, 2011,

4] Operating Systems: Design and Implementation, Third Edition, Andrew S. Tanenbaum, Albert S. Woodhull, Prentice Hall, 2006.

5] https://www.cs.helsinki.fi/u/niklande/opetus/kj/2008/lectures/os-2008-l05-slides

6] https://www.cs.princeton.edu/courses/archive/fall16/cos318/lectures/14.VM-Design.pdf

7] https://www.cs.unm.edu/~cris/481/481.170memory.pdf

8] https://www.cpp.edu/~gsyoung/CS4310/Notes/Part2Memory Management.pdf

9] https://kgut.ac.ir/useruploads/1552306818833dqu.pdf

10] https://www.cs.helsinki.fi/u/niklande/opetus/kj/2008/lectures/os-2008-l05-handout6.pdf

11] https://sritsense.weebly.com/uploads/5/7/2/7/57272303/case_study_on_linux.pdf

2.11 UNIT END EXERCISES

1] Explain the concept of memory management.

2] Write a note on relocation and protection.

3] Write in brief about swapping concept.

4] Discuss the concept of virtual memory.

5] What do you mean by Paging?

6] Explain the concept of Page table.

7] Write a note on various techniques used for structuring the page table.

8] Write a short note on translational look aside buffer.

9] Discuss various page replacement algorithms.

10] Explain First in First Out page replacement algorithm along with an example.

11] Explain with an example the concept of Least Recently Used Algorithm.

12] Discuss on optimal page replacement algorithm.

13] Write a note on Last in First Out page replacement algorithm.

14] Discuss the various design issues for paging system.

15] What is the need of Segmentation?

16] Explain the concept of segmentation and state its characteristics.

17] What are the types of segmentation? Explain in brief.

18] Write a note on segmentation hardware.

19] Explain segmentation architecture and state its advantages, disadvantages along with an example.

20] Discuss on the case study of Linux memory management.

❄❄❄❄❄❄❄

3

INPUT/ OUTPUT IN LINUX

Unit Structure

3.0 OBJECTIVE

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

3.1 HISTORY

Linux is a modern, free operating system based on UNIX standards. First developed as a small but self-contained kernel in 1991 by Linus Torvalds, with the major design goal of UNIX compatibility, released as open source. Its history has been one of collaboration by many users from all around the world, corresponding almost exclusively over the Internet. It has been designed to run efficiently and reliably on common PC hardware, but also runs on a variety of other platforms. The core Linux operating system kernel is entirely original, but it can run much existing free UNIX software, resulting in an entire UNIX-compatible operating system free from proprietary code. Linux system has many, varying Linux distributions including the kernel, applications, and management tools.

3.2 PRINCIPLES OF I/O HARDWARE

Input and Output

To the user, the I/O system in Linux looks much like that in any UNIX system. That is, to the extent possible, all device drivers appear as normal files. A user can open an access channel to a device in the same way she opens any other file—devices can appear as objects within the file system.

The system administrator can create special files within a file system that contain references to a specific device driver, and a user opening such a file will be able to read from and write to the device referenced. By using the normal file-protection system, which determines who can access which file, the administrator can set access permissions for each device. Linux splits all devices into three classes: block devices, character devices, and network devices.

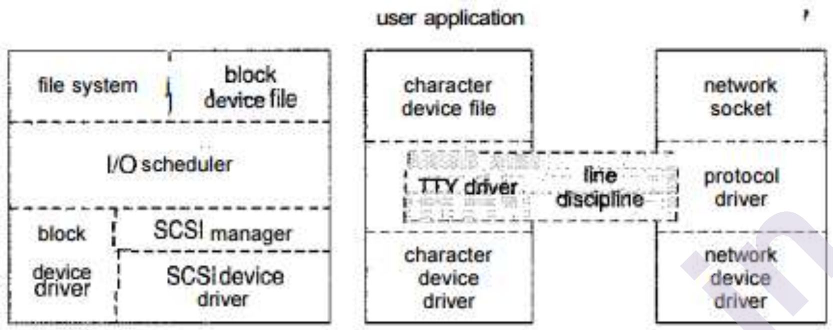


Figure 3.1: Structure of Device Driver System

Figure illustrates the overall structure of the device-driver system. Block devices include all devices that allow random access to completely independent, fixed-sized blocks of data, including hard disks and floppy disks, CD-ROMs, and flash memory. Block devices are typically used to store file systems, but direct access to a block device is also allowed so that programs can create and repair the file system that the device contains.

Applications can also access these block devices directly if they wish; for example, a database application may prefer to perform its own, fine-tuned laying out of data onto the disk, rather than using the general-purpose file system. Character devices include most other devices, such as mice and keyboards. The fundamental difference between block and character devices is random access—block devices may be accessed randomly, while character devices are only accessed serially.

For example, seeking to a certain position in a file might be supported for a DVD but makes no sense to a pointing device such as a mouse. Network devices are dealt with differently from block and character devices. Users cannot directly transfer data to network devices; instead, they must communicate indirectly by opening a connection to the kernel's networking subsystem.

Block Devices

Block devices provide the main interface to all disk devices in a system. Performance is particularly important for disks, and the block-device system must provide functionality to ensure that disk access is as fast as possible. This functionality is achieved through the scheduling of I/O operations In the context of block devices, a block represents the unit with which the kernel performs I/O. When a block is read into memory, it is stored in a buffer. The request manager is the layer of software that manages the reading and writing of buffer contents to and from a block-device driver. A separate list of requests is kept for each block-device driver. Traditionally, these requests have been scheduled according to a unidirectional-elevator (C-SCAN) algorithm that exploits the order in which requests are inserted in and removed from the per-device lists. The request lists are maintained in sorted order of increasing starting-sector number. When a request is accepted for processing by a block-device driver, it is not removed from the list. It is removed only after the I/O is complete, at which point the driver continues with the next request in the list, even if new requests have been inserted into the list before the active request. As new I/O requests are made, the request manager attempts to merge requests in the per-device lists. The scheduling of I/O operations changed somewhat with version 2.6 of the kernel. The fundamental problem with the elevator algorithm is that I/O operations concentrated in a specific region of the disk can result in starvation of requests that need to occur in other regions of the disk.

The deadline I/O scheduler used in version 2.6 works similarly to the elevator algorithm except that it also associates a deadline with each request, thus addressing the starvation issue. By default, the deadline for read requests is 0.5 second and that for write requests is 5 seconds. The deadline scheduler maintains a sorted queue of pending I/O operations sorted by sector number. However, it also maintains two other queues—a read queue for read operations and a write queue for write operations. These two queues are ordered according to deadline.

Every I/O request is placed in both the sorted queue and either the read or the write queue, as appropriate. Ordinarily, I/O operations occur from the sorted queue. However, if a deadline expires for a request in either the read or the write queue, I/O operations are scheduled from the queue containing the expired request. This policy ensures that an I/O operation will wait no longer than its expiration time

Character Devices

A character-device driver can be almost any device driver that does not offer random access to fixed blocks of data. Any character-device drivers registered to the Linux kernel must also register a set of functions that implement the file I/O operations that the driver can handle. The kernel performs almost no preprocessing of a file read or write request to a character device; it simply passes the request to the device in question and lets the device deal with the request.

The main exception to this rule is the special subset of character-device drivers that implement terminal devices. The kernel maintains a standard interface to these drivers by means of a set of tty_struc t structures. Each of these structures provides buffering and flow control on the data stream from the terminal device and feeds those data to a line discipline.

A line discipline is an interpreter for the information from the terminal device. The most common line discipline is the tt y discipline, which glues the terminal's data stream onto the standard input and output streams of a user's running processes, allowing those processes to communicate directly with user's terminal. This job is complicated by the fact that several such processes may be running simultaneously, and the tt y line discipline is responsible for attaching and detaching the terminal's input and output from the various processes connected to it as those processes are suspended or awakened by the user.

Other line disciplines also are implemented that have nothing to do with I/O to a user process. The PPP and SLIP networking protocols are ways of encoding a networking connection over a terminal device such as a serial line. These protocols are implemented under Linux as drivers that at one end appear to the terminal system as line disciplines and at the other end appear to the networking system as network-device drivers. After one of these line disciplines has been enabled on a terminal device, any data appearing on that terminal will be routed directly to the appropriate network-device driver.

I/O Hardware:-

One of the important jobs of an Operating System is to manage various I/O devices including mouse, keyboards, touch pad, disk drives, display adapters, USB devices, Bit-mapped screen, LED, Analog-to-digital converter, On/off switch, network connections, audio I/O, printers etc.

An I/O system is required to take an application I/O request and send it to the physical device, then take whatever response comes back from the device and send it to the application. I/O devices can be divided into two categories −

Block devices − A block device is one with which the driver communicates by sending entire blocks of data. For example, Hard disks, USB cameras, Disk-On-Key etc.

Character devices − A character device is one with which the driver communicates by sending and receiving single characters (bytes, octets). For example, serial ports, parallel ports, sounds cards etc

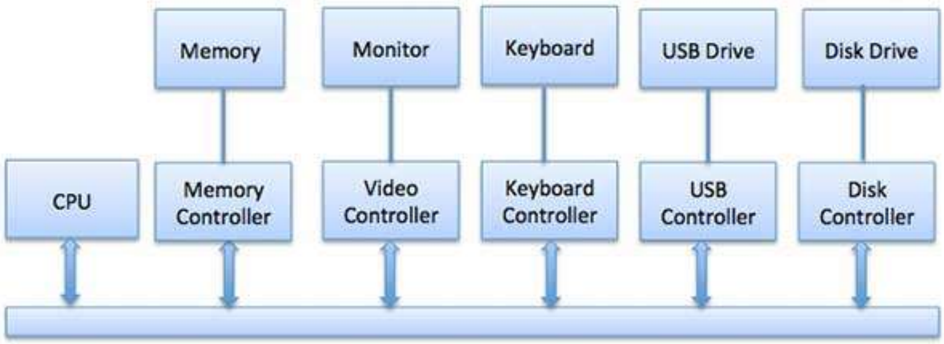
Device Controllers

Device drivers are software modules that can be plugged into an OS to handle a particular device. Operating System takes help from device drivers to handle all I/O devices.

The Device Controller works like an interface between a device and a device driver. I/O units (Keyboard, mouse, printer, etc.) typically consist of a mechanical component and an electronic component where electronic component is called the device controller.

There is always a device controller and a device driver for each device to communicate with the Operating Systems. A device controller may be able to handle multiple devices. As an interface its main task is to convert serial bit stream to block of bytes, perform error correction as necessary.

Any device connected to the computer is connected by a plug and socket, and the socket is connected to a device controller. Following is a model for connecting the CPU, memory, controllers, and I/O devices where CPU and device controllers all use a common bus for communication.



Synchronous vs asynchronous I/O

Synchronous I/O − In this scheme CPU execution waits while I/O proceeds

Asynchronous I/O − I/O proceeds concurrently with CPU execution

Communication to I/O Devices

The CPU must have a way to pass information to and from an I/O device. There are three approaches available to communicate with the CPU and Device.

Special Instruction I/O

Memory-mapped I/O

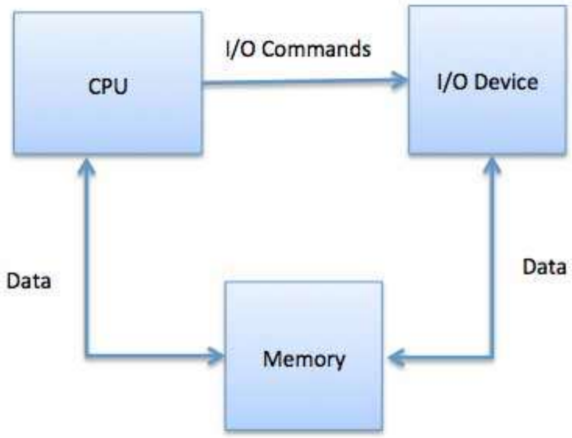
Direct memory access (DMA)

Special Instruction I/O

This uses CPU instructions that are specifically made for controlling I/O devices. These instructions typically allow data to be sent to an I/O device or read from an I/O device.

Memory-mapped I/O

When using memory-mapped I/O, the same address space is shared by memory and I/O devices. The device is connected directly to certain main memory locations so that I/O device can transfer block of data to/from memory without going through CPU.



While using memory mapped IO, OS allocates buffer in memory and informs I/O device to use that buffer to send data to the CPU. I/O device operates asynchronously with CPU, interrupts CPU when finished.

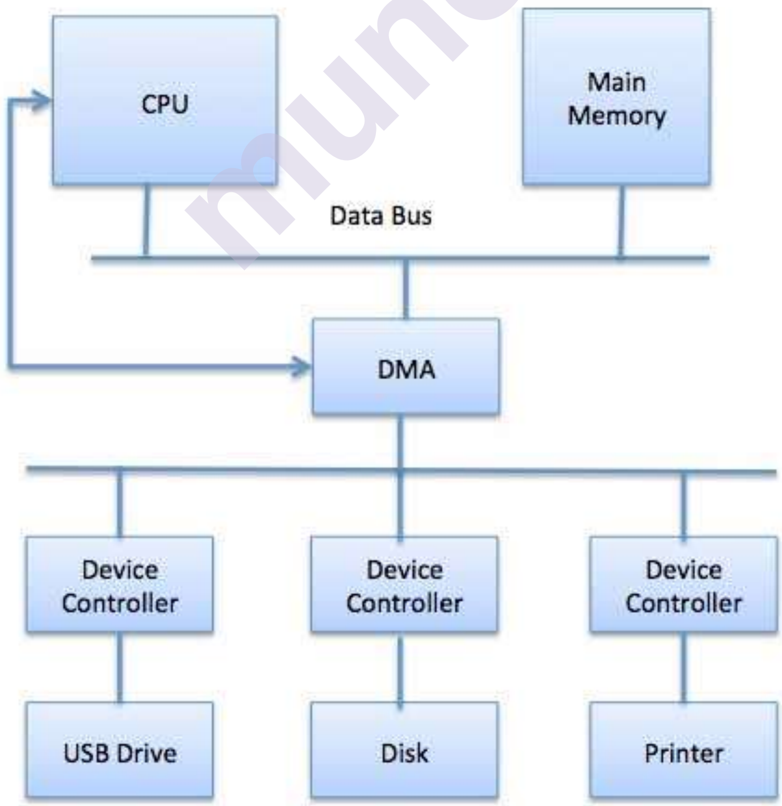
The advantage to this method is that every instruction which can access memory can be used to manipulate an I/O device. Memory mapped IO is used for most high-speed I/O devices like disks, communication interfaces.

Direct Memory Access (DMA)

Slow devices like keyboards will generate an interrupt to the main CPU after each byte is transferred. If a fast device such as a disk generated an interrupt for each byte, the operating system would spend most of its time handling these interrupts. So a typical computer uses direct memory access (DMA) hardware to reduce this overhead.

Direct Memory Access (DMA) means CPU grants I/O module authority to read from or write to memory without involvement. DMA module itself controls exchange of data between main memory and the I/O device. CPU is only involved at the beginning and end of the transfer and interrupted only after entire block has been transferred.

Direct Memory Access needs a special hardware called DMA controller (DMAC) that manages the data transfers and arbitrates access to the system bus. The controllers are programmed with source and destination pointers (where to read/write the data), counters to track the number of transferred bytes, and settings, which includes I/O and memory types, interrupts and states for the CPU cycles.



Polling vs Interrupts I/O

A computer must have a way of detecting the arrival of any type of input. There are two ways that this can happen, known as polling and interrupts. Both of these techniques allow the processor to deal with events that can happen at any time and that are not related to the process it is currently running.

Polling I/O

Polling is the simplest way for an I/O device to communicate with the processor. The process of periodically checking status of the device to see if it is time for the next I/O operation, is called polling. The I/O device simply puts the information in a Status register, and the processor must come and get the information.

Most of the time, devices will not require attention and when one does it will have to wait until it is next interrogated by the polling program. This is an inefficient method and much of the processors time is wasted on unnecessary polls.

Compare this method to a teacher continually asking every student in a class, one after another, if they need help. Obviously the more efficient method would be for a student to inform the teacher whenever they require assistance.

Interrupts I/O

An alternative scheme for dealing with I/O is the interrupt-driven method. An interrupt is a signal to the microprocessor from a device that requires attention.

A device controller puts an interrupt signal on the bus when it needs CPU's attention when CPU receives an interrupt, It saves its current state and invokes the appropriate interrupt handler using the interrupt vector (addresses of OS routines to handle various events). When the interrupting device has been dealt with, the CPU continues with its original task as if it had never been interrupted.

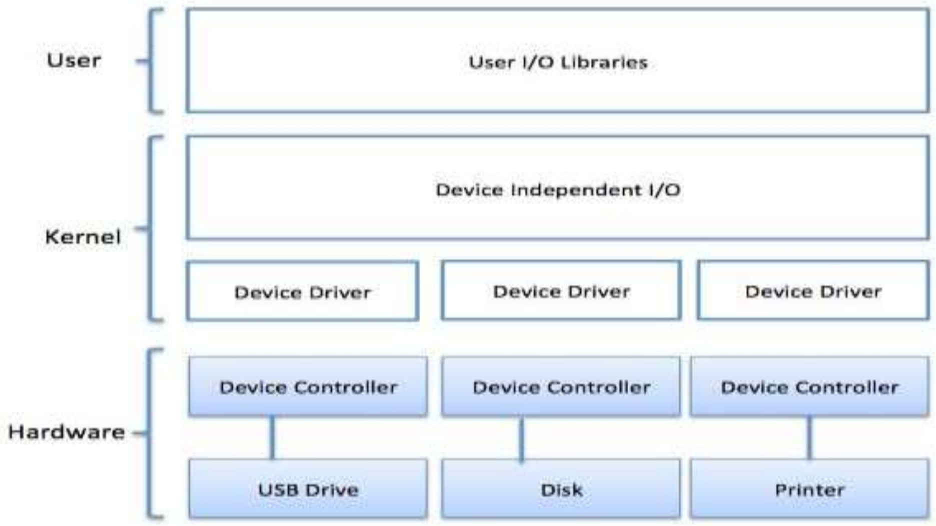
I/O software is often organized in the following layers −

User Level Libraries − This provides simple interface to the user program to perform input and output. For example, stdio is a library provided by C and C++ programming languages.

Kernel Level Modules − This provides device driver to interact with the device controller and device independent I/O modules used by the device drivers.

Hardware − This layer includes actual hardware and hardware controller which interact with the device drivers and makes hardware alive.

A key concept in the design of I/O software is that it should be device independent where it should be possible to write programs that can access any I/O device without having to specify the device in advance. For example, a program that reads a file as input should be able to read a file on a floppy disk, on a hard disk, or on a CD-ROM, without having to modify the program for each different device.



Device Drivers

Device drivers are software modules that can be plugged into an OS to handle a particular device. Operating System takes help from device drivers to handle all I/O devices. Device drivers encapsulate device-dependent code and implement a standard interface in such a way that code contains device-specific register reads/writes. Device driver, is generally written by the device's manufacturer and delivered along with the device on a CD-ROM.

A device driver performs the following jobs −

To accept request from the device independent software above to it.

Interact with the device controller to take and give I/O and perform required error handling

Making sure that the request is executed successfully

How a device driver handles a request is as follows: Suppose a request comes to read a block N. If the driver is idle at the time a request arrives, it starts carrying out the request immediately. Otherwise, if the driver is already busy with some other request, it places the new request in the queue of pending requests.

Interrupt handlers

An interrupt handler, also known as an interrupt service routine or ISR, is a piece of software or more specifically a callback function in an operating system or more specifically in a device driver, whose execution is triggered by the reception of an interrupt.

When the interrupt happens, the interrupt procedure does whatever it has to in order to handle the interrupt, updates data structures and wakes up process that was waiting for an interrupt to happen.

The interrupt mechanism accepts an address ─ a number that selects a specific interrupt handling routine/function from a small set. In most architectures, this address is an offset stored in a table called the interrupt vector table. This vector contains the memory addresses of specialized interrupt handlers.

Device-Independent I/O Software

The basic function of the device-independent software is to perform the I/O functions that are common to all devices and to provide a uniform interface to the user-level software. Though it is difficult to write completely device independent software but we can write some modules which are common among all the devices. Following is a list of functions of device-independent I/O Software −

Kernel I/O Subsystem

Kernel I/O Subsystem is responsible to provide many services related to I/O. Following are some of the services provided.

Scheduling − Kernel schedules a set of I/O requests to determine a good order in which to execute them. When an application issues a blocking I/O system call, the request is placed on the queue for that device. The Kernel I/O scheduler rearranges the order of the queue to improve the overall system efficiency and the average response time experienced by the applications.

Buffering − Kernel I/O Subsystem maintains a memory area known as buffer that stores data while they are transferred between two devices or between a device with an application operation. Buffering is done to cope with a speed mismatch between the producer and consumer of a data stream or to adapt between devices that have different data transfer sizes.

Caching − Kernel maintains cache memory which is region of fast memory that holds copies of data. Access to the cached copy is more efficient than access to the original.

Spooling and Device Reservation − A spool is a buffer that holds output for a device, such as a printer, that cannot accept interleaved data streams. The spooling system copies the queued spool files to the printer one at a time. In some operating systems, spooling is managed by a system daemon process. In other operating systems, it is handled by an in-kernel thread.

Error Handling − An operating system that uses protected memory can guard against many kinds of hardware and application errors.

3.3 FILE, DIRECTORIES AND IMPLEMENTATION: -

A file is a named collection of related information that is recorded on secondary storage such as magnetic disks, magnetic tapes and optical disks. In general, a file is a sequence of bits, bytes, lines or records whose meaning is defined by the files creator and user.

File Structure

A File Structure should be according to a required format that the operating system can understand.

A file has a certain defined structure according to its type.

A text file is a sequence of characters organized into lines.

A source file is a sequence of procedures and functions.

An object file is a sequence of bytes organized into blocks that are understandable by the machine.

When operating system defines different file structures, it also contains the code to support these file structure. Unix, MS-DOS support minimum number of file structure.

File Type

File type refers to the ability of the operating system to distinguish different types of file such as text files source files and binary files etc. Many operating systems support many types of files. Operating system like MS-DOS and UNIX have the following types of files −

Ordinary files

These are the files that contain user information. These may have text, databases or executable program. The user can apply various operations on such files like add, modify, delete or even remove the entire file.

Directory files

These files contain list of file names and other information related to these files.

Special files

These files are also known as device files. These files represent physical device like disks, terminals, printers, networks, tape drive etc.

These files are of two types −

Character special files − data is handled character by character as in case of terminals or printers.

Block special files − data is handled in blocks as in the case of disks and tapes.

File Access Mechanisms

File access mechanism refers to the manner in which the records of a file may be accessed. There are several ways to access files −

- Sequential access
- Direct/Random access
- Indexed sequential access

Sequential access

A sequential access is that in which the records are accessed in some sequence, i.e., the information in the file is processed in order, one record after the other. This access method is the most primitive one. Example: Compilers usually access files in this fashion.

Direct/Random access

Random access file organization provides, accessing the records directly. Each record has its own address on the file with by the help of which it can be directly accessed for reading or writing. The records need not be in any sequence within the file and they need not be in adjacent locations on the storage medium.

Indexed sequential access

This mechanism is built up on base of sequential access. An index is created for each file which contains pointers to various blocks. Index is searched sequentially and its pointer is used to access the file directly.

Space Allocation

Files are allocated disk spaces by operating system. Operating systems deploy following three main ways to allocate disk space to files.

Contiguous Allocation

Linked Allocation

Indexed Allocation

Contiguous Allocation

Each file occupies a contiguous address space on disk.

Assigned disk address is in linear order.

Easy to implement.

External fragmentation is a major issue with this type of allocation technique.

Linked Allocation

Each file carries a list of links to disk blocks.

Directory contains link / pointer to first block of a file.

No external fragmentation

Effectively used in sequential access file.

Inefficient in case of direct access file.

Indexed Allocation

Provides solutions to problems of contiguous and linked allocation.

A index block is created having all pointers to files.

Each file has its own index block which stores the addresses of disk space occupied by the file.

Directory contains the addresses of index blocks of files.

3.4 SECURITY

Security refers to providing a protection system to computer system resources such as CPU, memory, disk, software programs and most importantly data/information stored in the computer system. If a computer program is run by an unauthorized user, then he/she may cause severe damage to computer or data stored in it. So a computer system must be protected against unauthorized access, malicious access to system memory, viruses, worms etc.

Authentication

Authentication refers to identifying each user of the system and associating the executing programs with those users. It is the responsibility of the Operating System to create a protection system which ensures that a user who is running a particular program is authentic. Operating Systems generally identifies/authenticates users using following three ways −

Username / Password − User need to enter a registered username and password with Operating system to login into the system.

User card/key − User need to punch card in card slot, or enter key generated by key generator in option provided by operating system to login into the system.

User attribute - fingerprint/ eye retina pattern/ signature − User need to pass his/her attribute via designated input device used by operating system to login into the system.

One Time passwords

One-time passwords provide additional security along with normal authentication. In One-Time Password system, a unique password is required every time user tries to login into the system. Once a one-time password is used, then it cannot be used again. One-time password are implemented in various ways.

Random numbers − Users are provided cards having numbers printed along with corresponding alphabets. System asks for numbers corresponding to few alphabets randomly chosen.

Secret key − User are provided a hardware device which can create a secret id mapped with user id. System asks for such secret id which is to be generated every time prior to login.

Network password − Some commercial applications send one-time passwords to user on registered mobile/ email which is required to be entered prior to login.

Program Threats

Operating system's processes and kernel do the designated task as instructed. If a user program made these process do malicious tasks, then it is known as Program Threats. One of the common example of program threat is a program installed in a computer which can store and send user credentials via network to some hacker. Following is the list of some well-known program threats.

Trojan Horse − Such program traps user login credentials and stores them to send to malicious user who can later on login to computer and can access system resources.

Trap Door − If a program which is designed to work as required, have a security hole in its code and perform illegal action without knowledge of user then it is called to have a trap door.

Logic Bomb − Logic bomb is a situation when a program misbehaves only when certain conditions met otherwise it works as a genuine program. It is harder to detect.

Virus − Virus as name suggest can replicate themselves on computer system. They are highly dangerous and can modify/delete user files, crash systems. A virus is generatlly a small code embedded in a program. As user accesses the program, the virus starts getting embedded in other files/ programs and can make system unusable for user

System Threats

System threats refers to misuse of system services and network connections to put user in trouble. System threats can be used to launch program threats on a complete network called as program attack. System threats creates such an environment that operating system resources/ user files are misused. Following is the list of some well-known system threats.

Worm − Worm is a process which can choked down a system performance by using system resources to extreme levels. A Worm process generates its multiple copies where each copy uses system resources, prevents all other processes to get required resources. Worms processes can even shut down an entire network.

Port Scanning − Port scanning is a mechanism or means by which a hacker can detects system vulnerabilities to make an attack on the system.

Denial of Service − Denial of service attacks normally prevents user to make legitimate use of the system. For example, a user may not be able to use internet if denial of service attacks browser's content settings.

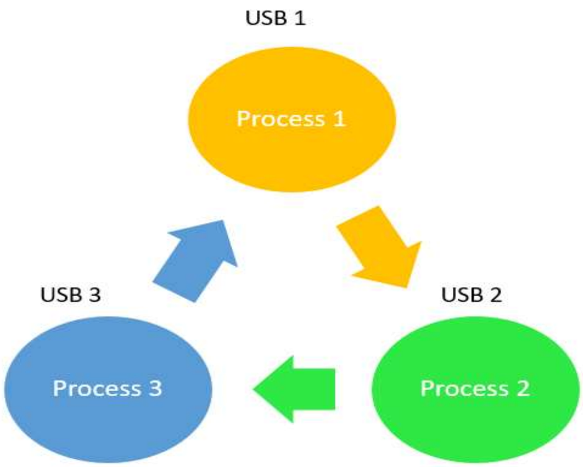
What is Deadlock?

Deadlock is a situation that occurs in OS when any process enters a waiting state because another waiting process is holding the demanded resource. Deadlock is a common problem in multi-processing where several processes share a specific type of mutually exclusive resource known as a soft lock or software

What is Circular wait?

One process is waiting for the resource, which is held by the second process, which is also waiting for the resource held by the third process etc. This will continue until the last process is waiting for a resource held by the first process. This creates a circular chain.

For example, Process A is allocated Resource B as it is requesting Resource A. In the same way, Process B is allocated Resource A, and it is requesting Resource B. This creates a circular wait loop.



Deadlock Detection

A deadlock occurrence can be detected by the resource scheduler. A resource scheduler helps OS to keep track of all the resources which are allocated to different processes. So, when a deadlock is detected, it can be resolved using the below-given methods:

Deadlock Prevention:

It's important to prevent a deadlock before it can occur. The system checks every transaction before it is executed to make sure it doesn't lead the deadlock situations. Such that even a small change to occur dead that an operation which can lead to Deadlock in the future it also never allowed process to execute.

It is a set of methods for ensuring that at least one of the conditions cannot hold.

No preemptive action:

No Preemption – A resource can be released only voluntarily by the process holding it after that process has finished its task If a process which is holding some resources request another resource that can't be immediately allocated to it, in that situation, all resources will be released. Preempted resources require the list of resources for a process that is waiting. The process will be restarted only if it can regain its old resource and a new one that it is requesting. If the process is requesting some other resource, when it is available, then it was given to the requesting process. If it is held by another process that is waiting for another resource, we release it and give it to the requesting process.

Mutual Exclusion:

Mutual Exclusion is a full form of Mutex. It is a special type of binary semaphore which used for controlling access to the shared resource. It includes a priority inheritance mechanism to avoid extended priority inversion problems. It allows current higher priority tasks to be kept in the blocked state for the shortest time possible. Resources shared such as read-only files never lead to deadlocks, but resources, like printers and tape drives, needs exclusive access by a single process.

Hold and Wait:

In this condition, processes must be stopped from holding single or multiple resources while simultaneously waiting for one or more others.

Circular Wait:

It imposes a total ordering of all resource types. Circular wait also requires that every process request resources in increasing order of enumeration.

Deadlock Avoidance

It is better to avoid a deadlock instead of taking action after the Deadlock has occurred. It needs additional information, like how resources should be used. Deadlock avoidance is the simplest and most useful model that each process declares the maximum number of resources of each type that it may need.

Avoidance Algorithms

The deadlock-avoidance algorithm helps you to dynamically assess the resource-allocation state so that there can never be a circular-wait situation.

A single instance of a resource type.

Use a resource-allocation graph

Cycles are necessary which are sufficient for Deadlock

Multiples instances of a resource type.

Cycles are necessary but never sufficient for Deadlock.

Uses the banker's algorithm

Advantages of Deadlock

Here, are pros/benefits of using Deadlock method

This situation works well for processes which perform a single burst of activity

No preemption needed for Deadlock.

Convenient method when applied to resources whose state can be saved and restored easily

Feasible to enforce via compile-time checks

Needs no run-time computation since the problem is solved in system design

Disadvantages of Deadlock method

Here, are cons/ drawback of using deadlock method

Disks

The ideal storage device is

1. Fast
2. Big (in capacity)
3. Cheap
4. Impossible

Disks are big and cheap, but slow.

Disk Hardware

Show a real disk opened up and illustrate the components

- Platter
- Surface
- Head
- Track
- Sector
- Cylinder
- Seek time
- Rotational latency
- Transfer rate

Overlapping I/O operations is important. Many controllers can do overlapped seeks, i.e. issue a seek to one disk while another is already seeking.

Modern disks cheat and do not have the same number of sectors on outer cylinders as on inner one. However, the disks have electronics and software (firmware) that hides the cheat and gives the illusion of the same number of sectors on all cylinders.

(Unofficial) Despite what tanenbaum says later, it is not true that when one head is reading from cylinder C, all the heads can read from cylinder C with no penalty. It is, however, true that the penalty is very small.

Choice of block size

- We discussed this before when studying page size.
- Current commodity disk characteristics (not for laptops) result in about 15ms to transfer the first byte and 10K bytes per ms for subsequent bytes (if contiguous).
    - Rotation rate is 5400, 7600, or 10,000 RPM (15K just now available).
    - Recall that 6000 RPM is 100 rev/sec or one rev per 10ms. So half a rev (the average time for to rotate to a given point) is 5ms.
    - Transfer rates around 10MB/sec = 10KB/ms.
    - Seek time around 10ms.
- This favors large blocks, 100KB or more.
- But the internal fragmentation would be severe since many files are small.
- Multiple block sizes have been tried as have techniques to try to have consecutive blocks of a given file near each other.
- Typical block sizes are 4KB-8KB.

RAID (Redundant Array of Inexpensive Disks) (Skipped)

- The name RAID is from Berkeley.
- IBM changed the name to Redundant Array of Independent Disks. I wonder why?
- A simple form is mirroring, where two disks contain the same data.
- Another simple form is striping (interleaving) where consecutive blocks are spread across multiple disks. This helps bandwidth, but is not redundant. Thus it shouldn't be called RAID, but it sometimes is.
- One of the normal RAID methods is to have N (say 4) data disks and one parity disk. Data is striped across the data disks and the bitwise parity of these sectors is written in the corresponding sector of the parity disk.
- On a read if the block is bad (e.g., if the entire disk is bad or even missing), the system automatically reads the other blocks in the stripe and the parity block in the stripe. Then the missing block is just the bitwise exclusive or of all these blocks.

- For reads this is very good. The failure free case has no penalty (beyond the space overhead of the parity disk). The error case requires N+1 (say 5) reads.
- A serious concern is the small write problem. Writing a sector requires 4 I/O. Read the old data sector, compute the change, read the parity, compute the new parity, write the new parity and the new data sector. Hence one sector I/O became 4, which is a 300% penalty.
- Writing a full stripe is not bad. Compute the parity of the N (say 4) data sectors to be written and then write the data sectors and the parity sector. Thus 4 sector I/Os become 5, which is only a 25% penalty and is smaller for larger N, i.e., larger stripes.
- A variation is to rotate the parity. That is, for some stripes disk 1 has the parity, for others disk 2, etc. The purpose is to not have a single parity disk since that disk is needed for all small writes and could become a point of contention.

Disk Arm Scheduling Algorithms

There are three components to disk response time: seek, rotational latency, and transfer time. Disk arm scheduling is concerned with minimizing seek time by reordering the requests.

These algorithms are relevant only if there are several I/O requests pending. For many PCs this is not the case. For most commercial applications, I/O is crucial and there are often many requests pending.

1. FCFS (First Come First Served): Simple but has long delays.
2. Pick: Same as FCFS but pick up requests for cylinders that are passed on the way to the next FCFS request.
3. SSTF or SSF (Shortest Seek (Time) First): Greedy algorithm. Can starve requests for outer cylinders and almost always favors middle requests.
4. Scan (Look, Elevator): The method used by an old fashioned jukebox (remember ``Happy Days") and by elevators. The disk arm proceeds in one direction picking up all requests until there are no more requests in this direction at which point it goes back the other direction. This favors requests in the middle, but can't starve any requests.
5. C-Scan (C-look, Circular Scan/Look): Similar to Scan but only service requests when moving in one direction. When going in the other direction, go directly to the furthest away request. This doesn't favor any spot on the disk. Indeed, it treats the cylinders as though they were a clock, i.e. after the highest numbered cylinder comes cylinder 0.
6. N-step Scan: This is what the natural implementation of Scan gives.
    - While the disk is servicing a Scan direction, the controller gathers up new requests and sorts them.

- At the end of the current sweep, the new list becomes the next sweep.

Minimizing Rotational Latency

Use Scan based on sector numbers not cylinder number. For rotational latency Scan which is the same as C-Scan. Why?

Ans: Because the disk only rotates in one direction.

Security Mechanism

OS security mechanisms:

Memory Protection:

One of the important aspects of Operating system security is Memory Protection. Memory provides powerful indirect way for an attacker to circumvent security mechanism, since every piece of information accessed by any program will need to reside in memory at some point in time, and hence may potentially be accessed in the absence of memory protection mechanisms.

Memory protection is a way for controlling memory usage on a computer, and is core to virtually every operating system. The main purpose of memory protection is to prevent a process running on an operating system from accessing the memory of other processes, or is used by the OS kernel. This prevents a bug within the process from affecting other processes, and also prevents malicious software from gaining unauthorized access to the system, e.g., suppose that process A is permitted access to a file F, while process B is not. Process B can bypass this policy by attempting to read F's content that will be stored in A's memory immediately after A reads F. Alternatively, B may attempt to modify the access control policy that is stored in the OS memory so that the OS thinks that B is permitted access to this file.

How to protect memory of one process from another?

The virtual memory mechanism supported on most OSes ensures that the memory of different processes are logically disjoint. The virtual addresses, which are logical addresses, are transformed into a physical memory address using address translation hardware. To speed up translation, various caching mechanisms are utilized. First, most L1 processor caches are based on virtual addresses, so cache accesses don't need address translation. Next, the paging hardware uses cache-like mechanisms (TLBs) to avoid performing bounds checks on every virtual access.

In order to secure the virtual address translation mechanism, it is important to ensure that processes cannot tamper with the address translation mechanisms. To ensure this, processors have to provide some protection primitives. Typically, this is done using the notion of privileged execution modes.

Specifically, 2 modes of CPU execution are introduced: privileged and unprivileged. (Processors may support multiple levels of privileges, but today's OSes use only two levels.) Certain instructions, such as those relating to I/O, DMA, interrupt processing, and page table translation are permitted only in the privileged mode.

OSes rely on the protection mechanism provided by the processor as follows. All user processes (including root-processes) execute in unprivileged mode, while the OS kernel executes in privileged mode. Obviously, user level processes need to access OS kernel functionality from time time to time. Typically, this is done using system calls that represent a call from unprivileged code to privileged code. Uncontrolled calls across the privilege boundary can defeat security mechanism, e.g., it should not be possible for arbitrary user code to call a kernel function that changes the page tables. For this reason, privilege transitions need to be carefully controlled. Usually, "software trap" instructions are used to effect transition from low to high privilege mode. (Naturally, no protection is needed for transitioning from privileged to unprivileged mode.) On Linux, software interrupt 0x80 is used for this purpose. When this instruction is invoked, the processor starts executing the interrupt handler code for this interrupt in the privileged mode. (Note that the changes to interrupt handler should itself be permitted only in the privileged mode, or else this mechanism could be subverted.) This code should perform appropriate checks to ensure that the call is legitimate, and then carry it out. This basically means that the parameters to system calls have to be thoroughly checked.

UNIX Processes and Security:

Processes have different type of ID's. These Ids are inherited by a child process from its parent, except in the case of setuid processes – in their case, their effective userid is set to be the same as the owner of the file that is executed.

1. User ID:
   a) Effective User ID (EUID)
      Effective user id is used for all permission checking operations.
   b) Real User ID (RUID)
      Real user ID is the one that represents the "real user" that launched the process.
   c) Saved User ID (SUID)
      Saved userid stores the value of userid before a setuid operation.

A privileged process (ie process with euid of 0) can change these 3 uids to arbitrary values, while unprivileged processes can change them to one of the values of these 3 uids. This constraint on unprivileged processes prevents them for assuming the userid of arbitrary users, but allows limited changes. For instance, an FTP server initially starts off with euid = ruid = 0. When a user U logs in, the euid and ruid are set to U, but the saved uid remains as root. This allows the FTP server to later change its euid to 0 for the purpose of binding to a low-numbered port. (The original FTP protocol requires this binding for each data connection.)

2. Group ID:

Group identifier (GID) is used to represent a specific group. As single user can belongs to multiple groups, a single process can have multiple group ids. These are organized as a “primary” group id, and a list of supplementary gids. The primary gid has 3 flavors (real, effective and save), analogous to uids. All objects created by a process will have the effective gid of the process. Supplementary gids are used only for permission checking.

3. Group Passwords :

If a user is not listed as belonging to a group G, and there is a password for G, this user can change her group by providing this group password.

Inter-processes communication:

A process can influence the behavior of another process by communicating with it. From a security point of view, this is not an issue if the two processes belong to the same user. (Any damage that can be effected by the second process can be effected by the first process as well, so there is no incentive for the first process to attack the second --- this is true on standard UNIX systems, where application-specific access control policies (say, DTE) aren't used.) If not, we need to be careful. We need to pay particular attention to situations where an unprivileged process communicates with a privileged process in ways that the privileged process did not expect.

1. Parent to child communication

If the parent has a child with higher privilege- e.g. the child is a setuid program, then certain mechanism is needed to prevent the child taking advantage of the setuid program. In particular, such a child program should expect to receive parameters from an unprivileged process and validate them. But it may not expect subversion attacks, for example, the parent may modify the path (specified in an environment variable) for searching for libraries. To prevent this, the loader typically ignores these path specifications for seuid processes. Parents are still permitted to send signals to children, even if their uids are different.

2. Signals are a mechanism for the OS to notify user-level processes about exceptions, e.g.,invalid memory access. Their semantics is similar to that of interrupts ---- processes typically install a “signal handler,” which can be different for different signals. (UNIX defines about 30 such signals.) When a signal occurs, process execution in interrupted, and control transferred to the handler for that signal. Once the handler finishes execution, the execution of application code resumes at the point where it was interrupted.

Signals can also be used for communication: one process can send a signal to another process using the “kill” system call. Due to security considerations, this is permitted only when the userid of the process sending the signal is zero, or equals that of the receiving process.

3. Debugging and Tracing

OSes need to provide some mechanisms for debugging. On Linux, this takes the form of the ptrace mechanism. It allows a debugger to read or write arbitrary locations in the memory of a debugged process. It can also read or set the values of registers used by the debugged process. The interface allows code regions to be written as well --- typically, code regions are protected and hence the debugged process won't be able to overwrite code without first using a system call to modify the permissions on the memory page(s) that contains the code. But the debugger is able to change code as well.

Obviously, the debugging interface enables a debugging process to exert a great deal of control over the debugged process. As such, UNIX allows debugging only if the debugger and the debugged processes are both run by the same user.

(On Linux, ptrace can also be used for system call interception. In this case, every time the debugged process makes a system call, the call is suspended inside the kernel, and the information delivered to the debugger. At this point, the debugger can change this system call or its parameters, and then allow the system call to continue. When the system call is completed, the debugger is notified again, and it can change the return results or modify the debugged process memory. It can then let the system call return to the debugged process, which then resumes execution.)

4. Network Connection

a) Binding: Programs use the socket abstraction for network communication. In order for a socket, which represents a communication endpoint, to become visible from outside, it needs to be associated with a port number. (This holds for TCP and UDP, the two main protocols used for communication on the Internet.) Historically, ports below 1024 were considered “privileged” ports --- binding to them required root privileges. The justification is in the context of large, multi-user systems where a number of user applications are running on the same system as a bunch of services. The assumption was that user processes are not trusted, and could try to masquerade as a server. (For instance, a user process masquerading as a telnet server could capture passwords of other users and forward them to the attacker.) To prevent this possibility, trusted servers would use only ports below 1024. Since such ports cannot be bound to normal user processes, this masquerading wont be possible.

b) Connect:

   A client initiates a connection. There are no access controls associated with the connect operation on most contemporary OSes.

c) Accept :

   Accept is used by a server to accept an incoming connection (i.e., in response to a connect operation invoked by a client). No permission checks are associated with this operation on most contemporary OSes.

Boot Security:

A number of security-critical services get started up at boot time. It is necessary to understand this sequence in order to identify the relevant security issues.

1) Loader loads the Kernel

Loader loads the kernel and init process starts. The PID of init process is 0.

2) Kernel modules get loaded and devices are initialized

Some kernel modules are loaded immediately; others are loaded explicitly by boot scripts.

3) Boot scripts are stored at /etc/init.d

4) Run Levels

0 halt

1 single user

2 Full Multi-User mode (default)

3-5 Same as 2

6 Reboot

Scripts that will be run at different run levels can be different. To support this, UNIX systems typically use one directory per run level (named /etc/rcN.d/) for storing these scripts.

These directories contain symbolic links to the actual files stored in /etc/init.d. Script names that start with "S" are run at startup, while those starting with "K" are run at shutdown time. The order of running the scripts is determined by its name --- for instance, S01 will be run before S02 and so on.

Other UNIX security issues

1) Devices

a) Hard disk

b) /dev/mem & /dev/kmem : (virtual memory and kernel memory)

c) /dev/tty

Access to raw devices must be carefully controlled, or else it can defeat higher level security primitives. For instance, by directly accessing the contents of a hard drive, a process can modify any thing on the file system, thereby bypassing any permissions set on the files stored therein. Similarly, one process can interfere with the network packets that belong to another user's process by directly reading (or writing to) the network interface. Finally, memory can be accessed indirectly through low-level devices. In UNIX, all these devices are typically configured so that only root processes can access them.

2) Mounting File Systems

When we want to attach a file system to an operating system we need to specify where in a directory structure we want to attach it, this process is called mounting. This ability to mount raises several security issues.

(a) For removable media (USB drives, CDROMs, etc), an ordinary user may create a setuid-toroot executable on a different system (on which she has root access). My mounting this file system on a machine on which she has no root access, she can obtain root privileges by running the suid application. So. one should be careful about granting mount privileges to ordinary users. One common approach is to grant these privileges while disabling setuid option for filesystems mounted by ordinary users.

(b) UNIX allows the same file system to be mounted at more than one mount point. When this is done, the user has effectively created aliases for file names. For instance, if a filesystem is mounted on /usr and /mnt/usr, then a file A in this filesystem can be accessed using the name /usr/A and /mnt/usr/A.

3) **Search Path**

A search path is a sequence of directories that a system uses to locate an object (program, library, or file). Because programs rely on search paths, users must take care to set them appropriately.

Some systems have many types of search paths. In addition to searching for executables, a common search path contains directories that are used to search for libraries when the system supports dynamic loading. If an attacker is able to influence this search path, then he induce other users (including root) to execute code of his choice. For instance, suppose that an attacker A can modify root's path to include /home/A at its beginning. Then, when root types the command ls, the file /home/A/ls may get executed with root privileges. Since the attacker created this file, it gives the attacker the ability to run arbitrary code with root privileges.

4) **Capabilities.** Modern UNIX systems have introduced some flexibility in places were policies were hard-coded previously. For instance, the ability to change file ownerships is now treated as a capability within Linux. (These are not fully transferable, in the sense of classical capabilities, but they can inherited across a fork.) A number of similar capabilities have been defined. (On Linux, try "man capabilities.")

5) **Network Access**

Linux systems provide a built-in firewalling capabilities. This is administered using the iptables program. When this service is enabled, iptables related scripts are run at boottime. You can figure out how to configure this by looking at the relevant scrips and the documentation on iptables configuration. In addition to iptables, additional mechanisms are available for controlled network access. The most important ones among these are the hosts.allow and hosts.deny files that specify which hosts are allowed to connect to the local system.

Database security

Main issue in database security is fine granularity – it is not enough to permit or deny access to an entire database. SQL supports an access control mechanism that can be used to limit access to tables in much the same way that access can be limited to specific files using a conventional access control specification, e.g., a user may be permitted to read a table, another may be permitted to update it, and so on.

Sometimes, we want to have finer granularity of protection, e.g., suppressing certain columns and/or rows. This can be achieved using database views. Views are a mechanism in databases to provide a customized view of a database to particular users. Typically, a view can be defined as the result of a database query. As a result, rows can be omitted, or columns can be projected out using a query. Thus, by combining views with SQL access control primitives, we can realize fairly sophisticated access control objectives.

Statistical security and the inference problem

When dealing with sensitive data, it is often necessary permit access to aggregated even when access to individual data items may be too sensitive to reveal. For instance, the census bureau collects a lot of sensitive information about individuals. In general, no one should be able to access detailed individual records. However, if we dont permit aggregate queries, e.g., the number of people in a state that are african americans, then the whole purpose of conducting census would be lost. The catch is that it may be possible to identify sensitive information from the results of one of more aggregate queries. This is called the inference problem. As an example, consider a database that contains grade information for this course. We may permit aggregate queries, e.g., average score on the final exam. But if this average is computed over a small set, then it can reveal sensitive information. To illustrate this, consider a class that has only a single woman. By making a query that selects the students whose gender is female, and asking for the average of these students, one can determine the grade of a single individual in the class.

One can attempt to solve this problem by prescribing a minimum size on the sets on which aggregates are computed. But an attacker can circumvent this by computing aggregates on the complement of a set, e.g., by comparing the average of the whole class with the average for male students, an attacker can compute the female student's grade. Another possibility is to insert random errors in outputs. For instance, in the above calculation, a small error in the average grades can greatly increase the error in the inferred grade of the female student.

3.5 SUMMARY

This chapter mainly focuses on Memory management in Operating System.

3.6 EXERCISE

Homework: Consider a disk with an average seek time of 10ms, an average rotational latency of 5ms, and a transfer rate of 10MB/sec.

1. If the block size is 1KB, how long would it take to read a block?
2. If the block size is 100KB, how long would it take to read a block?

3. If the goal is to read 1K, a 1KB block size is better as the remaining 99KB are wasted. If the goal is to read 100KB, the 100KB block size is better since the 1KB block size needs 100 seeks and 100 rotational latencies. What is the minimum size request for which a disk with a 100KB block size would complete faster than one with a 1KB block size?

3.7 REFERENCES

- An Introduction to Operating Systems: Concepts and Practice (GNU/Linux), 4th19 edition, Pramod Chandra P. Bhatt, Prentice-Hall of India Pvt. Ltd, 2014.
- Operating System Concepts with Java Eight Edition, Avi Silberschatz, Peter Baer Galvin, Greg Gagne, John Wiley & Sons, Inc., 2009, http://codex.cs.yale.edu/avi/os-book/OS8/os8j
- UNIX and Linux System Administration Handbook, Fourth Edition, Evi Nemeth, Garth Snyder, Tren Hein, Ben Whaley, Pearson Education, Inc, 2011,

❄❄❄❄❄❄❄

4

ANDROID OPERATING SYSTEM

Unit Structure

4.0 INTRODUCTION

This chapter provides information about Android Operating System and Implementation

4.1 THE ANDROID SOFTWARE STACK

Understanding the Android Software Stack

The Android software stack is, put simply, a Linux kernel and a collection of C/C++ libraries exposed through an application framework that provides services for, and management of, the run time and applications. The Android software stack is composed of the elements shown in Figure 1-1.

- **Linux kernel -** Core services (including hardware drivers, process and memory management, security, network, and power management) are handled by a Linux 2.6 kernel. The kernel also provides an abstraction layer between the hardware and the remainder of the stack.
- **Libraries -** Running on top of the kernel, Android includes various C/C++ core libraries such as libc and SSL, as well as the following:
    - A media library for playback of audio and video media
    - A surface manager to provide display management
    - Graphics libraries that include SGL and OpenGL for 2D and 3D graphics
    - SQLite for native database support
    - SSL and WebKit for integrated web browser and Internet security
- **Android run time -** The run time is what makes an Android phone an Android phone rather than a mobile Linux implementation.

Including the core libraries and the Dalvik VM, the Android run time is the engine that powers your applications and, along with the libraries, forms the basis for the application framework.

- **Core libraries -** Although most Android application development is written using the Java language, Dalvik is not a Java VM. The core Android libraries provide most of the functionality available in the core Java libraries, as well as the Androidspecific libraries.
- **Dalvik VM -** Dalvik is a register-based Virtual Machine that's been optimized to ensure that a device can run multiple instances efficiently. It relies on the Linux kernel for threading and low-level memory management.
- **Application framework -** The application framework provides the classes used to create Android applications. It also provides a generic abstraction for hardware access and manages the user interface and application resources.
- **Application layer -** All applications, both native and third-party, are built on the application layer by means of the same API libraries. The application layer runs within the Android run time, using the classes and services made available from the application framework.

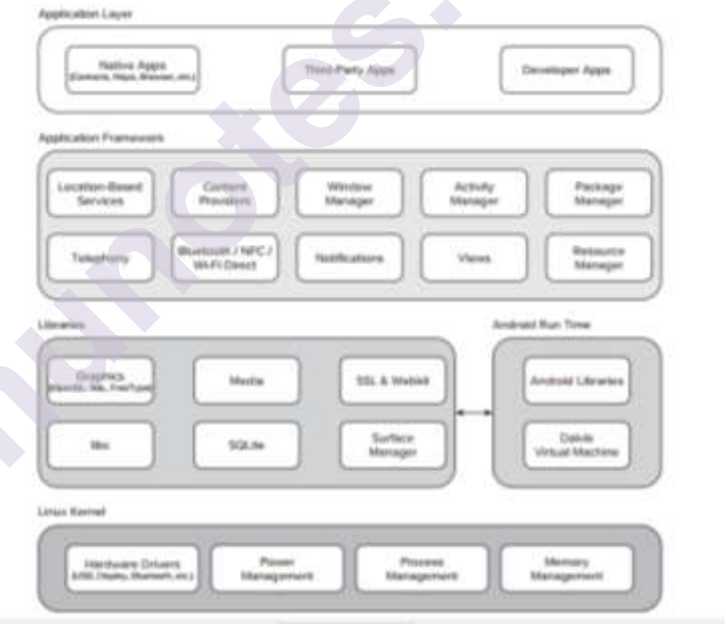


Figure 4.1 Android Software Stack

4.2 THE LINUX KERNEL

The Linux Kernel – its functions

The kernel has 4 jobs:

- Memory management: Keep track of how much memory is used to store what, and where
- Process management: Determine which processes can use the central processing unit (CPU), when, and for how long

- Device drivers: Act as mediator/interpreter between the hardware and processes
- System calls and security: Receive requests for service from the processes

The kernel, if implemented properly, is invisible to the user, working in its own little world known as kernel space, where it allocates memory and keeps track of where everything is stored. What the user sees—like web browsers and files—are known as the user space. These applications interact with the kernel through a system call interface (SCI).

Think about it like this. The kernel is a busy personal assistant for a powerful executive (the hardware). It's the assistant's job to relay messages and requests (processes) from employees and the public (users) to the executive, to remember what is stored where (memory), and to determine who has access to the executive at any given time and for how long.

To put the kernel in context, you can think of a Linux machine as having 3 layers:

- The hardware: The physical machine—the bottom or base of the system, made up of memory (RAM) and the processor or central processing unit (CPU), as well as input/output (I/O) devices such as storage, networking, and graphics. The CPU performs computations and reads from, and writes to, memory.
- The Linux kernel: The core of the OS. (See? It's right in the middle.) It's software residing in memory that tells the CPU what to do.
- User processes: These are the running programs that the kernel manages. User processes are what collectively make up user space. User processes are also known as just processes. The kernel also allows these processes and servers to communicate with each other (known as inter-process communication, or IPC).

Code executed by the system runs on CPUs in 1 of 2 modes: kernel mode or user mode. Code running in the kernel mode has unrestricted access to the hardware, while user mode restricts access to the CPU and memory to the SCI. A similar separation exists for memory (kernel space and user space). These 2 small details form the base for some complicated operations like privilege separation for security, building containers, and virtual machines.

This also means that if a process fails in user mode, the damage is limited and can be recovered by the kernel. However, because of its access to memory and the processor, a kernel process crash can crash the entire system. Since there are safeguards in place and permissions required to cross boundaries, user process crashes usually can't cause too many problems.

4.3 LIBRARIES

SQLite Library used for data storage and light in terms of mobile memory footprints and task execution.

WebKit Library mainly provides Web Browsing engine and a lot more related features.

The surface manager library is responsible for rendering windows and drawing surfaces of various apps on the screen.

The media framework library provides media codecs for audio and video.

The OpenGl (Open Graphics Library) and SGL(Scalable Graphics Library) are the graphics libraries for 3D and 2D rendering, respectively.

The FreeType Library is used for rendering fonts.

Application Framework

It is a collection of APIs written in Java, which gives developers access to the complete feature set of Android OS.

Developers have full access to the same framework APIs used by the core applications, so that they can enhance more in terms of functionalities of their application.

Enables and simplify the reuse of core components and services, like:

Activity Manager: Manages the Lifecycle of apps & provide common navigation back stack.

Window Manager: Manages windows and drawing surfaces, and is an abstraction of the surface manager library.

Content Providers: Enables application to access data from other applications or to share their own data i.e it provides mechanism to exchange data among apps.

View System: Contains User Interface building blocks used to build an application's UI, including lists, grids, texts, boxes, buttons,etc. and also performs the event management of UI elements(explained in later tutorials).

Package Manager: Manages various kinds of information related to the application packages that are currently installed on the device.

Telephony Manager: Enables app to use phone capabilities of the device.

Resource Manager: Provides access to non-code resources (localized Strings, bitmaps, Graphics and Layouts).

Location Manager: Deals with location awareness capabilities.

Notification Manager: Enable apps to display custom alerts in the status bar.

Applications

Top of the Android Application Stack, is occupied by the System apps and tonnes of other Apps that users can download from Android's Official Play Store, also known as Google Play Store. A set of Core applications are pre-packed in the handset like Email Client, SMS Program, Calendar, Maps, Browser, Contacts and few more. This layer uses all the layers below it for proper functioning of these mobile apps. So as we can see and understand, Android holds layered or we can say grouped functionalities as software stack that makes Android work very fluently in any device.

Media framework

A multimedia framework is a software framework that handles media on a computer and through a network. ... It is meant to be used by applications such as media players and audio or video editors, but can also be used to build videoconferencing applications, media converters and other multimedia tools.

SQLite

SQLite is a opensource SQL database that stores data to a text file on a device. Android comes in with built in SQLite database implementation. SQLite supports all the relational database features. In order to access this database, you don't need to establish any kind of connections for it like JDBC,ODBC e.t.c

Database - Package

The main package is android.database.sqlite that contains the classes to manage your own databases

Database - Creation

In order to create a database you just need to call this method openOrCreateDatabase with your database name and mode as a parameter. It returns an instance of SQLite database which you have to receive in your own object.Its syntax is given below

SQLiteDatabase mydatabase = openOrCreateDatabase("your database name",MODE_PRIVATE,null);

Apart from this , there are other functions available in the database package , that does this job. They are listed below

Sr.No	Method & Description
1	**openDatabase(String path, SQLiteDatabase. CursorFactory factory, int flags, DatabaseErrorHandler errorHandler)** This method only opens the existing database with the appropriate flag mode. The common flags mode could be OPEN_READWRITE OPEN_READONLY
2	**openDatabase(String path, SQLiteDatabase. CursorFactory factory, int flags)** It is similar to the above method as it also opens the existing database but it does not define any handler to handle the errors of databases
3	**openOrCreateDatabase(String path, SQLiteDatabase. CursorFactory factory)** It not only opens but create the database if it not exists. This method is equivalent to openDatabase method.
4	**openOrCreateDatabase(File file, SQLiteDatabase. CursorFactory factory)** This method is similar to above method but it takes the File object as a path rather then a string. It is equivalent to file.getPath()

Database - Insertion

we can create table or insert data into table using execSQL method defined in SQLiteDatabase class. Its syntax is given below

mydatabase.execSQL("CREATE TABLE IF NOT EXISTS TutorialsPoint(Username VARCHAR,Password VARCHAR);");

mydatabase.execSQL("INSERT INTO TutorialsPoint VALUES('admin','admin');");

This will insert some values into our table in our database. Another method that also does the same job but take some additional parameter is given below

Sr.No	Method & Description
1	**execSQL(String sql, Object[] bindArgs)** This method not only insert data , but also used to update or modify already existing data in database using bind arguments

Database – Fetching

We can retrieve anything from database using an object of the Cursor class. We will call a method of this class called rawQuery and it will return a resultset with the cursor pointing to the table. We can move the cursor forward and retrieve the data.

Cursor resultSet = mydatbase.rawQuery("Select * from TutorialsPoint",null);
resultSet.moveToFirst();
String username = resultSet.getString(0);
String password = resultSet.getString(1);

There are other functions available in the Cursor class that allows us to effectively retrieve the data. That includes

Sr.No	Method & Description
1	**getColumnCount()** This method return the total number of columns of the table.
2	**getColumnIndex(String columnName)** This method returns the index number of a column by specifying the name of the column
3	**getColumnName(int columnIndex)** This method returns the name of the column by specifying the index of the column

4	**getColumnNames()** This method returns the array of all the column names of the table.
5	**getCount()** This method returns the total number of rows in the cursor
6	**getPosition()** This method returns the current position of the cursor in the table
7	**isClosed()** This method returns true if the cursor is closed and return false otherwise

Database - Helper class

For managing all the operations related to the database , an helper class has been given and is called SQLiteOpenHelper. It automatically manages the creation and update of the database. Its syntax is given below

```
public class DBHelper extends SQLiteOpenHelper {
  public DBHelper(){
    super(context,DATABASE_NAME,null,1);
  }
  public void onCreate(SQLiteDatabase db) {}
  public void onUpgrade(SQLiteDatabase database, int oldVersion, int newVersion) {}
}
```

Example

Here is an example demonstrating the use of SQLite Database. It creates a basic contacts applications that allows insertion, deletion and modification of contacts.

To experiment with this example, you need to run this on an actual device on which camera is supported.

Steps	Description
1	You will use Android studio to create an Android application under a package com.example.sairamkrishna.myapplication.
2	Modify src/MainActivity.java file to get references of all the

	XML components and populate the contacts on listView.
3	Create new src/DBHelper.java that will manage the database work
4	Create a new Activity as DisplayContact.java that will display the contact on the screen
5	Modify the res/layout/activity_main to add respective XML components
6	Modify the res/layout/activity_display_contact.xml to add respective XML components
7	Modify the res/values/string.xml to add necessary string components
8	Modify the res/menu/display_contact.xml to add necessary menu components
9	Create a new menu as res/menu/mainmenu.xml to add the insert contact option
10	Run the application and choose a running android device and install the application on it and verify the results.

Following is the content of the modified MainActivity.java.

```
package com.example.sairamkrishna.myapplication;

import android.content.Context;
import android.content.Intent;
import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;

import android.view.KeyEvent;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;

import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ListView;
import java.util.ArrayList;
import java.util.List;

public class MainActivity extends ActionBarActivity {
   public final static String EXTRA_MESSAGE = "MESSAGE";
   private ListView obj;
   DBHelper mydb;
```

```
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    mydb = new DBHelper(this);
    ArrayList array_list = mydb.getAllCotacts();
    ArrayAdapter                                   arrayAdapter=new
ArrayAdapter(this,android.R.layout.simple_list_item_1, array_list);

    obj = (ListView)findViewById(R.id.listView1);
    obj.setAdapter(arrayAdapter);
    obj.setOnItemClickListener(new OnItemClickListener(){
      @Override
      public  void  onItemClick(AdapterView<?>  arg0,  View  arg1,  int
arg2,long arg3) {
        // TODO Auto-generated method stub
        int id_To_Search = arg2 + 1;

       Bundle dataBundle = new Bundle();
        dataBundle.putInt("id", id_To_Search);

        Intent  intent  =  new  Intent  (getApplicationContext(),  Display
Contact.class);

        intent.putExtras(dataBundle);
        startActivity(intent);
      }
    });
  }
  @Override
  public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.menu_main, menu);
    return true;
  }

  @Override
  public boolean onOptionsItemSelected(MenuItem item){
    super.onOptionsItemSelected(item);

    switch(item.getItemId()) {
      case R.id.item1:Bundle dataBundle = new Bundle();
      dataBundle.putInt("id", 0);

      Intent  intent  =  new  Intent  (getApplicationContext(),  Display
Contact.class);
      intent.putExtras(dataBundle);

      startActivity(intent);
      return true;
      default:
      return super.onOptionsItemSelected(item);
    }
```

```
  }
      public boolean onKeyDown(int keycode, KeyEvent event) {
    if (keycode == KeyEvent.KEYCODE_BACK) {
      moveTaskToBack(true);
    }
    return super.onKeyDown(keycode, event);
  }
}
```

Following is the modified content of display contact activity DisplayContact.java

```
package com.example.sairamkrishna.myapplication;

import android.os.Bundle;
import android.app.Activity;
import android.app.AlertDialog;

import android.content.DialogInterface;
import android.content.Intent;
import android.database.Cursor;

import android.view.Menu;
import android.view.MenuItem;
import android.view.View;

import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

public class DisplayContact extends Activity {
  int from_Where_I_Am_Coming = 0;
  private DBHelper mydb ;

  TextView name ;
  TextView phone;
  TextView email;
  TextView street;
  TextView place;
  int id_To_Update = 0;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_display_contact);
    name = (TextView) findViewById(R.id.editTextName);
    phone = (TextView) findViewById(R.id.editTextPhone);
    email = (TextView) findViewById(R.id.editTextStreet);
    street = (TextView) findViewById(R.id.editTextEmail);
    place = (TextView) findViewById(R.id.editTextCity);

    mydb = new DBHelper(this);

    Bundle extras = getIntent().getExtras();
    if(extras !=null) {
      int Value = extras.getInt("id");
```

```
      if(Value>0){
        //means this is the view part not the add contact part.
        Cursor rs = mydb.getData(Value);
        id_To_Update = Value;
        rs.moveToFirst();

        String                                  nam                                  =
rs.getString(rs.getColumnIndex(DBHelper.CONTACTS_COLUMN_NA
ME));
        String                                  phon                                 =
rs.getString(rs.getColumnIndex(DBHelper.CONTACTS_COLUMN_PHO
NE));
        String                                  emai                                 =
rs.getString(rs.getColumnIndex(DBHelper.CONTACTS_COLUMN_EM
AIL));
        String                                  stree                                =
rs.getString(rs.getColumnIndex(DBHelper.CONTACTS_COLUMN_STR
EET));
        String                                  plac                                 =
rs.getString(rs.getColumnIndex(DBHelper.CONTACTS_COLUMN_CIT
Y));

        if (!rs.isClosed())  {
          rs.close();
        }
        Button b = (Button)findViewById(R.id.button1);
        b.setVisibility(View.INVISIBLE);

        name.setText((CharSequence)nam);
        name.setFocusable(false);
        name.setClickable(false);

        phone.setText((CharSequence)phon);
        phone.setFocusable(false);
        phone.setClickable(false);

        email.setText((CharSequence)emai);
        email.setFocusable(false);
        email.setClickable(false);

        street.setText((CharSequence)stree);
        street.setFocusable(false);
        street.setClickable(false);

        place.setText((CharSequence)plac);
        place.setFocusable(false);
        place.setClickable(false);
      }
    }
  }

  @Override
  public boolean onCreateOptionsMenu(Menu menu) {
```

```
    // Inflate the menu; this adds items to the action bar if it is present.
    Bundle extras = getIntent().getExtras();

    if(extras !=null) {
      int Value = extras.getInt("id");
      if(Value>0){
        getMenuInflater().inflate(R.menu.display_contact, menu);
      } else{
        getMenuInflater().inflate(R.menu.menu_main menu);
      }
    }
    return true;
  }

  public boolean onOptionsItemSelected(MenuItem item) {
    super.onOptionsItemSelected(item);
    switch(item.getItemId()) {
      case R.id.Edit_Contact:
      Button b = (Button)findViewById(R.id.button1);
      b.setVisibility(View.VISIBLE);
      name.setEnabled(true);
      name.setFocusableInTouchMode(true);
      name.setClickable(true);

      phone.setEnabled(true);
      phone.setFocusableInTouchMode(true);
      phone.setClickable(true);

      email.setEnabled(true);
      email.setFocusableInTouchMode(true);
      email.setClickable(true);

      street.setEnabled(true);
      street.setFocusableInTouchMode(true);
      street.setClickable(true);

      place.setEnabled(true);
      place.setFocusableInTouchMode(true);
      place.setClickable(true);

      return true;
      case R.id.Delete_Contact:

      AlertDialog.Builder builder = new AlertDialog.Builder(this);
      builder.setMessage(R.string.deleteContact)
        .setPositiveButton(R.string.yes,                              new
DialogInterface.OnClickListener() {
          public void onClick(DialogInterface dialog, int id) {
            mydb.deleteContact(id_To_Update);
            Toast.makeText(getApplicationContext(),              "Deleted
Successfully",
              Toast.LENGTH_SHORT).show();
```

```
            Intent              intent              =              new
Intent(getApplicationContext(),MainActivity.class);
            startActivity(intent);
          }
      })
      .setNegativeButton(R.string.no,                              new
DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int id) {
          // User cancelled the dialog
        }
      });

      AlertDialog d = builder.create();
      d.setTitle("Are you sure");
      d.show();

      return true;
      default:
      return super.onOptionsItemSelected(item);

    }
  }

  public void run(View view) {
    Bundle extras = getIntent().getExtras();
    if(extras !=null) {
      int Value = extras.getInt("id");
      if(Value>0){
        if(mydb.updateContact(id_To_Update,name.getText().toString(),
          phone.getText().toString(), email.getText().toString(),
                          street.getText().toString(),
place.getText().toString())){
          Toast.makeText(getApplicationContext(),            "Updated",
Toast.LENGTH_SHORT).show();
          Intent              intent              =              new
Intent(getApplicationContext(),MainActivity.class);
          startActivity(intent);
        } else{
          Toast.makeText(getApplicationContext(),     "not     Updated",
Toast.LENGTH_SHORT).show();
        }
      } else{
        if(mydb.insertContact(name.getText().toString(),
phone.getText().toString(),
                          email.getText().toString(),
street.getText().toString(),
                          place.getText().toString())){
            Toast.makeText(getApplicationContext(), "done",
                                  Toast.LENGTH_SHORT).show();
```

```
            } else{
              Toast.makeText(getApplicationContext(), "not done",
                                         Toast.LENGTH_SHORT).show();
            }
            Intent intent = new
Intent(getApplicationContext(),MainActivity.class);
            startActivity(intent);
          }
        }
      }
}
```

Following is the content of Database class DBHelper.java

```
package com.example.sairamkrishna.myapplication;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Hashtable;
import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.DatabaseUtils;
import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteDatabase;

public class DBHelper extends SQLiteOpenHelper {

  public static final String DATABASE_NAME = "MyDBName.db";
  public static final String CONTACTS_TABLE_NAME = "contacts";
  public static final String CONTACTS_COLUMN_ID = "id";
  public static final String CONTACTS_COLUMN_NAME = "name";
  public static final String CONTACTS_COLUMN_EMAIL = "email";
  public static final String CONTACTS_COLUMN_STREET = "street";
  public static final String CONTACTS_COLUMN_CITY = "place";
  public static final String CONTACTS_COLUMN_PHONE = "phone";
  private HashMap hp;

  public DBHelper(Context context) {
    super(context, DATABASE_NAME , null, 1);
  }

  @Override
  public void onCreate(SQLiteDatabase db) {
    // TODO Auto-generated method stub
    db.execSQL(
      "create table contacts " +
      "(id integer primary key, name text,phone text,email text, street
text,place text)"
    );
  }

  @Override
```

```
  public void onUpgrade(SQLiteDatabase db, int oldVersion, int
newVersion) {
    // TODO Auto-generated method stub
    db.execSQL("DROP TABLE IF EXISTS contacts");
    onCreate(db);
  }
  public boolean insertContact (String name, String phone, String email,
String street,String place) {
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues contentValues = new ContentValues();
    contentValues.put("name", name);
    contentValues.put("phone", phone);
    contentValues.put("email", email);
    contentValues.put("street", street);
    contentValues.put("place", place);
    db.insert("contacts", null, contentValues);
    return true;
  }

  public Cursor getData(int id) {
    SQLiteDatabase db = this.getReadableDatabase();
    Cursor res =  db.rawQuery( "select * from contacts where id="+id+"",
null );
    return res;
  }
  public int numberOfRows(){
  SQLiteDatabase db = this.getReadableDatabase();
  int numRows = (int) DatabaseUtils.queryNumEntries(db,
CONTACTS_TABLE_NAME);
    return numRows;
  }
public boolean updateContact (Integer id, String name, String phone,
String email, String street,String place) {
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues contentValues = new ContentValues();
    contentValues.put("name", name);
    contentValues.put("phone", phone);
    contentValues.put("email", email);
    contentValues.put("street", street);
    contentValues.put("place", place);
    db.update("contacts", contentValues, "id = ? ", new String[] {
Integer.toString(id) } );
    return true;
  }
      public Integer deleteContact (Integer id) {
    SQLiteDatabase db = this.getWritableDatabase();
    return db.delete("contacts",
    "id = ? ",
    new String[] { Integer.toString(id) });
```

```
  }
    public ArrayList<String> getAllCotacts() {
    ArrayList<String> array_list = new ArrayList<String>();

    //hp = new HashMap();
    SQLiteDatabase db = this.getReadableDatabase();
    Cursor res =  db.rawQuery( "select * from contacts", null );
    res.moveToFirst();
    while(res.isAfterLast() == false){
array_list.add(res.getString(res.getColumnIndex(CONTACTS_COLUMN
_NAME)));
      res.moveToNext();
    }
    return array_list;
  }
}
```

Following is the content of the **res/layout/activity_main.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  android:paddingBottom="@dimen/activity_vertical_margin"
tools:context=".MainActivity">

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/textView"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:textSize="30dp"
```

```
  android:text="Data Base" />

<TextView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Tutorials Point"
  android:id="@+id/textView2"
  android:layout_below="@+id/textView"
  android:layout_centerHorizontal="true"
  android:textSize="35dp"
  android:textColor="#ff16ff01" />

<ImageView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:id="@+id/imageView"
  android:layout_below="@+id/textView2"
  android:layout_centerHorizontal="true"
  android:src="@drawable/logo"/>

<ScrollView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:id="@+id/scrollView"
  android:layout_below="@+id/imageView"
  android:layout_alignParentLeft="true"
  android:layout_alignParentStart="true"
  android:layout_alignParentBottom="true"
  android:layout_alignParentRight="true"
```

```
    android:layout_alignParentEnd="true">


    <ListView

      android:id="@+id/listView1"

      android:layout_width="match_parent"

      android:layout_height="wrap_content"

      android:layout_centerHorizontal="true"

      android:layout_centerVertical="true" >

    </ListView>


  </ScrollView>


</RelativeLayout>
```

Following is the content of the **res/layout/activity_display_contact.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView
xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/scrollView1"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  tools:context=".DisplayContact" >

  <RelativeLayout
    android:layout_width="match_parent"
    android:layout_height="370dp"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin">

    <EditText
      android:id="@+id/editTextName"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentLeft="true"
      android:layout_marginTop="5dp"
      android:layout_marginLeft="82dp"
      android:ems="10"
      android:inputType="text" >
    </EditText>
```

```
<EditText
  android:id="@+id/editTextEmail"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_alignLeft="@+id/editTextStreet"
  android:layout_below="@+id/editTextStreet"
  android:layout_marginTop="22dp"
  android:ems="10"
  android:inputType="textEmailAddress" />

<TextView
  android:id="@+id/textView1"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_alignBottom="@+id/editTextName"
  android:layout_alignParentLeft="true"
  android:text="@string/name"
  android:textAppearance="?android:attr/textAppearanceMedium" />

<Button
  android:id="@+id/button1"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_alignLeft="@+id/editTextCity"
  android:layout_alignParentBottom="true"
  android:layout_marginBottom="28dp"
  android:onClick="run"
  android:text="@string/save" />

<TextView
  android:id="@+id/textView2"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_alignBottom="@+id/editTextEmail"
  android:layout_alignLeft="@+id/textView1"
  android:text="@string/email"
  android:textAppearance="?android:attr/textAppearanceMedium" />

<TextView
  android:id="@+id/textView5"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_alignBottom="@+id/editTextPhone"
  android:layout_alignLeft="@+id/textView1"
  android:text="@string/phone"
  android:textAppearance="?android:attr/textAppearanceMedium" />

<TextView
  android:id="@+id/textView4"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_above="@+id/editTextEmail"
```

```
        android:layout_alignLeft="@+id/textView5"
        android:text="@string/street"
        android:textAppearance="?android:attr/textAppearanceMedium" />
    <EditText
        android:id="@+id/editTextCity"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignRight="@+id/editTextName"
        android:layout_below="@+id/editTextEmail"
        android:layout_marginTop="30dp"
        android:ems="10"
        android:inputType="text" />
    <TextView
        android:id="@+id/textView3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBaseline="@+id/editTextCity"
        android:layout_alignBottom="@+id/editTextCity"
        android:layout_alignParentLeft="true"
        android:layout_toLeftOf="@+id/editTextEmail"
        android:text="@string/country"
        android:textAppearance="?android:attr/textAppearanceMedium" />
    <EditText
        android:id="@+id/editTextStreet"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/editTextName"
        android:layout_below="@+id/editTextPhone"
        android:ems="10"
        android:inputType="text" >
        <requestFocus />
    </EditText>
    <EditText
        android:id="@+id/editTextPhone"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/editTextStreet"
        android:layout_below="@+id/editTextName"
        android:ems="10"
        android:inputType="phone|text" />
  </RelativeLayout>
</ScrollView>
```

Following is the content of the **res/value/string.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">Address Book</string>
  <string name="action_settings">Settings</string>
  <string name="hello_world">Hello world!</string>
  <string name="Add_New">Add New</string>
  <string name="edit">Edit Contact</string>
  <string name="delete">Delete Contact</string>
  <string name="title_activity_display_contact">DisplayContact</string>
  <string name="name">Name</string>
  <string name="phone">Phone</string>
  <string name="email">Email</string>
  <string name="street">Street</string>
  <string name="country">City/State/Zip</string>
  <string name="save">Save Contact</string>
  <string name="deleteContact">Are you sure, you want to delete
it.</string>
  <string name="yes">Yes</string>
  <string name="no">No</string>
</resources>
```

Following is the content of the **res/menu/main_menu.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >

  <item android:id="@+id/item1"
    android:icon="@drawable/add"
    android:title="@string/Add_New" >
  </item>

</menu>
```

Following is the content of the **res/menu/display_contact.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
  <item
    android:id="@+id/Edit_Contact"
    android:orderInCategory="100"
    android:title="@string/edit"/>

  <item
    android:id="@+id/Delete_Contact"
    android:orderInCategory="100"
    android:title="@string/delete"/>

</menu>
```

This is the defualt **AndroidManifest.xml** of this project

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.example.sairamkrishna.myapplication" >

  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >

    <activity
      android:name=".MainActivity"
      android:label="@string/app_name" >

      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category    android:name="android.intent.category.LAUNCHER"
/>
      </intent-filter>

    </activity>

    <activity android:name=".DisplayContact"/>

  </application>
</manifest>
```

Webkit

WebKit is the web browser engine used by Safari, Mail, App Store, and many other apps on macOS, iOS, and Linux. Get started contributing code, or reporting bugs.

Web developers can follow development, check feature status, download Safari Technology Preview to try out the latest web technologies, and report bugs.

OpenGL

Android supports OpenGL both through its framework API and the Native Development Kit (NDK). This topic focuses on the Android framework interfaces. For more information about the NDK, see the Android NDK.

There are two foundational classes in the Android framework that let you create and manipulate graphics with the OpenGL ES API: GLSurfaceView and GLSurfaceView.Renderer. If your goal is to use OpenGL in your Android application, understanding how to implement these classes in an activity should be your first objective.

GLSurfaceView

This class is a View where you can draw and manipulate objects using OpenGL API calls and is similar in function to a SurfaceView. You can use this class by creating an instance of GLSurfaceView and adding your Renderer to it. However, if you want to capture touch screen events, you should extend the GLSurfaceView class to implement the touch listeners, as shown in OpenGL training lesson, Responding to touch events.

GLSurfaceView.Renderer

This interface defines the methods required for drawing graphics in a GLSurfaceView. You must provide an implementation of this interface as a separate class and attach it to your GLSurfaceView instance using GLSurfaceView.setRenderer().

The GLSurfaceView.Renderer interface requires that you implement the following methods:

- onSurfaceCreated(): The system calls this method once, when creating the GLSurfaceView. Use this method to perform actions that need to happen only once, such as setting OpenGL environment parameters or initializing OpenGL graphic objects.
- onDrawFrame(): The system calls this method on each redraw of the GLSurfaceView. Use this method as the primary execution point for drawing (and re-drawing) graphic objects.
- onSurfaceChanged(): The system calls this method when the GLSurfaceView geometry changes, including changes in size of the GLSurfaceView or orientation of the device screen. For example, the system calls this method when the device changes from portrait to landscape orientation. Use this method to respond to changes in the GLSurfaceView container.

The Dalvik Virtual Machine

One of the key elements of Android is the Dalvik VM. Rather than using a traditional Java VM such as Java ME, Android uses its own custom VM designed to ensure that multiple instances run efficiently on a single device.

The Dalvik VM uses the device's underlying Linux kernel to handle low-level functionality, including security, threading, and process and memory management. It's also possible to write C/C++ applications that run closer to the underlying Linux OS. Although you can do this, in most cases there's no reason you should need to.

If the speed and efficiency of C/C++ is required for your application, Android provides a native development kit (NDK). The NDK is designed to enable you to create C++ libraries using the libc and libm libraries, along with native access to OpenGL.

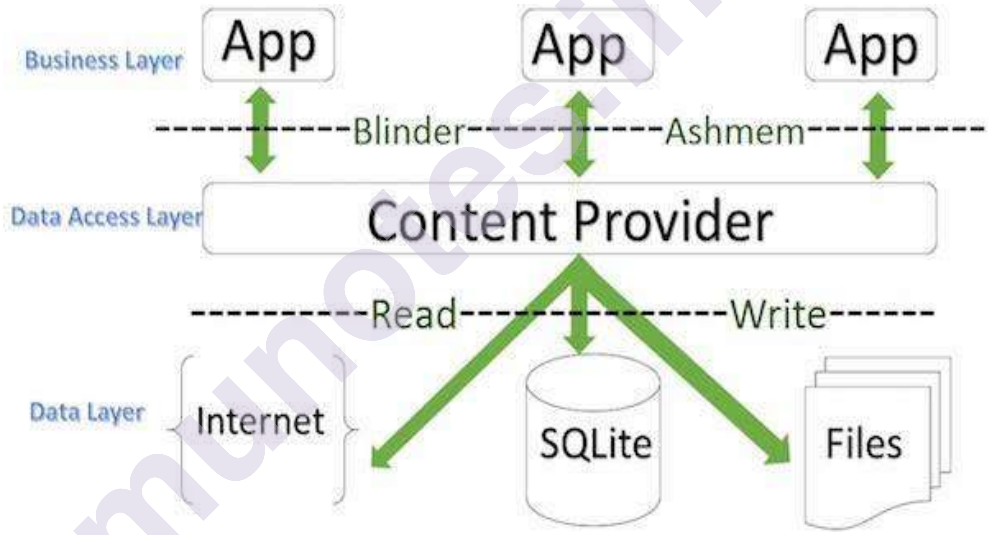
All Android hardware and system service access is managed using Dalvik as a middle tier. By using a VM to host application execution, developers have an abstraction layer that ensures they should never have to worry about a particular hardware implementation.

The Dalvik VM executes Dalvik executable files, a format optimized to ensure minimal memory footprint. You create .dex executables by transforming Java language compiled classes using the tools supplied within the SDK

4.4 APPLICATION FRAMEWORK :-

Activity Manager − Controls all aspects of the application lifecycle and activity stack. Content Providers − Allows applications to publish and share data with other applications. Resource Manager − Provides access to non-code embedded resources such as strings, color settings and user interface layouts.

A content provider component supplies data from one application to others on request. Such requests are handled by the methods of the ContentResolver class. A content provider can use different ways to store its data and the data can be stored in a database, in files, or even over a network.



ContentProvider

sometimes it is required to share data across applications. This is where content providers become very useful.

Content providers let you centralize content in one place and have many different applications access it as needed. A content provider behaves very much like a database where you can query it, edit its content, as well as add or delete content using insert(), update(), delete(), and query() methods. In most cases this data is stored in an **SQlite** database.

A content provider is implemented as a subclass of **ContentProvider** class and must implement a standard set of APIs that enable other applications to perform transactions.

```
public class My Application extends  ContentProvider {

}
```

Content URIs

To query a content provider, you specify the query string in the form of a URI which has following format −

```
<prefix>://<authority>/<data_type>/<id>
```

Here is the detail of various parts of the URI −

Sr.No	Part & Description
1	**prefix** This is always set to content://
2	**authority** This specifies the name of the content provider, for example *contacts*, *browser* etc. For third-party content providers, this could be the fully qualified name, such as *com.tutorialspoint.statusprovider*
3	**data_type** This indicates the type of data that this particular provider provides. For example, if you are getting all the contacts from the *Contacts* content provider, then the data path would be *people* and URI would look like this*content://contacts/people*
4	**id** This specifies the specific record requested. For example, if you are looking for contact number 5 in the Contacts content provider then URI would look like this *content://contacts/people/5*.

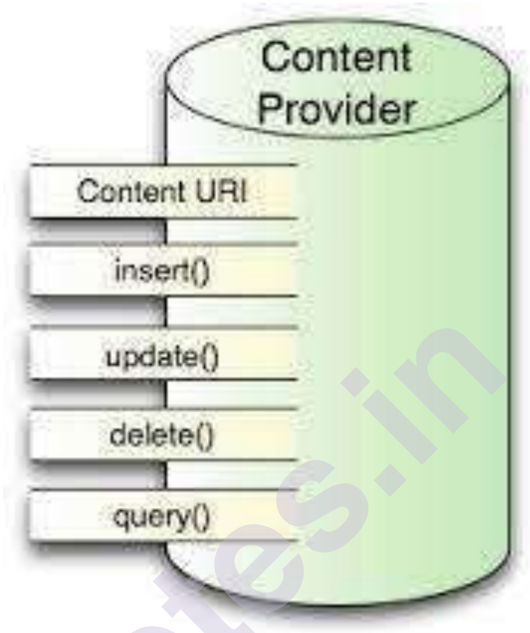
Create Content Provider

This involves number of simple steps to create your own content provider.

- First of all you need to create a Content Provider class that extends the ContentProviderbaseclass.
- Second, you need to define your content provider URI address which will be used to access the content.
- Next you will need to create your own database to keep the content. Usually, Android uses SQLite database and framework needs to override onCreate() method which will use SQLite Open Helper method to create or open the provider's database. When your application is launched, the onCreate() handler of each of its Content Providers is called on the main application thread.

- Next you will have to implement Content Provider queries to perform different database specific operations.
- Finally register your Content Provider in your activity file using <provider> tag.

Here is the list of methods which you need to override in Content Provider class to have your Content Provider working –



ContentProvider

- onCreate() This method is called when the provider is started.
- query() This method receives a request from a client. The result is returned as a Cursor object.
- insert()This method inserts a new record into the content provider.
- delete() This method deletes an existing record from the content provider.
- update() This method updates an existing record from the content provider.
- getType() This method returns the MIME type of the data at the given URI.

Example

This example will explain you how to create your own ContentProvider. So let's follow the following steps to similar to what we followed while creating Hello World Example−

Step	Description
1	You will use Android StudioIDE to create an Android application and name it as *My Application* under a package *com.example.MyApplication*, with blank Activity.
2	Modify main activity file *MainActivity.java* to add two new methods *onClickAddName()* and *onClickRetrieveStudents()*.
3	Create a new java file called *StudentsProvider.java* under the package *com.example.MyApplication* to define your actual provider and associated methods.
4	Register your content provider in your *AndroidManifest.xml* file using <provider.../> tag
5	Modify the default content of *res/layout/activity_main.xml* file to include a small GUI to add students records.
6	No need to change string.xml.Android studio take care of string.xml file.
7	Run the application to launch Android emulator and verify the result of the changes done in the application.

Following is the content of the modified main activity file **src/com.example.MyApplication/MainActivity.java.** This file can include each of the fundamental life cycle methods. We have added two new methods onClickAddName() and onClickRetrieveStudents() to handle user interaction with the application.

```
package com.example.MyApplication;

import android.net.Uri;
import android.os.Bundle;
import android.app.Activity;

import android.content.ContentValues;
import android.content.CursorLoader;

import android.database.Cursor;

import android.view.Menu;
import android.view.View;

import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends Activity {
```

```
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }
  public void onClickAddName(View view) {
    // Add a new student record
    ContentValues values = new ContentValues();
    values.put(StudentsProvider.NAME,
      ((EditText)findViewById(R.id.editText2)).getText().toString());

    values.put(StudentsProvider.GRADE,
      ((EditText)findViewById(R.id.editText3)).getText().toString());

    Uri uri = getContentResolver().insert(
      StudentsProvider.CONTENT_URI, values);

    Toast.makeText(getBaseContext(),
      uri.toString(), Toast.LENGTH_LONG).show();
  }
  public void onClickRetrieveStudents(View view) {
    // Retrieve student records
    String                                URL                                =
"content://com.example.MyApplication.StudentsProvider";

    Uri students = Uri.parse(URL);
    Cursor c = managedQuery(students, null, null, null, "name");

    if (c.moveToFirst()) {
      do{
        Toast.makeText(this,
          c.getString(c.getColumnIndex(StudentsProvider._ID)) +
            ",      "      +            c.getString(c.getColumnIndex(
StudentsProvider.NAME)) +
              ",        "        +        c.getString(c.getColumnIndex(
StudentsProvider.GRADE)),
        Toast.LENGTH_SHORT).show();
      } while (c.moveToNext());
    }
  }
}
```

Create new file StudentsProvider.java under com.example.MyApplication package and following is the content of **src/com.example.MyApplication/StudentsProvider.java** −

```
package com.example.MyApplication;
import java.util.HashMap;
import android.content.ContentProvider;
import android.content.ContentUris;
import android.content.ContentValues;
import android.content.Context;
import android.content.UriMatcher;
import android.database.Cursor;
import android.database.SQLException;

import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteQueryBuilder;
import android.net.Uri;
import android.text.TextUtils;
public class StudentsProvider extends ContentProvider {
  static final String PROVIDER_NAME =
"com.example.MyApplication.StudentsProvider";
  static final String URL = "content://" + PROVIDER_NAME +
"/students";
  static final Uri CONTENT_URI = Uri.parse(URL);
  static final String _ID = "_id";
  static final String NAME = "name";
  static final String GRADE = "grade";
  private static HashMap<String, String>
STUDENTS_PROJECTION_MAP;
  static final int STUDENTS = 1;
  static final int STUDENT_ID = 2;
  static final UriMatcher uriMatcher;
  static{
    uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
    uriMatcher.addURI(PROVIDER_NAME, "students", STUDENTS);
    uriMatcher.addURI(PROVIDER_NAME, "students/#",
STUDENT_ID);
  }
  /**
    * Database specific constant declarations
  */
private SQLiteDatabase db;
  static final String DATABASE_NAME = "College";
  static final String STUDENTS_TABLE_NAME = "students";
  static final int DATABASE_VERSION = 1;
  static final String CREATE_DB_TABLE =
    " CREATE TABLE " + STUDENTS_TABLE_NAME +
      " (_id INTEGER PRIMARY KEY AUTOINCREMENT, " +
      " name TEXT NOT NULL, " +
      " grade TEXT NOT NULL);";
  /**
    * Helper class that actually creates and manages
    * the provider's underlying data repository.
```

```
 */
   private static class DatabaseHelper extends SQLiteOpenHelper {
   DatabaseHelper(Context context){
     super(context,              DATABASE_NAME,              null,
DATABASE_VERSION);
   }
   @Override
   public void onCreate(SQLiteDatabase db) {
     db.execSQL(CREATE_DB_TABLE);
   }
   @Override
   public  void  onUpgrade(SQLiteDatabase  db,  int  oldVersion,  int
newVersion) {
     db.execSQL("DROP      TABLE      IF      EXISTS      "      +
STUDENTS_TABLE_NAME);
     onCreate(db);
   }
 }
 @Override
 public boolean onCreate() {
   Context context = getContext();
   DatabaseHelper dbHelper = new DatabaseHelper(context);
   /**
     * Create a write able database which will trigger its
     * creation if it doesn't already exist.
   */
    db = dbHelper.getWritableDatabase();
   return (db == null)? false:true;
 }
 @Override
 public Uri insert(Uri uri, ContentValues values) {
   /**
     * Add a new student record
   */
   long rowID = db.insert(   STUDENTS_TABLE_NAME, "", values);
   /**
     * If record is added successfully
   */
   if (rowID > 0) {
     Uri _uri = ContentUris.withAppendedId(CONTENT_URI, rowID);
     getContext().getContentResolver().notifyChange(_uri, null);
     return _uri;
   }
  throw new SQLException("Failed to add a record into " + uri);
 }
 @Override
 public Cursor query(Uri uri, String[] projection,
   String selection,String[] selectionArgs, String sortOrder) {
   SQLiteQueryBuilder qb = new SQLiteQueryBuilder();
   qb.setTables(STUDENTS_TABLE_NAME);
```

```
    switch (uriMatcher.match(uri)) {
      case STUDENTS:
        qb.setProjectionMap(STUDENTS_PROJECTION_MAP);
      break;

      case STUDENT_ID:
        qb.appendWhere( _ID + "=" + uri.getPathSegments().get(1));
      break;

      default:
    }

    if (sortOrder == null || sortOrder == ""){
      /**
        * By default sort on student names
      */
      sortOrder = NAME;
    }

    Cursor c = qb.query(db,  projection,  selection,
      selectionArgs,null, null, sortOrder);
    /**
      * register to watch a content URI for changes
    */
    c.setNotificationUri(getContext().getContentResolver(), uri);
    return c;
  }

  @Override
  public int delete(Uri uri, String selection, String[] selectionArgs) {
    int count = 0;
    switch (uriMatcher.match(uri)){
      case STUDENTS:
        count  =  db.delete(STUDENTS_TABLE_NAME,  selection,
selectionArgs);
      break;

      case STUDENT_ID:
        String id = uri.getPathSegments().get(1);
        count = db.delete( STUDENTS_TABLE_NAME, _ID +  " = " + id
+
          (!TextUtils.isEmpty(selection) ? "
          AND (" + selection + ')' : ""), selectionArgs);
        break;
      default:
        throw new IllegalArgumentException("Unknown URI " + uri);
    }

    getContext().getContentResolver().notifyChange(uri, null);
```

```
      return count;
   }

   @Override
   public int update(Uri uri, ContentValues values,
      String selection, String[] selectionArgs) {
      int count = 0;
      switch (uriMatcher.match(uri)) {
         case STUDENTS:
            count = db.update(STUDENTS_TABLE_NAME, values,
selection, selectionArgs);
         break;

         case STUDENT_ID:
            count = db.update(STUDENTS_TABLE_NAME, values,
               _ID + " = " + uri.getPathSegments().get(1) +
               (!TextUtils.isEmpty(selection) ? "
               AND (" +selection + ')' : ""), selectionArgs);
            break;
         default:
            throw new IllegalArgumentException("Unknown URI " + uri );
      }
       getContext().getContentResolver().notifyChange(uri, null);
      return count;
   }
   @Override
   public String getType(Uri uri) {
      switch (uriMatcher.match(uri)){
         /**
            * Get all student records
         */
         case STUDENTS:
            return "vnd.android.cursor.dir/vnd.example.students";
         /**
            * Get a particular student
         */
         case STUDENT_ID:
            return "vnd.android.cursor.item/vnd.example.students";
         default:
            throw new IllegalArgumentException("Unsupported URI: " + uri);
      }
   }
}
```

Following will the modified content of **AndroidManifest.xml** file. Here we have added <provider.../> tag to include our content provider:

```
<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

    package="com.example.MyApplication">


   <application

      android:allowBackup="true"

      android:icon="@mipmap/ic_launcher"

      android:label="@string/app_name"

      android:supportsRtl="true"

      android:theme="@style/AppTheme">

         <activity android:name=".MainActivity">

            <intent-filter>

               <action android:name="android.intent.action.MAIN" />

               <category  android:name="android.intent.category.LAUNCHER"
/>

            </intent-filter>

         </activity>


      <provider android:name="StudentsProvider"


android:authorities="com.example.MyApplication.StudentsProvider"/>

   </application>

</manifest>
```

Following will be the content of **res/layout/activity_main.xml file**–

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingBottom="@dimen/activity_vertical_margin"
```

```
android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
tools:context="com.example.MyApplication.MainActivity">

<TextView
  android:id="@+id/textView1"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Content provider"
  android:layout_alignParentTop="true"
  android:layout_centerHorizontal="true"
  android:textSize="30dp" />

<TextView
  android:id="@+id/textView2"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Tutorials point "
  android:textColor="#ff87ff09"
  android:textSize="30dp"
  android:layout_below="@+id/textView1"
  android:layout_centerHorizontal="true" />

<ImageButton
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:id="@+id/imageButton"
  android:src="@drawable/abc"
  android:layout_below="@+id/textView2"
  android:layout_centerHorizontal="true" />

<Button
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:id="@+id/button2"
  android:text="Add Name"
  android:layout_below="@+id/editText3"
  android:layout_alignRight="@+id/textView2"
  android:layout_alignEnd="@+id/textView2"
  android:layout_alignLeft="@+id/textView2"
  android:layout_alignStart="@+id/textView2"
  android:onClick="onClickAddName"/>

<EditText
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:id="@+id/editText"
  android:layout_below="@+id/imageButton"
  android:layout_alignRight="@+id/imageButton"
```

```
    android:layout_alignEnd="@+id/imageButton" />

  <EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/editText2"
    android:layout_alignTop="@+id/editText"
    android:layout_alignLeft="@+id/textView1"
    android:layout_alignStart="@+id/textView1"
    android:layout_alignRight="@+id/textView1"
    android:layout_alignEnd="@+id/textView1"
    android:hint="Name"
    android:textColorHint="@android:color/holo_blue_light" />

  <EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/editText3"
    android:layout_below="@+id/editText"
    android:layout_alignLeft="@+id/editText2"
    android:layout_alignStart="@+id/editText2"
    android:layout_alignRight="@+id/editText2"
    android:layout_alignEnd="@+id/editText2"
    android:hint="Grade"
    android:textColorHint="@android:color/holo_blue_bright" />

  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Retrive student"
    android:id="@+id/button"
    android:layout_below="@+id/button2"
    android:layout_alignRight="@+id/editText3"
    android:layout_alignEnd="@+id/editText3"
    android:layout_alignLeft="@+id/button2"
    android:layout_alignStart="@+id/button2"
    android:onClick="onClickRetrieveStudents"/>
</RelativeLayout>
```

Make sure you have following content of **res/values/strings.xml file:**

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="app_name">My Application</string>
</resources>;
```

Android TelephonyManager Tutorial

The **android.telephony.TelephonyManager** class provides information about the telephony services such as subscriber id, sim serial number, phone network type etc. Moreover, you can determine the phone state etc.

Android TelephonyManager Example

Let's see the simple example of TelephonyManager that prints information of the telephony services.

activity_main.xml

Drag one textview from the pallete, now the xml file will look like this.

File: activity_main.xml

```
<RelativeLayout
xmlns:androclass="http://schemas.android.com/apk/res/android"
 xmlns:tools="http://schemas.android.com/tools"
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 android:paddingBottom="@dimen/activity_vertical_margin"
 android:paddingLeft="@dimen/activity_horizontal_margin"
 android:paddingRight="@dimen/activity_horizontal_margin"
 android:paddingTop="@dimen/activity_vertical_margin"
tools:context=".MainActivity" >

<TextView
android:id="@+id/textView1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_alignParentLeft="true"
android:layout_alignParentTop="true"
android:layout_marginLeft="38dp"
android:layout_marginTop="30dp"
android:text="Phone Details:" />

</RelativeLayout>
```

Activity Class

Now, write the code to display the information about the telephony services.

File: MainActivity.java

```
package com.javatpoint.telephonymanager;

import android.os.Bundle;
```

```
import android.app.Activity;
import android.content.Context;
import android.telephony.TelephonyManager;
import android.view.Menu;
import android.widget.TextView;

public class MainActivity extends Activity {
TextView textView1;
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);

textView1=(TextView)findViewById(R.id.textView1);

//Get the instance of TelephonyManager
TelephonyManager tm=(TelephonyManager)getSystemService(Context.TELEPHONY_SERVICE);

//Calling the methods of TelephonyManager the returns the information
      String IMEINumber=tm.getDeviceId();
      String subscriberID=tm.getDeviceId();
      String SIMSerialNumber=tm.getSimSerialNumber();
      String networkCountryISO=tm.getNetworkCountryIso();
      String SIMCountryISO=tm.getSimCountryIso();
      String softwareVersion=tm.getDeviceSoftwareVersion();
      String voiceMailNumber=tm.getVoiceMailNumber();

      //Get the phone type
      String strphoneType="";

      int phoneType=tm.getPhoneType();

      switch (phoneType)
      {
            case (TelephonyManager.PHONE_TYPE_CDMA):
                    strphoneType="CDMA";
                       break;
            case (TelephonyManager.PHONE_TYPE_GSM):
                    strphoneType="GSM";
                       break;
            case (TelephonyManager.PHONE_TYPE_NONE):
```

```
                    strphoneType="NONE";
                      break;
       }

      //getting information if phone is in roaming
      boolean isRoaming=tm.isNetworkRoaming();

      String info="Phone Details:\n";
      info+="\n IMEI Number:"+IMEINumber;
      info+="\n SubscriberID:"+subscriberID;
      info+="\n Sim Serial Number:"+SIMSerialNumber;
      info+="\n Network Country ISO:"+networkCountryISO;
      info+="\n SIM Country ISO:"+SIMCountryISO;
      info+="\n Software Version:"+softwareVersion;
      info+="\n Voice Mail Number:"+voiceMailNumber;
      info+="\n Phone Network Type:"+strphoneType;
      info+="\n In Roaming? :"+isRoaming;

textView1.setText(info);//displaying the information in the textView
   }


}
```

AndroidManifest.xml

You need to provide **READ_PHONE_STATE** permission in the AndroidManifest.xml file.

File: AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:androclass="http://schemas.android.com/apk/res/android"
package="com.javatpoint.telephonymanager"
android:versionCode="1"
android:versionName="1.0" >

<uses-sdk
android:minSdkVersion="8"
android:targetSdkVersion="17" />

<uses-permission
android:name="android.permission.READ_PHONE_STATE"/>

```

```
<application
android:allowBackup="true"
android:icon="@drawable/ic_launcher"
android:label="@string/app_name"
android:theme="@style/AppTheme" >
<activity
android:name="com.javatpoint.telephonymanager.MainActivity"
android:label="@string/app_name" >
<intent-filter>
<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
</application>

</manifest>
```

LocationManager

Kotlin |Java

public class LocationManager

extends Object

java.lang.Object

↳ android.location.LocationManager

This class provides access to the system location services. These services allow applications to obtain periodic updates of the device's geographical location, or to be notified when the device enters the proximity of a given geographical location.

Unless otherwise noted, all Location API methods require the **Manifest.permission.ACCESS_COARSE_LOCATION** or **Manifest.permission.ACCESS_FINE_LOCATION** permissions. If your application only has the coarse permission then providers will still return location results, but the exact location will be obfuscated to a coarse level of accuracy.

Requires the **PackageManager#FEATURE_LOCATION** feature which can be detected using **PackageManager.hasSystemFeature**(String).

Summary

Constants

String	ACTION_GNSS_CAPABILITIES_CHANGED Broadcast intent action when GNSS capabilities change.
String	EXTRA_GNSS_CAPABILITIES Intent extra included with ACTION_GNSS_CAPABILITIES_CHANGED broadcasts, containing the new GnssCapabilities.
String	EXTRA_LOCATION_ENABLED Intent extra included with MODE_CHANGED_ACTION broadcasts, containing the boolean enabled state of location.
String	EXTRA_PROVIDER_ENABLED Intent extra included with PROVIDERS_CHANGED_ACTION broadcasts, containing the boolean enabled state of the location provider that has changed.
String	EXTRA_PROVIDER_NAME Intent extra included with PROVIDERS_CHANGED_ACTION broadcasts, containing the name of the location provider that has changed.
String	FUSED_PROVIDER Standard name of the fused location provider.
String	GPS_PROVIDER Standard name of the GNSS location provider.
String	KEY_FLUSH_COMPLETE Key used for an extra holding an integer request code when location flush completion is sent using a PendingIntent.
String	KEY_LOCATIONS Key used for an extra holding a array of Locations when a location change is sent using a PendingIntent.
String	KEY_LOCATION_CHANGED Key used for an extra holding a Location value when a location change is sent using a PendingIntent.
String	KEY_PROVIDER_ENABLED Key used for an extra holding a boolean enabled/disabled status value when a provider enabled/disabled event is broadcast using a PendingIntent.
String	KEY_PROXIMITY_ENTERING

	Key used for the Bundle extra holding a boolean indicating whether a proximity alert is entering (true) or exiting (false)..
String	KEY_STATUS_CHANGED This constant was deprecated in API level 29. Status changes are deprecated and no longer broadcast from Android Q onwards.
String	MODE_CHANGED_ACTION Broadcast intent action when the device location enabled state changes.
String	NETWORK_PROVIDER Standard name of the network location provider.
String	PASSIVE_PROVIDER A special location provider for receiving locations without actively initiating a location fix.
String	PROVIDERS_CHANGED_ACTION Broadcast intent action when the set of enabled location providers changes.

Public methods

boolean	addGpsStatusListener(GpsStatus.Listener listener) *This method was deprecated in API level 24.* *use registerGnssStatusCallback(android.location.GnssStatus.Callback) instead. No longer supported in apps targeting S and above.*
boolean	addNmeaListener(OnNmeaMessageListener listener, Handler handler) Adds an NMEA listener.
boolean	addNmeaListener(OnNmeaMessageListener listener) *This method was deprecated in API level 30.* *Use addNmeaListener(android.location.OnNmeaMessageListener, android.os.Handler) or addNmeaListener(java.util.concurrent.Executor, android.location.OnNmeaMessageListener) instead.*
boolean	addNmeaListener(GpsStatus.NmeaListener listener) *This method was deprecated in API level 24.* *Use addNmeaListener(GpsStatus.NmeaListener) instead.*
boolean	addNmeaListener(Executor executor, OnNmeaMessageListener listener) Adds an NMEA listener.
void	addProximityAlert(double latitude, double longitude, float radius, long expiration, PendingIntent pendingIntent) Sets a proximity alert for the location given by the position

	(latitude, longitude) and the given radius.
void	addTestProvider(String provider, ProviderProperties properties) Creates a test location provider and adds it to the set of active providers.
void	addTestProvider(String provider, ProviderProperties properties, Set<String> extraAttributionTags) Creates a test location provider and adds it to the set of active providers.
void	addTestProvider(String provider, boolean requiresNetwork, boolean requiresSatellite, boolean requiresCell, boolean hasMonetaryCost, boolean supportsAltitude, boolean supportsSpeed, boolean supportsBearing, int powerUsage, int accuracy) Creates a test location provider and adds it to the set of active providers.
void	clearTestProviderEnabled(String provider) *This method was deprecated in API level 29. Use setTestProviderEnabled(java.lang.String, boolean) instead.*
void	clearTestProviderLocation(String provider) *This method was deprecated in API level 29. This method has always been a no-op, and may be removed in the future.*
void	clearTestProviderStatus(String provider) *This method was deprecated in API level 29. This method has no effect.*
List<String>	getAllProviders() Returns a list of the names of all available location providers.
String	getBestProvider(Criteria criteria, boolean enabledOnly) Returns the name of the provider that best meets the given criteria.
void	getCurrentLocation(String provider, LocationRequest locationRequest, CancellationSignal cancellationSignal, Executor executor, Consumer<Location> consumer) Asynchronously returns a single current location fix from the given provider based on the given LocationRequest.
void	getCurrentLocation(String provider, CancellationSignal cancellationSignal, Executor executor, Consumer<Location> consumer) Asynchronously returns a single current location fix from the given provider.
List<GnssAntennaInfo>	getGnssAntennaInfos() Returns the current list of GNSS antenna infos, or null if unknown or unsupported.

GnssCapabilities	getGnssCapabilities() Returns the supported capabilities of the GNSS chipset.
String	getGnssHardwareModelName() Returns the model name (including vendor and hardware/software version) of the GNSS hardware driver, or null if this information is not available.
int	getGnssYearOfHardware() Returns the model year of the GNSS hardware and software build, or 0 if the model year is before 2016.
GpsStatus	getGpsStatus(GpsStatus status) *This method was deprecated in API level 24. GpsStatus APIs are deprecated, use GnssStatus APIs instead. No longer supported in apps targeting S and above.*
Location	getLastKnownLocation(String provider) Gets the last known location from the given provider, or null if there is no last known location.
LocationProvider	getProvider(String provider) *This method was deprecated in API level 31. This method has no way to indicate that a provider's properties are unknown, and so may return incorrect results on rare occasions.* *Use getProviderProperties(java.lang.String) instead.*
ProviderProperties	getProviderProperties(String provider) Returns the properties of the given provider, or null if the properties are currently unknown.
List<String>	getProviders(boolean enabledOnly) Returns a list of the names of available location providers.
List<String>	getProviders(Criteria criteria, boolean enabledOnly) Returns a list of the names of available location providers that satisfy the given criteria.
boolean	hasProvider(String provider) Returns true if the given location provider exists on this device, irrespective of whether it is currently enabled or not.
boolean	isLocationEnabled() Returns the current enabled/disabled state of location.
boolean	isProviderEnabled(String provider) Returns the current enabled/disabled status of the given provider.
boolean	registerAntennaInfoListener(Executor executor, GnssAntennaInfo.Listener listener) Registers a GNSS antenna info listener that will receive all changes to antenna info.
boolean	registerGnssMeasurementsCallback(Executor executor, GnssMeasurementsEvent.Callback callback)

	Registers a GNSS measurements callback.
boolean	registerGnssMeasurementsCallback(GnssMeasurementsEvent.Callback callback) *This method was deprecated in API level 30.* *Use registerGnssMeasurementsCallback(GnssMeasurementsEvent.Callback, Handler) or registerGnssMeasurementsCallback(Executor, GnssMeasurementsEvent.Callback) instead.* *Requires Manifest.permission.ACCESS_FINE_LOCATION*
boolean	registerGnssMeasurementsCallback(GnssMeasurementRequest request, Executor executor, GnssMeasurementsEvent.Callback callback) Registers a GNSS measurement callback.
boolean	registerGnssMeasurementsCallback(GnssMeasurementsEvent.Callback callback, Handler handler) Registers a GNSS measurements callback.
boolean	registerGnssNavigationMessageCallback(GnssNavigationMessage.Callback callback, Handler handler) Registers a GNSS navigation message callback.
boolean	registerGnssNavigationMessageCallback(Executor executor, GnssNavigationMessage.Callback callback) Registers a GNSS navigation message callback.
boolean	registerGnssNavigationMessageCallback(GnssNavigationMessage.Callback callback) *This method was deprecated in API level 30.* *Use registerGnssNavigationMessageCallback(android.location.GnssNavigationMessage.Callback, android.os.Handler) or registerGnssNavigationMessageCallback(java.util.concurrent.Executor, android.location.GnssNavigationMessage.Callback) instead.*
boolean	registerGnssStatusCallback(GnssStatus.Callback callback) *This method was deprecated in API level 30.* *Use registerGnssStatusCallback(android.location.GnssStatus.Callback, android.os.Handler) or registerGnssStatusCallback(java.util.concurrent.Executor, android.location.GnssStatus.Callback) instead.*
boolean	registerGnssStatusCallback(GnssStatus.Callback callback, Handler handler) Registers a GNSS status callback.
boolean	registerGnssStatusCallback(Executor executor, GnssStatus.Callback callback) Registers a GNSS status callback.
void	removeGpsStatusListener(GpsStatus.Listener listener)

	This method was deprecated in API level 24. use unregisterGnssStatusCallback(android.location.GnssStatus.Callback) instead. No longer supported in apps targeting S and above.
void	removeNmeaListener(OnNmeaMessageListener listener) Removes an NMEA listener.
void	removeNmeaListener(GpsStatus.NmeaListener listener) *This method was deprecated in API level 24. Use removeNmeaListener(android.location.OnNmeaMessageListener) instead.*
void	removeProximityAlert(PendingIntent intent) Removes the proximity alert with the given PendingIntent.
void	removeTestProvider(String provider) Removes the test location provider with the given name or does nothing if no such test location provider exists.
void	removeUpdates(LocationListener listener) Removes all location updates for the specified LocationListener.
void	removeUpdates(PendingIntent pendingIntent) Removes location updates for the specified PendingIntent.
void	requestFlush(String provider, LocationListener listener, int requestCode) Requests that the given provider flush any batched locations to listeners.
void	requestFlush(String provider, PendingIntent pendingIntent, int requestCode) Requests that the given provider flush any batched locations to listeners.
void	requestLocationUpdates(String provider, long minTimeMs, float minDistanceM, LocationListener listener) Register for location updates from the given provider with the given arguments, and a callback on the Looper of the calling thread.
void	requestLocationUpdates(long minTimeMs, float minDistanceM, Criteria criteria, LocationListener listener, Looper looper) *This method was deprecated in API level 31. Use requestLocationUpdates(java.lang.String, long, float, android.location.LocationListener, android.os.Looper) instead to explicitly select a provider.*
void	requestLocationUpdates(String provider, long minTimeMs, float minDistanceM, LocationListener listener, Looper looper) Register for location updates from the given provider with

	the given arguments, and a callback on the specified Looper.
void	requestLocationUpdates(String provider, long minTimeMs, float minDistanceM, Executor executor, LocationListener listener) Register for location updates using the named provider, and a callback on the specified Executor.
void	requestLocationUpdates(String provider, LocationRequest locationRequest, PendingIntent pendingIntent) Register for location updates from the specified provider, using a LocationRequest, and callbacks delivered via the provided PendingIntent.
void	requestLocationUpdates(String provider, LocationRequest locationRequest, Executor executor, LocationListener listener) Register for location updates from the specified provider, using a LocationRequest, and a callback on the specified Executor.
void	requestLocationUpdates(long minTimeMs, float minDistanceM, Criteria criteria, PendingIntent pendingIntent) *This method was deprecated in API level 31. Use requestLocationUpdates(java.lang.String, long, float, android.app.PendingIntent) instead to explicitly select a provider.*
void	requestLocationUpdates(long minTimeMs, float minDistanceM, Criteria criteria, Executor executor, LocationListener listener) *This method was deprecated in API level 31. Use requestLocationUpdates(java.lang.String, long, float, java.util.concurrent.Executor, android.location.LocationListener) instead to explicitly select a provider.*
void	requestLocationUpdates(String provider, long minTimeMs, float minDistanceM, PendingIntent pendingIntent) Register for location updates using the named provider, and callbacks delivered via the provided PendingIntent.
void	requestSingleUpdate(String provider, PendingIntent pendingIntent) *This method was deprecated in API level 30. Use getCurrentLocation(java.lang.String, android.os.CancellationSignal, java.util.concurrent.Executor,*

	java.util.function.Consumer) instead as it does not carry a risk of extreme battery drain.
void	requestSingleUpdate(Criteria criteria, PendingIntent pendingIntent) *This method was deprecated in API level 30. Use getCurrentLocation(java.lang.String, android.os.CancellationSignal, java.util.concurrent.Executor, java.util.function.Consumer) instead as it does not carry a risk of extreme battery drain.*
void	requestSingleUpdate(String provider, LocationListener listener, Looper looper) *This method was deprecated in API level 30. Use getCurrentLocation(java.lang.String, android.os.CancellationSignal, java.util.concurrent.Executor, java.util.function.Consumer) instead as it does not carry a risk of extreme battery drain.*
void	requestSingleUpdate(Criteria criteria, LocationListener listener, Looper looper) *This method was deprecated in API level 30. Use getCurrentLocation(java.lang.String, android.os.CancellationSignal, java.util.concurrent.Executor, java.util.function.Consumer) instead as it does not carry a risk of extreme battery drain.*
boolean	sendExtraCommand(String provider, String command, Bundle extras) Sends additional commands to a location provider.
void	setTestProviderEnabled(String provider, boolean enabled) Sets the given test provider to be enabled or disabled.
void	setTestProviderLocation(String provider, Location location) Sets a new location for the given test provider.
void	setTestProviderStatus(String provider, int status, Bundle extras, long updateTime) *This method was deprecated in API level 29. This method has no effect.*
void	unregisterAntennaInfoListener(GnssAntennaInfo.Listener listener) Unregisters a GNSS antenna info listener.
void	unregisterGnssMeasurementsCallback(GnssMeasurementsEvent.Callback callback) Unregisters a GPS Measurement callback.
void	unregisterGnssNavigationMessageCallback(GnssNavigatio

	nMessage.Callback callback) Unregisters a GNSS Navigation Message callback.
void	unregisterGnssStatusCallback(GnssStatus.Callback callback) Removes a GNSS status callback.

Inherited methods

From class **java.lang.Object**

Constants

ACTION_GNSS_CAPABILITIES_CHANGED
Added in API level 31

public static final String ACTION_GNSS_CAPABILITIES_CHANGED

Broadcast intent action when GNSS capabilities change. This is most common at boot time as GNSS capabilities are queried from the chipset. Includes an intent extra, EXTRA_GNSS_CAPABILITIES, with the new GnssCapabilities.

See also:

- EXTRA_GNSS_CAPABILITIES
- getGnssCapabilities()

Constant Value: "android.location.action.GNSS_CAPABILITIES_CHANGED"

EXTRA_GNSS_CAPABILITIES

Added in API level 31

public static final **String** EXTRA_GNSS_CAPABILITIES

Intent extra included with **ACTION_GNSS_CAPABILITIES_CHANGED** broadcasts, containing the new **GnssCapabilities**.

See also:

- ACTION_GNSS_CAPABILITIES_CHANGED

Constant Value: "android.location.extra.GNSS_CAPABILITIES"

EXTRA_LOCATION_ENABLED

Added in API level 30

public static final String EXTRA_LOCATION_ENABLED

Intent extra included with MODE_CHANGED_ACTION broadcasts, containing the boolean enabled state of location.

See also:

- MODE_CHANGED_ACTION

Constant Value: "android.location.extra.LOCATION_ENABLED"

EXTRA_PROVIDER_ENABLED

Added in API level 30

public static final String EXTRA_PROVIDER_ENABLED

Intent extra included with PROVIDERS_CHANGED_ACTION broadcasts, containing the boolean enabled state of the location provider that has changed.

See also:

- PROVIDERS_CHANGED_ACTION
- EXTRA_PROVIDER_NAME

Constant Value: "android.location.extra.PROVIDER_ENABLED"

EXTRA_PROVIDER_NAME

Added in API level 29

public static final String EXTRA_PROVIDER_NAME

Intent extra included with PROVIDERS_CHANGED_ACTION broadcasts, containing the name of the location provider that has changed.

See also:

- PROVIDERS_CHANGED_ACTION
- EXTRA_PROVIDER_ENABLED

Constant Value: "android.location.extra.PROVIDER_NAME"
FUSED_PROVIDER
Added in API level 31
public static final String FUSED_PROVIDER
Standard name of the fused location provider.
If present, this provider may combine inputs from several other location providers to provide the best possible location fix. It is implicitly used for all requestLocationUpdates APIs that involve a Criteria.
Constant Value: "fused"
GPS_PROVIDER
Added in API level 1
public static final String GPS_PROVIDER
Standard name of the GNSS location provider.

If present, this provider determines location using GNSS satellites. The responsiveness and accuracy of location fixes may depend on GNSS signal conditions.

The extras Bundle for locations derived by this location provider may contain the following key/value pairs:

- satellites - the number of satellites used to derive the fix

Constant Value: "gps"

KEY_FLUSH_COMPLETE

Added in API level 31

public static final String KEY_FLUSH_COMPLETE

Key used for an extra holding an integer request code when location flush completion is sent using a PendingIntent.

See also:

- requestFlush(String, PendingIntent, int)

Constant Value: "flushComplete"

KEY_LOCATIONS

Added in API level 31

public static final String KEY_LOCATIONS

Key used for an extra holding a array of Locations when a location change is sent using a PendingIntent. This key will only be present if the location change includes multiple (ie, batched) locations, otherwise only KEY_LOCATION_CHANGED will be present. Use Intent#getParcelableArrayExtra(String) to retrieve the locations.

The array of locations will never be empty, and will ordered from earliest location to latest location, the same as with LocationListener#onLocationChanged(List).

See also:

- requestLocationUpdates(String, LocationRequest, PendingIntent)

Constant Value: "locations"

KEY_LOCATION_CHANGED

Added in API level 3

public static final String KEY_LOCATION_CHANGED

Key used for an extra holding a Location value when a location change is sent using a PendingIntent. If the location change includes a list of batched locations via KEY_LOCATIONS then this key will still be present, and will hold the last location in the batch. Use Intent#getParcelableExtra(String) to retrieve the location.

See also:

- requestLocationUpdates(String, LocationRequest, PendingIntent)

Constant Value: "location"

KEY_PROVIDER_ENABLED

Added in API level 3

public static final String KEY_PROVIDER_ENABLED

Key used for an extra holding a boolean enabled/disabled status value when a provider enabled/disabled event is broadcast using a PendingIntent.

See also:

- requestLocationUpdates(String, LocationRequest, PendingIntent)

Constant Value: "providerEnabled"

KEY_PROXIMITY_ENTERING

Added in API level 1

public static final String KEY_PROXIMITY_ENTERING

Key used for the Bundle extra holding a boolean indicating whether a proximity alert is entering (true) or exiting (false)..

Constant Value: "entering"

KEY_STATUS_CHANGED

Resource Manager

The job of a resource manager is, quite simply, to manage all available resources that your company has, especially employees. One of the many responsibilities of a resource manager (more commonly known as a human resource manager, or HR manager) is to assign the right people to a job.

There are many more items which you use to build a good Android application. Apart from coding for the application, you take care of various other resources like static content that your code uses, such as bitmaps, colors, layout definitions, user interface strings, animation instructions, and more. These resources are always maintained separately in various sub-directories under res/ directory of the project.

This tutorial will explain you how you can organize your application resources, specify alternative resources and access them in your applications.

Organize resource in Android Studio

```
MyProject/
  app/
    manifest/
      AndroidManifest.xml
  java/
    MyActivity.java
    res/
      drawable/
        icon.png
      layout/
        activity_main.xml
        info.xml
      values/
        strings.xml
```

Sr.No.	Directory & Resource Type
1	**anim/** XML files that define property animations. They are saved in res/anim/ folder and accessed from the **R.anim** class.
2	**color/** XML files that define a state list of colors. They are saved in res/color/ and accessed from the **R.color** class.
3	**drawable/** Image files like .png, .jpg, .gif or XML files that are compiled into bitmaps, state lists, shapes, animation drawable. They are saved in res/drawable/ and accessed from the **R.drawable** class.
4	**layout/** XML files that define a user interface layout. They are saved in res/layout/ and accessed from the **R.layout** class.
5	**menu/** XML files that define application menus, such as an Options Menu, Context Menu, or Sub Menu. They are saved in res/menu/ and accessed from the **R.menu** class.
6	**raw/** Arbitrary files to save in their raw form. You need to call *Resources.openRawResource()* with the resource ID, which is *R.raw.filename* to open such raw files.
7	**values/** XML files that contain simple values, such as strings, integers, and colors. For example, here are some filename conventions for resources you can create in this directory − • arrays.xml for resource arrays, and accessed from the **R.array** class. • integers.xml for resource integers, and accessed from the **R.integer** class. • bools.xml for resource boolean, and accessed from the **R.bool** class. • colors.xml for color values, and accessed from the **R.color** class. • dimens.xml for dimension values, and accessed from the **R.dimen** class. • strings.xml for string values, and accessed from the **R.string** class. • styles.xml for styles, and accessed from the **R.style** class.
8	**xml/** Arbitrary XML files that can be read at runtime by calling *Resources.getXML()*. You can save various configuration files here which will be used at run time.

Alternative Resources

Your application should provide alternative resources to support specific device configurations. For example, you should include alternative drawable resources (i.e.images) for different screen resolution and alternative string resources for different languages. At runtime, Android detects the current device configuration and loads the appropriate resources for your application.

To specify configuration-specific alternatives for a set of resources, follow the following steps −

- Create a new directory in res/ named in the form <resources_name>-<config_qualifier>. Here resources_name will be any of the resources mentioned in the above table, like layout, drawable etc. The qualifier will specify an individual configuration for which these resources are to be used. You can check official documentation for a complete list of qualifiers for different type of resources.
- Save the respective alternative resources in this new directory. The resource files must be named exactly the same as the default resource files as shown in the below example, but these files will have content specific to the alternative. For example though image file name will be same but for high resolution screen, its resolution will be high.

Below is an example which specifies images for a default screen and alternative images for high resolution screen.

```
MyProject/
  app/
    manifest/
      AndroidManifest.xml
  java/
    MyActivity.java
    res/
      drawable/
        icon.png
        background.png
      drawable-hdpi/
        icon.png
        background.png
      layout/
        activity_main.xml
        info.xml
      values/
        strings.xml
```

Below is another example which specifies layout for a default language and alternative layout for Arabic language.

```
MyProject/
  app/
    manifest/
      AndroidManifest.xml
  java/
    MyActivity.java
    res/
      drawable/
        icon.png
        background.png
      drawable-hdpi/
        icon.png
        background.png
      layout/
        activity_main.xml
        info.xml
      layout-ar/
        main.xml
      values/

        strings.xml
```

Accessing Resources

During your application development you will need to access defined resources either in your code, or in your layout XML files. Following section explains how to access your resources in both the scenarios −

Accessing Resources in Code

When your Android application is compiled, a R class gets generated, which contains resource IDs for all the resources available in your res/ directory. You can use R class to access that resource using sub-directory and resource name or directly resource ID.

Example

To access res/drawable/myimage.png and set an ImageView you will use following code −

```
ImageView imageView = (ImageView) findViewById(R.id.myimageview);

imageView.setImageResource(R.drawable.myimage);
```

Here first line of the code make use of R.id.myimageview to get ImageView defined with id myimageview in a Layout file. Second line of code makes use of R.drawable.myimage to get an image with name **myimage** available in drawable sub-directory under /res.

Example

Consider next example where res/values/strings.xml has following definition −

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string  name="hello">Hello, World!</string>
</resources>
```

Now you can set the text on a TextView object with ID msg using a resource ID as follows −

```
TextView msgTextView = (TextView) findViewById(R.id.msg);
msgTextView.setText(R.string.hello);
```

Example

Consider a layout res/layout/activity_main.xml with the following definition −

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="vertical" >

  <TextView android:id="@+id/text"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello, I am a TextView" />

  <Button android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello, I am a Button" />


</LinearLayout>
```

This application code will load this layout for an Activity, in the onCreate() method as follows −

```
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);
}
```

Accessing Resources in XML

Consider the following resource XML res/values/strings.xml file that includes a color resource and a string resource −

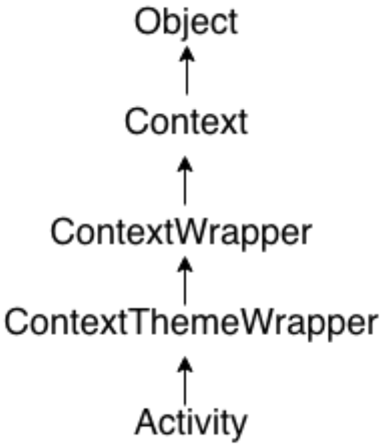
```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <color name="opaque_red">#f00</color>
   <string name="hello">Hello!</string>
</resources>
```

Now you can use these resources in the following layout file to set the text color and text string as follows −

```
<?xml version="1.0" encoding="utf-8"?>
<EditText xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:textColor="@color/opaque_red"
   android:text="@string/hello" />
```

Now if you will go through previous chapter once again where I have explained Hello World! example, and I'm sure you will have better understanding on all the concepts explained in this chapter. So I highly recommend to check previous chapter for working example and check how I have used various resources at very basic level.

Android Activity Lifecycle



Android Activity Lifecycle is controlled by 7 methods of android.app.Activity class. The android Activity is the subclass of ContextThemeWrapper class.

An activity is the single screen in android. It is like window or frame of Java.

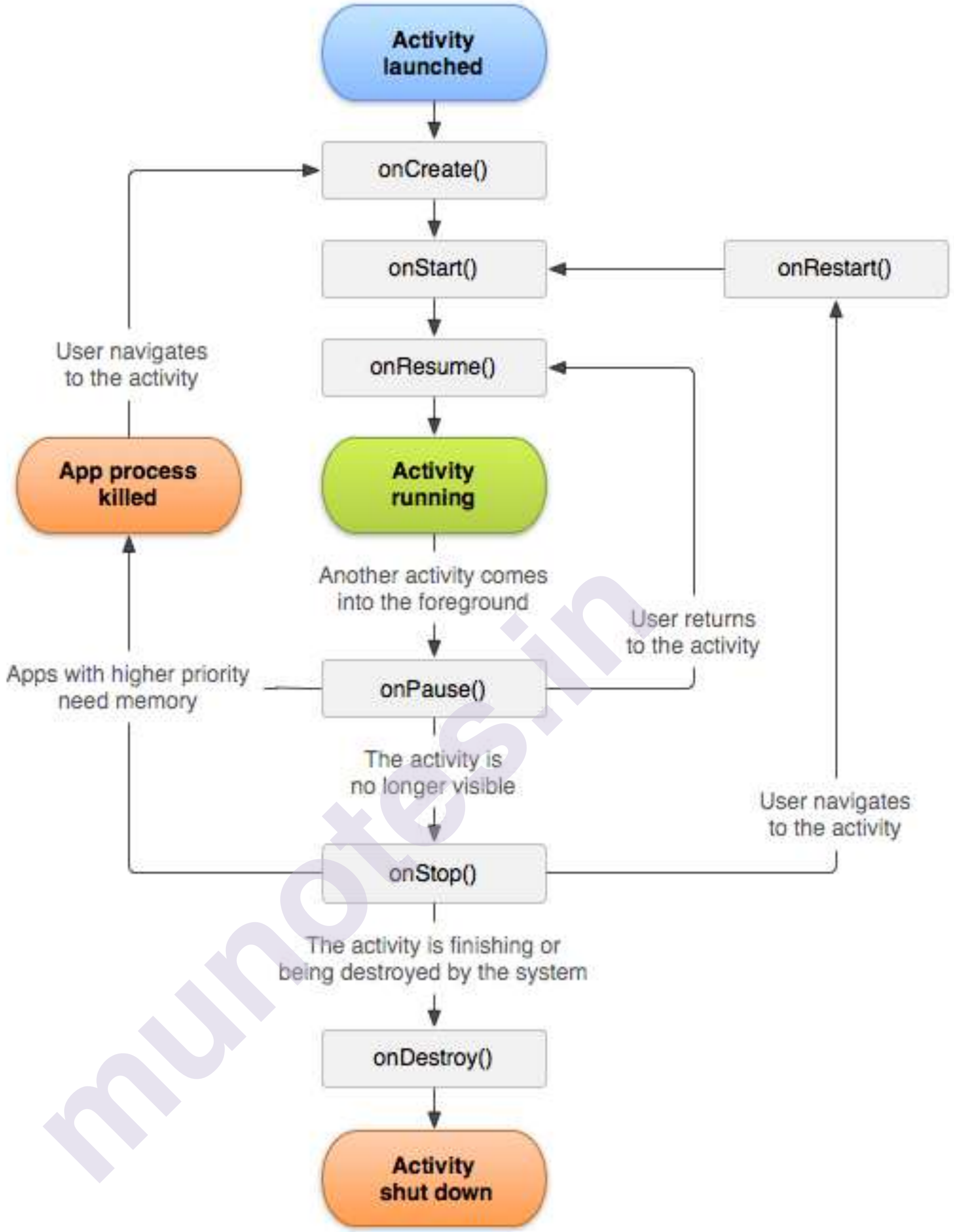
By the help of activity, you can place all your UI components or widgets in a single screen.

The 7 lifecycle method of Activity describes how activity will behave at different states.

Android Activity Lifecycle methods

Let's see the 7 lifecycle methods of android activity.

Method	Description
onCreate	called when activity is first created.
onStart	called when activity is becoming visible to the user.
onResume	called when activity will start interacting with the user.
onPause	called when activity is not visible to the user.
onStop	called when activity is no longer visible to the user.
onRestart	called after your activity is stopped, prior to start.
onDestroy	called before the activity is destroyed.



File: activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context="example.javatpoint.com.activitylifecycle.
MainActivity">

```

```
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Hello World!"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toTopOf="parent" />

</android.support.constraint.ConstraintLayout>
```

Android Activity Lifecycle Example

It provides the details about the invocation of life cycle methods of activity. In this example, we are displaying the content on the logcat.

File: MainActivity.java

```
package example.javatpoint.com.activitylifecycle;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;

public class MainActivity extends Activity {

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
Log.d("lifecycle","onCreate invoked");
}
@Override
protected void onStart() {
super.onStart();
Log.d("lifecycle","onStart invoked");
}
@Override
protected void onResume() {
super.onResume();
Log.d("lifecycle","onResume invoked");
}
@Override
protected void onPause() {
super.onPause();
Log.d("lifecycle","onPause invoked");
```

```
}
@Override
protected void onStop() {
super.onStop();
Log.d("lifecycle","onStop invoked");
}
@Override
protected void onRestart() {
super.onRestart();
Log.d("lifecycle","onRestart invoked");
}
@Override
protected void onDestroy() {
super.onDestroy();
Log.d("lifecycle","onDestroy invoked");
}
}
```

Output:

You will not see any output on the emulator or device. You need to open logcat.

4.5 SUMMARY

This chapter Describes basic things of Android.

4.6 EXERCISE

- Explain the Android application Architecture.
- What are the code names of android?

4.7 REFERENCES

- PROFESSIONAL Android™ 4 Application Development, Reto Meier, John Wiley & Sons, Inc. 2012.

❄❄❄❄❄❄❄