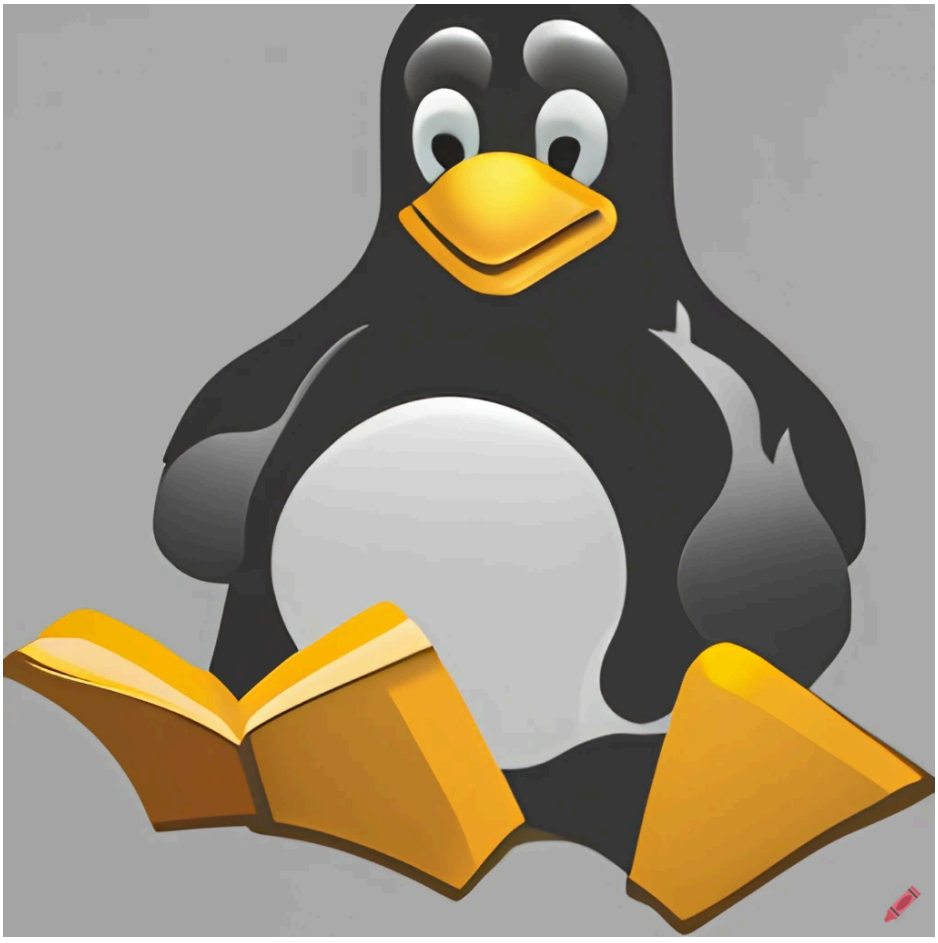
The Linux Process Journey

Version 10.0
October-2025

By Dr. Shlomi Boutnaru



Created using Craiyon, AI Image Generator

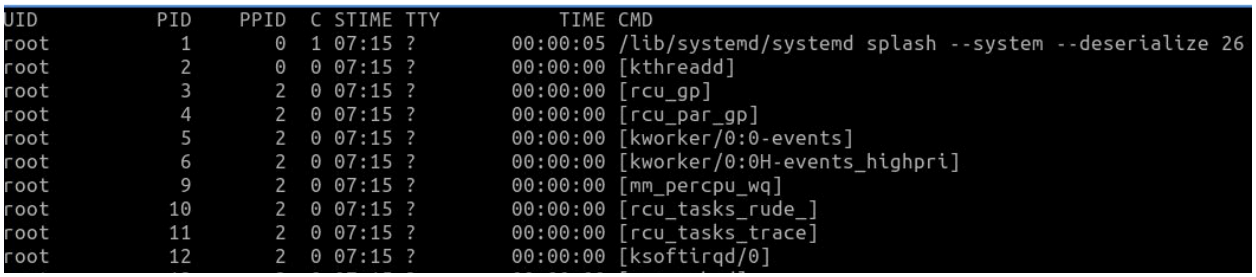
Table of Contents

Introduction

When starting to learn OS internals I believe that we must understand the default processes executing (roles, tasks, etc). Because of that I have decided to write a series of short writeups named "Process ID Card" (aimed at providing the OS vocabulary).

Overall, I wanted to create something that will improve the overall knowledge of Linux in writeups that can be read in 1-3 mins. I hope you are going to enjoy the ride.

In order to create the list of processes I want to explain, I have installed a clean Ubuntu 22.10 VM (Desktop version) and executed ps (as can be seen in the following image - not all the output was included).



```
UID          PID    PPID  C STIME TTY          TIME CMD
root           1       0  1 07:15 ?        00:00:05 /lib/systemd/systemd splash --system --deserialize 26
root           2       0  0 07:15 ?        00:00:00 [kthreadd]
root           3       2  0 07:15 ?        00:00:00 [rcu_gp]
root           4       2  0 07:15 ?        00:00:00 [rcu_par_gp]
root           5       2  0 07:15 ?        00:00:00 [kworker/0:0-events]
root           6       2  0 07:15 ?        00:00:00 [kworker/0:0H-events_highpri]
root           9       2  0 07:15 ?        00:00:00 [mm_percpu_wq]
root          10       2  0 07:15 ?        00:00:00 [rcu_tasks_rude_]
root          11       2  0 07:15 ?        00:00:00 [rcu_tasks_trace]
root          12       2  0 07:15 ?        00:00:00 [ksoftirqd/0]
```

Probably the best way to do it is to go over the processes by the order of their PID value.
The first one I want to talk about is the one we can't see on the list, that is PID 0 (we can see it is the PPID for PID 1 and PID 2 - on them in the next posts).

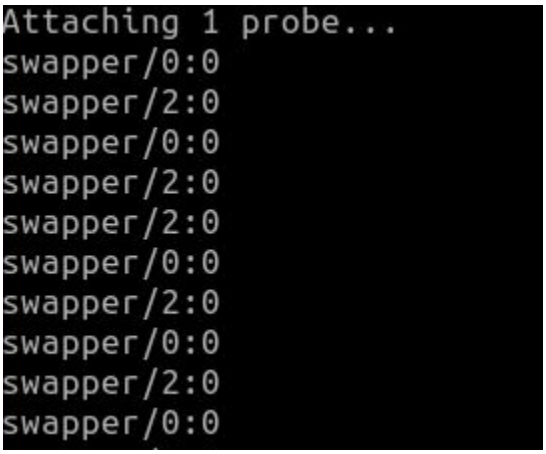
Lastly, you can follow me on twitter - @boutnaru (https://twitter.com/boutnaru). Also, you can read my other writeups on medium - https://medium.com/@boutnaru. Lastly, You can find my free eBooks at https://TheLearningJourneyEbooks.com.

Lets GO!!!!!!

swapper (PID 0)

Historically, old Unix systems used swapping and not demand paging. So, swapper was responsible for the “Swap Process” - moving all pages of a specific process from/to memory/backing store (including related process’ kernel data structures). In the case of Linux PID 0 was used as the “idle process”, simply does not do anything (like nops). It was there so Linux will always have something that a CPU can execute (for cases that a CPU can’t be stopped to save power). By the way, the idle syscall is not supported since kernel 2.3.13 (for more info check out “man 2 idle”). So what is the current purpose of swapper today? helping with pageout ? cache flushes? idling? buffer zeroning? I promise we will answer it in more detail while going through the other processes and explaining the relationship between them.

But how can you believe that swapper (PID 0) even exists? if you can’t see it using ps. I am going to use “bpftrace” for demonstrating that (if you don’t know about bpftrace, I strongly encourage you to read about it). In the demo I am going to trace the kernel function “hrtimer_wakeup” which is responsible for waking up a process and move it to the set of runnable processes. During the trace I am going to print the pid of the calling process (BTW, in the kernel everything is called a task - more on that in future posts) and the executable name (the comm field of the task_struct [/include/linux/sched.h]). Here is the command: sudo bpftrace -e 'kfunc:hrtimer_wakeup { printf("%s:%d\n",curtask->comm,curtask->pid); }'.



From the output we can see we have 3 instances of swapper: swapper/0, swapper/1 and swapper/2 all of them with PID 0. The reason we have three is because my VM has 3 virtual CPUs and there is a swapper process for each one of them - see the output of the command in the following image.

init (PID 1)

After explaining about PID 0, now we are going to talk about PID 1. Mostly known as “init”. init is the first Linux user-mode process created, which runs until the system shuts down. init manages the services (called demons under Linux, more on them in a future post). Also, if we check the process tree of a Linux machine we will find that the root of the tree is init.

There are multiple implementations for init, each of them provide different advantages among them are: SysVinit, launched, systemd, runit, upstart, busybox-init and OpenRC (those are examples only and not a full list). Thus, based on the implementation specific configuration files are read (such as /etc/inittab - SysVinit), different command/tools to manage demons (such as service - SysVinit and systemctl - systemd), and different scripts/profiles might be executed during the boot process (runlevels of SysVinit vs targets in systemd).

The creation of init is done by the kernel function “rest_init”[1]. In the code we can see the call to “user_mode_thread” which spawns init, later in the function there is a call to “kernel_thread” which creates PID 2 (more information about it in the upcoming pages ;-).

Now we will go over a couple of fun facts about init. First, in case a parent process exits before all of its children process, init adopts those child processes. Second, only the signals which have been explicitly installed with a handler can be sent to init. Thus, sending “kill -9 1” won’t do anything in most distributions (try it and see nothing happens). Remember that different init implementations handle signals in different ways.

Because they are multiple init implementations (as we stated before) we can determine the one installed in the following manner. We can perform “ls -l /sbin/init”. If it is not a symlink it is probably SysVinit, else if it points to “/lib/systemd/systmed” than systemd is in use (and of course they are other symlinks to the other implementation - you can read about it in the documentation of each init implementation). As you can see in the attached screenshot Ubuntu 22.10 uses systemd.

[1] https://elixir.bootlin.com/linux/v6.1.8/source/init/main.c#L683

Kernel Threads

Before we will go over kthreadd I have decided to write a short post about kernel threads (due to the fact kthreadd is a kernel thread). We will go over some characteristics of kernel threads. First, kernel threads always execute in Kernel mode and never in User mode. Thus, kernel threads have basically all privileges and have no userspace address associated with them.

Second, both user mode process and kernel threads are represented by a task_struct inside the Linux kernel. As with all other user tasks, kernel threads are also part of the OS scheduling flow and can be executed on any CPU (there are cases in which there is a specific kernel thread for each CPU, we have seen it with swapper in the first post). Third, all kernel threads are descendants of kthreadd - Why is that? We will explain it in the next post focused on kthreadd.

Lastly, let's investigate kernel threads using /proc and see the difference in information retrieved from a regular user process (aka user task). There are multiple file entries in "/proc/pid" that contain information in case of a user mode process but are empty in case of a kernel thread, such as: "maps", "environ", "auxv", "cmdline" (I suggest reading "man proc" to get more info about them). Also, the fd and fdinfo directories are empty and the link "exe" does not point to any executable. In the attached screenshot we can see some of the difference between PID 1 [example of a regular user mode process] and PID 2 [example for a kernel thread]. BTW, the screenshot below was taken from an online/browser based Linux implementation called JSLinux - https://bellard.org/jslinux.

```
localhost:/# uname -a
Linux localhost 4.12.0-rc6-g48ec1f0-dirty #21 Fri Aug 4 21:02:28 CEST 2017 i586
Linux
localhost:/# cat /etc/issue
Welcome to Alpine Linux 3.12
Kernel \r on an \m (\l)

localhost:/# ls -l /proc/1/exe
lrwxrwxrwx    1 root     root             0 Aug 11 23:17 /proc/1/exe -> /bin/bus
ybox
localhost:/# ls -l /proc/2/exe
ls: /proc/2/exe: cannot read link: No such file or directory
lrwxrwxrwx    1 root     root             0 Aug 11 23:16 /proc/2/exe
localhost:/# cat /proc/1/environ
HOME=/TERM=linuxTZ=UTC+07:00localhost:/#
localhost:/# cat /proc/2/environ
```

kthreadd (PID 2)

After explaining about PID 1, now we are going to talk about PID 2.Basically, kthreadd is the “kernel thread daemon". Creation of a new kernel thread is done using kthreadd (We will go over the entire flow). Thus, the PPID of all kernel threads is 2 (checkout ps to verify this). As explained in the post about PID 1 (init) the creation of “kthreadd” is done by the kernel function “rest_init”[2]. There is a call to the function “kernel_thread” (after the creation of init).

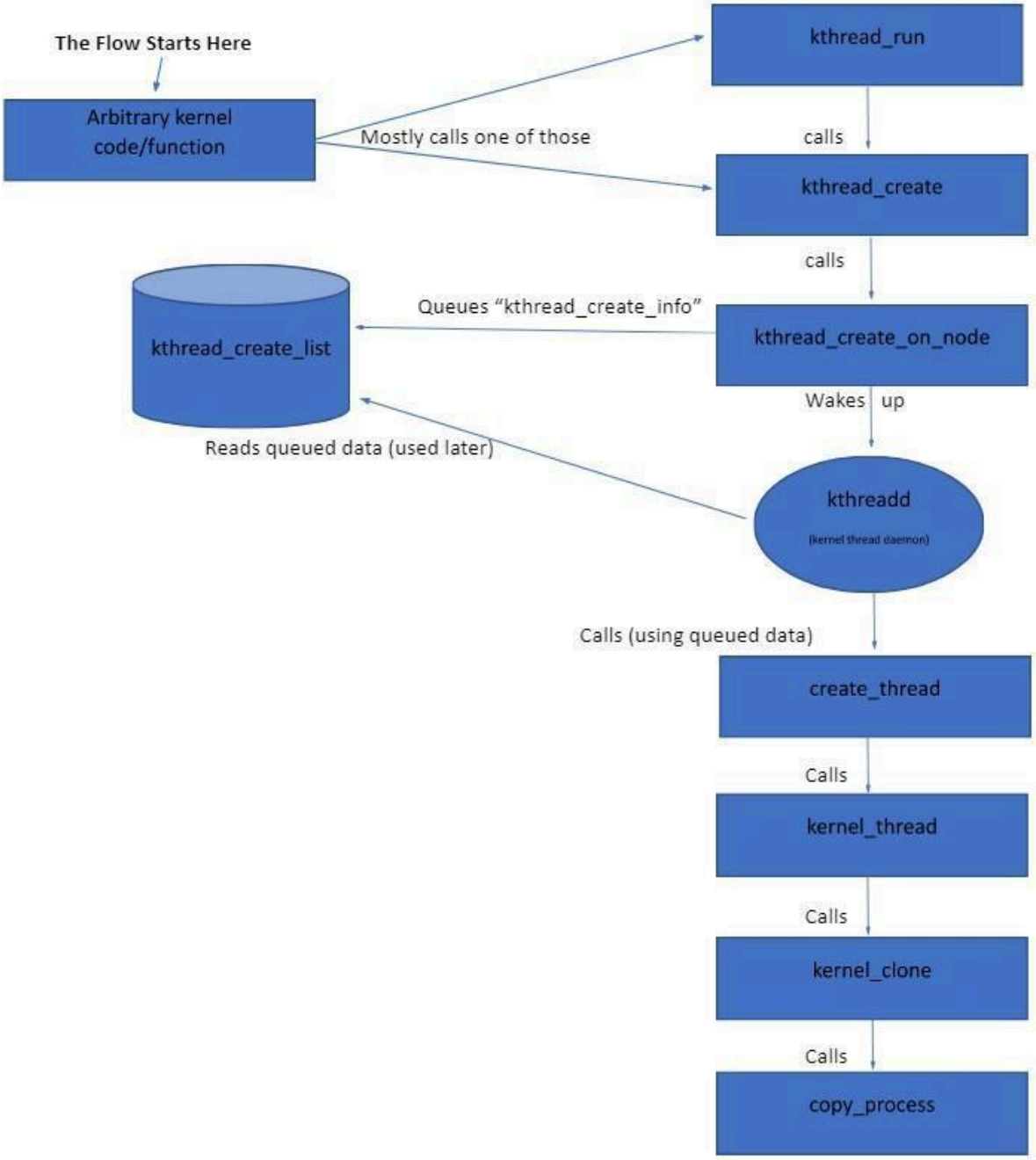
Basically, the kernel uses “kernel threads” (kthreads from now on) in order to run background operations. Thus, it is not surprising that multiple kernel subsystems are leveraging kthreads in order to execute async operations and/or periodic operations. In summary, the goal of kthreadd is to make available an interface in which the kernel can dynamically spawn new kthreads when needed.

Overall, kthreadd continuously runs (infinite loop[3]) and checks “kthread_create_list” for new kthreads to be created. In order to create a kthread the function “kthread_create”[4] is used, which is a helper macro for “kthread_create_on_node”[5]. We can also call “kthread_run”[6] could also be used, it is just a wrapper for “kthread_create”. The arguments passed to the creating function includes: the function to run in the thread, args to the function and a name.

While going over the source code we have seen that “kthread_create” calls “kthread_create_on_node”, which instantiates a “kthread_create_info” structure (based on the args of the function). After that, that structure is queued at the tail of “kthread_create_list” and “kthreadd” is awakened (and it waits until the kthread is created, this is done by “__kthread_create_on_node”[7]). What “kthreadd” does is to call “create_thread” based on the information queued. “create_thread” calls “kernel_thread”, which then calls “kernel_clone”. “kernel_clone” executes “copy_process”, which creates a new process as a copy of an old one - the caller needs to kick-off the created process (or thread in our case). By the way, the flow of creating a new task (recall every process/thread under Linux is called task and represented by “struct task_struct”) from user mode also gets to “copy_process”.

For the sake of simplicity, I have created a flow graph which showcases the flow of creating a kthread, not all the calls are there, only those I thought are important enough. Also, in both cases of macros/functions I used the verb “calls”. The diagram appears at the end of the post. Let me know if it is clear enough or do you think I should change something.

[2] https://elixir.bootlin.com/linux/v6.1.8/source/init/main.c#L683
[3] https://elixir.bootlin.com/linux/v6.1.12/source/kernel/kthread.c#L731
[4] https://elixir.bootlin.com/linux/v6.1.12/source/include/linux/kthread.h#L27
[5] https://elixir.bootlin.com/linux/v6.1.12/source/kernel/kthread.c#L503
[6] https://elixir.bootlin.com/linux/v6.1.12/source/include/linux/kthread.h#L51
[7] https://elixir.bootlin.com/linux/v6.1.12/source/kernel/kthread.c#L414


The Flow Starts Here
Arbitrary kernel code/function
Mostly calls one of those
kthread_run
calls
kthread_create
calls
kthread_create_on_node
Queues "kthread_create_info"
kthread_create_list
Wakes up
Reads queued data (used later)
kthreadd
(kernel thread daemon)
Calls (using queued data)
create_thread
Calls
kernel_thread
Calls
kernel_clone
Calls
copy_process

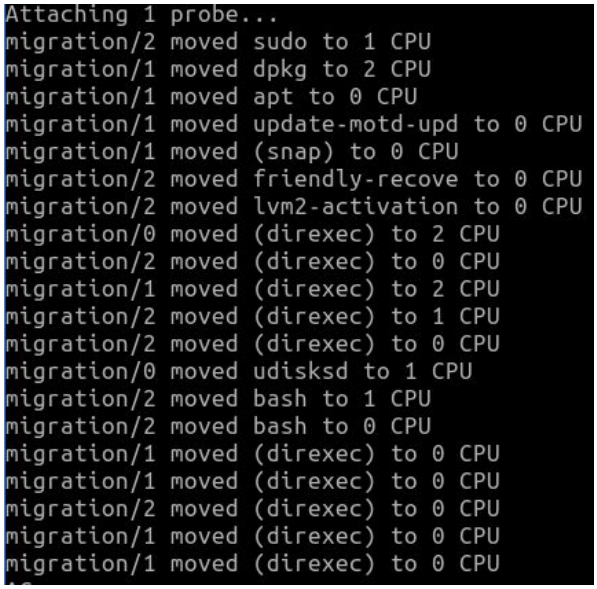
migration

One of the goals of an operating system is to handle and balance resources across the hardware of the compute entity. In order to do that, Linux has a kernel thread named “migration” which has an instance on every vCPU. By the way, the naming format is “migration/N” where N is the id of the vCPU.

By default threads are not constrained to a vCPU and can be migrated between them in the next call to “schedule()” (which calls the main scheduler function, which is “__scheduler()”[8]). It is done mainly in case the scheduler identifies an unbalanced across the runqueues (the queue in which processes which are in ready/runnable state are waiting to use the processor) of the vCPUs.

It is important to state that we can influence this flow by setting the affinity of a thread (for more read “man 2 sched_setaffinity”. We will talk about that in a future post). There could be performance, cache and other impacts for doing that (but that is also a topic for a different writeup).

I have created a small demo which shows the working of “migration”. For that I have created a VM running Ubuntu 22.04 with 3 vCPUs. In order to trace the usage of “move_queue_task” I have used bpftrace with the following command: **sudo bpftrace -e 'kfunc:move_queued_task { printf("%s moved %s to %d CPU\n",curtask->comm,args->p->comm,args->new_cpu); }'**. The output of the command is shown below. The one-liner prints: the name of the task calling “move_queued_task”, the name of the task which is moved and id the vCPU which the task is moved to.



[8] https://elixir.bootlin.com/linux/latest/source/kernel/sched/core.c#L6544

In summary, what the kernel thread “migration” does is to move threads from highly loaded vCPUs to others which are less crowded (by inserting them to a different run-queue). A function which is used by “migration” in order to move a task to a new run-queue is “move_queued_task”[9].

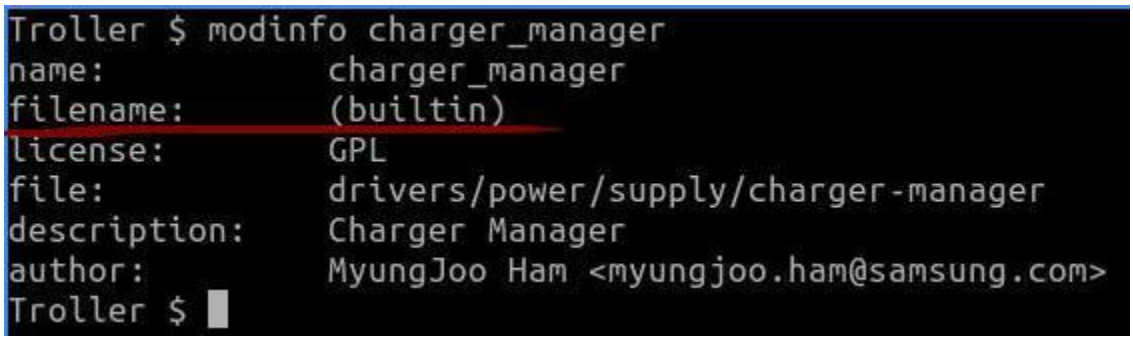
[9] https://elixir.bootlin.com/linux/latest/source/kernel/sched/core.c#L2325

charger_manager

The “charger_manager” kernel thread is created by a freezable workqueue[10]. Freezable workqueues are basically frozen when the system is moved to a suspend state[11]. Based on the kernel source code “charger_manager” is responsible for monitoring the health (like temperature monitoring) of the battery and controlling the charger while the system is suspended to memory[12]. The “Charger Manager” kernel module is written by MyungJoo Ham[13].

Moreover, the kernel documentation states that the “Charger Manager" also helps in giving an aggregated view to user-space in case there are multiple chargers for a battery. In case they are multiple batteries with different chargers on a system, that system would need multiple instances of “Charger Manager”[14] .

On my Ubuntu VM (22.04.1 LTS) this kernel module is not compiled as a separate “*.ko” file. It is compiled into the kernel itself (builtin), as you can see in the output of “modinfo” in the screenshot below.



[10] https://elixir.bootlin.com/linux/latest/source/drivers/power/supply/charger-manager.c#L1749
[11] https://lwn.net/Articles/403891/
[12] https://elixir.bootlin.com/linux/latest/source/drivers/power/supply/charger-manager.c
[13] https://elixir.bootlin.com/linux/latest/source/drivers/power/supply/charger-manager.c#L1768
[14] https://www.kernel.org/doc/html/v5.3/power/charger-manager.html

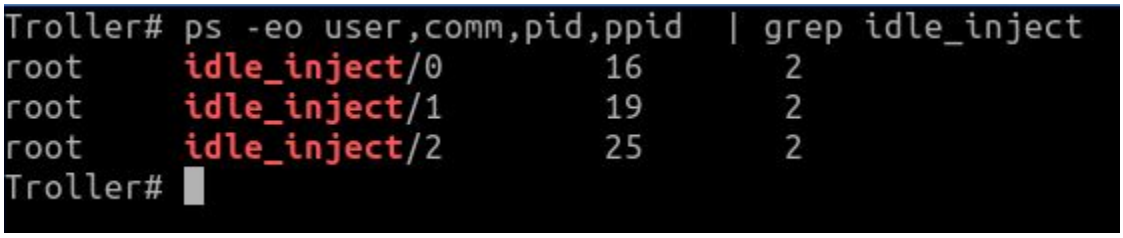
idle_inject

On our plate this time we are going to talk about the kernel thread “idle_inject”, which was merged to the kernel in about 2009. The goal of “idle_inject” is forcing idle time on a CPU in order to avoid overheating.

If we think about it, “idle_inject” adds latency, thus it should be considered only if CPUFreq (CPU Frequency scaling) is not supported. Due to the fact the majority of modern CPUs are capable of running a different clock frequency and voltage configuration we can use CPUFreq in order to avoid overheating.

Overall, there is one “idle_inject” kernel thread per processor (with the name pattern “idle_inject/N”, where N is the id of the processor) - as shown in the screenshot below. Also, all of them are created at init time.

The “idle_inject” kernel threads will call “idle_inject_fn()”->”play_idle_precise()” to inject a specified amount of idle time. After all of the kernel threads are woken up, the OS sets a timer for the next cycle. When the timer interrupt handler wakes the threads for all processors based on a defined “cpu-mask” (affected by idle injection). By the way, when I set a kprobe on “idle_inject_fn()” for 3 hours on my VM it was never called ;-)



kworker (Kernel Thread Worker)

A kworker is a kernel thread that performs processing as part of the kernel, especially in the case of interrupts, timers, I/O, etc. It is based on workqueues which are async execution mechanisms, that execute in “process context” (I will post on workqueus in more details separately, for now it is all that you need to know).

Overall, there are a couple of kworkers running on a Linux machine. The naming pattern of kworkers includes: the number of the core on which it is executed, the id of the thread and can contain also string that hints what the kworker does (check the output of ‘ps -ef | grep kworker’).

```
    6       2  0 07:15 ?        00:00:00 [kworker/0:0H-events_highpri]
   82       2  0 07:15 ?        00:00:02 [kworker/0:1H-kblockd]
  113       2  0 07:15 ?        00:00:00 [kworker/u3:0]
46277       2  0 11:11 ?        00:00:00 [kworker/u2:1-events_unbound]
46547       2  0 11:20 ?        00:00:01 [kworker/0:1-events]
46624       2  0 11:23 ?        00:00:00 [kworker/u2:2-kcryptd/253:0]
46867       2  0 11:28 ?        00:00:00 [kworker/0:0-inet_frag_wq]
47091       2  0 11:33 ?        00:00:00 [kworker/u2:0-events_unbound]
47299       2  0 11:36 ?        00:00:00 [kworker/0:2-events]
```

The big question is - “How do we know what each kwoker is doing?”. It’s a great question, the way in which we are going to answer it is by using ftrace (function tracing inside the kernel[15]). The command we are going to use are:

echo workqueue:workqueue_queue_work > /sys/kernel/debug/tracing/set_event
cat /sys/kernel/debug/tracing/trace_pipe > /tmp/trace.log

The first one enables the tracing regarding workqueus. The second reads the tracing data and saves it to a file. We can also run “cat /sys/kernel/debug/tracing/trace_pipe | grep kworker” and change the grep filter to a specific kworker process. In the trace we will see the function name that each kworker thread is going to execute.

```
   kworker/u2:2-46624   [000] d.... 17855.481276: workqueue_queue_work: work struct=00000000da1e6721 function=flush_to_ldisc
workqueue=events_unbound req_cpu=8192 cpu=4294967295
   kworker/u2:1-48183   [000] d.... 17855.525798: workqueue_queue_work: work struct=00000000be96cc25 function=ata_sff_pio_ta
k workqueue=ata_sff req_cpu=8192 cpu=0
   kworker/u2:1-48183   [000] d.... 17856.038232: workqueue_queue_work: work struct=000000001e1ee94f function=kcryptd_crypt
dm_crypt] workqueue=kcryptd/253:0 req_cpu=8192 cpu=4294967295
   kworker/u2:1-48183   [000] d.... 17857.542509: workqueue_queue_work: work struct=00000000be96cc25 function=ata_sff_pio_ta
k workqueue=ata_sff req_cpu=8192 cpu=0
   kworker/u2:1-48183   [000] d.... 17859.558293: workqueue_queue_work: work struct=00000000be96cc25 function=ata_sff_pio_ta
k workqueue=ata_sff req_cpu=8192 cpu=0
   kworker/u2:1-48183   [000] d.... 17860.134032: workqueue_queue_work: work struct=000000001e1ee94f function=kcryptd_crypt
dm_crypt] workqueue=kcryptd/253:0 req_cpu=8192 cpu=4294967295
   kworker/u2:1-48183   [000] d.... 17860.134074: workqueue_queue_work: work struct=00000000e0b6b12c function=kcryptd_crypt
dm_crypt] workqueue=kcryptd/253:0 req_cpu=8192 cpu=4294967295
```

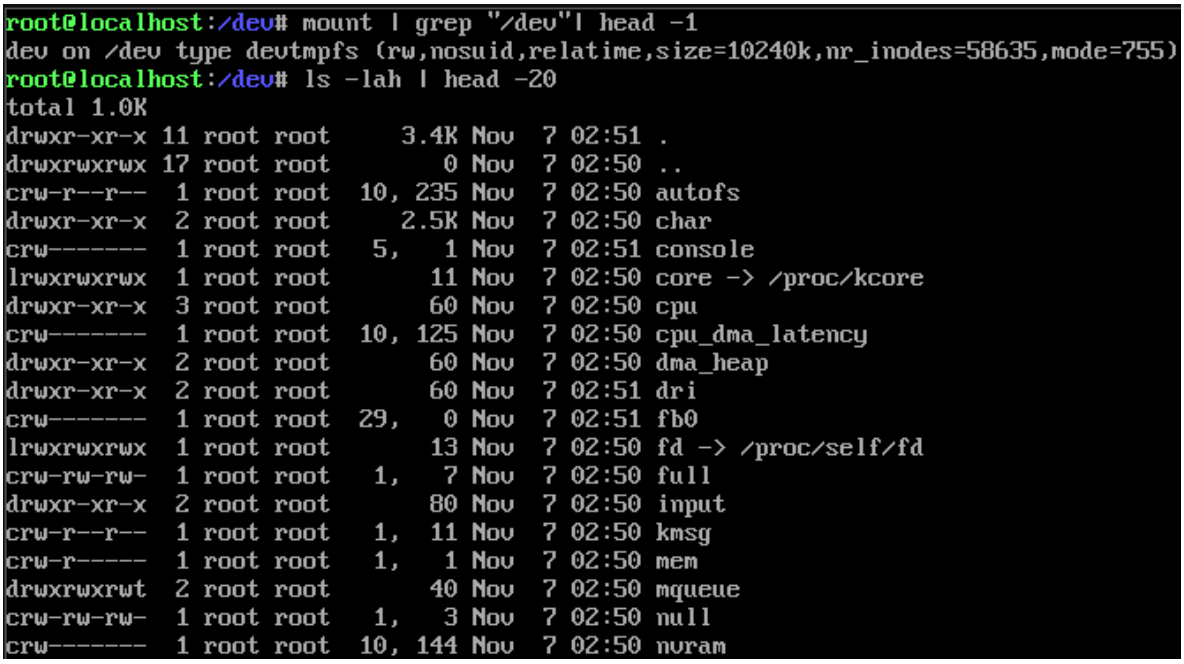
[15] https://www.kernel.org/doc/Documentation/trace/ftrace.txt

kdevtmpfs

“kdevtmpfs” is a kernel thread which was created using the “kthread_run” function[16]. “kdevtmpfs” creates a devtmpfs which is a tmpfs-based filesystem (/dev). The filesystem is created during bootup of the system, before any driver code is registered. In case a driver-core requests a device node it will result in a node added to this filesystem[17].

We can see the specific line of code that is used in order to create the mounting point “/dev”[18]. The mountpoint is created using the function “init_mount”[19]. A nice fact is that it is part of “init_*” functions which are routines that mimic syscalls but don’t use file descriptors or the user address space. They are commonly used by early init code[20].

Thus, we can say the “kdevtmpfs” is responsible for managing the “Linux Device Tree”. Also, by default the name created for nodes under the filesystem is based on the device name (and owned by root) - as shown in the screenshot below (taken from copy.sh based Linux). By the way, not all devices have a node in “/dev” think about network devices ;-)



[16] https://elixir.bootlin.com/linux/v6.2-rc1/source/drivers/base/devtmpfs.c#L474
[17] https://elixir.bootlin.com/linux/v6.2-rc1/source/drivers/base/devtmpfs.c#L3
[18] https://elixir.bootlin.com/linux/v6.2-rc1/source/drivers/base/devtmpfs.c#L377
[19] https://elixir.bootlin.com/linux/v6.2-rc1/source/fs/init.c#L16
[20] https://elixir.bootlin.com/linux/v6.2-rc1/source/fs/init.c#L3

cpuhp (CPU Hotplug)

This kernel thread is part of the CPU hotplug support. It enables physically removing/adding CPUs on a specific system. There is one kernel thread per vCPU, and the pattern of the thread's name is "cpuhp/N" (where N is the id of the vCPU) - as can be seen in the screenshot below. Also, today the CPU hotplug can be used to resume/suspend support for SMP (Symmetric Multiprocessing).

If we want our kernel to support CPU hotplug the CONFIG_HOTPLUG_CPU should be enabled (it's supported on a couple of architectures such as: MIPS, ARM, x86 and PowerPC). The kernel holds the current state for each CPU by leveraging "struct cpuhp_cpu_state"[21].

We can configure the CPU hotplug mechanism using sysfs (/sys/devices/system/cpu). For example we can shut down and bring up a CPU by writing "0" and "1" respectively to the "online" file in the directory representing the CPU (for which we want to change the status) - checkout the screenshot below (the Linux VM I am testing on has 3 vCPUs).

In order to bring the CPU down the function "cpu_device_down"[22] is called. In order to bring up a CPU function "cpu_device_up"[23] is called.

```
Troller # pwd
/sys/devices/system/cpu
Troller # ls
cpu0  cpufreq  isolated    offline   power    uevent
cpu1  cpuidle  kernel_max  online    present  vulnerabilities
cpu2  hotplug  modalias    possible  smt
Troller # echo 0 > ./cpu2/online
Troller # dmesg | tail -2
[147586.057954] kvm-clock: cpu 1, msr b7001041, secondary cpu clock
[148846.125346] smpboot: CPU 2 is now offline
Troller # echo 1 > ./cpu2/online
Troller # dmesg | tail -2
[148846.125346] smpboot: CPU 2 is now offline
[148874.835266] smpboot: Booting Node 0 Processor 2 APIC 0x2
```

[21] https://elixir.bootlin.com/linux/latest/source/kernel/cpu.c#L65
[22] https://elixir.bootlin.com/linux/latest/source/kernel/cpu.c#L1225
[23] https://elixir.bootlin.com/linux/latest/source/kernel/cpu.c#L1439

khungtaskd (Kernel Hang Task Daemon)

This kernel thread “khungtaskd” is used in order to help with identifying and debugging “Hung Tasks”. This kernel thread is scheduled every 120 seconds (that is the default value). We can say “khungtaskd” is used for detecting tasks which are stuck in uninterruptible sleep (state “D” in ps output). We can also go over the code of the kernel thread as part of the Linux kernel source code[24].

The basic algorithm of “khungtaskd” is as follows: Iterate over all running tasks on the system and if there are ones marked as TASK_UNINTERRUPTIBLE and it was scheduled at least once in the last 120 seconds it is considered as hung. When a task is considered hung it’s “call stack” is dumped and if the CONFIG_LOCKDEP is also enabled then all of the locks held by the tasks are outpted also.

If we want we can change the sampling interval using the sysctl interface, “/proc/sys/kernel/hung_task_timeout_secs” .We can also verify that the default is 120 seconds by reading it - as shown in the screenshot below.

In order to demonstrate the operation of “khungtaskd” I have executed the following bpftrace one liner - “sudo bpftrace -e 'kfunc:check_hung_uninterruptible_tasks { printf(" %s:%d\n",curtask->comm,curtask->pid); }'”. The trace prints the name of the task and it’s pid when the function “check_hung_uninterruptible_tasks” is called[25] - You can see the output in the screenshot below.

```
Troller $ sudo cat /proc/sys/kernel/hung_task_timeout_secs
120
Troller $ sudo bpftrace -e 'kfunc:check_hung_uninterruptible_tasks { printf(" %s:%d\n",curtask->comm,curtask->pid); }'
Attaching 1 probe...
 khungtaskd:34
 khungtaskd:34
 khungtaskd:34
 khungtaskd:34
 khungtaskd:34
 khungtaskd:34
 khungtaskd:34
 khungtaskd:34
 khungtaskd:34
 khungtaskd:34
 khungtaskd:34
 khungtaskd:34
 khungtaskd:34
 khungtaskd:34
 khungtaskd:34
 khungtaskd:34
 khungtaskd:34
 khungtaskd:34
 khungtaskd:34
 khungtaskd:34
```

[24] https://elixir.bootlin.com/linux/latest/source/kernel/hung_task.c
[25] https://elixir.bootlin.com/linux/latest/source/kernel/hung_task.c#L178

kswapd

The kernel thread “kswapd” is the background page-out daemon of Linux (swaps processes to disk). You can see the creation of the kernel thread in the source of the kernel - https://elixir.bootlin.com/linux/latest/source/mm/vmscan.c#L4642. In the code we can see that a dedicated instance of “kswapd” is created for each NUMA zone (on my Ubuntu 22.10 VM I have only “kswapd0” - as shown in the screenshot below).

Overall, the goal of the “kswapd” is to reclaim pages when memory is running low. In the old days, the “kswapd” was woken every 10 seconds but today it is only wakened by the page allocator, by calling “wakeup_kswapd”[26]. The code of the page allocator is located at “mm/page_alloc.c”[27].

Basically, “kswapd” trickles out pages so the system has some free memory even if no other activity frees up anything (like by shrinking cache). Think about cases in which operations work in asynchronous contexts that cannot page things out.

The major function which is called by “kswapd” is “balance_pgdat()”[28]. In order to see that process happening we can use the following bpftrace one-liner: “**sudo bpftrace -e 'kfunc:balance_pgdat { printf("%s:%d\n",curtask->comm,curtask->pid); }**” - You can see “kswapd0” calling it in the screenshot below. The flow of “kswapd” is based on limits, when to start shirking and “until when” to shrink (low and high limits).

[26] https://elixir.bootlin.com/linux/latest/source/mm/vmscan.c#L4555
[27] https://elixir.bootlin.com/linux/latest/source/mm/page_alloc.c
[28] https://elixir.bootlin.com/linux/latest/source/mm/vmscan.c#L4146

kcompactd

When a Linux system is up and running, memory pages of different processes/tasks are scattered and thus are not physically-contiguous (even if they are contiguous in their virtual address). We can move to bigger pages size (like from 4K to 4M) but it still has its limitations like: waste of space in case of regions with small sizes and the need for multiple pages in case of large regions that can still be fragmented. Due to that, the need for memory compaction was born[29].

"kcompatd" is performing in the background the memory compaction flow. The goal of memory compaction is to reduce external fragmentation. This procedure is heavily dependent on page migration[30] to do all the heavy lifting[31]. In order for "kcompactd" to work we should compile the kernel with "CONFIG_COMPACTION" enabled. Also, when a Linux system identifies that it is tight low in available memory the "kcompactd" won't perform memory compaction memory[32].

Overall, the "kcompactd" kernel thread is created in "kcompactd_run" function[33] which is called by "kcompactd_init"[34].. The function "kcompactd_init" is started by "subsys_initcall"[35], which is responsible for initializing a subsystem.

The kernel thread starts the function "static int kcompactd(void *p)"[36].. An instance of the kernel thread is created for each node (like vCPU) on the system[37].. The pattern of the kernel thread name is "kcompactd[IndexOfNode]" for example "kcompactd0" as we can see in the screenshot below.

"kcompactd" can be called in one of two ways: woken up or by using a timeout. It can be woken up by kswapd[38].. Also, we can configure it using modification of the filesystem ("/proc/sys/vm/compact_memroy" for example). By the way, in the memory compaction flow of the function "compact_zone"[39] is executed in the context of "kcompactd". In order to demonstrate that we can use the following one-liner using bpftrace: **sudo bpftrace -e 'kfunc:compact_zone { printf("%s:%d\n",curtask->comm,curtask->pid); }'** - The output can be seen in the screenshot below.

[29] https://lwn.net/Articles/368869/
[30] https://lwn.net/Articles/157066/
[31] https://elixir.bootlin.com/linux/v6.2-rc3/source/mm/compaction.c#L5
[32] https://www.linux-magazine.com/Issues/2015/179/Kernel-News
[33] https://elixir.bootlin.com/linux/v6.2-rc3/source/mm/compaction.c#L2996
[34] https://elixir.bootlin.com/linux/v6.2-rc3/source/mm/compaction.c#L3048
[35] https://elixir.bootlin.com/linux/v6.2-rc3/source/mm/compaction.c#L3065
[36] https://elixir.bootlin.com/linux/v6.2-rc3/source/mm/compaction.c#L2921
[37] https://elixir.bootlin.com/linux/v6.2-rc3/source/mm/compaction.c#L3061
[38] https://www.slideshare.net/AdrianHuang/memory-compaction-in-linux-kernelpdf
[39] https://elixir.bootlin.com/linux/v6.2-rc3/source/mm/compaction.c#L2289

```
Troller # ps -ef | grep -v grep | grep kcompactd
root          37       2  0 00:15 ?        00:00:09 [kcompactd0]
Troller # ls -l /proc/sys/vm/compact_memory
--w------- 1 root root 0 Jan 14 11:54 /proc/sys/vm/compact_memory
Troller # sudo bpftrace -e 'kfunc:compact_zone { printf("%s:%d\n",curtask->comm,curtask->pid); }'
Attaching 1 probe...
kcompactd0:37
kcompactd0:37
kcompactd0:37
```

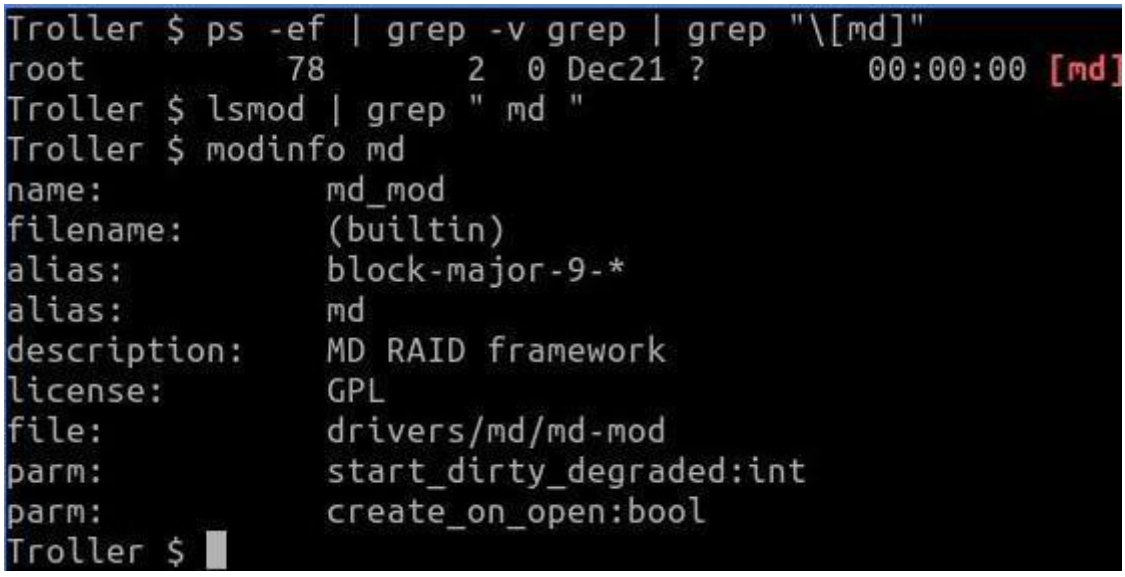
md (Multiple Device Driver)

“md” is a kernel thread which is based on a workqueue[40]. It is responsible for managing the Linux md (multiple device) driver which is also known as the “Linux software RAID”. RAID devices are virtual devices (created from two or more real block devices). This allows multiple devices (typically disk drives or partitions thereof) to be combined into a single device to hold (for example) a single filesystem[41].

By using the “md” driver we can create from one/more physical devices (like disk drivers) a virtual device(s). By the use of an array of devices we can achieve redundancy, which is also known as RAID (Redundant Array of Independent Disks)[42].

Overall, “md” supports different RAID types: RAID 1 (mirroring), RAID 4, RAID 5, RAID 6 and RAID 10[43]. Besides that, “md” also supports pseudo RAID technologies like: RAID 0, LINAR, MULTIPATH and FAULTY[44].

The code of “md” is included as a driver/kernel module in the source code of Linux. Thus, it can be compiled directly into the kernel or as a separate “*.ko” file. In my VM (Ubuntu 22.04) it is compiled directly into the kernel image as shown in the screenshot below.



[40] https://elixir.bootlin.com/linux/v6.1/source/drivers/md/md.c#L9615
[41] https://linux.die.net/man/8/mdadm
[42] https://man7.org/linux/man-pages/man4/md.4.html
[43] https://www.prepressure.com/library/technology/raid
[44] https://doxfer.webmin.com/Webmin/Linux_RAID

The block devices that can be used in order to access the software RAID on Linux are in the pattern “/dev/mdN” (where N is a number [0–255])[45]. It can also be configured to allow access using “/dev/md/N” or “/dev/md/name”. If we want information about the current state of “md” we can query the file “/proc/mdstat”[46]. There is also the command line utility “mdadm” that can help with managing those devices[47].

Lastly, the init function is declared using “subsys_initcall” (and not the “module_init”) which ensures that it will run before the device drivers that needs it (if they are using “module_init”)[48]. More information about initcalls will be included on a future writeup.

[45] https://www.oreilly.com/library/view/managing-raid-on/9780596802035/ch01s03.html
[46] https://raid.wiki.kernel.org/index.php/Mdstat
[47] https://linux.die.net/man/8/mdadm
[48] https://elixir.bootlin.com/linux/v6.1/source/drivers/md/md.c#L9947

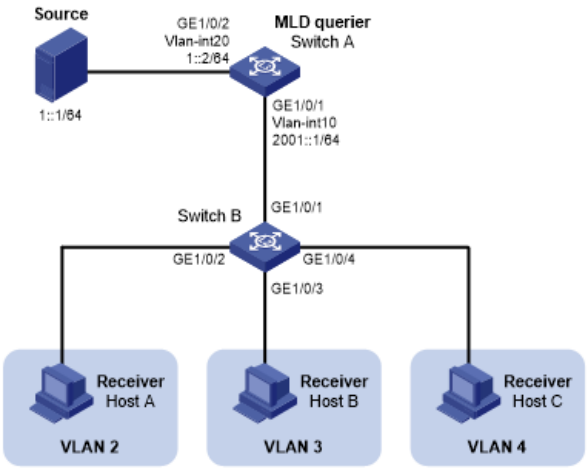
mld (Multicast Listener Discovery)

“mld” is a kernel thread which was created using a workqueue[49]. It is the Linux implementation for the multicast listener (MLD) protocol. This protocol is used by IPv6 based routers in order to discover multicast listeners on the local network and identify which multicast addresses are of interest to those listeners. MLD is supported on different operating systems such as Windows[50] and Linux[51].

We can think about it like IGMP[52] which is used on IPv4 based networks (MLDv1 is derived from IGMPv2 and MLDv2 is similar to IGMPv3). One important difference is that MLD uses ICMPv6 message types, rather than IGMP message types[53].

Overall, MLD has three major message types: “Multicast Listener Query”, “Multicast Listener Report” and “Multicast Done”[54]. For more information about them I suggest reading the following link. Also, a more detailed explanation about the different MLD operations can be found in https://ipcisco.com/lesson/mld-operations/.

What “mld” does is to send MLD report messages[55] which are sent by an MLD host (see the diagram below[56]) and processes messages[57]. From the source code we can see that there are definitions for structs representing both MLDv1 and MLDv2 headers.



[49] https://elixir.bootlin.com/linux/latest/source/net/ipv6/mcast.c#L3185
[50] https://learn.microsoft.com/en-us/windows/win32/winsock/igmp-and-windows-sockets
[51] https://lwn.net/Articles/29489/
[52] https://www.cloudflare.com/learning/network-layer/what-is-igmp/
[53] https://www.ibm.com/docs/en/zos/2.2.0?topic=protocol-multicast-listener-discovery
[54] https://community.cisco.com/t5/networking-knowledge-base/multicast-listener-discovery-mld/ta-p/3112082
[55] https://elixir.bootlin.com/linux/latest/source/net/ipv6/mcast.c#L3185
[56] https://techhub.hpe.com/eginfolib/networking/docs/switches/5130ei/5200-3944_ip-multi_cg/content/images/image33.png
[57] https://elixir.bootlin.com/linux/latest/source/net/ipv6/mcast.c#L1359

ksmd (Kernel Same Page Merging)

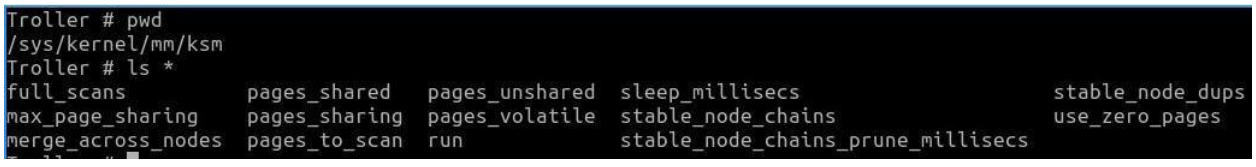
The kernel thread “ksm” is also known as “Kernel Same Page Merging” (and “ksmd” is ksm demon). It is used by the KVM hypervisor to share identical memory pages (supported since kernel 2.6.32) Those shared pages could be common libraries or even user data which is identical. By doing so KVM (Kernel-based Virtual Machine) can avoid memory duplication and enable more VMs to run on a single node.

In order for “ksmd” to save memory due to de-duplication we should compile the kernel with “CONFIG_KSM=y”. It is important to understand that the sharing of identical pages is done even if they are not shared by fork(). We can also go over the source code of “ksmd” source code as part of the Linux kernel[58].

The way “ksmd” works is as follows. Scanning main memory for frames (“physical pages”) holding identical data and collectes the virtual memory address that they are mapped. “ksmd” leaves one of those frames and remaps each duplicate one to point to the same frame. Lastly, “ksmd” frees the other frames. All of the merge pages are marked as COW (Copy-on-Write) for cases in which one of the processes using them will want to write to the page. There is a concern that even if the memory usage is reduced the CPU usage is increased.

The kernel thread “ksmd” is created using the function kthread_run[59]. We can see from the code that the function which is the entry point of the thread is “ksm_scan_therad()” which is calling “ksm_do_scan()” which is the ksm’s scanner main worker function (it gets as input the number of pages to scan before returning). “ksmd” only merges anonymous private pages and not pagecache. Historically, the merged pages were pinned into kernel memory. Today they can be swapped like any other pages.

“ksmd” can be controlled by a sysfs interface (“/sys/kernel/mm/ksm”) - as can be seen in the screenshot below. One of the files exported by sysfs is “run” that can react to one of the following values 0/1/2. “0” means stop “ksmd” from running but keep the merged pages. “1” means run “ksmd”. “2” means stop “ksmd” from running and unmerge all currently merge pages (however leave the mergeable areas registered for next time).



[58] https://elixir.bootlin.com/linux/v6.13.7/source/mm/ksm.c
[59] https://elixir.bootlin.com/linux/v6.0/source/mm/ksm.c#L3188

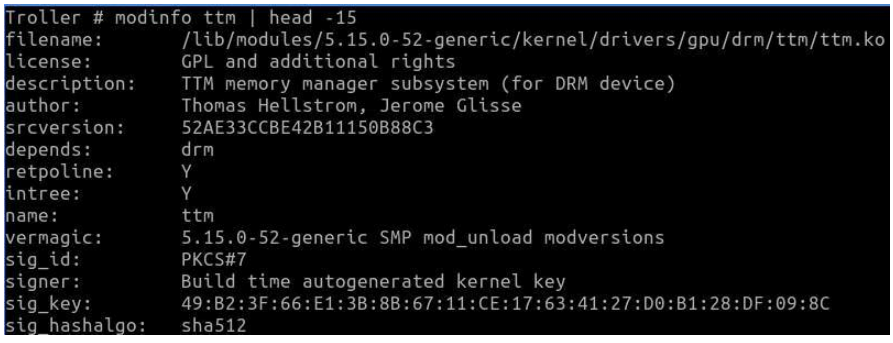
ttm_swap

The kernel thread “ttm_swap” is responsible for swapping GPU’s (Graphical Processing Unit) memory. Overall, TTM (Translation-Table Maps) is a memory manager that is used to accelerate devices with dedicated memory. Basically, all the resources are grouped together by objects of buffers in different sizes. TTM then handles the lifetime, the movements and the CPU mapping of those objects[60].

Based on the kernel documentation, each DRM (Direct Rendering Manager) driver needs a memory manager. There are two memory managers supported by DRM: TTM and GEM (Graphics Execution Manager)[61].

Moreover, “ttm_swap” is a single threaded workqueue as seen in the Linux source code[62].
Also, the man pages describe TTM as a generic memory-manager provided by the kernel, which does not provide a user-space interface (API). In case we want to use it you should checkout the interface of each driver[63].

TTM is at the end a kernel module, you can find the source code and the Makefile in the kernel source tree[64]. Based on the module source code it is written by Thomas Hellstrom and Jerome Glisse[65]. Also, it is described as “TTM memory manager subsystem (for DRM device)”[66]. As you can see it is part of the “drivers/gpu/drm” subdirectory, which holds the code and Makefile of the drm device driver, which provides support for DRI (Direct Rendering Infrastructure) in XFee86 4.1.0+. Lastly, on my VM (Ubuntu 22.04.01) it is compiled as a separate “*.ko” file (/lib/modules/[KernelVersion]/kernel/drivers/gpu/drm/ttm.ko) - as shown in the screenshot below.



[60] https://docs.kernel.org/gpu/drm-mm.html
[61] https://docs.kernel.org/gpu/drm-internals.html
[62] https://elixir.bootlin.com/linux/v5.12.19/source/drivers/gpu/drm/ttm/ttm_memory.c#L424
[63] https://www.systutorials.com/docs/linux/man/7-drm-ttm/
[64] https://elixir.bootlin.com/linux/v6.1-rc2/source/drivers/gpu/drm/ttm
[65] https://elixir.bootlin.com/linux/v6.1-rc2/source/drivers/gpu/drm/ttm/ttm_module.c#L89
[66] https://elixir.bootlin.com/linux/v6.1-rc2/source/drivers/gpu/drm/ttm/ttm_module.c#L89

watchdogd (Watchdog Daemon)

This kernel thread “watchdogd” is used in order to let the kernel know that a serious problem has occurred so the kernel can restart the system. It is sometimes called COP (Computer Operating Properly). The way it is implemented is by opening “/dev/watchdog”, then writing at least once a minute. Every time there is a write the restart of the system is delayed.

In case of inactivity for a minute the watchdog should restart the system. Due to the fact we are not talking about a hardware watchdog the compilation of the operation depends on the state of the machine. You should know that the watchdog implementation could be software only (there are cases in which it won’t restart the machine due to failure) or using a driver/module in case of hardware support[67].

If we are talking about hardware support then the watchdog module is specific for a chip or a device hardware. It is most relevant to systems that need the ability to restart themself without any human intervention (as opposed to a PC we can reboot easily) - think about an unmanned aircraft. We need to be careful because a problem in the watchdog configuration can lead to unpredictable reboot, reboot loops and even file corruption due to hard restart[68].

The relationship between the hardware and software is as follows: the hardware is responsible to set up the timer and the software is responsible to reset the timer[69]. When the timer gets to a specific value (configured ahead) and it is not elapsed by the software the hardware will restart the system[70].

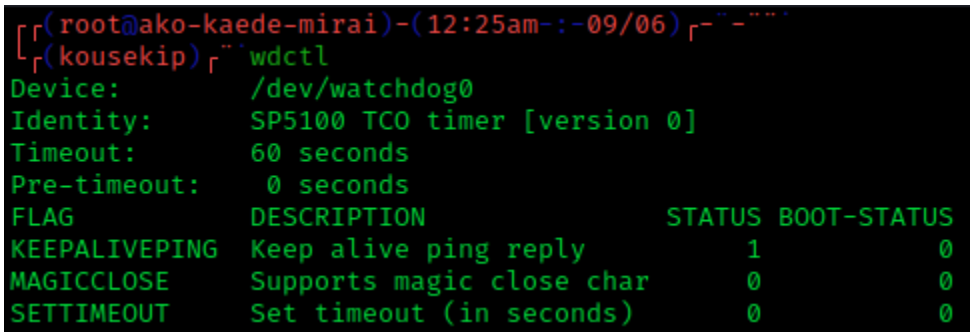
The software part is being conducted by the “watchdogd” (the software watchdog daemon) which opens “/dev/watchdog” and writes to it in order to postpone the restart of the system by the hardware[71]. Examples for different watchdog drives/modules for specific chips can be found in the source tree of Linux kernel[72]. Some example are apple_wdt (Apple’s SOC), ath79_wdt (Atheros AR71XX/AR724X/AR913X) and w83977f_wdt (Winbond W83977F I/O Chip).

We can stop the watchdog without restarting the system by closing “/dev/watchdog”. It is not possible if the kernel was compiled with “CONFIG_WATCHDOG_NOWAYOUT” enabled.
Overall, in order for the watchdog to operate the kernel needs to be compiled with CONFIG_WATCHDOG=y and “/dev/watchdog” character device should be created (with major number of 10 and minor number of 130 - checkout “man mknod” if you want to create it).

---

[67] https://github.com/torvalds/linux/blob/master/Documentation/watchdog/watchdog-api.rst
[68] https://linuxhint.com/linux-kernel-watchdog-explained/
[69] https://en.wikipedia.org/wiki/Watchdog_timer
[70] https://developer.toradex.com/linux-bsp/how-to/linux-features/watchdog-linux/
[71] https://linux.die.net/man/8/watchdog
[72] https://elixir.bootlin.com/linux/v6.0.11/source/drivers/watchdog

Lastly, if you want to see the status of the watchdog you can use the command “wdctl”[73] - As can be seen in the screenshot below[74].



[73] https://man7.org/linux/man-pages/man8/wdctl.8.html
[74] https://en.wikipedia.org/wiki/Watchdog_timer#/media/File:Wdctl_screenshot.png

zswap-shrink

Based on the kernel source code zswap is a backend for frontswap. Frontswap provides a "transcendent memory" interface for swap pages. In some cases we can get increased performance by saving swapped pages in RAM (or a RAM-like device) and not on disk as swap partition\swapfile[75]. The frontends are usually implemented in the kernel while the backend is implemented as a kernel module (as we will show soon). Zswap takes pages that are in the process of being swapped out and attempts to compress and store them in a RAM-based memory pool[76].

We can say that zswap trades CPU cycles for potentially reduced swap I/O. A significant performance improvement can happen in case the reads from the swap device are much slower than the reads from the compressed cache[77]. The "zswap_frontswap_store" is the function that attempts to compress and store a single page[78].

The kernel thread "zswap-shrink" is created created based on a workqueue[79]. On my VM (Ubuntu 22.04.1) zswap is compiled part of the kernel itself and not as a separate "*.ko" (kernel module). You can see in the screenshot below that it does not appear in the output of "lsmod" and is marked as builtin (look at the filename field) in the output of "modinfo".

For more information like the compression used by zswap (the default one is lzo) and other parameters that can be configured for zswap[80]. We can also read the parameters on "/sys/module/zswap/parameters".

[75] https://www.kernel.org/doc/html/v4.18/vm/frontswap.html
[76] https://elixir.bootlin.com/linux/latest/source/mm/zswap.c
[77] https://www.kernel.org/doc/html/v4.18/vm/zswap.html
[78] https://elixir.bootlin.com/linux/v6.1-rc2/source/mm/zswap.c#L1097
[79] https://elixir.bootlin.com/linux/v6.1-rc2/source/mm/zswap.c#L1511
[80] https://wiki.archlinux.org/title/zswap

khugepaged (Kernel Huge Pages Daemon)

The kernel thread “kugepaged” is created using the “kthread_run()” function[81]. It is responsible for the “Transparent Hugepage Support” (aka THP). “kugepaged” scans memory and collapses sequences of basic pages into huge pages[82].

We can manage and configure TPH using sysfs[83] or by using the syscalls “madvise”[84] and “prctl”[85]. The scan of memory is done by calling “khugepaged_do_scan()”[86] which in turn calls “khugepaged_scan_mm_slot()”[87]. In order to demonstrate that I have used the following bpftrace oneliner “**sudo bpftrace -e 'kfunc:khugepaged_scan_mm_slot{ printf("%s:%d\n",curtask->comm,curtask->pid); }'**”. The output is shown in the screenshot below.

Lastly, we can also monitor the modifications made by “khugepaged” by checking the information on “/proc”. For example we can check the “AnonHugePages”/”ShmemPmdMapped”/”ShmemHugePages” in “/proc/meminfo”, which is global for the entire system. If we want information regarding a specific process/task we can use “/proc/[PID]/smaps” and count “AnonHugePages”/”FileHugeMapped” for each mapping[88].

[81] https://elixir.bootlin.com/linux/latest/source/mm/khugepaged.c#L2551
[82] https://www.kernel.org/doc/html/latest/admin-guide/mm/transhuge.html
[83] https://www.kernel.org/doc/html/latest/admin-guide/mm/transhuge.html#thp-sysfs
[84] https://man7.org/linux/man-pages/man2/madvise.2.html
[85] https://man7.org/linux/man-pages/man2/prctl.2.html
[86] https://elixir.bootlin.com/linux/latest/source/mm/khugepaged.c#L2404
[87] https://elixir.bootlin.com/linux/v6.1.12/source/mm/khugepaged.c#L2250
[88] https://www.kernel.org/doc/html/latest/admin-guide/mm/transhuge.html

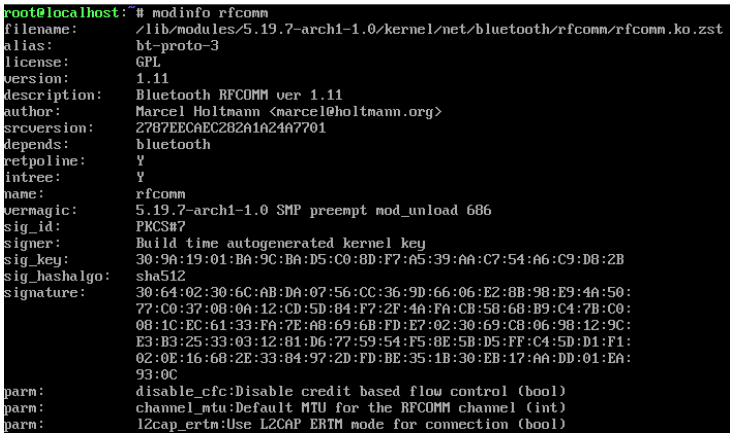
krfcommd (Kernel Radio Frequency Communication Daemon)

“krfcommd” is a kernel which is started by executing “kthread_run()” function[89]. The kernel thread executes the “rfcomm_run()” function[90]. Thus, we can say that “krfcommd” is responsible for RFCOMM connections[91].

RFCOMM (Radio Frequency Communication)[92] is a set of transport protocols on top of L2CAP which provides emulated RS-232 serial ports. It provides a simple reliable data stream (like TCP). It is used directly by many telephony related profiles as a carrier for AT commands, as well as being a transport layer for OBEX over Bluetooth[93].

Moreover, there is also an “rfcomm” cli tool in Linux. It is used to inspect and maintain RFCOMM configuration[94]. We can also go over the protocol specification[95]. Also, RFCOMM protocol supports up to 60 simultaneous connections between two Bluetooth devices. The number of connections that can be used simultaneously is implementation-specific. For the purposes of RFCOMM, a complete communication path involves two applications running on different devices (the communication endpoints) with a communication segment between them[96].

Lastly, RFCOMM is implemented as a kernel module. Thus, it can be compiled directly to the kernel or separate kernel module - in the screenshot below we can see it compiled as a separate file.



[89] https://elixir.bootlin.com/linux/latest/source/net/bluetooth/rfcomm/core.c#L2215
[90] https://elixir.bootlin.com/linux/latest/source/net/bluetooth/rfcomm/core.c#L2109
[91] https://stackoverflow.com/questions/57152408/what-is-the-internal-mechanics-of-socket-function
[92] https://www.btframework.com/rfcomm.htm
[93] https://en.wikipedia.org/wiki/List_of_Bluetooth_protocols
[94] https://linux.die.net/man/1/rfcomm
[95] https://www.bluetooth.com/specifications/specs/rfcomm-1-1/
[96] https://www.amd.e-technik.uni-rostock.de/ma/gol/lectures/wirlec/bluetooth_info/rfcomm.html

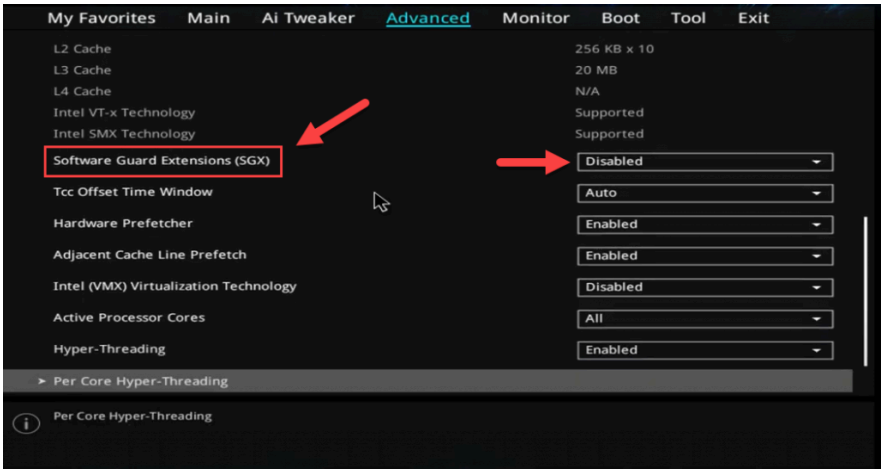
ksgxd (Kernel Software Guard eXtensions Daemon)

The kernel thread "ksgxd" is part of the Linux support for SGX (Software Guard eXtensions). Overall, SGX is a hardware security feature of Intel's CPU that enables applications to allocate private memory regions for data and code. There is a privilege opcode "ENCLS" which allows creation of regions and "ENCLU" which is a privilege opcode that allows entering and executing code inside the regions[97]. For more information about SGX you can read my writeup about it[98].

"ksgxd" is a kernel which is started by executing "kthread_run()" function[99]. The kernel thread executes the "ksgxd" function[100]. "ksgxd" is started while SGX is initializing and at boot time it re-initializes all enclave pages. In case of over commitment "ksgxd" is also responsible for swapping enclave memory[101] like "kswapd"[102].

If you want to know if your CPU supports SGX you can use the following command: "cat /proc/cpuinfo | grep sgx" (you can also use lscpu). You can also check your UEFI (legacy BIOS) configuration to check if you - check out the screenshot below[103].

Lastly, there is a great guide for an example SGX app using a Linux VM on Azure that I encourage you to read[104]. There is also specific documentation\information regarding the Linux stack for SGX[105]. By the way, we can go over the following Intel's GitHub repository dedicated to SGX[106].



[97] https://docs.kernel.org/x86/sgx.htmlhttps://docs.kernel.org/x86/sgx.html
[98] https://medium.com/@boutnaru/security-sgx-software-guard-extension-695cab7dbcb2
[99] https://elixir.bootlin.com/linux/v6.1.10/source/arch/x86/kernel/cpu/sgx/main.c#L427
[100] https://elixir.bootlin.com/linux/v6.1.10/source/arch/x86/kernel/cpu/sgx/main.c#L395
[101] https://elixir.bootlin.com/linux/v6.1.10/source/arch/x86/kernel/cpu/sgx/main.c#L188
[102] https://medium.com/@boutnaru/the-linux-process-journey-kswapd-22754e783901
[103] https://phoenixnap.com/kb/intel-sgx
[104] https://tsmatz.wordpress.com/2022/05/17/confidential-computing-intel-sgx-enclave-getting-started/
[105] https://download.01.org/intelsgxstack/2021-12-08/Getting_Started.pdf
[106] https://github.com/intel/linux-sgx

jbd2 (Journal Block Device 2)

"JBD" stands for "Journal Block Device"[107]. "jbd2" is a kernel which is started by executing "kthread_run()" function[108]. The name of the kernel thread has the following pattern "jbd2/[DeviceName]". The code is part of a kernel module - as you can see in the screenshot below.

Moreover, as we can see from the code it is a file system journal-writing code (part of the ext2fs journaling system). The journal is an area of reserved disk space used for logging transactional updates. The goal of "jbd2" is to schedule updates to that log[109].

The kernel thread executes the "kjournald2()" function[110]. This main thread function is used to manage a logging device journal. Overall, the thread has two main responsibilities: commit and checkpoint. Commit is writing all metadata buffers of the journal. Checkpoint means flushing old buffers in order to reuse an "unused section" of the log file[111].

Lastly, JBD was written by Stephen Tweedie and it is filesystem independent. There are different filesystems that are using it like etx3,etx4 and OCFS2. There are two versions: JBD created in 1998 with ext3 and JBD2 forked from JBD in 2006 with ext4[112].

```
root@localhost:~# modinfo jbd2
filename:       /lib/modules/5.19.7-arch1-1.0/kernel/fs/jbd2/jbd2.ko.zst
license:        GPL
srcversion:     7072394A13F8B3E5FCCE03C
depends:
retpoline:      Y
intree:         Y
name:           jbd2
vermagic:       5.19.7-arch1-1.0 SMP preempt mod_unload 686
sig_id:         PKCS#7
signer:         Build time autogenerated kernel key
sig_key:        30:9A:19:01:BA:9C:BA:D5:C0:8D:F7:A5:39:AA:C7:54:A6:C9:D8:2B
sig_hashalgo:   sha512
signature:      30:64:02:30:0E:96:1E:1D:03:C4:F6:FD:71:26:C9:EC:8A:98:49:B8:
                91:E7:00:8A:90:43:6B:B9:D9:DD:F2:D0:64:27:8E:3B:4F:0A:CA:BD:
                3F:EC:76:4B:AD:26:79:0E:72:28:FC:C6:02:30:01:CA:42:28:FD:AA:
                D5:66:C5:16:05:2A:59:D5:BA:BE:4B:B4:DA:5E:DE:5F:1B:1B:01:06:
                7D:7B:59:12:58:D2:C5:5D:99:63:81:6B:60:D2:63:6C:0F:18:5A:26:
                9D:93
```

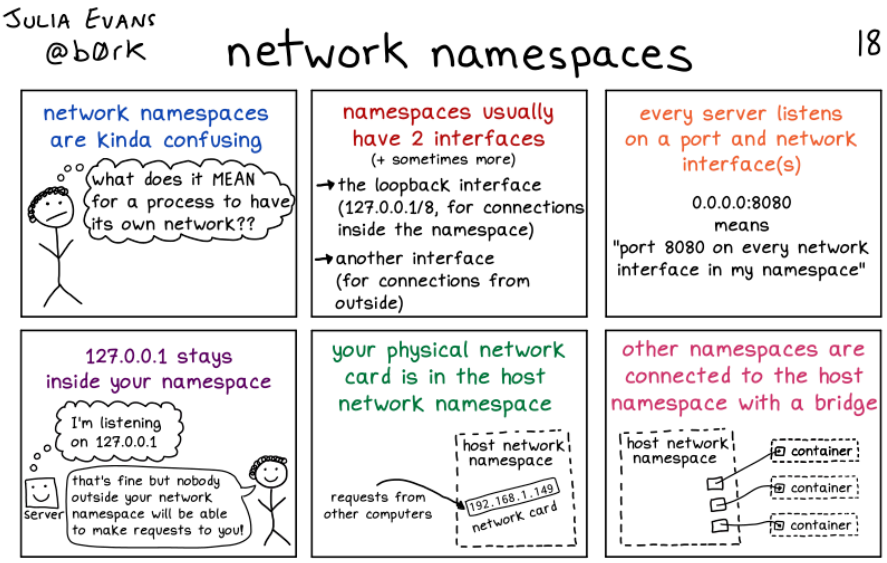
[107] https://manpages.ubuntu.com/manpages/jammy/man1/pmdajbd2.1.html
[108] https://elixir.bootlin.com/linux/v6.2.1/source/fs/jbd2/journal.c#L277
[109] https://elixir.bootlin.com/linux/v6.2.1/source/fs/jbd2/journal.c
[110] https://elixir.bootlin.com/linux/v6.2.1/source/fs/jbd2/journal.c#L169
[111] https://elixir.bootlin.com/linux/v6.2.1/source/fs/jbd2/journal.c#L152
[112] https://en.wikipedia.org/wiki/Journaling_block_device

netns (Network Namespace)

The kernel thread “netns” is based on a single threaded workqueue[113], which is created when the network namespace[114] is initialized (net_ns_init()). Also, for a reminder you can also check out the diagram below[115].

“netns” is responsible for cleaning up network namespaces. When a namespace is destroyed the kernel adds it to a cleanup list. The kernel thread “netns” goes over the list and performs the cleanup process using the “cleanup_net()” function[116].

If you want to see where all the magic happens is in “__put_net()” which queues the work on the “netns” to execute “cleanup_net()” function[117].



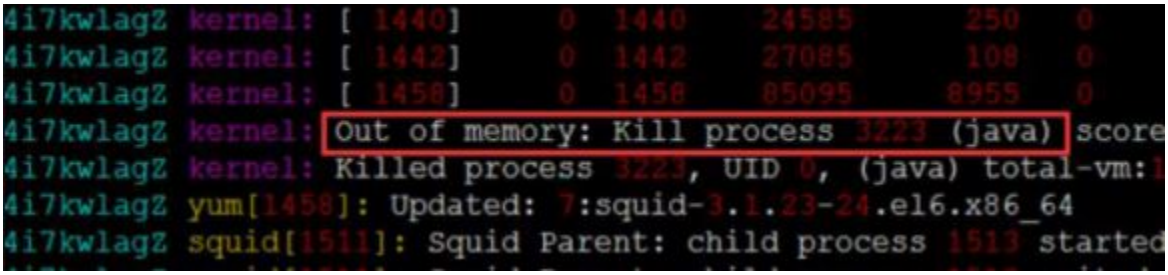
[113] https://elixir.bootlin.com/linux/v6.2-rc4/source/net/core/net_namespace.c#L1106
[114] https://medium.com/@boutnaru/linux-namespaces-network-namespace-part-3-7f8f8e06fef3
[115] https://wizardzines.com/comics/network-namespaces/
[116] https://elixir.bootlin.com/linux/v6.2.3/source/net/core/net_namespace.c#L565
[117] https://elixir.bootlin.com/linux/v6.2-rc4/source/net/core/net_namespace.c#L649

oom_reaper (Out-of-Memory Reaper)

“oom_reaper” is a kernel thread which was created using the “kthread_run” function[118]. Basically, it is the implementation of the OMM (Out–of-Memory) killer[119] function of the Linux kernel.The function which is executed by the thread is “oom_reaper”[120] which calls “oom_reap_task”[121].

Overall, based on the documentation the goal of the “oom_reaper” kernel thread is to try and reap the memory used by the OOM victim[122]. “oom_reaper” sleeps until it is waked up[123] which is after OOM kills the process[124].

Lastyl, after killing the process the victim is queued so the “oom_reaper” can release the resources[125]. You can see an example of the log created by OOM after killing a process[126].



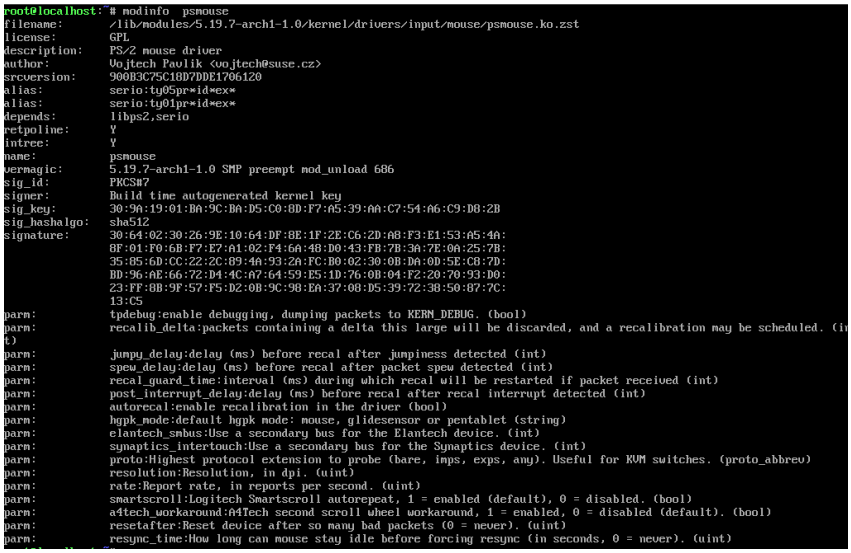
[118] https://elixir.bootlin.com/linux/v6.2.5/source/mm/oom_kill.c#L735
[119] https://medium.com/@boutnaru/linux-out-of-memory-killer-oom-killer-bb2523da15fc
[120] https://elixir.bootlin.com/linux/v6.2.5/source/mm/oom_kill.c#L640
[121] https://elixir.bootlin.com/linux/v6.2.5/source/mm/oom_kill.c#L609
[122] https://elixir.bootlin.com/linux/v6.2.5/source/mm/oom_kill.c#L504
[123] https://elixir.bootlin.com/linux/v6.2.5/source/mm/oom_kill.c#L680
[124] https://elixir.bootlin.com/linux/v6.2.5/source/mm/oom_kill.c#L947
[125] https://elixir.bootlin.com/linux/v6.2.5/source/mm/oom_kill.c#L992
[126] https://blog.capdata.fr/index.php/linux-out-of-memory-killer-oom-killer-pour-un-serveur-base-de-donnees-postgresql/

kpsmoused (Kernel PS/2 Mouse Daemon)

“kpsmoused” is a kernel thread which based on an ordered workqueue[127] which is allocated inside the “pmouse_init” function. “kpsmoused” is responsible for handling the input from PS/2 mouse devices.

Thus, “kpsmoused” transforms the raw data to high level event of mouse movements that be can consume from “/dev/input/mice”, “/dev/input/mouseX”, or “/dev/input/eventX”[128].

The kernel thread is created by the “psmouse” kernel module which is described as “PS/2 mouse driver” - as shown in the screenshot below (which was created using copy.sh). By the way, the “kpsmoused” is created as part of “/drivers/input/mouse/psmouse-base.c” since kernel 2.5.72[129].



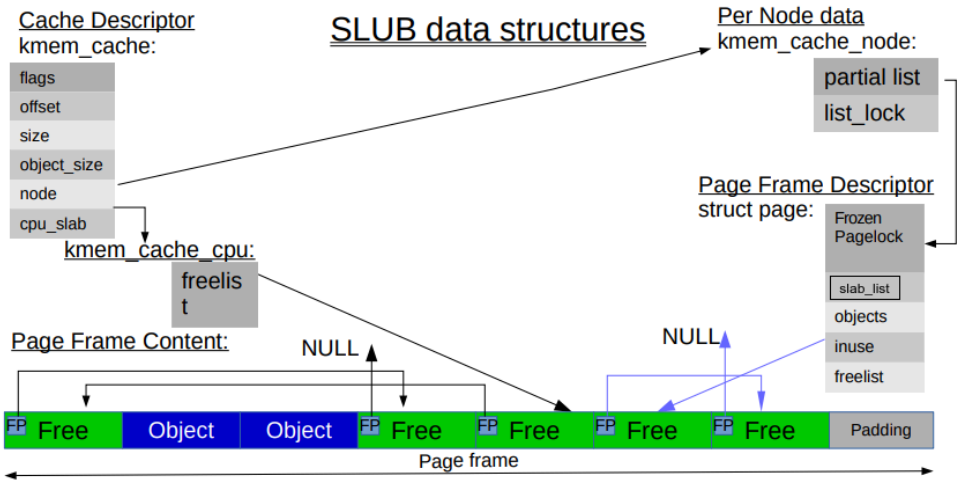
[127] https://elixir.bootlin.com/linux/v6.2.6/source/drivers/input/mouse/psmouse-base.c#L2046
[128] https://www.kernel.org/doc/html/v5.5/input/input.html
[129] https://elixir.bootlin.com/linux/v2.5.72/source/drivers/input/mouse/psmouse-base.c

Slub_flushwq (SLUB Flush Work Queue)

"slub_flushwq" is a kernel thread which based on a workqueue[130] which is allocated inside the "kmem_cache_init_late" function. Based on the source code the allocation is done only if "CONFIG_SLUB_TINY" is enabled[131]. From the documentation "CONFIG_SLUB_TINY" is for configuring SLUB allocation in order to achieve minimal memory footprint, it is not recommended for systems with more than 16 GB of RAM[132]. The queuing of work is done inside the "flush_all_cpus_locked" function[133].

SLUB is also known as the "Unqueued Slab Allocator"[134]. Slab allocation[135] is a memory management mechanism which allows efficient memory allocation of objects. It is done using reduction of fragmentation that is caused due to allocations/deallocations[136].

Thus, SLUB is a slab allocator that limits the use of cache lines instead of using queued object per cpu/per node list[137]. So, it is less complicated because it does not keep queues (like for each CPU). The only queue is a linked list for all the objects in each of the slub pages[138]. The interplay between the three main data structures (kmem_cache, kmem_cache_cpu, kmem_cache_node) used by the SLUB allocator is shown in the diagram below[139] .



[130] https://elixir.bootlin.com/linux/v6.2.6/source/mm/slub.c#L5057
[131] https://elixir.bootlin.com/linux/v6.2.6/source/mm/slub.c#L5056
[132] https://cateee.net/lkddb/web-lkddb/SLUB_TINY.html
[133] https://elixir.bootlin.com/linux/v6.2.6/source/mm/slub.c#L2822
[134] https://lwn.net/Articles/229096/
[135] https://hammertux.github.io/slab-allocator
[136] https://en.wikipedia.org/wiki/Slab_allocation
[137] https://elixir.bootlin.com/linux/v6.2.6/source/mm/slub.c#L3
[138] https://hammertux.github.io/slab-allocator
[139] https://hammertux.github.io/img/SLUB-DS.png

pgdatinit

“pgdatinit” is a kernel which is started by executing the “kthread_run()” function[140]. The kernel thread executes the “deferred_init_memmap()” function[141].

Thus, “pgdatinit” is responsible for initializing memory on every node of the system. For each node a dedicated kernel thread is created with the name pattern “pgdatinit[NodeNumber]”[142].

Overall, the kernel thread is created in case CONFIG_DEFERRED_STRUCT_PAGE_INIT is enabled when compiling the kernel. Which states that initialization of struct pages is deferred to kernel threads[143].

Lastly, after the initialization flow is finished an information message is sent to the kernel ring buffer[144] - as you can see in the image below[145].

```
[    0.212320] .... node  #0, CPUs:        #1  #2  #3  #4  #5  #6  #7  #8  #9
#10 #11 #12 #13 #14 #15 #16 #17 #18 #19 #20 #21 #22 #23
[    0.260348] smp: Brought up 1 node, 24 CPUs
[    0.260348] smpboot: Max logical packages: 2
[    0.260348] smpboot: Total of 24 processors activated (182404.32 BogoMIPS)
[    0.357570] node 0 deferred pages initialised in 96ms
```

[140] https://elixir.bootlin.com/linux/v6.3-rc4/source/mm/page_alloc.c#L2284
[141] https://elixir.bootlin.com/linux/v6.3-rc4/source/mm/page_alloc.c#L2108
[142] https://elixir.bootlin.com/linux/v6.3-rc4/source/mm/page_alloc.c#L2283
[143] https://cateee.net/lkddb/web-lkddb/DEFERRED_STRUCT_PAGE_INIT.html
[144] https://elixir.bootlin.com/linux/v6.3-rc4/source/mm/page_alloc.c#L2177
[145] https://www.mail-archive.com/debian-bugs-dist@lists.debian.org/msg1822096.html

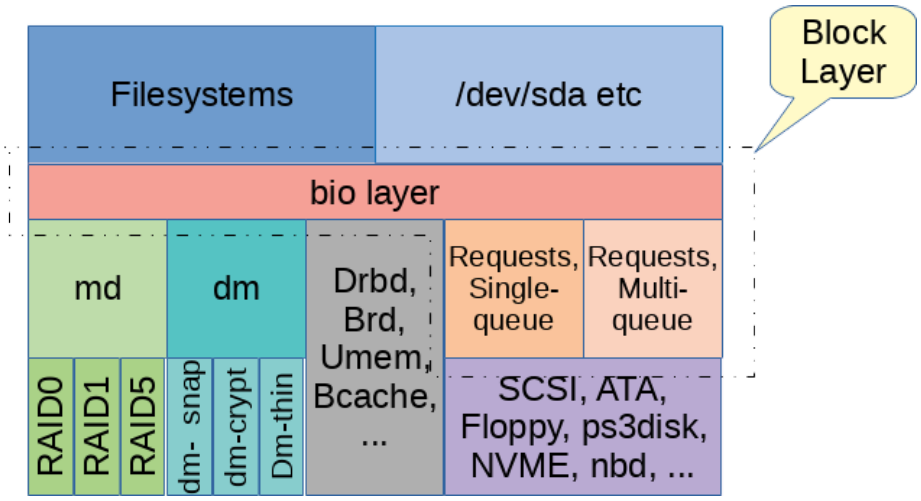
kblockd (Kernel Block Daemon)

“kblockd” is a kernel thread based on a workqueue[146] which is marked with high priority and that it can be used for memory reclaim. It is used for performing I/O disk operations.

Moreover, we can deduct based on the location of the file in the Linux source tree (/block) that “kblockd” is part of the “Block Layer” (which is responsible for managing block devices) - as shown in the diagram below[147].

Overall, one might think that we can use keventd[148] for performing I/O operations. However, because they can get blocked on disk I/O. Due to that, “kblockd” was created to run low-level disk operations like calling relevant block device drivers[149].

Thus, “kblockd” must never block on disk I/O so all the memory allocations should be GFP_NOIO. We can sum up that it is used to handle all read/writes requests to block devices[150].



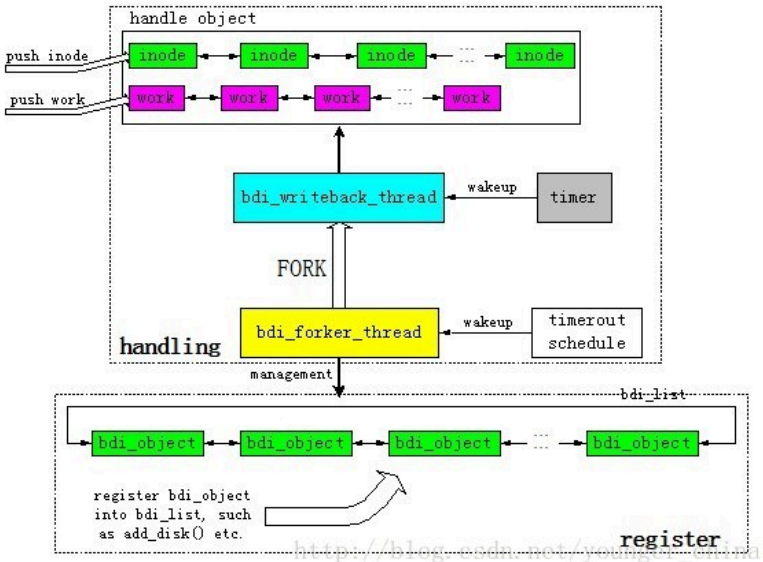
[146] https://elixir.bootlin.com/linux/v6.2.9/source/block/blk-core.c#L1191
[147] https://lwn.net/Articles/736534/
[148] https://lwn.net/Articles/11351/
[149] https://mirrors.edge.kernel.org/pub/linux/kernel/people/akpm/patches/2.5/2.5.70/2.5.70-mm8/broken-out/kblockd.patch
[150] https://elixir.bootlin.com/linux/v6.3-rc4/source/block/blk-core.c#L13

writeback

The kernel thread “writeback” is based on a workqueue[151]. The goal of the kernel thread is to serve all async writeback tasks[152]. Thus, “writeback” is flushing dirty information from the page cache (aka disk cache) to disks. The page cache is the main disk cache used by the kernel. The kernel references the page cache when reading from/writing to disk[153].

Overall, they are two ways of flushing dirty pages using writeback. The first is in case of an explicit writeback request - like syncing inode pages of a superblock. Thus, the “wb_start_writeback()” is called with the superblock information and the number of pages to flush. The second one is when there is no specific writeback request, in this case there is a timer that wakes up the thread periodically to flush dirty data[154].

Moreover, from kernel 3.2 the original mechanism of “pdflush” was changed to “bdi_writeback”. By doing so it solves one of the biggest limitations of “pdflush” in a multi-disk environment. In that case “pdflush” manages the buffer/page cache of all the disks which creates an IO bottleneck. On the other hand, “bdi_writeback” creates a thread for each disk[155]. By the way, “bdi” stands for “Backing Device Information”[156]. Lastly, to get an overview of the “writeback” mechanism you can checkout the diagram below[157].



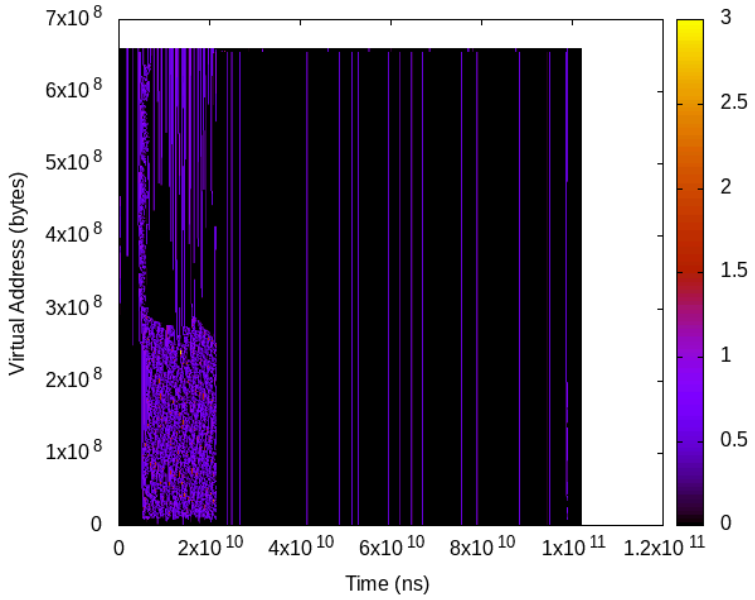
[151] https://elixir.bootlin.com/linux/v6.2.5/source/mm/backing-dev.c#L363
[152] https://elixir.bootlin.com/linux/v6.2.5/source/mm/backing-dev.c#L35
[153] https://www.oreilly.com/library/view/understanding-the-linux/0596005652/ch15s01.html
[154] https://lwn.net/Articles/326552/
[155] https://blog.csdn.net/younger_china/article/details/55187057
[156] https://lwn.net/Articles/326552/
[157] https://blog.csdn.net/younger_china/article/details/55187057

kdamond (Data Access MONitor)

“kdamond” is a kernel thread which is created using the “kthread_run()” function[158] which is part of the DAMON (Data Access MONitor) subsystem. The kernel thread executes the “kdamon_fn()” function[159].Overall, DAMON provides a lightweight data access monitoring facility that can help users in analyzing the memory access patterns of their systems[160]. Based on the documentation DAMON increases the memory usage by 0.12% and slows the workloads down by 1.39%[161].

Also, DAMON has an API for kernel programs[162]. Moreover, there is also DAMOS (DAMon-Based Operations Schemas). Using that, users can develop and run access-aware memory management with no code and just using configurations[163].

Probably the best way to go over DAMON data is by using visualization. A great demonstration for that has been done by SeongJae Park using the PARSEC3/SPLASH-2X benchmarks[164]. The output was heatmaps of the dynamic access patterns for heap area, mmap()ed area and the stack area. One example is shown in the image below, it visualizes the data access pattern of the stack area when running the parsec3-blackscholes[165]. Lastly, there are also other mechanisms in Linux that can help with data access monitoring such as “Perf Mem” and “Idle Page Tracking”



[158] https://elixir.bootlin.com/linux/v6.3-rc5/source/mm/damon/core.c#L632
[159] https://elixir.bootlin.com/linux/v6.3-rc5/source/mm/damon/core.c#L1304
[160] https://www.kernel.org/doc/html/latest/admin-guide/mm/damon/index.html
[161] https://damonitor.github.io/doc/html/v20/vm/damon/eval.html
[162] https://www.kernel.org/doc/html/v5.17/vm/damon/api.html#functions
[163] https://sjp38.github.io/post/damon/
[164] https://parsec.cs.princeton.edu/parsec3-doc.htm
[165] https://lwn.net/Articles/813108/

kintegrityd (Kernel Integrity Daemon)

“kintegrityd” is a kernel thread based on a workqueue[166] which is responsible for verifying the integrity of block devices by reading/writing data from/to them. The function which is executed by the workqueue is “bio_integrity_verify_fn”[167]. The function is called to complete a read request by verifying the transferred integrity metadata and then calls the original bio end_io function[168].

This procedure is done to ensure that the data was not changed by mistake (like in a case of a bug or an hardware failure[169]. This mechanism is also called “bio data integrity extensions“. And it allows the user to get protection for the entire flow: from the application to storage device. The implementation is transparent to the application itself and it is part of the block layer[170].

Moreover, in order for it to work we should enable CONFIG_BLK_DEV_INTEGRITY, which is defined as “Block layer data integrity support”[171]. The filesystem does not have to be aware that the block device can include integrity metadata. The metadata is generated as part of the block layer when calling the submit_bio() function[172]. We can toggle the writing of metadata using “/sys/block/<BlockDevice>/integrity/write_generate“ and the verification of the metadata using “/sys/block/<BlockDevice>/integrity/read_verify” - as shown in the screenshot below.

Lastly, there are also file systems which are integrity aware (and they will generate/verify the metadata). There are also options for sending the metadata information from userspace[173].

```
Troller $ ls
device_is_integrity_capable  format  protection_interval_bytes  read_verify  tag_size  write_generate
Troller $
```

[166] https://elixir.bootlin.com/linux/v6.1/source/block/bio-integrity.c#L455
[167] https://elixir.bootlin.com/linux/v6.1/source/block/bio-integrity.c#L317
[168] https://elixir.bootlin.com/linux/v6.1/source/block/bio-integrity.c#L313
[169] https://www.quora.com/What-is-the-purpose-of-kintegrityd-Linux-Kernel-Daemon/answer/Liran-Ben-Haim
[170] https://www.kernel.org/doc/Documentation/block/data-integrity.txt
[171] https://elixir.bootlin.com/linux/v6.1/source/block/Kconfig#L60
[172] https://www.kernel.org/doc/Documentation/block/data-integrity.txt
[173] https://www.kernel.org/doc/Documentation/block/data-integrity.txt

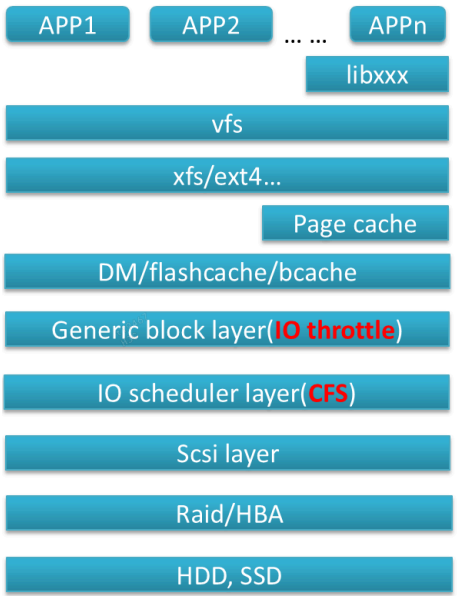
kthrotld (Kernel Throttling Daemon)

“kthrotld” is a kernel thread which was created using an workqueue[174] which acts as an interface for controlling IO bandwidth on request queues (throttling requests). Overall, read and write requests to block devices are placed on request queues[175].

In order to understand how request queues are used the best way is to check the source code of the kernel. The first step is going over the definition of “struct request_queue”[176] and then where is it referenced[177]. By the way, in kernel version 6.1.1 it is referenced in 199 files. We can summarize that a request queue holds I/O requests in a linked list. Also, it is a best practice to create a separate request queue for every device[178].

Thus, we can say that “kthrotld” acts as a block throttle, which provides block QoS (Quality of Service). It is used to limit IOPS (I/O per second)/BPS (Bits per second) per cgroup (control group)[179].

Overall, IO throttling is done as part of the generic block layer and before the IO scheduler as seen in the diagram below[180]. As described above this is also known as “Block Throttling”[181].



[174] https://elixir.bootlin.com/linux/v6.1.1/source/block/blk-throttle.c#L2470
[175] https://www.halolinux.us/kernel-architecture/request-queues.html
[176] https://elixir.bootlin.com/linux/v6.1.1/source/include/linux/blkdev.h#L395
[177] https://elixir.bootlin.com/linux/v6.1.1/C/ident/request_queue
[178] https://www.oreilly.com/library/view/linux-device-drivers/0596000081/ch12s04.html
[179] https://developer.aliyun.com/article/789736
[180] https://blog.csdn.net/yiyeguzhou100/article/details/104044419
[181] https://developer.aliyun.com/article/789736

scsi_eh (Small Computer System Interface Error Handling)

The kernel thread “scsi_eh” is executed using the “kthread_run” function. The name pattern of the kernel thread is “scsi_eh_<SCSI_HOST_NUMBER>”[182]. It is the “SCSI error handler” which is responsible for all of the error handling targeting every SCSI host[183]. The kernel thread is executing the “scsi_error_handler” function[184].

Moreover, a SCSI controller which coordinates between other devices on the SCSI bus is called a “host adapter”. It can be a card connected to a slot or part of the motherboard. You can see an example of a SCSI connector in the image below[185].

Lastly, SCSI stands for “Small Computer System Interface”[186]. It is a set of standards (from ANSI) for electronic interfaces in order to communicate with peripheral hardware like CD-ROM drives, tap drivers, printers, disk drives and more[187]. For more information about SCSI I suggest going over.

[182] https://elixir.bootlin.com/linux/v6.4-rc1/source/drivers/scsi/hosts.c#L504
[183] https://elixir.bootlin.com/linux/v6.4-rc1/source/drivers/scsi/scsi_error.c#L2230
[184] https://elixir.bootlin.com/linux/v6.4-rc1/source/drivers/scsi/scsi_error.c#L2233
[185] https://computer.howstuffworks.com/scsi.htm
[186] https://hackaday.com/2023/03/02/scsi-the-disk-bus-for-everything/
[187] https://www.techtarget.com/searchstorage/definition/SCSI

blkcg_punt_bio

“blkcg_punt_bio” is a kernel thread based on a workqueue. The workqueue itself is created in the “blkcg_init” function[188]. It is part of the common block controller cgroup interface[189].

Overall, when a shared kernel thread tries to issue a synchronized block I/O (bio) request for a specific cgroup it can lead to a priority inversion. It can happen if the kernel thread is blocked waiting for that cgroup[190]. An example of priority inversion is shown in the diagram below[191].

Thus, to avoid the problem mentioned above the function “submit_bio”[192] punts the issuing of the bio request to a dedicated work item (per-block cgroup).
It calls “blkcg_punt_bio_submit”[193], which will call “__blkcg_punt_bio_submit”[194].

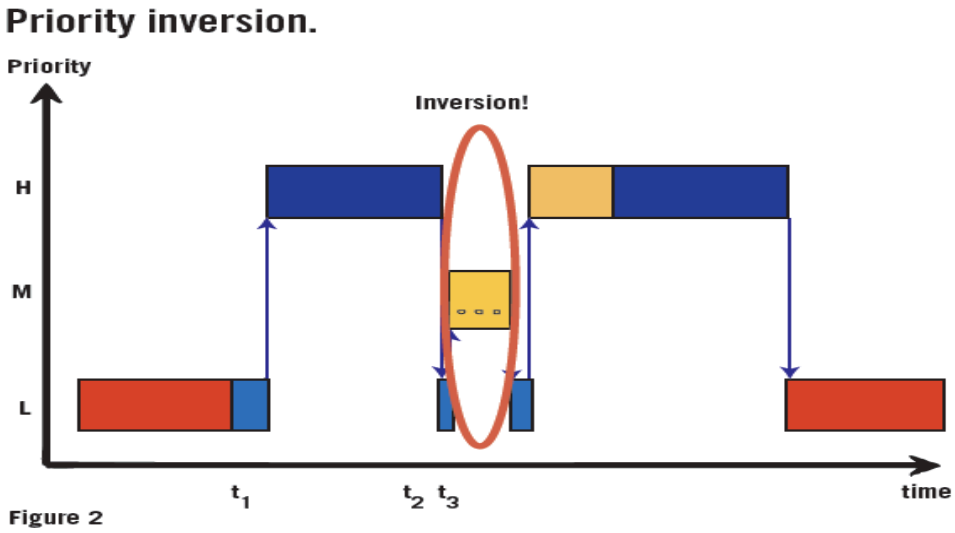


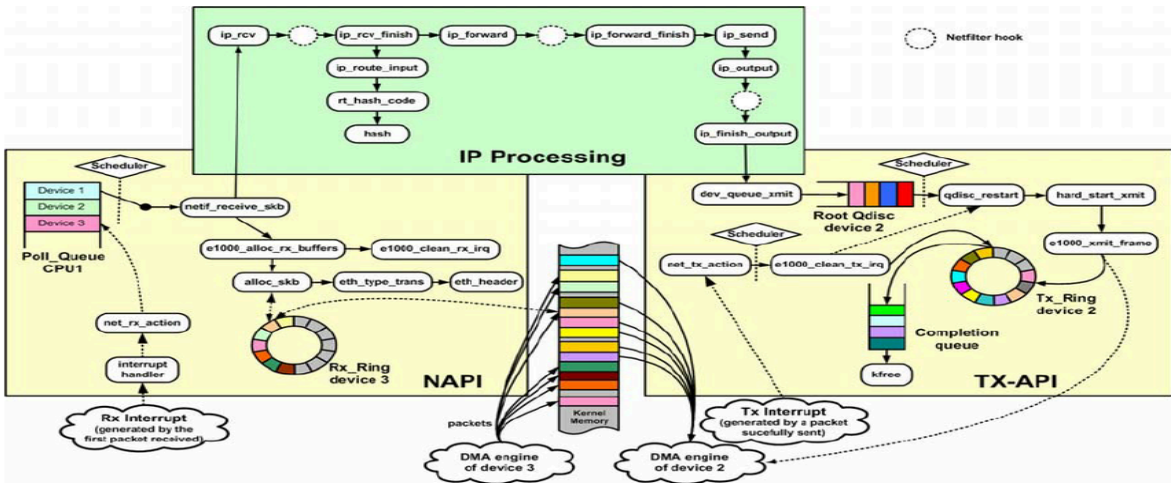
Figure 2

[188] https://elixir.bootlin.com/linux/v6.2.5/source/block/blk-cgroup.c#L2058
[189] https://elixir.bootlin.com/linux/v6.2.5/source/block/blk-cgroup.c#L3
[190] https://patchwork.kernel.org/project/linux-block/patch/20190627203952.386785-6-tj@kernel.org/
[191] https://embeddedgurus.com/barr-code/2010/11/firmware-specific-bug-8-priority-inversion/
[192] https://elixir.bootlin.com/linux/v6.2.5/source/block/blk-core.c#L829
[193] https://elixir.bootlin.com/linux/v6.2.5/source/block/blk-cgroup.h#L380
[194] https://elixir.bootlin.com/linux/v6.2.5/source/block/blk-cgroup.c#L1657

napi (New API)

NAPI stands for “New API” which is used to reduce the number of received interrupts. Think about cases in which the network driver receives a large number of packets at a fast pace[195]. If we think about it in the case of a Gigabit network card and an MTU of 1500 the CPU will get about 90K of interrupt per second. Thus, we can say that NAPI is an extension to the Linux packet processing framework, which is done for improving performance for high speed networking. This is performed using interrupt mitigation and packet throttling. It is important to say that the addition of NAPI does not break backward compatibility[196]. “napi” is a kernel thread which is created using the “kthread_run()”[197] function which is part of the NAPI (New API) subsystem. The name of the kernel thread is based on the pattern “napi[DeviceName]-[NAPI-ID]”. It executes the “napi_threaded_poll”[198] function.

Due to that, drivers that support NAPI can disable hardware interrupts as a mechanism for packet reception. In that case the network stack relies on polling for new packets at a specific interval. It might seem that polling is less efficient but in case the network device is busy any time the kernel will poll for a packet it will get something[199]. Lastly, the way NAPI does that is by combining hardware interrupts and polling. When a hardware interrupt is received, the driver disables it and notifies the kernel to read the packets.Then a kernel software interrupt polls the network device for a specific time. When the time runs out/there is no more data the kernel will enable the hardware interrupt again[200]. A detailed diagram of the NAPI flow is shown in the diagram below[201].



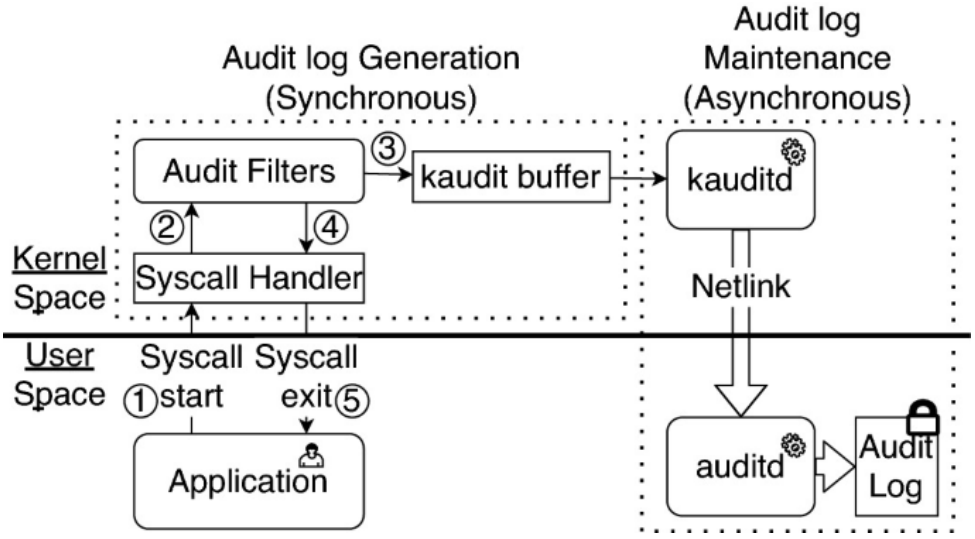
[195] https://www.hitchhikersguidetolearning.com/2023/04/09/handling-receive-packets-via-napi/
[196] https://wiki.linuxfoundation.org/networking/napi
[197] https://elixir.bootlin.com/linux/v6.4-rc4/source/net/core/dev.c#L1371
[198] https://elixir.bootlin.com/linux/v6.4-rc4/source/net/core/dev.c#L662
[199] https://lwn.net/Articles/833840/
[200] https://www.jianshu.com/p/7d4e36c0abe8
[201] https://tinyurl.com/mrtnus8w

kauditd (Kernel Audit Daemon)

“kauditd” is a kernel thread which is started using the “kthread_run” function[202]. The kernel thread is calling the “kauditd_thread” function, this function is responsible for sending audit logs to userspace[203].Overall, the kernel mechanism in the Linux kernel has a couple of goals: integrate fully with LSMs[204], minimal run-time overhead when performing auditing, ability to disable system call auditing at boot time, allow to be used by other parts of the kernel for auditing, netlink interface to userspace and support for filtering to minimize the information sent to user-mode[205].

Thus, we can say “kauditd” is the kernel component of the “Linux Auditing System” which handles the audit events - as shown in the diagram below[206] . In order to configure which set of rules are going to be loaded in the kernel audit system we can use the “/etc/audit/audit.rules” file. This file can hold configuration in one of three categories: control (configuring the audit system), file system rules monitoring rules and system call monitoring rules[207].

Lastly, by using the “Linux Auditing System” the system administrator can investigate what happens in the system for the purpose of debugging or in case of a security incident. We can also use the “auditctl” utility get/add/delete rules as part of Linux's kernel audit system[208]. Also, there are great examples for “audit.rules” in GitHub[209].



[202] https://elixir.bootlin.com/linux/v6.4-rc4/source/kernel/audit.c#L1700
[203] https://elixir.bootlin.com/linux/v6.4-rc4/source/kernel/audit.c#L828
[204] https://medium.com/@boutnaru/linux-security-lsm-linux-security-modules-907bbcf8c8b4
[205] https://elixir.bootlin.com/linux/v6.4-rc4/source/kernel/audit.c#L11
[206] https://link.springer.com/chapter/10.1007/978-3-031-17143-7_30
[207] https://manpages.debian.org/unstable/auditd/audit.rules.7.en.html
[208] https://linux.die.net/man/8/auditctl
[209] https://github.com/Neo23x0/auditd/blob/master/audit.rules

tpm_dev_ wq (Trusted Platform Module Device Work Queue)

“tpm_dev_wq” is a kernel thread base on a workqueue[210]. It belongs a device file system interface for “Trusted Platform Module” aka TPM[211].

Overall, TPM is an international standard for secure cryptoprocessors. Those are microprocessors which are used for a variety of security applications such as secure boot, random number generating and crypto key storage[212].

Moreover, a work is queued for “tpm_dev_wq” as part of the function “tpm_common_write”[213]. In case we are working in non-blocking mode an async job for sending the command is scheduled[214].

Lastly, “tpm-dev-common.c” is compiled as part of the kernel TPM device drivers as shown in the Makefile[215]. The information about the TPM module is shown in the screenshot below. I am using Ubuntu “22.04.2”, in which the TPM module is compiled directly into the kernel itself.

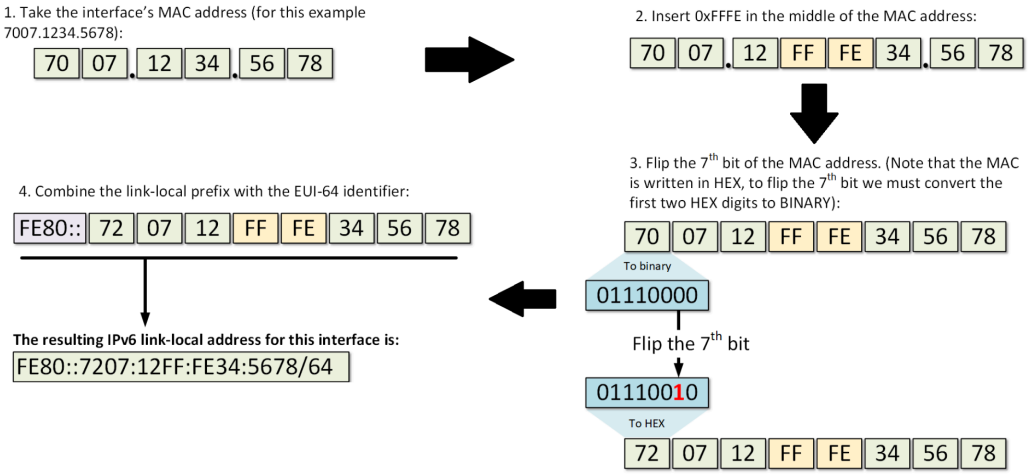
```
Troller $ cat /etc/issue
Ubuntu 22.04.2 LTS \n \l

Troller $ modinfo tpm
name:           tpm
filename:       (builtin)
license:        GPL
file:           drivers/char/tpm/tpm
version:        2.0
description:    TPM Driver
author:         Leendert van Doorn (leendert@watson.ibm.com)
parm:           suspend_pcr:PCR to use for dummy writes to facilitate flush on suspend. (uint)
```

[210] https://elixir.bootlin.com/linux/v6.3-rc7/source/drivers/char/tpm/tpm-dev-common.c#L273
[211] https://elixir.bootlin.com/linux/v6.3-rc7/source/drivers/char/tpm/tpm-dev-common.c#L13
[212] https://wiki.archlinux.org/title/Trusted_Platform_Module
[213] https://elixir.bootlin.com/linux/v6.3-rc7/source/drivers/char/tpm/tpm-dev-common.c#L209
[214] https://elixir.bootlin.com/linux/v6.3-rc7/source/drivers/char/tpm/tpm-dev-common.c#L202
[215] https://elixir.bootlin.com/linux/v6.3-rc7/source/drivers/char/tpm/Makefile

ipv6_addrconf (IPv6 Address Auto Configuration)

"ipv6_addrconf" is a kernel thread which is based on a workqueue[216]. This code is part of the Linux INET6 implementation and is responsible for the IPv6 Address auto configuration[217]. Overall, each IPv6 entity in the network needs a globally unique address for communicating outside of the local segment. In order to get such an address there are a few options: manual assignment of an address, DHCPv6 (Dynamic Host Configuration Protocol version 6) and SLAAC (Stateless Address Autoconfiguration). When talking about stateless and stateful it means if there is a server/device that keep tracks of a state for each address assignment[218].

Moreover, the stateless address autoconfiguration has the following phases. The node configures itself with a link-local address. The most known way for doing that is using the link-local prefix "FE80::/64" and combining that with the EUI-64 identifier generated from the MAC address - as shown in the diagram below.



The flow above It is done by the function "addrconf_addr_gen"[219]. We can see there the link-local prefix[220] and the call for generating the EUI-64 identifier by the function "ipv6_generate_eui64"[221]. After that, the node performs DAD (Duplicate Address Detection) in order to ensure that the address is unique in the local segment. It is done using NDP (Neighbor Discovery Protocol), which defines 5 new packets types to ICMPv6 that allows to provide different functionality like DAD and others like parameter discovery, next hop determination and more[222]. If there are no issues with the link-local address it is assigned to the specific device. The DAD operation is performed by the function "addrconf_dad_work"[223].

[216] https://elixir.bootlin.com/linux/v6.2.11/source/net/ipv6/addrconf.c#L7292
[217] https://elixir.bootlin.com/linux/v6.2.11/source/net/ipv6/addrconf.c#L3
[218] https://www.networkacademy.io/ccna/ipv6/stateless-address-autoconfiguration-slaac
[219] https://elixir.bootlin.com/linux/v6.2.11/source/net/ipv6/addrconf.c#L3314
[220] https://elixir.bootlin.com/linux/v6.2.11/source/net/ipv6/addrconf.c#L3326
[221] https://elixir.bootlin.com/linux/v6.2.11/source/net/ipv6/addrconf.c#L3345
[222] https://datatracker.ietf.org/doc/html/rfc4862
[223] https://elixir.bootlin.com/linux/v6.2.11/source/net/ipv6/addrconf.c#L4058

Lastly, there is also a similar flow for configuring a global unicast address. The difference is that there is also a need for sending a “Router Solicitation” message for getting the global prefix of the segment, I will leave the details of that for a future writeup.

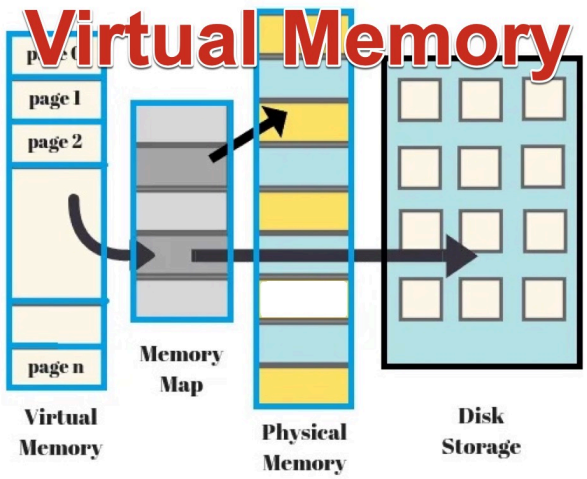
mm_percpu_wq (Per-CPU Memory Work Queue)

“mm_percpu_wq” is a kernel thread based on a workqueue which is created in the “init_mm_internals” function[224]. It is part of the the statistics management regarding virtual memory[225]. An overview diagram of virtual memory is shown below[226].

Overall, “mm_percpu_wq” is the worker thread which updates different counters about the virtual memory of a Linux system. It is also called the “vmstat worker”[227]. “vmstat” stands for “Virtual Memory Statistics” which includes information such as: number of free pages, number of mapped pages, number or dirty pages, amount of memory allocated to kernel stacks and more (there are more than 150 different counters).

The statistics can be read from the file “/proc/vmstat”[228]. This proc entry is created with others (“buddyinfo, “pagetypeinfo” and “zoneinfo”) in the same file in which “mm_percpu_mm” is allocated[229]. We can see the list of the metric counters in the source code[230].

As it names suggested the kernel thread is responsible for accumulating the vm events among all CPUs[231]. It is done by going over all the “online” CPUs[232]. Lastly, we can use different cli tools to review the different statistic counters. One of those tools is “vmstat”[233].



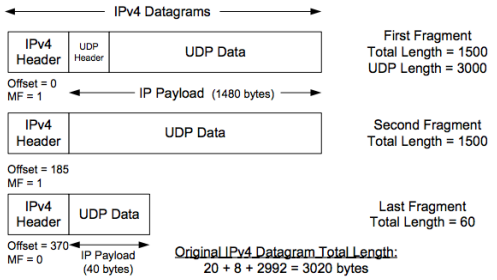
[224] https://elixir.bootlin.com/linux/v6.4-rc5/source/mm/vmstat.c#L2100
[225] https://elixir.bootlin.com/linux/v6.4-rc5/source/mm/vmstat.c#L5
[226] https://iboysoft.com/wiki/virtual-memory.html
[227] https://elixir.bootlin.com/linux/v6.4-rc5/source/mm/vmstat.c#L2021
[228] https://man7.org/linux/man-pages/man5/proc.5.html
[229] https://elixir.bootlin.com/linux/v6.4-rc5/source/mm/vmstat.c#L2123
[230] https://elixir.bootlin.com/linux/v6.4-rc5/source/mm/vmstat.c#L1168
[231] https://elixir.bootlin.com/linux/v6.4-rc5/source/mm/vmstat.c#L126
[232] https://elixir.bootlin.com/linux/v6.4-rc5/source/mm/vmstat.c#L117
[233] https://linux.die.net/man/8/vmstat

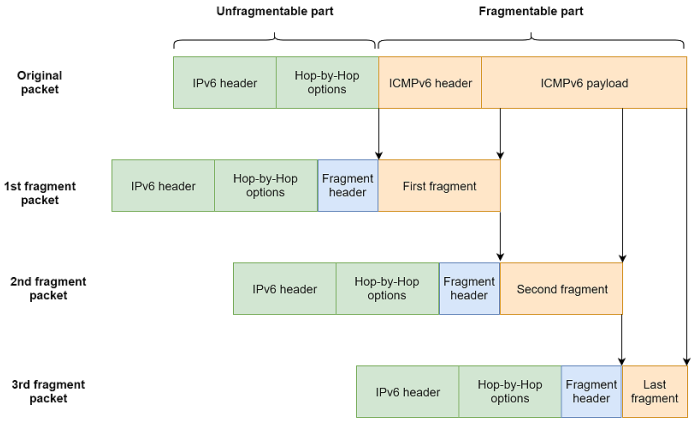
inet_frag_wq (IP Fragmentation Work Queue)

The kernel thread “inet_frag_wq” is created using a workqueue[234], we could have guessed it based on a workqueue do to the “wq” suffix. It is used for fragment management of IP packets. Thus the goal of “inet_frag_wq” is to reassemble fragmented IPv4/IPv6 packets[235].

Overall, the goal of IP fragmentation is to split packets into smaller chunks in order to allow them to meet the MTU (Maximum Transmission Unit) requirement of a specific network. There is an implementation difference between IP fragmentation in IPv4 and IPv6. On IPv4 the information needed for fragmentation is part of the IPv4 header which in IPv6 there is a specific “Fragmentation Header”[236]. An illustration of the flow is shown in the diagram below both for IPv4[237] and IPv6[238].

Thus, “inet_frag_wq” is relevant when a fragmented IP packet arrives at a specific system. The OS stores the fragmented packets in a queue and reassembles them before they are passing the data to the upper layers of the network stack. The fragment queue is represented by "struct inet_frag_queue"[239]. Moreover, we can see in the source code the function “ip_frag_reasm” which is responsible for building a new IP datagram from all of its fragments[240].





[234] https://elixir.bootlin.com/linux/v6.2-rc1/source/net/ipv4/inet_fragment.c#L211
[235] https://elixir.bootlin.com/linux/v6.2-rc1/source/net/ipv4/inet_fragment.c#L6
[236] https://www.geeksforgeeks.org/ipv6-fragmentation-header/
[237] https://notes.shichao.io/tcpv1/ch10/
[238] https://blog.quarkslab.com/analysis-of-a-windows-ipv6-fragmentation-vulnerability-cve-2021-24086.html
[239] https://elixir.bootlin.com/linux/v6.2-rc1/source/include/net/inet_frag.h#L66
[240] https://elixir.bootlin.com/linux/v6.2-rc1/source/net/ipv4/ip_fragment.c#L411

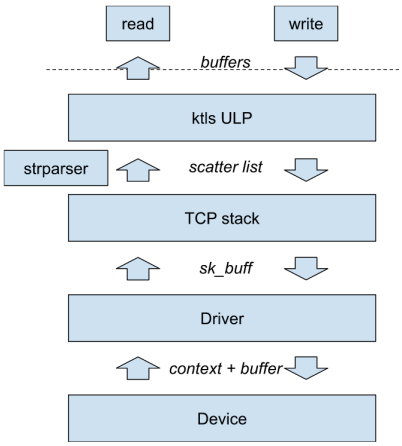
kstrp (Stream Parser)

“kstrp” is based on a single threaded workqueue[241]. Form the source code documentation we can see that “strparser” means “Stream Parser”[242]. A stream parser is a utility that gets data streams and parsers the application layer protocol over those streams. A stream parser can work in one of two modes: general mode or receive callback mode.

In general mode, a sequence of socket buffers (skbs) are given to the stream parser from an outside source. Messages are parsed and delivered as the sequence is processed. This mode allows a stream parser to be applied to any arbitrary stream of data. In receive callback mode, the stream parser is called from the data_ready callback of the TCP socket. Messages are parsed and delivered as they are received on the socket[243].

Thus, we can say that we can parse application layer protocol messages in TCP. It is basically a generalization of KCM (Kernel Connection Multiplexor)[244].
KMC provides a message based interface over TCP for generic application protocols. With the use of KMC applications can send/receive application messages efficiently over TCP[245].

Lastly, “strparser” allows intercepting packets on TCP connections. This is done at the kernel level which provides the ability to perform custom processing. The processing can be done using the BPF/Kernel module[246]. One example for that is the implementation of KTLS[247] (a Linux TLS/DTLS kernel module). An illustration of the flow is shown below[248].



[241] https://elixir.bootlin.com/linux/v6.1.1/source/net/strparser/strparser.c#L539
[242] https://elixir.bootlin.com/linux/v6.1.1/source/net/strparser/strparser.c#L3
[243] https://www.kernel.org/doc/html/v5.10/networking/strparser.html
[244] https://lwn.net/Articles/695982/
[245] https://www.kernel.org/doc/html/latest/networking/kcm.html
[246] https://zhuanlan.zhihu.com/p/543663512
[247] https://github.com/ktls/af_ktls
[248] https://docs.kernel.org/networking/tls-offload.html

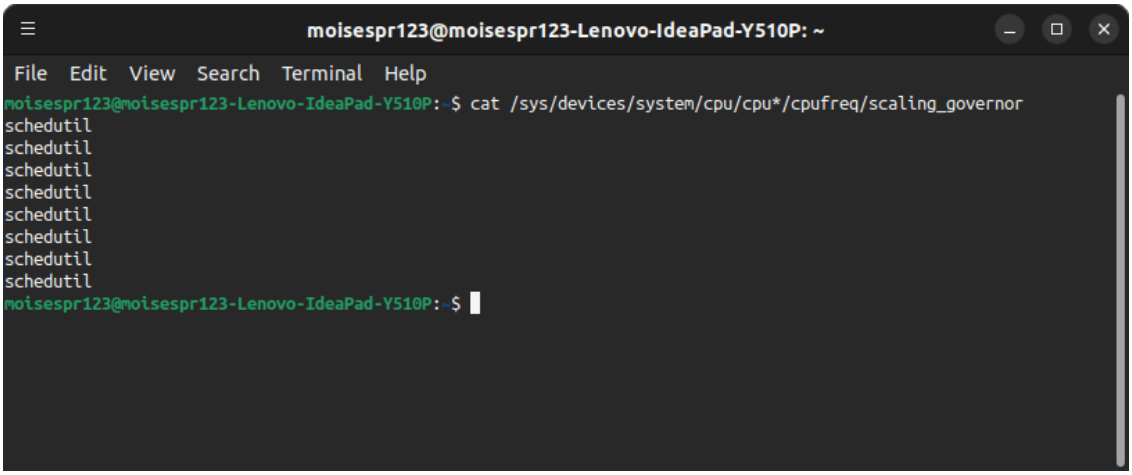
devfreq_wq

"devfreq_wq" is a kernel thread which is based on a freezable workqueue[249]. It is part of the Generic Dynamic Voltage and Frequency Scaling (DVFS) Framework for Non-CPU devices[250].

Overall, DVFS enables Linux to scale the CPU frequency in order to minimize the power usage. It is mostly done when the full performance of the CPU is not needed. By using DVFS the system can set min/max CPU frequency. There is also the ability to set a "scaling governor" which monitors the performance requirements and decides what CPU frequency to use each time[251].

Moreover, based on the Linux documentation there are 6 governors: "Performance", "Powersave", "Userspace", "Ondemand", "Conservative" and "Schedutil"[252]. We can also develop our own governor as a kernel module, we just need to register it using the function "cpufreq_register_governor"[253].

Lastly, we can use the sysfs filesystem to configure/read information regarding "cpufeq". An example of a file path for the first cpu is "/sys/devices/system/cpu/cpu0/cpufreq/" (if sysfs is mounted at "/sys"). It might contain the information like (but not limited to): current frequency of the CPU, the time it takes the CPU to switch frequencies (in nanosecs) and more[254]. An example of reading the current configure governor is shown below[255].

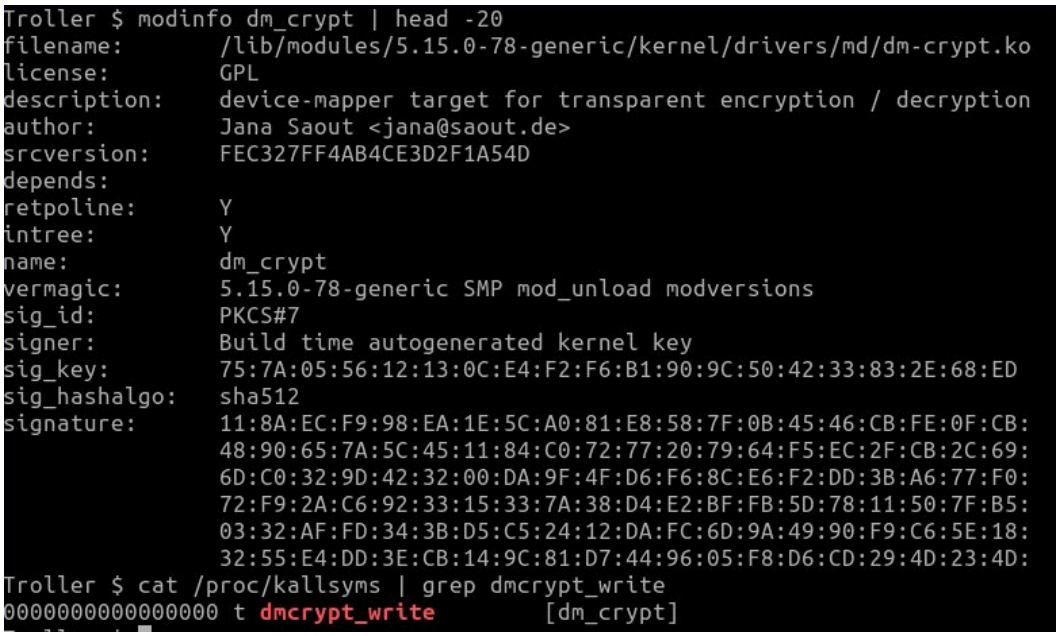


[249] https://elixir.bootlin.com/linux/v6.2.5/source/drivers/devfreq/devfreq.c#L1997
[250] https://elixir.bootlin.com/linux/v6.2.5/source/drivers/devfreq/devfreq.c#L3
[251] https://wiki.somlabs.com/index.php/How_to_scale_CPU_frequency_with_DVFS_framework
[252] https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt
[253] https://elixir.bootlin.com/linux/v6.5-rc2/source/drivers/cpufreq/cpufreq.c#L2443
[254] https://www.kernel.org/doc/Documentation/cpu-freq/user-guide.txt
[255] https://moisescardona.me/changing-the-cpu-governor-to-performance-in-linux/

dmcrypt_write (Device Mapper for Transparent Encryption/Decryption)

“dmcrypt_write” is a kernel thread which is created using the “kthread_run” function[256]. The name of the kernel thread is in the pattern of "dmcrypt_write/%s", where the added string represents the device name.Overall, “dm-crypt” is a device-mapper target[257] supported from kernel version 2.6.4[258]. It is responsible for transparent (aka real-time/on-the-fly encryption) block device encryption while using the kernel crypto API[259] .

This means the data is encrypted/decrypted while it is read/written. To enable the “dm-crypt” support we need to enable “CONFIG_DM_CRYPT” in the compilation config of the kernel[260]. Moreover, the function that is executed as part of the kernel thread is “dmcrypt_write” function[261]. This function is part of the kernel module “dm_crypt” - as shown in the screenshot below. We can use “modinfo dm_crypt” for more information, also shown in the screenshot below.



[256] https://elixir.bootlin.com/linux/v6.5-rc3/source/drivers/md/dm-crypt.c#L3388
[257] https://elixir.bootlin.com/linux/v6.5-rc3/source/drivers/md/dm-crypt.c#L3689
[258] https://elixir.bootlin.com/linux/v2.6.4/source/drivers/md/dm-crypt.c
[259] https://gitlab.com/cryptsetup/cryptsetup/-/wikis/DMCrypt
[260] https://elixir.bootlin.com/linux/v6.5-rc3/source/drivers/md/Makefile#L59
[261] https://elixir.bootlin.com/linux/v6.5-rc3/source/drivers/md/dm-crypt.c#L1922

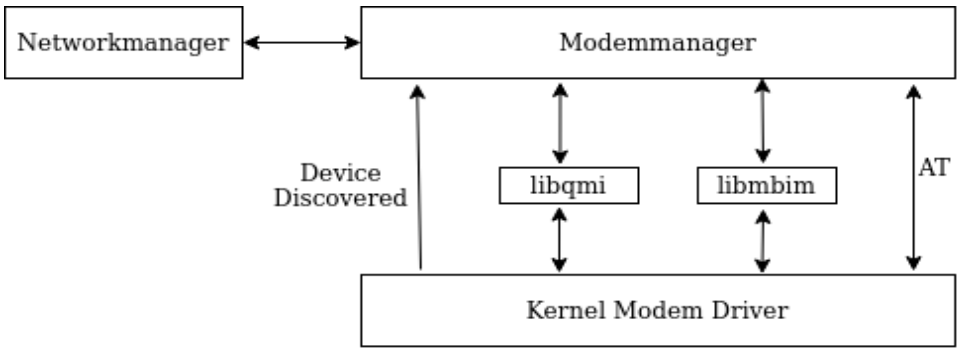
ModemManager (Modem Management Daemon)

“ModemManager” is an ELF binary located by default at “/usr/sbin/ModemManager” which is used to provide a unified high level API for communication with mobile broadband modems[262]. Alos, it is started by PID 1 (init/systemd) with the permission of the root user.

Overall, it is a DBus-powered[263] Linux daemon which acts as a standard RIL (Radio Interface Layer). “ModemManager” can be used by different connection managers (think about “NetworkManager” for example). Moreover, if we want to control and manage “ModemManager” we can use the CLI tool “mmcli”. By using it we can list all available modems, connect to a modem, get/set properties of the modem and more[264].

Thus, we can summarize “ModemManager” as a system daemon that controls WWAN (2G/3G/4G/5G) devices and connections. It is the default mobile broadband management system in most Linux distributions (like Ubuntu, Debian, Fedora and Arch). By the way, it is also used by routers running OpenWRT[265].

It is important to understand that “ModemManager” leverages “libqmi”[266] and “libmbim”[267] to communicate over QMI (Qualcomm MSM Interface) and MBIM (Mobile Interface Broadband Model) for setting connection to to the cellular network[268]. It does not matter if the modem is builtin, USB connected or bluetooth-paired. A diagram of the architecture is shown below. Lastly, if we want to go over the source code on “ModemManager” or contribute we can use its repo[269]. I also suggest going over the documentation site of “ModemManager” and the relevant libraries: libmbim, libqmi and libqrtr-glib[270].



[262] https://manpages.ubuntu.com/manpages/trusty/man8/ModemManager.8.html
[263] https://www.freedesktop.org/software/ModemManager/api/latest/
[264] https://manpages.ubuntu.com/manpages/trusty/man8/mmcli.8.html
[265] https://modemmanager.org/
[266] https://github.com/linux-mobile-broadband/libqmi
[267] https://github.com/linux-mobile-broadband/libmbim
[268] https://developer.toradex.com/software/connectivity/modem-support/
[269] https://gitlab.freedesktop.org/mobile-broadband/ModemManager
[270] https://modemmanager.org/docs/

kerneloops

"kerneloops" is an ELF binary located "/usr/sbin/kerneloops" (Ubuntu for example). It is used to collect kernel crash information (as part of a kernel oops) and submit them to kerneloops.org[271]. An example of such oops is shown in the screenshot below[272]. By the way, they are also known as "soft panic"[273].

Overall, a kernel oops is a serious but non-fatal error in the Linux kernel. It is a way for the kernel to signal that it has found a problem that could potentially cause the system to crash. However, the kernel will continue to run after an oops, although it may be unstable and can lead to a kernel panic. This helps in debugging the error in order to find a solution for the problem[274].

Moreover, if we want to debug the kernel with gdb it is suggested to compile it with "CONFIG_DEBUG_INFO" enabled, which causes the kernel to be built with full debugging information[275]. Also, I recommend also enabling "CONFIG_FRAME_POINTER", which gives very useful debugging information in case of kernel bugs - precise oopses/stacktraces/warnings[276].

Lastly, there is also a setting called "oops_limit" which states after what number of oops should cause a panic. The default value by the way is 10000[277].

```
[12710.153112] oops init (level = 1)
[12710.153115] triggering oops via BUG()
[12710.153127] ------------[ cut here ]------------
[12710.153128] kernel BUG at /home/duck/Articles/linuxoops/oops.c:17!
[12710.153132] invalid opcode: 0000 [#1] PREEMPT SMP PTI
[12710.153748] CPU: 0 PID: 5531 Comm: insmod
[12710.156191] RSP: 0018:ffffb41340e6fdd8 EFLAGS: 00010246
[12710.156849] RAX: 0000000000000019 RBX: ffffffffc1015040 RCX: 0000000000000000
[12710.157513] RDX: 0000000000000000 RSI: ffffffff83bc9d39 RDI: 00000000ffffffff
[12710.158171] RBP: ffff8d6101bd1d50 R08: 0000000000000000 R09: ffffb41340e6fc90
[12710.158826] R10: 0000000000000003 R11: ffffffff83f3d1e8 R12: ffffb41340e6fde0
[12710.159483] R13: 0000000000000000 R14: 0000000000000000 R15: 0000000000000000
[12710.160143] FS:  00007f6c290b31c0(0000) GS:ffff8d6411a00000(0000) knlGS:00000
[12710.160820] CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
[12710.161478] CR2: 00000000004134f0 CR3: 000000018be34005 CR4: 0000000003706f0
[12710.162156] DR0: 0000000000000000 DR1: 0000000000000000 DR2: 0000000000000000
[12710.162824] DR3: 0000000000000000 DR6: 00000000fffe0ff0 DR7: 0000000000000400
[12710.163474] Call Trace:
[12710.164129]  <TASK>
[12710.164779]  do_one_initcall+0x56/0x230
[12710.165424]  do_init_module+0x4a/0x210
[12710.166050]  __do_sys_finit_module+0x9e/0xf0
[12710.166711]  do_syscall_64+0x37/0x90
```

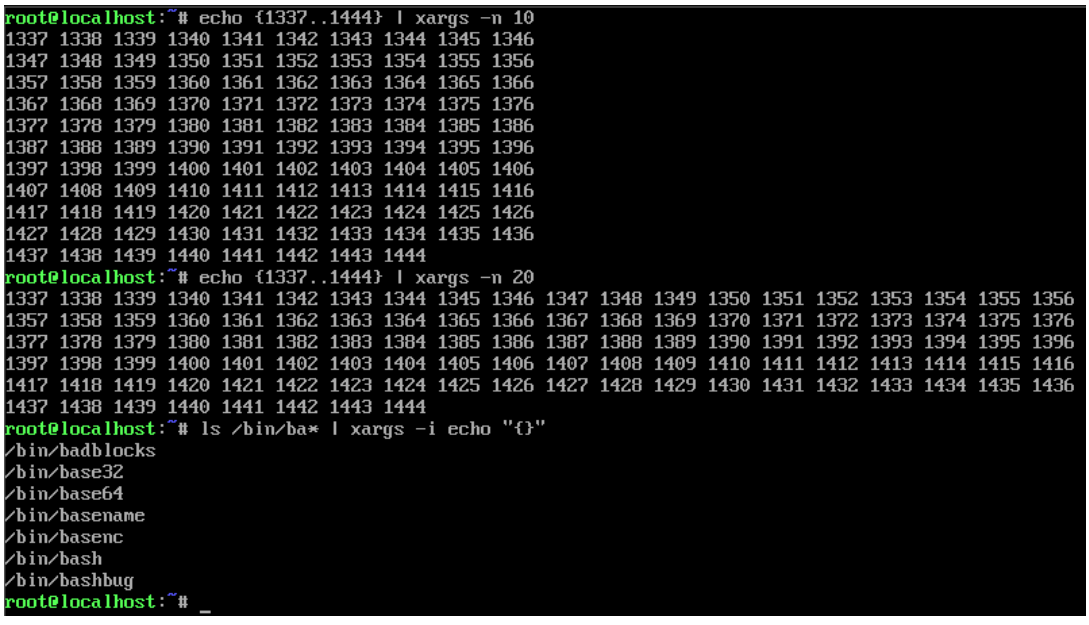
[271] https://linux.die.net/man/8/kerneloops
[272] https://nakedsecurity.sophos.com/2023/03/13/linux-gets-double-quick-double-update-to-fix-kernel-oops/
[273] https://www.opensourceforu.com/2011/01/understanding-a-kernel-oops/
[274] https://en.wikipedia.org/wiki/Linux_kernel_oops
[275] https://www.oreilly.com/library/view/linux-device-drivers/0596005903/ch04.html
[276] https://cateee.net/lkddb/web-lkddb/FRAME_POINTER.html
[277] https://docs.kernel.org/admin-guide/sysctl/kernel.html#oops-limit

xargs (Extended Arguments)

“xargs” is an ELF file which is located at “/bin/xargs”, by the way “/bin” is usually a symbolic link to “/usr/bin” so we can find the binary at “/usr/bin/xargs”. It is used mainly for building commands based on the given standard input (the can be the standard output of a different command) and executing them[278] - as shown in the screenshot below.

Overall, the default delimiter of xargs is blanks. We can override this behavior using “--delimiter” or “-d” switches. Also, “xargs” can read the input from a file using the “-a” or “--arg-file” switches. There is also the ability to limit the number of arguments (“-n” or “--max-args”), number of lines (“-l” or “--max-lines”) or the number of processes at a time (“-P” or “--max-procs”). If the value is more than the default 1 the command executed should handle race conditions, it is not done by xargs[279].

Moreover, we can control the number of arguments at a to pass as input for “xargs” - as shown in the screenshot below. I suggest going over different xargs examples for better understanding its usage[280]. Lastly, there are different implementations for xargs two examples are Apple’s one[281] and the busybox one[282].



[278] https://www.wikiwand.com/en/Xargs
[279] https://man7.org/linux/man-pages/man1/xargs.1.html
[280] https://www.tecmint.com/xargs-command-examples/
[281] https://opensource.apple.com/source/shell_cmds/shell_cmds-149/xargs/
[282] https://github.com/josefbacik/busybox/blob/master/findutils/xargs.c

cpp (The C Preprocessor)

“cpp” is an ELF file which is the C preprocessor. It is mostly located at “/bin/cpp” or “/usr/bin/cpp” (where “/bin” is usually a symbolic link to “/usr/bin”). In some distributions like Ubuntu “/bin” is a symbolic link to “/usr/bin”. “cpp” is a macro processor that is used by the compiler in order to transform our code before compilation. In general, macros are abbreviations for longer contracts. We use it when writing code in C/C++/Objective-C[283].

Moreover, the macros’ transformations can be the inclusion of header files, macro expansions and more. Header files are included using “#include” while defining a macro is done using “#define”. Also, we can use the preprocessor to instruct whether to include/or not a block of code as part of the compilation phase[284]. This is do using “ifdef”, “#ifndef”,”#define”,”#else” and “#endif”[285].

Lastly, we can say that the preprocessor prepares the source code for the compilation phase by performing different transformations - as shown in the screenshot below (we can see the replacement of the macro name with a string and the removal of a function).

```
root@localhost:/troller# cat  troller.c


#define TROLLER_STRING "Tr0L1Er"

#ifdef BLA
void troller_func()
{
        return 2222;
}
#endif

void main()
{

        char[]=TROLLER_STRING;
        return;

}
root@localhost:/troller# cpp troller.c
# 0 "troller.c"
# 0 "<built-in>"
# 0 "<command-line>"
# 1 "/usr/include/stdc-predef.h" 1 3 4
# 0 "<command-line>" 2
# 1 "troller.c"
# 12 "troller.c"
void main()
{

 char[]="Tr0L1Er";
 return;

}
```

[283] https://linux.die.net/man/1/cpp
[284] https://www.programiz.com/c-programming/c-preprocessor-macros
[285] https://www.cprogramming.com/reference/preprocessor/ifndef.html

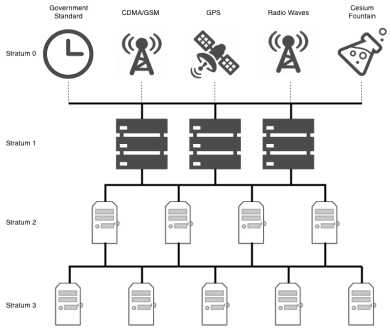
ntpd (Network Time Protocol Daemon)

“ntpd” is an ELF located by default at “/usr/bin/ntpd”. It is used to ensure that the system time is synchronized with standard time servers. It fully implements the “Network Time Protocol” (NTP) version 4, while also maintaining compatibility with older versions. “ntpd” performs most calculations using 64-bit floating-point arithmetic, only resorting to 64-bit fixed-point operations when necessary to maintain the highest possible precision, which is around 232 picoseconds[286].

Also, while such precision is not attainable with ordinary computers and networks today, it may be necessary in the future with gigahertz CPU clocks and gigabit LANs. “ntpd” reads its configuration from “/etc/ntp.conf” at startup. It uses the configuration in order to define the synchronization sources[287]. Overall, the “Network Time Protocol” was developed in 1981 by David Mills (professor at the University of Delaware). It was designed to be highly fault-tolerant/scalable, while supporting time synchronization. It is based on a Client/Server architecture using UDP with a default port number of 123[288].

Moreover, there is an open source implementation of NTP from the University of Delaware[289]. You can also check out a port of it for the Windows operating system[290]. There are two utilities that can be used when “ntpd” is running for tasks like troubleshooting/configuration/monitoring: “ntpq”[291] “ntpmon”[292]. Also, “Stratum” covers the accuracy of the time source.

Lastly, “Stratum 0” sources are the most accurate time sources, such as GPS, Cesium clocks, or cell networks. “Stratum 1” sources are systems that get their time from “Stratum 0” sources. “Stratum 2” sources get their time from “Stratum 1” sources, and so on. The lower the stratum number, the more accurate the time source. “Stratum 16” represents an unsynchronized clock, which is not reliable - as shown in the NTP stratum hierarchy diagram below[293].



[286] https://linux.die.net/man/8/ntpd
[287] https://www.mankier.com/8/ntpd
[288] https://www.techtarget.com/searchnetworking/definition/Network-Time-Protocol
[289] http://www.ntp.org/
[290] https://www.meinbergglobal.com/english/sw/ntp.htm
[291] https://www.mankier.com/1/ntpq
[292] https://www.mankier.com/1/ntpmon
[293] https://chrisshort.net/ntp-i-need-you-to-go-ahead-and-love-it/

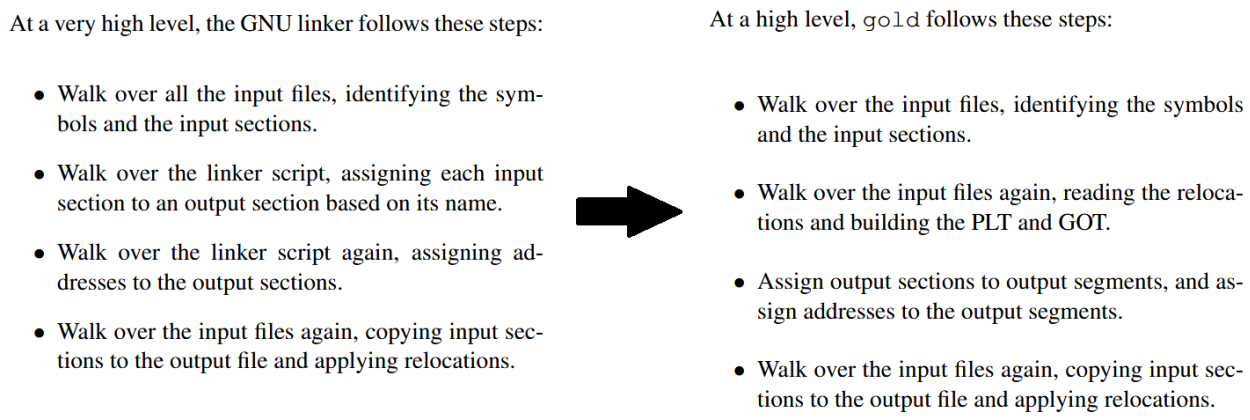
gold (The GNU ELF Linker)

“gold” is a symbolic link to an ELF file which is the GNU ELF linker. As an example under Ubuntu it points to x86_x64-linux-gnu-gold which can be a symbolic link to x86_x64-linux-ld.gold. “gold” is mostly located at “/bin/gold” or “/usr/bin/gold”. In some distributions like Ubuntu “/bin” is a symbolic link to “/usr/bin”. We can also find “gold” located at “/usr/bin/ld.golf”. It was developed by Ian Lance Taylor and other members of Google, which measured it 5 times faster than the old GNU linker when linking C++ applications[294].

Moreover, “gold” today is part of GNU binutils[295]. As opposed to the GNU linker it does not use Binary File Descriptor library (BFD), which limits “gold” to process only ELF files but results in cleaner and faster implementations without the need for an abstraction layer[296].

Thus, we can specify “gold” as the the linker to use by setting LD as part of a makefile, setting the LD environment variable[297] or by using the “-fuse-ld=gold” option of gcc[298].

Lastly, we can also go over the paper that introduced gold titled “A new ELF Linker” by Ian Lance as part of the “2008 GCC Developers’ Summit”. The main difference between the GNU linker is that “gold” has a second walk over the input file to read the relocations, and the omission of the linker script - as shown in the image below taken from the paper about “gold”[299]. “gold” (sometimes also “ld.gold”) can take different command line arguments which can affect the linking processes[300].



[294] https://opensource.googleblog.com/2008/04/gold-google-releases-new-and-improved.html
[295] https://sourceware.org/binutils/
[296] https://en.wikipedia.org/wiki/Gold_(linker)
[297] https://en.wikipedia.org/wiki/Gold_(linker)
[298] https://man7.org/linux/man-pages/man1/gcc.1.html
[299] https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/34417.pdf
[300] https://www.mankier.com/1/ld.gold

strace (System Call Tracing)

"strace" (System Call Tracing) is an ELF file which is located at "/bin/strace", by the way "/bin" is usually a symbolic link to "/usr/bin" so we can find the binary at "/usr/bin/strace". It can be leveraged for: tracing\monitoring\diagnosing\debugging Linux's tasks (processes\threads)[301]. It is used for displaying information about system calls[302] executed by a specific application. It is important to understand that strace is dependent on the "ptrace" syscall for performing its job[303].

Overall, by default every line in the output of the strace command (which writes to standard error) contains: the system call name, its arguments in parentheses and its return value of the syscall. In case the syscall is not identified by strace it is displayed as "syscall_0x{NUM}", where {NUM} is the syscall number in hex. Thus, we can say strace is used to trace system calls and signals until the monitored application exits[304].

Lastly, it is important to know that using the different switches of strace we can use more advanced features such as (but not limited to): attach to an already running process (-p), print paths and more information associated with file descriptors (-y), filter by type of syscall (-e trace={SYSCALL}), trace only syscalls accessing a specific path (-P), perform syscall fault injection (-e fault={SYSCALL}), dump all data read\written to file descriptors in hex\ascii (-ewrite=1) and count time\calls\errors for every syscall (-c) - as shown in the screenshot below[305]. For more information I suggest going over the source code as part of strace's GitHub repository[306].

```
$ strace -c ls > /dev/null
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 89.76    0.008016           4      1912           getdents
  8.71    0.000778           0     11778           lstat
  0.81    0.000072           0      8894           write
  0.60    0.000054           0       943           open
  0.11    0.000010           0       942           close
  0.00    0.000000           0         1           read
  0.00    0.000000           0       944           fstat
  0.00    0.000000           0         8           mmap
  0.00    0.000000           0         4           mprotect
  0.00    0.000000           0         1           munmap
  0.00    0.000000           0         7           brk
  0.00    0.000000           0         3         3 access
  0.00    0.000000           0         1           execve
  0.00    0.000000           0         1           sysinfo
  0.00    0.000000           0         1           arch_prctl
------ ----------- ----------- --------- --------- ----------------
100.00    0.008930                 25440         3 total
```

[301] https://www.geeksforgeeks.org/linux-unix/strace-command-in-linux-with-examples/
[302] https://medium.com/@boutnaru/the-linux-concept-journey-syscalls-system-calls-efcd7703e072
[303] https://man7.org/linux/man-pages/man2/ptrace.2.html
[304] https://man7.org/linux/man-pages/man1/strace.1.html
[305] https://strace.io/
[306] https://github.com/strace/strace

ltrace (Library Call Tracer)

“ltrace” (Library Call Tracing) is an ELF file which is located at “/bin/ltrace”, by the way “/bin” is usually a symbolic link to “/usr/bin” so we can find the binary at “/usr/bin/ltrace”. It is a CLI utility which can be used to monitor\trace a specific program until it exists. Its main goal is to tap and log the dynamic library calls which are invoked by the executed process. Also, it can record the signals which are received by that process. By the way, it can also display syscalls as well as library calls using the “-S” switch[307]. We can also filter the output of the command based on specific library functions and display statistics on the function used by a traced task - as shown in the screenshot below[308].

Overall, as with strace utility[309] also ltrace leverages the “ptrace” syscall[310] in order to set breakpoints in the task (process\thread) being traced[311]. From the source code of ltrace we deduct the following flow: starting a process\attaching to a running process, setting breakpoint(s) on library calls\syscalls and decoding library\syscall name and arguments[312].

Lastly, it is important to understand that ltrace is dependent that the binary being traced is compiled and linked with lazy loading (gcc -z lazy {SOURCE_FILE} -o {OUTPUT_FILE}), if not the library information is not displayed[313].

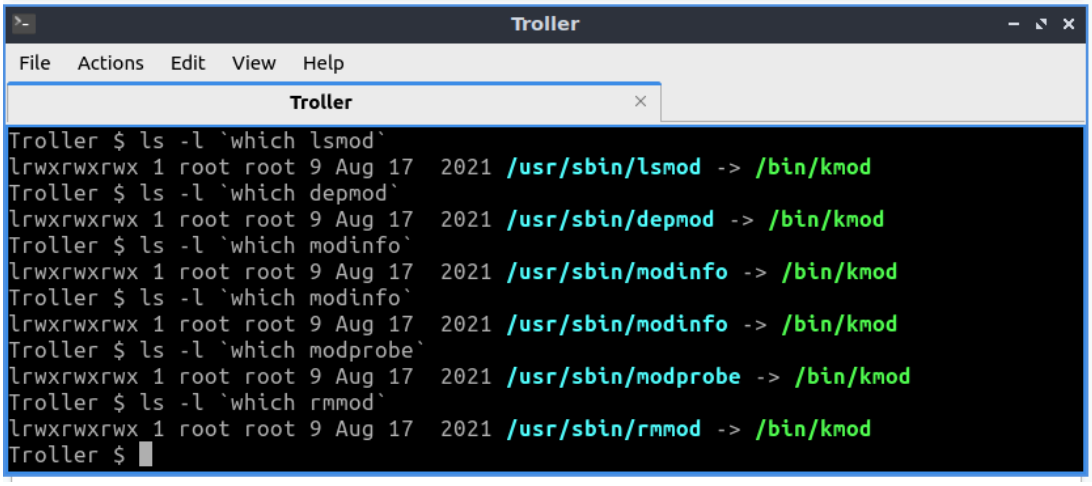
```
michael-nb ~ # ltrace -c -p 5092
^C% time     seconds  usecs/call     calls      function
------ ----------- ----------- --------- --------------------
 35.88    2.090027        2163       966 epoll_wait
 18.89    1.100212         742      1482 clock_gettime
 12.43    0.723912         750       965 __errno_location
  8.26    0.481053        1886       255 syscall
  8.22    0.478858         913       524 pthread_mutex_lock
  8.19    0.477084         910       524 pthread_mutex_unlock
  5.32    0.309916         895       346 pthread_getspecific
  0.96    0.056133         891        63 _ZSt18_Rb_tree_incrementPKSt18_Rb_tree_node_base
  0.72    0.041822         929        45 _ZSt29_Rb_tree_insert_and_rebalancebPSt18_Rb_tree_node_baseS0_RS_
  0.27    0.015894         836        19 memchr
  0.19    0.011085         791        14 memcpy
  0.08    0.004682        1170         4 floor
  0.06    0.003546         886         4 strncmp
  0.05    0.003056         764         4 ceil
  0.05    0.003053         763         4 _ZNSsC1ERKSs
  0.05    0.002725         681         4 strlen
  0.05    0.002714         678         4 memmove
  0.04    0.002168        2168         1 write
  0.04    0.002122        1061         2 ceilf
  0.03    0.001897        1897         1 send
  0.03    0.001881         940         2 _ZNSs4_Rep10_M_destroyERKSaIcE
  0.03    0.001703        1703         1 _ZNSt8__detail15_List_node_base9_M_unhookEv
  0.02    0.001296         648         2 gettimeofday
  0.02    0.001295         647         2 memset
  0.02    0.001236        1236         1 read
  0.02    0.001217        1217         1 _ZNSsC1EPKcmRKSaIcE
  0.02    0.001179        1179         1 pthread_cond_destroy
  0.01    0.000718         718         1 memcmp
  0.01    0.000678         678         1 pthread_mutex_destroy
  0.01    0.000574         574         1 _ZNSs6assignEPKcm
  0.01    0.000550         550         1 _ZSt18_Rb_tree_incrementPSt18_Rb_tree_node_base
------ ----------- ----------- --------- --------------------
100.00    5.824286                  5245 total
michael-nb ~ #                                    linux-audit.com
```

[307] https://man7.org/linux/man-pages/man1/ltrace.1.html
[308] https://linux-audit.com/monitor-file-access-by-linux-processes/
[309] https://medium.com/@boutnaru/the-linux-concept-journey-strace-system-call-tracing-11354f94b00d
[310] https://linux.die.net/man/2/ptrace
[311] https://www.kernel.org/doc/ols/2007/ols2007v1-pages-41-52.pdf
[312] https://gitlab.com/cespedes/ltrace
[313] https://medium.com/@boutnaru/linux-why-ltrace-does-not-work-on-new-versions-of-ubuntu-b2d89b55b70f

kmod (Linux Kernel Module Handling)

“kmod” is an ELF file which is located at “/bin/kmod”, by the way “/bin” is usually a symbolic link to “/usr/bin” so we can find the binary at “/usr/bin/kmod”. It is used mainly for managing Linux kernel modules[314]. It is important to understand that “kmod” is a multi-call binary that implements utilities used to manage kernel modules. Hence, most users will not run it directly[315].

Overall, by using “kmod” we can load/remove/insert/show information/resolve dependencies of kernel modules[316]. “kmod” is developed by Lucas De Marchi[317]. Lastly, the most well known utilities used to control kernel modules are just symbolic links to “kmod” examples are: lsmod, rmmod, modinfo, modprobe and depmod - as shown in the screenshot below.



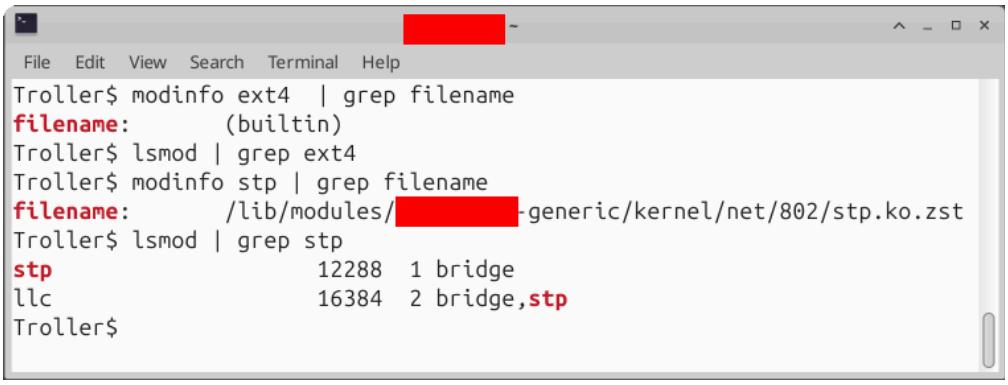
[314] https://linux-kernel-labs.github.io/refs/heads/master/labs/kernel_modules.html
[315] https://man7.org/linux/man-pages/man8/kmod.8.html
[316] https://linuxhint.com/linux-kmod-command/
[317] https://github.com/kmod-project/kmod

lsmod (List Kernel Modules)

“lsmod” is a symbolic link[318] to “/sbin/lsmod” or “/usr/sbin/lsmod”. It references the ELF binary “kmod” which is located at “/bin/kmod” or “/usr/bin/kmod”[319]. lsmod is used for showing the status of LKMs[320]. This is done by parsing the content of “/proc/modules”[321].

Overall, it is important to understand that in case of builtin kernel modules[322] which are compiled as part of the kernel itself, they won’t appear in the output of lsmod - as shown in the screenshot below.

Lastly, lsmod also reads information form “/sys/module/*”. For example in the function “kmod_module_get_size”[323] which is called from “kmod_list_foreach”[324]. The output of lsmod provides the following data: “Module” (the name of the module), “Size” (in bytes and not memory in use) and “Used By” (the number of times it is used by running programs). Also, the list on the right is the name(s) of the module(s) which refer to the current one described in the specific row[325] - as shown below. It is important to understand that lsmod could be incomplete in regards to the used by\reference counters[326].



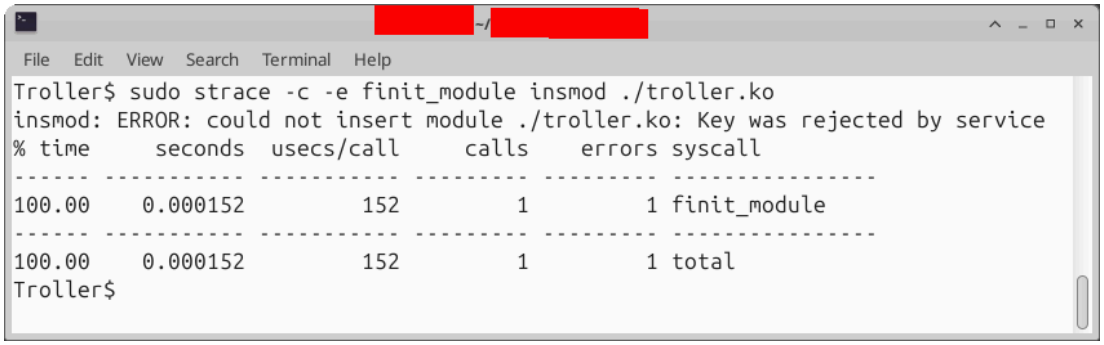
[318] https://medium.com/@boutnaru/the-linux-concept-journey-linux-file-types-4cb622887331
[319] https://medium.com/@boutnaru/the-linux-process-journey-kmod-linux-kernel-module-handling-8ec844216436
[320] https://medium.com/@boutnaru/the-linux-concept-journey-loadable-kernel-module-lkm-5eaa4db346a1
[321] https://man7.org/linux/man-pages/man8/lsmod.8.html
[322] https://medium.com/@boutnaru/the-linux-concept-journey-builtin-kernel-modules-0b9fd2bc524c
[323] https://github.com/kmod-project/kmod/blob/master/libkmod/libkmod-module.c#L1482
[324] https://github.com/kmod-project/kmod/blob/master/tools/lsmod.c#L98C2-L98C19
[325] https://en.wikipedia.org/wiki/Lsmod
[326] https://stackoverflow.com/questions/4073152/how-to-get-complete-dependency-list-of-kernel-modules-at-runtime/4238010#4238010

insmod (Insert Kernel Module)

“insmod” is a symbolic link[327] to that is located at “/sbin/insmod” or “/usr/sbin/insmod”. It references the ELF binary “kmod” which is located at “/bin/kmod” or “/usr/bin/kmod”[328]. insmod is used for inserting a LKM[329] into the Linux kernel. We can use the “-f” (“--force”) switch of “insmod” for ignoring the module version and vermagic fields when loading and “-s” (“--syslog”) for sending errors to syslog instead of standard error[330].

Overall, in order to load a kernel module we can use the “init_module” syscall or the “finit_module” syscall. The difference between the two is that “finit_module” reads the module to be loaded from a file descriptor (fd) while “init_module” gets a pointer to a buffer containing the binary image to be loaded[331]. The current version of insmod uses the “finit_module” syscall in order to load a LKM - as shown in the screenshot below (the syscall fails because the kernel module is not signed). I suggest going over the code for more details[332].

Lastly, it is important to understand that insmod is intended to insert\load a single kernel module from any location. This is due to the fact it does not manage dependencies and does not read modules from a specific location[333]. In case of any errors it's best to check out “dmesg”[334]. By the way, insmod causes the module’s init function of the LKM[335].



[327] https://medium.com/@boutnaru/the-linux-concept-journey-linux-file-types-4cb622887331
[328] https://medium.com/@boutnaru/the-linux-process-journey-kmod-linux-kernel-module-handling-8ec844216436
[329] https://medium.com/@boutnaru/the-linux-concept-journey-loadable-kernel-module-lkm-5eaa4db346a1
[330] https://man7.org/linux/man-pages/man8/insmod.8.html
[331] https://linux.die.net/man/2/finit_module
[332] https://github.com/kmod-project/kmod/blob/master/tools/insmod.c
[333] https://www.computerhope.com/unix/insmod.htm
[334] https://medium.com/@boutnaru/the-linux-process-journey-dmesg-print-control-the-kernel-ring-buffer-dc78abeb87b7
[335] https://slideplayer.com/slide/15437239/

rmmod (Remove Kernel Module)

"rmmod" is a symbolic link[336] to that is located at "/sbin/rmmod" or "/usr/sbin/rmmod". It references the ELF binary "kmod" which is located at "/bin/kmod" or "/usr/bin/kmod"[337]. rmmod is used for removing a LKM[338] into the Linux kernel. We can use the "-f" ("--force") switch of "rmmod" for removing modules which are being used\which are not designed to be removed\have been marked as unsafe. This is relevant only if the kernel was compiled with "CONFIG_MODULE_FORCE_UNLOAD"[339]. Also, the "-s" switch ("--syslog") is used for sending errors to syslog instead of standard error[340].

Overall, in order to unload\remove a kernel module we can use the "delete_module" syscall. After glibc version 2.23 there is no wrapper for "delete_module"[341]. Hence, we need to use the "syscall" for indirect syscall invocation[342]. I suggest going over the code for more details[343]. Using rmmod causes the module's exit function to be called before unloading the LKM[344] - as shown in the screenshot below[345].

Lastly, until kernel 2.4 the "delete_module" syscall[346] took only one argument[347]. As of the newer kernel it takes two arguments[348]. By the way, the syscall's implementation calls "free_module"[349] after performing different checks and operations. For more information I suggest going over the "free_module" implementation[350].

```
debian@npi:/mnt$ ls
platform_device.ko  platform_driver.ko
debian@npi:/mnt$ sudo insmod platform_device.ko
[ 1360.553491] led_device_init
debian@npi:/mnt$ sudo insmod platform_driver.ko
[ 1367.615716] led_driver_init
[ 1367.619280] led_probe
[ 1367.621822] led major:244
debian@npi:/mnt$ ls /dev/my_led
/dev/my_led
debian@npi:/mnt$ sudo sh -c "echo on > /dev/my_led"
debian@npi:/mnt$ sudo sh -c "echo off > /dev/my_led"
debian@npi:/mnt$ sudo rmmod platform_driver.ko
debian@npi:/mnt$ sudo rmmod platform_device.ko
[ 1439.791876] led device release!
debian@npi:/mnt$ ls /dev/my_led
ls: cannot access '/dev/my_led': No such file or directory
                                        CSDN @Couvrir洪荒猛兽
```

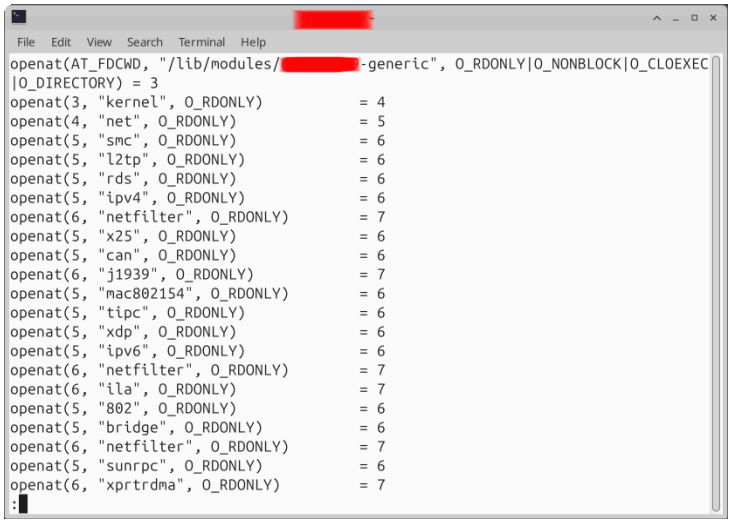
[336] https://medium.com/@boutnaru/the-linux-concept-journey-linux-file-types-4cb622887331
[337] https://medium.com/@boutnaru/the-linux-process-journey-kmod-linux-kernel-module-handling-8ec844216436
[338] https://medium.com/@boutnaru/the-linux-concept-journey-loadable-kernel-module-lkm-5eaa4db346a1
[339] https://cateee.net/lkddb/web-lkddb/MODULE_FORCE_UNLOAD.html
[340] https://man7.org/linux/man-pages/man8/insmod.8.html
[341] https://man7.org/linux/man-pages/man2/delete_module.2.html
[342] https://man7.org/linux/man-pages/man2/syscall.2.html
[343] https://github.com/kmod-project/kmod/blob/master/tools/rmmod.c
[344] https://slideplayer.com/slide/15437239/
[345] http://www.wjks.cn/news/2246.html
[346] https://linux.die.net/man/2/delete_module
[347] https://elixir.bootlin.com/linux/2.4.0/source/include/asm-arm/unistd.h#L407
[348] https://elixir.bootlin.com/linux/v6.16/source/kernel/module/main.c#L748
[349] https://elixir.bootlin.com/linux/v6.16/source/kernel/module/main.c#L817
[350] https://elixir.bootlin.com/linux/v6.16/source/kernel/module/main.c#L1367

depmod (Dependency Modules)

“depmod” (Dependency Modules) is a symbolic link[351] to that is located at “/sbin/depmod” or “/usr/sbin/depmod”. It references the ELF binary “kmod” which is located at “/bin/kmod” or “/usr/bin/kmod”[352]. “depmod” is used for generating a list of dependency descriptions of kernel modules and its associated map files. This is done by going over the “/lib/modules/{kernel-release}” and creating the “modules.dep” file based on the symbols present in the set of modules[353] - as shown in the output of the strace utility below.

Overall, “depmod” is also leveraged for building mappings between hardware identifiers and the modules which handle them. Hence, it can be used for finding the correct module to load when a specific hardware is identified[354]. “depmod” has different switches like (but not limited to): probing all modules and building the dependency map (“-a”/”-all”), changing input/output directories, report unresolved symbols (“-e”/”-erryms”) and report on symbol version mismatches (“-E Module.symvers”\”--symvers=Module.symvers”). For more information I suggest going over “depmod” documentation[355].

Lastly, it is important to understand that beside creating “modules.dep” also a binary hashed version file named “modules.dep.bin” is created in the same directory[356]. For better understanding I suggest going over the source code of “depmod” as part of kmod’s Github repository[357].



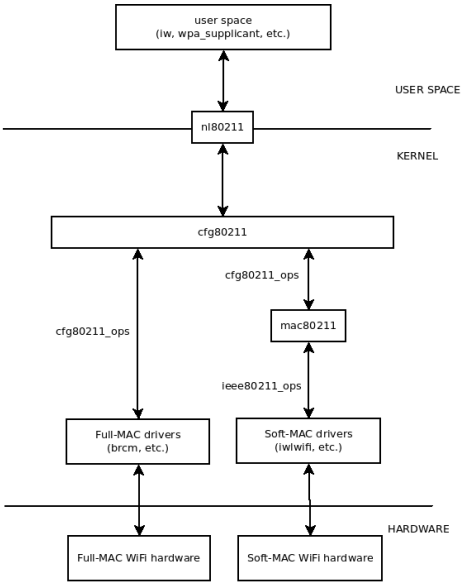
[351] https://medium.com/@boutnaru/the-linux-concept-journey-linux-file-types-4cb622887331
[352] https://medium.com/@boutnaru/the-linux-process-journey-kmod-linux-kernel-module-handling-8ec844216436
[353] https://www.geeksforgeeks.org/linux-unix/depmod-command-in-linux-with-examples/
[354] https://stackoverflow.com/questions/8697132/why-is-depmod-necessary-for-building-and-working-with-kernel-modules
[355] https://man.archlinux.org/man/depmod.8.en
[356] https://man7.org/linux/man-pages/man8/depmod.8.html
[357] https://github.com/kmod-project/kmod/blob/master/tools/depmod.c

cfg80211 (Wireless Configuration)

“cfg80211” is a Linux kernel thread which is based on an ordered workqueue[358]. It is defined as part of “/net/wireless/core.c” in the “cfg80211_init()” function, it is the part of the Linux wireless configuration interface[359]. It interfaces with “nl8021”, together they are basically a replacement of wext aka” Wireless Extensions”[360].

Overall, the “cfg802011” (subsystem for the Linux kernel regarding wireless configuration) was created by “Johannes Berg”, he also works on “mac80211” which is the Linux wireless stack[361]. “cfg80211” uses “struct cfg80211_ops” as a backend description for wireless configuration. This structure is registered by card drivers/wireless stacks for handling configuration requests on their interfaces[362].

Moreover, “cfg80211” is a thin layer between userspace and drivers/mac80211 which includes mainly sanity checks and protocol translations. We can summarize the main flows of “cfg80211” as: device registration, regulatory enforcement, station management (AP), key management (AP only), mesh management, virtual interface management and scanning[363]. Lastly, “cfg80211” interfaces with “nl80211” (as shown in the below) using netlink and not ioctl[364]. By the way, the name “nl80211” represents 802.11 netlink interface[365].



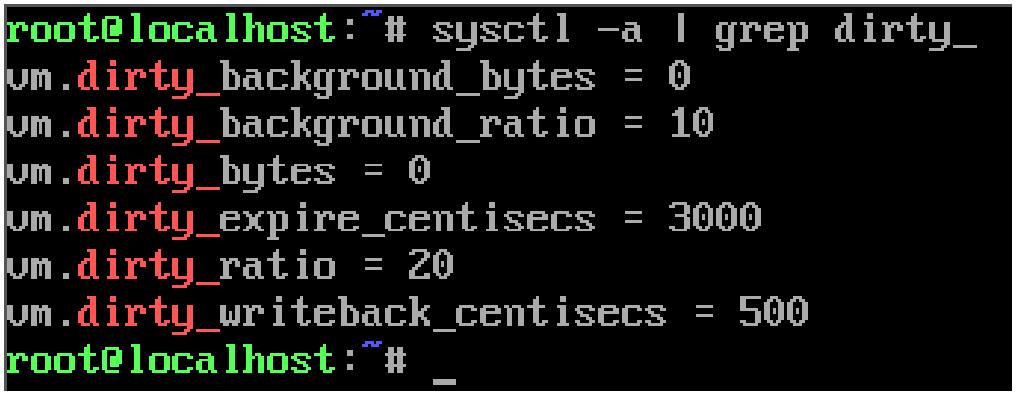
[358] https://elixir.bootlin.com/linux/v6.7.5/source/net/wireless/core.c#L1719
[359] https://elixir.bootlin.com/linux/v6.7.5/source/net/wireless/core.c#L4
[360] https://elinux.org/images/7/75/DeRosier_WirelessInterfacing.pdf
[361] https://johannes.sipsolutions.net/Projects/
[362] https://elixir.bootlin.com/linux/v6.7.5/source/include/net/cfg80211.h#L4501
[363] https://wireless.wiki.kernel.org/_media/en/developers/documentation/control.pdf
[364] https://stackoverflow.com/questions/21456235/how-do-the-nl80211-library-cfg80211-work
[365] https://elixir.bootlin.com/linux/v6.7.5/source/include/uapi/linux/nl80211.h#L4

kdmflush (Kernel Device Mapper Flush)

“kdmflush” is a kernel thread which is based on a workqueue[366]. Its name is created in the pattern of "kdmflush/%s" (where %s is the mapped device name). It is used by the “Device Mapper” (dm) in order to queue up deferred work to other context if doing them immediately so would be problematic[367] .

It is part of the “Device Mapper” framework and used in order to process deferred work hat it has queued up from other contexts where doing immediately so would be problematic[368]. Also, the kernel parameter “vm.dirty_background_ratio” controls the percentage of system memory that can be filled with “dirty” pages (memory pages not written to disk) before “kdmflush” kicks in to write them to disk[369].

Lastly, there are also other kernel parameters which affect the handling of “dirty” pages, we can check them out using the “sysctl” utility[370] - as shown in the screenshot below. By the way, “kdmflush” is created as part of the “alloc_dev” function that is responsible for allocating and initializing a blank device with a given minor[371].



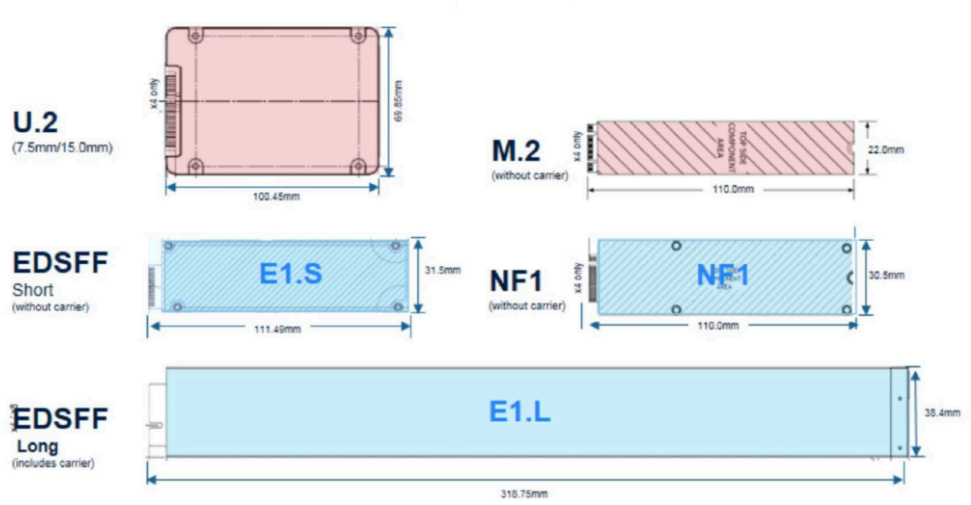
[366] https://elixir.bootlin.com/linux/v6.5-rc3/source/drivers/md/dm.c#L2131
[367] https://www.compuhoy.com/how-do-you-check-which-process-is-using-more-disk-in-linux/
[368] https://askubuntu.com/questions/986211/what-is-kdmflush
[369] https://www.cnblogs.com/cobbliu/articles/11792193.html
[370] https://linux.die.net/man/8/sysctl
[371] https://elixir.bootlin.com/linux/v6.5-rc3/source/drivers/md/dm.c#L2044

nvme-wq (Non-Volatile Memory Express Work Queue)

“nvme-wq” is a Linux kernel based on a workqueue, which is created as part of the “nvme_core_init” function in “/drivers/nvme/host/core.c”[372]. It is used for hosting NVMe related work which is not reset or delete[373]. Reset/delete are handled by other kernel threads which we are going to elaborate on in future writeups.

Overall, NVMe is a family of specifications that define how host software communicates with non-volatile memory across multiple transports (like PCI Express/RDMA/TCP). It is used as an industry standard for SSDs (solid state drives) in all form factors (U.2/M.2/AIC/EDSFF) - as shown in the diagram below[374]. By the way, NVM Express is a non-profit consortium[375].

Lastly, “nvme-wq” is used to host work such as: scanning, AEN (Asynchronous Event Notifications) handling, FW activation, periodic reconnects and keep-alive[376]. Examples of those operations being queued from the Linux source are the following functions: “nvme_queue_scan”[377], “nvme_enable_aen”[378], and “nvme_fw_act_work”[379].



[372] https://elixir.bootlin.com/linux/v6.8.6/source/drivers/nvme/host/core.c#L4831
[373] https://elixir.bootlin.com/linux/v6.8.6/source/drivers/nvme/host/core.c#L93
[374] https://venturebeat.com/business/meet-edsff-1pb-of-flash-storage-in-a-single-rack/
[375] https://nvmexpress.org/
[376] https://elixir.bootlin.com/linux/v6.8.6/source/drivers/nvme/host/core.c#L97
[377] https://elixir.bootlin.com/linux/v6.8.6/source/drivers/nvme/host/core.c#L136
[378] https://elixir.bootlin.com/linux/v6.8.6/source/drivers/nvme/host/core.c#L1682
[379] https://elixir.bootlin.com/linux/v6.8.6/source/drivers/nvme/host/core.c#L4257

] (Checking File Types and Comparing Values)

“[” is an ELF binary located at “/usr/bin/[“ (or “/bin/[“) - as shown in the screenshot below. It is an equivalent to the “test” utility[380]. Thus, it is used for checking file types and comparing values[381]. The binary is part of the “coreutils” package[382] - as shown in the screenshot below (taken from “copy.sh”, using the “Arch Linux”).

Overall, using “[“ we can check expressions, compare integers, check the type of a file[383], check if a file has been modified since it was last read, if the file exists and the user has read/execute access and more[384]. We can retrieve the result of the check using “$?” - as shown in the screenshot below.

Lastly, in some shells like bash there is also an implementation of “[“ as a builtin command[385] - as shown in the screenshot below. Also, “[“ is commonly used in scripts. For example, searching for it in scripts (using “grep.app”) yields more than 43K results[386].

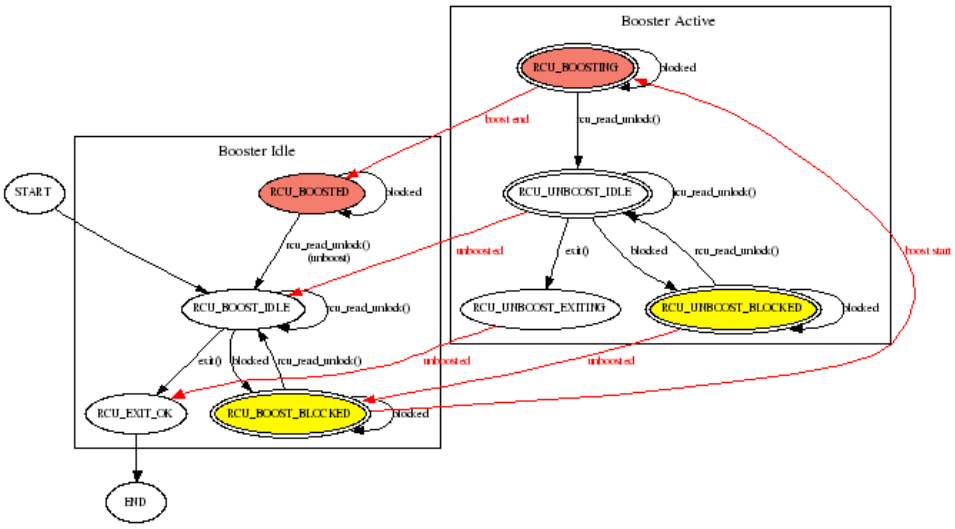
[380] https://superuser.com/questions/334549/what-is-usr-bin-and-how-do-i-use-it
[381] https://www.man7.org/linux/man-pages/man1/test.1.html
[382] https://github.com/coreutils/coreutils/blob/master/src/test.c#L36
[383] https://medium.com/@boutnaru/the-linux-concept-journey-linux-file-types-4cb622887331
[384] https://linux.die.net/man/1/test
[385] https://linux.die.net/man/1/builtins
[386] https://grep.app/search?q=%5B%20&filter[path][0]=scripts/

rcub (Read-Copy Update Boost)

“rcub” is a Linux kernel thread created as part of the “rcu_spawn_one_boost_kthread” function[387]. This function is used to create an RCU-boost kernel thread for a specific node. Due to that, the name of the kernel thread is in the format of “rcub/%d” (where %d is replaced with the node id). The kernel thread is created only for preemptible RCU[388].

Overall, RCU (Read-Copy Update) is a mutual exclusion mechanism[389]. RCU is sometimes in place of read/write locks, due to the fact that based on properties of RCU updaters can block readers. Each task has its own RCU-booster state in order to avoid failure scenarios -as shown in the diagram below[390]. We can enable RCU priority boosting using the “CONFIG_RCU_BOOST” configuration item when compiling the kernel[391].

Lastly, The “rcub” kernel thread executes the code of the “rcu_boost_kthread” function[392]. We can summarize that “rcub” is used for boosting the priority of preempted RCU readers that block the current preemptible RCU grace period for too long. It is used for preventing heavy loads from blocking RCU callback invocation for all flavors of RCU. Thus, it is relevant for working with real-time apps\heavy loads[393].



[387] https://elixir.bootlin.com/linux/v6.8.5/source/kernel/rcu/tree_plugin.h#L1202
[388] https://elixir.bootlin.com/linux/v6.8.5/source/kernel/rcu/tree_plugin.h#L1188
[389] https://medium.com/@boutnaru/linux-rcu-read-copy-update-c2793ee7a4cd
[390] https://lwn.net/Articles/220677/
[391] https://lwn.net/Articles/777214/
[392] https://elixir.bootlin.com/linux/v6.8.5/source/kernel/rcu/tree_plugin.h#L1110
[393] https://cateee.net/lkddb/web-lkddb/RCU_BOOST.html

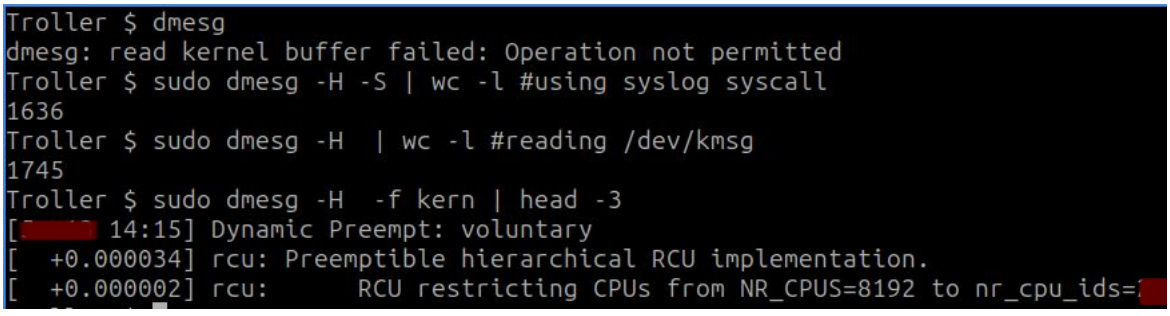
dmesg (Print/Control the Kernel Ring Buffer)

“dmesg” is an ELF binary located at “/usr/bin/dmesg“ (or “/bin/dmesg“). The Linux kernel writes different messages to a kernel ring buffer (during the boot time and while the system is running). Thus, the kernel ring buffer holds various log messages, the buffer is fixed in size so in case it is full older entries are overwritten. There are different log facilities that can write to the kernel ring buffer like (but not limited to): “kern” (kernel messages), “user” (user level message), “mail” (mail system), “daemon” (system daemons), “auth” (security/authorization messages), “syslog” - as shown in the screenshot below[394].

Overall, whether we can execute “dmesg” without root permissions depends on the “kernel.dmesg_restrict” sysctl key. The “0” value indicates there are no restrictions, while “1” restricts access to those users that have the CAP_SYSLOG capability. Also, we can use the “CONFIG_SECURITY_DMESG_RESTRICT” kernel config option for setting the default value while compiling the kernel[395].

Moreover, “dmesg” is based on reading information for the character device “/dev/kmsg”[396]. The utility can get different arguments for clearing the ring buffer after printing it (“-c”/”--read-clear”), setting the level of logging messages, enabling human readable format (“-H”/”--human”), using JSON format (“-J”/”--json”), printing human-readable timestamps (“T”/”--ctime”) and more[397].

Lastly, “dmesg” is part of the “util-linux” package, we can go over the source code of the utility in repo of the package[398]. We can also force “dmesg” to use the “syslog” syscall[399] to read kernel messages instead of using “/dev/kmsg”[400]. By the way, it does not mean we will get the same number of entries - as shown in the screenshot below.



[394] https://linuxize.com/post/dmesg-command-in-linux/
[395] https://linuxsecurity.expert/kb/sysctl/kernel_dmesg_restrict/
[396] https://www.kernel.org/doc/Documentation/ABI/testing/dev-kmsg
[397] https://man7.org/linux/man-pages/man1/dmesg.1.html
[398] https://github.com/util-linux/util-linux/blob/master/sys-utils/dmesg.c
[399] https://man7.org/linux/man-pages/man2/syslog.2.html
[400] https://github.com/util-linux/util-linux/blob/master/sys-utils/dmesg.c#L343

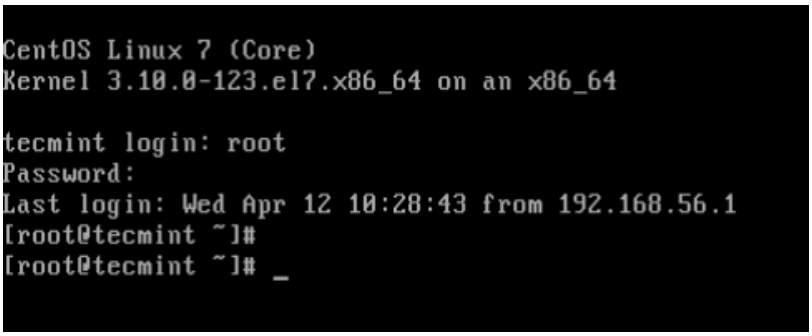
login (Begin Session on The System)

Overall, “login” is an ELF binary located by default at “/usr/bin/login” which is used to begin a session on a Linux system. It is used when signing onto a Linux system, if no arguments are passed a prompt of a user is shown[401] - as shown in the screenshot below[402].

Overall, login is part of the “util-linux” package which is a standard package that is distributed by the “Linux Kernel Organization”. It is important to understand that there are also other executables in that package like: kill, more, renice, su and more[403]. We can check the source code of “login” as part of the “util-linx” github repository[404].

Moreover, login is based on PAM (Package Authentication Modules) which provides a framework for system-wide user authentication. You can see that also using “ldd”[405] which will show libpam.so and probably also libpam_misc.so[406].

Lastly, there are different configuration files that affect the behavior of “loing” (beside PAM conf files). Among those configuration files are: “/etc/login.def”, /etc/motd, /etc/passwd and /etc/nologin - more information about them in future writeups. Also, there are logging based files which are handled by “login” such as: /var/run/utmp, /var/log/wtmp and /var/log/lastlog - more on them in future writeups[407].



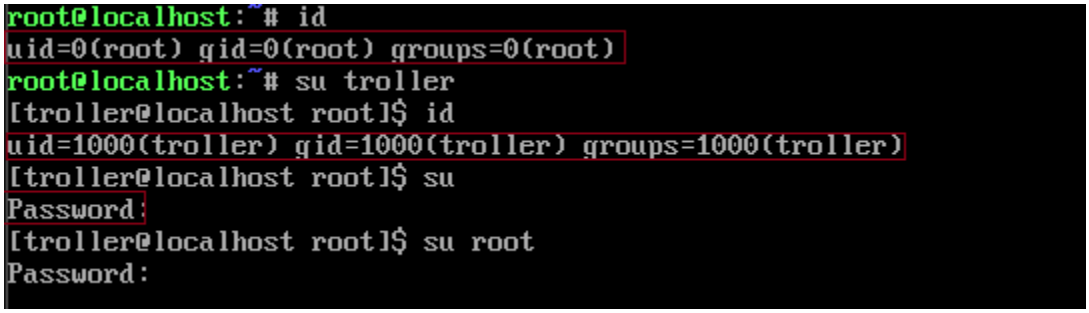
[401] https://man7.org/linux/man-pages/man1/login.1.html
[402] https://www.tecmint.com/understanding-shell-initialization-files-and-user-profiles-linux/
[403] https://en.wikipedia.org/wiki/Util-linux
[404] https://github.com/util-linux/util-linux/blob/master/login-utils/login.c
[405] https://medium.com/@boutnaru/linux-instrumentation-part-4-ldd-888502965a9b
[406] https://medium.com/@boutnaru/the-linux-security-journey-pam-pluggable-authentication-module-388496a8785c
[407] https://linux.die.net/man/1/login

su (Substitute\Switch User)

"su" is an ELF binary located at "/usr/bin/su" (or "/bin/su") and used for running a command with a substitute user\group identifier. "su" stands for "Substitute User" or "Switch User"[408]. "su" is part of the "util-linux" package, we can go over the implementation of the command as part of the package's Github repository[409].

Overall, when executing "su [USERNAME]" we are prompted for the password of [USERNAME]. In case we don't provide any [USERNAME] by default we are prompted for the password of root[410] - as shown in the screenshot below. By the way, if we are running as root we can change to a different user without providing its password - as shown in the screenshot below.

Lastly, there is a difference between "su" to "su -". The first one retains the user's environment variables, working directory, current user's shell setting and the target user's PATH variable is not updated. The second option "su -" resets the environment variables, changes the working directory to the target home folder, resets all shell's settings and updates the PATH variable[411]. By the way, it is recommended to use "su --login" instead of "su -" as a shortcut to avoid side effects caused by mixing environments[412].



[408] https://linuxize.com/post/su-command-in-linux/
[409] https://github.com/util-linux/util-linux/blob/master/login-utils/su-common.c
[410] https://www.tecmint.com/difference-between-su-and-su-commands-in-linux/
[411] https://www.geeksforgeeks.org/difference-between-su-and-su-command-in-linux/
[412] https://man7.org/linux/man-pages/man1/su.1.html

ps (Process Status)

“ps” is an ELF binary located at “/usr/bin/ps” (or “/bin/ps”) it is used for retrieving a snapshot of the current processes\threads\tasks[413]. By the way, the command is relevant for various operating systems (not only Linux) such as: “Unix”, “Plan 9”, “IBM i”, “KolibriOS” and more[414].

Overall, “ps” is based on reading the information it reports from “/proc”. Also, in case we want to override the default output format of the command by leveraging the “PS_FORMAT” environment variable, which is not the only one that affects “ps”[415]. “ps” on Linux supports both a standard syntax and a BSD syntax. For example “ps -e”\”ps -ef”\”ps -ely” shows all processes on the system in standard syntax while “ps ax”\”ps aux” does the same using the BSD syntax[416] - as shown in the syntax below (taken using https://copy.sh/v86/?profile=archlinux).

Lastly, among the information that “ps” can displayed we can find the following: CPU usage, memory usage, nice value, pid, ppid, priority, status code, time when the process started, associated terminal, uid and more[417]. “ps” is also one of the utilities which is included as part of “busybox”[418]. For a reference implementation we can check out the “procps” repository in GitLab[419].

```
root@localhost:~# ps aux | wc -l
58
root@localhost:~# ps -ef | wc -l
58
root@localhost:~# ps aux | head -5
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.3  0.3   2596  1580 ?        Ss   12:19   0:01 init [3]
root         2  0.0  0.0      0     0 ?        S    12:19   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        I<   12:19   0:00 [rcu_gp]
root         4  0.0  0.0      0     0 ?        I<   12:19   0:00 [rcu_par_gp]
root@localhost:~# ps -ef | head -5
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 12:19 ?        00:00:01 init [3]
root         2     0  0 12:19 ?        00:00:00 [kthreadd]
root         3     2  0 12:19 ?        00:00:00 [rcu_gp]
root         4     2  0 12:19 ?        00:00:00 [rcu_par_gp]
root@localhost:~# _
```

[413] https://man7.org/linux/man-pages/man1/ps.1.html
[414] https://en.wikipedia.org/wiki/Ps_(Unix)
[415] https://wiki.gentoo.org/wiki/Ps
[416] https://man.archlinux.org/man/ps.1.en
[417] https://www.wikiwand.com/en/articles/Ps_(Unix)
[418] https://medium.com/@boutnaru/the-linux-concept-journey-busybox-234cfdc8808a
[419] https://gitlab.com/procps-ng/procps

kacpid (Kernel Advanced Configuration and Power Interface Daemon)

“kacpid” is a Linux kernel thread which is based on an workqueue[420]. It is defined as part of “drivers/acpi/osl.c” in the “acpi_os_initialize1()” function, as part of the ACPI support of Linux[421].

Overall, ACPI (Advanced Configuration and Power Interface) is a standard developed by Toshiba, Microsoft and Intel. It is used for power and system management. Think about controlling the amount of power each device is given (by putting a device on standby/powering it off) - we can see the ACPI states in the table below[422]. Also, it can be used for checking battery levels, PCI IRQ routing, CPUs, NUMA domains, fan speeds, temperature sensors and more[423].

Lastly, work is queued for the “kacpid” kernel as part of the “acpi_os_execute()” function[424]. This kernel thread is used in case of a GPE (General Purpose Event) handler is needed[425]. We can think about a GPE as an interrupt. In which the hardware is informing the OS (using ACPI) that “something” has happened. Examples are plugging/unplugging your AC adapter, closing/opening the lid of your laptop[426].

Table 4.3 ACPI operating states

Global System State	Sleep State	Description
G0 Working	(S0)	The computer is fully functional. Software, such as the AutoSave function used with Microsoft products, can be optimized for performance or lower battery usage.
G1 Sleeping		Requires less power than the G0 state and has multiple sleeping states.
	(S1)	CPU is still powered, and unused devices are powered down. RAM is still being refreshed. Hard disks are not running.
	(S2)	CPU is not powered. RAM is still being refreshed. System is restored instantly upon user intervention.
	(S3)	Power supply output is reduced. RAM is still being refreshed. Some info in RAM is restored to CPU and cache.
	(S4)	Lowest-power sleep mode and takes the longest to come up. Info in RAM is saved to hard disk. Some manufacturers call this the hibernate state.
G2	(S5)	Also called soft off. Power consumption is almost zero. Requires the operating system to reboot. No information is saved anywhere.
G3		Also called off, or mechanical off. This is the only state where the computer can be disassembled. You must power on the computer to use it again.

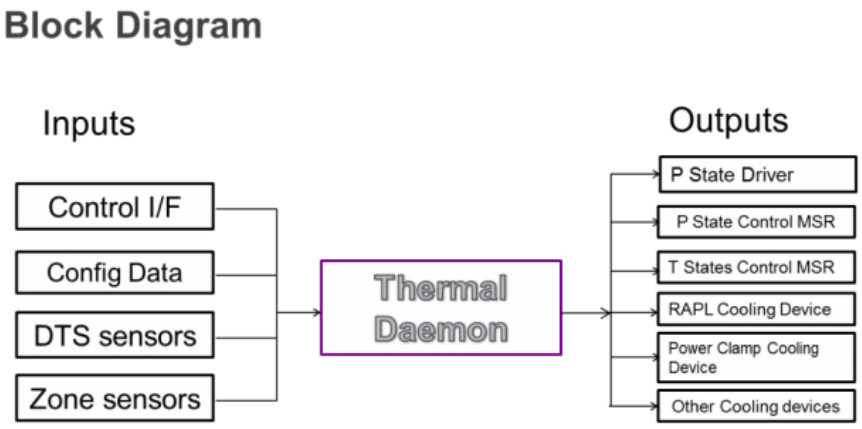
[420] https://elixir.bootlin.com/linux/v6.9.5/source/drivers/acpi/osl.c#L1665
[421] https://elixir.bootlin.com/linux/v6.9.5/source/drivers/acpi
[422] https://www.slideserve.com/reece/chapter-4-disassembly-and-power
[423] https://wiki.osdev.org/ACPI
[424] https://elixir.bootlin.com/linux/v6.9.5/source/drivers/acpi/osl.c#L1113
[425] https://elixir.bootlin.com/linux/v6.9.5/source/drivers/acpi/osl.c#L1105
[426] https://askubuntu.com/questions/148726/what-is-an-acpi-gpe-storm

thermald (Thermal Daemon)

“thermald” is a Linux daemon which is responsible for controlling/monitoring the temperature of laptops/tablets/PCs containing the latest Intel CPU (at least Sandy Bridge). It is used for activating cooling methods when the system temperature reaches a specific temperature value[427]. We can go over the source code of the thermal daemon in GitHub[428].

Overall, the goal is to prevent the BIOS from thermal throttling (by using proactive control) and to avoid the need of relying on the ACPI configuration. Thus, the “thermal daemon” provides a proactively user-mode controlled temperature solution that leverages existing kernel infrastructure and can be easily enhanced if needed. This is done by using sensors to calculate set a points and based on predefined configuration use the best cooling method[429] - the general architecture is shown in the diagram below.

Lastly, “thermald” has two modes of operations: “Zero Mode Configuration” and “User Defined Configuration Mode”. By using the latest kernel drivers/modules like DTS temperature sensor (the output can be read at “/sys/devices/platform/coretemp.x interface) , Intel’s P state driver (performance state), RAPL (running average power limit) and the power clamp driver the “thermal daemon” allows developers to more precisely control the temperature set point for a given use case. It is important to know that not all are accessible to all hardware configurations. For example the “Intel’s P state” drivers are specific to “Sandy Bridge” and “Ivy Bridge” CPUs[430].



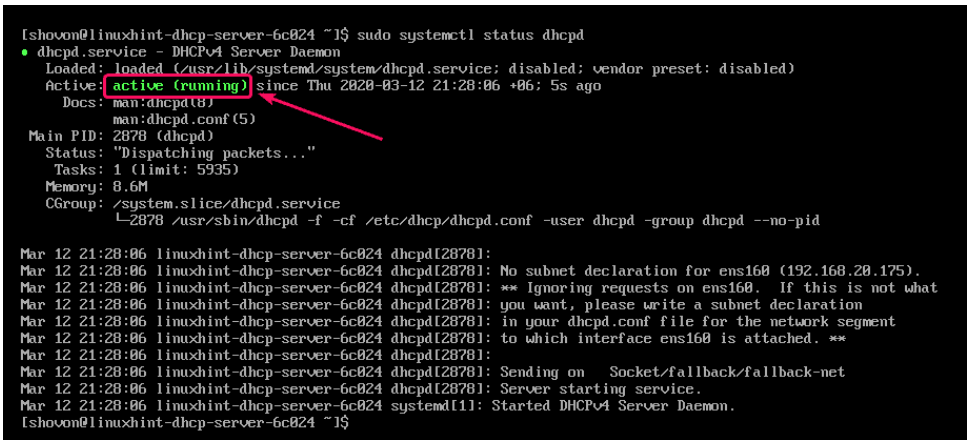
[427] https://wiki.debian.org/thermald
[428] https://github.com/intel/thermal_daemon
[429] https://web.archive.org/web/20211123224127/https://01.org/linux-thermal-daemon/documentation/introduction-thermal-daemon
[430] https://www.linux.com/news/linux-thermal-daemon-monitors-and-controls-temperature-tablets-laptops/

dhcpd (Dynamic Host Configuration Protocol Daemon)

“dhcpd” is an ELF binary located by default at “/usr/bin/dhcpd” (or “/bin/dhcpd”). It is an implementation of a DHCP (Dynamic Host Configuration Protocol) server for Linux. DHCP allows automatic configuration of network elements. Among that configuration can be included attributes such as: IP address, subnet mask and dns servers. The protocol operates on top UDP port 67 (server)/68 (client)[431].

Overall, we can find the “dhcpd” configuration in the “dhcpd.conf” file located at “/etc”. Among the different configurations we can include: DHCP scope, subnets (including net masks) and subnet specific parameters like default gateway, lease timeout and more[432]. “dhcpd” is the older ISC (Internet Systems Consortium) implementation of a DHCP server, which is not maintained since January 2023 and thus marked as “end of life”[433].By the way, the ISC replacement is called “Kea” (more on that in a future writeup).

Lastly, it is important to understand that “dhcpd” and “dhcpcd” are not the same. The first is a server implementation while the second is a client implementation of DHCP. Also, “dhcpd” includes both an implementation for DHCPv4 and DHCPv6[434]. More information about “dhcpd” can be found as part of the ISC documentation[435]. On some Linux distributions we can use “systemctl”[436] for performing different operations of the “dhcp” daemon like checking its status - as shown in the screenshot below[437].



[431] https://www.spiceworks.com/tech/networking/articles/what-is-dhcp/
[432] https://linux.die.net/man/5/dhcpd.conf
[433] https://www.isc.org/blogs/isc-dhcp-eol/
[434] https://wiki.archlinux.org/title/dhcpd
[435] https://kb.isc.org/docs/aa-00333
[436] https://www.man7.org/linux/man-pages/man1/systemctl.1.html
[437] https://linuxhint.com/dhcp_server_centos8/

ata_sff (Advanced Technology Attachment Small Form Factor)

“ata_sff” is a Linux kernel thread which is based on a workqueue[438] - as shown in the screenshot below (taken from https://copy.sh/v86/?profile=archlinux). The source code file continuing that (libata-sff.c) states it is a used as a helper library for PCI IDE BMDMA[439]. By the way, BMDMA stands for “Bus-Master Direct Memory Access”[440].

Overall, to enable “ata_sff” support (for legacy IDE and PATA) we need to enable “CONFIG_ATA_SFF” while compiling the kernel[441]. Also, ATA originally was designed to work with hard disks (and devices that could emulate them). However a committee called SFF (Small Form Factor) (SFF) introduced the ATAPI which can be used for a variety of other devices, which includes functions beyond those necessary for hard disks[442].

Moreover, libATA is a library used inside the Linux kernel to support ATA (Advanced Technology Attachment) host controllers/devices. It is used for providing: class transport for ATA/ATAPI devices, SCSI<->ATA translation based on T10 SAT specification for ATA devices and ATA driver API[443]. Among the features that are supported are: power management, analysis and reporting, hot swapping, NCQ (native command queueing) and port multiplying[444]. I suggest going over the libATA’s developer guide for more information about the API[445].

Lastly, work is queued to “ata_sff” using the “ata_sff_queue_work” function[446]. An example of using it can be found as of the Arasan Compact Flash host controller source[447]. By the way, there is also a possibility to queue the work after a specific delay using the "ata_sff_queue_delayed_work" function[448].

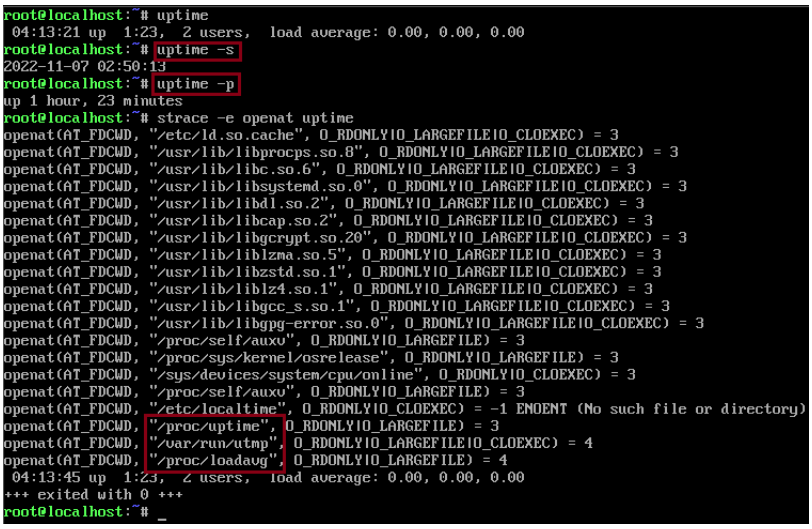
[438] https://elixir.bootlin.com/linux/v6.7.4/source/drivers/ata/libata-sff.c#L3197
[439] https://elixir.bootlin.com/linux/v6.7.4/source/drivers/ata/libata-sff.c#L3
[440] https://www.linux.org/threads/the-linux-kernel-configuring-the-kernel-part-13.9143/
[441] https://cateee.net/lkddb/web-lkddb/ATA_SFF.html
[442] https://en.wikipedia.org/wiki/ATAPI
[443] https://www.kernel.org/doc/Documentation/driver-api/libata.rst
[444] https://www.linux-ata.org/features.html
[445] https://www.kernel.org/doc/html/v4.14/driver-api/libata.html
[446] https://elixir.bootlin.com/linux/v6.7.4/source/drivers/ata/libata-sff.c#L1165
[447] https://elixir.bootlin.com/linux/v6.7.4/source/drivers/ata/pata_arasan_cf.c#L686
[448] https://elixir.bootlin.com/linux/v6.7.4/source/drivers/ata/libata-sff.c#L1171

uptime (System's Uptime)

"uptime" is a Linux ELF binary located "/bin/uptime" and/or "/usr/bin/uptime". It is used for telling how long the specific Linux system has been running. This means the amount of time a system remains operational and accessible without interruptions[449]. Among the information which is displayed by uptime we can find the following: the current time, how much the system is running, how many users are logged on and the system load averages for the past 5/10/15 minutes[450] - as shown in the screenshot below (taken from copy.sh).

Overall, for collecting information about the logged on user "uptime" accesses "/var/run/utmp". In case of the system load averages "/proc/loadavg"[451] file is used for getting the information. To get the total time the system is running the "/proc/uptime"[452] file is accessed - as shown in the output of the system call tracing (using strace) below.

Lastly, "uptime" also has command line switches that can be used. Examples are: "-p" which prints the output of the uptime in human readable format and "-s" which prints the full timestamp from which the system is alive[453] - as shown in the screenshot below. "uptime" is part of the "coreutils" package, we can also check the source code for more information[454].



[449] https://phoenixnap.com/glossary/what-is-uptime
[450] https://linux.die.net/man/1/uptime
[451] https://man7.org/linux/man-pages/man5/proc_loadavg.5.html
[452] https://man7.org/linux/man-pages/man5/proc_uptime.5.html
[453] https://phoenixnap.com/kb/linux-uptime
[454] https://github.com/coreutils/coreutils/blob/master/src/uptime.c

agetty (Alternative Linux getty)

“getty” is a Linux ELF binary located at “/bin/agetty” and/or “/usr/bin/agetty”. The binary is used for opening a new tty port and prompting for a login by leveraging “/bin/login”[455]. Also, “agetty” is normally launched by PID 0 (init/systemd). Moreover, “agetty” has extra features which can be usual for dial up\hardwired connections such as: deducing baud rate, not displaying the content of /etc/issue, invoking a different login program (not /bin/login), forcing the line to be local with no need for carrier detect and more[456].

Overall, “agetty” is the Linux version of “getty” (which is short for “get TTY”). In the realm of Unix “getty” is programming running on a host computer which is used for managing physical/virtual terminals for providing multi-user access[457]. We can pass different arguments to “agetty” to configure its operation such as “-8” which is assume 8-bit tty mode - as shown in the screenshot below[458]

Lastly, “agetty” is part of the “util-linux” set of utilities. We can think about it as a replacement for getty on SunOS 4.1.x or the SAC ttymon/ttyadm/sacadm/pmadm suite on Solaris and other SVR4 systems. We can find the code of “agetty” as part of the “util-linux” GitHub repository[459].

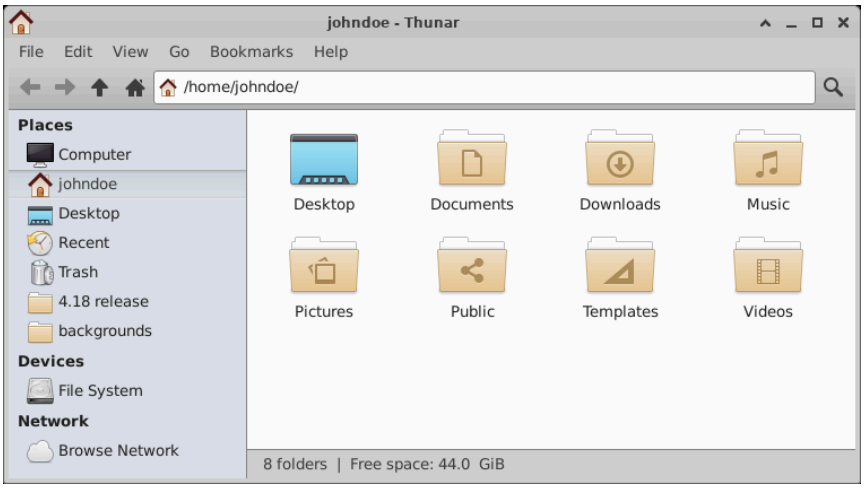
[455] https://medium.com/@boutnaru/the-linux-process-journey-login-02b6d83ab6c5
[456] https://man7.org/linux/man-pages/man8/agetty.8.html
[457] https://www.geeksforgeeks.org/agetty-command-in-linux-with-examples/
[458] https://linuxgenie.net/agetty-command-in-linux/
[459] https://github.com/util-linux/util-linux/blob/master/term-utils/agetty.c

Thunar (Xfce File Manager)

“Thunar” is a symbolic link located at “/bin/Thunar” and/or “/usr/bin/Thunar”. It points to “/bin/thunar” and “/usr/bin/thunar” respectively, which are Linux ELF binaries. Thunar provides basic file manipulation and handling like: renaming, deleting, watching attributes, searching and more. Also, Thunar allows working with removal media and network storage such as NFS/SMB[460].

Overall, Thunar is a modern file manager as part of the Xfce desktop environment[461]. We can think about it as similar to Windows’ “File Explorer”[462] - as shown on the screenshot below[463]. Thunar also supports installing plugins which are basically additional packages that are used in order to extend functionality[464].

Lastly, it is highly recommended to go over the source code of thunar[465]. Thunar can be configured to run commands automatically when media are connected (If both gvfs and thunar-volman are installed). In case of mobile devices (which usually are based on MTP) , an additional gvfs-mtp package is required. For Thunar to support automatic mounting, we need to launch thunar in daemon mode. This is done by providing the “--daemon” argument to the thunar executable[466].



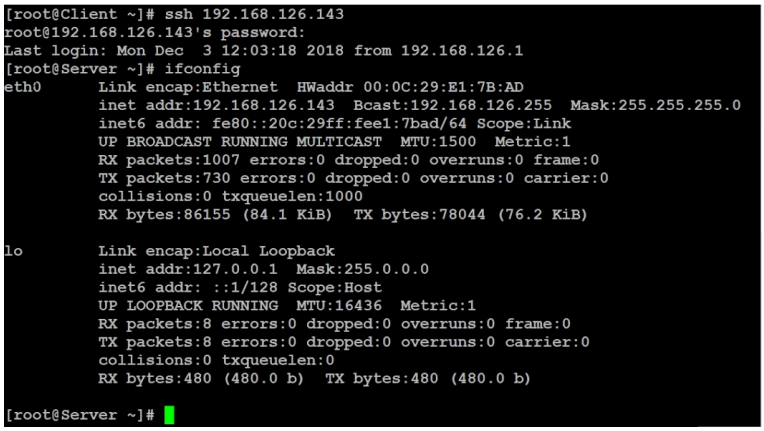
[460] https://docs.xfce.org/xfce/thunar/start
[461] https://medium.com/@boutnaru/the-linux-concept-journey-xfce-desktop-environment-f91b5272f5ed
[462] https://medium.com/@boutnaru/the-windows-concept-journey-file-explorer-previously-windows-explorer-e48077b135a0
[463] http://docs.xfce.org/xfce/thunar/the-file-manager-window
[464] https://goodies.xfce.org/projects/thunar-plugins/start
[465] https://gitlab.xfce.org/xfce/thunar
[466] https://wiki.archlinux.org/title/Thunar

ssh (Secure Shell Client)

“ssh” (Secure Shell Client) is a Linux ELF binary located at “/bin/ssh” and/or “/usr/bin/ssh” (could be also “/bin/ssh” or “/usr/sbin/ssh”). It provides the ability to remote login to a Linux system. After logging in we can execute commands on the remote machine - as shown in the screenshot below[467]. Also, it is used as a replacement for rlogin/rsh. SSH creates a secure encrypted channel between the client and server. By the way, TCP communications (such as X11) can be forwarded over the SSH tunnel[468].

Overall, the SSH client can authenticate using passwords (less secure) or SSH keys (which is considered to be more secure). When authenticating with SSH keys we need to have the key pair on the local machine. Also, the remote system needs the public key to be stored in “~/.ssh/authorized_keys” (where “~” stands for the home folder of the connecting user). The “authorized_keys” file contains a list of public keys (one public key per line) which are authorized to log into the specific user account[469].

Lastly, “ssh” also has its own configuration which is stored in “~.ssh/config”[470]. Among the configuration options we can find: enabling X11 forwarding and agent forwarding, port forwarding, configuring public key authentication and more[471]. For more information we can check out the OpenSSH GitHub repository[472].



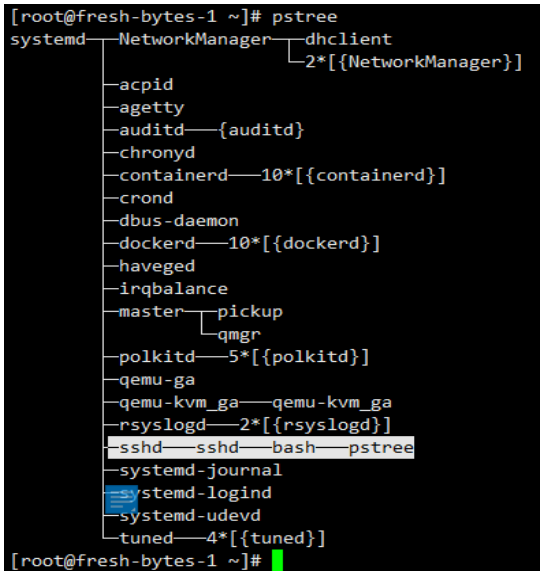
[467] https://www.youtube.com/watch?v=0PRd4aCyrkc
[468] https://linux.die.net/man/1/ssh
[469] https://www.digitalocean.com/community/tutorials/ssh-essentials-working-with-ssh-servers-clients-and-keys
[470] https://www.geeksforgeeks.org/how-to-configure-ssh-client-in-linux/
[471] https://www.ssh.com/academy/ssh/config
[472] https://github.com/openssh/openssh-portable/blob/master/ssh.c

sshd (OpenSSH SSH Daemon)

“sshd” (OpenSSH SSH Daemon) is a Linux ELF binary located at “/sbin/sshd” and/or “/usr/sbin/sshd” (In rare cases “/bin/sshd” or “/usr/bin/sshd”). It is the demon part for the “ssh”[473] client. “sshd” is the server part which provides a secure encrypted channel between two endpoints over the network. “sshd” is part of the openssh-server Linux package[474].

Overall, “sshd” listens for incoming connections from clients. By default it listens on port 22 which can be modified by configuration or by using the “-p” argument. Also, it mostly is being run in the background as an operating system service. For every incoming connection the process is forked. The forked instance handles key exchange, encryption, authentication, command execution, and data exchange[475] - as shown in the screenshot below[476].

Lastly, the best practice for configuring the SSH server is by leveraging a configuration file which by default is “/etc/ssh/ssh_config”. We can also use a different configuration file by specifying the “-f” argument. Examples for configuration options are: chipers, chroot directory, allowed users, authorized key files and more[477]. For more information we can check out the OpenSSH GitHub repository[478].



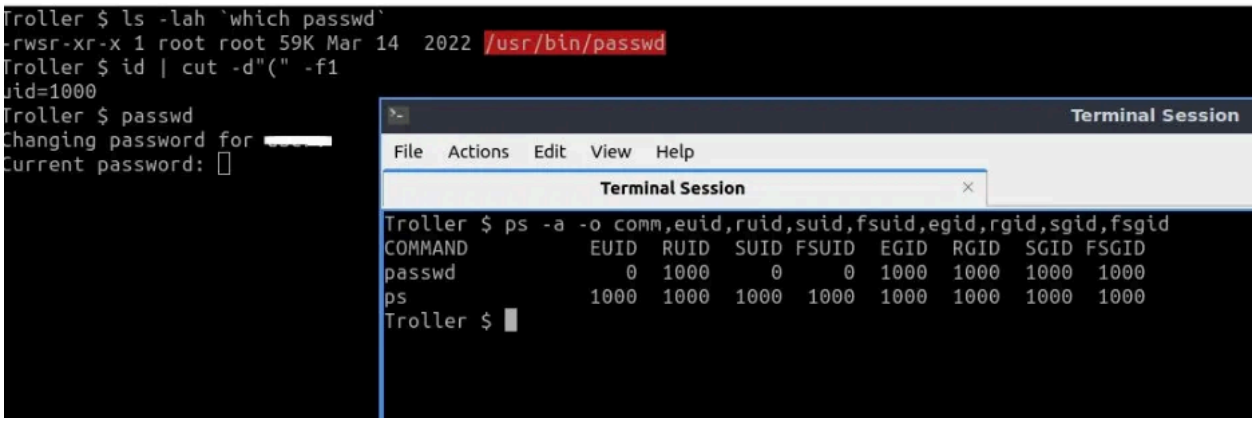
[473] https://medium.com/@boutnaru/the-linux-process-journey-ssh-secure-shell-client-01cc67897629
[474] https://launchpad.net/ubuntu/+source/openssh
[475] https://linux.die.net/man/8/sshd
[476] https://wumanho.cn/posts/cmd/
[477] https://linux.die.net/man/5/sshd_config
[478] https://github.com/openssh/openssh-portable/blob/master/sshd.c

passwd (Change User Password)

“passwd” is an ELF binary located at “/usr/bin/passwd” (or “/bin/passwd”). It is used for changing the password for a specific user account. Due to the fact it needs to modify root only writable content (“/etc/shadow” and/or “/etc/passwd”) the binary is owned by root and marked with “suid” bit[479].

Overall, every user can change only its own password thanks to the “real user ID” aka RUID[480] - as shown in the screenshot beow. Root\supersuer can of course alter the password of any user (given as an argument to the command). The user is first prompted for its old password which is verified, if the password is correct the user can modify the password. In case of a superuser\root the first phase is bypassed to allow resetting a password if the user has forgotten it. Also, the password is tested for its complexity for security reasons[481].

Lastly, when the user changes a password a prompt is presented twice. The second time is compared to the first time in order to ensure the two password input match (this is a safety mechanism). The password is checked against different requirements like: length, history checking, ensuring it is not part of a configured dictionary and more[482]. This can be done using the “/etc/default/passwd” configuration file and\or PAM[483].



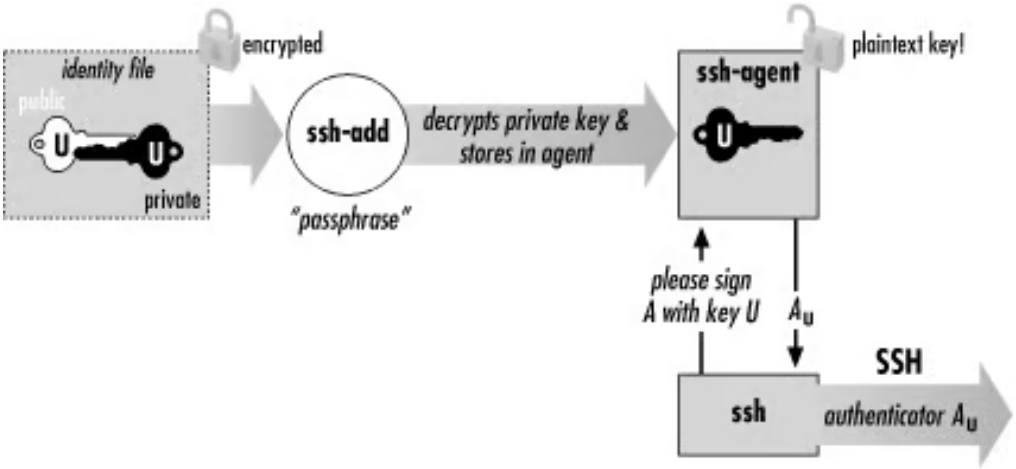
[479] https://medium.com/@boutnaru/linux-security-suid-bit-d4f553e7d99e
[480] https://medium.com/@boutnaru/the-linux-security-journey-ruid-real-user-id-b23abcbca9c6
[481] https://man7.org/linux/man-pages/man1/passwd.1.html
[482] https://docs.oracle.com/cd/E88353_01/html/E37839/passwd-1.html
[483] https://medium.com/@boutnaru/the-linux-security-journey-pam-pluggable-authentication-module-388496a8785c

ssh-agent (Secure Shell Authentication Agent)

“ssh-agent” (Secure Shell Authentication Agent) is a Linux ELF binary located at “/bin/ssh-agent” and/or “/usr/bin/ssh-agent”. It is an integral part of the OpenSSH package. “ssh-agent” is the key manager for SSH. This means it holds keys/certificates in memory in an unencrypted manner ready for ssh to use (saves us retyping passphrases every time we connect to a server). It runs separately from the ssh binary in the background after running ssh for the first time[484] - as shown in the diagram below[485].

Overall, the reason “ssh-agent” keeps crypto materials (RSA/DSA/etc) safe is due to the fact they are not exported nor written to disk. “ssh-agent” is started at the beginning of an X-session or a login session. Thus, all other windows or programs are started as clients of it. By using environment variables “ssh-agent” can be located and used for authentication when logging using SSH[486].

Lastly, an ssh client and the “ssh-agent” are “speaking” to each other using the “SSH Agent Protocol”[487] - more on that in future writeups. The basic operations that “ssh-agent” is providing are: adding a key pair, removing a key/key pair, sign messages using the stored keys, listing keys stored and more. We can check out OpenSSH’s GitHub repository for more information[488].



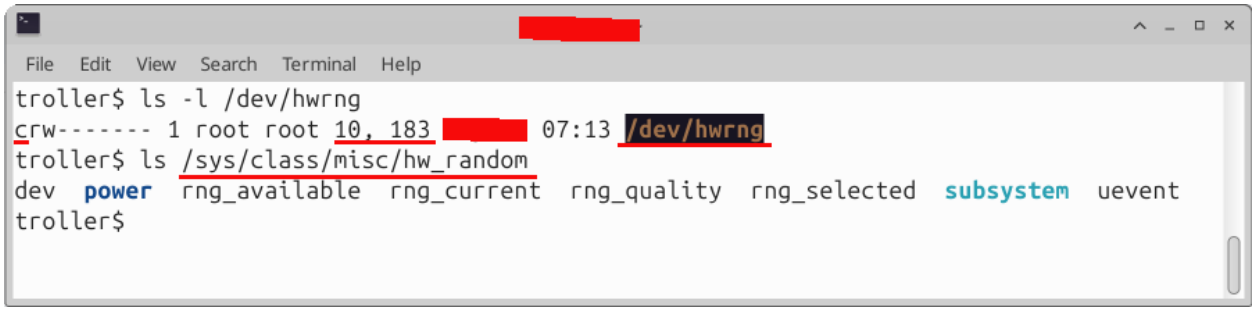
[484] https://smallstep.com/blog/ssh-agent-explained/
[485] https://dev.to/samuyi/ssh-agents-in-depth-4116
[486] https://linux.die.net/man/1/ssh-agent
[487] https://datatracker.ietf.org/doc/html/draft-miller-ssh-agent-04
[488] https://github.com/openssh/openssh-portable/blob/master/ssh-agent.c

hwrng (Hardware Number Random Generator)

“hwrng” is a is a Linux kernel thread which is executed using the “kthread_run” (a wrapper for ktrehad_create) function[489]. The specific function which is used for that is “hwrng_fillfn”[490]. It is provided by the “hw_random” framework, which is a software component that makes use of a hardware based RNG (Random Number Generator) on the CPU\motherboard. It has two major parts: a character device[491] “/dev/hwrng” including a sysfs support and a hardware-specific driver[492].

Overall, “/dev/hwrng” has the following <major,minor>[493] number pair: <10, 183> and its relevant sysfs attributes are located at “/sys/class/misc/hw_random” - as shown in the screenshot below. We can also see the specific minor number is the source code of the kernel[494]. By the way, the creation of the character device is done as part of the initialization of the module[495], it's the same “C file” which contains the creation of the “hwrng” kernel thread.

Lastly, to best leverage that we should use “rng-tools”. They use the “/dev/hwrng” to fill the kernel entropy pool,which is used internally and exported by the /dev/urandom and/dev/random special files[496]. By using them we monitor the entropy because just opening and reading from “/dev/hwrng” does not perform any fitness check for the randomness quality[497].



[489] https://elixir.bootlin.com/linux/v6.14.5/source/drivers/char/hw_random/core.c#L92
[490] https://elixir.bootlin.com/linux/v6.14.5/source/drivers/char/hw_random/core.c#L479
[491] https://medium.com/@boutnaru/the-linux-concept-journey-character-devices-0c75aa70ceb2
[492] https://www.kernel.org/doc/Documentation/admin-guide/hw_random.rst
[493] https://medium.com/@boutnaru/the-linux-concept-journey-major-minor-numbers-56abe372482e
[494] https://elixir.bootlin.com/linux/v6.14.5/source/include/linux/miscdevice.h#L46
[495] https://elixir.bootlin.com/linux/v6.14.5/source/drivers/char/hw_random/core.c#L670
[496] https://www.kernel.org/doc/html/next/admin-guide/hw_random.html
[497] https://github.com/nhorman/rng-tools

psimon (Pressure Stall Information Monitoring)

“psimon” is a kernel thread which is created using the “kthread_create” function[498]. PSI stands for “Pressure Stall Information”, it is used for showing how resource pressure increases as they develop. This is done by leveraging new pressure metrics targeting memory, CPU and IO. Together with other components (cgroup2) we can identify resource shortage while they are developing and react with operations like: pausing\stopping non-essential processes, reallocating memory, load shedding and more[499].

Overall, PSI is only relevant since kernel version 4.20[500]. The pressure information for each resource (CPU/IO/Memory) is exported to user-mode using profcs: “/proc/pressure/” - as shown in the screenshot below. All of those are created by the “psi_proc_init”[501] function. By the way, the some at the beginning of each line means (in regards to a specific resource) at least one task is delayed on a resource. This affects the workload's ability to perform work, however the CPU may still be executing other tasks[502]. The full keyword means all non-idle tasks are delayed on that resource such that nobody is advancing and the CPU goes idle. Based on the definition the full state does not exist for the CPU[503].

Lastly, for registering a trigger with PSI we just need to open the relevant file related to the resource we want to monitor and write the desired threshold and time window. Then we can leverage select()/poll()/epoll() on the returned file descriptor. An example is “some 170000 2000000 > /proc/pressure/memory” would add 170 ms threshold for partial memory stall measured within 2 sec time window[504].

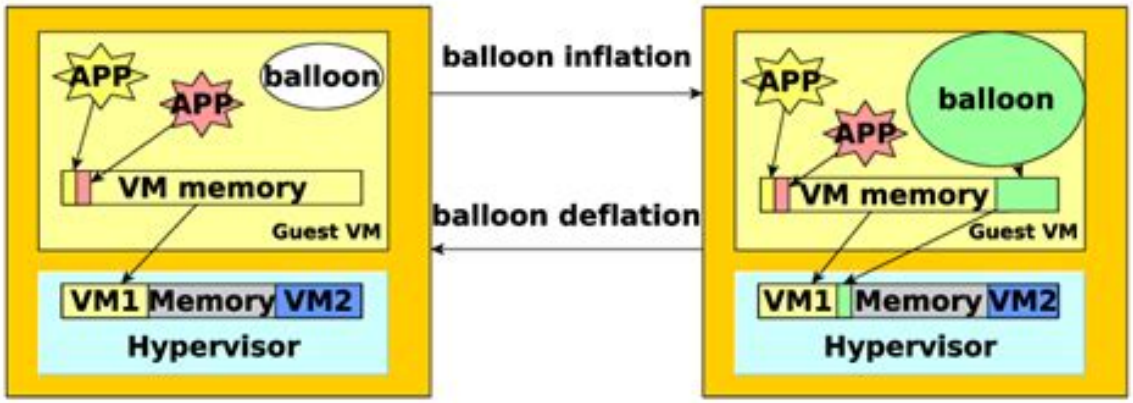
[498] https://elixir.bootlin.com/linux/v6.14.5/source/kernel/sched/psi.c#L1357
[499] https://facebookmicrosites.github.io/psi/docs/overview
[500] https://elixir.bootlin.com/linux/v4.20/source/kernel/sched/psi.c
[501] https://elixir.bootlin.com/linux/v4.20/source/kernel/sched/psi.c#L764
[502] https://elixir.bootlin.com/linux/v4.20/source/kernel/sched/psi.c#L26
[503] https://elixir.bootlin.com/linux/v4.20/source/kernel/sched/psi.c#L30
[504] https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/Documentation/accounting/psi.rst

hv_balloon (Microsoft Hyper-V Memory Ballooning)

"hv_balloon" is a is a Linux kernel thread which is created using the "kthread_run" macro[505], which wraps the "kthread_create" function[506]. The function to run within the thread is "dm_thread_func"[507] among its operations it posts information regarding the memory pressure to the host every second[508]. "hv_balloon" is part of the Microsoft Hyper-V guest support for Linux[509], that provides memory ballooning.

Overall, memory ballooning provides the ability of avoiding the need for overcommitting host memory for guest virtual machines (VMs). This is done by letting each VM "give back" unused pages of virtual memory to the host. The balloon driver (inside the guest like "hv_balloon") allocates unused memory into a pool of memory aka "balloon inflation" (this makes the memory unavailable for applications\processes on the guest). The host (hypervisor\operating system) maps those pages and thus reclaim them aka "balloon deflation"[510] - as shown in the diagram below[511].

Lastly, the module name of "Microsoft Hyper-V Balloon driver" is "hv_balloon.ko". It is part of the Microsoft Hyper-V guest support which is part of the Linux kernel since version 3.1 (release date 2011-10-24). In order to include the support as part of the Linux system we should enable "CONFIG_HYPERV_BALLOON"[512]. This support by the way is based on the Hyper-V's client drivers for Linux[513].



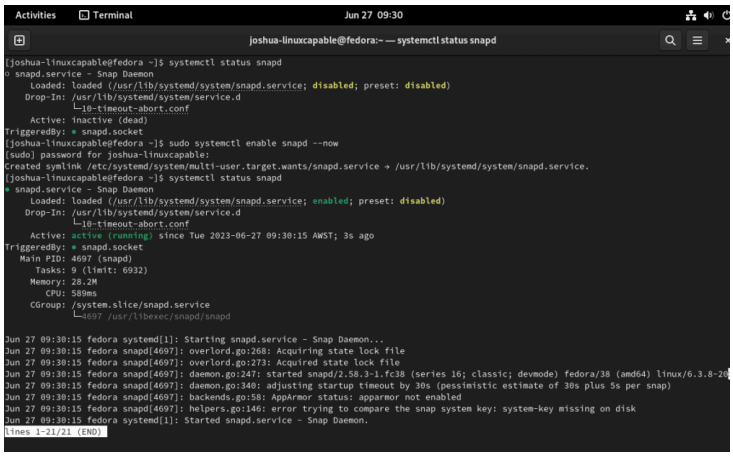
[505] https://elixir.bootlin.com/linux/v6.14.6/source/drivers/hv/hv_balloon.c#L1992
[506] https://elixir.bootlin.com/linux/v6.14.6/source/include/linux/kthread.h#L42
[507] https://elixir.bootlin.com/linux/v6.14.6/source/drivers/hv/hv_balloon.c#L1373
[508] https://elixir.bootlin.com/linux/v6.14.6/source/drivers/hv/hv_balloon.c#L1381
[509] https://elixir.bootlin.com/linux/v6.14.6/source/drivers/hv/Kconfig#L3
[510] https://en.wikipedia.org/wiki/Memory_ballooning
[511] https://www.researchgate.net/figure/Memory-ballooning-principles_fig2_330462327
[512] https://www.kernelconfig.io/config_hyperv_balloon
[513] https://elixir.bootlin.com/linux/v6.14.6/source/drivers/hv/Kconfig#L51

snapd (Snap Daemon)

“snapd” is an ELF binary that might be located at “/usr/lib/snapd/snapd”. It is a background service that is used for managing and maintaining snaps[514]. Although “snapd” is pre-installed with Ubuntu[515] it can be installed on other Linux distributions[516] such as (but not limited to) Fedora[517] - as shown in the screenshot below.

Overall, “snapd” exposes a REST (Representational State Transfer) API for managing snap[518] packages. We can access those APIs by leveraging the “snap” client which is included in the same package as “snapd”. The snap client can talk with “snapd” by using a Unix domain socket[519] called “snapd.socket”. The socket is commonly located at “/run/snapd.socket”[520].

Lastly, “snapd” is an open-source project created by Canonical. It implements confinement policies that isolate snaps from the base system and from each other. Also, it governs the interfaces that allow snaps to access specific system resources outside of their confinement. We can check-out its source code as part of the “snapd” GitHub repository[521].



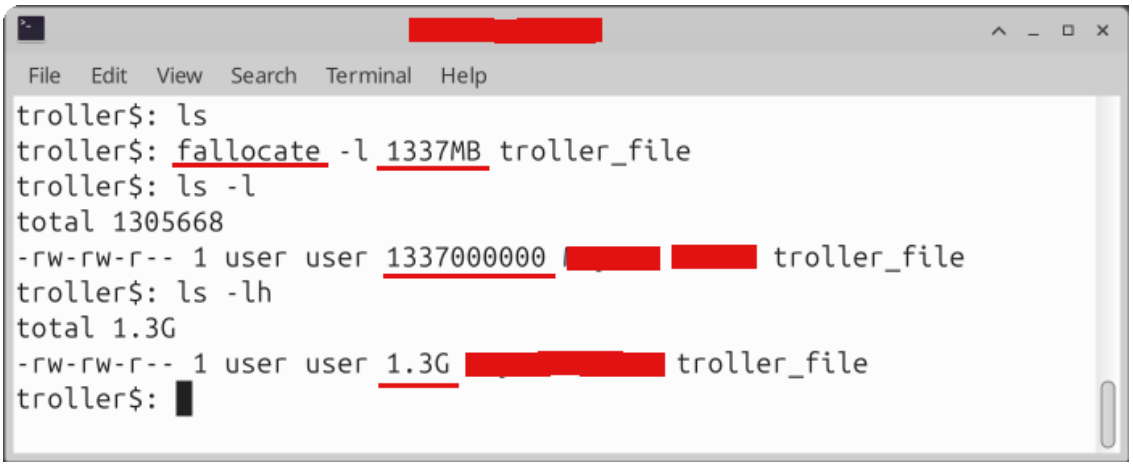
514 https://snapcraft.io/docs/installing-snapd
515 https://en.wikipedia.org/wiki/Snap_(software)
516 https://medium.com/@boutnaru/the-linux-concept-journey-linux-distribution-2dfc68aaa4f4
517 https://linuxcapable.com/how-to-install-snap-on-fedora-linux/
518 https://medium.com/@boutnaru/the-linux-concept-journey-snap-f39201df078d
519 https://medium.com/@boutnaru/the-linux-concept-journey-socket-file-0ab8245290d6
520 https://wiki.archlinux.org/title/Snap
521 https://github.com/canonical/snapd

fallocate (Preallocate\Deallocate Space for a File)

“fallocate” is an ELF binary located at “/bin/fallocate” and/or “/usr/bin/allocate”. It is used for preallocating and/or deallocating disk space for a file - as shown in the screenshot below. Thus, we can use this command line utility as an alternative for creating a file and filling it with zeros. Due to that users can quickly allocate space without writing any data. It is a great way for creating files with a specific size and/or preventing the case of running out of disk space while writing a file[522].

Overall, in case we want to preallocate space on a file-system which natively supports the “fallocate” system call we don’t need to create a file and fill it up with zeros. The preallocation can be done just by allocating blocks and marking them as uninitialized (which might be performed without any IO operations in regards to the data blocks). Thus, it is much more quick than the first option[523].

Lastly, the implementation of the “fallocate” system call as part of libc (using a wrapper) is Linux specific and non-portable. For a POSIX portable use-case we should leverage the “posix_fallocate” library call[524]. When calling the “fallocate” system call we get into the “vfs_fallocate” that calls the specific file-system code[525]. We can check the “ntfs_fallocate” function as part of the NTFS code for an implementation example[526].



[522] https://phoenixnap.com/kb/fallocate
[523] https://www.man7.org/linux/man-pages/man1/fallocate.1.html
[524] https://www.man7.org/linux/man-pages/man2/fallocate.2.html
[525] https://elixir.bootlin.com/linux/v6.14.6/source/fs/open.c#L338
[526] https://elixir.bootlin.com/linux/v6.14.6/source/fs/ntfs3/file.c#L561

ufw (Uncomplicated Firewall)

“ufw” (Uncomplicated Firewall) is a Python script located at “/usr/sbin/ufw” (or “/sbin/ufw”). It is a firewall configuration tool for easing netfilter[527] based firewall configuration. Thus, it provides a user-friendly way for creating an IPv4\IPv6 host-based firewall. UFW is the default tool for the job in case of Ubuntu, which is by default disabled. In case we enable ufw (“sudo ufw enable”) it uses a default set of rules (profile) that should be fine for the average home user (or so the developers believe). All 'incoming' is denied, with some exceptions to make things easier for home users[528].

Overall, UFW is available by default as part of Ubuntu since version 8.04 TLS. Also, it is included as part of Debian since version 10. For easier usage there is also “Gufw” (GUI for Uncomplicated Firewall) which is built using Python, GTK and of course ufw[529] - as shown in the screenshot below.

Lastly, “ufw” provides different capabilities such as (but not limited to): default incoming policy (allow/deny), IPv6 support, logging, per rule logging, rsyslog support, rate limiting and filtering by interface. We can checkout the configuration files located at “/etc/ufw/*”[530].

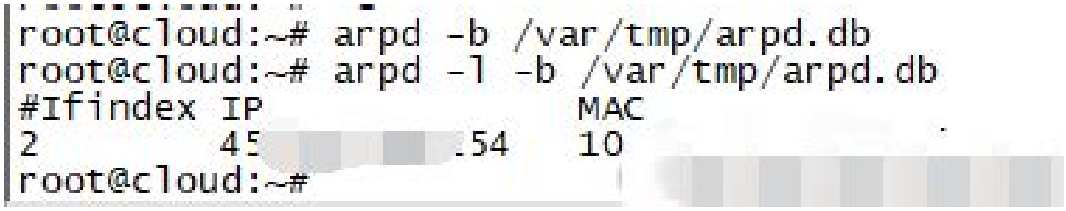
[527] https://medium.com/@boutnaru/the-linux-security-journey-netfilter-90c6cf12ca40
[528] https://help.ubuntu.com/community/UFW
[529] https://en.wikipedia.org/wiki/Uncomplicated_Firewall
[530] https://wiki.ubuntu.com/UncomplicatedFirewall

arpd (Address Resolution Protocol Daemon)

“arpd” (Address Resolution Protocol Daemon) is an ELF binary located at “/usr/sbin/arpd” or “/sbin/arpd”. It is a user-space arp daemon used for collecting gratuitous ARP messages, which is an ARP response that had been received without sending a request. Gratuitous ARP is a broadcast message announcing\updating the mapping (IP->MAC) of a specific node[531].

Overall, “arpd” saves the data to disk (by default “/var/lib/arpd/arpd.db ”) and feeds it to the kernel on demand in order to avoid redundant broadcasting - as shown in the screenshot below[532]. By the way, enabling “CONFIG_ARPD” adds support to have a user space daemon to perform ARP resolution, which is defined in the source code of the Linux kernel up to and including version 3.11.10[533]. This configuration basically sets “arpd” as an ARP resolver[534].

Lastly, “arpd” supports different switches: “-l” for dumping the database to stdout and “-f” to load\read the database from a file. Also, “-a” sets the number of ARP broadcast queries to send before considering the destination is data; it is often used with “-k” to suppress kernel broadcasts. The “-n” flag configures a timeout for negative cache entries, while “-R” and “-B” control the rate and burst of broadcast packets. We can configure “arpd” to monitor a specific network interface or all of them if none is specified[535]. For more information we can go over the source code of “arpd” as part of the “iproute2” repository[536].



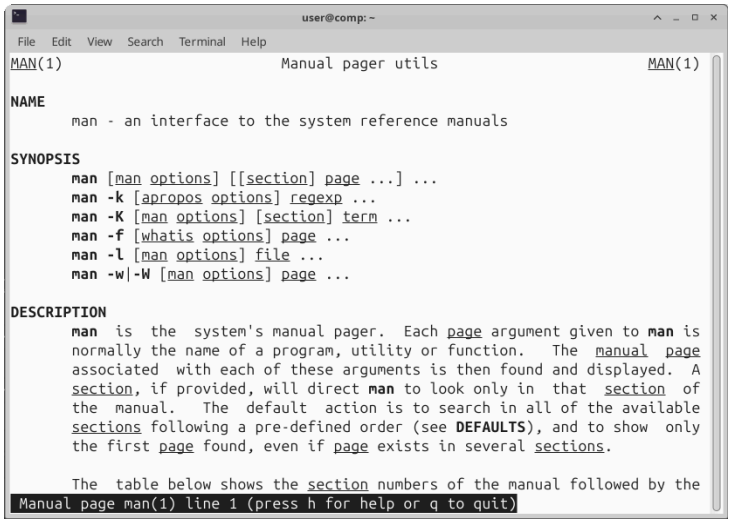
[531] https://www.practicalnetworking.net/series/arp/gratuitous-arp/
[532] https://www.zonghengcloud.com/article/9155.html
[533] https://elixir.bootlin.com/linux/v3.11.10/A/ident/CONFIG_ARPD
[534] https://linux.die.net/man/8/arpd
[535] https://man7.org/linux/man-pages/man8/arpd.8.html
[536] https://github.com/iproute2/iproute2/blob/main/misc/arpd.c

man (Manual Pages)

man (Manual Pages) is a Linux ELF binary located at "/usr/bin/man" and/or "/bin/man". It is an interface to the system reference manuals - as shown in the screenshot below. Thus, man is the system's manual pager[537]. We can search the manual pages using one of the following equivalent commands: "man -k {expression}", "man --apropos {expression}" and "apropos {expression}". By the way, for searching keywords in a whole page text we should use the "-K" option[538].

Overall, the Linux man pages are clustered into different topics called "sections". Among the sections are: "section 1" user commands, "section 2" system calls, "section 3" library calls, "section 4" special files, "section 5" file formats, "section 6" games, "section 7" conventions and miscellany, "section 8" administration and privileged commands, "section L" math library functions and "section N" tcl functions[539].

Lastly, "man-db" is an implementation of the standard Unix documentation system, which is leveraged by the "man" utility. It is based on the Berkeley DB database and not the traditional flat-text whatis databases. "man-db" is used by several popular GNU/Linux distributions[540] such as (but not limited to): Ubuntu, Debian, Arch Linux, Gentoo and OpenSUSE[541]. For more information we can check the "man-db" GitLab's repository[542].



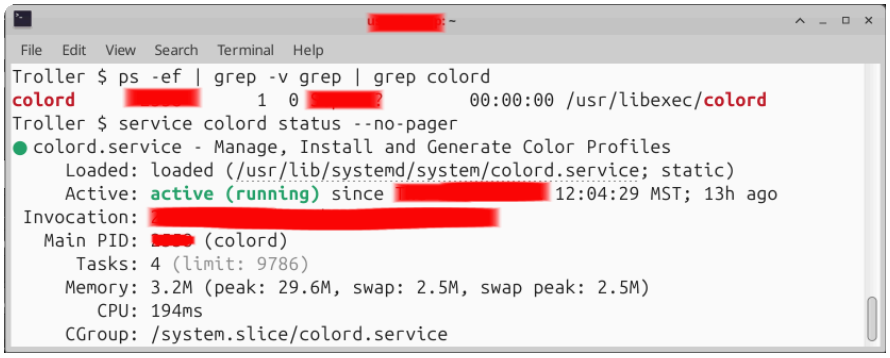
[537] https://man7.org/linux/man-pages/man1/man.1.html
[538] https://wiki.archlinux.org/title/Man_page
[539] https://linux.die.net/man/
[540] https://medium.com/@boutnaru/the-linux-concept-journey-linux-distribution-2dfc68aaa4f4
[541] https://man-db.nongnu.org/
[542] https://gitlab.com/man-db/man-db

colord (Color Daemon)

“colord” is an ELF binary that can be located at “/usr/libexec/colord”. It is a system service which provides the ability to install\generate color profiles (this is for color-managed input and output devices). It is used by “GNOME Color Manager” for system integration. It is also used when no user is logged in[543]. By default, the “colord” service is executed with the permissions of the “colord” user - as shown in the screenshot below.

Overall, “colord” provides different capabilities such as (but not limited to): persistent database backed store that is preserved across reboots, a D-Bus[544] API for system frameworks to query and provides the ability to set color based system settings. “colord” supports various subsystems like: XRandR (monitors), SANE (scanners), UDEV (cameras), CUPS (printers) and Virtaul (scanners/cameras/printers). Both “GNOME Color Manager” and “colord-kde” (used with KDE desktop) act as clients of “colord”[545].

Lastly, colord leverages PolKit[546] for user authentication. Thus, we can configure what users can do and what not[547]. The configuration of devices to profiles is stored in the colord’s mapping database (/var/lib/colord/mapping.db). Persistent devices\profiles can also be saved to “/var/lib/colord/storage.db” aka the storage database[548]. For more information I suggest going over colord’s GitHub repository[549].



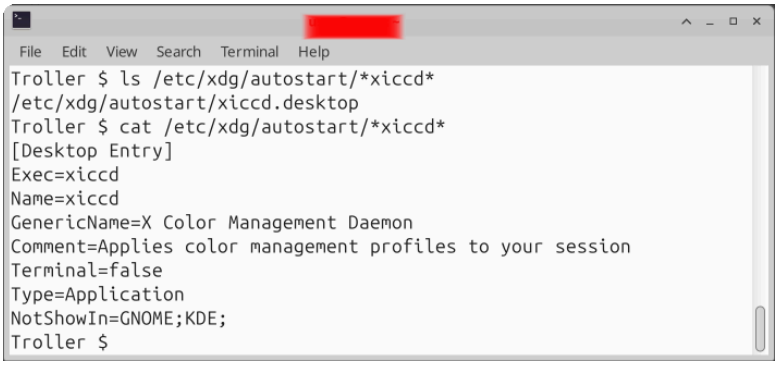
[543] https://www.linuxfromscratch.org/blfs/view/svn/general/colord.html
[544] https://medium.com/@boutnaru/the-linux-concept-journey-d-bus-desktop-bus-dd8c69ade019
[545] https://www.freedesktop.org/software/colord/intro.html
[546] https://medium.com/@boutnaru/the-linux-security-journey-polkit-authorization-manager-459c2dd2c3cf
[547] https://www.freedesktop.org/software/colord/specifics.html
[548] https://www.freedesktop.org/software/colord/specifics.html
[549] https://github.com/hughsie/colord

xiccd (X Color Management Daemon)

xiccd (X Color Management Daemon) is an ELF binary located at "/usr/bin/xiccd" and/or "/bin/xiccd". It is basically a bridge between colord[550] and X[551]. It can be used for performing various operations such as (but not limited to): enumerating displays, registering displays in "colord", creating default ICC (International Color Consortium) profiles based on EDID (Extended Display Identification Data), applying ICC profiles and maintaining private ICC storage directory[552].

Overall, it allows non-GNOME and non-KDE desktop environments to load and apply icc profiles[553]. By the way, an ICC profile is set of data that characterizes a color input or output device, or a color space[554]. xiccd is not dependent on any specific desktop environment\toolkit. Also, it should not be used in desktop environments that support color management natively, for example GNOME and KDE[555].

Lastly, one example of a desktop environment is xfce that does not support applying display profiles without xiccd[556]. xiccd is typically launched automatically on login like by leveraging "/etc/xdg/autostart"[557] - as shown in the screenshot below. For more information I suggest going over xiccd's code in its GitHub repository[558].



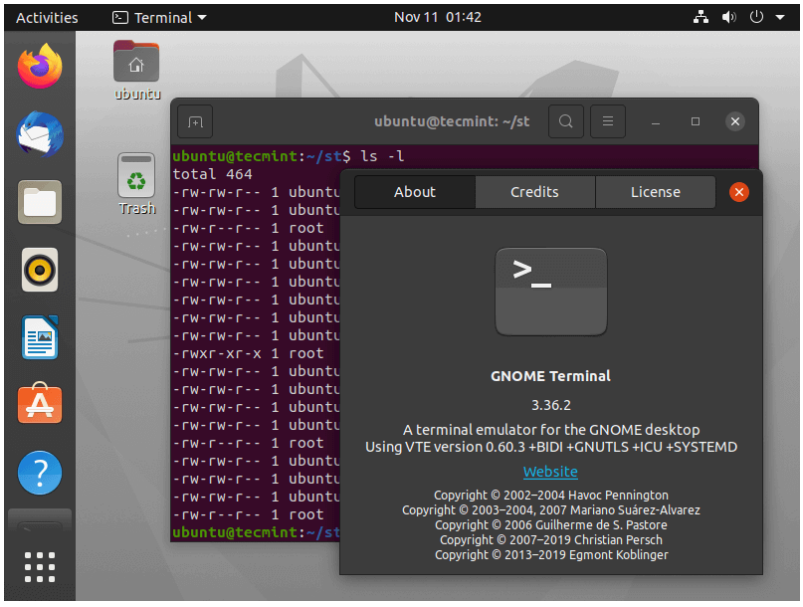
[550] https://medium.com/@boutnaru/the-linux-process-journey-colord-color-daemon-9e718013c679
[551] https://medium.com/@boutnaru/the-linux-concept-journey-xorg-0d7ab9697a34
[552] https://manpages.debian.org/trixie/xiccd/xiccd.8.en.html
[553] https://wiki.archlinux.org/title/ICC_profiles
[554] https://en.wikipedia.org/wiki/ICC_profile
[555] https://community.linuxmint.com/software/view/xiccd
[556] https://docs.xfce.org/xfce/xfce4-settings/color
[557] https://discussion.fedoraproject.org/t/abhis3k-xiccd/83440
[558] https://github.com/agalakhov/xiccd

gnome-terminal

“gnome-terminal” is an ELF binary located at “/usr/bin/gnome-terminal” and/or “//bin/gnome-terminal”. It is a terminal emulator application which can be leveraged for accessing Linux shell environments. For example running programs available on your system[559] - as shown in the screenshot below[560].

Overall, it supports different capabilities such as (but not limited to): using tabs, executing commands, working with text (copy/paste/search/selection/etc), customizing appearance (changing cursor/setting terminal size/zooming in/zooming out/color selection/etc), customizing behaviour (disabling user input/manage profiles/keyboard accessibility/etc) and resetting terminal state[561].

Lastly, “gnome-terminal” has multiple command line switches like which can be clustered to different groups: general information (like “-h”\”--help” for overview of all options), window management (like “--tab” which opens a new tab), executing commands (“--command”\”-c”), appearance management (like “-t”\”--title” for setting the terminal title) and others[562]. For more information I suggest going over the code repository of “gnome-terminal”[563].



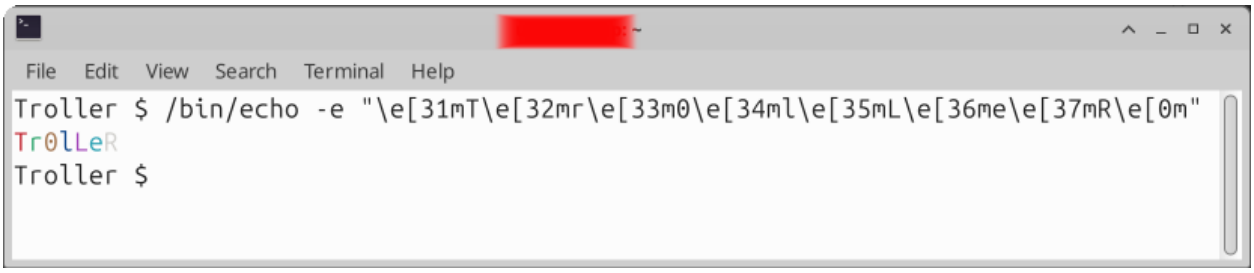
[559] https://help.gnome.org/users/gnome-terminal/stable/introduction.html.en
[560] https://www.tecmint.com/linux-terminal-emulators/
[561] https://help.gnome.org/users/gnome-terminal/stable/
[562] https://gemini.google.com/app/c557a41018e27760
[563] https://gitlab.gnome.org/GNOME/gnome-terminal

echo

“echo” is an ELF binary located at “/usr/bin/echo” and/or “/bin/echo”. Its simplest usage form is to display a line of text[564]. echo also supports specific escaped characters for different use-cases like (but not limited to): horizontal tab (“\t”), vertical tab (“\v”), “new line” (“\n”), remove spaces (“\b”) and the usage of backslash itself (“\\”). Also, when using bash (and and also other shells) we can leverage color displaying using the following pattern: “\e[<STYLE>;<COLOR>m”[565] - as shown in the screenshot below.

Overall, we can use different colors such as (but not limited to): black, green, yellow, blue and purple. Moreover, there is also support for underline, bold, background color and more[566]. By the way, “echo” is part of the coreutils package[567]. Hence, we can also check the source code of “echo” as part of the coreutils’ GitHub repository[568].

Lastly, it is important to understand that shells like bash can have a builtin implementation of “echo”. In that case invoking it without the full path causes the internal implementation to run[569]. There are also equivalent implementations for other operating systems like Windows[570].



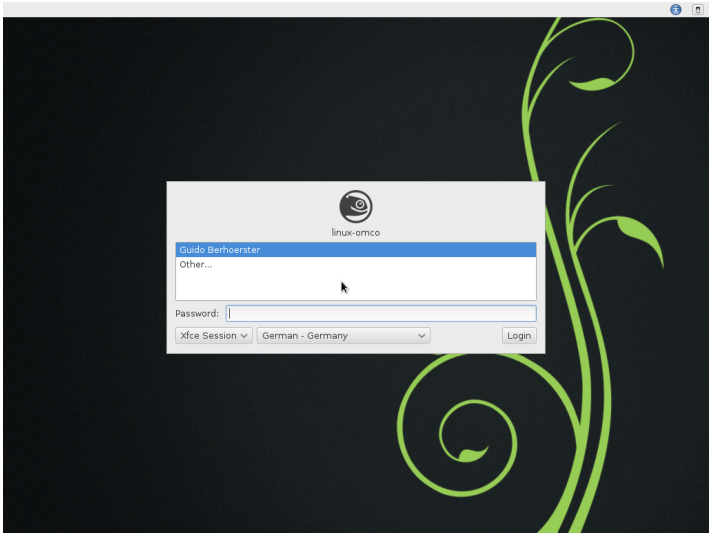
[564] https://linux.die.net/man/1/echo
[565] https://ss64.com/bash/echo.html
[566] https://gist.github.com/lomedil/41b739a74b481c4b0a47fca09f42bea3
[567] https://www.gnu.org/software/coreutils/
[568] https://github.com/coreutils/coreutils/blob/master/src/echo.c
[569] https://www.gnu.org/software/bash/manual/html_node/Bash-Builtins.html
[570] https://ss64.com/nt/echo.html

lightdm

LightDM is an ELF binary located at "/usr/sbin/lightdm" and\or "/sbin/lightdm". It is used as a cross-desktop display manager for Linux based systems[571]. LightDM was developed (at Canonical) as a relatively light-weight and highly customizable alternative to GDM (GNOME Display Manager)[572]. The configuration file is usually stored at "/etc/lightdm/lightdm.conf". By the way, we can create "/etc/lightdm/lightdm.conf.d/" and place our configuration files there[573].

Overall, among LightDM features are (but not limited to): supporting various desktop environments, supporting different display technologies (like X\Wayland\Mir), low memory usage, supporting guest sessions, supporting remote logins (incoming XDMCP\VNC and outgoing XDMCP/PAM) and a comprehensive test suite. Moreover, if we want a GUI that prompts the user for credentials\lets the user select a session\etc we need to install a greeter - as shown below[574] leveraging gtk greeter. It is possible to use LightDM without a greeter only in case of an automatic login[575].

Lastly, by default it is executed with the permissions of the root user[576]. For more information I suggest going over LightDM's GitHub repository. Based on the repo's statistics it is ~89% of the source code is written in the C programming language[577]. As opposed to GDM it does not load any GNOME libraries[578].



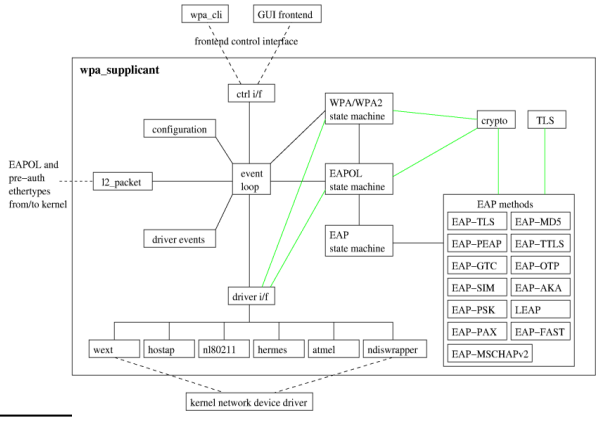
[571] https://medium.com/@boutnaru/the-linux-concept-journey-display-manager-dm-05634c3e8c29
[572] https://wiki.debian.org/LightDM
[573] https://wiki.debian.org/LightDM
[574] https://en.opensuse.org/File:Lightdm-display-manager-gtk-greeter.png
[575] https://wiki.archlinux.org/title/LightDM
[576] https://man.archlinux.org/man/lightdm.1.en
[577] https://github.com/canonical/lightdm
[578] https://en.wikipedia.org/wiki/LightDM

wpa_supplicant (Wi-Fi Protected Access client)

“wpa_supplicant” is an ELF binary located at “/usr/sbin/wpa_supplicant” and/or “/sbin/wpa_supplicant”. It is a cross-platform supplicant (a client device that needs to be verified by a server before using a dedicated network link) with support for WPA\WPA2\WPA3 (IEEE 802.11i). Thus, it is used by desktops\laptops\mobile devices\embedded systems. It implements key negotiation with a WPA authenticator and it controls the roaming and IEEE 802.11 authentication/association of the wireless driver[579].

Overall, “wpa_supplicant” is meant for executing as a "daemon" (user-mode) in the background and acting as the backend component controlling the wireless connection. Hence, it will run with the permissions of the “root” user. It is important to know that “wpa_supplicant” has also frontend programs “wpa_cli” (text-based frontend) and “wpa_gui” (GUI based)[580]. As an example the phases associating with an AP using WPA are as follows. First, requests the kernel driver to scan neighboring BSSes and then selects and associates with one. If using WPA-EAP, it completes EAP authentication to get the master key; if using WPA-PSK, it uses the PSK as the master key. Next, completing the WPA 4-Way and Group Key Handshakes with the AP. In the last phase, it configures the encryption keys, allowing for normal data transmission[581].

Lastly, “wpa_supplicant” supports different configurations stored by default at “/etc/wpa_supplicant/”. Probably the most fundamental one is “wpa_supplicant.conf”[582]. For more details about the configuration file I suggest going over a detailed one as an example[583]. Also, “wpa_supplicant” exposes a control interface for providing the ability of external programs to control the operations of the daemon and to get status information and event notifications. For more information I suggest going over the source code[584]. A general map of the modules related to “wpa_supplicant” is shown below[585].



[579] https://wiki.archlinux.org/title/Wpa_supplicant
[580] https://w1.fi/wpa_supplicant/
[581] https://linux.die.net/man/8/wpa_supplicant
[582] https://wiki.gentoo.org/wiki/Wpa_supplicant
[583] https://git.w1.fi/cgit/hostap/tree/wpa_supplicant/wpa_supplicant.conf
[584] https://git.w1.fi/cgit/hostap/
[585] https://w1.fi/wpa_supplicant/devel/

wpa_cli (Wi-Fi Protected Access Command Line Client)

“wpa_cli” (Wi-Fi Protected Access Command Line Client) is an ELF binary located at “/usr/sbin/wpa_cli” and/or “/sbin/wpa_cli”. It is a text-based frontend program for which interacts with the user-mode daemon “wpa_supplicant”[586]. We can leverage it for querying current status\changing configuration\triggering events and requesting interactive user input[587].

Overall, “wpa_cli” provides an interactive mode which supports tab compilation and description of complemented commands - as shown in the screenshot below. By default, “wpa_cli” uses the control socket located at “/var/run/wpa_supplicant/”. It is important to know that for running “wpa_cli” with all of its features we need to have root permissions or the relevant capabilities[588].

Lastly, for more information about the commands supported by “wpa_cli” and their arguments I suggest going over different cheatsheet[589]. Moreover, “wpa_cli” is packaged as part of the “wpa_supplicant” in most Linux based operating systems[590]. In order to get a better understanding about “wpa_cli” we can go over its source code[591].

```
> scan
OK
<3>CTRL-EVENT-SCAN-RESULTS
> scan_results
bssid / frequency / signal level / flags / ssid
00:00:00:00:00:00 2462 -49 [WPA2-PSK-CCMP][ESS] MYSSID
11:11:11:11:11:11 2437 -64 [WPA2-PSK-CCMP][ESS] ANOTHERSSID
```

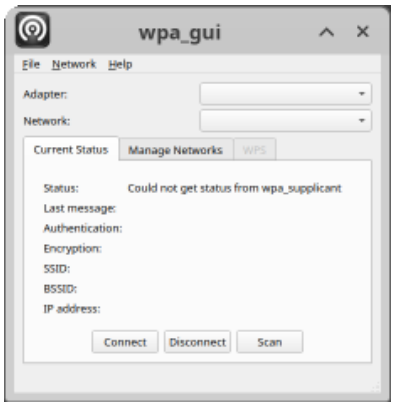
[586] https://medium.com/@boutnaru/the-linux-process-journey-wpa-supplicant-wi-fi-protected-access-client-cef744a71e00
[587] https://linux.die.net/man/8/wpa_cli
[588] https://wiki.archlinux.org/title/Wpa_supplicant
[589] https://gist.github.com/penguinpowernz/1d36a38af4fac4553562410e0bd8d6cf
[590] https://packages.debian.org/trixie/amd64/wpasupplicant/filelist
[591] https://android.googlesource.com/platform/external/wpa_supplicant/+/6eb6225dda14c6eaf34fcb3498e17d28060e1e16/wpa_cli.c

wpa_gui (Wi-Fi Protected Access Graphical User Interface)

“wpa_gui” (Wi-Fi Protected Access Graphical User Interface) is an ELF binary. We can find it at “/usr/sbin/wpa_gui” and/or “/sbin/wpa_gui”. However, those locations can be a symbolic link to “/usr/bin/wpa_gui” and\or “/bin/wpa_gui”. It is a GUI based frontend program for which interacts with the user-mode daemon “wpa_supplicant”[592] - as shown below. Thus, we can use “wpa_gui” instead of NetworkManager[593] for configuring and managing wireless interfaces[594].

Overall, “wpa_gui” is based on QT which supports almost all the interactive statuses\configuration features of “wpa_cli”[595]. It also uses that default location for “wpa_supplicant” that is “/var/run/wpa_supplicant/”. It can be overridden using the “-p” command line switch of “wpa_gui”[596].

Lastly, on some Linux distributions (like Ubuntu) “wpa_gui” is included in a different package from “wpa_cli” and “wpa_supplicant” called “wpagui”[597]. For more information I suggest going over an implementation of “wpagui”[598].



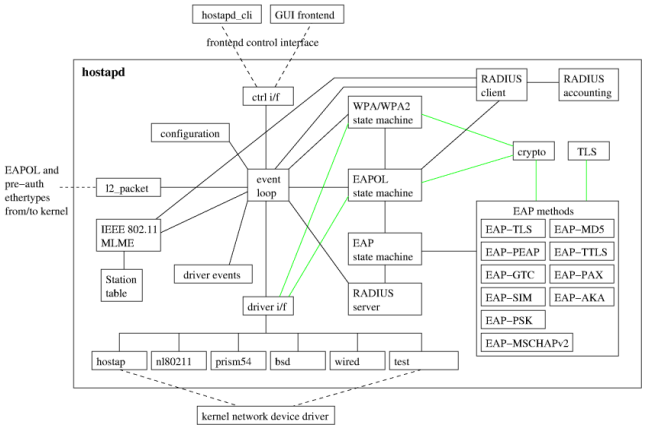
[592] https://medium.com/@boutnaru/the-linux-process-journey-wpa-supplicant-wi-fi-protected-access-client-cef744a71e00
[593] https://medium.com/@boutnaru/the-linux-process-journey-modemmanager-modem-management-daemon-71d3b5cf9db1
[594] https://ansible-linux-postinstall.readthedocs.io/en/latest/guide-task-wpagui.html
[595] https://medium.com/@boutnaru/the-linux-process-journey-wpa-cli-wi-fi-protected-access-command-line-client-931dc8b28ffb
[596] https://linux.die.net/man/8/wpa_gui
[597] https://launchpad.net/ubuntu/noble/+package/wpagui
[598] https://android.googlesource.com/platform/external/wpa_supplicant/+/refs/heads/tools_r20/wpa_gui/

hostapd (Host Access Point Daemon)

“hostapd” (Host Access Point Daemon) is an ELF binary located at “/usr/sbin/hostapd” and\or “/sbin/hostapd”. It is a user-mode daemon used as an AP (access point) and an authentication server. Among the protocols which are supported are: IEEE 802.1X/WPA/EAP/RADUIS[599]. By the way, in order to communicate with a kernel driver it leverages different interfaces. All new cfg80211 (and mac80211) based drivers that implement AP functionality are supported using the nl80211 interface. For old kernel drivers it has drivers, there are 3 other drivers you can use: “madwifi”, “prism54” and “HostAP”[600].

Overall, hostapd provides various capabilities such as (but not limited to): creating an AP, creating multiple APs on the he same card (if the card supports it), create different APs on different interfaces using the same instance of the daemon and using both 2.4GHz and 5GHz at the same time on the same card (requires a support form the hardware side also). It is important to understand that there are features not implemented\supported by hostapd like: assigning an IP address to the AP itself, assigning IPs to the devices connecting to the AP (DHCP server is needed), creating a dual-band AP, even with two cards (it can create two APs with the same SSID) and creating multiple APs on different channels (on the same card). Multiple APs on the same card will share the same channel[601].

Lastly, the source code of “hostapd” is divided into separate C files - as shown below. All hardware/driver specific functionality is in separate files that implement a well-defined driver AP[602]. There is also specific information regarding porting to different target boards and operating systems[603]. For more information I suggest going over the source code of hostapd[604].



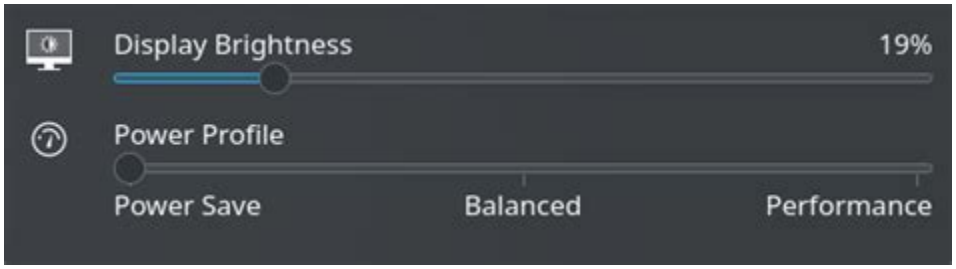
[599] https://w1.fi/
[600] https://wireless.docs.kernel.org/en/latest/en/users/documentation/hostapd.html
[601] https://wiki.gentoo.org/wiki/Hostapd
[602] https://w1.fi/wpa_supplicant/devel/
[603] https://w1.fi/wpa_supplicant/devel/porting.html
[604] https://w1.fi/wpa_supplicant/devel/dir_68267d1309a1af8e8297ef4c3efbcdba.html

power-profiles-daemon

“power-profiles-daemon” is an ELF binary located by default at “/usr/libexec/power-profiles-daemon”. It is used for modifying the system power/behavior state. Thus, it is leveraged on various laptops\desktop environments to activate power saving and\or performance CPU governors through D-BUS[605]. “power-profiles-daemon” is used as a tool for setting the CPU governor in order to increase system performance at the cost of energy usage[606]. It is mostly executed with the permissions of the “root” user.

Overall, there are three different power profiles that a user can select from: "balanced" (default mode), "power-saver" mode and "performance" mode. The first two are available on every system while "performance" is available on selected systems only (implemented by different "drivers"). By the way, we can also hook to specific changes in the behaviour of a particular device. For example, we can disable fast-charging for some USB devices in power-saver mode[607].

Lastly, the selection of profiles can be done by using different settings as part of the desktop environment - as shown in the screenshot below from Arch Linux running KDE[608]. Hence, we can summarize that “power-profiles-daemon” is a CPU throttle that is most effective when high\medium load applications are in use[609]. For more information I suggest going over the code repository of “power-profiles-daemon”[610].



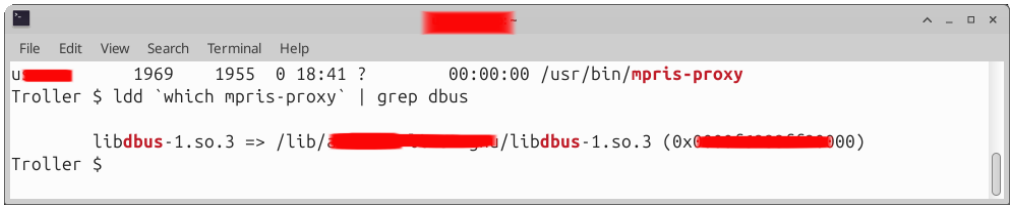
[605] https://medium.com/@boutnaru/the-linux-concept-journey-d-bus-desktop-bus-dd8c69ade019
[606] https://www.linuxfromscratch.org/blfs/view/systemd/general/power-profiles-daemon.html
[607] https://github.com/Rongronggg9/power-profiles-daemon
[608] https://www.reddit.com/r/kde/comments/r74fjp/change_cpu_governor_using_powerprofilesdaemon/
[609] https://linrunner.de/tlp/faq/ppd.html
[610] https://gitlab.freedesktop.org/upower/power-profiles-daemon

mpris-proxy (Media Player Remote Interfacing Specification Proxy)

“mpris-proxy” (Media Player Remote Interfacing Specification Proxy) is an ELF binary located at “/usr/bin/mpris-proxy” and\or “/bin/mpris-proxy”. It is part of the “bluez” package[611]. MPRIS is a standard D-Bus[612] interface which is used for providing a common API for controlling media players - as shown below. This API is leveraged for different tasks such as (but not limited to) discovery and playback[613].

Overall, by using “mpris-proxy” we enable a proxy forwarding Bluetooth MIDI (Musical Instrument Digital Interface) controls via MPRIS to control media players[614]. The process of the “mpris-proxy” is executed with the permissions of the logged on user - as shown below.

Lastly, we can summarize the functionality of “mpris-proxy” to the following: registering media players, interface translation (BlueZ<->MPRIS), volume management, cover art support and more . For more information I suggest going over the source code of “mpris-proxy” as part of the “bluez” GitHub’s repository[615].



[611] https://askubuntu.com/questions/1349701/bluetooth-speaker-or-headphones-volume-and-playback-control-mpris
[612] https://medium.com/@boutnaru/the-linux-concept-journey-d-bus-desktop-bus-dd8c69ade019
[613] https://wiki.archlinux.org/title/MPRIS
[614] https://mynixos.com/home-manager/options/services.mpris-proxy
[615] https://github.com/bluez/bluez/blob/master/tools/mpris-proxy.c