

<p>How to decode kernel trace</p>	<p>There are 3 options objdump,addr2line and crash viewer.</p>
<p>git</p>	<p>Git is a distributed version control system(VCS) that enables where the complete codebase — including its full version history — is mirrored on every developer's computer.</p> <p>In Git, "origin" is a shorthand name for the remote repository that a project was originally cloned from. More precisely, it is used instead of that original repository's URL - and thereby makes referencing much easier.</p> <p>If you run the command git remote -v it will list all the remote repositories that are linked to your local repository. There you will see that in order to push or fetch code from your remote repository you will use the shortname 'origin'.</p> <pre>\$ git remote -v origin https://github.com/schacon/ticgit (fetch) origin https://github.com/schacon/ticgit (push) git remote show origin git remote rename pb paul \$ git remote origin paul git tag git tag -a v1.4 -m "my version 1.4"</pre> <p>Git Aliases</p> <pre>git config --global alias.co checkout \$ git config --global alias.br branch \$ git config --global alias.ci commit</pre> <p>You clone a repository with <code>git clone <url></code>. For example, if you want to clone the Git linkable library called libgit2, you can do so like this:</p> <pre>\$ git clone https://github.com/libgit2/libgit2</pre> <p>Git has three main states that your files can reside in: modified, staged, and committed:</p> <p>The Git directory is where Git stores the metadata and object database for your project. This is The most important part of Git, and it is what is copied when you clone a repository from another computer.</p> <p>Git Large File Storage (LFS) replaces large files such as audio samples, videos, datasets, and graphics with text pointers inside Git, while storing the file contents on a remote server like GitHub.com or GitHub Enterprise.</p>

Git log is a utility tool to review and read a history of everything that happens to a repository

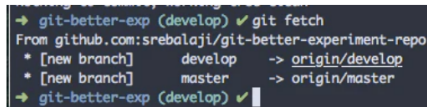
git fetch <remote>

git pull=git fetch + merge.

git fetch <remote>

To say it in simple words, it fetches all the changes from the remote and stores it in your local but doesn't merge those changes locally. Simple..!

```
git fetch
```



```
git-better-exp (develop) ✓ git fetch
From github.com:srebalaji/git-better-experiment-repo
 * [new branch]      develop    -> origin/develop
 * [new branch]      master     -> origin/master
git-better-exp (develop) ✓
```

In the above image, you can see that two new branches are created **origin/develop** and **origin/master**. These branches will have the updated changes from the remote. But these changes are not affected to **master** and **develop** branches.

The command goes out to that remote project and pulls down all the data from that remote project that you don't have yet

After fetching, you can view the changes made in the remote

```
git diff develop origin/develop
```

```
git diff develop origin/develop --stat
```

git push <remote> <branch>.

The above command will first execute **git fetch** first. Then it will execute the **git merge** command in the working directory. It means that, if you are in the master branch then the remote changes in master will be merged to the local master branch.

```
git pull origin develop
```

This command will only fetch the remote changes of develop branch and not other branches. And it also merges the remote changes to the branch.

Basic

1. `git --version`
2. `cat ~/.gitconfig`

3. git config --list
4. git diff
5. git diff --staged same git diff --cached

1. git help
2. git help config
3. git add -h

ignoring files

```
$ cat .gitignore
*.[oa]
*~
```

Connect to remote repository

Method 1> create a new file using command line

1. git init
2. git add README.md
3. git commit -m "first commit"
4. git status

Method 2> cloning to existing repository

1. git clone <https://github.com/vishalgupta95/c-programming.git>
2. git remote -v
3. git fetch
4. git pull
5. git add g.txt
6. git checkout -- <file>..." to discard changes in working directory)
7. git diff --cached to see what you've staged so far (--staged and --cached are synonyms):
8. git status -s
9. git log
10. git log -p -2
11. git checkout -- <file> is a dangerous command revert
12. git restore --staged CONTRIBUTING.md
13. git diff --cached to see what you've staged so far (--staged and --cached are synonyms):
14. git rm
15. git mv
16. git stash
17. git branch -r
18. git log --oneline
19. git log --pretty=format:"%h - %an, %ar : %s"

20. `git log --oneline --decorate`
21. `git commit -m 'Initial project version'`

`git revert fd73451 // undo`

1. `git revert HEAD // redo (important)`

Git branch

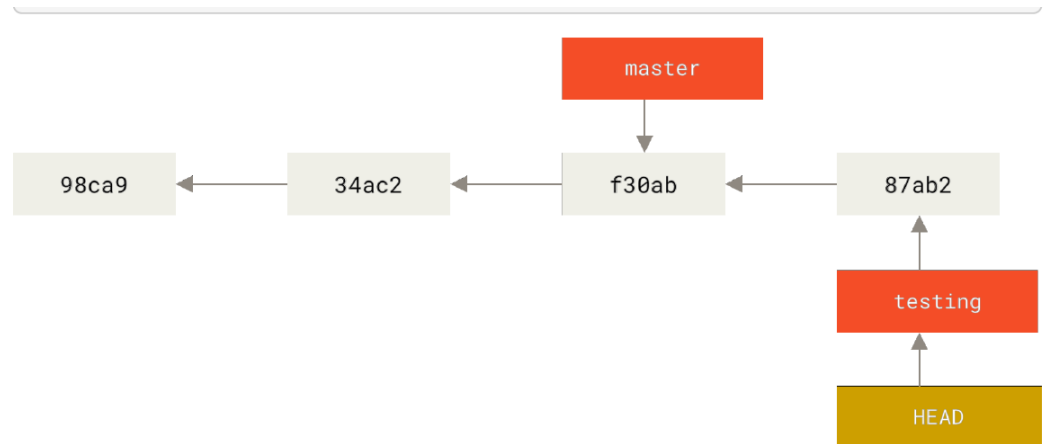


Figure 15. The HEAD branch moves forward when a commit is made

1. `git log --oneline --decorate`
2. `git log --oneline --decorate --graph --all`
3. `git branch feature`
4. `git branch`
5. `git checkout feature (now in feature)`
6. `git branch -D feature // unsafe`
7. `git checkout -b new // create a new branch on master using Switched to a new branch 'new'`

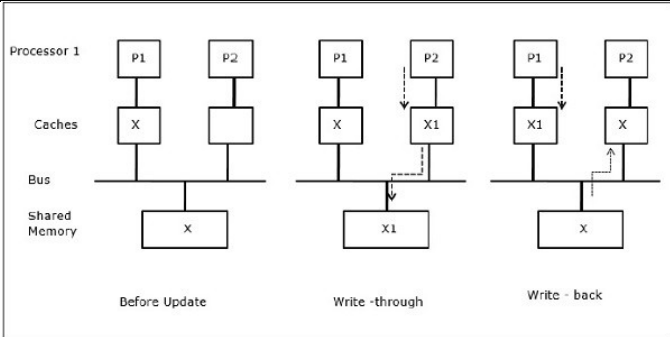
Git merge

1. `git checkout master -> Switched to branch 'master'`
2. `git merge feature --> merge`
3. `git branch -d feature // safe, taken care of unmerge changes`

`git log testing`. To show all of the branches, add `--all` to your `git log` command.

	<p>Git stash</p> <p>Stashing takes the dirty state of your working directory — that is, your modified tracked files and staged changes — and saves it on a stack of unfinished changes that you can reapply at any time</p> <p>(even on a different branch).git stash : Stash the changes in a dirty working directory away .</p> <p>git add git status git stash : Stash the changes in a dirty working directory away . git checkout some_branch (change to intended branch -) git stash list (list stashes) git stash apply (to apply stash to working tree in current branch) git stash apply stash@{12} (if you will have many stashes you can choose what stash will apply -- in git stash drop stash@{0} (to remove from stash list -- in this case stash 0) git stash pop stash@{1} (to apply selected stash and drop it from stash list)</p> <p><i>Figure 79. Example history before a cherry-pick</i></p> <p>If you want to pull commit <code>e43a6</code> into your <code>master</code> branch, you can run:</p> <pre>\$ git cherry-pick e43a6 Finished one cherry-pick. [master]: created a0a41a9: "More friendly message when locking the index fails." 3 files changed, 17 insertions(+), 3 deletions(-)</pre>
<p>YAML</p>	<p>YAML (Yet Another Markup Language) is data serialization language. Generally, it is used to store configuration. YAML is preferred because it is in human readable format., YAML is a strict superset of JSON, But because it's a strict superset, it can do everything that JSON can and more. One major difference is that newlines and indentation mean something in YAML, as opposed to JSON, which uses brackets and braces</p>
<p>How GDB works</p>	<p>GDB will replace a program instruction with some other instruction (The INT 3 instruction) that will cause an exception, and then when it's encountered, GDB will take the exception and stop the program. When the user says to continue, GDB will restore the original instruction, single step, re-insert the trap, and continue.</p> <p>Downside of SW breakpoints is that debugger needs to be able to modify running program, which is not possible if program is running from read-only memory (quite common in embedded world).</p>

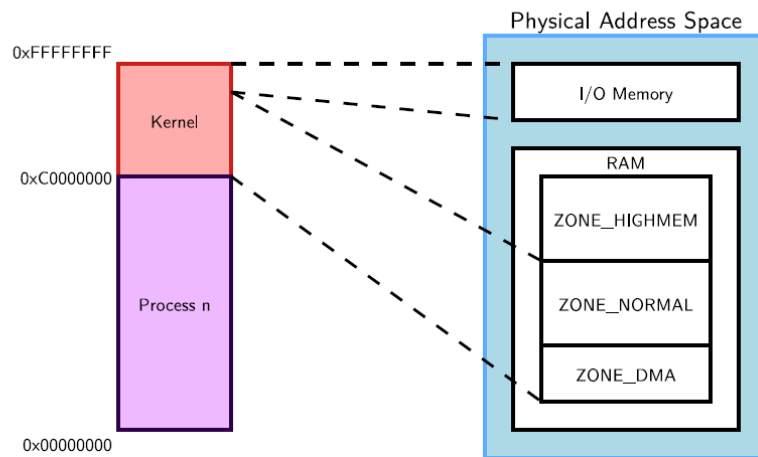
iptables	<p>iptables is a command-line firewall utility that uses policy chains to allow or block traffic. iptables uses three different chains: input, forward, and output.</p> <p>lot of protocols will require two-way communication, so both the input and output chains will need to be configured</p> <pre>iptables --policy INPUT ACCEPT iptables --policy OUTPUT ACCEPT iptables --policy FORWARD ACCEPT</pre> <p>//This example shows how to block all connections from the IP address 10.10.10.10. iptables -A INPUT -s 10.10.10.10 -j DROP</p>
Sub interfaces	<p>A sub interface is a virtual interface created by dividing one physical interface into multiple logical interfaces.</p> <p>Why: If we had a router with only 1 physical interface, but needed to have the router connected to two IP networks, so that it could do routing, we could create 2 logical sub interfaces, assign each sub interface an IP address within each subnet, and we can then route between them.</p>
Virtual Routing and Forwarding	<p>Virtual Routing and Forwarding or VRF allows a router to run more than one routing table simultaneously. When running more routing tables in the same time, they are completely independent.. It is possible to use same VRF instance on more routers and connect every instance separately using VRF dedicated router port or only a sub-interface.</p> <p>You can find VRFs to be used on ISP side. Provider Edge (PE) routers are usually running one VRF per customer VPN so that one router can act as a PE router for multiple Customer Edge (CE) routers even with more customers exchanging the same subnets across the VPN. By running VRF per customer, those subnets will never mix in-between them.</p> <p>VRF uses the same methods of virtualization as virtual LANs (VLANs). Also, VLAN can make a single switch appears as a multiswitch, whereas VRF can make a single router appear to be multiple routers.</p> <p>On any newly installed Nexus switch, there are two VRFs that exist by default, which are named default and management.</p> <p>Routing protocols are run in the default VRF context unless another VRF context is Specified. The default VRF uses the default routing context for all show commands. The default VRF is similar to the global routing table concept.</p> <p>Only the mgmt0 interface can be in the management VRF; the mgmt0 interface cannot be assigned to another VRF. No routing protocols can run in the management VRF (static routing only).</p>

	<p>On the Nexus platform, two VRFs exist by default—management and default:</p> <p>NX-7K-1-Daisy# sho vrf</p> <table><tr><th>VRF-Name</th><th>VRF-ID</th><th>State</th><th>Reason</th></tr><tr><td>default</td><td>1</td><td>Up</td><td>--</td></tr><tr><td>management</td><td>2</td><td>Up</td><td>--</td></tr></table>	VRF-Name	VRF-ID	State	Reason	default	1	Up	--	management	2	Up	--
VRF-Name	VRF-ID	State	Reason										
default	1	Up	--										
management	2	Up	--										
Page Table	<p>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.</p>												
TLB	<p>Translation Lookaside Buffer (TLB) is special cache used to keep track of recently used transactions. TLB contains page table entries that have been most recently used. CPU generates virtual (logical) address. the processor examines if the page is already in main memory (TLB hit) and Corresponding frame number is retrieved, if not in main memory a page fault is issued. then the TLB is updated to include the new page entry. if space is not there, one of the replacements techniques comes into picture i.e either FIFO, LRU or MFU etc</p>												
Cache	<div><p>The diagram illustrates three cache write policies for Processor 1, which has two processors (P1, P2) and two caches (X, Y). The bus connects the caches to shared memory.</p><ul style="list-style-type: none">Before Update: P1 is in cache X, P2 is in cache Y. Both caches contain data X. Shared memory contains X.Write-through: P2 updates its cache (X1) with new data. The data is immediately written to shared memory (X1).Write-back: P2 updates its cache (X1) with new data. The data is not immediately written to shared memory. Instead, the cache is marked as dirty. When P2 later reads, it gets the new data from the cache. When P1 later reads, it gets the old data from the cache, and the system must write back the dirty data from P2's cache to shared memory.</div> <p>What is cache flushing?</p> <p>When the amount of unwritten data in the cache reaches a certain level, the controller periodically writes cached data to a drive. This write process is called "flushing."</p> <p>When the processor needs to read or write a location in main memory, it first checks for a corresponding entry in the cache.</p> <p>If the processor finds that the memory location is in the cache, a cache hit has occurred and data is read from cache. If the processor does not find the memory location in the cache, a cache miss has occurred. For a cache miss, the cache allocates a new entry and copies in data from main memory, then the request is fulfilled from the contents of the cache.</p> <p>Cache Write Policies</p>												

	<p>Write back: Write operations are usually made only to the cache. Main memory is only updated when the corresponding cache line is flushed from the cache.</p> <p>Write through: All write operations are made to main memory as well as to the cache, ensuring that main memory is always valid.</p> <p>From the above description write back policy results in inconsistency. If two caches contain the same line, and the line is updated in one cache, the other cache will unknowingly have an invalid value. Subsequently read to that invalid line produce invalid results. But if we think deeper even the Write through policy also has consistency issues. Even though memory is updated inconsistency can occur unless other cache monitor the memory traffic or receive some direct notification of the update.</p>
Cache Coherence (consistent) Protocols in multiprocessor system	<p>1>software Level Solution — Potential cache coherence problem is transferred from run time to compile time. How to handle shared data– time wise . etc</p> <p>2> Hardware Solutions</p> <p>1>When a processor writes a new value into its cache, the new value is also written into the memory module. We update the other cache copies by doing a broadcast</p> <p>2> When a processor writes a new value into its cache, broadcasting the invalidation request through the system.</p>
Data center fabric	<p>A data center fabric is a system of switches and servers and the interconnections between them that can be represented as a fabric. Cisco offering includes fabric management via Application Policy Infrastructure Controller (APIC) or Data Center Network Manager (DCNM)</p>
Switch words	<p>Supervisor engine: it is basically the control plane.</p> <p>Line card: data plane/responsible for packet forwarding</p> <p>Fabric module: It interconnects two different line-cards also supervisor card of the switch.</p> <p>Fabric extender: a line card connected using fabric has no capability to store a forwarding table or run any control plane protocols. They are fully managed as part of the parent switch and do not require independent software upgrades, config backups, or maintenance.</p>

Virtual Memory Organization in 32-bit systems

Physical / virtual memory mapping (on 32 bit)



Paging is a function of memory management where a computer will store and retrieve data from a device's secondary storage to the primary storage. OS reads data from secondary storage in blocks called **pages**, all of which have identical size. The physical region of memory containing a single page is called a frame.

Virtual Memory Organization in 32-bit systems

High 1GB reserved for kernel-space

Next 3GB exclusive mapping available for each user space

4GB space is also configurable CONFIG_VMSPLIT_2G

Because of hardware limitations, the **kernel cannot treat all pages as identical**. Some pages, because of their physical address in memory, cannot be used for certain tasks. Because of this limitation, the **kernel divides pages into different zones**. The kernel uses the zones to group pages of similar properties. Linux has to deal with two shortcomings of hardware with respect to memory addressing:

Some hardware devices are capable of performing DMA (direct memory access) to only certain memory addresses.

Some architectures are capable of physically addressing larger amounts of memory than they can virtually address. Consequently, some memory is not permanently mapped into the kernel address space.

Because of these constraints, there are three memory zones in Linux:

ZONE_DMA This zone contains pages that are capable of undergoing DMA.

	<p>ZONE_NORMAL This zone contains normal, regularly mapped, pages.</p> <p>ZONE_HIGHMEM This zone contains "high memory," which are pages not permanently mapped into the kernel's address space.</p>
copy_from_user	<div data-bbox="542 384 1170 638"></div> <p>The access ok function</p> <p>In a virtual memory environment, there's no guarantee that the whole block of memory that you pass to write () is actually in RAM at the time. It (or part of it) could be a memory-mapped file, or it (or part of it) could be paged out.</p> <p>When any data is passed to the kernel space from userspace, it is the responsibility of the kernel developer to make sure that everything is sanitized.</p> <p>The copy_to_user function copies a block of data from the kernel into user space. the page tables could change at any time, requiring the desired pages to be pinned into memory so that they could not be swapped out while being addressed.</p> <p>If a kernel can handle page faults, there is perhaps no use (ignoring glory details like security for simplicity).</p> <p>One of the additional things that copy_from_user does is disable SMAP . supervisor Mode Access Prevention) (if enabled) while accessing userspace memory. If SMAP is enabled, accessing userspace address is not allowed.</p>
Definitions	<p>The root filesystem is the top-level directory of the filesystem. It must contain all the files required to boot the Linux system before other filesystems are mounted.</p> <p>The "sysroot" is the location the cross compiler will look for header files and libraries. The sysroot directory acts as if it is the root of the system,</p> <p>clangd understands your C++ code and adds smart features to your editor: code completion compile errors, go-to-definition and more.</p> <p>Electric Make® ("eMake"), is a new Make version .You can invoke eMake interactively or through build scripts. Electric Accelerator is a software build</p>

	accelerator that dramatically reduces software build times by distributing the build over a large cluster of inexpensive servers.
Yocto build	<p>Yocto Project: A Linux Foundation project that acts as an umbrella for various efforts to improve Embedded Linux.</p> <p>BitBake: Bit Bake is a program written in the Python language. At the highest level, Bit Bake interprets metadata, decides what tasks are required to run, and executes those tasks. Similar to GNU Make, Bit Bake controls how software is built. GNU Make achieves its control through "make files". Bit Bake uses "recipes".</p> <p>Metadata is any data that describes other data. Document metadata gives information About a document such as the author, when it was created when it was last modified, and its size.</p> <p>In Yocto project – Metadata is collection of below items</p> <div><div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></</div></div></div>

the source code, what are the flags should pass at the time of configuration or compilation.

Yocto terms

1>**Poky** is the name of build system used by yocto

2> **Layer**: A collection of recipes. Typically, each layer is organized around a specific theme, e.g. adding recipes for building web browser software. Open Embedded-Core is a base layer of recipes, classes and associated files that is meant

3> All the artefacts generated are stored in the deploy folder.tmp/log/cooker will have all logs

4>**Recipe script**: Bit Bake Recipes, which are denoted by the file extension .bb, are the most basic metadata files. These recipe files provide Bit Bake with the following: version, existing Dependencies
how to compile .

1>Locate and download source code ,

2>Unpack source into working directory

3>Apply any patches Perform any necessary pre-build configuration

4>Compile the source code

5>Installation of resulting build artifacts in WORKDIR

6>Copy artifacts to sysroot ,Create binary package(s)

We have a bbappend file that supplies a set of patches. It currently has the unintended side-effect of patching both the native version used during the Yocto build process, and the eventual target version. How do I modify the recipe such that it only acts upon the target version?

SRC_URI_append_class-target = " file://..."

SRC_URI Where to obtain the upstream sources and which patches to apply (this is called "fetching")

bitbake-layers create-layer ->Use a new custom layer for modularity and maintainability. They all start with "meta-" by convention

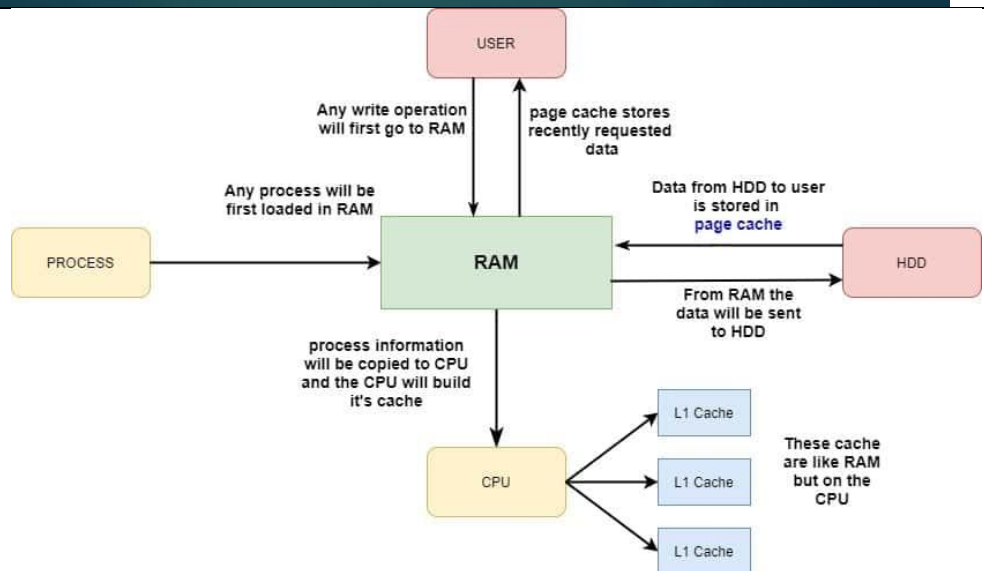
Class files(.bbclass) extension, contain information that is useful to share between metadata files. The BitBake source tree currently comes with one class metadata file called base.bbclass. You can find this file in the class's directory. The base.bbclass is special since it is always included automatically for all recipes and classes. This class contains definitions for standard basic tasks such as fetching, unpacking, configuring (empty by default),

	<p>compiling (runs any Makefile present), installing (empty by default) and packaging (empty by default). These tasks are often overridden or extended by other classes added during the project development process.</p> <p>Append files, which are files that have the .bbappend file extension, add or extend build information to an existing recipe file.</p> <p>busybox_1.21.%. bbappend That append file would match any busybox_1.21.x.bb version of the recipe. So, the append file would match the following recipe names: busybox_1.21.1.bb busybox_1.21.2.bb busybox_1.21.3.bb</p>
NX-Linux (NXL)	<p>NXL is a Linux Distribution for NXOS based on XE linux distro (Yocto Thud) with NXOS customizations including GCC 5.2 and Clang 7.0. We're based off of XELinux thud release. as of right now, we have a snapshot of their layers in our gitlab and are using that. Some of the layers XELinux uses come from yocto, XE Linux may have made changes on top of those layers. No changes have been directly made to any of the layers coming from them. If any changes need to be made to the recipes, then there's a bbappend file in the meta-nx-linux layer, which we created and control. Layer information is on the build wiki page, and are locally cloned into our gitlab group.</p> <p>Following git repository mirrors are setup from XE sources - meta-open embedded, meta-virtualization, meta-security, scripts, meta-nx-Linux (Specific to NXOS for FOSS customizations)</p> <p>Open-source packages should be sourced from the NX-Linux Distro</p>
initramfs	<p>initramfs, short for initial RAM filesystem, is a cpio archive of the initial filesystem loaded into memory after the kernel finishes initializing the system and before user-space begins the init process. The Linux kernel mounts the contents of initramfs as the initial root filesystem, before the real root (e.g. on your hard drive) is mounted. This initial root contains files needed to mount the real root filesystem and initialize your system—the most important bits being kernel modules. This feature is made up from a cpio archive of files that enables an initial root filesystem and init program to reside in kernel memory cache, rather than on a ramdisk, as with initrd filesystem initrd is for Linux kernels 2.4 and lower.</p>
SPI or I2C? serial port Communication	<p>SPI and I2C were both conceived in the 1980s—SPI by Motorola and I2C by Philips (now NXP). SPI exists as a de facto standard, while I2C is more formalized. Both are synchronous protocols, appropriate for short distance communications, and they generally operate in the 3.3 or 5V range.</p> <p>SPI is better for high speed and low power applications, while I2C is better suited for communication with a large number of peripherals, as well as in situations involving dynamic changing of the primary device role among peripherals on the I2C bus.</p>

I2C vs SPI

I2C	SPI
Supports multi-master	Doesn't support multi-master
2 wire connection	4 wire connection
Half duplex	Full duplex
Slower	Faster
Cheaper and easier	Costly and harder
Acknowledgement bit	No acknowledgement bit
Extra overhead	No extra overhead

Kernel interview



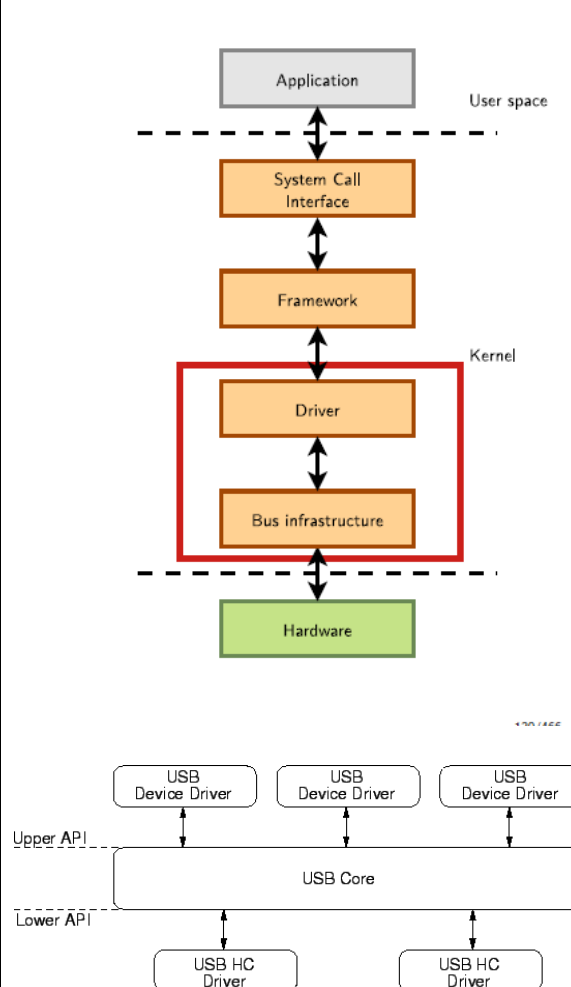
kernel code has to supply its own library implementations (memcpy, crypto, tar No memory protection, oops. Never use floating point numbers in kernel code. Your code may need to run on low-end processor without a floating-point unit. Fixed stack (8 or 4 KB) size Unlike user space, no Swapping, don't used recursion

User mode driver- written using user space language Perl, propriety, cannot crash the kernel. can use floating-point computation. Potentially higher performance. Especially for memory-mapped Devices due to avoidance of system calls UMD, drawback is Increased interrupt latency Less straightforward to handle interrupts

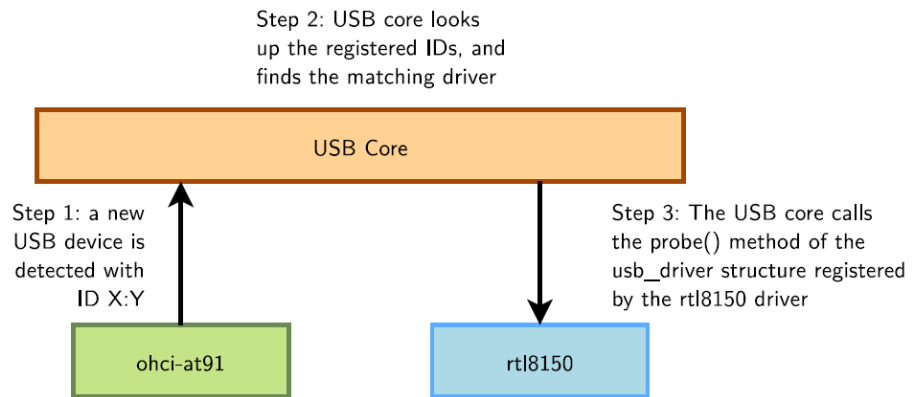
Many embedded architectures (x86,ppc) have lot of non-discoverable hardware (serial, Ethernet, I2C, Nand flash, UART controllers, Ethernet controllers, SPI or I2C controllers, graphic or audio devices. Depending on the architecture, such hardware is either described in BIOS ACPI tables (x86), using C code directly within the kernel, or using a special hardware description language in a *Device Tree*. Each node can have a number of properties describing various properties of the devices: addresses, interrupts, clocks, power, pin muxing, consumptions etc. At boot time, the kernel is given a compiled version, the Device Tree Blob, which is parsed to instantiate all devices described in the DT.

► The typical boot process is therefore:

1. Load zImage at address X in memory
 2. Load <board>.dtb at address Y in memory
 3. Start the kernel with `bootz X - Y`
- The - in the middle indicates no *initramfs*



The USB core now knows the association between the vendor/product IDs

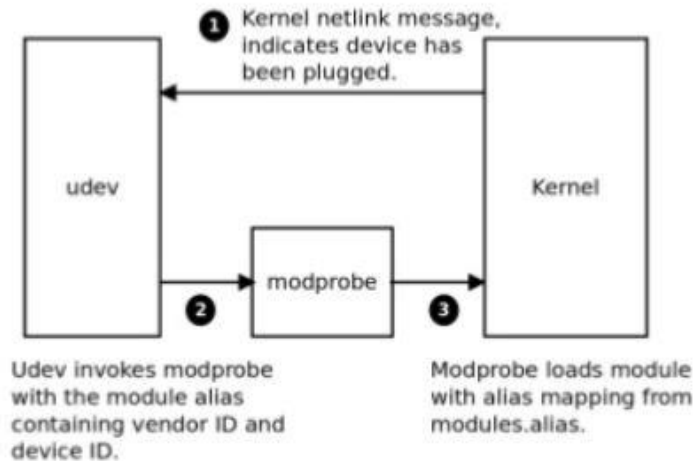


When a USB device is detected with id xxx USB Device controller try to find matching device driver and Called Probe function . The -probe() function is responsible for initializing the device , mapping I/O memory, registering the interrupt and registering it in the appropriate kernel

Slab allocation is a memory management mechanism intended for the efficient memory allocation of kernel objects. It eliminates fragmentation caused by allocations and deallocations. The technique is used to retain allocated memory that contains a data object of a certain type for reuse upon subsequent allocations of objects of the same type.

Modules are dynamic plugin, stored as a separate file in the filesystem so no possible With Module reduce boot time and image size, signed modules

udev (userspace /dev) Udev is the device manager for the Linux 2.6 kernel that creates/removes device nodes in the /dev directory dynamically. It is the successor of devfs and hotplug. It runs in userspace, and the user can change device names using Udev rules.



A very important UNIX design decision was to represent most system objects as Files. It allows applications to manipulate all system objects with the normal file API (open, read, write, close, etc.) So, devices had to be represented as files to the applications

```
ls -l /dev/ttyS0 /dev/tty1 /dev/sda /dev/sda1 /dev/sda2 /dev/sdc1 /dev/zero
```

Example C code that uses the usual file API to write data to a serial port

```
int fd;
fd = open("/dev/ttyS0", O_RDWR);
write(fd, "Hello", 5);
close(fd);
```

Within the kernel, all block and character devices are identified using a *major* and a *minor* number. The *major number* typically indicates the family of the device. The *minor number* allows drivers to distinguish the various devices they manage. Most major and minor numbers are statically allocated

1. Dmesg - kernel keeps its messages in a circular buffer
2. CONFIG_COMPAT is a config flag. 64 bit kernel supports for 32 bit emulation
3. Modern SoCs (System on Chip) include more and more hardware blocks pins are Multiplexed
4. three types of devices: - network, block(usb,harddisk) ,serial and others (graphics)
all block and character devices are identified using a major and a minor number. – represent as file
5. "Zero-copy" describes computer operations in which the CPU does not perform

	<p>the task of copying data from one memory area to another.</p> <ol style="list-style-type: none"> 6. Kmalloc calls slab page is usually 4K, but can be 8k 16k, PIO - IN and OUT instructions 7. Kernel pre-emption, if enabled, causes the kernel to switch from the execution 8. Mutex - The kernel's main locking primitive. It's a binary lock , mutex_trylock Use mutexes in code that is allowed to sleep -not in spinlock 9. Spinlocks cause kernel pre-emption to be disabled on the CPU executing them, No sleeping, several variants like Doesn't disable interrupts, Disables software interrupts, but not hardware ones 10. lock-free algorithms -rcu lock, atomic instructions 11. Some device controllers embedded their own DMA controller, DMA deals with physical addresses But the DMA does not access the CPU cache, so one needs to take care of cache coherency (cache content vs. memory content).
interrupt handler	<p>Each device must register its interrupt handler. whenever an interrupt occurs the OS does the most important part of handler "upper half" to respond to interrupt, create a data structure containing device specific data called "lower half" for later processing when CPU becomes available. This way interrupt handlers can be used in Bottom halves are required because as we know when ISR executes, it disables all other interrupts on running processor and same interrupt on all processors. To increase the response time and throughput of the system, we need to finish ISR as soon as possible.</p> <p>Acknowledge the interrupt to the device (otherwise no more interrupts will be generated, or the interrupt will keep firing repeatedly</p>

choice of different bottom half implementations

	Softirqs	Tasklets	Work Queues
Execution context	Deferred work runs in interrupt context.	Deferred work runs in interrupt context.	Deferred work runs in process context.
Reentrancy	Can run simultaneously on different CPUs.	Cannot run simultaneously on different CPUs. Different CPUs can run different tasklets, however.	Can run simultaneously on different CPUs.
Sleep semantics	Cannot go to sleep.	Cannot go to sleep.	May go to sleep.
Preemption	Cannot be preempted/scheduled.	Cannot be preempted/scheduled.	May be preempted/scheduled.
Ease of use	Not easy to use.	Easy to use.	Easy to use.
When to use	If deferred work will not go to sleep and if you have crucial scalability or speed requirements.	If deferred work will not go to sleep.	If deferred work may go to sleep.

The softirqs handlers are executed with all interrupts enabled. They are executed once all interrupt handlers have completed, before the kernel resumes scheduling processes, The number of softirqs is fixed in the system, so softirqs are not directly used by drivers, but by complete kernel subsystems (network, etc.)

HI_SOFTIRQ and TASKLET_SOFTIRQ are used to execute tasklets, Example usage of softirqs – NAP

Work queues typically be used for background work with can be scheduled.

Tasklet	Priority	Softirq	Description
HI_SOFTIRQ	0		High-priority tasklets
TIMER_SOFTIRQ	1		Timer bottom half
NET_TX_SOFTIRQ	2		Send network packets
NET_RX_SOFTIRQ	3		Receive network packets
SCSI_SOFTIRQ	4		SCSI bottom half
TASKLET_SOFTIRQ	5		Tasklets

Linux memory

swap: The primary function of swap space is to substitute disk space for RAM memory when real RAM fills up and more space is needed.

Using **ramfs or tmpfs** you can allocate part of the physical memory to be used as a partition. You can mount this partition and start writing and reading files like a hard disk partition. Since you'll be reading and writing to Viv the RAM, it will be faster.

Non-Uniform Memory Access is a computer memory design used in multiprocessing, where the memory access time depends on the memory location relative to the processor.

Allocating kernel memory (buddy system and slab system)

The main drawback in buddy system is internal fragmentation as larger block of memory is acquired then required.

cat/proc/buddyinfo displays as follows:

Node 0, Zone DMA 0 4 5 4 4 3 ...

Node 0, Zone Normal 1 0 0 1 101 8 ...

Node 0, Zone highmem 2 0 0 1 1 0 ...

static memory allocation linux kernel : boottime by driver reserve contagious memory

A second strategy for allocating kernel memory is known as slab allocation. It eliminates fragmentation caused by allocations and deallocations. A slab is made up of one or more physically contiguous pages. cat /proc/slabinfo

`_get_free_pages(gfp_t gfp_mask, unsigned int order)` get the original pages directly from the buddy system.

kmalloc(size_t size, gfp_t flags)

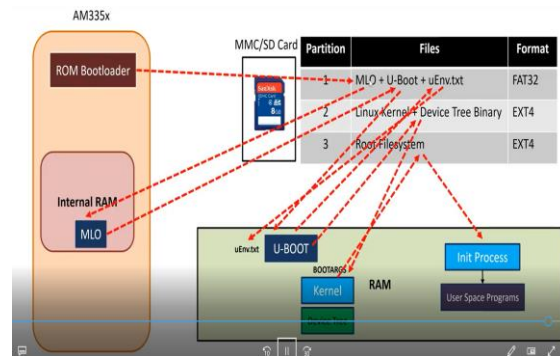
Memory allocation function which allocates size_t bytes according to the flags provided. Usually used for memory allocation sizes up to a PAGE size of memory allocates contiguous region from the physical memory. But keep in mind, allocating and free'ing memory is a lot of work.

kmem_cache_create(his is part of the kernel slab cache interface. This is used to create memory areas of size_t size which can be reused after the memory is used. Instead of the memory being freed after the memory is used, it is returned back to the cache for re-use. All the memory to be used should be of the same size.

	<p>void * vmalloc(unsigned long size) allocate virtually contiguous memory</p>
Linux Scheduling Policy	<p>Linux supports real-time scheduling out of the box. There is a misconception that Linux has to be patched to provide support for real-time scheduling. The only issue is that the scheduling latencies may not satisfy the hard real-time requirements of critical applications. There are patches that try to address this, like the CONFIG_PREEMPT_RT patch</p> <p>We have two categories of scheduling policies. Normal and real-time. Real-time scheduling policy has two sub-types, round-robin and first-in first-out, identified by SCHED_RR and SCHED_FIFO. The sched_setscheduler() system call set scheduling policy of thread to real Time SCHED_FIFO, SCHED_RR .</p> <p>SCHED_FIFO and SCHED_RR priorities allow priorities from 1 to 99. SCHED_OTHER, which is the default supports only the value of 0. In case of SCHED_FIFO, for tasks of the same priority, the currently running task has to yield before the next one can run . Tasks of the same priority when running with RR_SCHEDULING will get an equal interval run.</p> <p>Default Priority is 20 with nice value 0</p> <p>Linux Kernel implements two separate priority ranges –</p> <ul style="list-style-type: none"> • The nice value range is -20 to +19 where -20 is highest, 0 is default/ Nice value: minus 20 to plus 19; larger (+19) nice correspond to lower priority. • Real-time priority: 0 to 99; higher real-time priority values correspond to a greater priority. PR = 20 plus nice , 0 is default nice priority of process

	<div><pre>top - 19:11:17 up 21:45, 5 users, load average: 0.23, 0.25, 0.31 Tasks: 182 total, 1 running, 180 sleeping, 0 stopped, 1 zombie Cpu(s): 6.9%us, 1.8%sy, 0.0%ni, 91.3%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st Mem: 2060736k total, 1918216k used, 142520k free, 496712k buffers Swap: 473876k total, 11748k used, 462128k free, 683312k cached</pre><table><thead><tr><th>PID</th><th>USER</th><th>PR</th><th>NI</th><th>VIRT</th><th>RES</th><th>SHR</th><th>S</th><th>%CPU</th><th>%MEM</th><th>TIME+</th><th>COMMAND</th></tr></thead><tbody><tr><td>594</td><td>mark</td><td>20</td><td>0</td><td>484m</td><td>260m</td><td>30m</td><td>S</td><td>16</td><td>13.0</td><td>7:35.74</td><td>firefox</td></tr><tr><td>26885</td><td>mark</td><td>9</td><td>-11</td><td>157m</td><td>5684</td><td>4288</td><td>S</td><td>1</td><td>0.3</td><td>0:17.96</td><td>pulseaudio</td></tr><tr><td>26733</td><td>root</td><td>20</td><td>0</td><td>500m</td><td>47m</td><td>14m</td><td>S</td><td>1</td><td>2.3</td><td>1:26.56</td><td>Xorg</td></tr><tr><td>19739</td><td>mark</td><td>20</td><td>0</td><td>46196</td><td>16m</td><td>10m</td><td>S</td><td>0</td><td>0.8</td><td>0:03.44</td><td>gnome-terminal</td></tr><tr><td>26976</td><td>mark</td><td>20</td><td>0</td><td>67260</td><td>43m</td><td>14m</td><td>S</td><td>0</td><td>2.1</td><td>0:15.94</td><td>compiz.real</td></tr><tr><td>27048</td><td>mark</td><td>20</td><td>0</td><td>21880</td><td>8540</td><td>6996</td><td>S</td><td>0</td><td>0.4</td><td>0:17.81</td><td>multiloader-apple</td></tr><tr><td>1</td><td>root</td><td>20</td><td>0</td><td>3084</td><td>596</td><td>496</td><td>S</td><td>0</td><td>0.0</td><td>0:01.36</td><td>init</td></tr><tr><td>2</td><td>root</td><td>15</td><td>-5</td><td>0</td><td>0</td><td>0</td><td>S</td><td>0</td><td>0.0</td><td>0:00.00</td><td>kthreadd</td></tr></tbody></table></div>	PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND	594	mark	20	0	484m	260m	30m	S	16	13.0	7:35.74	firefox	26885	mark	9	-11	157m	5684	4288	S	1	0.3	0:17.96	pulseaudio	26733	root	20	0	500m	47m	14m	S	1	2.3	1:26.56	Xorg	19739	mark	20	0	46196	16m	10m	S	0	0.8	0:03.44	gnome-terminal	26976	mark	20	0	67260	43m	14m	S	0	2.1	0:15.94	compiz.real	27048	mark	20	0	21880	8540	6996	S	0	0.4	0:17.81	multiloader-apple	1	root	20	0	3084	596	496	S	0	0.0	0:01.36	init	2	root	15	-5	0	0	0	S	0	0.0	0:00.00	kthreadd
PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND																																																																																																		
594	mark	20	0	484m	260m	30m	S	16	13.0	7:35.74	firefox																																																																																																		
26885	mark	9	-11	157m	5684	4288	S	1	0.3	0:17.96	pulseaudio																																																																																																		
26733	root	20	0	500m	47m	14m	S	1	2.3	1:26.56	Xorg																																																																																																		
19739	mark	20	0	46196	16m	10m	S	0	0.8	0:03.44	gnome-terminal																																																																																																		
26976	mark	20	0	67260	43m	14m	S	0	2.1	0:15.94	compiz.real																																																																																																		
27048	mark	20	0	21880	8540	6996	S	0	0.4	0:17.81	multiloader-apple																																																																																																		
1	root	20	0	3084	596	496	S	0	0.0	0:01.36	init																																																																																																		
2	root	15	-5	0	0	0	S	0	0.0	0:00.00	kthreadd																																																																																																		
Packet journey	<p>The high-level path a packet takes from arrival to socket receive buffer is as follows:</p> <p>Driver is loaded and initialized.</p> <p>Receive Side</p> <ol style="list-style-type: none">1. Packet arrives at the NIC from the network.2. Packet is copied (via DMA) to a ring buffer in kernel memory.3. Hardware interrupt is generated to let the system know a packet is in memory.4. Data that was DMA'd into memory is passed up the networking layer as an 'skb' for more processing.5. Tapping in eth layer (tcp dump), net filter in ip layer, state machine in tcp layer later queue to socket and copy to application and invoke poll.6. Tcp maintain state machine, fragmentation and assembly7. Skb and packet memory is free after socket read and copy to user memory <p>send side</p> <ol style="list-style-type: none">1. skb is allocated in TCP layer and enqueue in write queue, empty space in beginning of skb. Mac or neighbor discover if not cached, ipl layer net filter on send side2. Add Checksum before sending it. Nic does frame checksum on both recv and send side ,nic checksum offload3. Modern nic has more hardware queue (faster) , in recv side packet belong to same Stream Goes to same queue useful in SMP <p>NAPI = New API, Principe: when the network traffic exceeds a given threshold ("budget"), Disable network interrupts and consume incoming packets through a polling function, instead of processing each new packet with an interrupt.</p> <p>incoming network data frames are distributed among multiple CPUs if packet steering is enabled or if the NIC has multiple receive queues.</p>																																																																																																												
UBOOT	<p>U-Boot is both a first stage and second-stage bootloader. If there are size constraints, U-Boot may be split into stages: U-Boot performs both first-stage (e.g., configuring memory controllers and SDRAM) and second-stage booting. , U-Boot Linux booting requires its boot commands to explicitly specify the physical memory</p>																																																																																																												

addresses as destinations for copying data (kernel, ramdisk, device tree, etc.) and for jumping to the kernel and as arguments for the kernel



The way bootloader works is that after doing some setup, it simply jumps onto Linux entry point. In the old versions, it had a function called TheKernel. I don't know how it is called nowadays but the idea is the same.

```
void TheKernel(char *cmdline, void* dtb);
```

The kernel is passed the command line, and a pointer to the device tree binary, and then the function gets called, simple as that.

From user point of view, these are the steps for booting:

- 1- set the variable \$cmdline to the desired kernel command line
- 2- use fatload or similar command to read the kernel from the sdcard and put it to some address at the memory, let's say at the address 20000000.
- 3- use fatload again to read the device tree binary (dtb) to another memory address, like 21000000.
- 4- use the bootm (boot from memory) command to start the boot
bootm 20000000 21000000.

In user space programs, main() is the entry point to the program that is called by the libc initialization code when the binary is executed. Kernel code does not have the luxury to rely on libc, as libc itself relies on the kernel syscall interface for memory allocation, I/O, process managements etc.

That said, the equivalent of main() in kernel code is `start_kernel()`, which is called by the bootloader

he asm keyword allows you to embed assembler instructions within C code.

Linux booting facts

Hot plug is the addition of a component to a running computer system without significant interruption to the operation of the system. Hot plugging a device does not require a restart of the system.

**Synchronization
mechanisms inside
Linux kernel**

Critical Region: A critical section is a piece of code which should be executed under mutual exclusion. Suppose that two threads are updating the same variable which is in parent process's address space

Atomic operation: This is the very simple approach to avoid race condition or deadlock. Atomic operators are operations, like add and subtract, which perform in one clock cycle (uninterruptible operation). and another that operates on individual bits. All atomic functions are inline functions.

Semaphore: This is another kind of synchronization mechanism which will be provided by the Linux kernel. When some process is trying to access semaphore, which is not available, semaphore puts process on wait queue (FIFO) and puts task on sleep. That's why semaphore is known as a sleeping lock. After this processor is free to jump to another task which is not requiring this semaphore. As soon as it is invoked. There two flavors of semaphore is present. Basic semaphore **Reader-Writer Semaphore**

Semaphore

Semaphore puts a task on sleep. So, the semaphore can be only used in process context. Interrupt context cannot sleep. Operation to put task on sleep is time consuming(overhead) semaphore is suitable for lock which is holding for long term. A code holding a semaphore can be preempted. It does not disable kernel preemption. After disabling interrupts from some tasks, semaphore should not acquire. Because task would sleep if it failed to acquire the semaphore, at this time the interrupt has been disabled and current task cannot be scheduled out.

Semaphore wait list is FIFO in nature. So, the task which tried to acquire semaphore first will be waken up from wait list first.. Semaphore can be acquired/release from any process/thread.

Spin-lock: This is special type of synchronization mechanism which is preferable to use in multi-processor (SMP) system. Basically, its a busy-wait locking mechanism until the lock is available. In case of unavailability of lock, it keeps thread in light loop and keep checking the availability of lock. Spin-lock is not recommended to use in single processor system. If some process_1 has acquired a lock and other process_2 is trying to acquire lock, in this case process 2 will spins around and keep processor core busy until it acquires lock. process_2 will create a deadlock, it doesn't allow any other process to execute because CPU core is in loop by semaphore.

Couple of observations about nature of spinlocks:

1. Spinlocks are very much suitable to use in interrupt(atomic) context because it doesn't put process/thread in sleep.

2. In the uni processor environment, if the kernel acquires a spin lock, it would **disable preemption first** ; if the kernel releases the spin lock, it would enable preemption. This is to avoid dead lock on uni processor system
3. Spin-locks is not recursive. A thread may call lock on a recursive mutex repeatedly. Ownership will only be released after the thread makes a matching number of calls to unlock
1. Special care must be taken in case where spinlock is shared b/w interrupt handler and thread. Local interrupts must be disabled on the same CPU(core) before acquiring spin-lock. In the case where interrupt occurs on a different processor, and it spins on the same lock, does not cause deadlock because the processor who acquire lock will be able to release the lock using the other core.
5. When data is shared between two tasklet, there is not need to disable interrupts because tasklet dose not allow another running tasklet on the same processor.

There two flavors of spin-lock is present.

Basic spin-lock

Reader-Writter Spin-lock

With increasing the level of concurrency in Linux kernel read-write variant of spin-lock is introduces. This lock is used in the scenario where many readers and few writers are present. Any reader will not get lock until writer finishes it.

If the kernel is running on a uniprocessor and **CONFIG_SMP, CONFIG_PREEMPT** aren't enabled while compiling the kernel then spinlock will not be available. Because there is no reason to have a lock when no one else can run at the same time.

Locking between User context

But if you have disabled CONFIG_SMP and enabled CONFIG_PREEMPT then spinlock will simply disable preemption, which is sufficient to prevent any races

spin_trylock(

spin_is_locked(

Locking between User context and Bottom Halves)

spin_lock_bh(spinlock_t *lock)

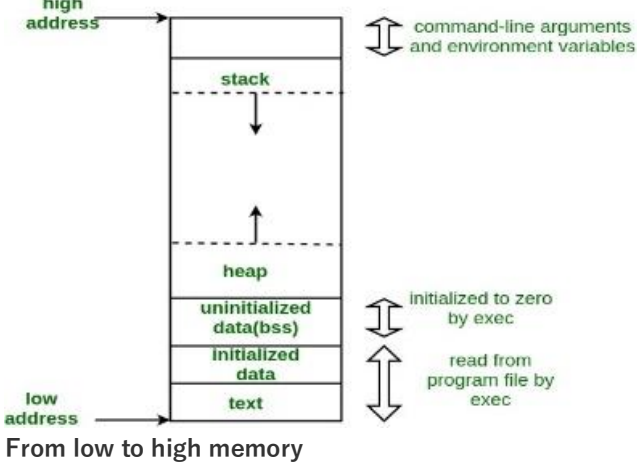
t disables soft interrupts on that CPU, then grabs the lock. This has the effect of preventing softirqs, tasklets, and bottom halves from running on the local CPU.

Locking between Hard IRQ and Bottom Halves)

spin_lock_irq(spinlock_t *lock):If you share data between Hardware ISR and Bottom halves then you have to disable the IRQ before lock. Because the bottom halves

	<p>processing can be inter</p> <p>If process context code and a bottom half share data, you need to disable bottom-half processing and obtain a lock before accessing the data. Doing both ensures local and SMP protection and prevents a deadlock.</p> <p>Sequence Lock: This is very useful synchronization mechanism to provide a lightweight and scalable lock for the scenario where many readers and a few writers are present. Sequence lock maintains a counter for sequence. When the shared data is written, a lock is obtained and a sequence counter is incremented by 1. Write operation makes the sequence counter value to odd and releasing it makes even.</p> <ol style="list-style-type: none"> 2. atomic variables and cpu local variables 3. spin locks (and reader/writer locks) 4. sequential locks 5. RCU locks 6. mutex 7. semaphores 8. completions 9. wait queues 10. memory barriers
Kmem_cache_alloc	<p>Kmalloc - allocates contiguous region from the physical memory. But keep in mind, allocating and free'ing memory is a lot of work.</p> <p>Kmem_cache_alloc - Here, your process keeps some copies of the some pre-defined size objects pre-allocated. Say you have struct that you know you will be requiring very frequently, so instead of allocating it from the main memory (kmalloc) when you need it, you already keep multiple copies of it allocated & when you want it, it returns the address of the block already allocated (saves a lot of time). Similarly, when you free it, you don't give it back, it actually isn't free'd, it goes back to the allocated pool so that if some process again asks for it, you can return this address of the already allocated struct.</p>
barrier	<p>A memory barrier, also known as a membar, memory fence or fence instruction, is a type of barrier instruction that causes a central processing unit (CPU) or compiler to enforce an ordering constraint on memory operations issued before and after the barrier instruction. This typically means that operations issued prior to the barrier are guaranteed to be performed before operations issued after the barrier. he following two-process</p> <p>Initially, memory locations x and f both hold the value 0. The program running on processor #1 loops steps of the program correspond to individual processor instructions.</p>

	<p>Processor #1:</p> <pre>while (f == 0); // Memory fence required here print x;</pre> <p>Processor #2:</p> <pre>x = 42; // Memory fence required here f = 1;</pre>
fork vs/ vfork	<p>During the fork() system call the Kernel makes a copy of the parent process's address space and attaches it to the child process.</p> <p>But the vfork() system call do not makes any copy of the parent's address space, so it is faster than the fork() system call. The child process as a result of the vfork() system call executes exec() system call.</p> <p>The child process from vfork() system call executes in the parent's address space (this can overwrite the parent's data and stack) which suspends the parent process until the child process exits.</p>

<p>Memory Layout of C Programs</p>	 <p>From low to high memory</p> <ol style="list-style-type: none"> 1 Text Segment: contains executable instructions. 2. Data Segment: contains global & static variables 3. heap goes up 4. stack goes down
<p>Mutex V/s Semaphore</p>	<p>1> In mutexes has an ownership property, only the thread that took the lock has the key. Only that thread alone can release the lock. Binary semaphores doesn't have an ownership property, as any thread can take the key to open the lock.</p> <p>2)A semaphore is a generalized mutex. In lieu of single buffer, we can split the 4 KB buffer into four 1 KB buffers (identical resources). A semaphore can be associated with these four buffers. The consumer and producer can work on different buffers at the same time.</p>
<p>Dockers</p>	<p>A software container is a way to bundle and isolate processes (software) running on a server. Virtual machines and containers differ in several ways, but the primary difference is that containers provide a way to virtualize an OS so that multiple workloads can run on a single OS instance. With VMs, the hardware is being virtualized to run multiple OS instances.</p> <p>Docker is an open source project that makes it easier to create, deploy and run Applications in containers .The applications are packaged in a docker container which contains all the dependencies (libraries, packages) that are needed to deploy the application. By using docker, an application can be easily moved around from the developer's laptop, into the testing environment and finally into production.</p> <p>Podman, new in Red Hat Enterprise Linux 7.6 Beta, can replace the docker CLI, allowing you to run standalone (non-orchestrated) containers ...</p>

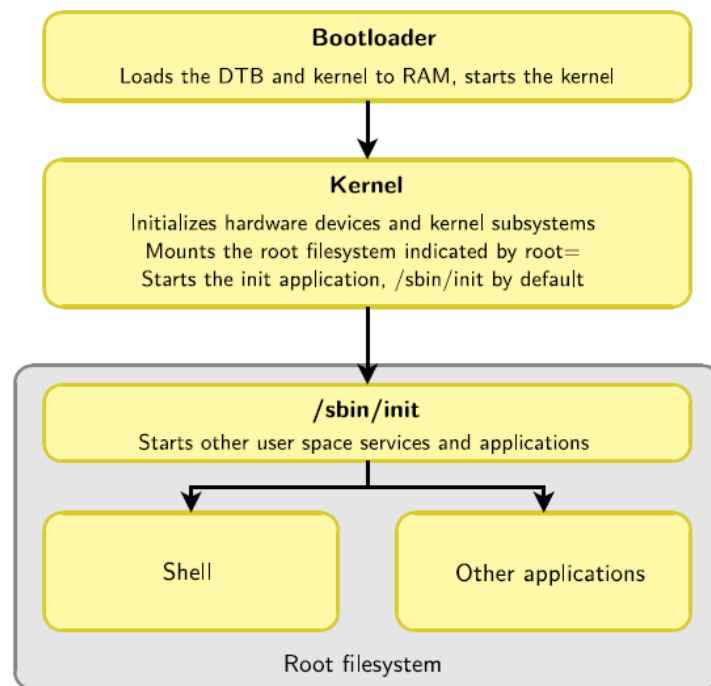
Docker/ Kubernetes	<p>Docker is what enables us to run, create and manage containers on a single host</p> <p>Kubernetes can then allow you to automate container provisioning, networking, load-balancing, security and scaling across all these nodes from a single command line or dashboard.</p> <p>A collection of nodes that is managed by a single Kubernetes instance is referred to as a Kubernetes cluster.</p> <p>Now, why would you need to have multiple nodes in the first place? The two main motivations behind it are:</p> <ol style="list-style-type: none">1. To make the infrastructure more robust: Your application will be online, even if2. some of the nodes go offline, i.e, High availability.3. To make your application more scalable: If workload increases, simply spawn more4. containers and/or add more nodes to your Kubernetes cluster. <p>Kubernetes automates the process of scaling, managing, updating and removing containers. In other words, it is a container orchestration platform. While Docker is at the heart of the containerization, it enables us to have containers in the first place. Differences Between Kubernetes and Docker In principle, Kubernetes can work with any containerization technology.</p> <p>Kubernetes pods :A Kubernetes pod is a group of containers that are deployed together on the same host. If you frequently deploy single containers, you can generally replace the word "pod" with "container" and accurately understand the concept.</p> <p>Docker is a run time engine running on your computer. It's a daemon that is in charge of containers start, stop on that single computer. So Docker is about managing works within a single machine.</p> <p>Kubernetes is kind of a cluster management software. It is a group of daemons that is in charge of a cluster of machines. Though there is a single daemon (kubelet) running on an individual machine, the kubelet by itself does not have much value on the table; it is these group of kubelets (along with kubernetes controllers that control them) make decisions about the whole cluster. So k8s is about managing works for a cluster of machines.</p>

ASLR	<p>Address space layout randomization (ASLR) is a memory-protection process for operating systems (OSes) that guards against buffer-overflow attacks by randomizing the location where system executables are loaded into memory.</p> <p>The success of many cyberattacks, particularly zero-day exploits, relies on the hacker's ability to know or guess the position of processes and functions in memory. ASLR is able to put address space targets in unpredictable locations. If an attacker attempts to exploit an incorrect address space location, the target application will crash, stopping the attack and alerting the system.</p> <p>ASLR works alongside virtual memory management to randomize the locations of different parts of the program in memory. Every time the program is run, components (including the stack, heap, and libraries) are moved to a different address in virtual</p>
Object Size Checking (OSC)	Object Size Checking (OSC) leverages a builtin compiler technique to determine buffer overflows in C/C++ code. various optimization passes enabled with -O2
xspace	<p>Making the stack (and heap) non-executable provides a high degree of protection against many types of buffer overflow attacks for existing programs.</p> <p>is that execution occurs in the code section, which is neither stack nor heap.</p>
Reader-writer lock	When a writer is writing the data, all other writers or readers will be blocked until the writer is finished writing. Readers-writer locks are usually constructed on top of mutexes and condition variables, or on top of semaphores.
STACK	<p>Key Differences Between Stack and Heap Allocations</p> <ol style="list-style-type: none"> 1. In a stack, the allocation and deallocation is automatically done by whereas, in heap, it needs to be done by the programmer manually. 2. Memory shortage problem is more likely to happen in stack whereas the main issue in heap memory is fragmentation. 3. Stack is not flexible, the memory size allotted cannot be change
OS concept	<p>Page Fault: A page is a fixed-length block of memory that is used as a unit of transfer between physical memory and external storage like a disk, and a page fault is an interrupt (or exception) to the software raised by the hardware when a program accesses a page that is mapped in address space, but not loaded in physical memory.</p>
DMA	<p>Direct Memory is a feature which provides direct access (read/write) to system memory without interaction from the CPU. using "DMA Controller"</p> <p>DMA controller is a device which directly drives the data and address bus during data transfer. So, it is purely Physical address. (It never needs to go through MMU & Virtual addresses).</p>

STACK protection	<p>in a multi-threaded environment, there can be multiple stacks in a process.</p> <p>One threat to the stack is malicious program input, which can overflow a buffer and overwrite stack pointers, simple method GCC, you use -fstack-protector-all.</p>
vmalloc	<p>vmalloc allocates virtually contiguous memory space (not necessarily physically contiguous), while kmalloc allocates physically contiguous memory (also virtually contiguous). Most of the memory allocations in Linux kernel are done using kmalloc, due to the following reasons:</p> <p>On many architectures, hardware devices don't understand virtual address. Therefore, their device drivers can only allocate memory using kmalloc. • kmalloc has better performance in most cases because physically contiguous memory region is more efficient than virtually contiguous memory. interval of time.</p>
Kernel mode	<p>Kernel mode</p> <p>-----</p> <p>Enter using interrupt/Trap</p> <ol style="list-style-type: none"> 1. Access to privileged instructions <ul style="list-style-type: none"> --> CPU control instructions (CLI, STI, HLT, WAIT, LOCK, ...) --> IN, OUT (direct hardware access) 2. Full access to physical memory (RAM) <p>User mode</p> <p>-----</p> <ol style="list-style-type: none"> 1. Restricted instruction set 2. No direct hardware access 3. No access to entire physical memory (RAM) 4. Memory access only by virtual addresses (Virtual memory) 5. Memory access can happen via demand-paging
How system call works	<ol style="list-style-type: none"> 1. Application program makes a system call by invoking wrapper function in C library 2. This wrapper functions makes sure that all the system call arguments are available to trap-handling routine 3. Generally, a stack is used to pass these arguments to wrapper function. But the Kernel looks into specific registers for these arguments. Hence the wrapper function also takes care of copying these arguments to specific registers 4. Each system call has a unique call number which is used by kernel to identify which system call is invoked? The wrapper function again copies the system call number into specific CPU registers 5. Now the wrapper function executes trap instruction (int 0x80). This instruction causes the processor to switch from 'User Mode' to 'Kernel Mode'

6. The code pointed out by location 0x80 is executed (Most modern machines use sysenter rather than 0x80 trap instruction)
7. In response to trap to location 0x80, kernel invokes system_call() routine which is located in assembler file arch/i386/entry.S (also called handler)
8. This **handler saves register values** onto kernel stack and does some validations like verifying system call number etc.
9. A map of system call number as key and the appropriate system call as value exists. This is called system call table. After proper validations, the service routine performs required actions like modify values at addresses specified in arguments or transfer data between user memory and kernel memory. After all these actions, service routine returns status of execution to the system_call routine
10. Now the handler restores register values from kernel stack and places the system call return value on the stack
11. Thus handler is returned to wrapper function, simultaneously returning processor to user mode
12. Just in case if the return value of system call service routine indicated an error, then wrapper function sets 'errno' a global variable and then returns to caller providing integer return value that indicates the status of execution

Linux booting



Stage 1

When a system is booted, Processor executed a code from a well-known location known as BIOS (Basic Input Output System) which is stored in flash memory of motherboard. Its Job is to find the boot device (floppy/hard disk, cd). When boot device is detected, it passes control to first stage bootloader.

A master boot record (often shortened as MBR) is a kind of boot sector stored on a hard disk drive or other storage device that contains the necessary computer code to start the boot process.

Stage 2

The first-stage loader (stage1) is loaded into the RAM and executed by the BIOS from the Master boot record (MBR). This Boot Loader is 512 bytes in size with 64 bytes partition table). Its job is to find the SECOND order Boot Loader (grub) and load it into RAM and passed control to 2nd stage bootloader.

Stage 3

Grub1 is embedded in an MBR (size issue).

Grub2 is knowledge about the Linux file system (ext2,ext3)

Grub2 copy the Linux kernel image into the RAM using /boot/grub/grub.conf..

Step 4 Kernel stage

Kernel is in compressed **cpio format** file present in /boot directory .

Mounts the root file system as specified in the "root=" in grub.conf

```
grub> root (hd0,0)
grub> kernel /vmlinuz-i686-up-4GB root=/dev/hda9
grub> boot
```

kernel /vmlinuz-i686-up-4GB root=/dev/hda9 - Specifies the kernel location which is inside the /boot folder. This location is related to the root(hd0,0) statement. The root partition is specified according to the Linux naming convention (/dev/hda9/)

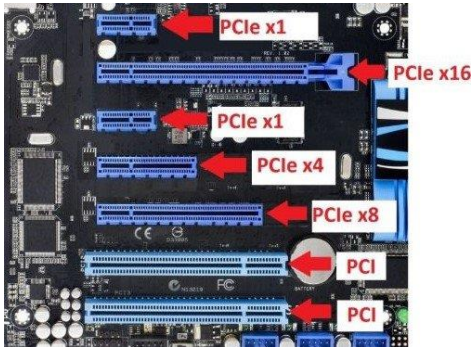
initrd/ Initramfs is used by kernel as temporary root file system until kernel is booted and thereal root file system is mounted. It also contains necessary drivers compiled inside, which helps it to access the hard drive partitions, and other hardware. insmod for loading kernel modules, and lvm (logical volume manager tools).

Initramfs/initrd is an image file in /boot containing the basic root file system with all Kernel modules. The kernel then Mount this image file as a starting memory-based root file system. The kernel then starts to detect the system's hardware. The root file system on disk takes over from the memory. The boot process then starts INIT (SYSTEMD)

Step 5 INIT

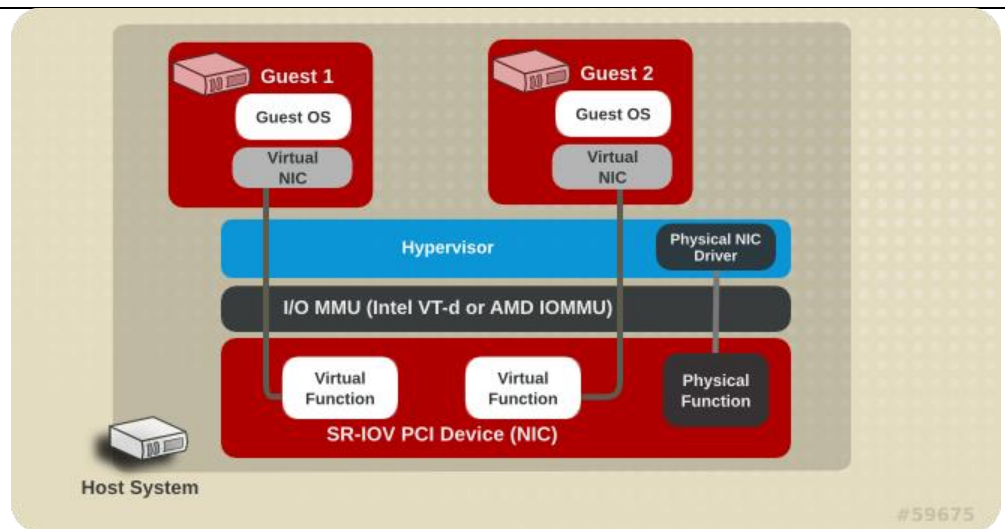
	<p>The kernel, once it is loaded in step 4, it finds init in/sbin (/sbin/init) and executes it. The first thing init does is reading the initialization file, /etc/inittab. The program init is the process with process ID 1.</p>
Hardware Security	<p>Secure Boot is a security standard developed by members of the PC industry to help make sure that your PC boots using only software that is trusted by the PC manufacturer. When the PC starts, the Bios checks the signature of each piece of boot software, including drivers and the operating system. If the signatures are good, the PC boots, and the Bios gives control to the operating system or else it would halt the boot up process and thrown error.</p> <p>A Trusted Platform Module (TPM) is a hardware chip on the computer's motherboard that stores cryptographic keys used for encryption. Once enabled, the Trusted Platform Module provides full disk encryption capabilities. It becomes the "root of trust" for the system to provide integrity and authentication to the boot process. It keeps hard drives locked/sealed until the system completes a system verification, or authentication check. The TPM includes a unique RSA key burned into it, which is used for asymmetric encryption. Additionally, it can generate, store, and protect other keys used in the encryption and decryption process. A hardware security module (HSM) are external devices connected to a network using TCP/IP. encryption capabilities by storing and using RSA keys.</p>
Extended Berkeley Packet Filter (eBPF)	<p>eBPF is a feature available in Linux kernels that allows you to run a virtual machine inside the kernel. This virtual machine allows you to safely load programs into the kernel to customize its operation.</p> <p>eBPF is a revolutionary technology with origins in the Linux kernel that can run sandboxed programs in an operating system kernel. It is used to safely and efficiently extend the capabilities of the kernel without requiring changing kernel source code or load kernel modules.</p> <pre>graph TD subgraph USER_SPACE [USER SPACE] tcpdump[tcpdump] libpcap[libpcap] BPF_byte_code[BPF byte-code] end subgraph KERNEL [KERNEL] verifier[verifier] BPF_VM[BPF VM] byte_code_instructions[byte-code instructions] end tcpdump -- "1 Defined expressions, such as ip or tcp and port 80" --> libpcap libpcap -- "2 compile / parse" --> BPF_byte_code BPF_byte_code -- "3 attached to tap interface" --> verifier verifier -- "4 checks program ends and it is safe to execute" --> BPF_VM BPF_VM -- "5 inject in kernel simple freeBSD and linux instructions" --> byte_code_instructions byte_code_instructions -- "6 every packet (event based) is interpreted" --> eth0[eth0]</pre> <p>First BPF use case: tcpdump</p>

	<p>Traffic control</p> <p>There are several use cases for eBPF, including traffic control, creating network policy, connect-time load balancing and observability. packets bypass complex routing and simply arrive at their final destination</p> <p>Creating Network Policy</p> <p>When creating network policy, there are two instances where eBPF can be used:</p> <ol style="list-style-type: none">1. eXpress Data Path (XDP) – As a raw packet buffer enters the system, eBPF gives you an e2. Network policy – eBPF allows you to efficiently examine a packet and apply network policy, both for pods and hosts. <p>Connect-time Load Balancing</p> <p>With eBPF, you can avoid this packet translation by using an eBPF program that you've loaded into processing pat</p>
DPDK	<p>DPDK (Data Plane Development Kit) is a framework (under the Linux Foundation) comprised of various userspace libraries and drivers for fast packet processing. Originally developed by Intel to run on x86 based CPUs, DPDK now supports other CPU types. DPDK leverages existing Intel Processor technologies like SIMD instructions (Singles Instruction Multiple Data), Huge-pages memory, multiple Memory channels and Caching to provide acceleration with its own libraries.</p> <p>Though DPDK uses a number of techniques to optimise packet throughput, how it works (and the key to its performance) is based upon Fast-Path and PMD.</p> <p>Fast-Path (Kernel bypass) - A fast-path is created from the NIC to the application within user space, in turn, bypassing the kernel. This eliminates context switching when moving the frame between user space/kernel space. Additionally, further gains are also obtained by negating the kernel stack/network driver, and the performance penalties they introduce.</p> <p>Poll Mode Driver - Instead of the NIC raising an interrupt to the CPU when a frame is received, the CPU runs a poll mode driver (PMD) to constantly poll the NIC for new packets. However, this does mean that a CPU core must be dedicated and assigned to running PMD.</p>

<p>KSM</p>	<p>KSM (kernel samepage merging) is a Linux kernel feature that allows share identical memory pages among different process or virtual machines on the same server.</p> <p>User kernel tracing with ftrace to get back trace https://blog.selectel.com/kernel-tracing-ftrace/</p>
<p>PCI/PCIE</p>	<p>Peripheral Component Interconnect (PCI) slots are such an integral part of a computer's architecture that most people take them for granted. For years, PCI has been a versatile, functional way to connect sound, video and network cards to a motherboard.</p> <p>The Peripheral Component Interconnect Express (PCI Express or PCIe) is a high-speed interface standard for connecting additional graphics cards (GPUs), Local Area Network (LAN) ports. This is accomplished using expansion cards, also known as add-on cards.</p> <p>Simply put, the PCI Express interface allows for the expansion of a motherboard beyond its default GPU, network and storage configurations.</p>  <p>But PCI has some shortcomings. As processors, video cards, sound cards and networks have gotten faster and more powerful, PCI has stayed the same. It has a fixed width of 32 bits and can handle only 5 devices at a time. The newer, 64-bit PCI-X bus provides more bandwidth, but its greater width compounds some of PCI's other issues. Some of the most common serial protocols include SPI, I2C, CAN, and USB</p> <p>The example of parallel communications are computer to printer and communication between internal components in embedded system</p> <p>PCI Express is a serial connection that operates more like a network than a bus. Instead of one bus that handles data from multiple sources, PCIe has a switch that controls several point-to-point serial connections. (See How LAN Switches Work for details.) These connections fan out from the switch, leading directly to the devices where the data needs to go. Every device has its own dedicated connection, so devices no longer share bandwidth like the</p>

PCI Express: Unidirectional Bandwidth in x1 and x16 Configurations				
Generation	Year of Release	Data Transfer Rate	Bandwidth x1	Bandwidth x16
PCIe 1.0	2003	2.5 GT/s	250 MB/s	4.0 GB/s
PCIe 2.0	2007	5.0 GT/s	500 MB/s	8.0 GB/s
PCIe 3.0	2010	8.0 GT/s	1 GB/s	16 GB/s
PCIe 4.0	2017	16 GT/s	2 GB/s	32 GB/s
PCIe 5.0	2019	32 GT/s	4 GB/s	64 GB/s
PCIe 6.0	2021	64 GT/s	8 GB/s	128 GB/s

Single Root I/O Virtualization (SR-IOV)



The single root I/O virtualization (SR-IOV) interface the SR-IOV allows different virtual machines (VMs) in a virtual environment to share a single PCI Express hardware interface.

In contrast, MR-IOV allows I/O PCI Express to share resources among different VMs on different physical machines.

The Peripheral Component Interconnect (PCI) passthrough feature enables you to access and manage hardware devices from a virtual machine. When PCI passthrough is configured, the PCI devices function as if they were physically

attached to the guest operating system.

The Intel VT-d extensions provides hardware support for directly assigning a physical devices to guest. The main benefit of the feature is to improve the performance as native for device access.

The VT-d extensions are required for PCI passthrough with Red Hat Enterprise Linux. The extensions must be enabled in the BIOS. Some system manufacturers disable these extensions by default. The AMD IOMMU extensions are required for PCI passthrough with Red Hat Enterprise Linux.

Developed by the **PCI-SIG (PCI Special Interest Group)**, the Single Root I/O Virtualization (SR-IOV) specification is a standard for a type of PCI device assignment that can share a single device to multiple virtual machines. SR-IOV improves device performance for virtual machines.

SR-IOV uses two PCI functions:

Physical Functions (PFs) are full PCIe devices that include the SR-IOV capabilities. Physical Functions configure and manage the SR-IOV functionality by assigning Virtual Functions.

Virtual Functions (VFs) are simple PCIe functions that only process I/O. Each Virtual Function is derived from a Physical Function. The number of Virtual Functions a device may have is limited by the device hardware. A single Ethernet port, the Physical Device, may map to many Virtual Functions that can be shared to virtual machines.

The hypervisor can map one or more Virtual Functions to a virtual machine.

The Virtual Function's configuration space is then mapped to the configuration space presented to the guest

Each Virtual Function can only be mapped to a single guest at a time, as Virtual Functions require real hardware resources. A virtual machine can have multiple Virtual Functions. A Virtual Function appears as a network card in the same way as a normal network card would appear to an operating system.

The SR-IOV drivers are implemented in the kernel. The core implementation is contained in the PCI subsystem, but there must also be driver support for both the Physical Function (PF) and Virtual Function (VF) devices. An SR-IOV capable device can allocate VFs from a PF. The such as queues and register sets.

SR-IOV Virtual Functions (VFs) can be assigned to virtual machines by adding

	<p>a device entry in <hostdev> with the virsh edit or virsh attach-device command. However, this can be problematic because unlike a regular network device, an SR-IOV VF network device does not have a permanent unique MAC address, and is assigned a new MAC address each time the host is rebooted.</p> <p>Using the <interface type='hostdev'> interface device requires: an SR-IOV-capable network card, host hardware that supports either the Intel VT-d or the AMD IOMMU extensions, and the PCI address of the VF to be assigned.</p> <p>To attach an SR-IOV network device on an Intel or an AMD system, follow this procedure:</p> <p>1>Enable Intel VT-d or the AMD IOMMU specifications in the BIOS and kernel</p> <p>2> Verify if the PCI device with SR-IOV capabilities is detected. Using lspci</p> <p>3>Start the SR-IOV kernel modules</p> <p>the Intel 82576 network interface card uses the igb driver kernel module.</p> <p>Activate Virtual Functions</p>
<p>Virtualizing I/O and SR-IOV</p>	<p>The I/O performance of virtual machines has long suffered because I/O performance is largely the result of I/O devices' ability to perform DMA--direct memory access--whereby the I/O device can write directly to the compute host's memory without having to interrupt the host CPU. Not having to interrupt the CPU means that I/O operations can bypass the thousands (or tens of thousands) of cycles that the host OS's I/O stack may impose for the operation, and as a result, be performed with very low latency.</p> <p>When virtualization enters the picture, though, DMA isn't so simple because the memory address space within the VM (and, by extension, the address to which the DMA operation should write) is not the same as the underlying host's real memory address space. Thus, while a VM can trigger a DMA operation, the VM's hypervisor needs to intercept that DMA and translate the operation's memory address from the VM's address space to the host's. As a result, DMAs originating within a VM still wind up interrupting the host CPU so that the hypervisor can perform this address translation. The situation gets worse if the I/O operation is coming from a network adapter or HCA, because if multiple VMs are running on the same host, the hypervisor must then also act as a virtual network switch:</p>

	<p>SR-IOV is--a standardized way for a single I/O device to present itself as multiple separate devices. These virtual devices, called virtual PCIe functions (or just virtual functions), are lightweight versions of the true physical PCIe function in that, under the hood, most of the I/O device's functionality shares the same hardware. However, each virtual function has its own</p> <ol style="list-style-type: none"> 1. PCIe route ID - thus, it really appears as a unique PCIe function on the bus 2. Configuration space, base address registers, and memory space 3. Send/receive queues (or work queues), complete with their own interrupts <p>these features allow the virtual functions to be interrupted independently of each other and process their own DMAs:</p> <p>Let it suffice to say that SR-IOV is what allows a 10gig NIC or InfiniBand HCA to present itself as multiple separate I/O devices (virtual functions), and these virtual functions can all interact with VT-d independently. This, in turn, allows all VMs to bypass the hypervisor entirely when performing DMA operations.</p>
<p>soc</p>	<p>SoC stands for system on a chip. This is a chip/integrated circuit that holds many components of a computer—usually the CPU (via a microprocessor or microcontroller), memory, input/output (I/O) ports and secondary storage—on a single substrate, such as silicon. Having all these components on one substrate means SoCs use less power and take up less space than their multi-chip counterparts. SoCs are becoming increasingly popular with the growth of Internet of Things and edge and mobile computing. Take, for example, Intel's September 2018 acquisition of SoC firm Silicon Engineering Group and older acquisitions of Altera and others.</p> <p>One common example of tech that uses an SoC is video game consoles, such as the Nvidia Tegra X1 used in the Nintendo Switch or the AMD Flute and AMD Gonzalo SoCs expected to power the next Xbox and PlayStation, respectively. Raspberry Pi computers, Arduino boards and STEM kits also use SoCs. SoCs are often used in STEM kits because they are easy to use, and, therefore, helpful in teaching STEM. You also find SoCs in smartphones and tablets.</p>

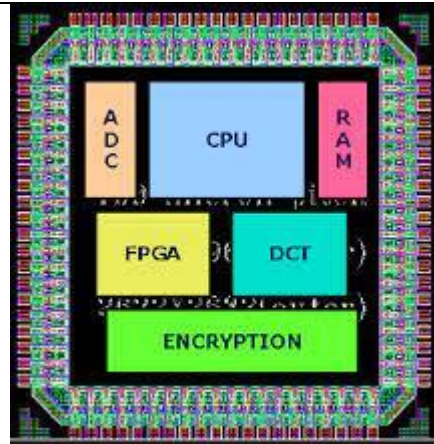


Fig. 2. Typical SoC