**======================================================================**

**LINUX(General)**

**======================================================================**

**red 5**

1.What is Linux booting process? Explain each stage in the Linux booting process from power is ON? What will be initailized in ROM code?

Ans:

<http://www.thegeekstuff.com/2011/02/linux-boot-process>

2.Explain Booting process of any SOC? Explain each stage of booting process?

Ans:Soc means System-on-chip.

<http://www.thegeekstuff.com/2011/02/linux-boot-process>

or Boot process related to omap.

[http://changyuheng.me/slides-boot-sequence/#/12](http://changyuheng.me/slides-boot-sequence/" \l "/12)

3)Why we need X-loader? Can we modIfy the X-loader code? Who will modIfy the

x-loader code?

Ans:

Because the internal static ram is very small (64k), xloader is stripped down to the essentials and is used to initialize memory and enough of the peripheral devices to access and load the second stage loader (UBoot) into main memory.

Yes,We can modify the xloader code.

End user can modify the code.

4) How to get event on booting process?

Ans:

5)What is the difference between RTOS and GPOS? What is difference between Linux and RTOS?

Ans:

GPOS  
  
1.It is used in multiuser environment   
2.protected memory model  
3.non scalable  
4.high interrupt latency   
5.time insensitive  
  
RTOS  
  
1.it is dedicated to a single work  
2.flat memory model  
3.scalable   
4.low interrupt latency   
5.time sensitive

6)What is U-boot?

Ans:

Monolithic code image

Runs processor in physical or a single address space

Enables clocking, sets up some of the pin mux settings

Reads in Kernel image (uImage)

Jumps to load address pointed to in uImage header

What are environment variables and how are they used

Default environment variables and how they are used

Passes Kernel Command Line to Kernel

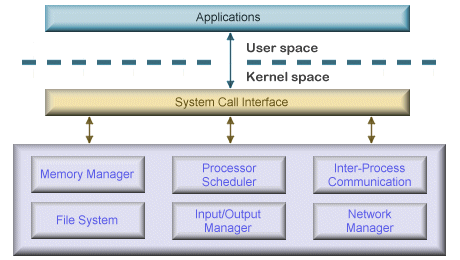
ATAGs (Memory, Command Line, etc)

Flattened Device Trees (FTDs) in the future

Debugging capabilities (just mentioning, not used during boot process)

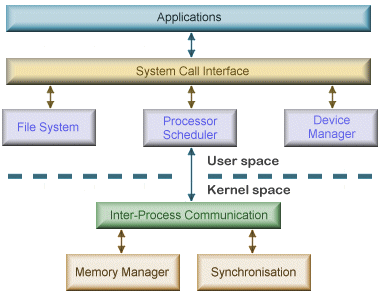
7)Explain OS architecture?

Ans:



Monolithic kernel architecture

The core services of an operating system are collectively known as the kernel. The kernel has unrestricted access to all of the resources on the system. In early monolithic systems, each component of the operating system was contained within the kernel, could communicate directly with any other component, and had unrestricted system access. While this made the operating system very efficient, it also meant that errors were more difficult to isolate, and there was a high risk of damage due to erroneous or malicious code.

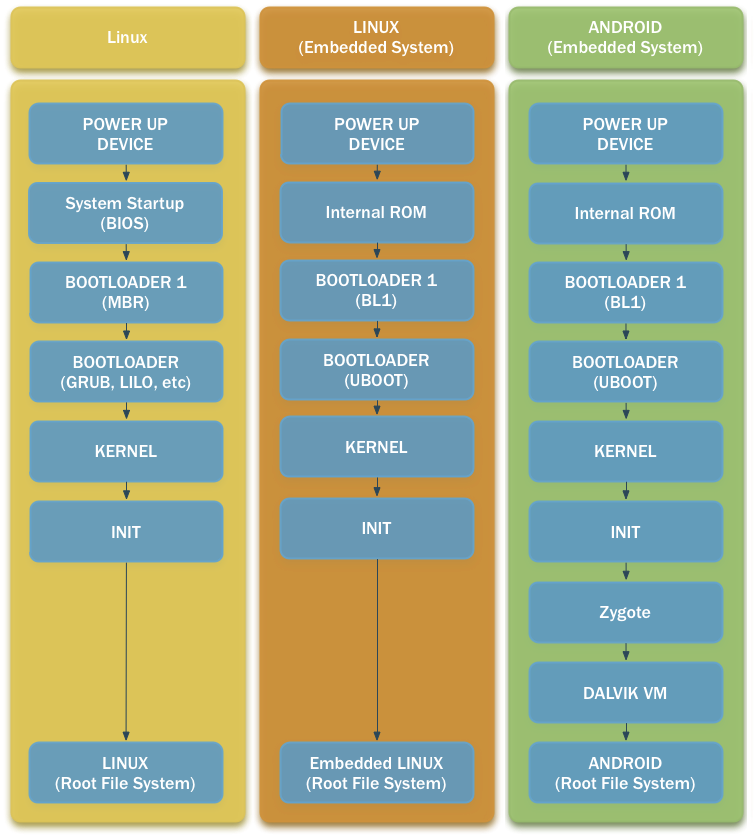


MicroKernel OS architecture

A microkernel architecture includes only a very small number of services within the kernel in an attempt to keep it small and scalable. The services typically include low-level memory management, inter-process communication and basic process synchronisation to enable processes to cooperate. In microkernel designs, most operating system components, such as process management and device management, execute outside the kernel with a lower level of system access. Microkernels are highly modular, making them extensible, portable and scalable. Operating system components outside the kernel can fail without causing the operating system to fall over. Once again, the downside is an increased level of inter-module communication which can degrade system performance.

8)DIfference b/w Linux boot and android booting process?

Ans:



9)Explain the flow of program execution with compilation process and memory?

Ans: Please refer to memory layout and compilation process of C program

10)Explain about eCos operating system system calls and OS concepts?

Ans:The **Embedded Configurable Operating System** (**eCos**) is a free and open source real-time operating system intended for embedded systems and applications which need only one process with multiple threads.

11)What is Buffer address? where we will use?

Ans:A data **buffer** or just **buffer** is a region of a physical memory storage used to temporarily store data while it is being moved from one place to another.

12)What is cross compilation?

Ans:Cross-compilation is the act of compiling code for one computer system (often known as the target) on a different system, called the host.

It's a very useful technique, for instance when the target system is too small to host the compiler and all relevant files.

Common examples include many embedded systems, but also typical game consoles.

13)When you run the program on target board if program crashes then How you can analyse the code?

Ans:By enabling debugging,in the host system during cross compilation.

By using debugging procedure we can analyse where the crash occured.

14)What is coredump?

Ans:A core dump is a file containing a process's address space (memory) when the process terminates unexpectedly. Core dumps may be produced on-demand (such as by a debugger), or automatically upon termination. Core dumps are triggered by the kernel in response to program crashes, and may be passed to a helper program (such as systemd-coredump) for further processing. A core dump is not typically used by an average user, but may be passed on to developers upon request where it can be invaluable as a post-mortem snapshot of the program's state at the time of the crash, especially if the fault is hard to reliably reproduce.

15)How to do flashing?

Ans:

16)How to debug whether the device has hardware or software issue?

Ans:

17)What is a stack frame, stack pointer & frame pointer ?

Ans:

A **stack pointer** is a small register that stores the address of the last program request in a **stack**. A **stack** is a specialized buffer which stores data from the top down. As new requests come in, they "push down" the older ones.

A **stack frame** is a **frame** of data that gets pushed onto the **stack**. In the case of a call **stack**, a **stack frame** would represent a function call and its argument data. If I remember correctly, the function return address is pushed onto the **stack** first, then the arguments and space for local variables.

A **frame pointer** of a given invocation of a function is a copy of the stack **pointer** as it was before the function was invoked.

18)Write steps in Boot sequence in Linux based systems?

Ans:<http://www.thegeekstuff.com/2011/02/linux-boot-process>

19)What is object file and what are symbols ?

Ans:

20)Can you tell the memory layout of a process based on Data,BSS,HEAP and STACK?

Ans:<http://www.geeksforgeeks.org/memory-layout-of-c-program/>

21)What is the use of the poll(file, polltable) API ?

Ans:

22)List key features and achivements of Linux. Why it is so popular?

Ans:

Portable – Portability means softwares can works on different types of hardwares in same way.Linux kernel and application programs supports their installation on any kind of hardware platform.

Open Source – Linux source code is freely available and it is community based development project. Multiple teams works in collaboration to enhance the capability of Linux operating system and it is continuously evolving.

Multi-User – Linux is a multiuser system means multiple users can access system resources like memory/ ram/ application programs at same time.

Multiprogramming – Linux is a multiprogramming system means multiple applications can run at same time.

Hierarchical File System – Linux provides a standard file structure in which system files/ user files are arranged.

Shell – Linux provides a special interpreter program which can be used to execute commands of the operating system. It can be used to do various types of operations, call application programs etc.

Security – Linux provides user security using authentication features like password protection/ controlled access to specific files/ encryption of data.

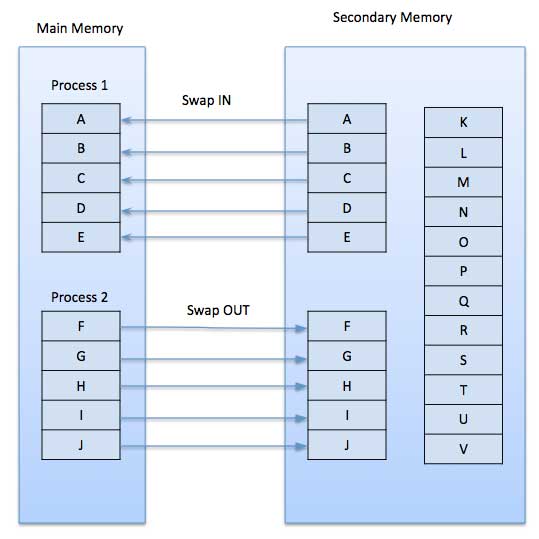
#### Linux is fast, free and easy to use, power laptops and servers around the world. Linux has many more features to amaze its users such as:

* Live CD/USB: Almost all Linux distributions have Live CD/USB feature by which user can run/try the OS even without installing it on the system.
* Graphical user interface (X Window System): People think that Linux is a command line OS, somewhere its true also but not necessarily, Linux have packages which can be installed to make the whole OS graphics based as Windows.
* Support’s most national or customized keyboards: Linux is used worldwide and hence available in multiple languages, and supports most of their custom national keyboards.
* Application Support: Linux has its own software repository from where users can download and install thousands of applications just by issuing a command in Linux Terminal or Shell. Linux can also run Windows applications if needed.

23)Explain demand paging with an example?

Ans:

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. 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.



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.

### Advantages

Following are the advantages of Demand Paging −

* Large virtual memory.
* More efficient use of memory.
* There is no limit on degree of multiprogramming.

### Disadvantages

* Number of tables and the amount of processor overhead for handling page interrupts are greater than in the case of the simple paged management techniques.

24)What is use of Chmod – command?

Ans:To change the read ,write and execute permissions of a file.

On Linux and other Unix-like operating systems, there is a set of rules for each file which defines who can access that file, and how they can access it. These rules are called file permissions or file *modes*. The command name **chmod** stands for "change mode", and it is used to define the way a file can be accessed.

25)What is meant by gcc, gmake, ctags?

Ans:

The GCC (GNU Compiler Collection) is widely regarded as the most important piece of *free software*. Formerly called the GNU C Compiler, the GCC now contains compilers for the C, C++, Objective C, Fortran, Java and Ada programming languages.

The **GNU Compiler Collection** (**GCC**) is a compiler system produced by the GNU Project supporting various programming languages. GCC is a key component of the GNU toolchain and the standard compiler for most Unix-like Operating Systems. The Free Software Foundation (FSF) distributes GCC under the GNU General Public License (GNU GPL). GCC has played an important role in the growth of free software, as both a tool and an example.

**gmake (GNU make - called simply make on linux systems)** is a tool to help you build a program from its source. For our trivial Zoo program its possible to completely build the Zoo.exe from scratch in a few seconds. However, in HEP there are some very large programs made with a number of libraries with hundreds and possibly thousands of source files and then rebuilding from scratch can takes hours and is not practical during program development.

Having once built a program it isn't necessary to completely rebuild it after making some changes. Only things that are out of date need to be rebuilt and the rules are very simple:-

1. Recompile any .o file if its source has changed.
2. Recreate any library if any of its .o files have changed.
3. Rebuild any executable if its libraries have changed.

However, applying such rules to a large software project would be both tedious and error prone. Instead the gmake tool is used. You create a special file called a makefile (the default name is GNUmakefile) which contains instructions on how to create all object, library and executable files and then pass this file to gmake along with a request to build the program. gmake applies the instructions recursively and rebuilds just those parts that need to be rebuilt.

**Ctags** is a tool that will sift through your code, indexing methods, classes, variables, and other identifiers, storing the index in a tags file. The tags file contains a single tag per line. Depending on command line arguments and the language ctags is run against, a lot of information can be obtained from this index.

Ctags makes it much easier to navigate a larger project, particularly if the code you’re working with is unfamiliar. If you’re unsure of what a method does or how it’s supposed to be called, you can jump straight to it’s definition.

26)Can a Linux machine be configured as a router? How?

Ans: Yes.

<http://www.yourownlinux.com/2013/07/how-to-configure-ubuntu-as-router.html>

27)What is Boot loader? What are the functions of bootloader? What is the difference between primary boot loader and secondary bootloader? How does a bootloader passes the parameters to the Kernel?

Ans:

A boot loader primarily manages and executes the boot sequence of a computer system. A boot loader program is typically started after the computer or the BIOS have finished performing the initial power and hardware device checks and tests. It fetches the OS kernel from the hard disk or any specified boot device within the boot sequence, into the main memory. A boot loader is associated with only a single operating system. An operating system can also have multiple boot loader programs classified as primary and secondary boot loaders, where a secondary boot loader might be larger and more capable than the primary boot loader.

28)What is a toolchain? Why its required?

Ans:A toolchain is the set of tools that compiles source code into executables that can

run on your target device, and includes a compiler, a linker, and run-time libraries.

Initially, you need one to build the other three elements of an embedded Linux

system: the bootloader, the kernel, and the root filesystem. It has to be able to

compile code written in assembly, C, and C++ since these are the languages

used in the base open source packages.

29)What is the significance of $# in shell scripting?

Ans:<http://osr507doc.xinuos.com/en/OSUserG/_Passing_to_shell_script.html>

30)Explain about how to find 99th element using head and tail concept in shell scripting?

Ans:

head -99 /path/to/file/ | tail -1

31)Explain the scenario of test case creation in shell scripting?

Ans:

32)Difference between spinlock, mutex, and semaphore.?

Ans:

Mutex provides one person to access a single resource at a time, others must wait in a queue. Once this person is done, the guy next in the queue acquire the resource.  
So access is serial, one guy after other. Too aggressive.

Semaphore are useful if multiple instances (N) of a resource is to be shared among a set of users. As soon as all N resources are acquired, any new requester has to wait. Since there is no single lock to hold, there is as such no ownership of a semaphore.

Spinlock is an aggressive mutex. In mutex, if you find that resource is locked by someone else, you (the thread/process) switch the context and start to wait (non-blocking).  
Whereas spinlocks do not switch context and keep spinning. As soon as resource is free, they go and grab it. In this process of spinning, they consume many CPU cycles. Also, on a uni-processor machine they are useless and perform very badly.

33)Difference between mutex and spinlock? Where we used spinlock? How spinlock works ?

Ans:

<http://mescal.imag.fr/membres/arnaud.legrand/teaching/2010/OS_l7.pdf>

**Mutex vs. Spinlocks**

Can you use spinlocks instead of mutexes?

On x86, requires the CPU to lock memory system around read

and write

Prevents other uses of the bus (e.g., DMA)

Usually runs at memory bus speed, not CPU speed

Much slower than cached read/buffered write

Causes ping-pong cache line migration

Wastes CPU, especially if thread holding lock not running

**Spinlocks are often used by the kernel as a low-level mutex**

**to implement a higher level mutex**

**Spinlocks**

busy waiting but the only way to really implement

mutual exclusion

When the Critical Section is long, this is extremely inefficient

Higher-level

**mutex**

“few” busy waiting

With mutex, busy waiting is limited to the CSs of

lock,unlock,wait,signal which are short sections (about ten instructions).

Therefore, the CS os mutexes is almost never occupied and busy-waiting occurs rarely and only for a short time.

On multiprocessor, sometimes good to spin for a bit, then yield

34)What kind of priority is maintained in Linux?

Ans:<https://www.nixtutor.com/linux/changing-priority-on-linux-processes/>

**======================================================================**

**LINUX (User Space)**

**======================================================================**

1)Explain about IPC mechanisms. What is the fastest IPC? Why?

Ans:

<http://www.tldp.org/LDP/tlk/ipc/ipc.html>

Shared memory is faster because the data is not copied from one address space to another, memory allocation is done only once, andsyncronisation is up to the processes sharing the memory. mostly, shared memory just give more control but if you use your shared memory like a pipe, it probably won't make much difference depending on your kernel implementation.

2)If any crashes occurs while booting up any chip then how to debug that , how will get to know where is the crash.?

Ans:

3)Difference between Kernel space and User space. How context switching happens between user space to kernel space? Please write a few systems call examples, which switches from the program control from kernel space to user space.?

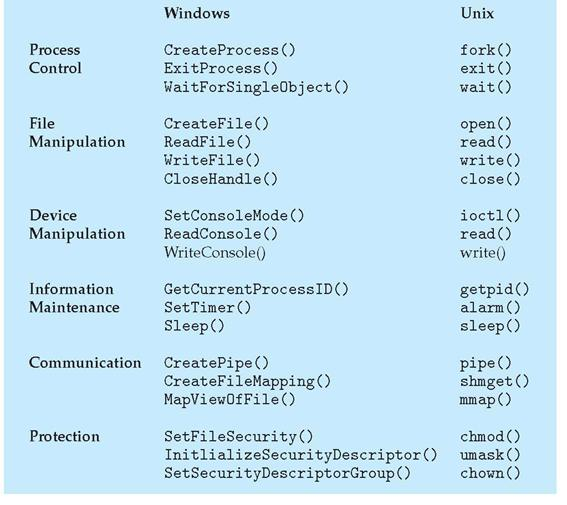
Ans:

The user space, which is a set of locations where normal user processes run (i.e everything other than the kernel). The role of the kernel is to manage applications running in this space from messing with each other, and the machine.

* The kernel space, which is the location where the code of the kernel is stored, and executes under.

Processes running under the user space have access only to a limited part of memory, whereas the kernel has access to all of the memory. Processes running in user space also don't have access to the kernel space. User space processes can only access a small part of the kernel via an interface exposed by the kernel - the system calls. If a process performs a system call, a software interrupt is sent to the kernel, which then dispatches the appropriate interrupt handler and continues its work after the handler has finished.

<http://www.cs.iit.edu/~cs561/cs450/system_calls/style/8.html>



4)What is synchronization? How we implement that in the regular process? What type of synchronization techniques are present in the Linux? What are different synchronization mechanisms, why to use them and when to use which?

Ans:

synchronization refers to one of two distinct but related concepts: synchronization of processes, and synchronization of data. Process synchronization refers to the idea that multiple processes are to join up or handshake at a certain point, in order to reach an agreement or commit to a certain sequence of action. Data synchronization refers to the idea of keeping multiple copies of a dataset in coherence with one another, or to maintain data integrity. Process synchronization primitives are commonly used to implement data synchronization.

Implementation of synchronisation B/w Process is by using Semaphores and b/w threads by using mutex and semaphores.

Suppose that two processes 1 and 2 execute concurrently to access a common integer variable, balance. An example would be let's say that thread Process 1 might handle credits to an account, while Process 2 handles the debits. Both need access to the account balance variable at different times. This Example involves two process with synchronisation using semaphore. If it is a single process involving both, then we use Thread and posix semaphores

5.What are the differences between process and thread. What is real time application of process and thread. How to synchronize both. Why thread is called as light weight Process? Generally what resources are shared between threads. How to handle signal in process and thread?

Ans:<http://www.thegeekstuff.com/2013/11/linux-process-and-threads>

<http://www.programmerinterview.com/index.php/operating-systems/thread-vs-process/>

<https://stackoverflow.com/questions/4623335/thread-synchronization-vs-process-synchronization>

<https://unix.stackexchange.com/questions/225687/what-happens-to-a-multithreaded-linux-process-if-it-gets-a-signal>

6.Is synchronization needed with two threads,If yes How u will do with semaphores and mutex?,If no What Is the reason?? How it can run with the virtual memory?

Ans:<https://unix.stackexchange.com/questions/149424/are-lock-mutex-and-semaphore-for-between-threads-or-between-processes>

<https://www.justsoftwaresolutions.co.uk/threading/locks-mutexes-semaphores.html>

<https://softwareengineering.stackexchange.com/questions/134333/do-threads-use-virtual-memory-or-real-memory>

7.What are the differences between Mutex vs Semaphore? Explain difference between binary semaphore and mutex? Are binary semaphore and mutex same? Give example of mutex and semaphore? When to use mutex and when to use semaphore?

Ans:<http://www.geeksforgeeks.org/mutex-vs-semaphore/>

8.What will happen When one process write the data in the pipe and exit, another process is not available to read? Can it read later?What type of error will occur?

Ans:It has to read at that time only,if one process write data completely,at that if it fails,it can’t read any more.If it tries to read it will raise an error,i.e,pipe broken.

9.Is it possible to communicate userspace with hardware without Kernel? NO? Yes?

Ans:Yes.By using ioctl calls and mmap.

10.What is userspace?

Ans:<https://unix.stackexchange.com/questions/87625/what-is-difference-between-user-space-and-kernel-space>

11.What is Deadlock? Give an example. How you will avoid that? Are deadlocks for the multiple processes? Write a code for deadlock?

Ans:<http://whatis.techtarget.com/definition/deadlock>

<https://www.quora.com/Can-a-single-process-be-deadlocked>

12.What is Race condition? Write code for racecondition?

Ans:<http://searchstorage.techtarget.com/definition/race-condition>

13.If I declare int a=10, int b=20, int c=30 in three threads should I use these variables each in other threads?

Ans:No

14.How to use one variable in all threads?

Ans:global variables

15.How many locking mechanisms we have? Explain thread locking mechanisms?

Ans:<https://www.slideshare.net/kerneltlv/linux-locking-mechanisms>

16.What is race around condition?

Ans:A *race condition* is a special condition that may occur inside a critical section. A *critical section* is a section of code that is executed by multiple threads and where the sequence of execution for the threads makes a difference in the result of the concurrent execution of the critical section.

When the result of multiple threads executing a critical section may differ depending on the sequence in which the threads execute, the critical section is said to contain a race condition. The term race condition stems from the metaphor that the threads are racing through the critical section, and that the result of that race impacts the result of executing the critical section.

17.What is atomic code?

Ans:

An operation during which a processor can simultaneously read a location and write it in the same bus operation. This prevents any other processor or I/O device from writing or reading memory until the operation is complete. Atomic implies indivisibility and irreducibility, so an atomic operation must be performed entirely or not performed at all.

18.What is atomic programming/non­locking operation? Explain atomic operations?

Ans:

In [concurrent programming](https://en.wikipedia.org/wiki/Concurrent_programming), an operation (or set of operations) is **atomic**, **linearizable**, **indivisible** or **uninterruptible** if it appears to the rest of the system to occur instantaneously. Atomicity is a guarantee of [isolation](https://en.wikipedia.org/wiki/Isolation_(computer_science)) from [interrupts](https://en.wikipedia.org/wiki/Interrupt), [signals](https://en.wikipedia.org/wiki/Signal_(IPC)), [concurrent](https://en.wikipedia.org/wiki/Concurrency_(computer_science)) [processes](https://en.wikipedia.org/wiki/Process_(computing)) and [threads](https://en.wikipedia.org/wiki/Thread_(computing)). It is relevant for [thread safety](https://en.wikipedia.org/wiki/Thread_safety) and [reentrancy](https://en.wikipedia.org/wiki/Reentrancy_(computing)). Additionally, atomic operations commonly have a [succeed-or-fail](https://en.wikipedia.org/wiki/Atomicity_(database_systems)) definition—they either successfully change the state of the system, or have no apparent effect.

In a concurrent system, processes can access a shared object at the same time. Because multiple processes are accessing a single object, there may arise a situation in which while one process is accessing the object, another process changes its contents

19.How to debug a multi threaded application code?

Ans:<https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/4/html/Debugging_with_gdb/threads.html>

20.Explain process memory section and used of all section?

Ans:

21.What is Shared library and Dynamic library?

Ans:<http://www.yolinux.com/TUTORIALS/LibraryArchives-StaticAndDynamic.html>

22.If two threads are using same global variable do we need a mutex in single processor system?

Ans:

yes.

23.Explain the procedure of child process creation? Give 3 examples for using fork and exec functions?

Ans: <http://www.makelinux.net/alp/024>

24.Explain Prority inversion?

Ans:<https://barrgroup.com/Embedded-Systems/How-To/RTOS-Priority-Inversion>

25.What is paging and How it is different from segmentation?

Ans:<https://en.wikipedia.org/wiki/Memory_segmentation>

26.How do you debug whether the crash happened in your code or application?

Ans:

27.What is cooperative multitasking and pre­emptive multitasking. What is limitation of cooperative multitasking.?

Ans:

28.What should be done for Shared library based linking in gcc? What should be done for static library based linking in gcc ?

Ans:<https://blog.feabhas.com/2014/04/static-and-dynamic-libraries-on-linux/>

29.What is dynamic linking ? What is static linking ? List the differences. What are the advantages of Dynamic linking or Sharedlibraries ?

Ans:http://cs-fundamentals.com/tech-interview/c/difference-between-static-and-dynamic-linking.php

30.What basis we decide to create static and dynamic libraries?

Ans:

31.Tell the role of brk() in malloc / Tell the relation between heap and brk?

Ans:

32.What is process descriptor. How to manipulate the current process. What is process context?

Ans:

33.How threads are implemented in Linux Kernel?

Ans:

34.What are different states of a process in Linux?

Ans:

35.What is thread\_info structure and what is the need of that structure?

Ans:

36.How parent less process is handles in Linux?

Ans:init process

37.What is process scheduling? What are the available scheduling classes in Linux? How next task is picked for scheduling. What is scheduler entry point in Linux?

Ans:

38)I/O bound versus Processor bound process.

Ans:

CPU Bound means the rate at which process progresses is limited by the speed of the CPU. A task that performs calculations on a small set of numbers, for example multiplying small matrices, is likely to be CPU bound.

I/O Bound means the rate at which a process progresses is limited by the speed of the I/O subsystem. A task that processes data from disk, for example, counting the number of lines in a file is likely to be I/O bound.

Memory bound means the rate at which a process progresses is limited by the amount memory available and the speed of that memory access. A task that processes large amounts of in memory data, for example multiplying large matrices, is likely to be Memory Bound.

Cache bound means the rate at which a process progress is limited by the amount and speed of the cache available. A task that simply processes more data than fits in the cache will be cache bound.

I/O Bound would be slower than Memory Bound would be slower than Cache Bound would be slower than CPU Bound.

39)What is context switch? Explain context switching, PCB. How context switching is handled in Linux.

Ans:

40)Explain fork system call in Linux. Explain differences between fork() and vfork()? Why is vfork() not much useful now?

Ans:

41)What is a Thread? Why Multithreading? How are threads scheduled in Linux? How to implement Multi threading and its uses.Please write an example C code which uses multithreading and thread synchronization. OS being linux.

Ans:<http://www.onlamp.com/pub/a/onlamp/2002/11/07/linux_threads.html>

42)What is segmentation­fault? When it occurs? Please write example code which generates segmentation fault using C and Linux calls. How you debug when it occurs?

Ans:A common run-time **error** for **C** programs by beginners is a "**segmentation** violation" or "**segmentation fault**." When you run your program and the system reports a "**segmentation** violation," it means your program has attempted to access an area of memory that it is not allowed to access.

<http://www.cprogramming.com/debugging/segfaults.html>

43)What is memory leak? How you detect and debug memory leak issues?

Ans:<http://www.geeksforgeeks.org/what-is-memory-leak-how-can-we-avoid/>

By using valgrind tool we can detect memory leaks.

44)What is stack over flow? How you detect and debug when it occures?

Ans:<http://whatis.techtarget.com/definition/stack-overflow>

45)Whats Priority Inversion? and How to overcome it?

Ans:<http://www.geeksforgeeks.org/priority-inversion-what-the-heck/>

priority inversion can be overcome by priority inheritance.

<http://www.geeksforgeeks.org/whats-difference-priority-inversion-priority-inheritance/>

46)Difference between Multithreading and Multitasking?

Ans:<http://techdifferences.com/difference-between-multitasking-and-multithreading-in-os.html>

47)What is POSIX createThread, joinThread? What is Thread safe code?

Ans:

**pthread\_create** creates a new thread and makes it executable. This routine can be called any number of times from anywhere within your code.

* pthread\_create arguments:
* thread: An opaque, unique identifier for the new thread returned by the subroutine.
* attr: An opaque attribute object that may be used to set thread attributes. You can specify a thread attributes object, or NULL for the default values.
* start\_routine: the C routine that the thread will execute once it is created.
* arg: A single argument that may be passed to *start\_routine*. It must be passed by reference as a pointer cast of type void. NULL may be used if no argument is to be passed.

**pthread\_join (threadid, status)**

The pthread\_join() function waits for the thread specified by *thread* to terminate. If that thread has already terminated, then pthread\_join() returns immediately. The thread specified by *thread* must be joinable.

**A threadsafe function** protects shared resources from concurrent access by locks. Thread safety concerns only the implementation of a function and does not affect its external interface.

In C language, local variables are dynamically allocated on the stack. Therefore, any function that does not use static data or other shared resources is trivially threadsafe, as in the following example:

/\* threadsafe function \*/

int diff(int x, int y)

{

int delta;

delta = y - x;

if (delta < 0)

delta = -delta;

return delta;

}

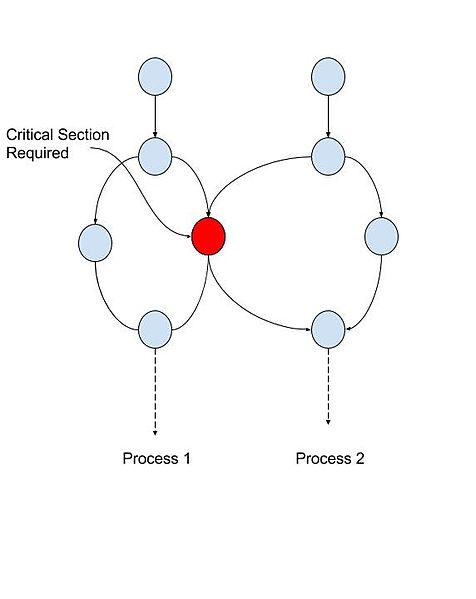
48)Whats preemption? Is it supported in Linux?

Ans:<http://www.embeddedlinux.org.cn/embeddedlinuxprimer/0136130550/ch17lev1sec2.html>

49)What is process starvation, how Linux handles it?

Ans:

50)What is Critical Section code?

Ans:In concurrent programming, concurrent accesses to shared resources can lead to unexpected or erroneous behavior, so parts of the program where the shared resource is accessed are protected. This protected section is the critical section or critical region. It cannot be executed by more than one process. Typically, the critical section accesses a shared resource, such as a data structure, a peripheral device, or a network connection, that would not operate correctly in the context of multiple concurrent accesses.

By carefully controlling which variables are modified inside and outside the critical section, concurrent access to the shared variable are prevented. A critical section is typically used when a multi-threaded program must update multiple related variables without a separate thread making conflicting changes to that data. In a related situation, a critical section may be used to ensure that a shared resource, for example, a printer, can only be accessed by one process at a time.

51)What is ALSA? explain the architecture.

Ans:

[https://www.google.co.in/search?q=alsa+architecture&client=ubuntu&hs=q0G&channel=fs&source=lnms&tbm=isch&sa=X&ved=0ahUKEwjFn4bMsvnVAhXErI8KHcwkAtYQ\_AUICigB&biw=1375&bih=771#imgrc=xYluQI9a4GDF8M](https://www.google.co.in/search?q=alsa+architecture&client=ubuntu&hs=q0G&channel=fs&source=lnms&tbm=isch&sa=X&ved=0ahUKEwjFn4bMsvnVAhXErI8KHcwkAtYQ_AUICigB&biw=1375&bih=771" \l "imgrc=xYluQI9a4GDF8M):

The Advanced Linux Sound Architecture (ALSA) provides audio and MIDI functionality to the Linux operating system. ALSA has the following significant features:

* Efficient support for all types of audio interfaces, from consumer sound cards to professional multi channel audio interfaces.
* Fully modularized sound drivers.
* SMP and thread-safe design.
* User space library (alsa-lib) to simplify application programming and provide higher level functionality.
* Support for the older Open Sound System (OSS) API, providing binary compatibility for most OSS programs.

52)How to find the process id in Shell scripting?

Ans:

#!/bin/bash

echo "PID of the script: $$"

echo "PPID if the script: $PPID"

echo "UID of this script: $UID"

53)What are daemons and please provide a few examples of daemons.?

Ans:A **daemon** (also known as background processes) is a Linux or UNIX program that runs in the background. For example, httpd the daemon that handles the Apache server, or, sshd which handles SSH remote access connections. Linux often start daemons at boot time. Shell scripts stored in /etc/init.d directory are used to start and stop daemons.

## List of service daemons for Linux and Unix-like systems

* amd - Auto Mount Daemon
* anacron - Executed delayed cron tasks at boot time
* apmd - Advancere or Android softwad Power Management Daemon
* atd - Runs jobs queued using the at tool
* autofs - Supports the automounter daemon allowing mount and unmount of devices on demand
* crond - The task scheduler daemon
* cupsd - CUPS printer daemon
* dhcpd - Dynamic Host Configuration Protocol and Internet Bootstrap Protocol Server
* ftpd - FTP Server Daemon
* gated - routing daemon that handles multiple routing protocols and replaces routed and egpup
* httpd - Web Server Daemon
* inetd - Internet Superserver Daemon
* imapd - An imap server daemon
* lpd - Line Printer Daemon
* memcached - In-memory distributed object caching daemon

54)Write a complete program to print the process ID in Linux?

Ans:

#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

int main()

{

printf("Process id is %d Parent:%d\n",getpid(),getppid());

return 0;

}

55)Explain the following signals with situation and examples a.SIGABRT b.SIGTERM c.SIGPIPE d.SIGUSR1?

Ans:

**SIGABRT**:The abort signal is normally sent by a program when it wants to terminate itself and leave a core dump around. This can be handy during debugging and when you want to check assertions in code. The easiest way for a process to send a *SIGABRT* to itself is via the (ANSI-C) *abort()* function, which does nothing more than a (also ANSI-C) *raise(SIGABRT),* which, in turn, does a (not ANSI-C) *kill(getpid(), SIGABRT*). Programs normally drop a core file and exit when receiving this signal

**SIGTERM**:This signal is sent when the user wants a program to terminate gracefully. By default, the program exits immediately, but this signal is often caught, and the program performs a graceful exit as quickly as possible. The kill command sends this signal by default, and many users know it as “signal 15.”

**SIGPIPE**:When you use the write or similar data sending functions and there is nobody to receive your data SIGPIPE is delivered. This is a very common case and you must remember that those functions may not only exit with error and setting the errno variable but also cause the SIGPIPE to be delivered to the program. An example is the case when you write to the standard output and the user uses the pipeline sequence to redirect your output to another program. If the program exits while you are trying to send data SIGPIPE is sent to your process. A signal is used in addition to the normal function return with error because this event is asynchronous and you can't actually tell how much data has been successfully sent. This can also happen when you are sending data to a socket. This is because data are buffered and/or send over a wire so are not delivered to the target immediately and the OS can realize that can't be delivered after the sending function exits.

**SIGUSR1:Both of these signals are left for processes to use for anything they like. There is no defined purpose for them, and they are often used for groups of processes to work together.**

**======================================================================**

**LINUX(Kernel Space)**

**======================================================================**

**red 5**

1.Explain Dynamic Memory Partitioning in paging

A:

<http://computersciencecafe.blogspot.in/2010/11/operating-system-memory-management-part.html>

<https://www.go4expert.com/articles/memory-allocation-schemes-t22406/>

2.Explain about the Kernel memory regions? How we allocate memory in kernel? Explain about kmalloc and its flags clearly?

A:

3.How will you make interrupt handlers as fast as possible? What we don’t do in the interrupt Handler ? How you allocate memory in

interrupt handler?

A:

4.How can we insert the module in run time? How many ways can do? What is dIfference between insmod and modprobe? How

modprobe works?

A:

5.What happens when an interrupt raised by an device? Explain interrupt mechanism in Linux?

6.What is top of interrupt and bottom of interrupt. Difference between top half and Bottom half? When we use Bottom half? What is the

difference between Softirq and Tasklet?

7.What are tasklets ? How are they activated ? when and How are they initialized ? When will you use it and How is it different to

ISRs?

8.hat are the types of softirqs ? Difference between Timer Softirq and Tasklet Softirq ?

9.What is a system call? How system call works? How system calls are implemented in Linux? Explain about trap. How system call

arguments are passed to the Kernel?

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

11.What is the interrupt latency? Explain interrupt mechanism in Linux?

12.When a module is added in the Kernel statically, then Kernel is compiled, but new Kernel image is not working? How you resolve it?

13.Have you come across Kernel panic? What is Kernel panic?

14.What is need for Kernel? How to debug Kernel?

15.Write a code for spin lock?

16.What are Kernel thread. How to create Kernel threads?

17.What is work queue?

18.Tell me how keyboard scans the input provided in terms of OS internals

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

20.kmalloc and vmalloc dIfferences?

21.user space malloc created then How Kernel knows about the created malloc??

22.Let us say, driver is not present then, how will you work on it for application??

23.What is the difference between virtual address and physical address. Write in detail about the address translation scheme with a

diagram and explanation

24.What are interrupts and How they work or how the interrupts are handled in any application?

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

26.What is virtual memory, why we need virtual memory and How to implement the virtual memory.

27.What happens if Interrupt comes in a thread execution and How to handle it.

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

29.What is Device tree ? How to enter the new device information in the Device tree? Which file is generated after device tree

compilation? How will you make changes in device tree

30.What is Kernel. How it works

31.Explain about Kernel switching

32.Explain MMU in detail with diagram. Also write example code and explain with relevant instructions.

33.Is it possible to do nesting of interrupts.

34.What are the different OS scheduling procedures. What are the types of Scheduling Mechanisms in Linux.Explain each in detail

Changes needed in source if you are adding a new driver to Kernel

35.How communication is done from Kernel to user level?

36.Flow of application to Kernel for particular module?

37.What is the difference between General purpose and Real time Operating systems

38.How the Interrupts are Working On Linux Kernel?

39.What is the latest Linux­Kernel version? What are the new features added in latest Kernel compare to previous release?

40.What are character devices? How does the character device driver adds and remove itself from the Kernel ?

41.How will you Access userspace memory from Kernel? What are the various methods?

42.What are the main components of a Linux device driver? Describe all. What is Call back mechanism in Linux device driver

programming ?

43.What is the use of file­>private\_data in a device driver structure ?

44.What is the role of interrupts in a device driver ? How are interrupts handled in device driver ? What are the precautions needs to be

taken while writing ISRs?

45.What is MMAP()? Advantages of MMAP over Read ?

46.Tell the relation between Malloc and MMAP. Example of using MMAP and MUNMAP in C ?

47.What is SMP, AMP? List the differences (Hint CPU)

48.What are the ways in which Linux Kernel can be compiled ?

49.How do you get the list of currently available drivers? How will get the driver added into the Kernel ? What are Kconfig files ?

50.What is a Kernel module? How will you list the modules?

51.What is the use of ioctl(inode,file,cmd,arg) API?

52.What is a device number ?

53.What are the two types of devices drivers from VFS point of view ?

54.What is task\_struct and How are task states maintained ?

55.What is rwlock and spinlock ? Briefly explain about both of them ?

56.When will you use rwlock instead of spinlock ?

57.Can spinlock/rwlock be used in Interrupt handler ?

58.How will you trace the system calls made into the Kernel of Linux ?

59.Tell about the method/steps in Linux Kernel Compilation.

60.What is Kmalloc and How does it differ from normal malloc ? or Why can’t we use malloc in Kernel code ?

61.What happens as soon as a packet arrives from the network in Linux ?

62.What is the difference between IO mapped vs memory mapped?

68.What is zombie process. Describe the situation when you create

69.What is virtual run time.

70.What is user preemption and Kernel preemption

71.What happens when process in userspace calls a syscall.

72.What is the need of verifying parameter in definition of syscall.

73.Why it is not recommended to writing new syscall.

74.How can ISR contribute to entropy in Linux Kernel to generate random numbers?

75.Can we acquire mutex/semaphore in an Interrupt Service Routine?

76.What is the difference between IO mapped vs memory mapped?

77.What are the ways in which Linux Kernel can be compiled ?

78.How will get the driver added into the Kernel ? What are kconfig files ?

79.Whats Inode in Linux?

80.What is GDB? Can you put a breakpoint in ISR?

81.What is Interrupt vector?

82.What is sk\_buff in Linux Kernel

83.What is page fault?

84.If I allocate the memory kmalloc(100), How much of memory will be allocated ?

85.What is page\_alloc() function ?

86.Explain About KGDB, gdb?

87.How to get the Kernel logs?

88.What is ISR mechanism in Linux, what are its characterisctics and functionalities? Can we use a printk in Interrupt service routine?

Justify the answer. Is conditional execution allowed in ISR.

90.What is “init” and its role?

91.What are the tools used to debug Kernel and describe them?

92.What are the COPY\_TO\_USER and COPY\_FROM\_USER functions ?

93.What is Make file? What type of changes we need to do when a new driver is added? How do you test the driver?

94.How do you debug a Kernel error using an address?

95.What is a filesystem? how do we build a filesytem busybox/ buildroot.

96.How do you share a function between different drivers in Kernel?

97.How does a Kernel space driver informs/notify a userspace program?

98.What is Linux sysfs and how do we use it to find the status of a device in Kernel?

99.What is a function pointer, give an example of its usage in Kernel.

100.Please draw the kernel memory management architecture and explain in detail each memory types in detail.

101.Please write in detail about cache coherence, explain the scenario with example and sample code

102.Please explain in detail Kernel Symbol table architecture

103.Write different types of schedulers and scheduling are available in the operating system