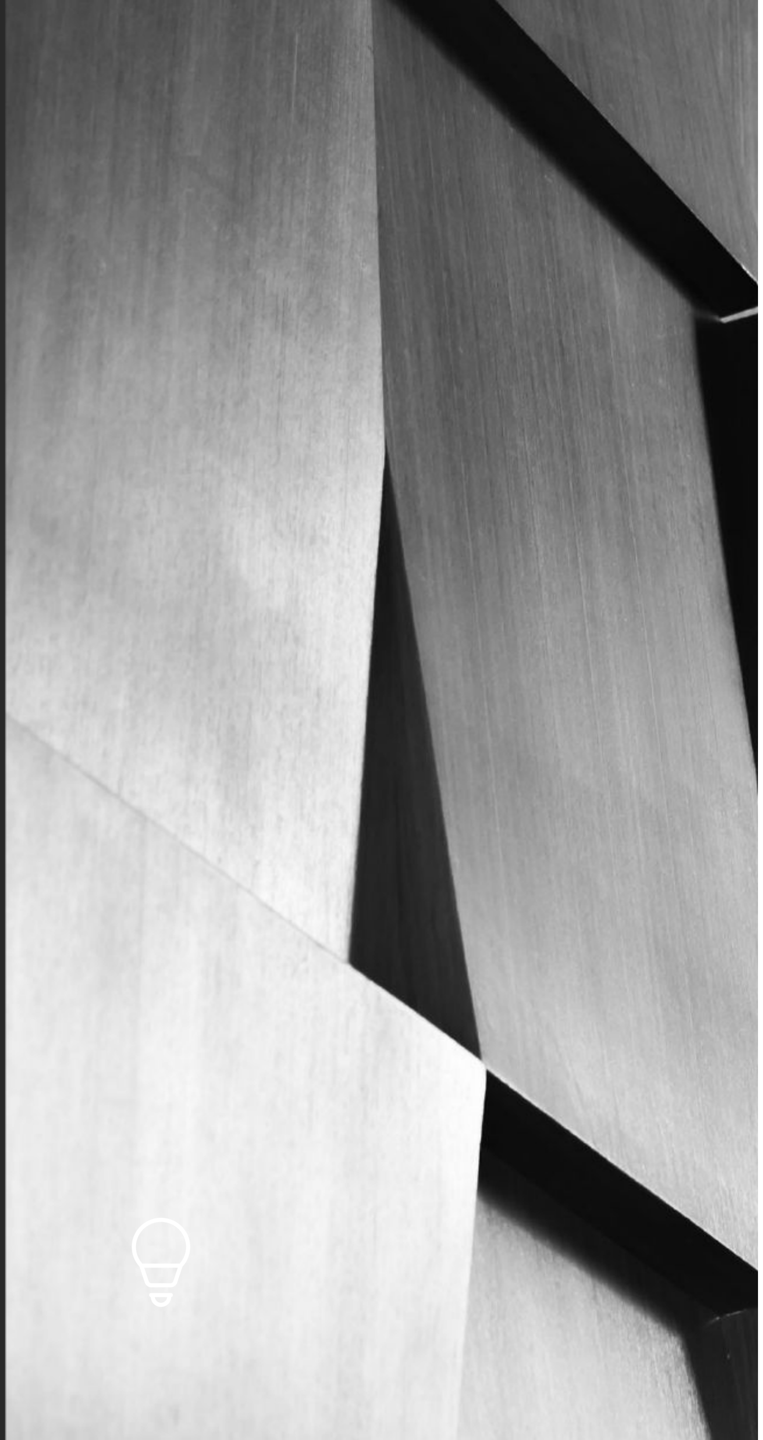


Basic intro to Linux Kernel

... at least some bits of it

Vratislav Bendel
Principal Software Maintenance Engineer

Red Hat

What we'll cover today:

- **Memory Management**
  High level overview
- **Scheduling & interrupts**
  Processing flow
- **Basic structs**
  Primitives you should know
- **Basic debugging**
  So you won't get lost

Role of the kernel

... and it’s scope

Red Hat

Role of the kernel

… and it’s scope

- Interface between userspace and HW
- HW control
- Syscall interface
- Memory & process management

Red Hat

Memory Management

... just high level basics

What is memory?

Kernel's perspective

- Memory is addressed in blocks called **"pages"**
- A single **page size** depends on architecture:

  x86 ~ 4 kB ppc64, s390 ~ 64 kB

- Kernel defines **memory zones**: DMA, DMA32, Normal **
- Contiguous memory is packed into blocks of higher **page order**,

  up to order 10 [4MB of contiguous memory]

- /proc/buddyinfo

** Zone definitions were in the past and may in the future be subject to change

Kernel's memory

It's a program as any other, right?

The Linux kernel is itself a C program with it's own structs and data.

Therefore it itself requires to allocate memory.

- Kernel uses a concept called **"slabs"**, a.k.a. **"kernel memory caches"**
- A page block gets "split" into dedicated objects of specific size (for example *inode cache, dentry cache, …*)
- Generic allocations via *kmalloc()* fall into *kmalloc-XX* slabs
- Large allocations (uncommon) fall to process-like vmalloc
- /proc/slabinfo

… more on the MM lecture

Memory “types”

Types by usage

- **Cache** memory

  Data fetched from disk, stored in main memory for faster access

- **Anonymous** memory

  Transient/working process data - structures, variables, …

- **Kernel** memory

  Slabs, per-cpu, page-tables – all kernel structures

- /proc/meminfo

Memory “types”

Process memory

- **Virtual** memory ← *can **overcommit***

  Kernel “acks” process’ allocation and creates
  a *Virtual Memory Area [VMA]*

- **Physical** memory

  Data actually stored in Main RAM
  RSS – Resident Set Size

Linux MM design

Page fault

Virtual addresses (process context) are ***mapped*** to physical addresses (in RAM). These mappings are stored in ***"page tables"*** and CPU caches them in the *Translation Lookaside Buffer [TLB]*.

- When a virtual-to-physical mapping does not exist, the CPU generates a HW exception ***"page_fault"***
- Kernel handles the page_fault by allocating the physical page/s and creating the mapping.

Linux MM design

Lazy allocation scheme

Linux kernel utilizes a *lazy* allocation scheme:

- Processes are not allocating memory, they create virtual mappings
- Physical memory is allocated via page_fault, only once a virtual address is actually accessed (read or write)

Linux MM design

Allocation algorithm

Linux kernel is designed to utilize available resources.

- Physical allocations take ‘free’ memory page/s while available *(fast path)*
- Once ‘free’ memory is low (based on zone watermarks), the kernel needs to ***reclaim*** some memory back into the ‘free’ pool *(slow path)*

Reclaim ~ Either drop cached or swap out anon pages

Scheduling & Interrupts

Processing flow

Scheduling

Scheduling vs. Load Balancing

- Scheduling

  How long should a process run (when it should be rescheduled)

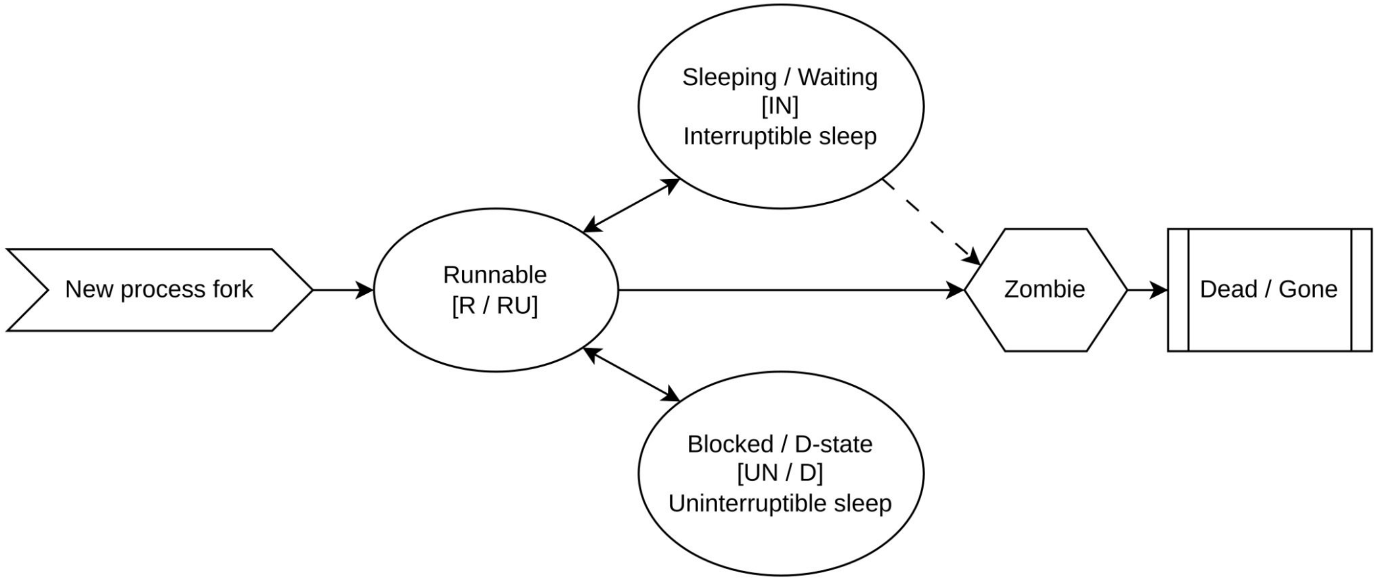
  Which process should be selected to run ‘next’

- Load Balancing

  Where (on which CPU) should a process be scheduled to run

Scheduling

Process states



- Runnable processes are *enqueued* to *runqueue* of one of the CPUs

Scheduling

Algorithm

- The function ***schedule()*** picks the 'next' task from local CPU's runqueue and switches it to be the now-actively running task.
- The 'prev' task, if still RUnnable, is enqueued back to runqueue

- The *pick_next_task()* depends on the Scheduling algorithm

  EEVDF (older CFS), Real-Time (FIFO / RR), Deadline

Ref.: Earliest Eligible Virtual Deadline First

$ man 7 sched

Interrupts

Control CPUs

A CPU executes binary code in incremental way. Hence in order to change what it does, you need to **interrupt** the flow (simply said).

- Hardware interrupts from various devices register an *irq_handler* function on the specific vector, which does the utmost necessary work.
- Bottom half (*"softirq"*) parts are processed in kernel thread context.

- IRQs can be disabled for certain critical sections (except NMIs, SMIs, ..)

Interrupts

Timers & IPIs

Kernel naturally used various “software interrupt” schemes

- **Timers** (basic [jiffies], high-resolution [cpu-clock], perf events [NMI])

  For example: **kernel tick** (scheduler tick), watchdogs, …

- **IPIs** (Inter-Processor Interrupts)

  Reschedule pings

  TLB shootdowns

  Remote function calls / sync (“*smp_call_function*()*”)

- /proc/interrupts

Basic structures

Primitives you should get familiar with

Linked List

The bread and butter

Linux kernel uses a **doubly-linked list** primitive data structure:

- *include/linux/types.h*

  *struct list_head { struct list_head *next, *prev; };*

- *include/linux/list.h*

  *LIST_HEAD[_INIT](name), list_empty() // **head == next == prev***

  list_add(), list_add_tail(), list_del(), ... // LIST_POISON[1|2]

- List heads are **embedded** in associated structs

Hlist

Deletion efficiency

Hash lists are used when efficient deletion is preferred, like hash tables.

- *include/linux/types.h*

  *struct hlist_head { struct hlist_node *first; };*

  *struct hlist_node { struct hlist_node *next, **pprev; };*

- *include/linux/list.h*

  HLIST_HEAD[_INIT](name) INIT_HLIST_NODE(n)

  *hlist_empty(hlist-head *h) // h->first == NULL*

  hlist_add_head(), hlist_add_before(), hlist_del(), ... // LIST_POISON[1|2]

container_of()

Offset macro

The *container_of()* function macro is used to offset a pointer to the beginning/root of the structure, where the pointer is an embedded linking data structure, for example list, rb_tree node or other structures. Effectively it simply translates to assembly pointer arithmetic.

- *include/linux/container_of.h*

  #define container_of(ptr, type, member)

  @ptr: the pointer to the member.

  @type: the type of the container struct this is embedded in.

  @member: the name of the member within the struct.

Red Hat

Red-Black tree

When you need ordering

Red-Black tree structure is used when ordered lookup is needed.

Optionally a variant with cached leftmost element for O(1) search.

- *include/linux/rbtree_types.h*

```
struct rb_root {
    struct rb_node *rb_node;
};
struct rb_root_cached {
    struct rb_root rb_root;
    struct rb_node *rb_leftmost; };
```

```
struct rb_node {
    unsigned long  __rb_parent_color;
    struct rb_node *rb_right;
    struct rb_node *rb_left;
};
```

- rb_nodes are also **embedded** in associated structs.

Red-Black tree

When you need ordering

- General API implemented in:

  lib/rbtree.c

  *void rb_insert_color(struct rb_node *node, struct rb_root *root)*

  *void rb_erase(struct rb_node *node, struct rb_root *root)*

- Although many subsystems implement their own customized functions.

Radix tree, Xarray, Maple tree

Lookup in huge datasets

Radix tree (oldest), *Xarray* and *Maple tree* are data structures used primarily to keep track of ***process' address space VMAs*** (incl. ***pagecache***).

- Radix tree is a "*compressed trie*" which maps long integer keys to pointer values. It's good at storing huge datasets (address space), but has certain inconveniences, namely when sparsely populated.
- Xarray and Maple tree (newest) are special data structures implemented** specifically to tackle the problematics of storing address space and other similar data more efficiently.

** Kudos to **Liam Howlett** and **Matthew Wilcox** from Oracle, inc.

Radix tree, Xarray, Maple tree

Lookup in huge datasets

- Radix tree: (being deprecated)

  https://lwn.net/Articles/175432/

- Xarray:

  https://docs.kernel.org/core-api/xarray.html

- Maple tree:

  https://docs.kernel.org/core-api/maple_tree.html

void* abstractions & unions

Polymorphism

Certain structures may be used in different ways or different contexts or different subsystems, …

- void* struct members are used for context-relative pointers
- C-lang unions define different uses of a given struct
- The code which works with the struct knows the context
- Commonly paired with specifying object types via flags members

void* abstractions & unions

Polymorphism

- Example excerpt: *include/linux/mm_types.h*

```
struct page {
        unsigned long flags;
        union {
            struct {   /* Page cache and anonymous pages */
                  ...
                  void* private;
            };
            struct {   /* page_pool used by netstack */     ...      };
            struct {   /* Tail pages of compound page */    ...      };
            ...
            struct {   /* Page table pages */               ...      };
     ...  };
```

Bitmasks

Flags, cpumasks, etc.

Bitmasks are widely used to track attributes, types or parameters of various structs or routines, or even control flow of complex algorithms.

- Examples:
  - GFP mask – *include/linux/gfp_types.h*
  - dentry flags – *include/linux/dcache.h*
  - cpumasks – include/linux/cpumask.h

Context macros

Know where you are

The CPU knows certain context information. Kernel code can get it whenever needed:

- *#define current*
    - pointer to the task_struct of the currently executing process on _*this*_ CPU
- *#define smp_processor_id()*
    - int number of the logical CPU where _*this*_ code is executing

Per-cpu structs

Efficiency

To optimize access and remove the need for synchronized access, certain structures are created as ***"per-cpu" copies***. The benefit is that each CPU has its own struct, so there's no need for mutual exclusion.

- Each CPU has its "per-cpu offset"
- Per-cpu structs simply define a pointer/value that must be added to the per-cpu offset to get the pointer to the structure belonging to the specific CPU.

Basic Debugging

... ahh, not again ...

Kernel OOPS log

(Don't) panic

```
[    5.396811] block dm-0: the capability attribute has been deprecated.
[514435.962056] oops_module: loading out-of-tree module taints kernel.
[514435.962106] oops_module: module verification failed: signature and/or required key missing - tainting kernel
[514480.714476] BUG: kernel NULL pointer dereference, address: 0000000000000000
[514480.714521] #PF: supervisor read access in kernel mode
[514480.714540] #PF: error_code(0x0000) - not-present page
[514480.714560] PGD 0 P4D 0
[514480.714575] Oops: 0000 [#1] PREEMPT SMP NOPTI
[514480.714593] CPU: 0 PID: 132782 Comm: insmod Kdump: loaded Tainted: G           OE     -------  ---  5.14.0-427.35.1.el9_4.x86_64 #1
[514480.714643] Hardware name: QEMU Standard PC (Q35 + ICH9, 2009), BIOS 1.16.3-1.fc39 04/01/2014
[514480.714680] RIP: 0010:hello_init+0x5/0xff0 [oops_module]
[514480.714712] Code: Unable to access opcode bytes at RIP 0xffffffffc062efeb.
[514480.714731] RSP: 0018:ffffb29700f33d90 EFLAGS: 00010246
[514480.714747] RAX: 0000000000000000 RBX: 0000000000000000 RCX: 0000000000000001
[514480.714767] RDX: 0000000000000000 RSI: ffffffff883eac23 RDI: ffffffffc062f010
[514480.714796] RBP: ffffffffc062f010 R08: 0000000000000010 R09: 0000000000000000
[514480.714821] R10: ffff9dedd8466401 R11: 0000000000000000 R12: ffff9dedd7dbb700
[514480.714845] R13: ffffb29700f33e28 R14: 0000000000000003 R15: 0000000000000000
[514480.714869] FS:  00007f3ff90fd740(0000) GS:ffff9dee37c00000(0000) knlGS:0000000000000000
[514480.714895] CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
[514480.714915] CR2: ffffffffc062efeb CR3: 00000001f98f6002 CR4: 0000000000770ef0
[514480.714935] PKRU: 55555554
[514480.714945] Call Trace:
[514480.714955]  <TASK>
[514480.714963]  ? show_trace_log_lvl+0x1c4/0x2df
[514480.714980]  ? show_trace_log_lvl+0x1c4/0x2df
[514480.714997]  ? __pfx_init_module+0x10/0x10 [oops_module]
[514480.715014]  ? do_one_initcall+0x41/0x210
[514480.715031]  ? __die_body.cold+0x8/0xd
[514480.715043]  ? page_fault_oops+0x134/0x170
[514480.715064]  ? sysfs_add_file_mode_ns+0x85/0x180
[514480.715084]  ? exc_page_fault+0x62/0x150
[514480.715101]  ? asm_exc_page_fault+0x22/0x30
[514480.715124]  ? __pfx_init_module+0x10/0x10 [oops_module]
[514480.715145]  ? do_init_module+0x23/0x270
[514480.715163]  ? __pfx_init_module+0x10/0x10 [oops_module]
[514480.715183]  ? hello_init+0x5/0xff0 [oops_module]
[514480.715203]  do_one_initcall+0x41/0x210
[514480.715220]  ? kmalloc_trace+0x25/0xa0
[514480.715237]  do_init_module+0x5c/0x270
[514480.715253]  __do_sys_finit_module+0xae/0x110
[514480.715273]  do_syscall_64+0x59/0x90
[514480.715289]  ? syscall_exit_work+0x103/0x130
[514480.715308]  ? syscall_exit_to_user_mode+0x22/0x40
[514480.715327]  ? do_syscall_64+0x69/0x90
[514480.715342]  ? exc_page_fault+0x62/0x150
[514480.715358]  entry_SYSCALL_64_after_hwframe+0x72/0xdc
[514480.716062] RIP: 0033:0x7f3ff883ee5d
[514480.716757] Code: ff c3 66 2e 0f 1f 84 00 00 00 00 00 90 f3 0f 1e fa 48 89 f8 48 89 f7 48 89 d6 48 89 ca 4d 89 c2 4d 89 c8 4c 8b 4c 24 08 0f 05 <48> 3d 01 f0 ff ff 73 01 c3 48 8b 0d 93 af 1b 00 f7 d8 64 89 01 48
[514480.718322] RSP: 002b:00007fff667b4b98 EFLAGS: 00000246 ORIG_RAX: 0000000000000139
[514480.719053] RAX: ffffffffffffffda RBX: 000055d52ce407c0 RCX: 00007f3ff883ee5d
[514480.719532] RDX: 0000000000000000 RSI: 000055d52b360962 RDI: 0000000000000003
[514480.720019] RBP: 0000000000000000 R08: 0000000000000000 R09: 0000000000000000
[514480.720481] R10: 0000000000000003 R11: 0000000000000246 R12: 000055d52b360962
[514480.720963] R13: 000055d52ce43200 R14: 000055d52b35f550 R15: 000055d52ce408d0
[514480.721451]  </TASK>
[514480.721915] Modules linked in: oops_module(OE+) tls nft_fib_inet nft_fib_ipv4 nft_fib_ipv6 nft_fib nft_reject_inet nf_reject_ipv4 nf_reject_ipv6 nft_reject nft_ct nft_chain_nat nf_nat nf_conntrack nf_defrag_ipv6 nf_defrag_ipv4 rfkill ip_set nf_tables
 nfnetlink sunrpc snd_hda_codec_generic ledtrig_audio snd_hda_intel intel_rapl_msr intel_rapl_common snd_intel_dspcfg snd_intel_sdw_acpi intel_pmc_core intel_vsec pmt_telemetry pmt_class snd_hda_codec kvm_intel snd_hda_core snd_hwdep snd_seq snd_seq_devi
ce snd_pcm kvm iTCO_wdt iTCO_vendor_support virtio_balloon snd_timer irqbypass rapl pcspkr snd lpc_ich i2c_i801 soundcore i2c_smbus joydev xfs libcrc32c virtio_gpu virtio_dma_buf drm_shmem_helper drm_kms_helper crct10dif_pclmul ahci libahci crc32_pclmul
syscopyarea sysfillrect sysimgblt crc32c_intel fb_sys_fops virtio_net net_failover libata drm failover virtio_console ghash_clmulni_intel virtio_scsi virtio_blk serio_raw dm_mirror dm_region_hash dm_log dm_mod fuse
[514480.724632] CR2: 0000000000000000
```

Kernel OOPS log

(Don’t) panic

- Error message
- Oops record   (depending on the error)
- Panic context – CPU, PID, command, kernel info
- Hardware info
- CPU registers’ contents
- Call trace
- Modules linked in

Basic issues

(Don't) panic

- BUG() and WARN()
  - Macros that include a condition and produce log output on 'true'; BUG also panics…
  - Print exact file+line of code
- NULL pointer dereference
- General protection fault

Basic issues

(Don't) panic

Kernel has various *panic* options:

- hung_task_panic

  When a process is in UNinterruptible_sleep longer than threshold

- soft/hard lockup

  When the CPU doesn't reschedule (soft) / process interrupts (hard)

  for longer than *watchdog_thresh* (double for soft lockup)

- RCU stall

- OOM panic

Debugging approaches

At least some basic

- echo ‘h’ > /proc/sysrq-trigger

  Instruct kernel to give you certain information… or panic

- printk()

  Print stuff to kernel log, opt. with specific log-level

  Ex.: *printk(KERN_INFO “My very informative message\n”);*

- kdump+vmcore analysis

  On kernel panic, there’s a possibility to save a memory snapshot,

  which can be later analyzed.

Kexec, kdump, crash

The heavy weight

- kexec

  Mechanism to boot into another kernel.

  [-p] flag can specify a kernel to boot into on panic()

- kdump

  A systemd service that automates kexec setup and further sets up the secondary (panic) kernel to save a memory snapshot (a *vmcore*)

- crash

  A tool to open an analyze kernel vmcores

Kexec, kdump, crash

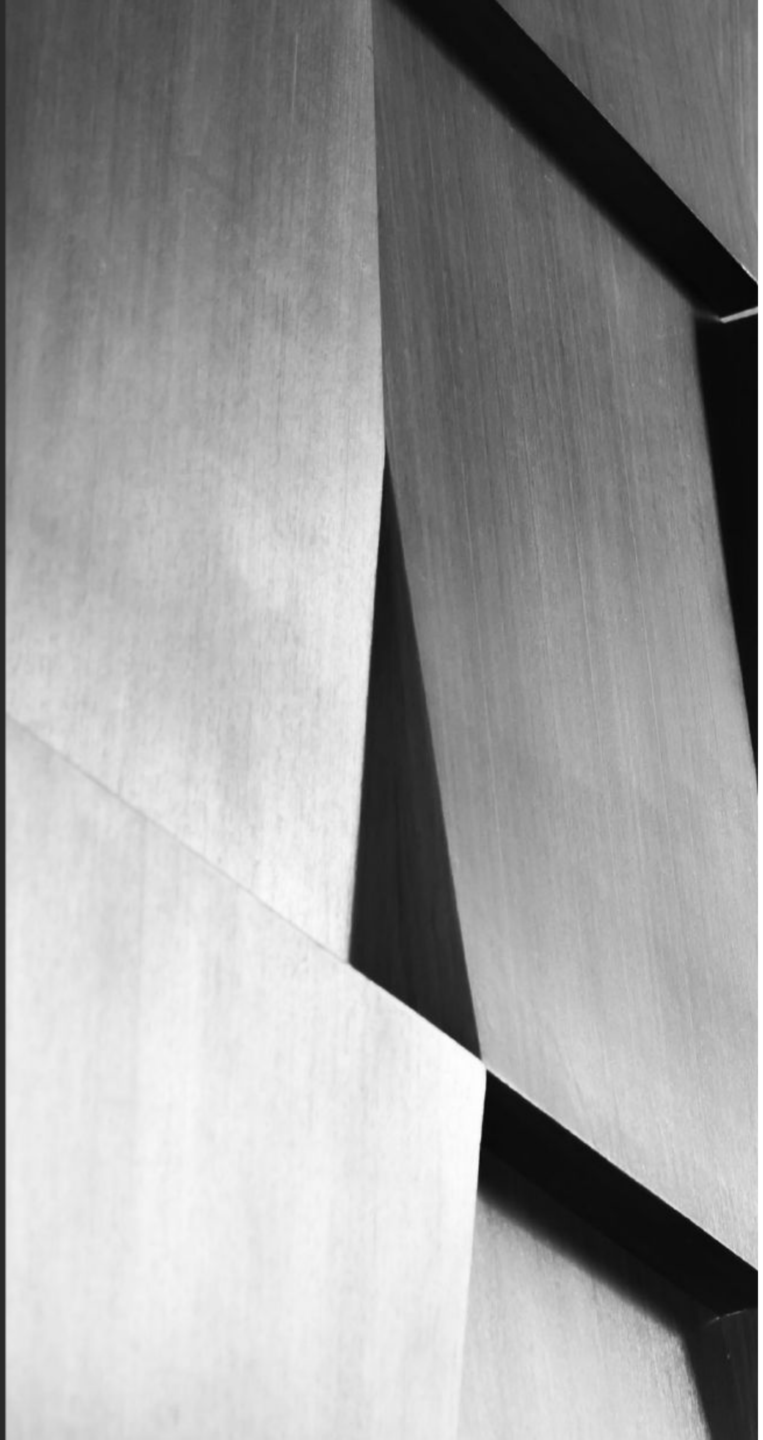
The heavy weight

- /etc/kdump.conf
    - dump target – *device* & relative *path*
    - *core_collector*
- Kernel command line param: *crashkernel=*
- kdump.service
    - kdump initramfs
    - kexec -p
- … Demo

Extra

If there's time left...

- Kernel processes
- VFS - virtual file system
- Control Groups [cgroups]



Thank you

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.

linkedin.com/company/red-hat

youtube.com/user/RedHatVideos

facebook.com/redhatinc

twitter.com/RedHat

Red Hat