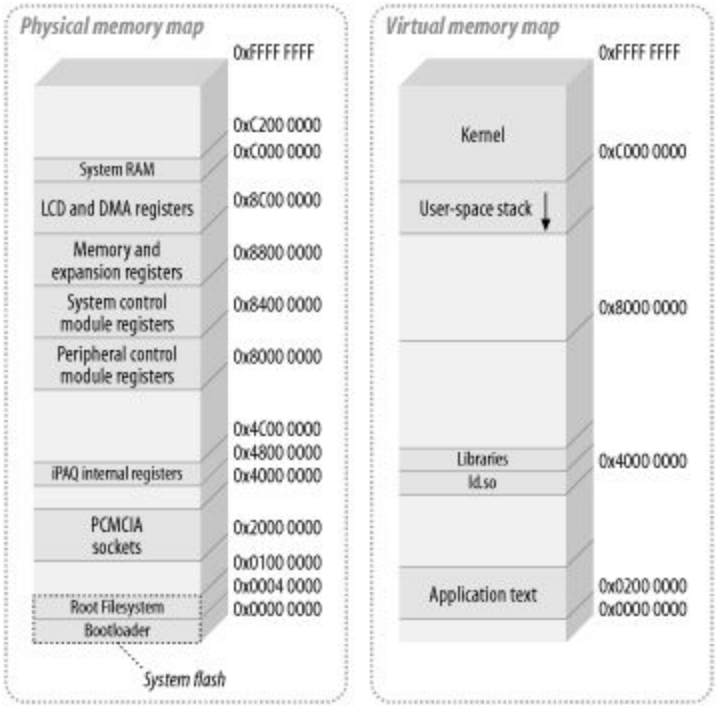
Facebook Production Engineer

Questions

[Linux inside](https://legacy.gitbook.com/book/0xax/linux-insides/details)

名词解释/Definition

Virtual memory



: Avoid fragmentation

: Allow virtualization/isolation between programs (same address space for multiple programs)

: Simplify the design of ABI

: Enable swapping

Stack

: per-thread, high address of VM

: grow from high addr to low addr

: common entry/exit

: push %rbp, mov %rsp %rbp, sub N, %rsp, mov %rbp %rsp, pop %rbp

Heap

: dynamic, share between threads

: low address of VM

: grow from low addr to high addr

Segmentation

: selector, offset

Swap

: Extra memory than physical available(pro)

: Store swapped out memory to disks

: Write to storage, which is slow(con)

: occupying disk(con)

Swap/Replacement algorithm

: (Approximate) LRU: reference bits for each page

: regular interval or each memory access, shift the byte right, placing a 0 in the MSB

: page fault, lowest number page kicked out

: LFU: least frequently used

: Clock/Second-choice: 482, recently used bit, circular queue

: WSClock, NRU, LRU-K, ARC, many more...

Paging

Demand paging

: Copy-on-write, make all pages read-only, raise a fault on write

Page type

: https://www.kernel.org/doc/html/latest/admin-guide/mm/pagemap.html

: locked, slab, buddy, huge, idle (no access yet)

: dirty, uptodate, writeback

: lru, active (in active lru list), unevictable (in unevictable LRU list, pinned), referenced, reclaim, mmap (memory mapped), anon (anonymous memory mapped, not part of file), swapcache (mapped to swap), swapbacked (backed by swap/RAM)

MMU

Segmentation, selector descriptor table, LDT, GDT, TSS

TLB

: hardware handling

: PCID/ASID/PID buildin features

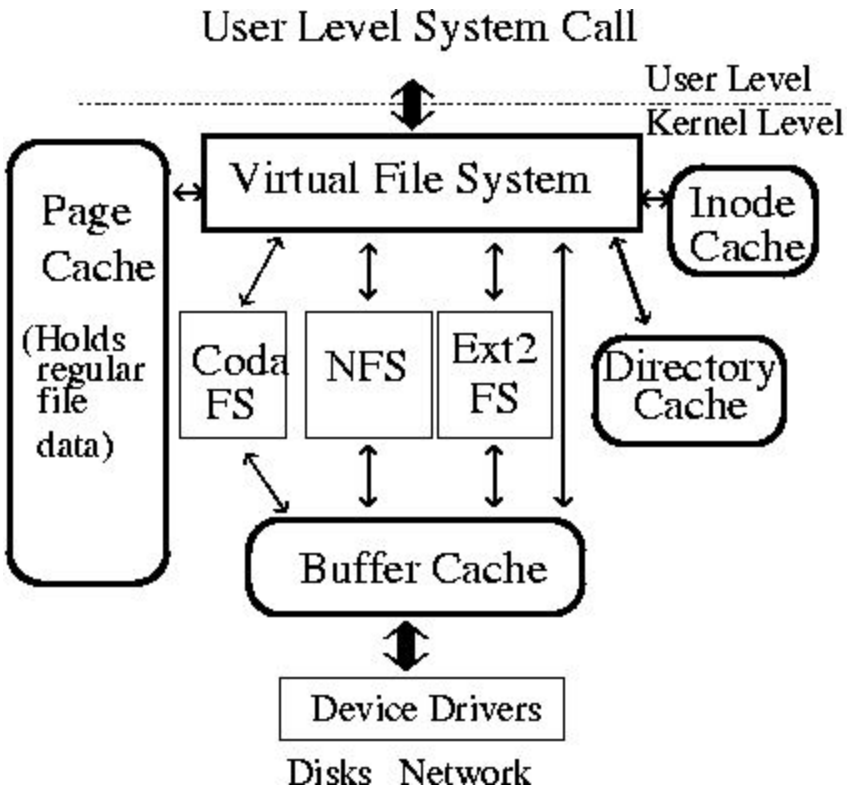
: software handling => OS

Page lookup: vaddr -> TLB -> Page table -> Disk -> paddr

Thrashing: thrashing occurs when a computer's virtual memory resources are overused, leading to a constant state of paging and page faults, inhibiting most application-level processing

Page buffering: cache disk blocks to optimize block I/O

Page cache: file-backed page + tmpfs, cache pages of files to optimize file I/O



Page reclaim

: kswapd controlled by watermark (low-kswapd, min-block this proces, high-stop)

: /proc/sys/vm/swappiness

: /proc/sys/vm/watermark_scale_factor

: /proc/sys/vm/vfs_cache_pressure

: drop_cache

Page table

4-level, COW, vm_struct

Page fault

L1/2/3 Cache

: Intel CPU: SRAM, L1 per core, L2 per process, L3 shared

Interrupt

: Interrupt Vector/Descriptor Table (IVT/IDT): protected vs real mode

External/Hardware Interrupt

: Critical, ASAP dealing, with maskable interrupt disabled

: Noncritical, maskable interrupt enabled

: Noncritical deferrable, sortirq

: NMI: non-maskable interrupt

: INTR

PIC (programmable interrupt controller)

APIC

Internal/Software: int $num

IRQ: interrupt request line, connected to the PIC/APIC pin (do_irq)

: The CPU checks after each instruction if there's an IRQ from the (A)PIC

: If so, consults the IDT to map the received vector to a function

: Checks if the interrupt was issued by a authorized source

: Saves the registers of the interrupted process

: Call the according function to handle the interrupt

: Load the recently saved registers of the interrupted process and try to resume it

Exception

Trap: ++%rip

Fault: same %rip

Interrupt

Malloc: user-level

Brk for small memory

Mmap for large memory

https://yangrz.github.io/blog/2017/12/20/ptmalloc/

Kmalloc: kernel level allocation

Buddy algorithm: Various sized (power of 2) blocks are divided when allocated and coalesced when freed to efficiently divide a big block into smaller blocks of various sizes as needed.

Slab: Very large blocks are allocated and divided once into equal-sized blocks. No other dividing or coalescing takes place and freed blocks are just held in a list to be assigned to subsequent allocations.

OOM Killer, select processes, send SIGKILL

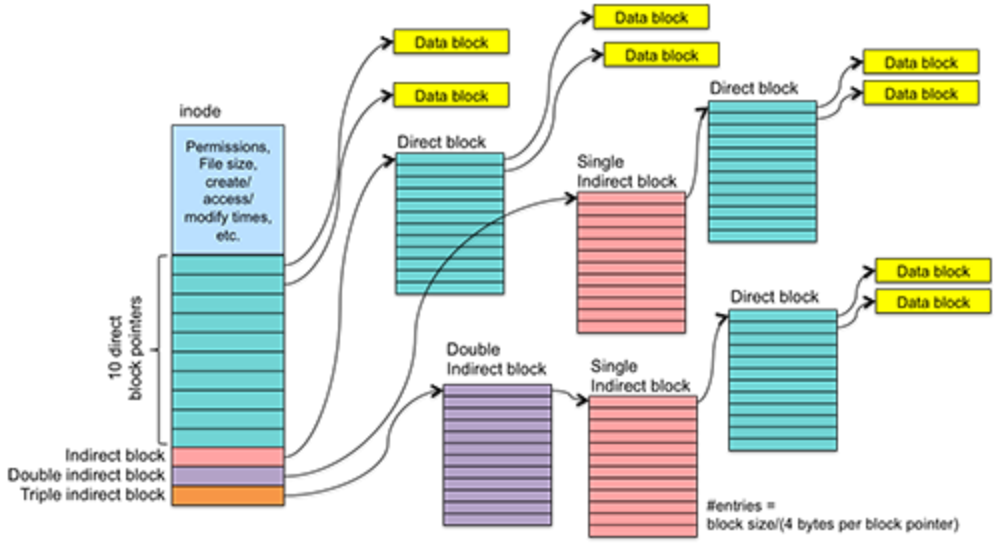
:

https://www.kernel.org/doc/gorman/html/understand/understand016.html

File system

Superblock: system metadata

Inode: metadata of files/directories



12 direct/3 indirect pointers

Data blocks

Dentry: hash data structure of path, found final inode of wanted file, bypassing path traversals

File descriptor: **When we open an existing file or create a new file, the kernel returns a file descriptor to the process.** The kernel maintains a table of all open file descriptors, which are in use. The allotment of file descriptors is generally sequential and they are allotted to the file as the next free file descriptor from the pool of free file descriptors. When we closes the file, the file descriptor gets freed and is available for further allotment.

: file descriptor table in kernel

File: V-node table, shared by all processes, stat structure / Open file table, shared by all processes, refcnt, position

HDD: head, track, cylinder, sector, platter

SSD

NAND flash: read/write pages, erase blocks

Wear leveling

: transparently re-map host address onto NAND address

: group multiple small writes

: update disk block => store new copy

: FTL (flash translation layer), logical block address -> physical memory cells

: Over-provision (advertise less space than reality)

NVMe: Non-Volatile Memory Host Controller Interface Specification because sata slow

RAID (Redundant Array of Independent Disks)

RAID0: disk striping, data written to multiple tasks

: no redundancy, no fault tolerance

RAID1: disk mirroring, same data written to 2 different disks

: half disk usage

RAID5: data & parity (XOR) striped across 3+ disks, can tolerate 1 disk failure recover from other disks (like quorum)

: https://www.zhihu.com/question/20164654

: https://www.v2ex.com/t/597017

RAID6: more parity block than RAID5, tolerate two disk failure

RAID10: RAID1 + RAID0

RAID Z2/3...

SCSI/SATA drivers

: /dev/sd{a, b, c}...

IDE drivers

: /dev/hd{a, b, c}...

: http://www.voidcn.com/article/p-whxeoisl-hm.html

Sector addressing: CHS vs LBA (logical block address)

MBR Partition: logically independent section of a hard disk drive

Primary: any of the four possible first-level partitions into which a HDD on an IBM-compatible personal computer can be divided

Extended: Only one primary partition can be used as an extended partition, and it can be created from any of the primary partitions.

Logical: a partition that has been created inside of an extended partition.

Mount

: All files accessible in a Unix system are arranged in one big tree, the file hierarchy, rooted at /. These files can be spread out over several devices. The mount command serves to attach the filesystem found on some device to the big file tree. Conversely, the umount(8) command will detach it again.

: https://linux.die.net/man/8/mount

Link

Hard link: same inode with different filename (refcount)

Soft/Symbolic link (-s): redirection, across file system & machine boundary

Permissions

(su)UID: user-id, run with file owner privilege

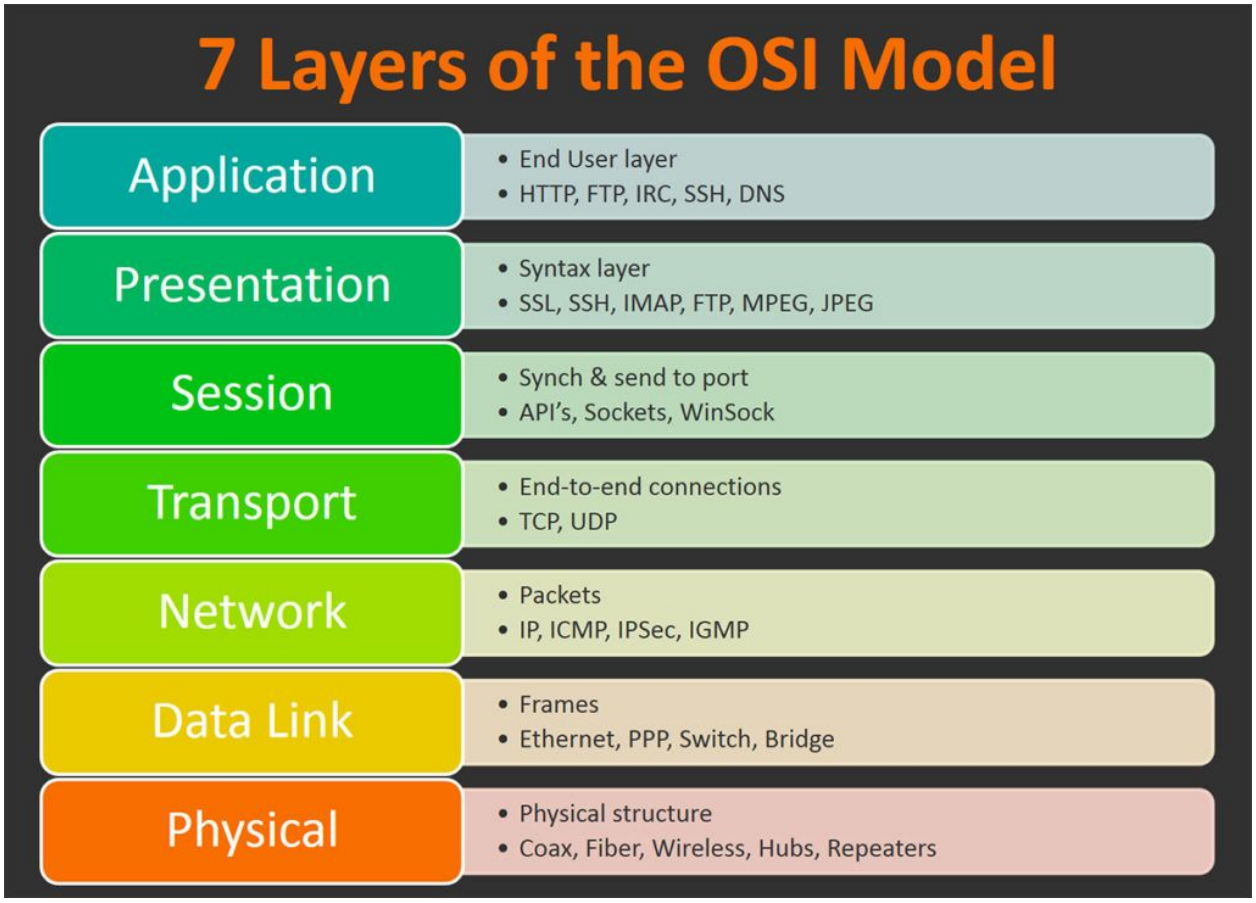
(sg)GID: group-id, run with file group owner privilege

(t)Sticky bits: directory only modified by owners

(uuugggooo)Mode/umask: inverse of user-group-other permission of read-write-execute

chmod, chown, chgrp: change mode/owner/group

Network



DNS lookup

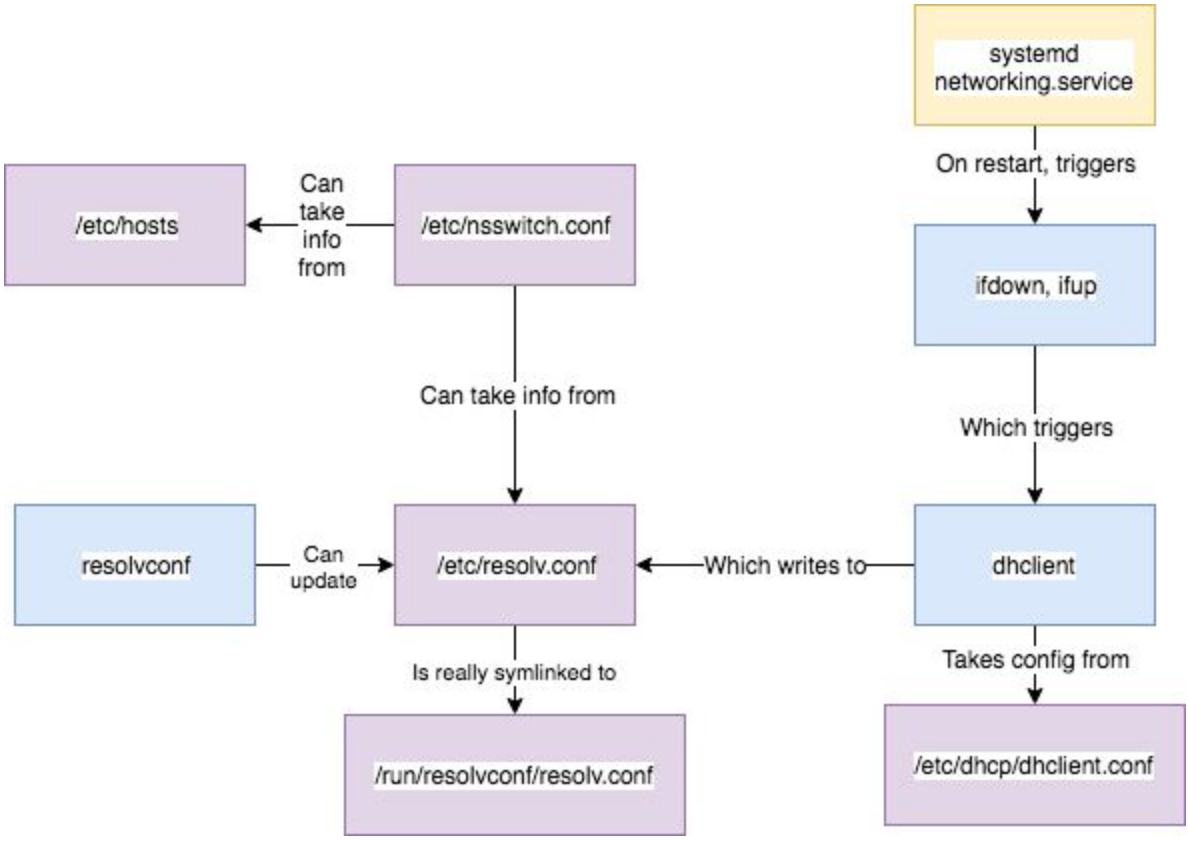
: Browser cache

: getaddrinfo() syscall

: OS DNS cache (dnsmasq)

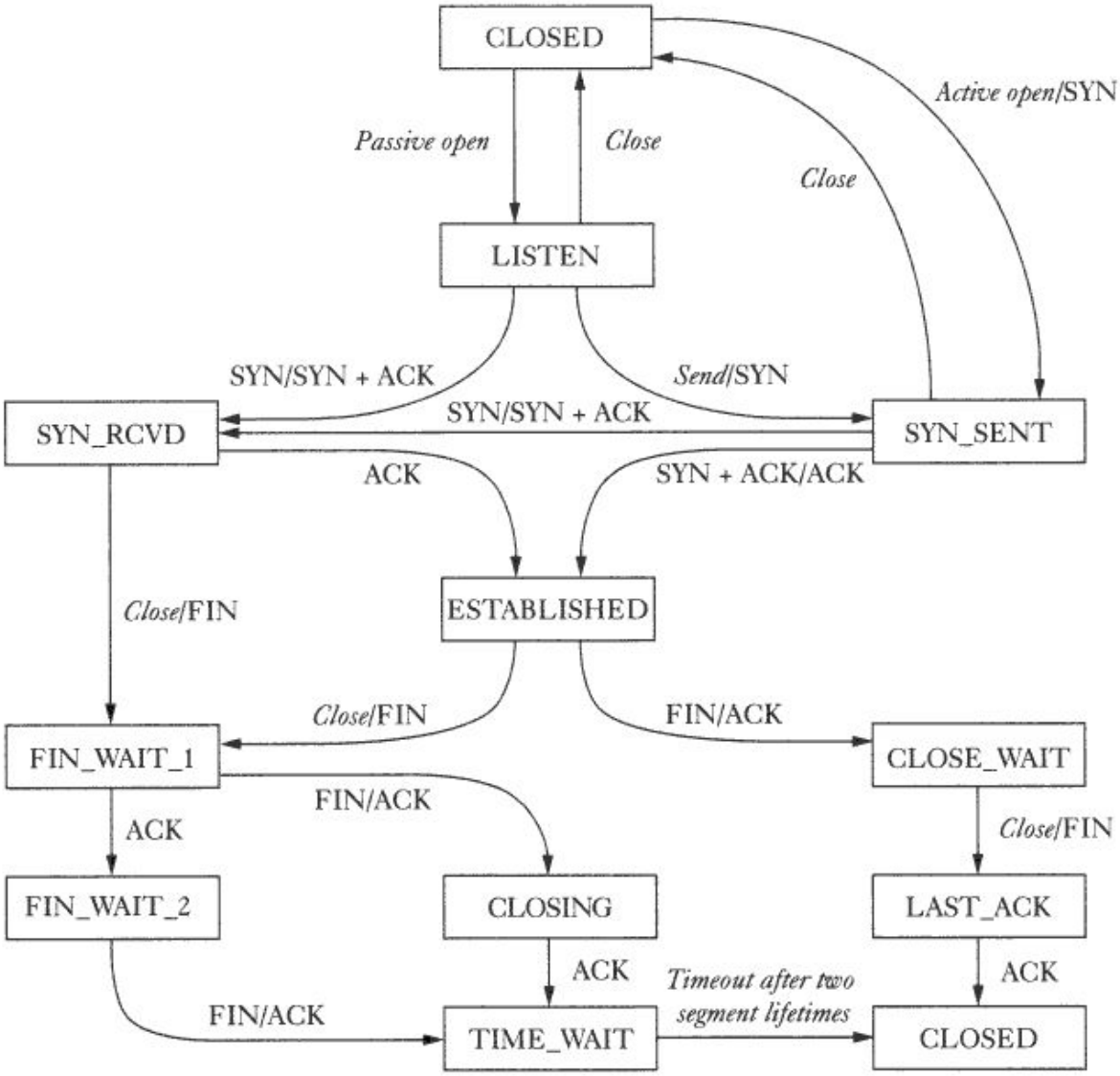
: DNS resolver (resolv.conf for nameservers, nsswitch.conf for lookup order)

: ISP DNS server -> root server -> TLD -> hierarchical



IP: transport packets

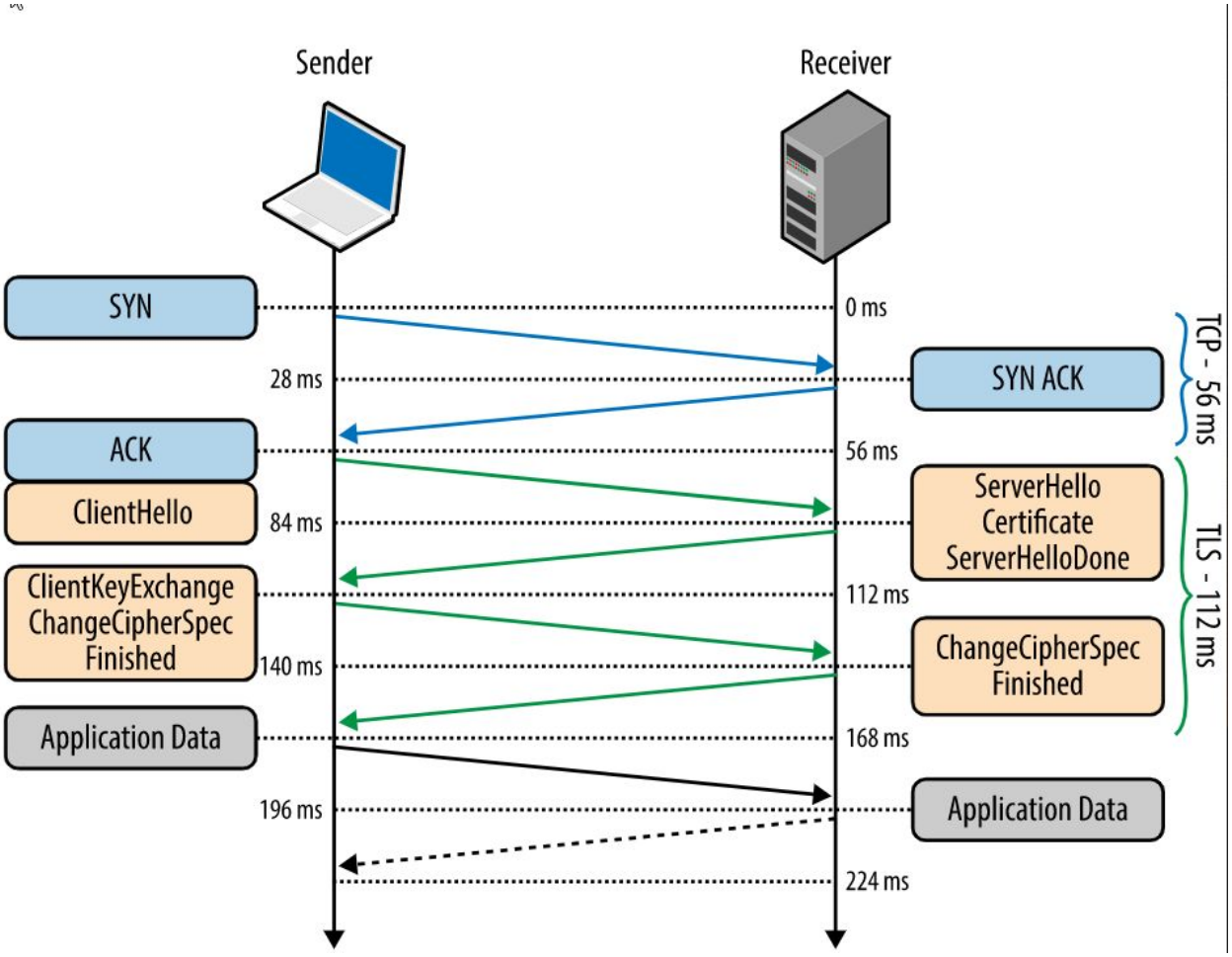
TCP: reliable, duplex, 3 handshake (SYN, SYN-ACK, ACK), 4 close



UDP: datagrams

HTTP: application layer, REST api

SSL handshake



Thread

```
Kernel thread (1 : 1)
      LinuxThreads/NPTL
User thread (m : 1): swapping registers
M : N model
Thread control blocks
      : preempt_count, show if handling interrupts
Tgid: thread group id
```

Process

ID

: pgid: process group id

: pid: process id

: ppid: parent process id

: sid: session id

: uid: user id

: euid: effective user id

:

https://stackoverflow.com/questions/41498383/what-do-the-identifiers-pid-ppid-sid-pgid-uid-euid-mean

ELF sections

: .text: machine code

: .rodata: read-only data, like string literal in C

: .data: initialized global variables

: .bss: block storage start, starts uninitialized with 0

: .symtab: symbol table about functions, global variables

: .rel.text: linker relocation text section

: .rel.data

: .debug: debugging symbol table for local/global variables (DWARF)

: .line: line number to machine code, debugging

: .strtab: string table for symbol tables in .symtab, .debug

: .plt: PLT (Procedure Linkage Table) (IAT equivalent).

: .got: GOT entries dedicated to dynamically linked global variables.

: .got.plt: GOT entries dedicated to dynamically linked functions.

: .dynamic: Holds all needed information for dynamic linking.

: .dynsym: symbol tables dedicated to dynamically linked symbols.

: .dynstr: string table of .dynsym section.

: .interp: RTLD embedded string.

: .rel.dyn: global variable relocation table.

: .rel.plt: function relocation table.

fork/vfork/clone/posix_spawn

: fork: COW

: vfork: don’t copy, parent/child share same page table, same address spaces, as child will immediately execve

: clone: syscall used by fork, creating new process, 1 thread

: posix_spawn: fork+exec

: https://www.microsoft.com/en-us/research/uploads/prod/2019/04/fork-hotos19.pdf

: https://stackoverflow.com/questions/4856255/the-difference-between-fork-vfork-exec-and-clone

exec/execl/execle/execlp/execv/execve/execvp

: transfer argv/envp to top of new stack

: make stack frame for crt startup

: set registers

: L vs V: pass parameters as individual parameters in the call / array of char*

: E: additionally pass environment variables

: P: environment path variable for searching executable file named to execute, otherwise absolute/relative path

: https://pubs.opengroup.org/onlinepubs/009695399/functions/environ.html

Exit

: cleanup, atexit handler, end process

IPC techniques

: pipe

: named pipe

: socket

: queue

: shared memory, shm_*

Shell

: > file, dup file fd to stdout

: < file, dup file fd to stdin

: 2> file: dup file fd to stderr

: 2 >& 1: dup stdout file to stderr

Static library: direct into binary

Dynamic library

Linked at runtime, specified in ELF

`dlsym/dlopen/dlclose`: dynamic load

`ld.so`: dynamic linker

LD_PRELOAD: dynamic hooking

Ldd: return dynamic load library

PLT/GOT:

https://zoepla.github.io/2018/04/%E9%9D%9E%E5%B8%B8%E8%AF%A6%E7%BB%86%E8%A7%A3%E9%87%8Aplt&got/

/proc

:

https://www.tecmint.com/exploring-proc-file-system-in-linux/

CRT

:

http://dbp-consulting.com/tutorials/debugging/linuxProgramStartup.html

PCB: process control block

: pointer to CPU register save area

: pid, ppid

: memory segment info

: scheduler info

Context switch

: save, restore contexts

Scheduling algorithm

SJF, STCF, SRTF

MLFQ (multi-level feedback queuing)

: RR on each priority queue, job slices expires => drop level, from I/O bound => up, random boosting/priority ceiling for each resource/priority donation => avoid priority inversion

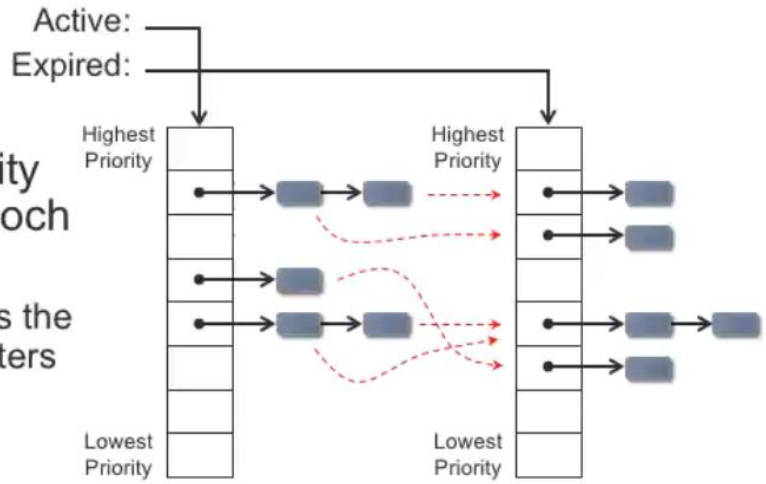
: Lottery scheduling

Multi-core scheduling

: O(1)

Linux 2.6 O(1) Scheduler (2)

- Linux O(1) scheduler maintains two priority arrays:
  - Active array contains processes with remaining time
  - Expired array holds processes that have used up their quantum
- When an active process uses entire quantum, it is moved to the expired array
  - When it is moved, a new priority is given to the process
- When the active priority array is empty, the epoch is over!
  - O(1) scheduler switches the active and expired pointers and starts over again!



: CFS (red-black tree)

Linux Completely Fair Scheduler

- Linux Completely Fair Scheduler (CFS):
  - Instead of maintaining processes in various queues, the CFS simply ensures that each process gets its “fair share” of the CPU
  - What constitutes a “fair share” is affected by the process’ priority; e.g. high-priority processes get a larger share, etc.
- Scheduler maintains a **virtual run time** for each process:
  - Records how long each process has run on the CPU
  - This virtual clock is inversely scaled by the process’ priority: the clock runs slower for high-priority processes, faster for low-priority
- All ready processes are maintained in a red-black tree, ordered by increasing virtual run times
  - O(log N) time to insert a process into the tree

: write to %cr3 will clear TLB, or INVLPG based on PCID

States

: Zombie/Defunct(Z): completed execution but still having entries in the process table, waiting statuses collected

: Stopped(T): suspended, stopped

: Running/Runnable(R): just waiting for CPU to schedule

: Interruptible Sleep, Blocked(S): waiting for an event to complete

: Uninterruptible sleep(U): can’t be killed by signal, may be bad

Daemon process: runs as a background process

: http://www.microhowto.info/howto/cause_a_process_to_become_a_daemon.html#idp24848

Orphan process: parent terminated (parent pid becomes 1)

Zeroing process

User mode

: ring 3

Kernel mode

: ring 0

Syscall

: https://github.com/torvalds/linux/blob/master/arch/x86/entry/syscalls/syscall_64.tbl

: syscall(num) -> passed in %rax, int 0x80

: parameter passing based on amd64 ABI

: rdi, rsi, rdx, r10, r8, r9

: switch from user stack to kernel stack

Context switch

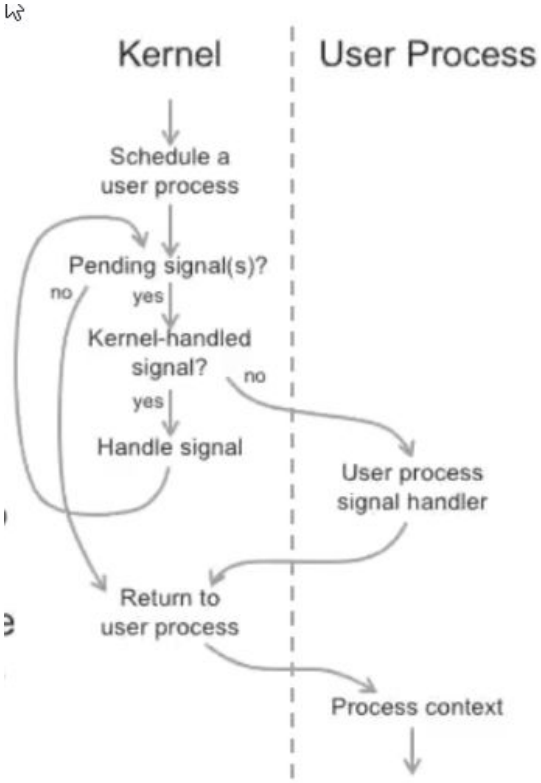
Signal

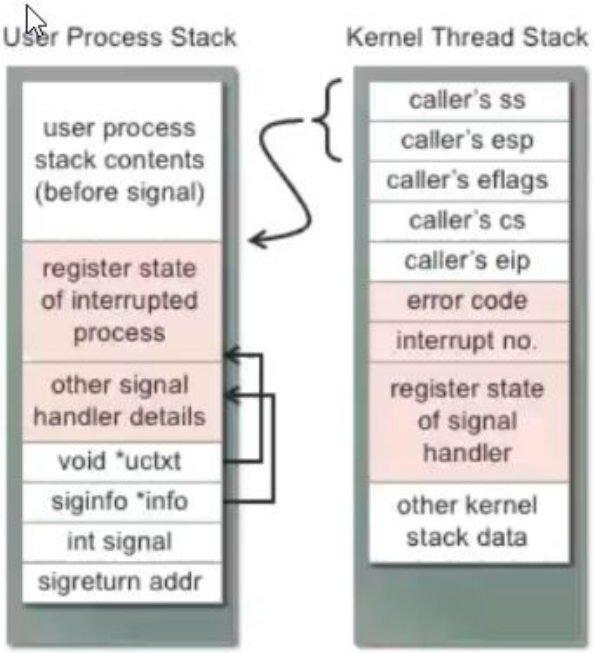
Term(inmate) / Core(dump) / Ign(ore) / Stop / Cont(inue)

Can’t ignore: SIGKILL, SIGSTOP, enforced immediately by the kernel the next time the process runs

Kernel pending bit-vector

Kernel blocked bit-vector





Signals	Operations
SIGINT	Produces receipt for an active signal
SIGTERM	Sends a termination request to the program
SIGBUS	Bus error which indicates access to an invalid address
SIGILL	Detects an illegal command
SIGALRM	This is used by alarm() function and indicates expiration of timer
SIGABRT	Termination of a program, abnormally
SIGSTOP	The signal cannot be blocked, handled and ignored and can stop a process
SIGSEGV	Invalid access to storage
SIGFPE	Overflow operations or mathematically incorrect operations like divide by zero.
SIGUSR1 SIGUSR2	User defined signals

The signal() function

Kernel module

: `ln -s module.ko /lib/module/`uname -r`` add module to module directory

: sudo depmod -a: update dependency list

: sudo modprobe module.ko install kernel module

: modprobe -r module.ko remove kernel module

Shell/tty

Terminal = tty = text input/output environment (device file with ioctls)

:

https://unix.stackexchange.com/questions/4126/what-is-the-exact-difference-between-a-terminal-a-shell-a-tty-and-a-con

Console = physical terminal

shell = command line interpreter

解决问题/Troubleshooting

I/O bound

: Disk is full? Use df

: typically if using SSD

: swap file issues

: /proc/sys/vm/swappiness

: /proc/sys/vm/watermark_scale_factor

: /proc/sys/vm/vfs_cache_pressure

: HDD is old? Bad physical block device

: fsck for disk consistency

: fragmentation (unlikely since ext4 handles data blocks across disk), e4defrag -c /

: replace disk

: Caching

: increase disk cache hit rate

: Fragmentation

: SSD leveling

how to know what's going on with hard disk or other devices which are used to write and read data.
problem with vmstat: only tells you the system globally.
Scenario: high load average (measures the # of processes that are waiting to be executed on the process queue, if blocked because of I/O, the cpu would just sit there being idle), say about 15,20.

- Vmstat check bo bi
- look at ps output, what process could be the cause
- look at the processes in uninterruptible sleep state. They are waiting for the disk to return back the data and return from their system calls. Likely to be involved in the high load system
- find the suspect ps aux | grep " D" (uninterruptible sleep state)
- they might just be normal, but to examine: attach strace with a specific id strace -p 12766
- If output of process is yes, shows us all the system calls the process is doing .
    - Verify system call are actually read and write.
- to kill or to do whatever

Kill fork bomb

: vmstat -f for # of forks

: kill $(ps -ax --sid <session id> -o pid=) (pgrep?)

OOM problem

: dmesg to find OOM killer

: force swap, reduce performance downgradation, I/O bandwidth occupied

: Monitoring scripts to avoid happening again

: /proc/pid/{cgroup, limits, oocm_score_adj, status}

: cgroup, Docker

: limit.conf

: ulimit

: use valgrind for memory leaking

Network issue

: netstat for loss rate

: ping, telnet, traceroute for connection issues

: NFS? Check iostat -x, lsof, pidstat for wait proportion

CPU load high, usage low

: Many processes waiting

: vmstat “proc.r” -> uninterrupted sleeping

: ps -aux checking -D process

: strace -p, lsof, lockdeps checking locking

: I/O issue

: pidstat -d: per process general io info

: iostat -x: which device has high io wait

: lsof: list open file by process/command

: perf sched record && perf sched latency

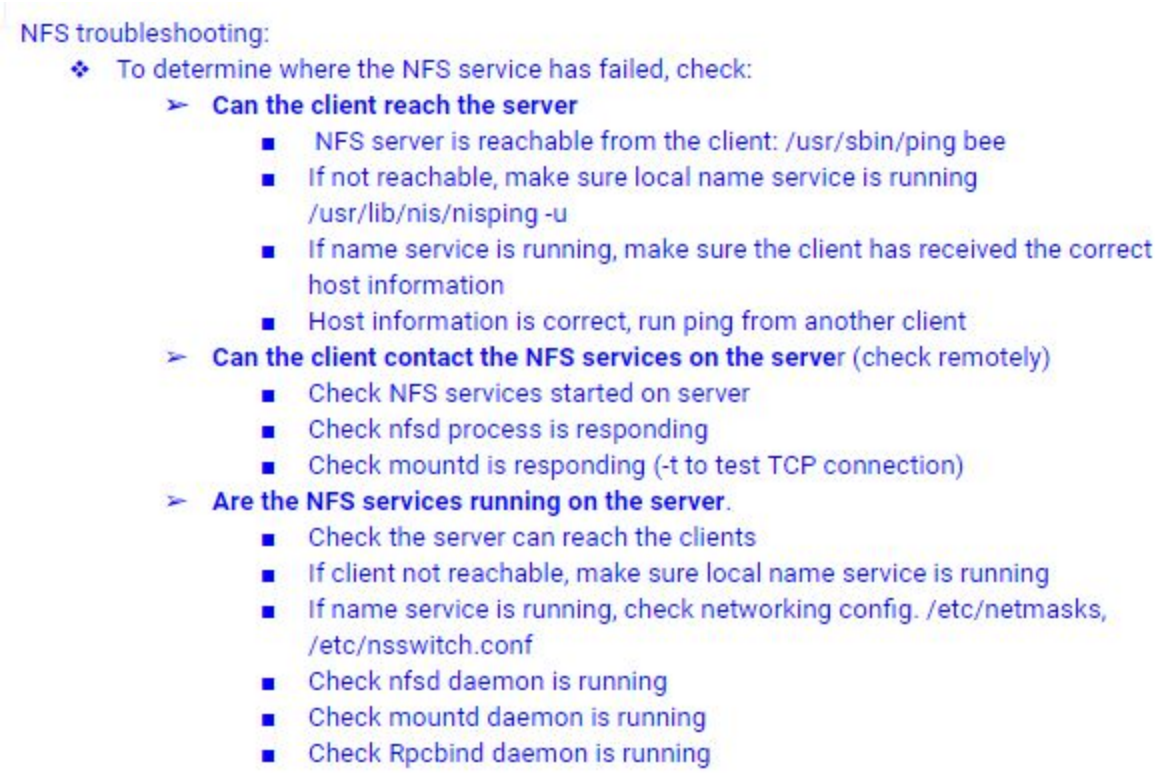
Swap usage, vmstat “swap.si/so

Increase bottleneck disk device performance

Increase memory if swap bounded

Solve deadlocking by {change logic, wound-wait, wait-die} if deadlocking

NFS issue



Database query slow (suddenly I/O increase?)

: RAID read/write performance

: reduce DB index size

: replication? Write bound

: not enough memory?

: small buffer pool?

: bad disks?

: rollbacks frequently

: set table indexes

: RDBMS -> NoSQL for scalability

工具/Tools

Vmstat: report virtual memory / swap / block devices / context switches statistics, used for identifying performance bottlenecks

: https://www.cnblogs.com/ggjucheng/archive/2012/01/05/2312625.html

: https://linux.die.net/man/8/vmstat

Raw

```
Procs
    r: The number of processes waiting for run time.
    b: The number of processes in uninterruptible sleep.
Memory
    swpd: the amount of virtual memory used.
    free: the amount of idle memory.
    buff: the amount of memory used as buffers.
    cache: the amount of memory used as cache.
    inact: the amount of inactive memory. (-a option)
    active: the amount of active memory. (-a option)
Swap
    si: Amount of memory swapped in from disk (/s).
    so: Amount of memory swapped to disk (/s).
IO
    bi: Blocks received from a block device (blocks/s).
    bo: Blocks sent to a block device (blocks/s).
System
    in: The number of interrupts per second, including the clock.
    cs: The number of context switches per second.
CPU
    These are percentages of total CPU time.
    us: Time spent running non-kernel code. (user time, including nice time)
    sy: Time spent running kernel code. (system time)
    id: Time spent idle. Prior to Linux 2.5.41, this includes IO-wait time.
    wa: Time spent waiting for IO. Prior to Linux 2.5.41, included in idle.
    st: Time stolen from a virtual machine. Prior to Linux 2.6.11, unknown.
```

Ps: show process information, often -aux for all processes

: https://cheatsheet.dennyzhang.com/cheatsheet-process-a4

Strace: display all syscalls for an operation

:

https://arthurchiao.github.io/blog/how-does-strace-work-zh/

Ltrace: display all dynamic library calls for an operation

:

https://arthurchiao.github.io/blog/how-does-ltrace-work-zh/

Top: display Linux processes with dynamic view of running system

:

https://askubuntu.com/questions/176001/what-do-virt-res-and-shr-mean-in-the-top-command

Free: display the amount of free & used physical & swap memory

Du: disk usage, size per file, cound sum of file size

Df: disk free, looks for superblock metadata

Telnet: communicate with another host using the TELNET protocol

Ping: uses the ICMP protocol's mandatory ECHO_REQUEST datagram to elicit an ICMP ECHO_RESPONSE from a host or gateway.

Netstat: network statistics

:

https://www.tecmint.com/20-netstat-commands-for-linux-network-management/

stat/xstat/lstat/lxstat: get file/symbolic file status

Lsof: report list of open files

Pstree: show all parents of this process

Dmesg: kernel log dump

FSCK: check file system consistency

: superblocks

: inode (free blocks) + inode state + inode link

: duplicate blocks, bad blocks, directories

Traceroute

Nslookup

Ifconfig

Iostat

iotop

What happens when

URL searching

: https://github.com/skyline75489/what-happens-when-zh_CN

: IRQ interrupt to PIC

: Browser parsing

: DNS lookup

: ARP for different subnets

: Socket for TCP connection

: TLS encryption

: HTTP protocol

: HTML/CSS parsing

: GPU rendering

Syscall

: int $0x80, dispatching from sys_call_table / sysenter

: pass arguments in registers (rdi, rsi, rdx, r10, r8, r9)

: rax for syscall number

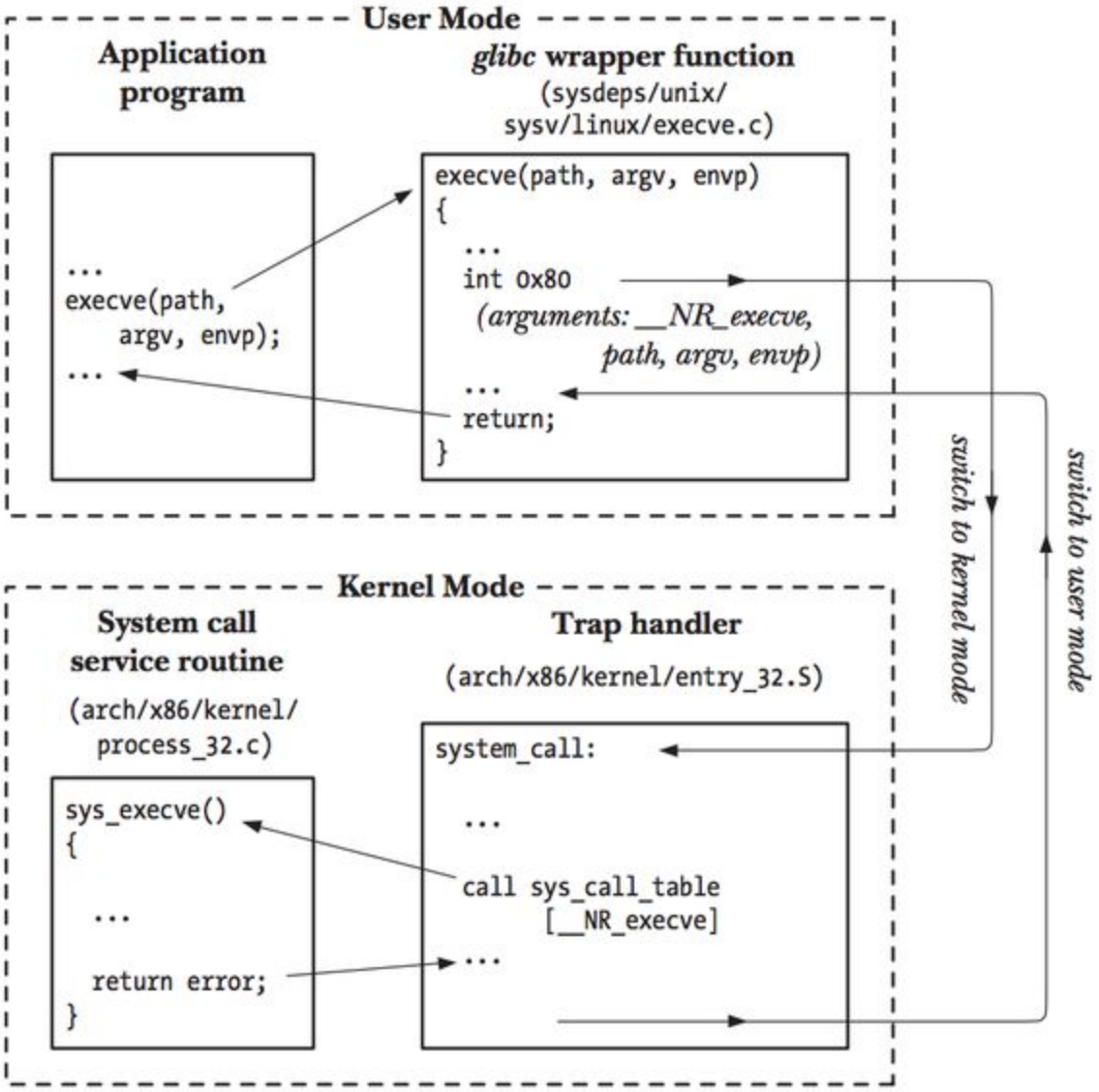
: rcx for return address to user space

: address should be within userspace

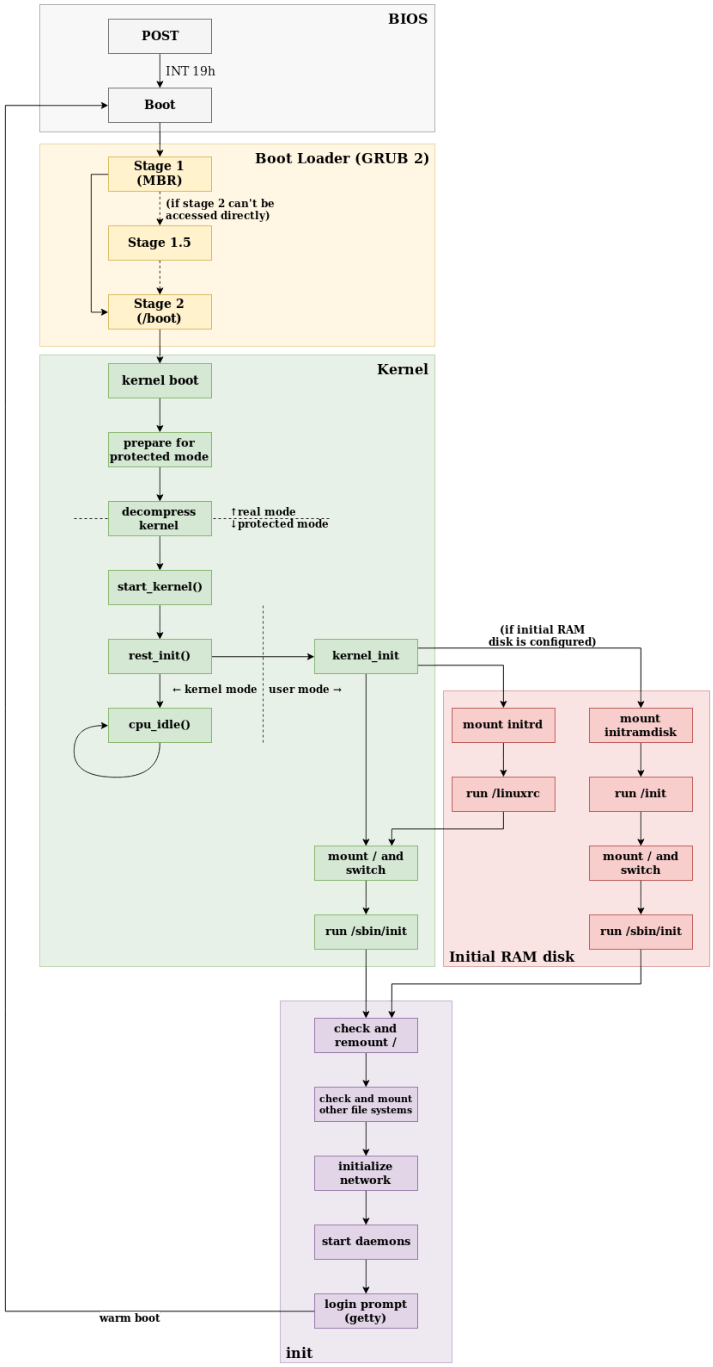
: save user stack, switch to kernel stack (swapgs)

: return from syscall, restore user stack

: https://cloud.tencent.com/developer/article/1492374


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

Bootstrap



: real mode, executing ROM code at 0xFFFF0 (jump to actual BIOS entry point)

: BIOS-MBR vs UEFI-GPT

: http://conanwhf.github.io/2016/08/26/GPTandMBR/

: BIOS

: code in read-only memory (EPROM-EEPROM-FLASH)

: configure hardware details (password, RAM rate, bus speed, boot device order)

: provide some device drivers

: perform POST (power on self test)

: scan for critical resources (RAM, GPU, Disk, keyboard)

: get list of bootable devices

: look for devices to be load, load its MBR into memory, jump there (boot-sector of first bootable device)

: Boot-Device search

: Boot-Sector launch

: Boot sector is copied to 0x7C00

: Execution is transferred to 0x7C00

: Extra step for hard disk or CD-ROM

• Boot sector (“MBR”) knows about partitions

• BIOS starts it running at 0x7C00, of course

• Copies itself elsewhere in memory, jumps there

• Loads “active” partition's boot sector at 0x7C00

Now we’re executing the boot loader – the first “software” to execute on the PC

: bootloader (512 bytes)

: GRUB

: load/use driver to read /boot (/boot/kernel.gz), configurations

: entire OS or multi-stage

: BIOS => first sector => load rest of bootloader => boot menu => switching mode for OS > 1MB => load done => jump to kernel entry point

: switch to protected mode if need more memory

: decompress kernel header image, load into memory

: kernel

: detect memory layout, prepare basics to devices

: prepare IVT, GDT to enter protected mode

: disable interrupts

: load Global Descriptor Table Register (GDTR) with a pointer to GDT

: load Task Register with a simple Task State Segment for protected-mode interrupt handling

: turn on protected mode by writing to %cr0

: setup %cr3

: force CPU to load protected mode segment selectors by performing a longjmp to next instruction

```
                    : set other segment register to kernel-data
segment selector value
                    : load Interrupt Descriptor Table Register (IDTR)
with a pointer to the interrupt descriptor table for the OS
                    : re-enable interrupts
                : decompress, fully loaded
                : initialize page table
                : kernel boot parameters
                    : mount initrd.img ram disk to /
                    : or initramfs
                : load ACPI table
                : start init process, (systemd)
            : init (systemd), pid = 1
                : init remount device by reading /etc/fstab
                : continue to start service
/etc/systemd/system/default.target
                : continue to satisfy other targets
            : https://medium.com/@cloudchef/linux-boot-process-part-1-e8fea015dd66
            : https://medium.com/@cloudchef/linux-boot-process-part-2-bd7514913495
```

Press Ctrl+C

```
        : push into kernel pending signal queue
```

: master device of pty receiving interrupt character (Ctrl+C)

: pgid equal to foreground pgid processes receives SIGINT

: process read from fd of atty, receives SIGTTIN

: tty terminated: matching session id process will receive SIGHUP, use nohup or setsid to let process in daemon

: `sigaction`

Execute “telnet google.com 80”

: bash layer operations…

: setup TCP/IP connections (sockets operation)

: down to link layer & physical layer

Type “ls -l” & Enter

: type `strace ls -al *.c` by yourself

: Bash parsing

: Bash converts wildcards

: Bash finds command in PATH

: stat(“ls”)

: pipe if needed (|)

: fork()

: COW, cr3 register

: kernel allocate process descriptors

: child process created

: dup() if needed (|, redirection)

: set foreground pgid

: parent process wait() (unless with &)

: execvp()

: allocate zeroed page to progress from zeroed pages queue, created in kernel free time (zeroing process)

: kernel replace most of the page entries (like text section)

: load env, arg based on systemABI

: glibc loades dynamic linked library into memory

: getopt()

: openat(“directory”) = fd

: getdents(fd)

: kernel uses inode information & data blocks

: readdir("directory name”) for sub-dirs

: lxstat, xstat(“file path”) for access control, metadata

: write() to stdout

: child process exit()

: parent process exit()

: along with SELinux & locale operations & string operations & getenv

Questions

1. How Linux preemption happens?

    https://juejin.im/post/5a97c9025188255579180e43

    https://blog.csdn.net/jnu_simba/article/details/11745743

    CFS - red-black tree, weighted by nice value

    User preemption

        进程状态转换的时刻：进程终止、进程睡眠；

        当前进程的时间片用完时（current->counter=0） -> 中断

        设备驱动程序

        从系统调用返回用户空间

        从中断处理程序返回用户空间

    Kernel preemption: nested preemption

        中断处理正在执行，且返回内核空间之前

        内核代码再一次具有可抢占性时

        内核中的任务显式地调用`schedule()`

        内核中的任务被阻塞

2. How Linux kernel starts threads in multi-core environment?

Affinity (sched_setaffinity)

Idle process (cpu_idle()):

https://elixir.bootlin.com/linux/v3.1.6/source/arch/x86/kernel/process_64.c#L109

Start at bootstrap process by main cpu interrupt and guiding: https://blog.csdn.net/cs0301lm/article/details/41078599

Load balancing:

https://blog.csdn.net/wh8_2011/article/details/52089438

Work stealing: child domain -> parent domain -> group -> queue

Push tasks into remote CPU's related kernel structures (memory), idle process detects and does schedule.

https://www.ibm.com/developerworks/cn/linux/kernel/l-k24sch/index.html

3. VDSO (Virtual Dynamically-linked Shared Object)

https://vvl.me/2019/06/linux-syscall-and-vsyscall-vdso-in-x86/

https://arthurchiao.github.io/blog/system-call-definitive-guide-zh/

Complementary to syscall (gettimeofday)

A virtual dynamic-linked library, mapped to every process, ASLR into ELF header

Update_vsyscall for updating information

4. Common Synchronization techniques

Spinlocks, Futex, Reader/Writer Locks

Barrier, Latch, Countdown

RCU (https://blog.csdn.net/junguo/article/details/8244530), Hazard pointer

5. Timer implementation

https://0xax.gitbooks.io/linux-insides/Timers/

6. Vmstat explanation

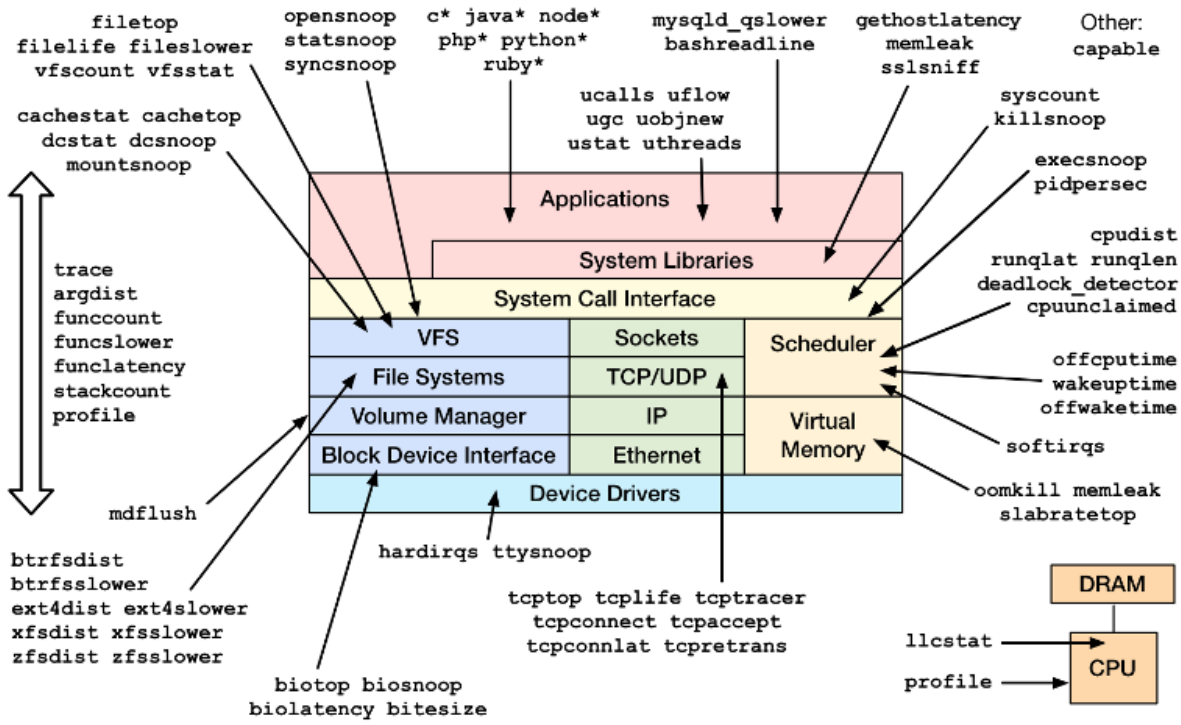
Explain parameters

https://access.redhat.com/solutions/1160343

System time bounded -> Syscall

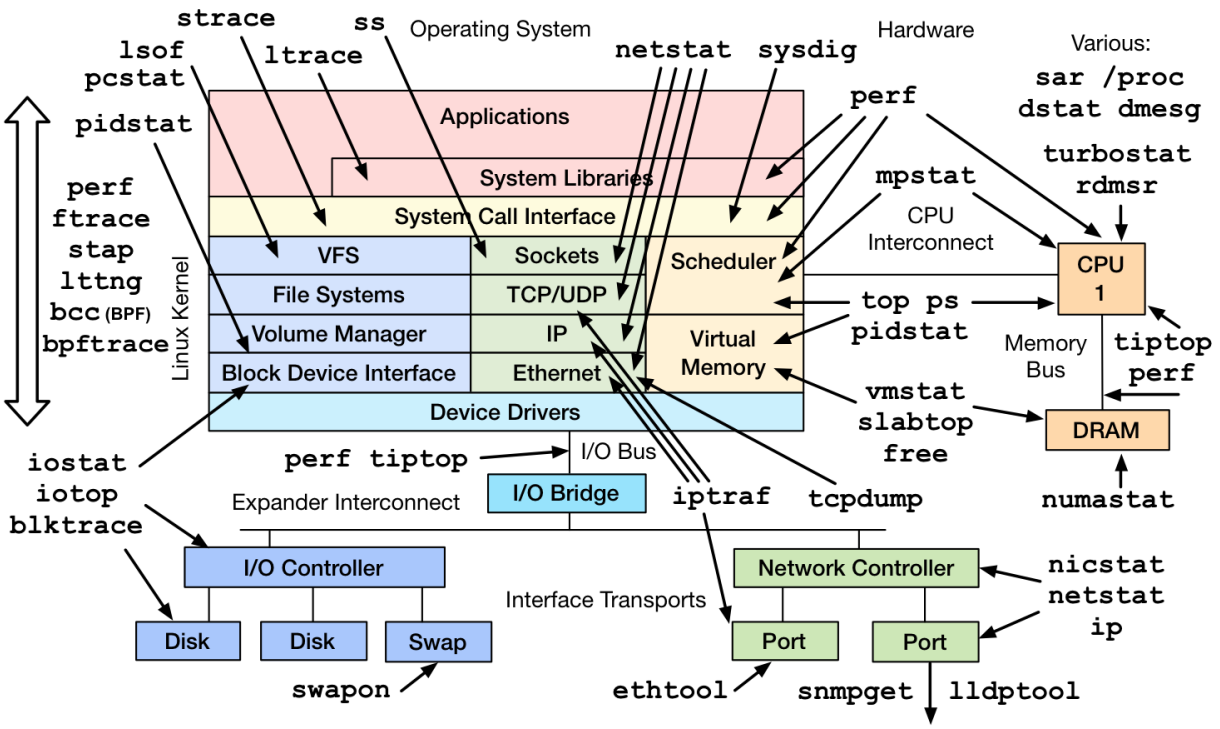
```
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu------
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 2  0 402944  54000 161912 745324    5   14    54    59  221  867 13  3 82  2  0
 1  0 402944  53232 161916 748396    0    0     0     0   30  213  3 97  0  0  0
 1  0 402944  49752 161920 751452    0    0     0     0   28  290  4 96  0  0  0
 1  0 402944  45804 161924 755564    0    0     0     0   29  188  2 98  0  0  0
 1  0 402944  42568 161936 758608    0    0     0 17456  272  509  7 93  0  0  0
```

Linux bcc/BPF Tracing Tools



https://github.com/iovisor/bcc#tools 2017

Linux Performance Observability Tools



http://www.brendangregg.com/linuxperf.html 2018

Linux Performance Tools