**Process Management**

1) How to manipulate the current process states?

**Linux kernel provide set\_task\_state(task,state) by which you can manipulate the state of given task. By using of set\_task\_state you can change to state of given task.**

2) What are kernel thread?

**Kernel perform some background operation by using of some thread that is called kernel thread.kernel thread are the special thread which doesnt have the process address space .They just run in the kernel space . We can reschedule kernel thread like a normal process .Kernel delegates several task to kernel thread like flush etc.**

3) How threads are implemented in Linux kernel?

**In Linux there is no special provision for thread . In Linux thread is a process which actual share some resources with other process . Every thread has a unique task\_struct.creating thread also created by using the clone sys call except some flags for sharing the data like file system , signal handler,open files also set. Flags are used to specify that which resources are shared between parent and child process or thread.E.g clone(CLONE\_VM | CLONE\_FS | CLONE\_FILES | CLONE\_SIGHAND, 0);**

4) What are different states of a process in linux?

**In linux a process can be in one of the following state. 1) TASK\_RUNNING :- This state specify that process is either running or in ready state and waiting in runqueue . 2) TASK\_INTRUPTTIBLE:- This condition states that the task is in sleeping state and waiting for some event to occur. When this condition occurs task comes in TASK\_RUNNING state. 3) TASK\_UNINTRRUPTIBLE:- This state is similar as previous state but in this state process does not wait for any event to occur. 4)\_TASK\_TRACED:- The process is getting traced by another process , such as a debugger.**

5) What is difference between process and thread?

**Process means a program is in execution, whereas thread means a segment of a process. A Process is not Lightweight, whereas Threads are Lightweight. A Process takes more time to terminate, and the thread takes less time to terminate. Process takes more time for creation, whereas Thread takes less time for creation.**

6) Generally what resources are shared between threads?

**Resources like code, data, and files can be shared among all threads within a process. Note: stack and registers can't be shared among the threads. Each thread has its own stack and registers.**

7) What is process descriptor?

**A set of information that defines the status of resources allocated to a process. When a system contains several processes, any of which may be active at any one time, there will be for each process a descriptor defining the status of that process.**

8) What is task struct?

**The task\_struct is a relatively large data structure, at around 1.7 kilobytes on a 32-bit machine. This size, however, is quite small considering that the structure contains all the information that the kernel has and needs about a process.**

9) What was the need of thread\_info structure?

**The task\_struct is a large data structure around 1.7 Kb on a 32 bit machine and kernel stack is either 4KB or 8KB . Hence the task of storing structure of 1.7 kb is very much difficult. So kernel introduced concept of thread\_info, which is very much slimmer than task\_thread and just points to task\_struct structure.**

10) Difference between fork() and vfork() ?

**The primary difference between fork and vfork is that the child process created by the fork has a separate memory space from the parent process. ... On the other hand, child process created using vfork suspend the execution of parent process till its execution is completed.**

11) What is process context?

**The kernel is executing in process context when it is running a system call.**

12) What is zombie process?

**A zombie process or defunct process is a process that has completed execution but still has an entry in the process table: it is a process in the "Terminated state". You can't kill a zombie process because it's already dead.**

13) How orphan process is handled in linux ?

**An orphan process is a running process whose parent process has finished or terminated. In a Unix-like operating system any orphaned process will be immediately adopted by the special init system process. This operation is called re-parenting and occurs automatically.**

**Process Scheduling**

1) what is process scheduling?

**Process of selecting which process to give chance to run is called Process scheduling. It is responsibility of processor to decide which process to run, when and for how long. For multiprogramming system scheduler is base . Scheduler has to decide which process to run and which process has to wait. Scheduler has to allot the time to each process efficiently and make the efficient use of Processor so that user can feel that all the processes is executing at the same time.**

2) What is cooperative multitasking and pre-emptive multitasking?

**Co-operative multitasking: - In this method process does not stop running until it voluntarily stops. The act of process voluntarily stopping itself called yielding.**

**Preemptive multitasking: - In this kind of multitasking scheduler decides which process to start running and which process to stop. Good thing in this multitasking is that scheduler can take the global decision. All the linux kind OS comes in preemptive scheduling flavor.**

3) What is yielding?

**The sched\_yield() function allows a thread to give up control of a processor so that another thread can have the opportunity to run.**

4) What is limitation of cooperative multitasking?

**It is difficult to stop a task once it has started. There is no way to suspend the thread that is currently operating when using cooperative multitasking.**

5) I/O bound versus Processor bound process?

**I/O bound processes spend most of the time waiting for some input/output event to occur and take very less time of processor. Processor bound process takes most of the time executing the code . These kind of process runs continuously without waiting for any input/output event and stopped while printed by compiler.**

6) What is process priority?

**A process' priority can range between 0 (lowest priority) and 127 (highest priority). User mode processes run at lower priorities (lower values) than system mode processes. A user mode process can have a priority of 0 to 65, whereas a system mode process has a priority of 66 to 95.**

7) What kind of priority is maintained in linux?

**Linux implements two types of process priority value. 1) Nice value:-A number from -20 to +19 with a default value of 0. Large nice value corresponds to less priority. 2) Real Time Priority:-A value from 0 to 99. Higher the real time value means higher priority.**

8)What is nice value?

**nice is used to invoke a utility or shell script with a particular CPU priority, thus giving the process CPU time than other processes.**

9) What is virtual run time?

**the virtual runtime of a task is its actual runtime normalized to the total number of running tasks.**

10) What are the available scheduling classes in linux?

**Six types of process scheduling algorithms are: First Come First Serve (FCFS), 2) Shortest-Job-First (SJF) Scheduling, 3) Shortest Remaining Time, 4) Priority Scheduling, 5) Round Robin Scheduling, 6) Multilevel Queue Scheduling**.

11) Which type of scheduling used in linux?

**Completely Fair Scheduling**

12) How next task is picked for scheduling?

13) what is scheduler entry point in linux?

**The main entry point into the process scheduler is the function \_\_schedule. This is the function that the rest of the kernel uses to invoke the process scheduler, deciding which process to run and then running it.**

14) what is waitqueus?

**waitqueue is the list of processes waiting for an event. In other words, A wait queue is used to wait for someone to wake you up when a certain condition is true.**

15) How is context switching handled in linux?

**A context switch is described as the kernel suspending execution of one process on the CPU and resuming execution of some other process that had previously been suspended. A context switch is required for every interrupt and every task that the scheduler picks**.

16) What is user preemption and kernel preemption?

**Linux supports. only user-level preemption, which allows the scheduler to suspend a. process as long it's running in user mode. It will not suspend a. running process that is in kernel mode, though**

**Syscalls**

1) What is syscalls ?

**A**[**system call**](https://www.geeksforgeeks.org/introduction-of-system-call/)**is a procedure that provides the interface between a process and the operating system. It is the way by which a computer program requests a service from the kernel of the operating system.**

2) How system calls are implemented in linux ?

**It goes through glibc, which issues a 0x80 interrupt after filling registers with parameters. The kernel's interrupt handler then looks up the syscall in the syscall table and invokes the relevant sys\_\*() function.**  **On Linux, the arguments are passed using ebx, ecx, edx, edi and esi and system call number using eax register**.

3) What happens when process in user space calls a syscall ?

Diagram, schematic

Description automatically generated

4) What is the need of verifying parameter in definition of syscall ?

**System calls must carefully verify all their parameters to ensure that they are valid and legal. The system call runs in kernel-space, and if the user can pass invalid input into the kernel without restraint, the system's security and stability can suffer.**

**Interrupt Handling**

1) What is interrupt ?:

**An interrupt is simply a signal that the hardware can send when it wants the processor's attention.**

2) What is interrupt handler or ISR ?

**The job of the interrupt handler is to service the device and stop it from interrupting. Once the handler returns, the CPU resumes what it was doing before the interrupt occurred. The DDI/DKI provides interfaces for registering and servicing interrupts.**

3) What is top halves and bottom halves ?

4) How is interrupt registered?

**Before a device driver can receive and service interrupts, the driver must call ddi\_intr\_add\_handler(9F) to register an interrupt handler with the system. Registering interrupt handlers provides the system with a way to associate an interrupt handler with an interrupt specification.**

5) What are different interrupt handler flags ?

**The Interrupt flag (IF) is a flag bit in the CPU's FLAGS register, which determines whether the (CPU) will respond immediately to maskable hardware interrupts. If the flag is set to 1 maskable interrupts are enabled.**

6) How interrupt are freed ?

7) What are the considerations needs to taken care while writing interrupt handler ?

8) What is interrupt context ?

**When executing an**interrupt handler**or bottom half, the kernel is in**interrupt context**.**

9) How to disable and enable interrupts ?

**The Interrupt Enable register is programmed through two addresses. To set the enable bit, you need to write to the SETENA register address; to clear the enable bit, you need to write to the CLRENA register address. In this way, enabling or disabling an interrupt will not affect other interrupt enable states.**

10) What are different bottom halves techniques in linux ?

**There are three main types of bottom halves: namely tasklets, workqueues and kernel threads. Tasklets are used to queue up work to be done at a later time. Tasklets can be run in parallel, but the same tasklet cannot be run on multiple CPUs at the same time.**

12) What is tasklets , softirq and workqueus and difference among them ?

13) When to choose which bottom halves ?

14) How to implements softirq , tasklets and workqueus ?

15) How to schedule tasklet ?

16) What is ksoftirqd ?

17) How to disable bottom halves?

18) How locking between bottom halves handled?

19) Why we need preemption.

**Kernel Synchronization**

1) What is synchronization

**synchronization is defined as a mechanism which ensures that two or more concurrent processes or threads do not simultaneously execute some particular program segment known as a critical section.**

2) What is critical section

**When more than one processes access a same code segment that segment is known as critical section. Critical section contains shared variables or resources which are needed to be synchronized to maintain consistency of data variable**

3) What is race condition

**Race conditions generally involve one or more processes accessing a shared resource (such a file or variable), where this multiple access has not been properly controlled.**

4) Why we need to take care of synchronization

**The need for synchronization originates when processes need to execute concurrently. The main purpose of synchronization is the sharing of resources without interference using mutual exclusion**

6) What is deadlocks.

**A deadlock happens in operating system when two or more processes need some resource to complete their execution that is held by the other process**

7) What is atomic operations.

**Atomic operations in concurrent programming are program operations that run completely independently of any other processes. Atomic operations are used in many modern operating systems and parallel processing systems**

8) What is spin locks

**A spinlock is a lock that causes a thread trying to acquire it to simply wait in a loop while repeatedly checking whether the lock is available.**

9) What is reader-writer spin lock.

**Sometimes, lock usage can be clearly divided into readers and writers. For example, consider a list that is both updated and searched. When the list is updated (written to), it is important that no other threads of execution concurrently write to or read from the list. Writing demands mutual exclusion. On the other hand, when the list is searched (read from), it is only important that nothing else write to the list.**

**When a data structure is neatly split into reader/writer paths like this, it makes sense to use a locking mechanism that provides similar semantics. In this case, Linux provides reader-writer spin locks. Reader-writer spin locks provide separate reader and writer variants of the lock.**

10) What is semaphore.

**A semaphore is a signaling mechanism, and a thread that is waiting on a semaphore can be signaled by another thread. It uses two atomic operations, 1) Wait, and 2) Signal for the process synchronization. A semaphore either allows or disallows access to the resource, which depends on how it is set up**.

11) What is binary semaphore

**A binary semaphore is restricted to values of zero or one, while a counting semaphore can assume any nonnegative integer value. A binary semaphore can be used to control access to a single resource. It can be used to enforce mutual exclusion for a critical section in user code.**

12) What is difference between semaphore and spin lock

**In spinlock, a process is waiting for lock will keep the processor busy by continuously polling the lock. In semaphore, a process is waiting for a semaphore will go into sleep to be woken up at a any time and the try for the lock again.**

14) What is difference between semaphore and mutex.

**A mutex object allows multiple process threads to access a single shared resource but only one at a time. On the other hand, semaphore allows multiple process threads to access the finite instance of the resource until available. In mutex, the lock can be acquired and released by the same process at a time.**

15) What is preemption disabling and what is the use of this.

**Memory Management**

1) How memory is managed in linux.

2) What are pages

**Paging is a storage mechanism used to retrieve processes from the secondary storage into the main memory in the form of pages. The main idea behind the paging is to divide each process in the form of pages. The main memory will also be divided in the form of frames**.

3) What are different memory zones in linux

**The Linux kernel divides memory into memory zones. On a mainframe, three zones are used: DMA, Normal , and Movable . Memory in the DMA zone is below 2 GB, and some I/O operations require that memory buffers are in this zone**

4) how to allocated pages and deallocate pages

**Allocating blocks of pages tends to fragment memory with larger blocks of free pages being broken down into smaller ones. The page deallocation code. recombines pages into larger blocks of free pages whenever it can. In fact the page block size is important as it allows for easy combination of blocks into larger blocks**.

6) what is kmalloc and what are action modifier we can pass while using kmalloc.

**kmalloc is the normal method of allocating memory for objects smaller than page size in the kernel. The flags argument may be one of: GFP\_USER - Allocate memory on behalf of user.**

7) what is zone modifier in linux.

**Zone modifiers specify from which memory zone the allocation should originate. Normally, allocations can be fulfilled from any zone. The kernel prefers ZONE\_NORMAL,**

8) what is vmalloc

**Vmalloc() is used to allocate memory from the kernel. The memory allocated is virtually contiguous but not physically contiguous. This API works in the similar manner as the user level malloc() works.It means vmalloc ensures memory is allocated from vitual address space.**

**Virtual File System**

1. What is virtual file system and what is the need of it in linux.

**The Virtual File System (also known as the Virtual Filesystem Switch) is the software layer in the kernel that provides the filesystem interface to userspace programs. It also provides an abstraction within the kernel which allows different filesystem implementations to coexist.**

2) What are different object types in VFS:

**The superblock object, which represents a specific mounted filesystem.**

**The inode object, which represents a specific file.**

**The dentry object, which represents a directory entry, a single component of a path.**

**The file object, which represents an open file as associated with a process.**

3) What are the operations possible on inode and superblock objects:

**Process Address Space**

1. What is process address space.

**An address space is a range of valid addresses in memory that are available for a program or process. That is, it is the memory that a program or process can access. The memory can be either physical or virtual and is used for executing instructions and storing data.**

1. What is memory descriptor in linux and which structure represents it.

**The kernel represents a process's address space with a data structure called the memory descriptor. This structure contains all the information related to the process address space. The memory descriptor is represented by struct mm\_struct and defined in <linux/sched**

3) How to allocate and destroy a memory descriptor

4) Does kernel thread has any association with mm\_struct.

5) What is VMA and what are various VMA operations possible in linux kernel.

6) How to manipulate memory area in linux.

7) What mmap and do\_mmap().

8) What is page tables.

**Page Table is a data structure used by the virtual memory system**to store the mapping between logical addresses and physical addresses**. Logical addresses are generated by the CPU for the pages of the processes therefore they are generally used by the processes.**

**Device Driver Questions**

1. What is device driver and what is the need of it?

**The main purpose of device drivers is to provide abstraction by acting as a translator between a hardware device and the applications or operating systems that use it.**

2) What are different kind of devices?

3) What is module in linux.

**Loadable kernel modules in Linux are loaded (and unloaded) by the modprobe command. They are located in /lib/modules or /usr/lib/modules with extension. ko**

4) How mudules are loaded in linux.

5) Difference between insmod and modprobe.

**Insmod**: Dependencies if present are not loaded, it requires complete module path

**Modprobe:** modprobe calculates dependencies, loads the dependencies and then the main module.

6) **How modprobe calculates dependencies?**

**Modprobe depends on depmod tool to calculate dependencies. depmod calculates dependencies of all the  modules present in /lib/modules/$(uname -r) folder, and places the dependency information in /lib/modules/$(uname -r)/modules.dep file**

6) How parameters are shared between driver modules.

**To allow arguments to be passed to your module, declare the variables that will take the values of the command line arguments as global and then use the MODULE\_PARM() macro, (defined in linux/module.h) to set the mechanism up. At runtime, insmod will fill the variables with any command line arguments that are given, like ./insmod mymodule.o myvariable=5.**

7) what are IOCTLS.

**is a**[**system call**](https://en.wikipedia.org/wiki/System_call)**for device-specific**[**input/output**](https://en.wikipedia.org/wiki/Input/output)**operations and other operations which cannot be expressed by regular system calls. It takes a parameter specifying a request code; the effect of a call depends completely on the request code.**

9) what are the benefits of syscalls.

**The most important benefit of a system call is simplicity. You should not have to write a complex program in order to open or save a file to the disk or print a document. Further, you don't want to have anything become compromised in the operating system, such as device drivers or other system components.**

10) how character driver is registered in linux.

11) What is init and exit function of a driver.

12) how and when init and exit function of driver get called.

13) what is probe function.

14) when probe is get called.

15) what is platform devices.

16) what is device tree.

A device tree is a tree data structure that describes the hardware configuration of the system to the Linux operating system. During boot, the Linux kernel will use the information in the device tree to recognize, load appropriate drivers and manage the hardware devices in the system.

18) what is sysfs and procfs.

19) how logs are printed in linux kernel and what are the logs level available in linux.

20) what is copy\_to\_user and copy\_from\_user.

copy\_to\_user. Copies a block of data from the kernel to user space. copy\_from\_user. Copies a block of data from user space to the kernel.

21) what do you mean by kernel configuration and what are the various way of configuring kernel.

22) what is menuconfig.

23) what is ioremap

24) what is seagmentation fault.

27) what is zimage and bzimage.

The file called zImage is the compressed kernel image that lives in arch/i386/boot after you issued make zImage or make boot -- the latter invocation is the one I prefer, as it works unchanged on other platforms. If you built a big zImage, instead, the file is called bzImage, and lives in the same directory

29) How parameters are passed from boot loader to kernel.

The U-Boot boot loader passes the kernel command-line transparently to the Linux kernel using a feature known as atags from a U-Boot variable named bootargs.

31) from which file kernel execution starts.

Once the kernel has started, it starts the init process. Historically this was the "SysV init", which was just called "init". More recent Linux distributions are likely to use one of the more modern alternatives such as systemd. Basically, these are grouped as operating system service-management.

33) what is primary and secondary bootloader.

You'd want a very basic bootloader to load a more complex one. For example, you might want to load your operating system from a filesystem. Code to handle a filesystem might not fit in your first bootloader but code to load a second bootloader does fit.

34) why we need two bootloader.

A bootloaders is used as a separate program in the program memory that executes when a new application needs to be reloaded into the rest of program memory. The bootloader will use a serial port, USB port, or some other means to load the application.

35) difference between poll and select.

The poll() API performs the same API as the existing select() API. The only difference between these two APIs is the interface provided to the caller.

The select() API requires that the application pass in an array of bits in which one bit is used to represent each descriptor number. When descriptor numbers are very large, it can overflow the 30KB allocated memory size, forcing multiple iterations of the process. This overhead can adversely affect performance.

The poll() API allows the application to pass an array of structures rather than an array of bits. Because each pollfd structure can contain up to 8 bytes, the application only needs to pass one structure for each descriptor, even if descriptor numbers are very large.

36) what is priority inheritance and priority inversion.

Basically, in *Priority Inversion*, higher priority task (H) ends up waiting for middle priority task (M) when H is sharing critical section with lower priority task (L) and L is already in critical section. Effectively, H waiting for M results in inverted priority i.e. Priority Inversion. One of the solutions for this problem is *Priority Inheritance*. In *Priority Inheritance*, when L is in critical section, L inherits priority of H at the time when H starts pending for critical section. By doing so, M doesn’t interrupt L and H doesn’t wait for M to finish.

37) what are different type of kernel.

* Monolithic Kernel.
* Micro Kernel.

38) what is DMA.

Direct memory access (DMA) is a method that allows an input/output (I/O) device to send or receive data directly to or from the main memory, bypassing the CPU to speed up memory operations.

39) what is cache coherency.

Cache coherence refers to the concept of shared resource data being stored in various local caches uniformly at the same time. This problem can be solved either through software or hardware, with the software option being tougher.

40) what is copy on write.

Copy-on-write or CoW is a technique to efficiently copy data resources in a computer system. If a unit of data is copied but not modified, the "copy" can exist as a reference to the original data.

41) what is highmem and lowmem.

Highmem is used to describe the virtual memory address space that is for user-space. This is in contrast with lowmem, the address space that is used for the kernel and is mapped into every user-space program's address space

42) what happens if we pass invalid address from userspace by using ioctls.

43) what are different ipc mechanism in linux.

* Shared files.
* Shared memory (with semaphores)
* Pipes (named and unnamed)
* Message queues.
* Sockets.
* Signals.

44) what are sockets.

A socket is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to. An endpoint is a combination of an IP address and a port number.

45) how page fault is handled in linux.

A page fault will happen if a program tries to access a piece of memory that does not exist in physical memory (main memory). The fault specifies the operating system to trace all data into virtual memory management and then relocate it from secondary memory to its primary memory such as a hard disk.

46) difference between memory based io and port based io.

Memory mapped I/O is mapped into the same address space as program memory and/or user memory and is accessed in the same way. Port mapped I/O uses a separate, dedicated address space and is accessed via a dedicated set of microprocessor instructions.

47) what is I2c and SPI.

I2C is a half-duplex communication protocol. SPI is a full-duplex commination protocol. I2C has the feature of clock stretching, which means if the slave cannot be able to send fast data as fast enough then it suppresses the clock to stop the communication. Clock stretching is not the feature of SPI.

48) how physical to virtual translations works in linux.

To map virtual memory addresses to physical memory addresses, page tables are used

49) what is thrashing, segmentation and fragmentation

Thrashing**is a condition or a situation when the system is spending a major portion of its time in servicing the page faults, but the actual processing done is very negligible.**

**Segmentation of memory is a method of allocating multiple chunks of memory (per task) for different purposes and allowing those chunks to be protected from each other. In Linux a task's code, data, and stack sections are all mapped to a single segment of memory.**

**(External) memory fragmentation is a long-standing Linux kernel programming issue.**

**As the system runs, it assigns various tasks to memory pages. Over time, memory gets fragmented,**

**and eventually, a busy system that is up for a long time may have only a few contiguous physical pages**

50) what is preempt\_count and what is the need of that.

**Each task has a field, preempt\_count, which marks whether the task is preemptible. The count is incremented every time the task obtains a lock and decremented whenever the task releases a lock**