# General Questions

**1. What do you understand by System Calls? How System Call Works? How the context switching is related to System call?**

**Ans:**

1. Provide interface between a process and operating system.

2. It is controlled entry point to the kernel, allowing a process to request that the kernel perform some action on the process behalf.

3. The kernel makes a range of services accessible to programs via the system call application programming interface (API).

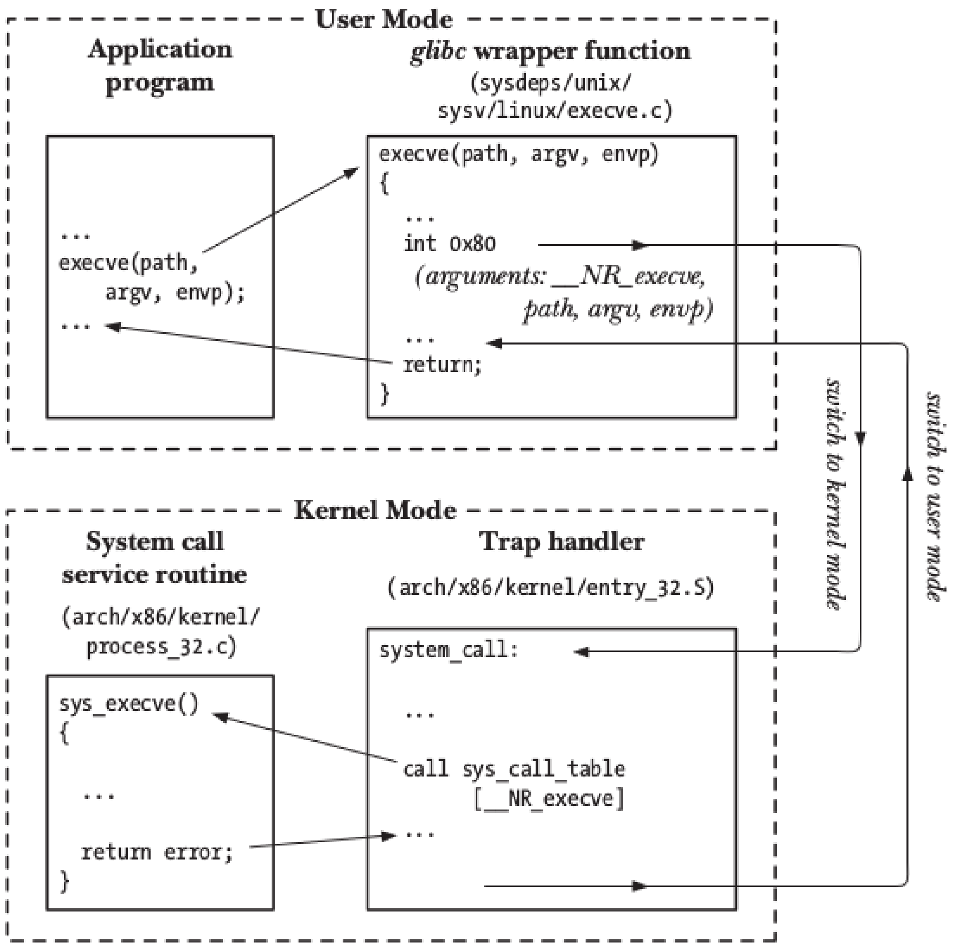
4. These service includes, example creating a new process, performing I/O , and creating pipe for interprocess communication.

5. A system call changes the processor state from user mode to kernel mode, so that the CPU can access protected kernel memory.

6. The set of system calls is fixed. Each system call is identified by a unique number.

7. Each system call may have a set of arguments that specify information to be transferred from user space to kernel space and vice versa.

8.Steps in executing system call:


User Mode
Application program
glibc wrapper function
(sysdeps/unix/sysv/linux/execve.c)
...
execve(path, argv, envp);
...
execve(path, argv, envp)
{
...
int 0x80
(arguments: __NR_execve, path, argv, envp)
...
return;
}
switch to kernel mode
switch to user mode
Kernel Mode
System call service routine
(arch/x86/kernel/process_32.c)
Trap handler
(arch/x86/kernel/entry_32.S)
sys_execve()
{
...
return error;
}
system_call:
...
call sys_call_table [__NR_execve]
...

9.When making a system call in Linux, a context switch is done from user-space to kernel space

10. Each process has an associated kernel mode stack, that is used by the system call. Before the system call is executed, the CPU registers of the process are stored on its user-mode stack, this stack is different from the kernel mode stack, and is the one which the process uses for user-space executions.

11. When a process is in kernel mode (or user mode), calling functions of the same mode will not require a context switch.

## 2. How to add new system call into the Kernel? What are the various steps involved?

**An:**

**Step 1: Get the source**

The first step is to download the source code of the Linux kernel. I used the one available from the repositories but feel free to get the sources from kernel.org.

```
$ apt-get source linux-image-$(uname -r)
```

This would download all the archives and unpack it into a directory linux-3.2.0.

**Step 2: Add system call to system call table**

Open the file arch/x86/kernel/syscall_table_32.S and add the following line.

```
.long sys_hello
```

**Step 3: Define macros associated with system call**

Open the file arch/x86/include/asm/unistd_32.h. You will notice that a macro is defined for each system call. At the end of the huge macro definition, add a definition for our new system call. I added the following line:

```
#define __NR_hello 349
```

and accordingly incremented the value of the macro NR_SYSCALLS:

```
#define NR_Syscalls 350
```

Also, add the macro definition to the file arch/x86/include/asm/unistd_64.h

```
#define __NR_hello 312
```

```
        __SYSCALL(__NR_hello, sys_hello)
```

Now to the file include/linux/syscalls.h, add the prototype of the system call.

```
        asmlinkage long sys_hello(void);
```

Now, in the root directory of the kernel sources, create a directory named hello and in it, a file hello.c with the following content:

```
#include <linux/kernel.h>
asmlinkage long sys_hello(void)
{
        printk("Hello world\n");
        return 0;
}
```

printk is similar to printf function of C but writes to the kernel log instead of the screen. asmlinkage is a key word used to indicate that all parameters of the function would be available on the stack.

After creating the function definition, create a file named Makefile within the hello directory and the following content to the file:

```
        obj-y := hello.o
```

This is to ensure that our hello.c file is compiled and included in the kernel.

Now, to the Makefile in the root directory of the kernel sources, edit the following line:

```
        core-y          += kernel/ mm/ fs/ ipc/ security/ crypto/ block/
```

to:

```
        core-y          += kernel/ mm/ fs/ ipc/ security/ crypto/ block/hello
```

This is to tell the compiler that the source files of our new system call are in present in the hello directory.

That’s it-you have added your own system call! Now all you need to compile the kernel. [1] is a good place to get information on how to compile the kernel. You needn’t do all the steps since you have the sources-read it carefully :)!

After you compile and reboot into the kernel you just compiled, try running the following program.

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

#define __NR_hello 312 //349 if you are running a 32bit kernel and following my tutorial

long hello_syscall(void)
{
        return syscall(__NR_hello);
}

int main(int argc, char *argv[])
{
        long int a = hello_syscall();
        printf("System call returned %ld\n", a);
        return 0;
}
```

The output of the program would be:

```
        System call returned 0
```

The printk's output get written to the kernel log. To view it, run the command

```
        $dmesg
```

**4. What is difference between GPOS (General Purpose Operating System) and Special Purpose Operating System such as RTOS?**

**Ans:**

| **General Purpose OS** | **Real Time OS** |
|---|---|
| 1.It is used in multiuser environment | 1.it is dedicated to a single work |
| 2.protected memory model | 2.flat memory model |
| 3.non scalable | 3.scalable |
| 4.high interrupt latency | 4.low interrupt latency |
| 5.time insensitive | 5.time sensitive |

**5. How crash is different from panic?**

**Ans:**

1. A kernel panic happen when the kernel detect an error from which it cant recover.

2. The reason for a kernel panic can be a bug in the kernel or a hardware or software issue that cause an unexpected or unpredictable condition for the kernel.

3. By using kdmup or crash we can see the crash.

4. A Kernel Crash Dump refers to a portion of the contents of volatile memory (RAM) that is copied to disk whenever the execution of the kernel is disrupted. The following events can cause a kernel disruption : Kernel Panic. Non Maskable Interrupts (NMI)

5. A kernel panic is a computer error from which the operating system (OS) cannot quickly or easily recover. The term applies primarily to Unix-based systems and to Mac OS X. In other systems, the equivalent of a kernel panic is known by slang terms such as blue screen of death, sad Mac or bomb.

**6. How user space is different from kernel space?**

**Ans:**

1. The random access memory (RAM) can be divided into two distinct regions namely – the kernel space and the user space.

2. The kernel runs in the part of memory entitled to it. This part of memory cannot be accessed directly by the processes of the normal users, while as the kernel can access all parts of the memory.

3. To access some part of the kernel, the user processes have to use the predefined system calls i.e. open, read, write etc. Also, the C library functions like printf call the system call write in turn.

**4.** The system calls act as an interface between the user processes and the kernel processes.

5. The access rights are placed on the kernel space in order to stop the users from messing up with the kernel, unknowingly.

6. So, when a system call occurs, a software interrupt is sent to the kernel. The CPU may hand over the control temporarily to the associated interrupt handler routine. The kernel process which was halted by the interrupt resumes after the interrupt handler routine finishes its job.

7.So, when a system call occurs, a software interrupt is sent to the kernel. The CPU may hand over the control temporarily to the associated interrupt handler routine. The kernel process which was halted by the interrupt resumes after the interrupt handler routine finishes its job.

**7. What are the main modules to implement OS Kernel Functionalities?**

Ans:

**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.

## 8. What are the different types of Kernels?

Ans:  Mainly two types of Kernel

1. **Monolithic Kernel**

**2. Micro-kernel**

### 1.Monolithic kernel

Earlier in this type of kernel architecture, all the basic system services like process and memory management, interrupt handling etc were packaged into a single module in kernel space. This type of architecture led to some serious drawbacks like 1) Size of kernel, which was huge. 2) Poor maintainability, which means bug fixing or addition of new features resulted in recompilation of the whole kernel which could consume hours

In a modern day approach to monolithic architecture, the kernel consists of different modules which can be dynamically loaded and un-loaded. This modular approach allows easy extension of OS's capabilities. With this approach, maintainability of kernel became very easy as only the concerned module needs to be loaded and unloaded every time there is a change or bug fix in a particular module. So, there is no need to bring down and recompile the whole kernel for a smallest bit of change. Also, stripping of kernel for various platforms (say for embedded devices etc) became very easy as we can easily unload the module that we do not want.

### 2. Micro Kernel

This architecture majorly caters to the problem of ever growing size of kernel code which we could not control in the monolithic approach. This architecture allows some basic services like device driver management, protocol stack, file system etc to run in user space. This reduces the kernel code size and also increases the security and stability of OS as we have the bare minimum code running in kernel. So, if suppose a basic service like network service crashes due to buffer overflow, then only the networking service's memory would be corrupted, leaving the rest of the system still functional.

In this architecture, all the basic OS services which are made part of user space are made to run as servers which are used by other programs in the system through inter process communication (IPC). eg: we have servers for device drivers, network protocol stacks, file systems, graphics, etc. Microkernel servers are essentially daemon programs like any others, except that the kernel grants some of them privileges to interact with parts of physical memory that are otherwise off limits to most programs. This allows some servers, particularly device drivers, to interact directly with hardware. These servers are started at the system start-up.

**9. How Kernel is different from OS?**

Ans:

1. Kernel is a part of OS

2.The operating system is the software package that communicates directly to the hardware and our application.

3. The kernel is the lowest level of the operating system.

4. The kernel is the main part of the operating system and is responsible for translating the command into something that can be understood by the computer.

5. The main functions of the kernel are:

- Memory Management
- Network Management
- Device Driver
- File Management
- Process Management
-

**10. How to allocate memory from Kernel Space?**

**Ans:** Main two methods

1. **Static Memory Allocaation**

**2. Dynamic Memory Allocation**

**1. 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.

### 3. Dynamic Memory Alloation:

There’re two functions available to allocate memory from heap in Linux kernel process,

- *vmalloc*

  The vmalloc function is defined in /lib/modules/$(uname -r)/build/include/linux/vmalloc.h as below,

  ```
  void *vmalloc(unsigned long size);
  ```

## 11. Explain about Linux Boot Loaders viz., LILO, GRUB

**Ans:**

### Boot Loaders:

Bootloader or Boot Manager is a piece of code that runs before any operating system is running.

When a computer is powered-up or restarted, the basic input/output system (BIOS) performs some initial tests, and then transfers control to the master boot record (MBR) where the boot loader resides.

Used to boot other operating systems, usually each operating system has a set of bootloaders specific for it

Usually contain several ways to boot the OS kernel and also contain commands for debugging and/or modifying the kernel environment.

### LILO (Linux Loader):

LILO is a simple yet powerful and stable Linux boot loader. The primary advantage of LILO is the fact that it allows for fast boot up of Linux when installed in the MBR. Its main limitation is the fact that not all computers tolerate modification of the MBR. In these situations, there are alternative approaches for using LILO, but it takes longer., LILO has become less popular among Linux users.

While it loads, the word “LILO” is displayed on the screen and each letter appears before or after a particular event has occurred. However, the development of LILO was stopped in December 2015, it has a number of features are:

The LILO bootstrap process involves locating the kernel by in essence pointing to the first logical-sector of the Kernel file
Does not offer an interactive command line interface
Supports several error codes
All its files are stored in the first 1024 cylinders of a drive
Faces limitation with BTFS, GPT and RAID plus many more.

## GNU GRUB:

GNU GRUB is a bootloader (can also be spelled boot loader) capable of loading a variety of free and proprietary operating systems. GRUB will work well with Linux, DOS, Windows, or BSD. GRUB stands for **GRand Unified Bootloader**.

By default, MBR code looks for the partition marked as active and once such a partition is found, it loads its boot sector into memory and passes control to it.

GRUB replaces the default MBR with its own code.

Furthermore, GRUB works in stages.

**Stage 1** is located in the MBR and mainly points to Stage 2, since the MBR is too small to contain all of the needed data.

**Stage 1.5** also exists and might be used if the boot information is small enough to fit in the area immediately after MBR.

**Stage 2** points to its configuration file, which contains all of the complex user interface and options we are normally familiar with when talking about GRUB. Stage 2 can be located anywhere on the disk. If Stage 2 cannot find its configuration table, GRUB will cease the boot sequence and present the user with a command line for manual configuration.

GRUB has the following prominent features:

- Supports multi-boot

- GRUB can read ext2 partitions, allows GRUB to access its configuration file,/boot/grub/grub.conf
- Supports multiple hardware architectures and operating systems such as Linux and Windows GRUB provides a true command-based, pre-OS environment on x86 machines.
- Enables access to GRUB editor
- Supports setting of passwords with encryption for security
- GRUB supports Logical Block Addressing (LBA) mode.

### 12. Check with compgen –c command.

**Ans: Commands related to compgen are:**

```
$compgen -c
$compgen -ac | grep find

$compgen -c | wc -l
```

# OS Concept

## 1.What is meant by Paging?

**Ans:** Paging is a memory management scheme that eliminates the need for contiguous allocation of physical memory. This scheme permits the physical address space of a process to be non – contiguous.

## 2.What is meant by Segmentation?

**Ans:** Segmentation is a Memory Management technique in which memory is divided into variable sized chunks which can be allocated to processes. Each chunk is called a segment.

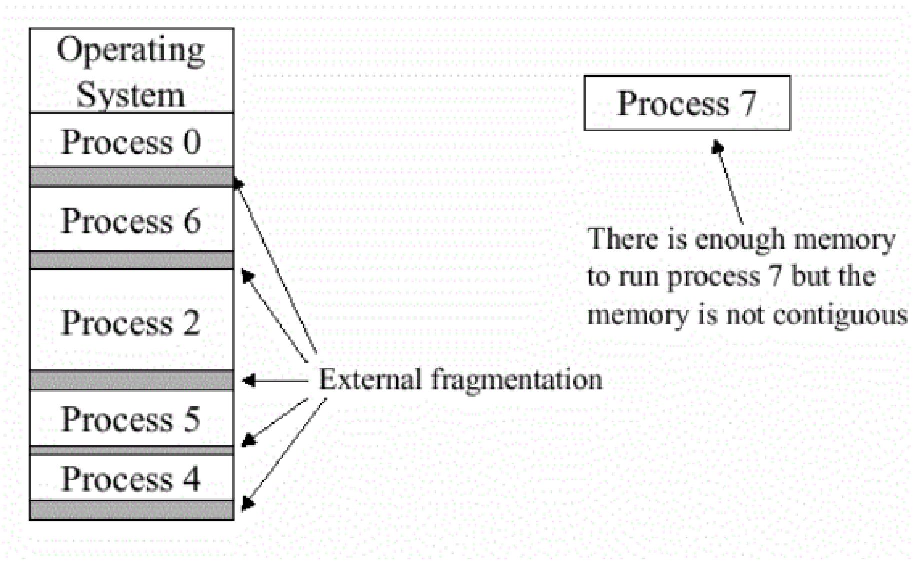
## 3.Explain Internal and External Fragmentations?

**Ans:**

### External Fragmenetation:

1.External Fragmentation happens when a dynamic memory allocation algorithm allocates some memory and a small piece is left over that cannot be effectively used.

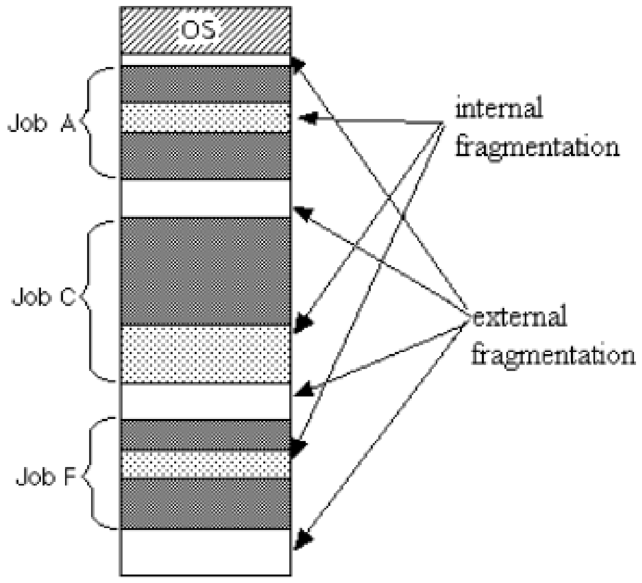
2.If too much external fragmentation occurs, the amount of usable memory is drastically reduced.Total memory space exists to satisfy a request, but it is not contiguous.



### Internal Fragmentation:

1.Internal fragmentation is the space wasted inside of allocated memory blocks because of restriction on the allowed sizes of allocated blocks.

2.Allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used.



## 4.Explain page fault? Also about bits such as modified flag or bit, dirty bit etc.

**Ans:**

### Page Fault:

**1.**A page fault occurs when the requested memory address (from the virtual address space) does not map to something that is in RAM.

2.A page need to be sent from RAM to swap, so that the requested new page can be brought from swap to RAM.

3.When the referenced page number is not in Main Memory, Page fault occurs. It decreases CPU utilization.

### Dirty Bit or Modified Bit:

1.A dirty bit or modified bit is a bit that is associated with a block of computer memory and indicates whether or not the corresponding block of memory has been modified. The dirty bit is set when the processor writes to (modifies) this memory.

2.Suppose you brought a page into memory and if you write to that page the dirty bit gets turned on . This bit is also known as modified bit since it tells whether a page has been modified . This bit increases the efficiency of demand paging.

## 5.What is Virtual Memory?

**Ans:**

1.Virtual memory is a component of most operating systems, such as MAC OS, Windows and Linux.

2. Virtual memory has a very important role in the operating system. It allows us to run more applications on the system than we have enough physical memory to support.

3. Virtual memory is simulated memory that is written to a file on the hard drive. That file is

often called page file or swap file.

4. It's used by operating systems to simulate physical RAM by using hard disk space.

5. virtual Memory is a space where large programs can store themselves in form of pages while their execution and only the required pages or portions of processes are loaded into the main memory.

6. This technique is useful as large virtual memory is provided for user programs when a very small physical memory is there.

### Benefits:

1. Large programs can be written, as virtual space available is huge compared to physical memory. Less I/O required, leads to faster and easy swapping of processes. More physical memory available, as programs are stored on virtual memory, so they occupy very less space on actual physical memory.

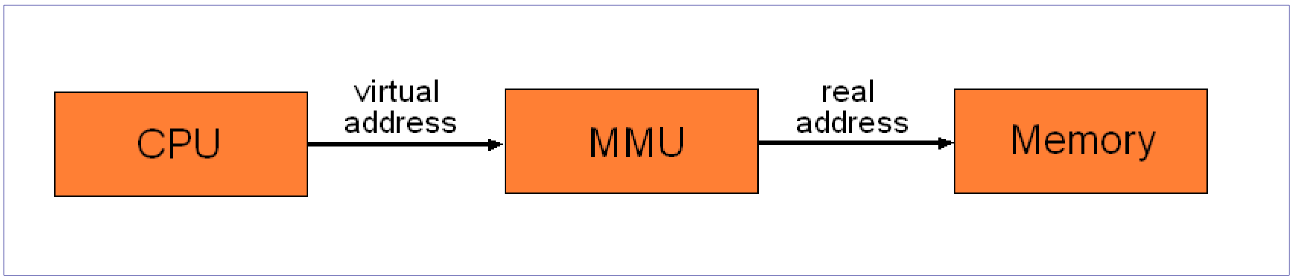
## 6.What is Physical Memory?

**Ans:**

1. Physical memory refers to the actual RAM of the system, which usually takes the form of cards (DIMMs) attached onto the motherboard.
2. Also called primary memory, it is the only storage type directly accessibly to the CPU and holds the instructions of programs to execute.
3. Physical memory is linearly addressable; memory addresses increase in a linear fashion and each byte is directly addressable.
4. Alternatively referred to as the **physical storage** or the **real storage**, **physical memory** is a term used to describe the total amount of memory installed in the computer. For example, if the computer has two 64 MB memory modules installed, it has a total of 128 MB of physical memory.

## 7.Describe virtual address translation mechanism.

### Basic Model:



1. A program is comprised of segments (data, code, stack, etc.)

2. A logical address is a pair consisting of a segment number and a byte offset (displacement) within the segment.

3. The hardware architecture defines a segment table containing one entry per segment.

4. It is normally implemented as an array indexed by segment number.
5. Each segment table entry contains the base address and limit or length of the segment.

## Logical Address:

With a virtual memory system, the main memory can be viewed as a local store for a cache level whose lower level is a disk. Since it is fully associative there is no need for a set field. The address just decomposes into an offset field and a page number field. The number of bits in the offset field is determined by the page size. The remaining bits are the page number.

### a. Page Table Base Register (PTBR)

1 Each process running on a processor needs its own logical address space. This can only be realized if each process has its own page table.

2. To support this, a processor that supports virtual memory must have a page table base register that is accessible by the operating system. For operating system security, this register is only accessible when the processor is in system mode.

3. The operating system maintains information about each process in a process control block. The page table base address for the process is stored there. The operating system loads this address into the PTBR whenever a process is dispatched.

### b. Page Table Entries

1. A page table entry contains information about an individual page in a process's logical address space. It typically has a size of 4 bytes (32 bits). It contains the following kinds of information.

2. 1 bit indicating if the entry is valid. It is valid only when the page is legal for the process and is in memory. The hardware will trigger an exception if this bit indicates the entry is not valid. The operating system needs another bit to distinguish between legal and illegal accesses.

3. 2 or 3 access control bits. These bits can control

   read access — load instructions

   write access — store instructions

   execute access — instruction fetches

4. A "dirty" bit to indicate if the page has been modified.

5. A page that has a frame allocated to it but has not been accessed for a while may need to have its frame assigned to a recently accessed page. This is called page replacement.

6. If a page is dirty and it needs to be replaced then the page needs to be written to the disk.

7. Bits used by the operating system for approximating how long it has been since the page has been accessed.

8. These bits determine good candidates for replacement. The hardware must provide some support for updating this information.

9. It can be as simple as a single bit that is set whenever a page is accessed.

10. When the hardware attempts to access memory and the valid bit is false, the hardware does not need any of the remaining information.

11. It just triggers an exception to bring up the operating system. The operating system can set the valid bit to false for pages that are swapped out to a disk.

12. It can use all of the bits except the valid bit to record the disk location.

13. With 32 bit physical addresses and 4KB pages the frame number only requires 20 bits.

14. Then a 4B page table entry provides more than enough bits.

## C.Logical Memory Access

1. A logical memory access (instruction fetch, load, or store) involves two physical memory acesses: an access to retrieve the page table entry, and an access to fetch an instruction or load or store data.

2. If nothing is done about it, this doubles the latency for logical memory accesses. All but the very earliest processors that supported virtual memory have used a translation lookaside buffer to eliminate most of the page table entry retrievals.

## D.Translation Lookaside Buffer

The translation lookaside buffer is just a cache that holds recently accessed page table entries. It can achieve very small miss rates with just a few thousand entries.

### Page Table Size:

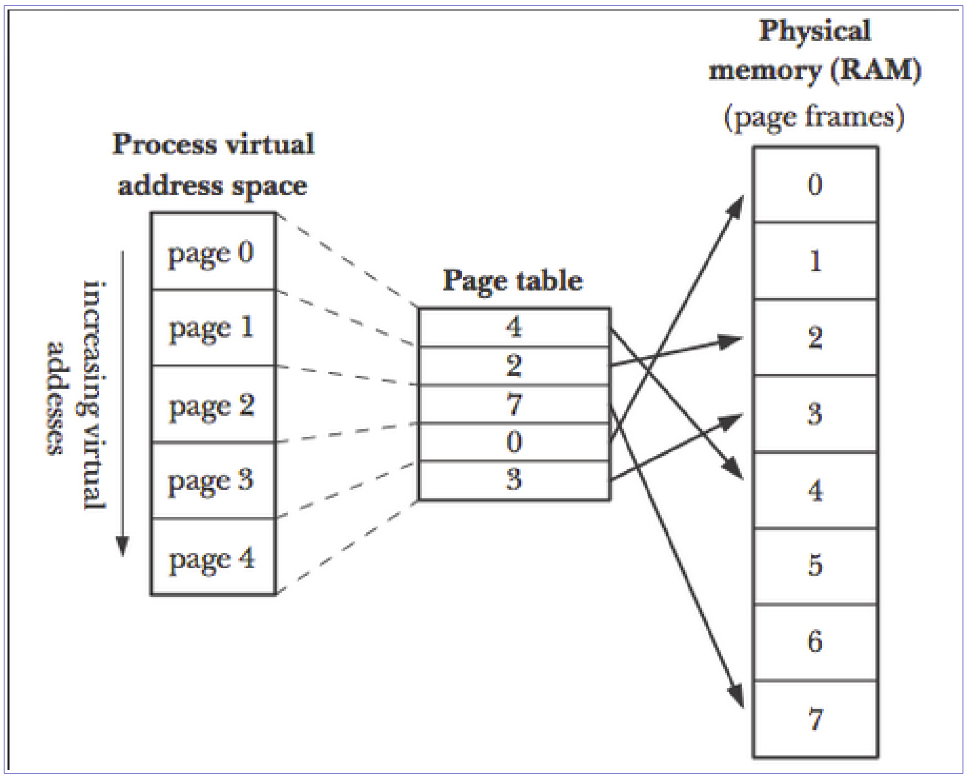
1. The size of a page table is given by the following equation.

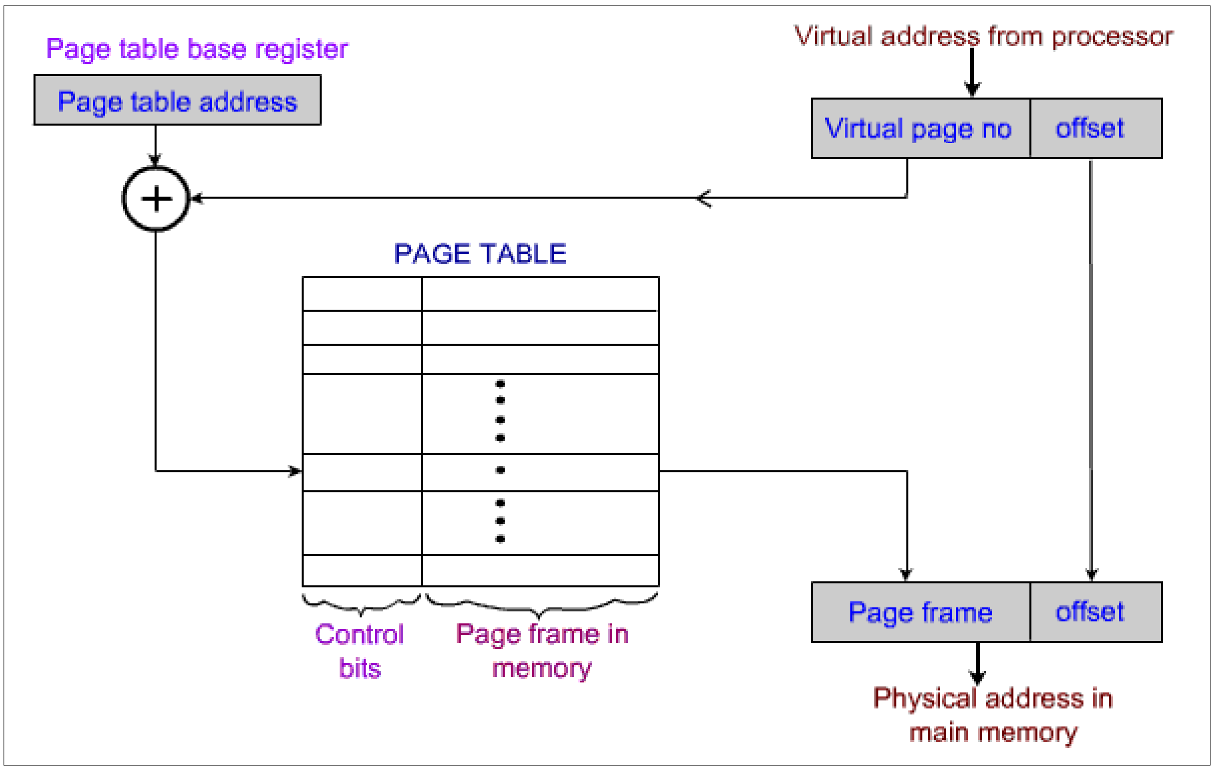
**page table size= (Logical address page size / Page size ) * Page Table Entry size**

For example, many processors have 32-bit logical addresses, which results in a 4GB logical address space size. The page table entries size is usually 4B. If the page size is 4KB then the page table size is

**page table size= (4G / 4KB) * 4B=4MB**

Most processors use multilevel page tables to reduce the size of page tables for small programs.


Physical memory (RAM) (page frames)
Process virtual address space
increasing virtual addresses
page 0
page 1
page 2
page 3
page 4
Page table
4
2
7
0
3
0
1
2
3
4
5
6
7



## 8. What is Demand Paging ?

**Ans:**

1. A demand paging system is quite similar to a paging system with swapping where processes reside in secondary memory and pages are loaded only on demand, not in advance.

2. When a context switch occurs, the operating system does not copy any of the old program's pages out to the disk or any of the new program's pages into the main memory Instead, it just begins executing the new program after loading the first page and fetches that program's pages as they are referenced.

3. While executing a program, if the program references a page which is not available in the main memory because it was swapped out a little ago, the processor treats this invalid memory reference as a page fault and transfers control from the program to the operating system to demand the page back into the memory.

## 9.Difference between swap in/out and page in/out

**Ans:**

### Page in and Page Out:

1. When pages are written to disk, the event is called a page-out, and when pages are returned to physical memory, the event is called a page-in.

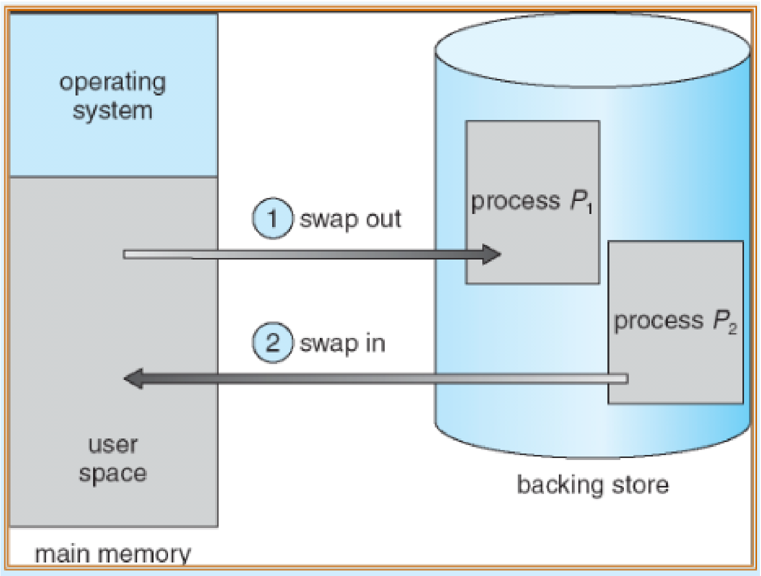
2. A page fault occurs when the kernel needs a page, finds it doesn't exist in physical memory because it has been paged-out, and re-reads it in from disk.

3. Page-ins are common, normal and are not a cause for concern. For example, when an application first starts up, its executable image and data are paged-in. This is normal behavior.

4. Page-outs, however, can be a sign of trouble. When the kernel detects that memory is running low, it attempts to free up memory by paging out. Though this may happen briefly from time to time, if page-outs are plentiful and constant, the kernel can reach a point where it's actually spending more time managing paging activity than running the applications, and system performance suffers. This woeful state is referred to as thrashing.

### Swap in and Swap out:

1. A process must be in physical memory for execution. A process can be swapped temporarily out of physical memory to a backing store and then brought back into memory for continued execution.



2. This applies specifically to multitasking environments where multiple processes are to be executed at the same time, and hence a cpu scheduler is implemented to decide which process to swap to the backing store.

## 10.Explain about kernel memory allocation and buddy system

**Ans:**

### Kernel Memory Allocation:

T1. here is also additional memory allocated to the kernel, however, which cannot be so easily paged.

2. Some of it is used for I/O buffering and direct access by devices, example, and must therefore be contiguous and not affected by paging.

3. Other memory is used for internal kernel data structures of various sizes, and since kernel memory is often locked (restricted from being ever swapped out), management of this resource must be done carefully to avoid internal fragmentation or other waste. (I.e. you would like the kernel to consume as little memory as possible, leaving as much as possible for user processes.)

4. Accordingly there are several classic algorithms in place for allocating kernel memory structures.

5. Allocating memory in the kernel is not as simple as allocating memory in user space.

6. A number of factors contribute to the complication, among them:

- The kernel is limited to about 1GB of virtual and physical memory.
- The kernel's memory is not pageable.
- The kernel usually wants physically contiguous memory.
- Often, the kernel must allocate the memory without sleeping.

- Mistakes in the kernel have a much higher price than they do elsewhere.

The general interface for allocating memory inside of the kernel is kmalloc()

```
#include <linux/slab.h>
void * kmalloc(size_t size, int flags);
```

## Buddy System:

1. The Buddy System allocates memory using a power of two allocator.

2. Under this scheme, memory is always allocated as a power of 2 ( 4K, 8K, 16K, etc ), rounding up to the next nearest power of two if necessary.

3. If a block of the correct size is not currently available, then one is formed by splitting the next larger block in two, forming two matched buddies. ( And if that larger size is not available, then the next largest available size is split, and so on. )

4. One nice feature of the buddy system is that if the address of a block is exclusively ORed with the size of the block, the resulting address is the address of the buddy of the same size, which allows for fast and easy coalescing of free blocks back into larger blocks.

- Free lists are maintained for every size block.
- If the necessary block size is not available upon request, a free block from the next largest size is split into two buddies of the desired size. ( Recursively splitting larger size blocks if necessary. )

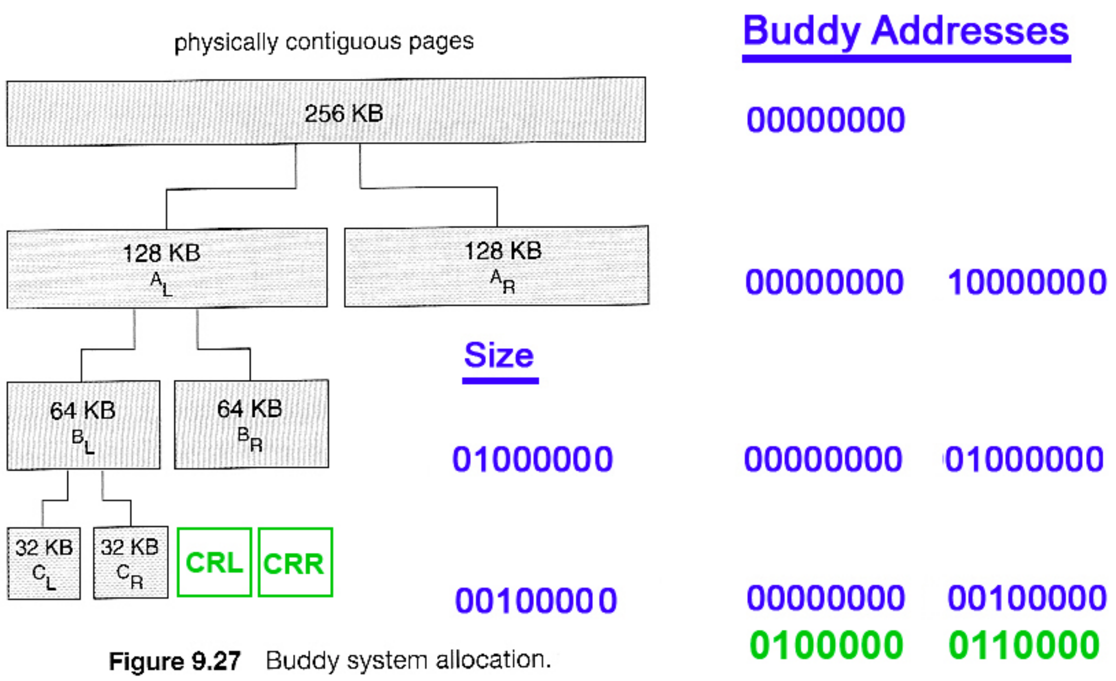


**Figure 9.27** Buddy system allocation.

- When a block is freed, its buddy's address is calculated, and the free list for that size block is checked to see if the buddy is also free. If it is, then the two buddies are coalesced into one larger free block, and the process is repeated with successively larger free lists.
-

## 11. What is meant by Thrashing ?

### Ans:

1. Thrashing in computing is an issue caused when virtual memory is in use.

2. It occurs when the virtual memory of a computer is rapidly exchanging data for data on hard disk, to the exclusion of most application-level processing.

3. As the main memory gets filled, additional pages need to be swapped in and out of virtual memory.

4. The swapping causes a very high rate of hard disk access. Thrashing can continue for a long duration until the underlying issue is addressed.

5. Thrashing can potentially result in total collapse of the hard drive of the computer.Thrashing is also known as disk thrashing.

## 12.What you understood from /proc/slabinfo and slabtop commands

**Ans:**

### proc/slabinfo:

1. This file gives full information about memory usage on the *slab* level. Linux kernels greater than version 2.2 use *slab pools* to manage memory above the page level. Commonly used objects have their own slab pools.

2. Instead of parsing the highly verbose /proc/slabinfo file manually, the /usr/bin/slabtop program displays kernel slab cache information in real time. This program allows for custom configurations, including column sorting and screen refreshing.

3. Some of the more commonly used statistics in /proc/slabinfo that are included into /usr/bin/slabtop include:

- OBJS — The total number of objects (memory blocks), including those in use (allocated), and some spares not in use.
- ACTIVE — The number of objects (memory blocks) that are in use (allocated).
- USE — Percentage of total objects that are active. ((ACTIVE/OBJS)(100))
- OBJ SIZE — The size of the objects.
- SLABS — The total number of slabs.
- OBJ/SLAB — The number of objects that fit into a slab.
- CACHE SIZE — The cache size of the slab.
- NAME — The name of the slab.

### Slabtop :

1. slabtop displays detailed kernel slab cache information in real time. It displays a listing of the top caches sorted by one of the listed sort criteria. It also displays a statistics header filled with slab layer information.

2. slabtop allows to interrogate /proc/slabinfo.

3. The Linux kernel needs to allocate memory for temporary objects such as task or device structures and inodes.

4. The caching memory allocator manages caches of these types of objects. The modern Linux kernel implements this caching memory allocator to hold the caches called the slabs.

5. Different types of slab caches are maintained by the slab allocator

## 13.Write program to get resource limits using getrlimit() system call and analyze the resource limits. Check with all resource limits available.

**Ans:**

```
#include <stdio.h>

#include <stdlib.h>

extern int etext, edata, end;

int main()

{

        printf("Addr etext: %p\t", &etext);

        printf("Addr edata: %p\t", &edata);

        printf("Addr end: %p\n", &end);

        char *s1 = "hello"; //in initialized data segment

        static int v1=1; //in initialized data segment

        static int v2; //in uninitialized data segment

        char s2[] = "hello"; //in the stack area.

        int * dynmem = malloc(4);
        printf("Initialized Data Segment %p\n", s1);

        printf("Initialized Data Segment %p\n", &v1);

        printf("Uninitialized Data Segment %p\n", &v1);

        printf("Stack Area%p\n", s2);

        printf("DMA %p\n", dynmem);

return 0;

}
```

**Output:**

```
Addr etext: 0x40075d        Addr edata: 0x60104c        Addr end: 0x601058

Initialized Data Segment 0x400796

Initialized Data Segment 0x601048

Uninitialized Data Segment 0x601048

Stack Area0x7ffd2bc76ae0

DMA 0xc11420
```

## 14. Do some analysis on vmstat with options –s, -a, -f, -m (need to be su for this), -n etc (-h gives all details)

**Ans:**

vmstat - Report virtual memory statistics

```
SYNOPSIS
       vmstat [options] [delay [count]]

DESCRIPTION
       vmstat  reports  information about processes, memory, paging, block IO,
       traps, disks and cpu activity.

       The first report produced gives averages since the last reboot.   Addi-
       tional  reports  give information on a sampling period of length delay.
       The process and memory reports are instantaneous in either case.

OPTIONS
```

delay The delay between updates in seconds. If no delay is specified,
only one report is printed with the average values since boot.

count Number of updates. In absence of count, when delay is defined,
default is infinite.

-a, --active
Display active and inactive memory, given a 2.5.41 kernel or
better.

-f, --forks
The -f switch displays the number of forks since boot. This
includes the fork, vfork, and clone system calls, and is equiva-
lent to the total number of tasks created. Each process is rep-
resented by one or more tasks, depending on thread usage. This
display does not repeat.

-m, --slabs
Displays slabinfo.

-n, --one-header
Display the header only once rather than periodically.

-s, --stats
Displays a table of various event counters and memory statis-
tics. This display does not repeat.

-h, --help
Display help and exit.

-d, --disk
Report disk statistics (2.5.70 or above required).

-D, --disk-sum
Report some summary statistics about disk activity.

-p, --partition device
Detailed statistics about partition (2.5.70 or above required).

-S, --unit character
Switches outputs between 1000 (k), 1024 (K), 1000000 (m), or
1048576 (M) bytes. Note this does not change the swap (si/so)
or block (bi/bo) fields.

-w, --wide
Wide output mode (useful for systems with higher amount of mem-

ory,  where the default output mode suffers from unwanted column breakage).  The output is wider than 80 characters per line.

-V, --version

# File System

### 1. What is File ?

Ans:

1. Computers can store information on various storage media, such as magnetic disks, magnetic tapes, and optical disks. So that the computer system will be convenient to use, the operating system provides a uniform logical view of stored information. The operating system abstracts from the physical properties of its storage devices to define a logical storage unit, the file.
2. The file system consists of two distinct parts: a collection of files, each storing related data, and a directory structure, which organizes and provides information about all the files in the system.
3. File systems live on devices.
4. Files are mapped by the operating system onto physical devices.
5. A file is a named collection of related information that is recorded on secondary storage.
6. The information in a file is defined by its creator. It may be

   **a. Text File b. Source File c. Executable File**
7. A text file is a sequence of characters organized into lines (and possibly pages)
8. A source file is a sequence of functions, each of\ which is further organized as declarations followed by executable statements.
9. An executable file is a series of code sections that the loader can bring into memory and execute.

### 2. What are various File Attributes ?

**Ans:**

1. A file is named, for the convenience of its human users, and is referred to by its name.
2. A file’s attributes vary from one operating system to another but typically consist of these:

**• Name. The symbolic file name is the only information kept in human-readable form.**

**• Identifier. This unique tag, usually a number, identifies the file within the file system; it is the non-human-readable name for the file.**

**• Type. This information is needed for systems that support different types of files.**

**• Location. This information is a pointer to a device and to the location of the file on that device.**

**• Size. The current size of the file (in bytes, words, or blocks) and possibly the maximum allowed size are included in this attribute.**

**• Protection. Access-control information determines who can do reading, writing, executing, and so on.**

**• Time, date, and user identification. This information may be kept for creation, last modification, and last use. These data can be useful for protection, security, and usage monitoring.**

### 3. What are various File Operations ?

**Ans:**

1. A file is an abstract data type.

2. The operating system can provide system calls to create, write, read, reposition, delete, and truncate files. Basic file operation are:

**• Creating a file**

**• Writing a file**

**• Reading a file**

**• Repositioning within a file**

**• Deleting a file**

**• Truncating a file**

### 5. What are various File Access method ? Explain in brief .

**Ans:**

**1. Files store information. When it is used, this information must be accessed and read into computer memory.**

- Sequential Access
- Direct Access

**Sequential Access:**The simplest access method is sequential access.nformation in the file is processed in order, one record after the other. This mode of access is by far the most common; for example, editors and compilers usually access files in this fashion.

**Direct Access:** It is also called as a relative access.Here, a file is made up of fixed-length logical records that allow programs to read and write records rapidly in no particular order. The direct-access method is based on a disk model of a file, since disks allow random access to any file block. For Direct access, the file is viewed as a numbered sequence of blocks or records. Thus, we may read block 14, then read block 53, and then write block 7. There are no restrictions on the order of reading or writing for a direct-access file.Direct-access files are of great use for immediate access to large amounts of information. Databases are often of this type. When a query concerning a particular subject arrives, we compute which block contains the answer and then read that block directly to provide the desired information.

## 6. What is File System Mounting ? Explain in brief.

**Ans:**

1. Just as a file must be opened before it is used, a file system must be mounted before it can be available to processes on the system. More specifically, the directory structure may be built out of multiple volumes, which must be mounted to make them available within the file-system name space.
2. The mount procedure is straightforward. The operating system is given the name of the device and the mount point—the location within the file structure where the file system is to be attached.
3. Typically, a mount point is an empty directory. For instance, on a UNIX system, a file system containing a user's home directories might be mounted as /home ; then, to access the directory structure within that file system, we could precede the directory names with /home , as in /home/jane .
4. Mounting that file system under /users would result in the path name /users/jane , which we could use to reach the same directory.

## 7. What is File Protection ? Explain in brief .

**Ans:**

1. When information is stored in a computer system, we want to keep it safe from physical damage (the issue of reliability) and improper access (the issue of protection).
2. Reliability is generally provided by duplicate copies of files.
3. Bugs in the file-system software can also cause file contents to be lost.
4. Protection mechanisms provide controlled access by limiting the types of file access that can be made. Access is permitted or denied depending on several factors, one of which is the type of access requested.
5. Access control:

- **Owner. The user who created the file is the owner.**
- **Group. A set of users who are sharing the file and need similar access is a group, or work group.**
- **Universe. All other users in the system constitute the universe.**

## 8. What is File System ? Draw File System Structure.

**Ans:**

1. A file system is an organized collection of regular files and directories. A file system is created using the mkfs command.

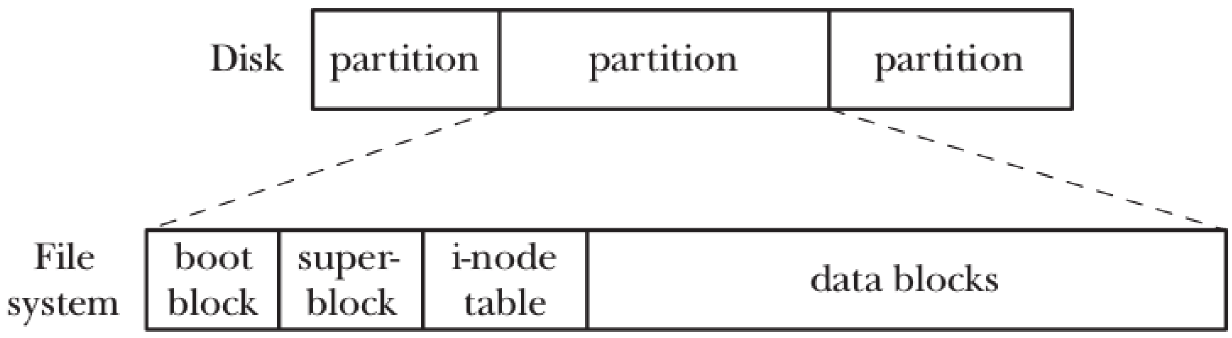
2. One of the strengths of Linux is that it supports a wide variety of file systems, including the following:

- **Traditional ext2 file system**
- **Various native UNIX file systems such as the Minix, System V, and BSD file systems**
- **Microsoft's FAT, FAT32, and NTFS file systems**
- **ISO 9660 CD-ROM file system;**
- **Range of journaling file systems, including ext3, ext4, Reiserfs, JFS, XFS, and Btrfs**

### File System Structure:

1. The basic unit for allocating space in a file system is a logical block, which is some multiple of contiguous physical blocks on the disk device on which the file system resides.

2. For example, the logical block size on ext2 is 1024, 2048, or 4096 bytes.



As show in above diagram hows the relationship between disk partitions and file systems, and shows the parts of a (generic) file system.

## 9. What is Boot Block, Super Block, i-Node Table and Data Block.

**Ans:**

**Boot Block:**

This is always the first block in a file system. The boot block is not used by the file system; rather, it contains information used to boot the operat- ing system. Although only one boot block is needed by the operating system, all file systems have a boot block (most of which are unused).

**Super Block:**

This is a single block, immediately following the boot block, which contains parameter information about the file system, including:

- **size of the i-node table;**
- **size of logical blocks in this file system**
- **size of the file system in logical blocks**

Different file systems residing on the same physical device can be of different types and sizes, and have different parameter settings (e.g., block size). This is one of the reasons for splitting a disk into multiple partitions.

**I-Node Table:**

1. Each file or directory in the file system has a unique entry in the i-node table. This entry records various information about the file. I-nodes are dis- cussed in greater detail in the next section. The i-node table is sometimes also called the i-list.

2. A file system's i-node table contains one i-node (short for index node) for each file residing in the file system.

3. I-nodes are identified numerically by their sequential location in the i-node table.

4. The i-node number (or simply i-number) of a file is the first field displayed by the ls –li command.

5. The information maintained in an i-node includes:

- File type (e.g., regular file, directory, symbolic link, character device).
- Owner (also referred to as the user ID or UID) for the file.
- Group (also referred to as the group ID or GID) for the file.

- Access permissions for three categories of user: owner (sometimes referred as user), group, and other
- Three timestamps: time of last access to the file (shown by ls –lu), time of last modification of the file (the default time shown by ls –l), and time of last status change (last change to i-node information, shown by ls –lc). As on other UNIX implementations, it is notable that most Linux file systems don't record the creation time of a file.
- Number of hard links to the file
- Size of the file in bytes
- Number of blocks actually allocated to the file
- Pointers to the data blocks of the file.

**Data Block:**

The great majority of space in a file system is used for the blocks of data that form the files and directories residing in the file system.

## 10. What is Virtual File System ?

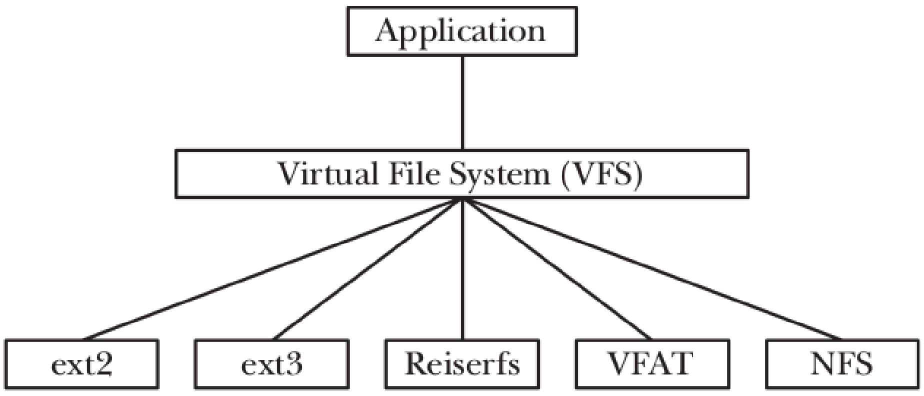
**Ans:**

1. Each of the file systems available on Linux differs in the details of its implementation. Such differences include, for example, the way in which the blocks of a file are allocated and the manner in which directories are organized. If every program that worked with files needed to understand the specific details of each file system, the task of writing programs that worked with all of the different file systems would be nearly impossible.

2. The virtual file system (VFS, sometimes also referred to as the virtual file switch) is a kernel feature create a abstraction layer for file system operation.

3. The VFS defines a generic interface for file-system operations. All programs that work with files specify their operations in terms of this generic interface.

4. Each file system provides an implementation for the VFS interface.



5. The VFS interface includes operations corresponding to all of the usual system calls for working with file systems and directories, such as open(), read(), write(), lseek(), close(), truncate(), stat(), mount(), umount(), mmap(), mkdir(), link(), unlink(), symlink(), and rename().

**11.How do you create special files like named pipes and device files?**

**Ans:**

The system call mknod creates special files in the following sequence.

1. kernel assigns new inode,
2. sets the file type to indicate that the file is a pipe, directory or special file,
3. If it is a device file, it makes the other entries like major, minor device numbers.

**For example:**

If the device is a disk, major device number refers to the disk controller and minor device number is the disk.

**12. What do you understand by VFS? Not simple abbreviation, provide the basic understanding.**

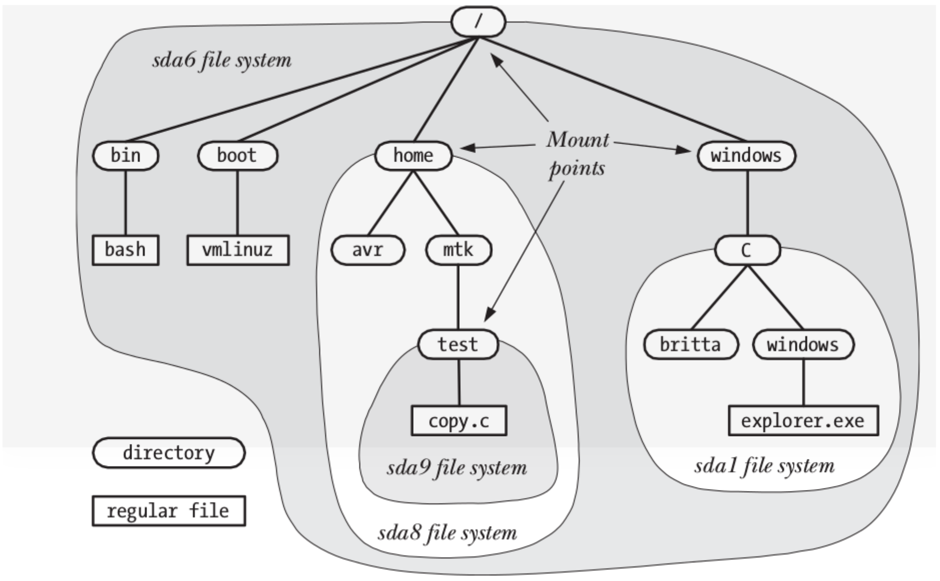
**Ans:**

1. In Linux, all files are accessed through the Virtual Filesystem Switch, or VFS.

2. This is a layer of code which implements generic filesystem actions and vectors requests to the correct specific code to handle the request.

3. Two main types of code modules take advantage of the VFS services, device drivers and filesystems.

**13. What is Mount Point ? Explain mounting and Unmounting File system ?**



1. The mount() and umount() system calls allow a privileged ( CAP_SYS_ADMIN ) process to mount and unmount file systems.

2. A list of the currently mounted file systems can be read from the Linux-specific proc/mounts virtual file. /proc/mounts is an interface to kernel data structures, so it always contains accurate information about mounted file systems.

3. The mount(8) and umount(8) commands automatically maintain the file /etc/mtab , which contains information that is similar to that in /proc/mounts , but slightly more detailed. In particular, /etc/mtab includes file system–specific options given to mount(8), which are not shown in /proc/mounts . However, because the mount() and umount() system calls don’t update /etc/mtab , this file may be inaccurate if some application that mounts or unmounts devices fails to update it.

4. The /etc/fstab file, maintained manually by the system administrator, contains descriptions of all of the available file systems on a system, and is used by the mount(8), umount(8), and fsck(8) commands.

5.  /etc/fstab file, maintained manually by the system administrator, contains descriptions of all of the available file systems on a system, and is used by the mount(8), umount(8), and fsck(8) commands.

6. The mount() system call mounts the file system contained on the device specified by source under the directory (the mount point) specified by target.

**14. What do you understand by VNODE? How it is different from inode?**

**Ans:**

1. A vnode is an in-memory structure that abstracts much of what an inode is (an inode can be one of its data fields) but also captures things like operations on files, locks, etc.
2. The inode is a data structure in a Unix-style file system that describes a filesystem object such as a file or a directory. Each inode stores the attributes and disk block location(s) of the object's data.
3. A directory contains an entry for itself, its parent, and each of its children.

**15. What is meant by file system? What are the different file systems supported in Linux?**

**Ans:** A file system is an organized collection of regular files and directories. A file system is created using the mkfs command.. Linux supports numerous file systems, but common choices for the system disk on a block device include the ext* family (ext2, ext3 and ext4), XFS, JFS, ReiserFS and btrfs.

**16. what is swap partition?**

**Ans:**

1. The swap partition is an independent section of the hard disk used solely for swapping; no other files can reside there.
2. To see what swap space you have, use the command swapon -s. The output will look something like this:

```
Filename     Type         Size       Used   Priority

/dev/sda5    partition    859436     0      -1
```

You can verify that it is being used by running swapon -s.

To mount the swap space automatically at boot time, you must add an entry to the /etc/fstab file, which contains a list of filesystems and swap spaces that need to be mounted at boot up.

**17. What is use of the commands df–T, mount, file, fsck? Also what is the purpose of /etc/fstab?**

**Ans:**

1. df -T:The -T option in the df command displays the file system type.

2. Mount:An existing directory on which to mount the file system.

3. file: As root, use the file command ,You need to pass the individual device name to the file command.

4. Fsck: This will display the file system type of a given device.

5. /etc/fstab: If a particular mount point is configured to be mounted automatically during system startup, you can identify its file system type by looking at the /etc/fstab file.

**18.The file system contains boot block, super block, inode table and data blocks. Explain this in more detail?**

**Ans:**

**1 .**Overview The structure of a simple **Unix** file system can generally be seperated into four parts, the boot **block**, the super **block**, the inode list and the **data blocks**. The inode list (table) is a list of inodes that are used to track and maintain information about each file created on the filesystem.

2. A superblock is a record of the characteristics of a filesystem, including its size, the block size, the empty and the filled blocks and their respective counts, the size and location of the inode tables, the disk block map and usage information, and the size of the block groups.

3. Each filesystem contains: 1. a boot block located in the first few sectors of a file system. The boot block contains the initial bootstrap program used to load the operating system. Typically, the first sector contains a bootstrap program that reads in a larger bootstrap program from the next few sectors, and so forth.

4. The inode is a data structure in a Unix-style file system that describes a filesystem object such as a file or a directory. Each inode stores the attributes and disk block location(s) of the object's data.

**19. Assuming total 15 blocks out of which 12 are direct and 3 are indirect (single, double, triple), block size is 4k, Can you calculate how much maximum size of file you can store?**

**Ans:** 12 direct blocks

1 single block

1 double block

1 triple block

Assuming 32 bit addresses,we have 4k per block,

4k/4=1024bytes

single : 1024*1k=1MB

double: 1024 * 1024*1k=1Gb

triple : 1024*1024*1024*1k=1TB

**Total size :1TB**

## 20. Explain about ext2, ext3 and ext4 file systems using in Linux?

**Ans:**

### 1. Ext2:

- Ext2 stands for second extended file system.
- It was introduced in 1993. Developed by Rémy Card.
- This was developed to overcome the limitation of the original ext file system.
- Ext2 does not have journaling feature.
- On flash drives, usb drives, ext2 is recommended, as it doesn't need to do the over head of journaling.
- Maximum individual file size can be from 16 GB to 2 TB
- Overall ext2 file system size can be from 2 TB to 32 TB

### 2. Ext3

- Ext3 stands for third extended file system.
- It was introduced in 2001. Developed by Stephen Tweedie.
- Starting from Linux Kernel 2.4.15 ext3 was available.
- The main benefit of ext3 is that it allows journaling.
- Journaling has a dedicated area in the file system, where all the changes are tracked. When the system crashes, the possibility of file system corruption is less because of journaling.
- Maximum individual file size can be from 16 GB to 2 TB
- Overall ext3 file system size can be from 2 TB to 32 TB
- There are three types of journaling available in ext3 file system.
  - Journal – Metadata and content are saved in the journal.
  - Ordered – Only metadata is saved in the journal. Metadata are journaled only after writing the content to disk. This is the default.
  - Writeback – Only metadata is saved in the journal. Metadata might be journaled either before or after the content is written to the disk.
- You can convert a ext2 file system to ext3 file system directly (without backup/restore).

### 3. Ext4

- Ext4 stands for fourth extended file system.
- It was introduced in 2008.
- Starting from Linux Kernel 2.6.19 ext4 was available.
- Supports huge individual file size and overall file system size.
- Maximum individual file size can be from 16 GB to 16 TB
- Overall maximum ext4 file system size is 1 EB (exabyte). 1 EB = 1024 PB (petabyte). 1 PB = 1024 TB (terabyte).
- Directory can contain a maximum of 64,000 subdirectories (as opposed to 32,000 in ext3)
- You can also mount an existing ext3 fs as ext4 fs (without having to upgrade it).
- Several other new features are introduced in ext4: multiblock allocation, delayed allocation, journal checksum. fast fsck, etc. All you need to know is that these new features have improved the performance and reliability of the filesystem when compared to ext3.
- In ext4, you also have the option of turning the journaling feature “off”.

## 21. What is the use of /etc/mtabfile and sysctl command?

**Ans:**

**1.** The /etc/mtab file is the list of mounted file systems it is maintained by the mount and unmount programs. It's format is similar to the fstab file The columns arw

- **device** the device or remote filesystem that is mounted.
- **mountpoint** the place in the filesystem the device was mounted.
- **filesystemtype** the type of filesystem mounted.
- **options** the mount options for the filesystem
- **dump** used by dump to decide if the filesystem needs dumping.
- **fsckorder** used by fsck to detrmine the fsck pass to use.

# Process Management

## 1. What are the process states available in this process management sub-system?

**Ans :**

**1.** As a process executes, it changes its state. The states of the process are New, Running, Waiting, Ready and Terminated.

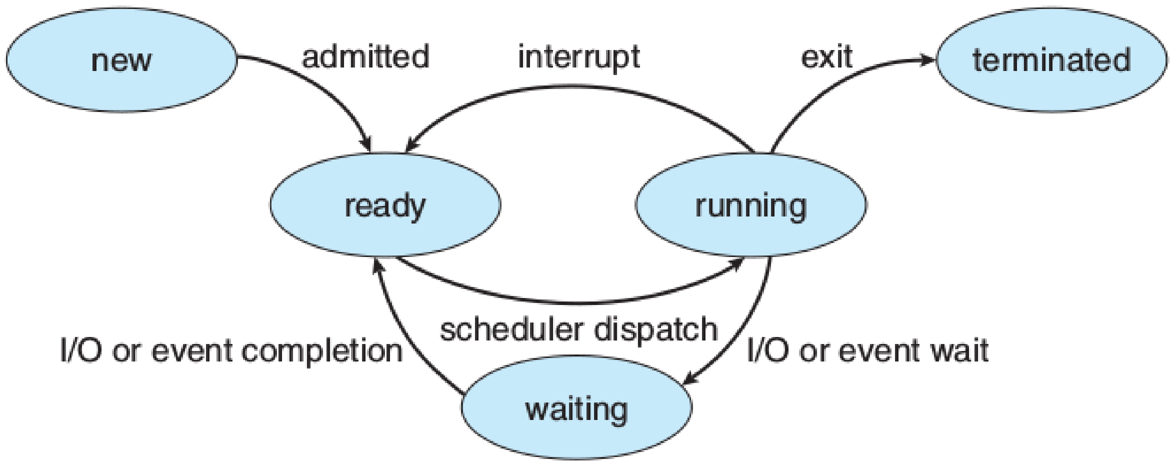
2. A process changes its state from Running to Ready (or vice versa) because of the process scheduler. A process switches from Waiting to Ready, if the I/O it has been waiting is completed. Process goes from Running to Waiting if it requests for an I/O.

3. Suppose we have three process A, B, C, that all are in Ready State. The scheduler picks up process A, and it starts to Run it (in Running State). Suppose Process makes a request fro I/O and therefore goes into Waiting State.

4. Now the scheduler picks up Process B and it is in Running state. Now suppose that the I/O for Process A is complete.

5. The I/O device will send an interrupt to CPU, saying that it has completed the I/O. As a result Process B is preempted and is pushed back into Ready State.

6. Using the Interrupt Service Routine, CPU handles the interrupt and as a result puts process A back into the Ready Queue.



7. Now the scheduler picks a process from the Ready Queue according to its scheduling policy (it may not be Process A!) and that process is in the running state.

## 2. What is the first process in the Kernel? What is the purpose of this process?

**Ans :**

**1.** Typically it is the init process, the path of which is hard coded into the kernel itself. init performs very low level functions like starting upstart in the case of Ubuntu or systemd in the case of Arch, Fedora, and others, which load the remaining processes and setup.

2. Init is the father of all processes. Its primary role is to create processes from a script stored in the file /etc/inittab. This file usually has entries which cause init to spawn gettys on each line that users can log in. It also controls autonomous processes required by any particular system.

## 3. What are different types of OS scheduling algorithms available?

**Ans :**

1. When multiple process executing at the same time then CPU has to decide which process has to execute first.

2. There are four scheduling algorithm:

   **1. First-Come First-Serve [FCFS]**

   **2. Shortest Job First [SJF]**

   **3. Priority Scheduling**

   **4. Round Robin**

3. In FCFS which process is coming first that will execute first.

4. In SJF which process require less time to execute that process will execute first.

5. In priority scheduling , execution of process will be according to priority.

6. In Round-Robin algorithm each process has a slice of time to execute.

## 4. What is process kernel stack and process user stack? What is the size of each and how they are allocated?

**Ans :**

**1.** There is one "kernel stack" per CPU. There is one "user stack" for each process, though each thread has its own stack, including both user and kernel threads.

2. In Linux system every user process has 2 stacks , a user stack and a dedicated kernel stack for the process.

3. User stacks resides in the user address space(First 3GB in 32-bit x86 arch.) and kernel stack resides in the kernel address space (3GB-4GB in 32bit-x86) of the process. When a user process needs to execute some privileged instruction (A system call) it traps to kernel mode and kernel executes the it on behalf of the user process.

5. This execution takes place on the processes' kernel stack.

## 5. Why do we need separate kernel stack for each process?

**Ans :** It simplifies pre-emption of processes in the kernel space.

## 6. List the system calls used for process management.

**Ans :**

System calls used for Process management:

| | |
|---|---|
| • Fork () | - Used to create a new process |
| • Exec() | - Execute a new program |
| • Wait() | - wait until the process finishes execution |
| • Exit() | - Exit from the process |
| • Getpid() | - get the unique process id of the process |
| • Getppid() | - get the parent process unique id |
| • Nice() | - to bias the existing property of process |

## 7. What happens when you execute a program and when you execute a command ?

**Ans :**

1. When a program is executed it becomes a process and is allocated its own process control block.

2. A command is usually associated with the shell(eg. Bash) and hence exerts its footprint on the process control block of the shell it is being run on.

## 8. What do you understand by short term scheduler and long term scheduler?

**Ans :**

**1. Short Term Scheduler -** It is also called as CPU scheduler. Its main objective is to increase system performance in accordance with the chosen set of criteria.

2. It is the change of ready state to running state of the process. CPU scheduler selects a process among the processes that are ready to execute and allocates CPU to one of them.

3. Short-term schedulers, also known as dispatchers, make the decision of which process to execute next. Short-term schedulers are faster than long-term schedulers.

**4. Long Term Scheduler -** It is also called a job scheduler. A long-term scheduler determines which programs are admitted to the system for processing.

5. It selects processes from the queue and loads them into memory for execution. Process loads into the memory for CPU scheduling.

## 9. What do you understand by PROC file system? Also, what are the commands related to process management system.

**Ans :**

1. The proc filesystem(procfs) is a special filesystem in UNIX like operating systems that presents about processes and other system information in a hierarchical file-like structure, providing a more convinient and standardized method for dynamically accessing process data held in he kernel than traditional tracing methtods or direct access to kernel memory.

2. Typically it is mapped to a mount point called /proc at the boot time. The proc file system acts as an interface to the internal data structures in the kernel.

3. It can be used to obtain informtion about the system and to change certain kernel parameters at runtime. The commands related to process management in Linux are - ps, pstree, kill, df, free, etc.

## 10. What do you understand by various processes zombie, orphan and daemon processes? What are the usual daemon (Disk and Execution Monitor) processes in the system?

**Ans :**

1. On Unix and Unix-like computer operating systems, a **zombie process** or defunct process is a process that has completed execution but still has an entry in the process table.

2. This entry is still needed to allow the parent process to read its child’s exit status. The term zombie process derives from the common definition of zombie — an undead person.

3. In the term’s metaphor, the child process has “died” but has not yet been “reaped”. Also, unlike normal processes, the kill command has no effect on a zombie process.

4. An **orphan process** is a computer process whose parent process has finished or terminated, though it remains running itself.

5. In a Unix-like operating system any orphaned process will be immediately adopted by the special init system process.

6. This operation is called re-parenting and occurs automatically. Even though technically the process has the init process as its parent, it is still called an orphan process since the process that originally created it no longer exists.

7. In Unix and other multitasking computer operating systems, a **daemon** is a computer program that runs as a background process, rather than being under the direct control of an interactive user. 8. Typically daemon names end with the letter d: for example, syslogd is the daemon that implements the system logging facility and sshd is a daemon that services incoming SSH connections.**Daemon process is a process orphaned intentionally.**

## 11. What do you understand by terms context switching, pre-emptive scheduling and non- preemptive scheduling?

**Ans :**

**1.** In computing, **context switching** is the process of storing and restoring the state (more specifically, the execution context) of a process or thread so that execution can be resumed from the same point at a later time.

2. A scheduling discipline is preemptive if, once a process has been given the CPU can taken away. The strategy of allowing processes that are logically runable to be temporarily suspended is called **Preemptive Scheduling.**

3. When a process enters the state of running, the state of that process is not deleted from the scheduler until it finishes its service time.

**4. Non-preemptive scheduling** is designed so that once a process enters the running state(is allowed a process), it is not removed from the processor until it has completed its service time ( or it explicitly yields the processor).

5. context_switch() is called only when the process terminates or blocks.

## 12. What is time slice? Explain PCB?

**Ans :**

**1.** The period of **time** for which a process is allowed to run in a preemptive multitasking system is generally called the **time slice** or quantum.

**2. Process Control Block** (PCB, also called Task Controlling Block, Entry of the Process Table, Task Struct, or Switchframe) is a data structure in the operating system kernel containing the information needed to manage a particular process.

3. The PCB is "the manifestation of a process in an operating system".

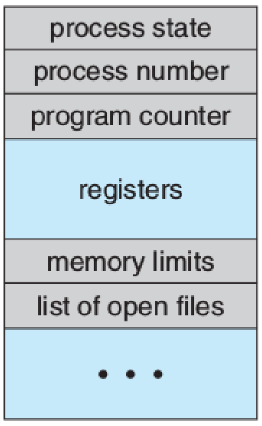


**Fig : Representation of a process control block**

### 13. What is Schedular ?

**Ans:**

1.Schedulers in Operating System are the process which decides which task and process should be accessed and run at what time by the system resources.

### 14. What is Context switch ?

**Ans:**

1. In computing, a context switch is the process of storing and restoring the state (more specifically, the execution context) of a process or thread so that execution can be resumed from the same point at a later time.

2. Switching the CPU to another process requires performing a state save of the current process and a state restore of a different process. This task is known as a context switch.

### 15. Explain Interprocess Communication Mechanism.

**Ans:**

1. There are total 4 IPC

  **1. Message Queue**

  **2. Shared Memory**

  **3. Pipe**

  **4. FIFO**

  **5. Semaphore**

1. In message queue the process memory allocation in is kernel space.  So protection is there, but the execution speed is slow.
2. In shared memory the process memory allocation is user space, so the speed of execution is very fast but no protection is there.
3. Pipe is related process, i.e. only related process can communicate with each other. Only half duplex communication is possible. It is created by using pipe() system call.
4. FIFO is also called as named pipe, it is created by using mknode() of fifo system call.
5. Semaphore is used for synchronization between the processes.

# Device Management

## 1. What is purpose of DMA (Direct Memory Access)?

### Ans:

**1.** DMA controller allows direct data transfer between the device and main memory without involving the processor.

2. DMA module MA request to the processor. DMA module transfer the entire block of data , one word at a time directly to or from memory, without involving the processor.

## 2. What happen when interrupt is raised by any Device ? Explain interrupt handling mechanism in linux.

### Ans:

1. An interrupt is an unexpected hardware initiated subroutine call or jump that temporarily suspends the running of the current program.

2. Whenever device driver raise interrupt, isr should process fast and respond to interrupt immediately.

3. After creating data structure for device that called top half and later process other works related to driver in bottom half.

4. Lengthy tasks inside interrupt handler degrade system responsiveness.

**a. Top Half:** Perform time critical tasks such as acknowledging receipt of interrupt, reseting hardware etc. It is the routine that actually respond to the interrupt –the one you register with request_irq().

**b. Bpttom Half:** Perform any interrupt related work not performed by interrupt handler. he bottom half is a routine that is scheduled by the top half to be executed later, at a safer time.A bottom half is a low-priority function, usually related to interrupt handling, that is waiting for the kernel to find a convenient moment to run it.

5. The big difference between the top-half handler and the bottom half is that all interrupts are enabled during execution of the bottom half—that’s why it runs at a safer time.

6. In the typical scenario, the top half saves device data to a device-specific buffer, schedules its bottom half, and exits: this operation is very fast.
7. The bottom half then performs whatever other work is required, such as awakening processes, starting up another I/O operation, and so on.
8. This setup permits the top half to service a new interrupt while the bottom half is still working.

## 3. Why is interrupt vector used in OS? What is the need of device status table?

**Ans:**

1. 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.
2. This vector contains the memory addresses of specialized interrupt handlers.
3. Device Status Table: Each entry indicates the status (e.g., busy and idle) of the corresponding device. When the status is "busy", the requests are queued and processed later when the device becomes "idle". A device entry can have a chain of the requests for the device.

## 4. Explain the positioning time for a disk (seek time, latency time).

**Ans:**

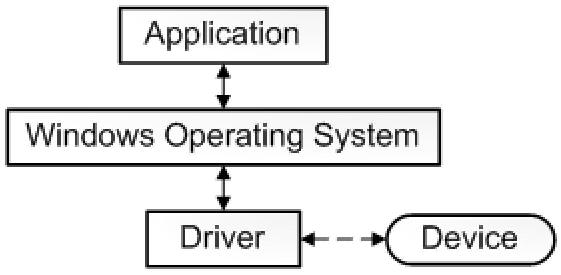
**1. Seek time:** Say you're reading some data from the (0,4). You receive instructions to read from track (2,5). The time it takes for you to move from track 0 to track 2 is seek time.
**2. Seek time:** Say you're reading some data from the (0,4). You receive instructions to read from track (2,5). The time it takes for you to move from track 0 to track 2 is seek time.

## 5. Explain the terms buses, devices and drivers?

**Ans:**

1. A system bus is a single computer bus that connects the major components of a computer system, combining the functions of a data bus to carry information, an address bus to determine where it should be sent, and a control bus to determine its operation.
2. The term devices in computer is a physical object, which is used as I/O objects in computer.

3. A device driver is a program that controls a particular type of device that is attatched to your computer. It is small piece of software that tells the OS and other software how to communicate with piece of hardware.



## 6. Analyze /dev directory and various character and block special files under /dev.

**Ans:**

1. Under Linux and UNIX each and every hardware device treated as a file. A device file allows to accesses hardware devices so that end users do not need to get technical details about hardware.

2. In short, a device file (also called as a special file) is an interface for a device driver that appears in a file system as if it were an ordinary file. This allows software to interact with the device driver using standard input/output system calls, which simplifies many tasks.

**Device file two types**

There are two types of device files based upon how data written to them and read from them is processed by the operating system and hardware:

- Character special files or Character devices
- Block special files or Block devices

Understanding Character special files or Character devices

- Talks to devices in a character by character (1 byte at a time)
- Examples: Virtual terminals, terminals and serial modems etc

Understanding Block special files or Block devices

- Talks to devices 1 block at a time ( 1 block = 512 bytes to 32KB)
- Examples: Hard disk, DVD/CD ROM, and memory regions etc

All device files are stored in /dev directory. Use cd and ls command to browse the directory:

```
cd /dev/
ls -l
```

A character device is marked with a c as the first letter of the permissions strings.

```
$ ls -l /dev/console
```

A block device is marked with a b as the first letter of the permissions strings:

```
$ ls -l /dev/sdb1
```

## 7. What do you understand by mount points?

**Ans:**

1. mount point is a directory (typically an empty one) in the currently accessible filesystem on which an additional filesystem is mounted (i.e., logically attached).
2. A filesystem is a hierarchy of directories (also referred to as a directory tree) that is used to organize files on a computer system.

3. On Linux and other Unix-like operating system, at the very top of this hierarchy is the root directory, which contains all other directories on the system, inclusive of their subdirectories, etc.

4. A variant of this definition is the part of the entire hierarchy of directories (i.e., of the directory tree) that is located on a single partition or disk.

5. A partition is a logically independent section of a hard disk drive (HDD).

6. The mount point becomes the root directory of the newly added filesystem, and that filesystem becomes accessible from that directory.

7. Any original contents of that directory become invisible and inaccessible until the filesystem is unmounted (i.e., detached from the main filesystem).

## 8. Check with information from /proc file system files viz., devices, partitions, interrupts, mounts, iomem, swaps, diskstats, softirqs, devices (under/ proc/ bus/ input).

**Ans:** /proc Files about the system information

- /proc/cpuinfo – information about CPU,
- /proc/meminfo – information about memory,
- /proc/loadvg – load average,

- /proc/partitions – partition related information,
- /proc/version – linux version
- /proc/cmdline – Kernel command line
- /proc/cpuinfo – Information about the processors.
- /proc/devices – List of device drivers configured into the currently running kernel.
- /proc/dma – Shows which DMA channels are being used at the moment.
- /proc/fb – Frame Buffer devices.
- /proc/filesystems – File systems supported by the kernel.
- /proc/interrupts – Number of interrupts per IRQ on architecture.
- /proc/iomem – This file shows the current map of the system's memory for its various devices
- /proc/ioports – provides a list of currently registered port regions used for input or output communication with a device
- /proc/loadavg – Contains load average of the system

  The first three columns measure CPU utilization of the last 1, 5, and 10 minute periods.

  The fourth column shows the number of currently running processes and the total number of processes.

  The last column displays the last process ID used.
- /proc/locks – Displays the files currently locked by the kernel

  Sample line:

  1: POSIX ADVISORY WRITE 14375 08:03:114727 0 EOF
- /proc/meminfo – Current utilization of primary memory on the system
- /proc/misc – This file lists miscellaneous drivers registered on the miscellaneous major device, which is number 10
- /proc/modules – Displays a list of all modules that have been loaded by the system
- /proc/mounts – This file provides a quick list of all mounts in use by the system
- /proc/partitions – Very detailed information on the various partitions currently available to the system
- /proc/pci – Full listing of every PCI device on your system

- /proc/stat – Keeps track of a variety of different statistics about the system since it was last restarted
- /proc/swap – Measures swap space and its utilization
- /proc/uptime – Contains information about uptime of the system
- /proc/version – Version of the Linux kernel, gcc, name of the Linux flavor installed.

## 9. Work on I/O control (ioctl()) in Linux

### Ans:

1. Input-output control (*ioctl*, in short) is a common operation or system call available with most of the driver categories. It is a "one bill fits all" kind of system call.

2. If there is no other system call, which meets the requirement, then definitely *ioctl()* is the one to use. Practical examples include volume control for an audio device, display configuration for a video device, reading device registers, basically anything to do with any device input / output, or for that matter any device specific operations.

3. In fact, it is even more versatile – need not be tied to any device specific things but any kind of operation. An example includes debugging a driver, say by querying of driver data structures.

4. The following had been its prototype in Linux kernel, for quite some time:

```
int ioctl(struct inode *i, struct file *f, unsigned int cmd, unsigned long arg);
```

5. Though, recently from kernel 2.6.35, it has changed to the following:

```
long ioctl(struct file *f, unsigned int cmd, unsigned long arg);
```

## 10. What is Polling ? Explain in brief.

**Ans:**

1. Polling is also called as Busy-waiting.
2. Assume that 2 bits are used to coordinate the producer–consumer relationship between the controller and the host. The controller indicates its state through the busy bit in the status register.
3. The controller sets the busy bit when it is busy working and clears the busy bit when it is ready to accept the next command.
4. The host signals its wishes via the command-ready bit in the command register. The host sets the command-ready bit when a command is available for the controller to execute.

5. For this example, the host writes output through a port, coordinating with the controller by handshaking as follows.

1. The host repeatedly reads the busy bit until that bit becomes clear.
2. The host sets the write bit in the command register and writes a byte into the data- out register.
3. The host sets the command-ready bit.
4. When the controller notices that the command-ready bit is set, it sets the busy bit.
5. The controller reads the command register and sees the write command. It reads the data out register to get the byte and does the I/O to the device.
6. The controller clears the command-ready bit, clears the error bit in the status register to indicate that the device I/O succeeded, and clears the busy bit to indicate that it is finished.

6. This loop is repeated for each byte.
7. In step 1, the host is busy-waiting or polling:it is in a loop, reading the status register over and over until the busy bit becomes clear.
8. For instance, when data are streaming in on a serial port or from a keyboard, the small buffer on the controller will overflow and data will be lost if the host waits too long before returning to read the bytes.

## 11. What is Interrupt ? Explain in brief.

## Ans:

1. The hardware mechanism that enables a device to notify the CPU is called an interrupt.
2. The CPU hardware has a wire called the interrupt-request line that the CPU senses after executing every instruction.
3. When the CPU detects that a controller has asserted a signal onv the interrupt-request line, the CPU performs a state save and jumps to the interrupt-handler routine at a fixed address in memory.
4. The interrupt handler determines the cause of the interrupt, performs the necessary processing, performs a state restore, and executes a return from interrupt instruction to return the CPU to the execution state prior to the interrupt.
5. ing a signal on the interrupt request line, the CPU catches the interrupt and dispatches it to the interrupt handler, and the handler clears the interrupt by servicing the device.

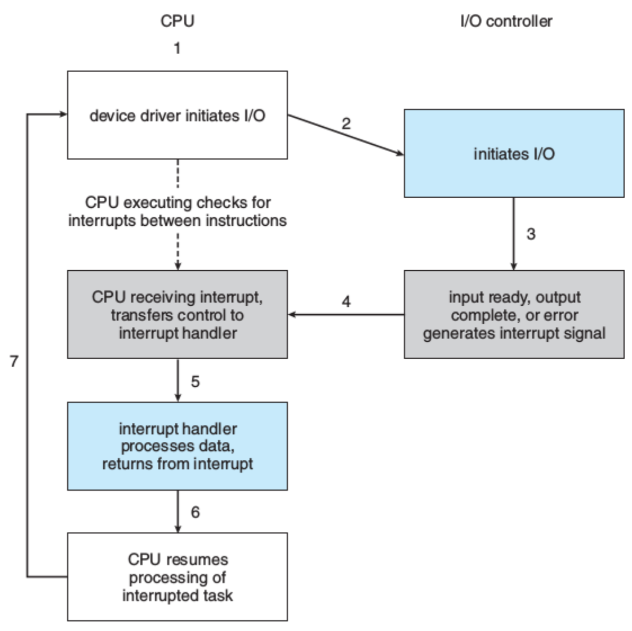

CPU
I/O controller
1
device driver initiates I/O
2
initiates I/O
CPU executing checks for interrupts between instructions
3
input ready, output complete, or error generates interrupt signal
4
CPU receiving interrupt, transfers control to interrupt handler
7
5
interrupt handler processes data, returns from interrupt
6
CPU resumes processing of interrupted task

**Fig. Interrupt driven I/O**

6. The basic interrupt mechanism just described enables the CPU to respond to an asynchronous event, as when a device controller becomes ready for service. In a modern operating system, however, we need more sophisticated interrupt-handling features.

7. Feature of interrupt handling features:

1.We need the ability to defer interrupt handling during critical processing.

2. We need an efficient way to dispatch to the proper interrupt handler for a device without first polling all the devices to see which one raised the interrupt.

3. We need multilevel interrupts, so that the operating system can distinguish between high- and low-priority interrupts and can respond wit the appropriate degree of urgency.

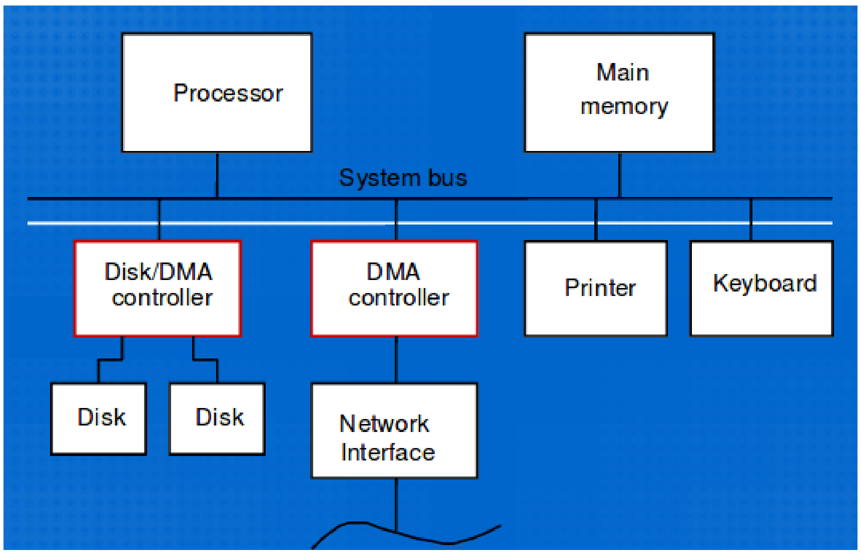
8. Most CPU s have two interrupt request lines. One is the nonmaskable interrupt, which is reserved for events such as unrecoverable memory errors. The second interrupt line is maskable: it can be turned off by the CPU before the execution of critical instruction sequences that must not be interrupted. The maskable interrupt is used by device controllers to request service.

## 12. What is DMA ? Explain in brief.

### Ans:

1. Direct memory access (DMA) is a feature of computer systems that allows certain hardware subsystems to access main system memory (RAM), independent of the central processing unit (CPU).

2.To initiate a DMA transfer, the host writes a DMA command block into memory. This block contains a pointer to the source of a transfer, a pointer to the destination of the transfer, and a count of the number of bytes to be transferred.



3 .The CPU writes the address of this command block to the DMA controller, then goes on with other work.

4. The DMA controller proceeds to operate the memory bus directly, placing addresses on the bus to perform transfers without the help of the main CPU . A simple DMA controller is a standard component in all modern computers, from smartphones to mainframes.

5. Handshaking between the DMA controller and the device controller is performed via a pair of wires called DMA -request and DMA -acknowledge.

6. The device controller places a signal on the DMA -request wire when a word of data is available for transfer. This signal causes the DMA controller to seize the memory bus, place the desired address on the memory-address wires, and place a signal on the DMA acknowledge wire. When the device controller receives the DMA -acknowledge signal, it transfers the word of data to memory and removes the DMA -request signal.

## 13. Draw kernel I/O structure.

**Ans:**

1. The purpose of the device-driver layer is to hide the differences among device controllers from the I/O subsystem of the kernel, much as the I/O system calls encapsulate the behavior of devices in a few generic classes that hide hardware differences from application.

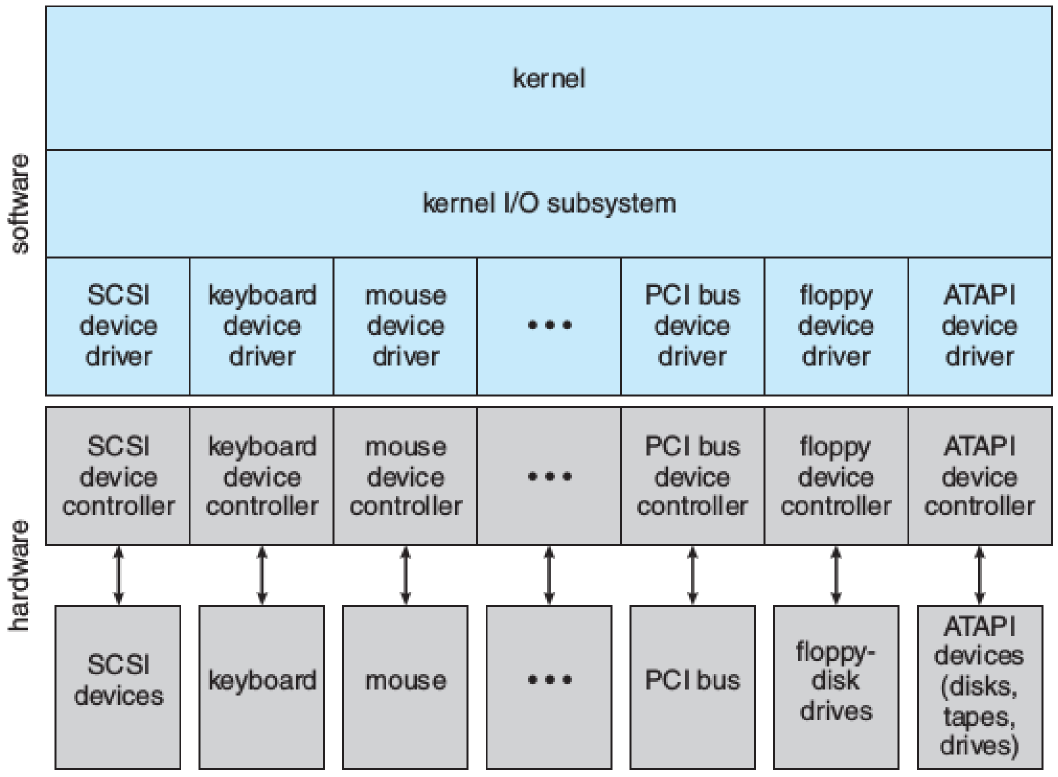


**Fig. Kernel I/O Structure**

## 14. Explain Block and Character Device

**Ans:**

1.A Character ('c') Device is one with which the Driver communicates by sending and receiving single characters (bytes, octets).

2. A Block ('b') Device is one with which the Driver communicates by sending entire blocks of data.

3. Examples for Character Devices: serial ports, parallel ports, sounds cards

4. Examples for Block Devices: hard disks, USB cameras, Disk-On-Key.

5. The block-device interface captures all the aspects necessary for accessing disk drives and other block-oriented devices. The device is expected to understand commands such as read() and write() .

6. If it is a random-access device, it is also expected to have a seek() command to specify which block to transfer next. Applications normally access such a device through a file-system interface. We can see that read() , write() , and seek() capture the essential behaviors of block-storage devices, so that applications are insulated from the low-level differences among those devices.

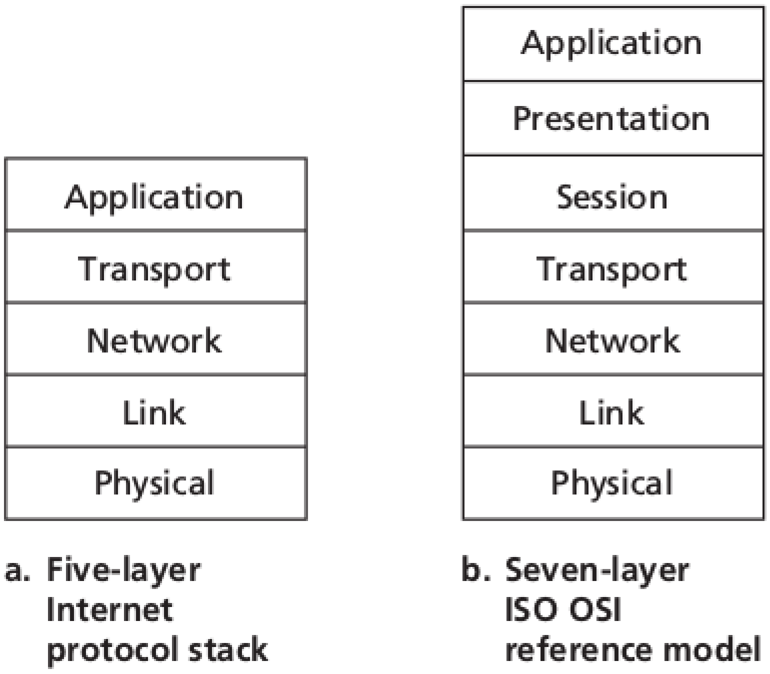
# Network Management

## 1. Explain OSI reference Layer/Model

**Ans:**

1. OSI is nothing but the Open System Interconnection Model for network management.

2. It is total 7 layer reference model for how application communicate over a network.



3. The Open Systems Interconnection model (OSI model) is a conceptual model that characterizes and standardizes the communication functions of a telecommunication or computing system without regard to their underlying internal structure and technology.

Layer 7: The application layer. This is the layer at which communication partners are identified (Is there someone to talk to?), network capacity is assessed (Will the network let me talk to them right now?), and that creates a thing to send or opens the thing received. (This layer is *not* the application itself, it is the set of services an application should be able to make use of directly, although some applications may perform application layer functions.)

Layer 6: The presentation layer. This layer is usually part of an operating system (OS) and converts incoming and outgoing data from one presentation format to another (for example, from clear text to encrypted text at one end and back to clear text at the other).

Layer 5: The session layer. This layer sets up, coordinates and terminates conversations. Services include authentication and reconnection after an interruption. On the Internet, Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) provide these services for most applications.

Layer 4: The transport layer. This layer manages packetization of data, then the delivery of the packets, including checking for errors in the data once it arrives. On the Internet, TCP and UDP provide these services for most applications as well.

Layer 3: The network layer. This layer handles the addressing and routing of the data (sending it in the right direction to the right destination on outgoing transmissions and receiving incoming transmissions at the packet level). IP is the network layer for the internet.

Layer 2: The data-link layer. This layer sets up links across the physical network, putting packets into network frames. This layer has two sub-layers, the Logical Link Control Layer and the Media Access Control Layer. Ethernet is the main data link layer in use.

Layer 1: The physical layer. This layer conveys the bit stream through the network at the electrical, optical or radio level. It provides the hardware means of sending and receiving data on a carrier network.

## 2. What is TCP/IP model of networking and what is UDP/IP model of networking? Also note the differences between them.

**Ans:**

1. TCP/IP (Transmission Control Protocol/Internet Protocol) is the basic communication language or protocol of the Internet.

2. TCP/IP is a two-layer program. The higher layer, Transmission Control Protocol, manages the assembling of a message or file into smaller packets that are transmitted over the Internet and received by a TCP layer that reassembles the packets into the original message.

3. The lower layer, Internet Protocol, handles the address part of each packet so that it gets to the right destination. Each gateway computer on the network checks this address to see where to forward the message.

6. Even though some packets from the same message are routed differently than others, they'll be reassembled at the destination.

7. UDP (User Datagram Protocol) is an alternative communications protocol to Transmission Control Protocol (TCP) used primarily for establishing low-latency and loss tolerating connections between applications on the Internet.

8. Both UDP and TCP run on top of the Internet Protocol (IP) and are sometimes referred to as UDP/IP or TCP/IP. Both protocols send short packets of data, called datagrams.

9. TCP has emerged as the dominant protocol used for the bulk of Internet connectivity owing to services for breaking large data sets into individual packets, checking for and resending lost packets and reassembling packets into the correct sequence.

10. But these additional services come at a cost in terms of additional data overhead, and delays called latency.

11. In contrast, UDP just sends the packets, which means that it has much lower bandwidth overhead and latency.

12. But packets can be lost or received out of order as a result, owing to the different paths individual packets traverse between sender and receiver.

13. UDP is an ideal protocol for network applications in which perceived latency is critical such as gaming, voice and video communications, which can suffer some data loss without adversely affecting perceived quality.

14. In some cases, forward error correction techniques are used to improve audio and video quality in spite of some loss.

## 3. How OSI is different from 4 layer TCP/IP model

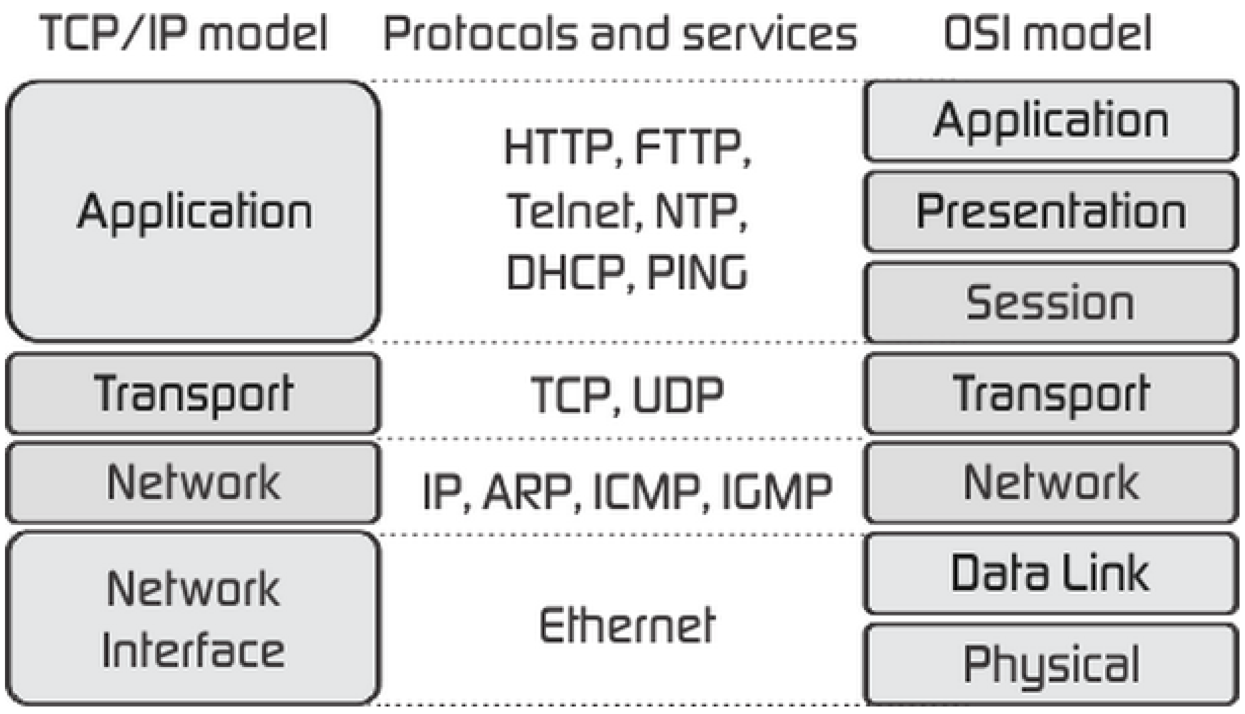
**Ans:**

| **OSI** <br> **(Open System Interconnection)** | **TCP/IP** <br> **(Transmission Control Protocol / Internet Protocol)** |
|---|---|
| 1. OSI is a generic, protocol independent standard, acting as a communication gateway between the network and end user. | 1. TCP/IP model is based on standard protocols around which the Internet has developed. It is a communication protocol, which allows connection of hosts over a network. |
| 2. In OSI model the transport layer guarantees the delivery of packets. | 2. In TCP/IP model the transport layer does not guarantees delivery of packets. Still the TCP/IP model is more reliable. |

| | |
|---|---|
| 3. Follows vertical approach. | 3. Follows horizontal approach. |
| 4. OSI model has a separate Presentation layer and Session layer. | 4. TCP/IP does not have a separate Presentation layer or Session layer. |
| 5. OSI is a reference model around which the networks are built. Generally it is used as a guidance tool. | 5. TCP/IP model is, in a way implementation of the OSI model. |
| 6. Network layer of OSI model provides both connection oriented and connectionless service. | 6. The Network layer in TCP/IP model provides connectionless service. |
| 7. OSI model has a problem of fitting the protocols into the model. | 7. TCP/IP model does not fit any protocol |
| 8. Protocols are hidden in OSI model and are easily replaced as the technology changes. | 8. In TCP/IP replacing protocol is not easy. |
| 9. OSI model defines services, interfaces and protocols very clearly and makes clear distinction between them. It is protocol independent. | 9. In TCP/IP, services, interfaces and protocols are not clearly separated. It is also protocol dependent. |

## 4. What are different networking protocols?

**Ans: As** shown in following figure different networking protocol:



## 5. Why do we have MAC and IP addresses?

### Ans:

1. The purpose of MAC addresses is to provide a unique hardware address or physical address for every node on a local area network (LAN) or other network.

2. A node is a point at which a computer or other device (e.g., a printer or router) is connected to the network. When a computer is connected to a network, a correspondence table relates the computer's IP address to its physical address on the network.

3. The MAC addresses of the sending computers are contained in the header of each packet, thus allowing packets to arrive at their intended destination.

4. To put in very simple words the function of IP Address is to allocate a unique address to a device on a network so that any information sent to that device can reach it by referring to its address.

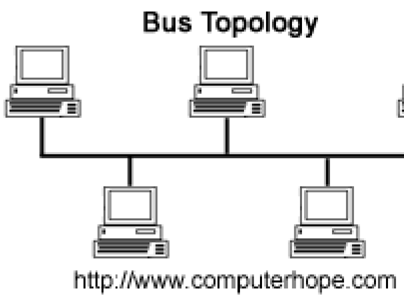
5. Its just like when you send a letter to your friend it reaches him when the postman searches for his house number that you have written on the address and on finding it delivers the letter to your friend living in that house.

6. Similar is the case with IP Address.

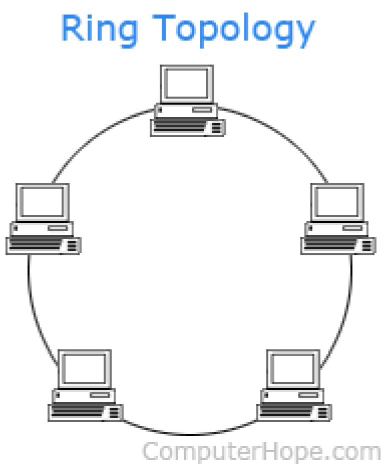
## 6. What are different types of topologies viz., bus, ring, star, mesh

### Ans:

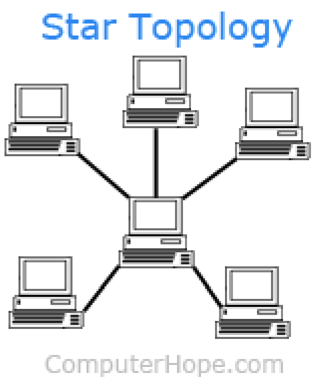
1. A **bus topology** is a network setup in which each computer and network device are connected to a single cable or backbone.



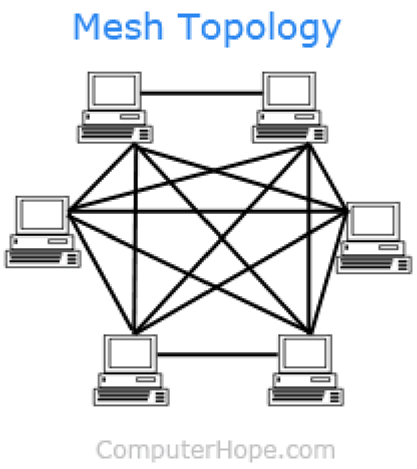
2. A **ring topology** is a computer network configuration where the devices are connected to each other in a circular shape. Each packet is sent around the ring until it reaches its final destination. Ring topologies are used in both LAN and WAN setups.



3. A **star topology** is one of the most common network setups. In this configuration, every node connects to a central network device, like a hub, switch, or computer. The central network device acts as a server and the peripheral devices act as clients.



4. A mesh topology is a network setup where each computer and network device is interconnected



with one another, allowing for most transmissions to be distributed, even if one of the connections go down. It is a topology commonly used for wireless networks.

## 7. What is meant by socket?

**Ans:**

1. Sockets allow communication between two different processes on the same or different machines.

2. To be more precise, it's a way to talk to other computers using standard Unix file descriptors.

3. In Unix, every I/O action is done by writing or reading a file descriptor.

4. A file descriptor is just an integer associated with an open file and it can be a network connection, a text file, a terminal, or something else.

5. A network *socket* is an internal endpoint for sending or receiving data at a single node in a computer network.

## 8. Explain TCP and UDP headers

**Ans:**

**Ans :**

1. To transmit a UDP datagram, a computer completes the appropriate fields in the UDP header and forwards the data together with the header for transmission by the IP network layer.
The UDP header consists of four fields each of 2 bytes in length:

- **Source Port** (UDP packets from a client use this as a service access point (SAP) to indicate the session on the local client that originated the packet. UDP packets from a server carry the server SAP in this field)
- **Destination Port** (UDP packets from a client use this as a service access point (SAP) to indicate the service required from the remote server. UDP packets from a server carry theA network *socket* is an internal endpoint for sending or receiving data at a single node in a computer network. client SAP in this field)
- **UDP length** (The number of bytes comprising the combined UDP header information and payload data)
- **UDP Checksum** (A checksum to verify that the end to end data has not been corrupted by routers or bridges in the network or by the processing in an end system. The algorithm to compute the checksum is the Standard Internet Checksum algorithm. This allows the receiver to verify that it was the intended destination of the packet, because it covers the IP addresses, port numbers and protocol number, and it verifies that the packet is not truncated or padded, because it covers the size field. Therefore, this protects an application against receiving

corrupted payload data in place of, or in addition to, the data that was sent. In the cases where this check is not required, the value of 0x0000 is placed in this field, in which case the data is not checked by the receiver.

2. As with UDP, TCP header includes source and destination port numbers, which are used for multiplexing/demultiplexing data from/to upper-layer applications. Also, as with UDP, the header includes a checksum field. A TCP segment header also contains the following fields:

3. The 32-bit sequence number field and the 32-bit acknowledgment number field are used by the TCP sender and receiver in implementing a reliable data transfer service, as discussed below.

4. The 16-bit receive window field is used for flow control. We will see shortly that it is used to indicate the number of bytes that a receiver is willing to accept.

5. The 4-bit header length field specifies the length of the TCP header in 32-bit words. The TCP header can be of variable length due to the TCP options field.

## 9. Explain the purpose of CRC/checksum

## Ans:

1. A cyclic redundancy check (CRC) is an error-detecting code commonly used in digital networks and storage devices to detect accidental changes to raw data. Blocks of data entering these systems get a short check value attached, based on the remainder of a polynomial division of their contents.

2. Checksums are used to ensure the integrity of data portions for data transmission or storage. A checksum is basically a calculated summary of such a data portion.

3. Network data transmissions often produce errors, such as toggled, missing or duplicated bits. As a result, the data received might not be identical to the data transmitted, which is obviously a bad thing.

## 10. Work on the network commands netstat, net.

## Ans:

1. netstat gives you the current network interface statistics and prints information about the Linux networking subsystem. net is a tool for administrating Samba and remote CIFS servers.
2. The netstat command is a Command Prompt command used to display *very* detailed information about how your computer is communicating with other computers or network devices
3. Specifically, the netstat command can show details about individual network connections, overall and protocol-specific networking statistics, and much more, all of which could help troubleshoot certain kinds of networking issues.

# Device Driver

### 1. How many ways of integrating module to Kernel ?

**Ans:.**

**1.** Two possible ways of integrating module to kernel:

**a. Dynamic Insertion as a kernel module [Modprobe & Insmod]**

**b. Linking statically to the kernel code**

**2.** Both insmod and modprobe is used to insert kernel module.

3. But everyone will prefer modprobe, because insmod have no capability to resolve dependency issue. But modprobe can do that.

3. While we are installing the module using modprobe, first it will check the dependencies in /lib/modules/<kernel-version>/modules.dep

4. This file contain entire file modules & its corresponding dependencies.

5. Modules.dep file is generated by "depmod" command.

### 2. How to insert module during run time?

Ans:

1. Dynamically we can insert module during run time.

2. 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.

## 3. What do the commands insmod, lsmod, modinfo, modprobe, rmmod perform?

**Ans:**

**insmod: To insert a module into kernel.**

**lsmod: It will diaplay information about a list of already loaded kernel module.**

**modinfo: It will diaplay information about kernel module.**

**rmmod: To remove a module from the kernel.**

## 4. What is meant by ISR? Why ISR’s are needed for Device Drivers?

**Ans:**

1. ISR is nothing but the interrupt service routine.

2. The function the kernel runs in response to a specific interrupt is called an interrupt handler or interrupt service routine (ISR).

3. Each device that generates interrupts has an associated interrupt handler.

4. For example, one function handles interrupts from the system timer, whereas another function handles interrupts generated by the keyboard.
5. he interrupt handler for a device is part of the device's driver—the kernel code that manages the device.
6. In Linux, interrupt handlers are normal C functions.
7. They match a specific prototype, which enables the kernel to pass the handler information in a standard way, but otherwise they are ordinary functions.
8. What differentiates interrupt handlers from other kernel functions is that the kernel invokes them in response to interrupts and that they run in a special context called interrupt context.
9. This special context is occasionally called atomic context because, as we shall see, code executing in this context is unable to block.
10. Because an interrupt can occur at any time, an interrupt handler can, in turn, be executed at any time.
11. For protecting critical sections inside interrupt handlers, you can't use mutexes because they may go to sleep. Use spinlocks instead, and use them only if you must.
12. Interrupt handlers cannot directly exchange data with user space because they are not connected to user land via process contexts.
13. Interrupt handlers are supposed to get out of the way quickly but are expected to get the job done.
14. You need not design interrupt handlers to be reentrant. When an interrupt handler is running, the corresponding IRQ is disabled until the handler returns. So, unlike process context code, different instances of the same handler will not run simultaneously on multiple processors.

## 5. What is meant by IPL's?

**Ans: IPL** is nothing but the Interrupt Priority Level.
1. It is the part of the current system interrupt state, which indicates the interrupt rerquests that will currently be accepted.
2. The operating systems' Interrupt Priority Level (IPL) is a software-only scheme that assigns a priority level to every driver/device combination configured in any given system, and also supports hardware interrupts.

## 6. What are generic driver entry points?

**Ans:**
1. Every driver must have one common entry point.

2. The entry point is called `init_module`() in Linux.

3. Remaining entry points for all drivers open(), close(), __init(), __fini(), __info(), attatch(), detach(), getinfo(), probe(), power(),dump(), identify() etc.

**7. What are the types of device drivers available?**

**Ans:**

1. There are three types of device driver available.

A: Character Device Driver eg. serial ports, parallel ports, sounds cards

B: Block Device Driver . eg. hard disks, USB cameras, Disk-On-Key.

C: Network Device Driver  Eg. Ethernet, wifi etc.

**8. How character driver is different from block device driver?**

**Ans:**

A Character ('c') Device is one with which the Driver communicates by sending and receiving single characters (bytes, octets).

A Block ('b') Device is one with which the Driver communicates by sending entire blocks of data.

**9. What is need of top half and bottom half routines (or upper half and lower half)?**

**Ans:**

1. Whenever device driver raise interrupt, isr should process fast and respond to interrupt immediately.

2. After creating data structure for device that called top half and later process other works related to driver in bottom half.

3. Lengthy tasks inside interrupt handler degrade system responsiveness.

**a. Top Half:** Perform time critical tasks such as acknowledging receipt of interrupt, reseting hardware etc. It is the routine that actually respond to the interrupt –the one you register with request_irq().

**b. Bpttom Half:** Perform any interrupt related work not performed by interrupt handler.  he bottom half is a routine that is scheduled by the top half to be executed later, at a safer time.A bottom half is a low-priority function, usually related to interrupt handling, that is waiting for the kernel to find a convenient moment to run it.

4. The big difference between the top-half handler and the bottom half is that all interrupts are enabled during execution of the bottom half—that's why it runs at a safer time.

5. In the typical scenario, the top half saves device data to a device-specific buffer, schedules its bottom half, and exits: this operation is very fast.

6. The bottom half then performs whatever other work is required, such as awakening processes, starting up another I/O operation, and so on.

7. This setup permits the top half to service a new interrupt while the bottom half is still working.

**10. What are the different types of bottom half routines are available and what is their purpose?**

**Ans: T**here are two type of bottom half routines are available:

**1] Tasklets:** Software interrupt context

**2] Softirq:**Software interrupt context

**2] Workqueue:** Process context

**1] Tasklets**

1.Tasklets are a special function that may be scheduled to run, in software interrupt context, at a system-determined safe time.

2. They may be scheduled to run multiple times, but tasklet scheduling is not cumulative; the tasklet runs only once, even if it is requested repeatedly before it is launched.

3. No tasklet ever runs in parallel with itself, since they run only once, but tasklets can run in parallel with other tasklets on SMP systems.

4. If your driver has multiple tasklets, they must employ some sort of locking to avoid conflicting with each other.

5. Tasklets are also guaranteed to run on the same CPU as the function that first schedules them.

6. Locking between the tasklet and the interrupt handler may still be required.

7. Tasklets must be declared with the `DECLARE_TASKLET` macro:

```
DECLARE_TASKLET(name, function, data);
```

8. Tasklet are often powerful mechanism for bottom half processing, they are very fast, but all tasklet code must be atomic.

**2] Workqueue:**

1. The alternative tasklet is workqueue, which may have higher latency, but that are allowed to sleep.

2. Since the workqueue function runs in process context, it can sleep if need be.

3. Most important, work queues are schedulable and can therefore sleep.

4. Normally, it is easy to decide between using work queues and softirqs/tasklets.

5. If the deferred work needs to sleep, work queues are used.

6. If the deferred work need not sleep, softirqs or tasklets are used.

7. If you need a schedulable entity to perform your bottom-half processing, you need work queues.

8. They are the only bottom-half mechanisms that run in process context, and thus, the only ones that can sleep.

9. If you do not need a kernel thread to handle your deferred work, consider a tasklet instead.

## 11. What would copy_from_user() and copy_to_user() functions perform?

Ans:

**a. Copy_to_user():**

1. Copy a block of data into user space.

2. unsigned long **copy_to_user** ( void_user *to, const void * from, unsigned long n);

    to = Destination address, in user space

    from = Source address in kernel space

    n = number of byte to copy

3. User context only, this function may sleep.

4. It will copy data from user space to kernael space.

5. Return number of byte that could not be copied. On successd, this will be zero.

**b. Copy_from_user():**

1. Copy a block of data from user space.

2. unsigned long **copy_from_user** ( void *to, const void_user * from, unsigned long n);

    to = Destination address in kernel space

    from = Sourec address in user space

    n = Number of bytes to copy.

3. User context only. This function may sleep.

4. Copy data from user space to kernel space.

5. Returns number of bytes that could not be copied. On success, this will be zero.

6. If some data could not be copied, this function will pad the copied data to the requested size using zero bytes.

## 12. What are the differences between SoftIRQ and Tasklets?

| SoftIrq | Tasklet |
|---|---|
| softirq is guaranteed to run on the CPU it was scheduled on | Tasklets don’t have that guarantee to run on the CPU it was scheduled on |
| The same SoftIrq can run on two seperate CPU’s at the same time | The same tasklet can not run on two separate CPU’S at the same time [Note- Two different tasklets can run on two different CPU’S, just not the same one] |
| SoftIRQs are re-entarant | Tasklets are not re-entarant. |
| SoftIrq can be only created/destroyed statically . | Tasklets can be created/destroyed statically or dynamically |

# Building Kernel Image

## 1. What is meant by cross-compilation? How it is different from general compilation?

**Ans:**

**1.** Compiler that runs on the development machine but generate code for target.

2. Cross compilers are used to generate software that can run on computers with a new architecture or on special-purpose devices that cannot host their own compilers.

3. A cross compiler is a compiler capable of creating executable code for a platform other than the one on which the compiler is running. For example, a compiler that runs on a Windows 7 PC but generates code that runs on Android smartphone is a cross compiler.

## 2. How to communicate to target board, what are the different ways available?

**Ans:**

1. The Primary Mode Of Development For Real-Time Embedded Software Has Been The "Host/Target" Approach.

2. In This Approach, The Software For The Application Is Developed On One Computer System To Be Executed On Another. The Increased Memory And Speed Of Embedded Hardware Has Now Reached

3. A Point Where These Systems Are Capable Of Supporting Both Their Real-Time Applications And The Software Engineering Environments For Their Development.

4. This Has Made The Acceptance Of The Higher Risk And Complexity Inherent To The Host/Target Approach An Undesirable Strategy.The Shift Back To Self-Targetted Development For Embedded Software Will Have Important Consequences In The Future Of Software Engineering Environments.

5. It Also Heralds The Emergence Of A New Type Of Computer System: The General Purpose Embedded Computer.

6. Host -Target System Development Target System Development Approach Approach During development process, a host system is used Then locating and burning the codes in the target board.

7. Target board hardware and software later copied to get the final embedded system  Final system function exactly as the one tested and debugged and finalized during the development process

## 3. Explain the process of building kernel Image (from kernel.org) with the detailed steps.

**Ans:**

1. Linux kernel is the life force of all Linux family of operating systems including Ubuntu, CentOS, and Fedora. For most part, you don't need to compile the kernel, as it is installed by default when you install the OS. Also, when there is a critical update done to the kernel, you can use yum, or apt-get to update the kernel on your Linux system.
2. 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.
3. To enable experimental features that are not part of the default kernel. To enable support for a new hardware that is not currently supported by the default kernel. To debug the kernel Or, just to learn how kernel works, you might want to explore the kernel source code, and compile it on your own.
4. In this tutorial, we'll explain how to compile Linux kernel from source. 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.

**1. Download the Latest Stable Kernel**

The first step is to download the latest stable kernel from kernel.org.

```
# cd /usr/src/
# wget https://www.kernel.org/pub/linux/kernel/v3.x/linux-3.9.3.tar.xz
```

**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
```

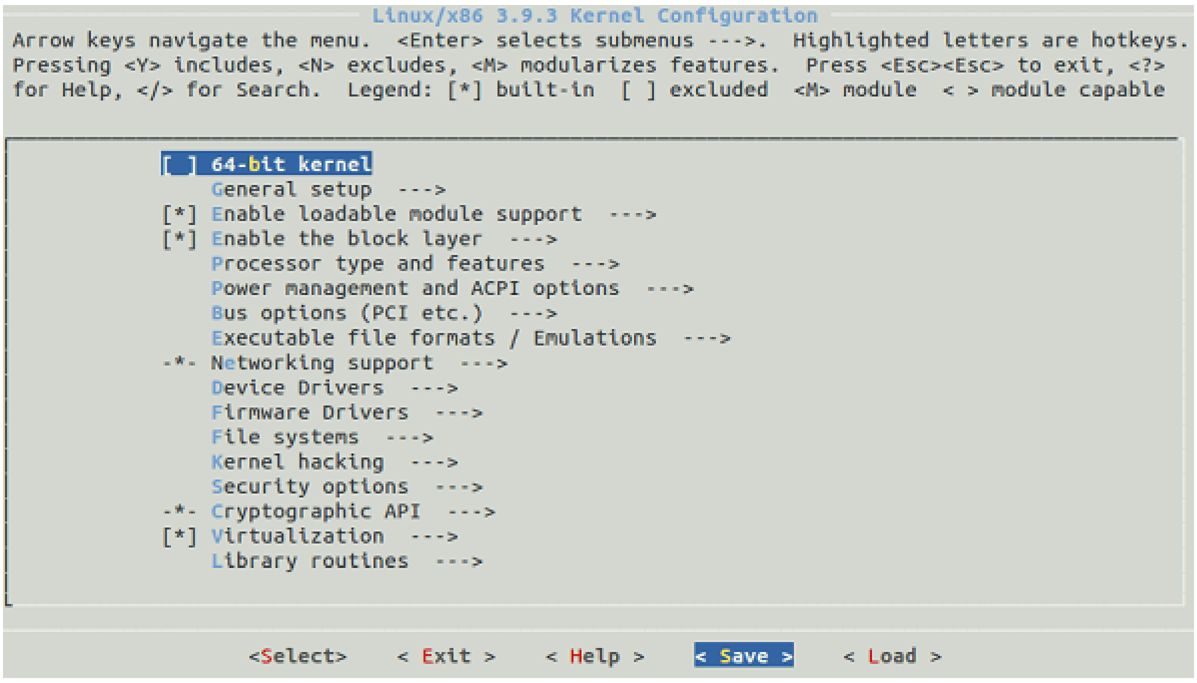
**3. Configure the Kernel**

The kernel contains nearly 3000 configuration options. To make the kernel used by most people on most hardware, the Linux distro like Ubuntu, Fedora, Debian, RedHat, CentOS, etc, will generally include support for most common hardware. You can take any one of configuration from the distro, and on top of that you can add your own configuration, or you can configure the kernel from scratch, or you can use the default config provided by the kernel.

```
# cd linux-3.9.3
```

**# make menuconfig**

The make menuconfig, will launch a text-based user interface with default configuration options as shown in the figure. You should have installed “libncurses and libncurses-devel” packages for this command to work.



```
Linux/x86 3.9.3 Kernel Configuration
Arrow keys navigate the menu.  <Enter> selects submenus --->.  Highlighted letters are hotkeys.
Pressing <Y> includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to exit, <?>
for Help, </> for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable

            [ ] 64-bit kernel
                General setup  --->
            [*] Enable loadable module support  --->
            [*] Enable the block layer  --->
                Processor type and features  --->
                Power management and ACPI options  --->
                Bus options (PCI etc.)  --->
                Executable file formats / Emulations  --->
            -*- Networking support  --->
                Device Drivers  --->
                Firmware Drivers  --->
                File systems  --->
                Kernel hacking  --->
                Security options  --->
            -*- Cryptographic API  --->
            [*] Virtualization  --->
                Library routines  --->

                    <Select>    < Exit >    < Help >    < Save >    < Load >
```

We will use the default config provided by the kernel. So select “Save” and save the config in the file name “.config”.

The following is a sample of the “.config” file:

```
CONFIG_MMU=y
CONFIG_NEED_DMA_MAP_STATE=y
CONFIG_NEED_SG_DMA_LENGTH=y
CONFIG_GENERIC_ISA_DMA=y
CONFIG_GENERIC_BUG=y
CONFIG_GENERIC_HWEIGHT=y
```

## 4. Compile the Linux Kernel

Compile the main kernel:

**# make**

Compile the kernel modules:

**# make modules**

Install the kernel modules:

**# make modules_install**

At this point, you should see a directory named /lib/modules/3.9.3/ in your system.

**5. Install the New Kernel**

Install the new kernel on the system:

**# 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 is 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**

To use the new kernel that you just compiled, reboot the system.

**# 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.

*By Rathod Raja V.*

# ARM

## 1. Explain ARM architecture?

Ans:

1. ARM statnd for Advanced RISC Machine.
2. It supports RISC architecture.
3. It use only two control signal Memory Read and Memory Write.
4. Support Pipeline.
5. In single clock, execution of multiple instruction.
6. Same address for I/O and Memory.
7. Emphasis on Software
8. Fast performance
9. Low power consumption.
10. Support Load and store Instruction set.
11. Use for specific purpose application.
12. Complexity in compiler.
13. ARM having total 4 profile
    - A. Classic Processor
    - B. Cortex-M Serries
    - C. Cortex-R serries
    - D. Cortes-A serries
14. Within the device the componant are connected together using AMBA bus.
15. AMBA specifies two buses:
    - A. High perforamance system bus AXI: For connecting peripherals
    - B. Low power bus APB : For memory Interface

16. ARM having total 37 register set.

## 2. Explain differences between CISC and RISC

**Ans:**

| | RISC | CISC |
|---|---|---|
| **Instruction Type** | Simple | Complex |
| **Opeartion of on instuction** | One cycle | More cycle |
| **Instruction format** | Fixed | Variable |
| **Instruction Execution** | In parallel | In sequential |
| **Addressing mode** | Simple | Complex |
| **Instruction accessing the memory** | Only two, Load and Store | Almost All from the Set. |
| **Register Set** | Multiple | Unique |
| **Complexity** | In Compiler | In CPU |
| **Address bus** | Same address bus for IO and Memory | Different address bus for memory and IO |
| **Emphasis** | More important to software | More important to hardware |
| **Power Consumption** | Less | More |
| **Transistor** | Require less transistor to function | Require more transistor to function |
| **Application** | Specific purpose applicatio | General Purpose Application |
| **Example** | Cell phone etc. | Desktop PC etc. |

## 3. Explain different modes in ARM?

**Ans:**

1. Arm support total 7 processor mode.
2. These mode are seperated into two parts:

A: Previledge Mode: Full read and write access to CPSR

B: Non-Previledge Mode: Only read access to CPSR

### A: Previledge Mode

1. Supervisor Mode
2. Fast Interrupt Request [FIQ] Mode
3. Interrupt Request [IRQ] Mode
4. Abort Mode
5. Undefined Mode

6. System Mode

**B:Non-Previledge Mode**

1. User Mode

1. Supersior Mode:Entered on reset and when supervisor call instruction is executed.

Supervisor mode is the mode that the processor is in after reset and is generally the mode that an operating system kernel operates in.

2. Fast Interrupt Request [FIQ]:Entered when high priority interrupt is raised

3. Interrupt Request Mode [IRQ]: Entered when normal priority interrupt is raised

4. Abort Mode : The processor enters abort mode when there is a failed attempt to access memory

5. Undefined Mod: Used to handle undefined instruction.Undefined mode is used when the processor encounters an instruction that is undefined or not supported by the implementation.

6. System Mode: System mode is a special version of user mode that allows full read-write access to the cpsr.

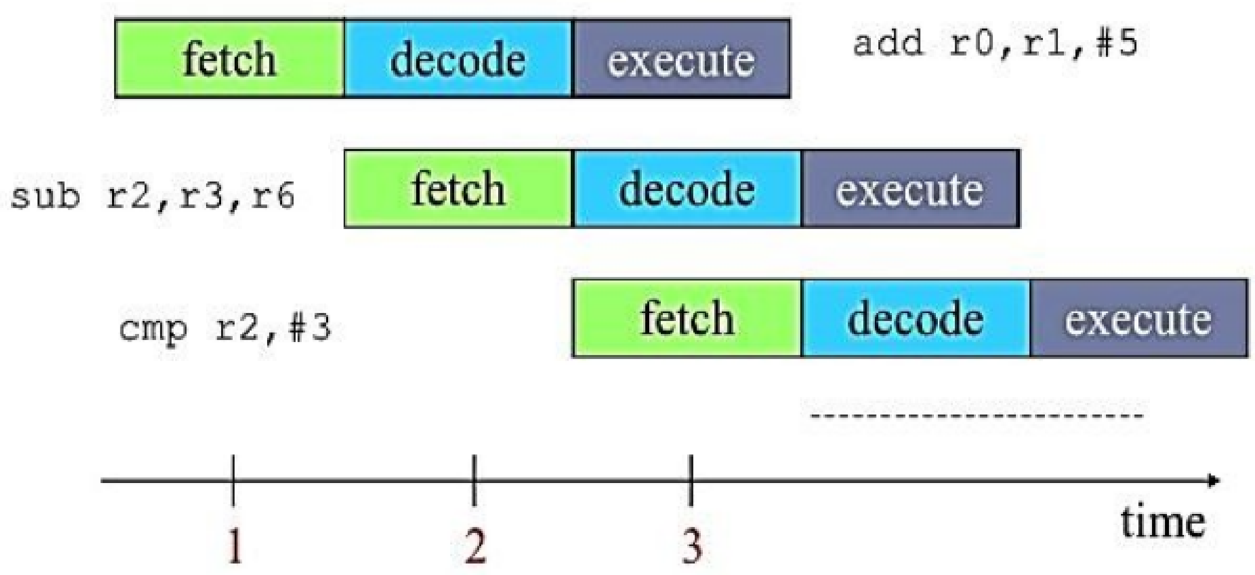
7. User Mode : User mode is used for programs and applications

**4. Explain pipelining in ARM?**

**Ans:**

1. ARM support 3-stage, 5-stage, 8-stage,and 13-stage pipelining.

2. Normal 3 stage pipeline is show in following figure:



3. In first cycle add instruction fetch from memory

4. In second cycle add instruction is decoded and at the same time sub instruction is fetched from memory

5. In third cycle add instruction will execute , sub instruction will decode and new instruction i.e cmp will fetch from mamory.

6. In this way by using pipeline we can execute multiple instruction in single cycle.

## 5. Explain Linux boot sequence in case of ARM architecture?

**Ans:** Following are the steps for Linux booting sequence in case of ARM architecture:

**1. Initiall switch on the power**

**2. ROM Code:**

> **1. Basic Hardware initialization**
>
> **2. Load second stage bootloader**

**3. X-Loader**

> **1. Initialization of DRAM**
>
> **2. Load u-boot bootloader**

**4. U-Boot Loader**

> **1. Board support package initialization[BSP] (Mux Ocnfiguration) and Driver initialization**
>
> **2. Load Kernel Image i.e. uImage**

**5. Kernel Image**

> **1. uImaage contain board specific information.**
>
> **2. Device driver , wifi, bluetooth, MMU initialization, interrupt initialize , kernel service initialization etc.**
>
> **3. Mount RFS i.e. Root File System**

**6. RFS**

> **1. Finally generate a root file system.**