C

Q 1. Re-entrant code, Re-entrant function, Example

- ➔ A function is re-entrant if it can be invoked while already in the process of executing.
- ➔ A function is re-entrant if it can be interrupted in the middle of the execution (eg. By a signal or interrupt) and invoked again before the interrupted execution complets.
- ➔ Can be used by more than one task without fear or data corruption.
- ➔ Can be interrupted at any time and resume at a later time without loss of data.
- ➔ A re-entrant function call satidfy the following condition:
- ➔ 1. Should not call another non-reentrant function.
- ➔ 2. Should not contain static time variables.
- ➔ 3. Should not use any global variables.
- ➔ 4. Should not use any shared resources.
- ➔ Function is non-reentrant if it breaks one or more condition of reentrancy conditions.
- ➔ Non-reentrant function can not be shared by more than one task unless mutual exclusion to the function is ensured by either the a semaphore by disabling interrupts during critical sections of the code.
- ➔ Non-reentrant functions cannot be used by multiple threads.
- ➔ It is impossible to make a non-reentrant function thread-safe.

Non-Reentrant Function	Reentrant Function

Eg.
 	
static int sum=0;
 	
int increment(int i)
{
  sum += i;
  return sum;
}
	Eg.
	
	int increment(int sum, int i)
	{
	  return sum + i;
	}

---

2. malloc and free function using linkedlist

3. How -ve number stored, How 2's compliment calculate

4. Count number of set bits in an integer

5. Find loop in linkedlist

6. How to determine direction of stack growth

7. NULL pointer, Difference between NULL & NUL

8. If you free pointer twice ?

9. Near , Far and Huge pointer. When should we use ?

10. Pointer to a function. Its use

11. Pointer substraction is possible ?

12. Difference between pointer and array

13. Difference between signed and unsigned variable. Difference between signed and unsigned char. How negative number are stored.

14. How static variable retain its value between the function call. What r static function and its use

15. Dagling pointer

16. use of pointer in c program

17. what happen with sign bit and vacated bit if the sign number is right shifted.

18. List various section in program object files

19. What happen when the function is calle in (W.r.t. Stack)

20. Typedef

Data Structure

Q 1. Difference between linear array & Linked List

Parameter	Linear Array	Linked List
1. Define	Collection of elements having same data types. It is linear data structure	Collection of element of different data types. It is linear or non-linear data structure.
2. Access	Element can be access using index/subscript value i.e. element can be randomly accessed like a[0],a[10] etc.	Element can be accessed sequentially only and it will take time O(n) time.
3. Memory Structure	Elements are stored consecutive manner in memory	Elements are stored at any available place as address of node stored in previous node
4. Insertion and Deletion	It will take more time, because elements are stored in consecutive memory locations	It is easy and fast as only value of pointer are needed to change
5. Memory allocation	Compile time allocation i.e. Static memory allocation	Tun time allocation i.e Dynamic memory allocation
6. Types	It can be 1. Single Dimensional 2. Two Dimensional 3. Three Dimensional or 4. Multidimensional	It can be 1. Singly linked list 2. Doubly linked list 3. Circular linked list
7. Dependency	Each element is independent, no connection with previous element or location	Location or address of elements are stored in link part of previous element/node
8. Space	Space is wasted	Space is not wasted
9. Time	Same amount of time required to access each element	Different amount of time required to access each element.
10. Modification of Elements	Elements can be modified easily by identifying the index value only.	Complex process for modifing the nodes

Linux

Q 1. Dynamic memory partitioning in paging

- ➔ **Memory problem aries in the continuous memory allocation**
- ➔ **So that we use the concept of dynamic memory allocation**
- ➔ **In dynamic memory allocation memory is divided into the various partition at run time**

Q 2. Boot Process

- ➔ **To run a program you need the operating system loaded into main memory [RAM] and running**
- ➔ **There are two form of RAM 1] SRAM 2] DRAM**
- ➔ **SRAM is built out of transistors where several transistor make up each cell (Storage location) , one cell store one bit.**
- ➔ **DRAM made up of one transistor and one capacitor per storage location (bit).**
- ➔ **DRAM is smaller and cheaper, which result in a greater amount of DRAM storage available than SRAM.**
- ➔ **SRAM is used to build cache memories and registers while DRAM makes up what often refer to as "Main memory".**

Type	Typical Amount	Relative Expense	Usages
DRAM	4-16 GB	Very cheap	Main Memory: Stores running program code and data, graphics
SRAM	1-2MB	Moderately Expensive	Stores recently and currently used portions of program code and data
ROM	4K or Less	Very Expensive	Stores unchanging information: the boot program, basic IO device driver, microcode

- ➔ **Upon rebooting the memory becomes the empty and thus we need to locate and load the operating system.**
- ➔ **We need some initialization program that can, upon starting the computer, locate and load the operating system into DRAM.**
- ➔ **This initialization process is called Booting (Taken from the term bootstrapping).**
- ➔ **The process is begins whenever a computer is cold booted [Turn on] or soft booted [Rebooted from Software]**
- ➔ **For any computer, the first step in the boot process is to access the ROM BIOS [Basic IO Sytem]**
- ➔ **In Linux the boot program start at ROM BIOS address 0xFFFFF0.**
- ➔ **The first task for the BIOS is to perform a Power On Self Test [POST], which examines various piece of hardware connected to the computer to ensure that they are working properly or not.**
- ➔ **The POST the CPU registers, main memory, hardware devices such as the interrupt controller, disk controller and timer, and then identifies all devices currently connected via systum bus (namely the keyboard, mouse ans monitor).**
- ➔ **The POST step may be skipped during a reboot (warm boot) as these devices are already on and funvtining.**
- ➔ **Loading the operating system may start automatically. The booting may be from bootable devices.**
- ➔ **The BIOS is not alterable , because it is stored in ROM, computer come with another form of memory, CMOS, which runs off the battery placed on the motherboard.**
- ➔ **The CMOS may be store other alterable boot information and include a clock of the current date and time.**

Q 3. Boot Loading in Linux

- ➔ **The typical situation in Linux to boot the OS from the internal hard disk. We will assume this case here.**
- ➔ **The very first sector of the Linux disk space is set aside for MBR [Master Boot Record].**

- ➔ **The MBR contains the boot loader program as well as partion table and a magic number.**
- ➔ **The magic number is used for a validation check only.**
- ➔ **The partition table contains the location on disk of active partitions where bootable operating system can be found.**
- ➔ **The MBR is of 512 bytes in length.**
- ➔ **The first part of MBR consist of 446 bytes that contain part of the bootloader program.**
- ➔ **For Linux OS two commanly used bootloaders 1] GRUB (Grand Unifield Bootloader) 2] LILO (Linux Loader)**
- ➔ **GRUB which can be use to load multiple types of operating system [Eg. Windows or Linux or two versions of Linux etc]**
- ➔ **LILO is use to load the multiple types of Linux operating system [Eg. CentOS and Ubuntu].**
- ➔ **These two bootloader programs are too large to fit in the 446 bytes available, so the boot loaders are divided into two parts [generally Installer and Kernel Loader], also referred to as Stage 1 and Stage 2 of the booting process.**
- ➔ **Stage 1 merely [only] finds and load stage 2 into memory.**
- ➔ **Stage 2 is responsible for finding and loading the operating system kernel.**
- ➔ **LILO is the older of the two boot loaders. It can boot an OS from either floppy or hard disk and it can select any of up to 16 different boot images.**
- ➔ **LILO is the file independent, which is both weakness and a strength [in that LILO is simpler and does not have reliance (dependance) on accessing the file system].**
- ➔ **You can alter LILO behavior [Once Linux has booted] by editing the files etc/lilo.conf.**
- ➔ **This will change how LILO works in future boots.**
- ➔ **This configuration file stores global options, such as boot location , type of installation (Eg. Menu of text-based) and location of file system mapping information for each of the boot images.**
- ➔ **GRUB can operate in two or three stages (Middle stage called stage 1.5)**
- ➔ **Stage 1.5 contain device driver so that GRUB can access different types of file systems.**
- ➔ **Because of this, GRUB can actually interact with the Linux file system prior to the kernel being run**

- ➔ **LILO on the other hand must perform the kernel loading operation without any access to information that might be stored in the file system.**
- ➔ **With this copability, GRUB is able to load stage 2 directly from the file system rather than the reserved location on the hard disk.**
- ➔ **GRUB stage 2 loads the default configuration file in order to present to the user a seletion of kernel to select between.**
- ➔ **Once kernel is selected, GRUB uses a mapper to locate the kernel and load it into memory.**
- ➔ **The third bootloader is called loadin.**
- ➔ **This boot loader runs under either DOS or Windows.**
- ➔ **This means you are able to boot to Linux from DOS/Windows as opposed to the more traditional bootloaders, which runs during the boot process.**
- ➔ **The loadlin replace the image in memory of DOS/Windows with the linux kernel and configuration parameters while it executes.**
- ➔ **So loadline is more useful if you desire booting to linux occasionally from a fully booted windows machine**
- ➔ **With access to the MBR and Boot loader program, the boot process has shifted from running program code to from ROM to running code from hard disk.**

#################### Loading the Linux Kernel ############################

- ➔ **The linux kernel is stored as a partially compressed image.**
- ➔ **What is loaded into memory is not entirely executable.**
- ➔ **The linux kernel is stored on hard disk as the file vmlinuz.**
- ➔ **The first portion of the kernel image is executable and it contain two parts, a 512-bytes boot sector and a kernel setup program.**
- ➔ **As the kernel setup portion runs, it perform some basic initial hardware setup.**
- ➔ **It then compress the letter portion of the vmlinuz file into vmlinux, the rest of the kernel.**

Boot Sector	Setup Sector	Compressed Kernel Image

		[Vmlinux]

Fig : vmlinuz file

- ➔ **The linux kernel can be stored on a USB drive or optical disk and inserted prior to the boot process as vmlinuz is bootable from these sources.**
- ➔ **With the kernel, vmlinux, uncompressed, kernel initialization begins.**
- ➔ **Its first step to continue initialization of hardware.**
- ➔ **Among other things, the power system tests the various components, searches for load ramdisk and also setting up interrupt handling mechanisms [IRQs].**
- ➔ **One of the ramdisksis loaded with initramfs, which is the initial linux root file system.**
- ➔ **This file transferred into RAM for use during kernel initialization for efficeient access.**
- ➔ **The initramfs file system allows the kernel to access a file system without having to mount any part of the normal linux system, thus permitting the kernel to operate using file operation.**
- ➔ **This file is only stored temporarily and removed from RAMdisk midway through kernel initialization.**
- ➔ **The initramfs contains directories that mirror the permanent linux file system:**
  bin, dev, etc, lib, loopfs, proc, sbin, sys and sysroot
- ➔ **Prior to Linux 2.6.13 , this initial file system was called initrd.**
- ➔ **One difference between initramfs and initrd is the initrd had to be stored via a block storage device where as initramfs is more flexible.**
- ➔ **Unlike the matching directories in the permanent Linux file system, the initramfs directories are minimal, containing thoes executables and configuration file needed to finalize the kernel initialization process.**
- ➔ **The /dev directory for instance parmits the kernel to communicate with hardware device during hardware initialization while /bin and /sbin contain the executable programs necessary for the kernel to start virtual memory and mount the root partition.**
- ➔ **At this point in time, the kernel executes a commnads called pivot_root, which alters the root partition from initrd to /.**

- ➔ **This dismisses initramfs (remove it from memory), and establishes the permanent file system.**
- ➔ **At this point only the root file system is mounted.**
- ➔ **The root partition will contain /bin, /sbin, and /etc. These directories will all be needed during the reminder of operating system initialization.**
- ➔ **Later, the other file systems in /etc/fstab (eg. /var , /home) will be mounted.**
- ➔ **To complete kernel execution, the kernel will find and execute the init process (usually located at /sbin/init) .**
- ➔ **It is the init process that is responsible for initializing and establishing the user space.**
- ➔ **In essence, init is responsible for bringing the system up to a usable state, doing everything that the kernel initialization did not.**
- ➔ **The init program is the first process to execute in any Linux session.**
- ➔ **This is the permanent process in that it will run during the entire session that Linux is running, but it will largely sit in the background.**
- ➔ **It is only stopped when Linux is shutdown.**
- ➔ **The init process is also if the telinit command is used [Which cause Linux to switch to different run-level, and a new init process is started].**
- ➔ **It is moved into forground as needed.**
- ➔ **For instance, init moves to the forground to adopt an orphaned process, and then it moves back into the background. [Init pid is always 1]**
- ➔ **The inittab allows us to define a number of situations whereby processes are invoked without changing the default runlevel.**
- ➔ **Linux has 7 different runlevels.**
- ➔ **These levels dictate [order] which services should start and which should not.**

Runlevel	Name	Comman Usage
0	Halt	Shuts down the system, not used in inittab as it would immediately shut down on initialization
1	Single-user mode	Useful for administrative tasks including unmounting partitions and reinstalling the portions of the OS; when used, only root access is available.
2	Multi-user mode	In this, Linux allows users other than root to log in. In this

		case network services are not started so that the user is limited to access via the console only.
3	Multi-user mode with Networking	Commanly used mode for servers or systems that do not require graphical user interface
4	Not-used	For special/undefined purpose
5	Multiuser mode with networking and GUI	Most common mode for a Linux workstation.
6	Reboot	Reboot the system; not used in inittab because it would reboot repeatedly

- The most common runlevel is either 5 for most workstations or 3 for some servers [Which arre not expected to be used as a regular workstation but instead logged into remotely by system and server administrators].
- The telinit instruction allows you to change runlevels from the command line.
- So, if you are currently in runlevel 5 and wish to switch to runlevel 1, telinit 1 will accomplish this. However, in switching from 5 to 1, you must kill any service that is started by runlevel 5 but which should not be running for runlevel 1.
- Similarly, exiting from runlevel 1 back to runlevel 5 will require restarting those services.
- The command dmesg will respond with the kernel ring buffer.
- In Linux, a service is generally referred to as a daemon (pronounced “demon”).
- A service is a piece of operating system code used to handle some type of request.
- The service has several distinctive features.
- First, it runs in the background so that it does not take up processor time unless called upon.
- Second, services can handle requests that come from many different sources: users, applications software, hardware, other operating system services, messages from the network.
- Third, services are configurable.
- Configuration is usually handled through configuration files, which are often stored in the /etc directory (or a subdirectory).

➔ There are usually four different things you can do to a service: start, stop, restart, or obtain the service's status.

Q 4. Linux File System

➔ The domain socket is used to open communication between two local processes

➔ This permits interprocess communication (IPC) so that the two processes can share data.

➔ We might, for instance, want to use IPC when one process is producing data that another process is to consume.

➔ There are several distinctions between a network and domain socket.

➔ The network socket is not treated as a file (although the network itself is a device that can interact via file system commands) while the domain socket is.

➔ The network socket is created by the operating system to maintain communication with a remote computer while domain sockets are created by users or running software.

➔ Network sockets provide communication lines between computers rather than between processes.

Inode:

✓ When the file system is first established, it comes with a set number of inodes.

✓ The inode is a data structure used to store file information.

✓ The information that every inode will store consists of

✓ The file type

✓ The file's permissions

✓ The file's owner and group

✓ The file's size

✓ The inode number

✓ A timestamp indicating when the inode was last modified, when the file was created, and when the file was last accessed

✓ A set number of pointers that point directly to blocks

- ✓ **A set number of pointers that point to indirect blocks; each indirect block contains pointers that point directly to blocks**
- ✓ **A set number of pointers that point to doubly indirect blocks, which are blocks that have pointers that point to additional indirect blocks**
- ✓ **A set number of pointers that point to doubly indirect blocks, which are blocks that have pointers that point to additional indirect blocks**
- ✓ **A set number of pointers that point to triply indirect blocks, which are blocks that have pointers that point to additional doubly indirect blocks**

- ✓ **Typically, an inode will contain 15 pointers broken down as follows:**
- ✓ • **12 direct pointers**
- ✓ • **1 indirect pointer**
- ✓ • **1 double indirect pointer**
- ✓ • **1 triply indirect pointer**
- ✓ **Typically, there is 1 inode for every 2–8 KB of file system space.**
- ✓ **If we have a 1 TB file system (a common size for a hard disk today), we might have as many as 128 K (approximately 128 thousand) inodes.**

Linux Commands to inspect inode and files:

- ✓ **stat—provides details on specific file usage, the option –c %i displays the file's inode number**
- ✓ **ls—the option –i displays the inodes of all entries in the directory**
- ✓ **df –i—this command provides information on the utilization of the file system, partition by partition. The -i option includes details on the number of inodes used.**
- ✓ **The opening of a file requires a special designator known as the file descriptor.**
- ✓ **The file descriptor is an integer assigned to the file while it is open.**
- ✓ • **0 – stdin**
- ✓ • **1 – stdout**
- ✓ • **2 – stderr**
- ✓ **When a file is to be opened, the operating system kernel gets involved.**
- ✓ **First, it determines if the user has adequate [Sufficient] access rights to the file.**

- ✓ **If so, it then generates a file descriptor.**
- ✓ **It then creates an entry in the system's file table, a data structure that stores file pointers for every open file.**
- ✓ **The location of this pointer in the file table is equal to the file descriptor generated.**
- ✓ **For instance, if the file is given the descriptor 185, then the file's pointer will be the 185th entry in the file table.**
- ✓ **The pointer itself will point to an inode for the given file.**
- ✓ **As devices are treated as files, file descriptors will also exist for every device, entities such as the keyboard, terminal windows, the monitor, the network interface(s), the disk drives, as well as the open files.**
- ✓ **There are many reasons to partition the file space. First, because each partition is independent of the others, we can perform a disk operation on one partition, which would not impact the availability of the others.**
- ✓ **Most of the entries are stored on a SATA [Standard Advanced Technology Attatchment] hard disk, which is indicated as /dev/sda#.**
- ✓ **What is blocking IO ?**
- ✓ **Two access the data file there are two options ro vs rw [read only, write only], and sync/Async.**
- ✓ **In the former case, the ro option means that data files on this partition can only be read.**
- ✓ **We might use this option if the files are all executable programs, as found in /bin and /sbin.**
- ✓ **Partitions with data files such as /var and /home would be rw.**
- ✓ **The latter option indicates whether files have to be accessed in a synchronized way or not.**
- ✓ **With sync, any read or write operation must be completed before the process moves on to the next step.**
- ✓ **This is sometimes referred to as blocking I/O because the process is blocked from continuing until I/O has completed.**
- ✓ **In asynchronous I/O, the process issues the I/O command and then continues on without stopping.**

- ✓ **bashrc—the file that executes whenever a new bash shell opens; edited by the system administrator**
- ✓ **init, inittab, init.d/—the startup service, a configuration file storing the default run level, and a directory containing all system service control scripts.**
- ✓ **The /var directory contains system and software data.**
- ✓ **The /usr directory stores application software and supporting files.**
- ✓ **Aside from the binary files (executables), the directories contain documentation files (under/usr/local/man), library files, C header and obje files, and programs that support the X windows system.**
- ✓ **The src subdirectory stores Linux kernel source files (as well as header files and documentation).**
- ✓ **The /var/log subdirectory stores log files that are automatically generated by the kernel and running software. The /var/mail and /var/spool directories store the mail files and print files, respectively.**
- ✓ **Finally, the /var/www directory is often used if your computer is running a web server.**
- ✓ **/root—the system administrator's home directory.**
- ✓ **/sbin—system administration binary files (commands, programs).**
- ✓ **/usr—application software and other common programs that are not found under /bin and /sbin**
- ✓ **/etc—stores system configuration files; system administrators will often use the files in this directory.**
- ✓ **/var—system data files that grow over time such as log files, email files, and print spooler files.**
- ✓ **/proc—stored in memory rather than on the file system, this directory stores information about all running processes.**
- ✓ **/dev—directory storing interfaces to most of the available devices (both physical like hard disk, optical disk, modem and logical like terminal windows (tty), programs like random and zero, and ramdisks).**
- ✓ **/bin—location of common binary files (Linux commands and programs).**
- ✓ **/boot—location of boot loader program (e.g., GRUB) and Linux kernel, required for booting Linux.**

- ✓ **tar—tape archive, historically used to perform backup to tape but today is most commonly used to create archives of files and directories.**
- ✓ **Pointer—an indicator of where a disk block is located.**
- ✓ **Partition—a logical division of the file system to protect the contents from other partitions.**
- ✓ **Network file system (nfs)—a form of file system that permits mounting of partitions over the network.**
- ✓ **Mount point—the logical location of a mounted partition, this will be some directory such as /opt, /mnt, or /usr/local/mountpoint.**
- ✓ **Mounting—making a partition available.**
- ✓ **Mount options—control access to the partition such as making it readonly (ro) or read/write (rw), synchronous (sync) or asynchronous (nosync) and permitting any- one to mount the partition (user) or not (nouser), among others.**
- ✓ **Logical volume manager (LVM)—a software means of partition management so that partition sizes can be changed without requiring direct changes to the file system itself; this makes partition management safer and easier.**
- ✓ **inode—a data structure storing information about a specific file including pointers to its blocks or indirect blocks, creation/modification/access information, permissions, ownership, file type, and device number; any Linux file system contains a set number of inodes.**
- ✓ **Indirect block—inodes come with several direct pointers to the first group of disk blocks for the file; the remainder of the disk blocks are pointed to by pointers in indirect blocks; the inode has pointers to indirect blocks, doubly indirect blocks and triply indirect blocks.**
- ✓ **ext (extended file system type)—family of file systems supported by Linux; ext is not used but ext2, ext3, and ext4 are all common (moreso ext4 today).**
- ✓ **FAT (file allocation table)—used in older Windows operating systems to store the disk block layout so that obtaining the ith block of a file can be easily determined without having to perform i-1 disk accesses.**
- ✓ **Domain socket—a mechanism to support interprocess communication; denoted by ‘s’ in a long listing.**

- ✓ **Block device—type of device, denoted by type 'b' in a long listing, that performs input/output on blocks (rather than characters); most storage devices are block devices.**
- ✓ **Character device—type of device, denoted by type 'c' in a long listing, that performs input/output on characters (rather than blocks); keyboard and mouse are examples of character devices.**

Q 5. Network Configuration

- ✓ **Address resolution—the process of converting an IP alias into an IP address using some resolver such as a DNS server, a DNS cache, or the entries stored in /etc/ hosts.**
- ✓ **Computer network—a collection of computers and computing resources connected together to facilitate communication between resources.**
- ✓ **Domain name system—the collection of servers and resolution information that permits the use of IP aliases on the Internet rather than IP addresses. The DNS includes DNS servers, caches, and local resolving programs.**
- ✓ **Dynamic IP address—an IP address issued to your computer temporarily (for instance, for a few days).**
- ✓ **Ethernet—a technology for local area networks.**
- ✓ **Firewall—software that helps enforce security. In some cases, a firewall is both hardware and software if an organization dedicates a computer to the server solely as a firewall.**
- ✓ **Gateway—a broadcast device responsible for connecting local area networks of different types together.**
- ✓ **Hub—a broadcast device operating on a subnetwork that, when it receives a message, broadcasts that message to all devices on that subnet.**
- ✓ **IP address—a unique address (number) assigned to a computing resource on the Internet. There are two types of IP addresses, version 4 (IPv4) and version 6 (Ipv6).**
- ✓ **IPv4 address—32-bit address usually written as four octets of numbers between 0 and 255, separated by periods, as in 10.11.12.13.**
- ✓ **IPv6 address—a 128-bit address offering far greater range of addresses. Usually written as 32 hexadecimal digits. IPv6 is a protocol created to replace IPv4 because IPv4 is outmoded and because we have run out of most available IPv4 addresses. IPv6**

includes features such as security and autoconfiguration that are not directly available in Ipv4.

- ✓ **Loopback device—an interface in Linux machines that allows software to communicate to the computer as if the messages were coming over the network. The loopback device does not send messages onto a network.**
- ✓ **MAC address—the media access control address given to devices such as Ethernet cards. This address is used at the lowest level of the TCP/IP protocol and is used by switches.**
- ✓ **Router—a broadcast device that examines a message's destination IP address and routes the message onto the proper network or subnetwork as the next link in the chain of the communication.**
- ✓ **Static IP address—an IP address assigned to a computing resource permanently or at least for a long period of time. The static address is not expected to change. Changing it will require modifying DNS tables.**
- ✓ **Switch—a broadcast device operating on a subnetwork; when it receives a message, it broadcasts that message to a single device on the subnetwork using an MAC address for addressing.**
- ✓ **TCP/IP—a commonly used network protocol that lets computers access the Internet. TCP/IP is known as a protocol stack, comprising several lesser protocols.**
- ✓ **ifconfig—older network command to configure or obtain network information such as IP address and router address.**
- ✓ **ip—newer network command that encapsulates the operations available in lesser programs such as ifconfig, route, and iptunnel.**
- ✓ **netstat—older network command to output statistics about network usage. Has been superseded with ss.**
- ✓ **ping—program to constantly send messages to another network-based resource to test for its availability.**
- ✓ **route—displays local router tables. Command replaced by ip.**
- ✓ **ss—socket investigation program.**
- ✓ **sshd—service that permits ssh access into your computer.**
- ✓ **/etc/sysconfig/iptables—stores the Linux IPv4 firewall rules.**

Q 6. Linux Interrupts

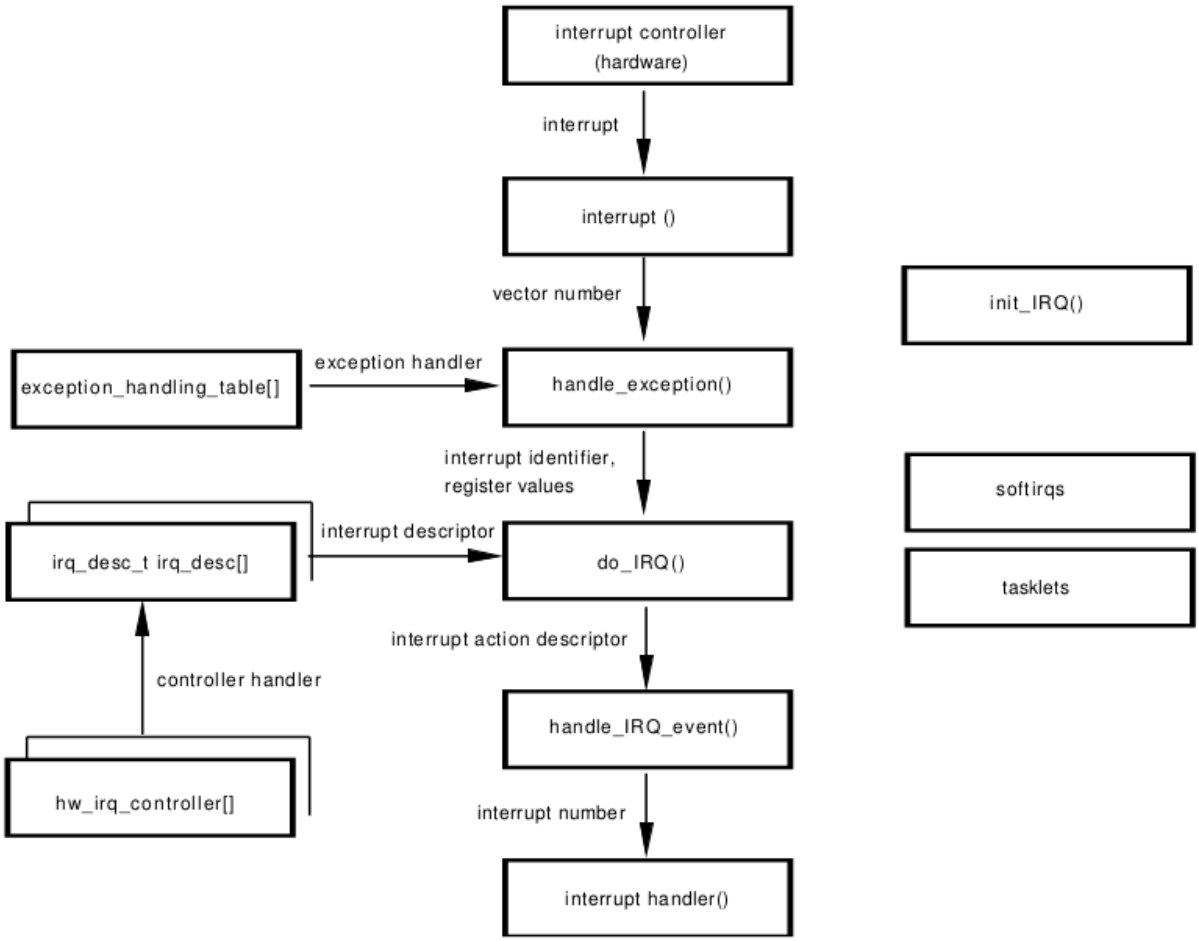


Figure 1. Linux interrupts, a pictoral representation.

Q 60. Steps in Linux Kernel Compilation

- ✓ For most part, you don't need to compile the kernel, as it is installed by default when you install the OS.
- ✓ However you might encounter certain situation, where you may have to compile kernel from source.
- ✓ The following are few situation where you may have to compile Kernel on your Linux system.
- ✓ 1. To enable experimental features that are not part of the default kernel.
- ✓ 2. To enable support for a new hardware that is not currently supported by the default kernel.
- ✓ 3. To debug the kernel
- ✓ 4. Or, just to learn how kernel works, you might want to explore the kernel source code, and compile it on your own.
- ✓ Also, please note that if you just want to compile a driver, you don't need to compile the kernel. You need only the linux-headers package of the kernel.
- ✓ Steps for kernel compilation:
- ✓ **1. Download the Latest Stable Kernel :** The first step is to download the latest stable kernel from kernel.org.
- ✓ **2. Untar the Kernel Source :** The second step is to untar the kernel source file for compilation.
- ✓ $ tar -xvJf linux-3.9.3.tar.xz
- ✓ **3. Configure the Kernel**
- ✓ $ cd linux-3.9.3
- ✓ $ make menuconfig
- ✓ The make menuconfig, will launch a text-based user interface with default configuration options. You should have installed "libncurses and libncurses-devel" packages for this command to work.We will use the default config provided by the kernel. So select "Save" and save the config in the file name ".config".
- ✓ **4. Compile the Linux Kernel**
- ✓ Compile the main kernel: **$ make**
- ✓ Compile the kernel modules: $ make modules
- ✓ Install the kernel modules: $ make modules_install
- ✓ **5. Install the New Kernel**

$ make install

✓ The make install command will create the following files in the /boot directory.

vmlinuz-3.9.3 – The actual kernel

System.map-3.9.3 – The symbols exported by the kernel

initrd.img-3.9.3 – initrd image temporary root file system used during boot process

config-3.9.3 – The kernel configuration file

✓ The command "make install" will also update the grub.cfg by default. So we don't need to manually edit the grub.cfg file.

✓ **6. Boot Linux to the new Kernel**

✓ **$ reboot**

✓ Since, in grub.cfg, the new kernel is added as default boot, the system will boot from the new kernel. Just in case if you have problems with the new kernel, you can select the old kernel from the grub menu during boot and you can use your system as usual.

✓ Once the system is up, use uname command to verify that the new version of Linux kernel is installed.

✓ **$ uname -r**

Q 3. How you will make interrupt handler as fast as possible ? What we dont do for interrupt handler ? How you allocate memory in interrupt handler ?

✓ **Interrupt enables hardware to signal a processor.**

✓ **The function in which kernel runs in responce to a specific interrupt is called an interrupt handler or interrupt service routine [ISR].**

✓ **Each device that generate interrupts has an associated interrupt handler.**

✓ **One of the major problem with interrupt handling is how to perform lengthy tasks within handler.**

✓ **The interrupt handler for a device is a part of the device's driver (The kernel code that manages the device).**

✓ **In linux interrupt handler are norman C function, but interrupt handler always run in special context called Interrupt Context.**

- ✓ **This special context occasionally called Aomic context, because code running to this context is unable to block.**
- ✓
- ✓ **Because interrupt can occur at any time, an interrupt handler can be executed at any time.**
- ✓ **It is essential that the handler run quickly, to resume execution of the interrupted code as soon as possible.**
- ✓ **It is important that:**
- ✓ **1. To the hardware: The operating system services the interrupt without delay**
- ✓ **2. To the rest of the system : The interrupt handler executes in as ahort period as possible.**
- ✓ **One of the main problems with interrupt handling is how to perform lengthy tasks within handler, often a substantial amount of work must be done in responce to a device interrupt, but interrupt handlers need to finish up quickly and not keep interrupts blocked for long.**
- ✓ **These two needs (work and speed) conflict with each other, leaving the driver writer in a bit of blind.**
- ✓ **Linux resolve these problem by splitting the interrupt handler into two halves.**
- ✓ **1. Top half**
- ✓ **2. Bottom half**
- ✓ **Top Half :** The interrupt handler is the top half . The top half is run immediately upon receipt of the interrupt and perform only the work that is time-critical, such as acknowledging receipt of the interrupt or reseting the hardware. The processing done in interrupt context is called as Top half. It is actually respond to the interrupt i.e the one you register with request_irq.
- ✓ **Bottom Half:** It is the routine that is scheduled by the top half to be executed later, art a safer time. Work that can performed later is deferred until the bottom half. The bottom half runs in the future, at a more convenient time, with all interrupt enabled.
- ✓ The big difference between the top-half handler and bottom half is that all interrupts are enabled during execution of the bottom half, thats why it runs at safer time.
- ✓ In a typical scenario, the top half saves device data to a device specific buffer, schedule its bottom half and exits. This operation is very fast. The bottom half then perform whatever other

work is required, such as awakening the process, starting up another IO operation, and so on. This setup permits the top half to service the new interrupt while the bottom half is still working.

- ✓ The linux kernel has to different mechanism that may be used to implement bottom-half processing.
- ✓ 1. Tasklet: Tasklet are often the preferred mechanism for bottom-half processing , they are very fast, but all tasklet code must be atomic .
- ✓ 2. Workqueues: The alternative to tasklet is workqueues, which may have a higher latency but that are allowed to sleep.

Q 4. How can you insert a module in run time ? How many ways ? Difference between insmod and modprobe. How modprobe works ?

- ✓ Dynamically we can insert module during run time.
- ✓ Two ways to insert the module dynamically,
- ✓ **a. Insmod**
- ✓ **b. Modprobe**
- ✓ **a. Insmod:** Following are steps to insert the module using Insmod
- ✓ 1. Write a simple module program, module programming is also called as kernel programming, For module programming no main function is required.
- ✓ Start -> init() function
- ✓ End -> exit() function
- ✓ 2. lsmod -> It will shows the list of module that loaded already.
- ✓ 3. insmod -> Insert the module into the kernel.
- ✓ Eg. Sudo insmod hello.ko
- ✓ 4. modinfo -> It will display information about the kernel module.
- ✓ Eg. Modinfo hello.ko
- ✓ 5. Dmesg -> It will display kernel ring buffer information
- ✓ It will display whatever you added the printk() in module programming, that information.
- ✓ 6. rmmod -> To remove the module
- ✓ Eg. Sudo rmmod hello

- ✓ When a module is inseted into the kernel, the module_init macro will be invoked, which will call the function hello_init().
- ✓ Similarly, when the module is removed with rmmod, module_exit() function will be invoked, which will call hello_exit.
- ✓ Using dmesg command, we can see th output from the kernel module.
- ✓ **b. Modprobe**
- ✓ **Modprobe utility is used to add loadable modules to the kernel.**
- ✓ **You can also add and remove module using modprobe copmmand.**
- ✓ **Modprobe -l : Display all available module.**
- ✓ **Lsmod : currently loaded module**
- ✓ **modprobe <module_name> : Install new module**
- ✓ **modprobe -r <module_name>: Remove a module**
- ✓ **We can also use rmmod , to delete module. But admins prefer modprobe with -r option, because**
- ✓ **modprobe is clever than insmod and rmmod.**
- ✓ **While we are installing the module using modprobe, first it will check the dependencies in /lib/modules/<kernel-version>/modules.dep**
- ✓ **This file contain entire file modules & its corresponding dependencies.**
- ✓ **Modules.dep file is generated by "depmod" command.**

---

5. What happen when interrupt raised by device ? Interrupt mechanism in Linux .

Q 6. Difference between softirq and tasklet

Softirq	Tasklet	Workqueues
A set of statically defined bottom halves	Flexible, dynamically created bottom halves built on top of softirqs	
Guaranteed to run on the CPU it was scheduled on	Dont have guaranty to run on the CPU that scheduled on	
Same softirq can run on two seperate CPU at a same time	Same tasklet can not run on two seperate CPU at same time . But two different tasklet can run on	Can run on different CPU core simultaneously

	two different CPU at a same time, just not the same one	
Re-entrant	Non-reentrant	
Can be created or destroyed ststically only	Can be created or destroyed either statically or dynamically	
Deferred work runs in interrupt context	Deferred work runs in interrupt context	Deferred work runs in process context
Runs in interrupt context , they cant sleep	Runs in interrupt context , they cant sleep	Runs in process context, they can sleep
Cant be preempted or scheduled	Cant be preempted or scheduled	Can be preempted or scheduled
Not easy to use	Easy to use	Easy to use

Q 7. What are tasklet ? How they activated ? When and how they initialized ? When you will use it and how it is different to ISR ?

- ✓ **Tasklet:**
- ✓ **Tasklet are bottom halves that runs in kernel context.**
- ✓ **They are suitable for long running and non-blocking tasks, and allow in contrast to softirq a dynamic allocation of new handlers.**
- ✓ **Internally tasklets are implemented by means of softirqs.**
- ✓ **Tasklet are the special function that may be scheduled to run in interrupt context, at a system determined safer time.**
- ✓ **They may be scheduled to run multiple times, but run only once.**
- ✓ **No tasklet will ever run parallel with itself, since they only run once , but tasklet can run parallely with other tasklets on SMP systems.**
- ✓ **Thus, if your driver has multiple tasklets , they must employsome sort of locking to avoid conflecting with each other.**
- ✓ **Tasklets are also guaranteed to run on same CPU that first scheduled them.**
- ✓ **An interrupt handler can thus be secure that a tasket will not being executing before the handler has completed.**

- ✓ **However another interrupt can be certainly be delivered while the tasklet is running, so locaking between the taskelt and interrupt handler may still required.**
- ✓ **Tasklet declaration:**
- ✓ **DECLARE_TASKLET(name, fucntion, data);**
- ✓ **name: Name given to the tasklet**
- ✓ **function: Fucntion that is called to execute a tasklet [it takes one unsigned long rgument and returns void]**
- ✓ **data: It is unsigned long value to be passed to the tasklet function**
- ✓ **The short driver declared tasklet as :**

```
void short_do_tasklet(unsigned long);
DECLARE_TASKLET(short_tasklet, short_do_tasklet,0);
```

The function tasklet_schedule is used to schedule a tasklet for running. If short is loaded with tasklet=1;

The structure declaration for tasklet is :

```
struct tasklet_struct
{
        struct tasklet_struct *next;        // linked list
        unsigned long state; // scheduled or running? (for waiting)
        atomic_t count; // enabled or disabled?
        void (*func)(unsigned long);// function pointer, i.e. bottom half
        unsigned long data; // argument for function
 };
```

The kernel provides two macros to declare tasklets. DECLARE TASKLET() sets the atomic counter to 0, which indicates that the tasklet is ready to be scheduled, where as DECLARE TASKLET DISABLED() sets the counter to 1, so that the tasklet cannot be scheduled without explicitly setting the counter to 0.

Q 8. What are the types of softirq. Difference between time softirq and Tasklet softirq

- ✓ **Types of Softirq along with priority:**

Types	Priority	Description
HI_SOFTIRQ	0	High priority tasklet
TIMER_SOFTIRQ	1	Timers
NET_TX_SOFTIRQ	2	Send network packets
NET_RX_SOFTIRQ	3	Receive network packets
BLOCK_SOFTIRQ	4	Block devices
TASKLET_SOFTIRQ	5	Normal priority tasklets
SCHED_SOFTIRQ	6	Scheduler
HRTIMER_SOFTIRQ	7	High resolution timer
RCU_SOFTIRQ	8	RCU locking

Q 9. What is system call ? How it works ? How it implemented in Linux ? Explain about trap .

- ✓ **A system call is a control entry point into the kernel, allowing a process to request that the kernel perform some action on the process's behalf.**
- ✓ **The kernel makes a range of services accessible to program via system call application programming interface.**
- ✓ **System call provide a layer between the hardware and user-space process.**
- ✓ **A system call changes the processor state from user mode to kernel mode, so that the CPU can access the protected kernel memory.**
- ✓ **The set of system call is fixed. Each system call is identified by a unique number.**
- ✓ **Each system call may have a set of arguments that specify information to be transferred from user space (i.e. the process virtual address space) to kernel space and vice-versa.**

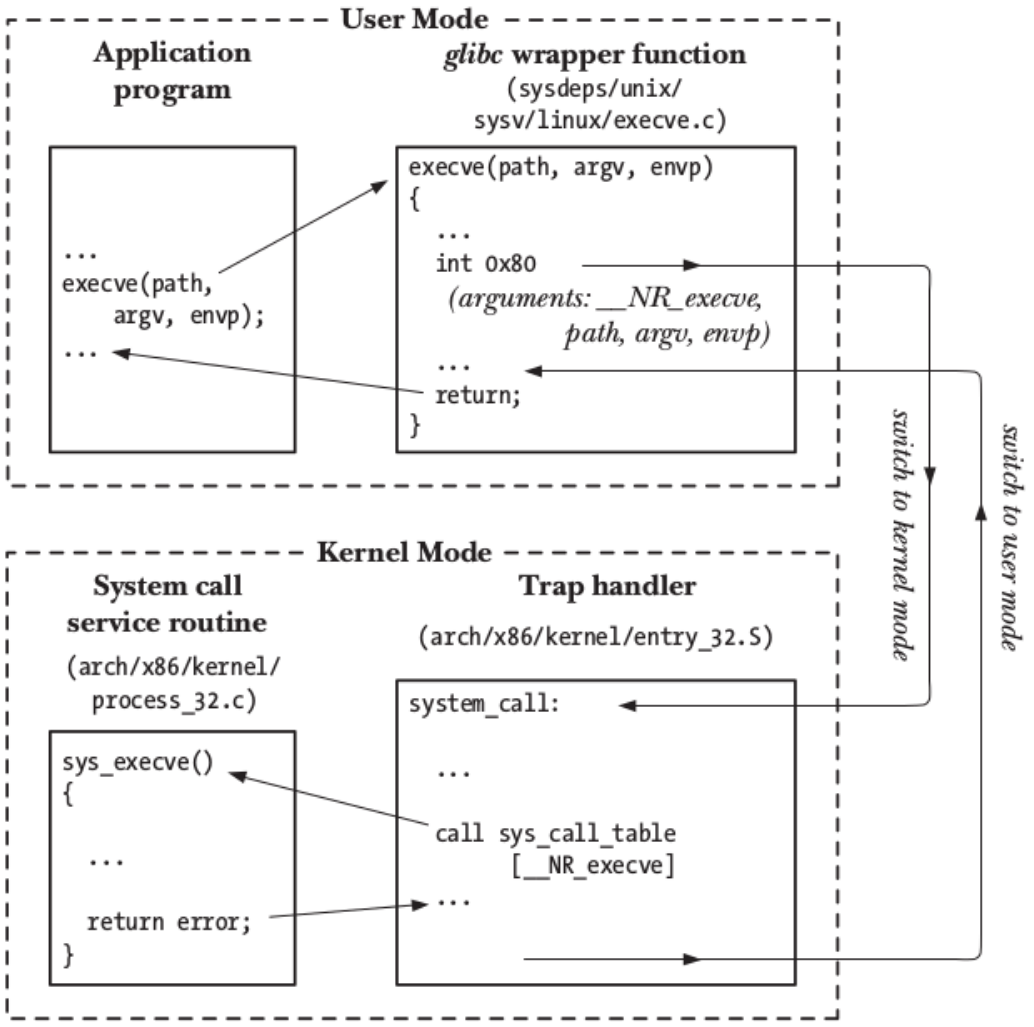


Figure 3-1: Steps in the execution of a system call

Working of system call (Eg. In x86-32):

- ✓ **The application program makes a system call by invoking a wrapper function in the C-library.**
- ✓ **These wrapper function must make all of the system call arguments available to the system call trap-handling routine. These arguments are passed to the wrapper via a stack, but the kernel expects them in specific registers. The wrapper function copies the arguments to these registers.**
- ✓ **Since all system call enter the kernel in the same way, the kernel need some method of identifying the system call. To permit this, the wrapper function copies the system call number into the specific register (%eax).**

- ✓ **The wrapper function executes a trap machine instruction (int 0x80), which cause the processor to switch from user mode to kernel mode and execute code pointed to the location 0x80 (128 decimal) of the system trap vector.**

  [More recent x86-32 architectures implement the sysenter instruction, which provides a faster method of entering kernel mode than the conventional int 0x80 trap instruction. The use of sysenter is supported in the 2.6 kernel and from glibc 2.3.2 onward.]

- ✓ **In responce to the trap to location 0x80, the kernel invokes its system_call() routine [located int the assembler file arch/i386/entry.S] to handle the trap. This handler :**
- ✓ **1. Saves register values onto kernel stack**
- ✓ **2. Checks the validity of the system call number**
- ✓ **3. Invokes the appropriate system call service routine, which is found by using system call number to index a table of all system call service routines (The kernel variable sys_call_table). If the system call service routine has any arguments, it first checks their validity. For example, it checks that addresses points to valid locations in user memory. Then service routine performs the required task (Eg. Transferring data between user memory to kernel memory or IO operations). Finally the service routine returns a result status to the system_call()routine.**
- ✓ **4. Restores the register values from kernel stack and place the system call return value on the stack.**
- ✓ **5. Returns to the wrapper function, simultaneously returning the processor to user mode.**
- ✓ **The retrun value is defined by the system call being invoked. In general, a 0 return value indicates success. A -1 returns value indicates an error, and erroe code is stored in errno.**
- ✓ **System call implementation in Linux: Eg . Implementation of myhello syscall**
- ✓ **1. Get the kernel to edit**
- ✓ **2. Edit the kernel**
- ✓ **a. Change syscall table**

  folder:arch/x86/entry/syscalls

  Edit syscall_32.tbl for 32-bit processor or syscall_64.tbl for 64-bit processor

At the end of non-specific system call number add number , name and entry point of system call.

The format is:

<number> <abi> <name> <entry point>

Eg : 333 64 myhello sys_myhello

b] Add the new system call (sys_myhello) in the system call header file:

File: include/linux/syscalls.h

Add the following line at the end of file:

asmlinkage int sys_myhello(void);

c] Create system call

File: kernel/myhello.c

eg:

```
#include<linux/kernel.h>
#include<linux/init.h>
#include<linux/syscalls.h>
#include<linux/linkage.h
asmlinkage int sys_myhello(void)
{
      printk("My myhello system call working !!!!");
       return 0;
}
```

c] Edit the Makefile of kernel directory

File:kernel/Makefile

Edit makefile: add myhello.o to obj-y list

d] Edit makefile of kernel: master make file

Add version , name , extraversion , patch level and sublevel also

Eg.

VERSION = 4

PATCHLEVEL = 12

SUBLEVEL = 4

EXTRAVERSION = .syscall

NAME = Rathod Raja V

3] Compile the kernel

Use following commands to configure and compile the kernel

$ Sudo make menuconfig : to configure the kernel

$ make : to compile the kernel

4] Install or update the kernel

$ sudo make modules_install install

5] Update the kernel

Restart the system and hold down shift to get grub menu

Select advanced option for ubuntu

Select your version of the kernel

6] Create a file to test the system call

Eg. userspace.c

```
#include<stdio.h>
#include<linux/kernel.h>
#include<sys/syscall.h>
#include<unistd.h>

int main()
{
        long int newsys=syscall(333);
        printf("\n system call sys_hello returned %ld \n ",newsys);
return 0;
```

}

7] To check the message of kernel

$ dmesg

It will display hello world at the end of the kernel message.

Q 10. What is the first function executed in the kernel ? What is first process in kernel ? When it will created ?

- ✓ **Start_kernel() is the first function executed in the kernel.**
- ✓ **Init is the first process in kernel**
- ✓ **When booting the system, the kernel creates a special process called init.**
- ✓ **Init is the parent of all the process which is derived from the program file /sbin/init.**
- ✓ **The init process always has the process id 1 and runs with superuser privileges.**
- ✓ **The init process cant kill (not even by the superuser), and it terminates only when the system is shut down.**
- ✓ **The main task of init is to create and monitor a range of processes required by running system.**

Q 11. What is interrupt latency . Interrupt mechanism in linux.

- ✓ **Interrupt latency is the time that elapses from when an interrupt is generated to when the source of interrupt is serviced. OR It is time taken between interrupt being raised and the interrupt handler being called.**

Q 12. How can come across Kernel Panic ? What is kernel panic ?

- ✓ **A kernel panic is an action taken by an operating system upon detecting an internel fatel error from which it cannot safely recovered.**
- ✓ **Kernel Panic is a computer error from which the OS can not quickly or easily recovered.**
- ✓ **The kernel provides basic services for all other parts of the operating system.**
- ✓ **Kernel panic are generally caused by an element beyond the linux kernel's control, including bad drivers, overtaxed memory and bugs.**

- ✓ **The platform is open-source, unlike mac and windows, so kernel development is open and collaborative.**
- ✓ **By knowing the Linux kernel panic logs we can determine the root causes, or dump the kernel contents with the crash utility for later analysis.**
- ✓ **Once the cause is determined, linux kernel panics with commands can be resolved with programming that instructs the kernel how to debug the offending program, change the configuration or file path or make changes in the BIOS.**

Q 14. What is the need of kernel ? How to debug it ?

- ✓ **A kernel is a central component of an operating system.**
- ✓ **It acts as an interface between the user application and hardware.**
- ✓ **The sole aim of the kernel is to manage the communication between the software (user level application) and hardware (CPU, disk memory etc).**
- ✓ **The main task of the kernel are:**
- ✓ **1. Process management**
- ✓ **2. Device management**
- ✓ **3. Memory Management**
- ✓ **4. Interrupt Handling**
- ✓ **5. IO communication**
- ✓ **6. File system**
- ✓ **7. Network Management**
- ✓ **8. Process Management etc**

Is Linux is Kernel or OS ?

- ✓ **There is difference between kernel or OS.**
- ✓ **Kernel is the heart of OS which manages the core features of an OS while if some useful applications and utilities are added over the kernel, then the complete package becomes OS.**
- ✓ **So, it can easily be said that an operating system consists of a kernel space and user space.**

- ✓ **So, we can say that linux is a kernel as it does not include applications like file-system utilities, windowing systems and graphical desktops, system administrator commands, text editors, compilers etc.**
- ✓ **So, various companies add these kind of applications over linux kernel and provide their operating system like ubuntu, centOS, redHat , suse etc.**
- ✓ **Every OS has a kernel.**

Kernel Debugging tool:

- ✓ **1. kdb**
- ✓ **2. kgdb**
- ✓ **3. ftrace**
- ✓ **Kdb : It is the kernel debugger of the linux kernel.**
- ✓ **It follows simplistic shell-style interface .**
- ✓ **We can use it to inspect memory, registers, process lists, dmesg, and even set breakpoints to stop in a certain location.**
- ✓ **Through kdb we can set breakpoints and execute some basic kernel run control (Although KDB is not source level debugger).**

Q 15. Write a code for spinlock

- ✓ **Spinlock is one of the locking mechanism provided in linux kernel. It is just like as semaphore but having higher performance . It has only two values 'locked' and 'unlocked' . If the process is in its critical section and lock is available then it set the locked bit and acquired the lock . If the lock is used by somebody then it goes into tight section and repeatedly checks until the lock is available. It is also called busy waiting .**

Q 16. What is kernel thread ? How to create kernel thread ?

- ✓ **Thread : Thread also known as light weight processes are the basic unit of cpu initialization. So, why do we call them as a light weight processes ? One of the reason is**

that the context switch between the threads takes much lesses time as compared to processes, which result from the fact that all the threads within the process share the same address space, so you dont need to switch the address space.

- ✓ **Kernel Thread : They are same as user space threads in many aspects, but one of the big differece is that they exist in the kernel space and execute in the priviledged mode and have full access to the kernel data structures. These are basically used to implement background tasks inside the kernel. The task can be handing of asynchronous events or waiting for an event occur.**
- ✓ `#include <kthread.h>`
- ✓ `kthread_create(int (*function)(void *data), void *data, const char name[], ...)`
- ✓ `function: The function that thread has to execute`
- ✓ `data: The data to be passed to the function`
- ✓ `name: The name by which the process will recognized in the kernel`

[A part of the program that can execute independently of other parts, ia called thread. A kernel thread is a task running only. A kernel thread is a kernel task running only in kernel mode. It usually has not been created by fork() or clone() system calls. It's not recommended to implement kernel thread if don't know well what are you doing]

Q 17. What is work queue ?

- ✓ The work queue runs in process context .
- ✓ Running in process context is the only way that can block (for instance, function that need to access some block of data on disk), because no process switch can take place in interrupt context.
- ✓ Neither deferrable functions nor function in a work queue can access the user mode address space of a process.
- ✓ In fact, a deferrable function cannot make any assumption about the process that is currently running when it is executed.

- ✓ On the other hand, a function in work queue is executed by kernel thread, so there is no user mode address space ro access.
- ✓ The main data structure assocaited with work queue is a descriptor called workqueue_struct, which contain among other things, an array of NR_CPUS elements, the maximum number of cpu in the systems.
- ✓ Workqueues have a type of struct workqueue_struct, which is defined in *<linux/workqueue.h>*. A workqueue must be explicitly created before use, using one of the following two functions:

```
struct workqueue_struct *create_workqueue(const char *name);
struct workqueue_struct *create_singlethread_workqueue(const char *name);
```

Q 19. What are software signal or interrupt in Linux ?

- ✓ **Signals are the software interrupts sent to a program to indicate that an important event has occured.**
- ✓ **The signal can be vary from user request to illigal access errors.**
- ✓ **Some signals, such as the interrupt signal, indicate that the user has asked the program to do something that is not in the usual flow of control.**
- ✓ **The list of comman signals :**

Signal Name	Signal Number	Description
SIGHUP	1	Hang up detected on controlling terminal or death of controlling process
SIGINT	2	Issued if the user sends an interrupt signal (Ctrl + C)
SIGQUIT	3	Issued if the user sends a quit signal (Ctrl + D)
SIGFPE	8	Issued if an illegal mathematical operation is attempted
SIGKILL	9	If a process gets this signal it must quit immediately and will not perform any clean-up operations
SIGALRM	14	Alarm clock signal (used for timers)
SIGTERM	15	Software termination signal (sent by kill by default)

Q 20. What is Kmalloc and Vmalloc ?

Memory allocation in Linux kernel is different from the user space counterpart. The following facts are noteworthy**.**

Kernel memory is not pageable

Kernel memory allocation mistakes can cause system oops (system crash) easily.

Kernel memory has limited hard stack size limit.

There’re two ways to allocate memory space for a kernel process, statically from the stack or dynamically from the heap.

Static Memory Allocation

The static memory allocation is normally used you know how much memory space you’ll need. For example,

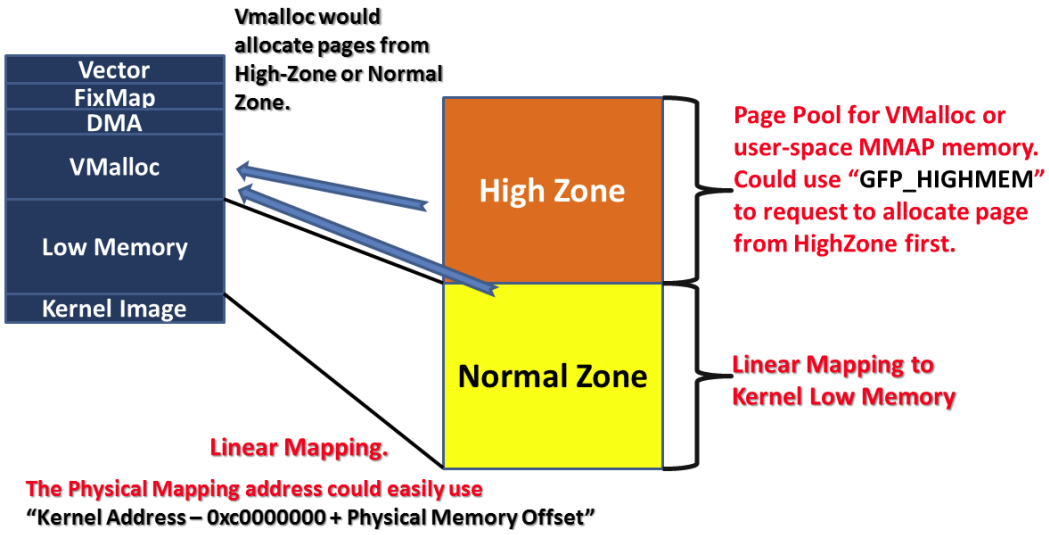
```
#define BUF_LEN    2048

char buf[BUF_LEN];
```

However, the kernel stack size is fixed and limited (the limit is architecture dependent, but normally it's only tens of kilobytes). Therefore people seldom request big chunk of memory in the stack. The better way is to allocate the memory dynamically from heap.

Vmalloc()	Kmalloc()

Physically non-contiguous. how does the CPU know it is contiguous or not? This is because of linear mapping, or also called one-to-one mapping (or direct mapping) *vmalloc is*where non-contiguous physically memory can be used that is contiguous in virtual memory. An area is reserved in the virtual address space between*vmalloc_start and vmalloc_end* The vmalloc function is defined in /lib/modules/ $(uname -r)/build/include/linux/vmalloc.h as below, *void *vmalloc(unsigned long size);* It's Linux kernel's version of malloc() function, which is used in user space. Like malloc, the function allocates virtually contiguous memory that may or may not physically contiguous. To free the memory space allocated by vmalloc, one simply call vfree(), which is defined as, *void vfree(const void *addr);*	Physically contiguous kmalloc() returns physically contiguous memory, malloc() does not guarantee anything about the physical memory mapping. Memory is reserved and locked, it cannot be swapped , malloc does not actually allocate physical memoey. Physical memory gets mapped later, during use. The most commonly used memory allocation function in kernel is kmalloc, which is defined in /lib/modules/$(uname -r)/build/include/linux/slab.h as below, *void * kmalloc(size_t size, int flags);* kmalloc allocates a region of physically contiguous (also virtually contiguous) memory and return the pointer to the allocated memory. It returns NULL when the operation fails. The behavior of kmalloc is dependent on the second parameter flags. Here only the two most popular flags are introduced, ***GFP_KERNEL***: this flag indicates a normal kernel memory allocation. The kernel might block the requesting code, free up enough memory and then continue the allocation. It cannot be used in places where sleep is not allowed. ***GFP_ATOMIC***: this flag indicates the kmalloc function is atomic operation. Atomic operation means the function is performed entirely or not performed at all. It cannot block at the middle of execution. As the kernel cannot jump out of the allocation and free up memory to satisfy the function request, this function has a higher chance of failure with this flag passed in. To free up the memory allocated by kmalloc, one can use kfree defined as below, *void kfree(const void *objp);* kmalloc allocates physically contiguous memory

vmalloc allocates virtually contiguous memory space (not necessarily physically contiguous), But when a large chunk of memory is needed, vmalloc is used often as it doesn't require physically contiguous memory and the kernel can satisfy the request with much less effort than using kmalloc.	(also virtually contiguous). Most of the memory allocations in Linux kernel are done using kmalloc. On many architectures, hardware devices don't understand virtual address. Therefore, their device drivers can only allocate memory using kmalloc. kmalloc has better performance in most cases because physically contiguous memory region is more efficient than virtually contiguous memory.



---

21. User space malloc created then how kernel knos about the created malloc ?

22. If driver is not present then, how you will you on it for application.

23. Difference between virtual address and physical address. Write in detail about address translation scheme with a diagram and explanation.

24. What are interrupt & How they works or how the interrupts are handled for any applications.

Q 25. Explain Memory Fragmentation ? How it will happen ?What is the way to eliminate them.

- ✓ **Fragmentation occurs in a dynamic memory allocation system when many of the free blocks are too small to satisfy the request.**
- ✓ **Two types:**
- ✓ **1. External Fragmentation**
- ✓ **2. Internal Fragmentation**
- ✓ **External fragmentation happens when a dynamic memory allocation algorithm allocates some memory and a small piece is left over that cannot be effictively used.**
- ✓ **If too month external fragmentation occurs, the amount of usable memory is drastically reduced.**
- ✓ **Total memory space exists to satisfy a request but it is not contiguous.**
- ✓ **Internal fragmentation is the space wasted inside of allocated memory blocks because of restriction on the allowed size of allocated blocks.**
- ✓ **This size difference is memory internal to a partition, but not being used.**

Q 26. What is virtual memory ? Why we need Virtual memory ? How to implement virtual memory ?

Q 27. What happen if interrupt comes in thread execution and how to handle it ?

Q 28. What is LK (Little Kernel) ? What it do ?

- ✓ **The Little kernel is the boot loader and perform the basic tasks of hardware initialization, reading the linux kernel and ramdisk from storage and loading it up to RAM, setting up initial registers and command line arguments for Linux kernel, and jumps to the kernel.**
- ✓ **Android bootloader is the LK bootloader .**
- ✓ **LK performs,**

- ✓ **1. Hardware initialization: setting up vector table, MMU, cache, initialize peripherals, storage, USB, crypto etc.**
- ✓ **2. Loads boot.img from storage.**
- ✓ **3. Support flashing and recovery**
- ✓ **LK runs in 32-bit mode even on a 64-bit architecture. The jump from LK 32-bit to 64-**

  bit kernel goes through secure mode.

29. What is device tree ? How to enter the new device information in the device tree ? Which file is generated after device tree compilation ? How you will make changes in device tree.

- ✓ **Device tree is the data structure for describing the system hardware.**
- ✓ **Device tree source [dts]:**
- ✓ **1. A simple structure of nodes and properties.**
- ✓ **2. Properties are key value pair and node may contain both properties and child nodes.**
- ✓ **3. Format of .dts file is like C and C++ style comments.**
- ✓ **4. For ARM architecture, the device tree source can be found in the :**
  - ✓ **kernel/arch/arm/boot/dts folder.**
  - ✓ **Device tree blob [dtb] :**
  - ✓ **1. The device tree compiler [.dtc] compiles the .dts into binaly object [.dtb] understandable by linux kernel.**
  - ✓ **This blob is appended to the kernel image as shown below during compilation:**

Boot img header
Kernel
RAM disk
Device tree table

---

30. What is kernel ? How it works ?

- ✓ **A kernel is a program that constitutes the core of a computer operating system.**

- ✓ **It has complete control over everything that occurs in the system.**
- ✓ **The kernel can be contrasted with a shell (such as bash, csh, or ksh in Unix like operating system), which is the outermost part of the operating system. And a program that intersects with user commands.**
- ✓ **The kernel itself does not interact directly with the user, but rather interact with the shell and other programs as well as with the hardware device on the system, including the processor (also called CPU), memory or disk drives.**
- ✓ **The kernel is a first part of the OS to load into memory during booting (i.e. system startup), and it remains there for the entire duration of the computer session because its service are required continuously. So it is important for it to be as small as possible while still providing all the essential services needed by the other parts of the operating system and by the various application programs.**
- ✓ **Because of its critical nature, the kernel code is usually loaded into a protected area of memory , which prevent it from being overwritten by other, less frequently used parts of the operating system or by application programs.**
- ✓ **The kernel performs it tasks, such as executing process and handling interrupts, in kernel space, whereas everything a user normally does, such as writing text in a text editor or running program in a GUI , is done in user space.**
- ✓ **This seperation is made in order to prevent user data and kernel data from interfacing with each other and thereby diminishing performance or causing the system to become unstable (and possibly crashing).**
- ✓ **When a computer crashes, its actually means that the kernel has crashed. If only a single program has crashed but rest of the system remains in the operation, then the kernel itself has not crashed.**
- ✓ **A crash is a situation in which a program , either a user application or a part of the operating system or by applicaton program or a part the operating system, stops performing its expected functions and responding to other parts of the system. The program might appear to the user to freeze. If such program is a critical to the operation of the kernel , the entire computer could stall or shut down.**

The kernel provides basic services for all other parts of the operating system, typically including memory management, process management, file management, and IO management. (i.e. accessing the peripheral device).

These services are requested by other parts of the operating system or by application program through a specified set of the program interface referred to as a system calls.

The contents of kernel vary considerably according to the operating system , but typically include:

1. Scheduler : Determines how the various processors share the kernel processing time.

2. Supervisor : Which grant use of the computer to each process when it is scheduled.

3. Interrupt handler: Handles all request from the various hardware devices.

4. Memory Manager : Allocates the system address spaces (i.e. location in memory) among all user of the kernel's services.

The kernel should not be confused with the BIOS . The BIOS is an independent program stored in a chip on the motherboard. That is used during the booting process for such tasks as initializing the hardware and loading the kernel into memory. Whereas the BIOS always remains in the computer and is specific to its particular hardware, the kernel can be easily replaced or upgraded by changing or upgrading the OS, or in case of Linux , by adding new kernel or modifing an existing kernel.

[**Categories of Kernels:**

Kernels can be classified into four broad categories: *monolithic kernels*, *microkernels*, *hybrid kernels* and *exokernels*. Each has its own advocates and detractors.

Monolithic kernels, which have traditionally been used by Unix-like operating systems, contain all the operating system core functions and the *device drivers* (small programs that allow the operating system to interact with hardware devices, such as disk drives, video cards and printers). Modern monolithic kernels, such as those of Linux and FreeBSD, both of which fall into the category of Unix-like operating systems, feature the ability to load *modules* at runtime, thereby allowing easy extension of the kernel's capabilities as required, while helping to minimize the amount of code running in kernel space.

A microkernel usually provides only minimal services, such as defining memory address spaces, interprocess communication (IPC) and process management. All other functions, such as hardware management, are implemented as processes running independently of the kernel. Examples of microkernel operating systems are AIX, BeOS, Hurd, Mach, Mac OS X, MINIX and QNX.

Hybrid kernels are similar to microkernels, except that they include additional code in kernel space so that such code can run more swiftly than it would were it in user space. These kernels represent a compromise that was implemented by some developers before it was demonstrated that pure microkernels can provide high performance. Hybrid kernels should not be confused with monolithic kernels that can load modules after booting (such as Linux).

Most modern operating systems use hybrid kernels, including Microsoft Windows NT, 2000 and XP. DragonFly BSD, a recent *fork* (i.e., variant) of FreeBSD, is the first non-Mach based BSD operating system to employ a hybrid kernel architecture.

Exokernels are a still experimental approach to operating system design. They differ from the other types of kernels in that their functionality is limited to the protection and multiplexing of the raw hardware, and they provide no hardware abstractions on top of which applications can be constructed. This separation of hardware protection from hardware management enables application developers to determine how to make the most efficient use of the available hardware for each specific program.

Exokernels in themselves they are extremely small. However, they are accompanied by *library operating systems*, which provide application developers with the conventional functionalities of a complete operating system. A major advantage of exokernel-based systems is that they can incorporate multiple library operating systems, each exporting a different API (application programming interface), such as one for Linux and one for Microsoft Windows, thus making it possible to simultaneously run both Linux and Windows applications.

The Monolithic Versus Micro Controversy

In the early 1990s, many computer scientists considered monolithic kernels to be obsolete, and they predicted that microkernels would revolutionize operating system design. In fact, the development of Linux as a monolithic kernel rather than a microkernel led to a famous *flame war* (i.e., a war of words on the Internet) between Andrew Tanenbaum, the developer of the MINIX operating system, and Linus Torvalds, who originally developed Linux based largely on MINIX.

Proponents of microkernels point out that monolithic kernels have the disadvantage that an error in the kernel can cause the entire system to crash. However, with a microkernel, if a kernel process crashes, it is still possible to prevent a crash of the system as a whole by merely restarting the service that caused the error. Although this sounds sensible, it is questionable how important it is in reality, because operating systems with monolithic kernels such as Linux have become extremely stable and can run for years without crashing.

Another disadvantage cited for monolithic kernels is that they are not *portable*; that is, they must be rewritten for each new *architecture* (i.e., processor type) that the operating system is to be used on. However, in practice, this has not appeared to be a major disadvantage, and it has not prevented Linux from being ported to numerous processors.

Monolithic kernels also appear to have the disadvantage that their *source code* can become extremely large. Source code is the version of *software* as it is originally *written* (i.e., typed into a computer) by a human in *plain text*(i.e., human readable alphanumeric characters) and before it is converted by a compiler into *object code* that a computer's processor can directly read and execute.

For example, the source code for the Linux kernel version 2.4.0 is approximately 100MB and contains nearly 3.38 million lines, and that for version 2.6.0 is 212MB and contains 5.93 million

lines. This adds to the complexity of maintaining the kernel, and it also makes it difficult for new generations of computer science students to study and comprehend the kernel. However, the advocates of monolithic kernels claim that in spite of their size such kernels are easier to design correctly, and thus they can be improved more quickly than can microkernel-based systems.

Moreover, the size of the compiled kernel is only a tiny fraction of that of the source code, for example roughly 1.1MB in the case of Linux version 2.4 on a typical Red Hat Linux 9 desktop installation. Contributing to the small size of the compiled Linux kernel is its ability to dynamically load modules at runtime, so that the basic kernel contains only those components that are necessary for the system to start itself and to load modules.

The monolithic Linux kernel can be made extremely small not only because of its ability to dynamically load modules but also because of its ease of customization. In fact, there are some versions that are small enough to fit together with a large number of utilities and other programs on a single floppy disk and still provide a fully functional operating system (one of the most popular of which is *muLinux*). This ability to miniaturize its kernel has also led to a rapid growth in the use of Linux in *embedded systems* (i.e., computer circuitry built into other products).

Although microkernels are very small by themselves, in combination with all their required auxiliary code they are, in fact, often larger than monolithic kernels. Advocates of monolithic kernels also point out that the two-tiered structure of microkernel systems, in which most of the operating system does not interact directly with the hardware, creates a not-insignificant cost in terms of system efficiency.]

Q 31. Kernel Switching ?

- ✓ **The context switching takes place in kernel level is called as kernel switching.**
- ✓ **The context switch is the process of storing and restoring the state (specifically the execution context) of a process or thread so that execution can be resumed from the same point at a later time.**
- ✓ **This enables a multiple process to share a single CPU and is an essential feature of a multitasking operating system.**
- ✓ **An operating system needs to keep switching between several process to maintain the "multiprocessing" property. However, before you switch from one process to another, it is a good idea to save the state and essential information about the process you are switching (called "context") away from so that you can come back to it later when you need to. This technique of saving context and then using it to restore the process operation is known as context switching.**

✓ **Context switching is used for switching processes and maintaining state. When an interrupt request for resourse used by a process, to provide the resourse to I/o device , the current working state of process and other parameters are maintained in PCB and the resources are given to I/o requested. When the process again gets the resources it look into it's PCB and starts from where it left it's execution.**

✓ **A context switch (also sometimes referred to as a process switch or a task switch) is the switching of the CPU (central processing unit) from one process or thread to another.**

A process (also sometimes referred to as a task) is an executing (i.e.running) instance of a program. In Linux, threads are lightweight processes that can run in parallel and share an address space (i.e., a range of memory locations) and other resources with their parent processes (i.e., the processes that created them).

A context is the contents of a CPU's registers and program counter at any point in time. A register is a small amount of very fast memory inside of a CPU (as opposed to the slower RAM main memory outside of the CPU) that is used to speed the execution of computer programs by providing quick access to commonly used values, generally those in the midst of a calculation. A program counter is a specialized register that indicates the position of the CPU in its instruction sequence and which holds either the address of the instruction being executed or the address of the next instruction to be executed, depending on the specific system.

Context switching can be described in slightly more detail as the kernel (i.e., the core of the operating system) performing the following activities with regard to processes (including threads) on the CPU: (1) suspending the progression of one process and storing the CPU's state (i.e., the context) for that process somewhere in memory, (2) retrieving the context of the next process from memory and restoring it in the CPU's registers and (3) returning to the location indicated by the program counter (i.e., returning to the line of code at which the process was interrupted) in order to resume the process.

A context switch is sometimes described as the kernel suspending execution of one process on the CPU and resuming execution of some other process that had previously been suspended. Although this wording can help clarify the concept, it can be confusing in itself because a process is, by definition, an executing instance of a program. Thus the wording suspending progression of a process might be preferable.

Context Switches and Mode Switches

Context switches can occur only in kernel mode. Kernel mode is a privileged mode of the CPU in which only the kernel runs and which provides access to all memory locations and all other system resources. Other programs, including applications, initially operate in user mode, but they can run portions of the kernel code via system calls. A system call is a request in a Unix-like operating system by an active process (i.e., a process currently progressing in the CPU) for a service performed by the kernel, such as input/output (I/O) or process creation (i.e., creation of a new process). I/O can be defined as any movement of information to or from the combination of the CPU and main memory (i.e. RAM), that is, communication between this combination and the computer's users (e.g., via the keyboard or mouse), its storage devices (e.g., disk or tape drives), or other computers.

The existence of these two modes in Unix-like operating systems means that a similar, but simpler, operation is necessary when a system call causes the CPU to shift to kernel mode. This is referred to as a mode switch rather than a context switch, because it does not change the current process.

Context switching is an essential feature of multitasking operating systems. A multitasking operating system is one in which multiple processes execute on a single CPU seemingly simultaneously and without interfering with each other. This illusion of concurrency is achieved by means of context switches that are occurring in rapid succession (tens or hundreds of times per second). These context switches occur as a result of processes voluntarily relinquishing their time in the CPU or as a result of the scheduler making the switch when a process has used up its CPU time slice.

A context switch can also occur as a result of a hardware interrupt, which is a signal from a hardware device (such as a keyboard, mouse, modem or system clock) to the kernel that an event (e.g., a key press, mouse movement or arrival of data from a network connection) has occurred.

Intel 80386 and higher CPUs contain hardware support for context switches. However, most modern operating systems perform software context switching, which can be used on any CPU, rather than hardware context switching in an attempt to obtain improved performance. Software context switching was first implemented in Linux for Intel-compatible processors with the 2.4 kernel.

One major advantage claimed for software context switching is that, whereas the hardware mechanism saves almost all of the CPU state, software can be more selective and save only that portion that actually needs to be saved and reloaded. However, there is some question as to how important this really is in increasing the efficiency of context switching. Its advocates also claim that software context switching allows for the possibility of improving the switching code, thereby further enhancing efficiency, and that it permits better control over the validity of the data that is being loaded.

The Cost of Context Switching :

Context switching is generally computationally intensive. That is, it requires considerable processor time, which can be on the order of nanoseconds for each of the tens or hundreds of switches per second. Thus, context switching represents a substantial cost to the system in terms of CPU time and can, in fact, be the most costly operation on an operating system.

Consequently, a major focus in the design of operating systems has been to avoid unnecessary context switching to the extent possible. However, this has not been easy to accomplish in practice. In fact, although the cost of context switching has been declining when measured in terms of the absolute amount of CPU time consumed, this appears to be due mainly to increases in CPU clock speeds rather than to improvements in the efficiency of context switching itself.

One of the many advantages claimed for Linux as compared with other operating systems, including some other Unix-like systems, is its extremely low cost of context switching and mode switching.
