

The Linux Concept Journey

Version 4.0

July-2024

By Dr. Shlomi Boutnaru



Created using [Craiyon AI Image Generator](#)

Introduction.....	5
The Auxiliary Vector (AUXV).....	6
command not found.....	7
Out-of-Memory Killer (OOM killer).....	8
Why doesn't "ltrace" work on new versions of Ubuntu?.....	9
vDSO (Virtual Dynamic Shared Object).....	10
Calling syscalls from Python.....	12
Syscalls' Naming Rule: What if a syscall's name starts with "f"?.....	13
Syscalls' Naming Rule: What if a syscall's name starts with "l"?.....	14
RCU (Read Copy Update).....	15
cgroups (Control Groups).....	17
Package Managers.....	18
What is an ELF (Executable and Linkable Format) ?.....	19
The ELF (Executable and Linkable Format) Header.....	20
File System Hierarchy in Linux.....	21
/boot/config-\$(uname-r).....	23
/proc/config.gz.....	24
What is an inode?.....	25
Why is removing a file not dependent on the file's permissions?.....	26
VFS (Virtual File System).....	27
tmpfs (Temporary Filesystem).....	28
ramfs (Random Access Memory Filesystem).....	29
Buddy Memory Allocation.....	30
DeviceTree.....	31
How can we recover a deleted executable of a running application?.....	32
Process Group.....	32
<Major:Minor> Numbers.....	34
Monolithic Kernel.....	35
Loadable Kernel Module (LKM).....	36
Builtin Kernel Modules.....	37
Signals.....	38
Real Time Signals.....	39
Memory Management - Introduction.....	40
Hard Link.....	41
Soft Link.....	42
BusyBox.....	43
Character Devices.....	44
Block Devices.....	45
Unnamed Pipe (Anonymous Pipe).....	46
Linux File Types.....	47

Regular File.....	48
Directory File.....	49
Link File (aka Symbolic Link).....	50
Socket File.....	51

Introduction

When starting to learn Linux I believe that they are basic concepts that everyone needs to know about. Because of that I have decided to write a series of short writeups aimed at providing the basic vocabulary and understanding for achieving that.

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.

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>. You can also find my other free eBooks at <https://TheLearningJourney.com>.

Lets GO!!!!!!

The Auxiliary Vector (AUXV)

There are specific OS variables that a program would probably want to query such as the size of a page (part of memory management - for a future writeup). So how can it be done?

When the OS executes a program it exposes information about the environment in a key-value data store called “auxiliary vector” (in short auxv/AUXV). If we want to check which keys are available we can go over¹ (on older versions it was part of elf.h) and look for all the defines that start with “AT_”.

Among the information that is included in AUXV we can find: the effective uid of the program, the real uid of the program, the system page size, number of program headers (of the ELF - more on that in the future), minimal stack size for signal delivery (and there is more).

If we want to see all the info of AUXV while running a program we can set the LD_SHOW_AUXV environment variable to 1 and execute the requested program (see the screenshot below, it was taken from JSLinux running Fedora 33 based on a riscv64 CPU². We can see that the name of the variable starts with “LD_”, it is because it is used/parsed by the dynamic linker/loader (aka ld.so).

Thus, if we statically link our program (like using the -static flag on gcc) setting the variable won't print the values of AUXV. Anyhow, we can also access the values in AUXV using the “unsigned long getauxval(unsigned long type)” library function³. A nice fact is that the auxiliary vector is located next to the environment variables check the following illustration⁴.

```
[root@localhost ~]# export LD_SHOW_AUXV=1
[root@localhost ~]# uname -a
AT_SYSINFO_EHDR: 0x200001f000
AT_HWCAP: 112d
AT_PAGESZ: 4096
AT_CLKTCK: 100
AT_PHDR: 0x2aaaaaa040
AT_PHENT: 56
AT_PHNUM: 9
AT_BASE: 0x2000000000
AT_FLAGS: 0x0
AT_ENTRY: 0x2aaaaabd38
AT_UID: 0
AT_EUID: 0
AT_GID: 0
AT_EGID: 0
AT_SECURE: 0
AT_RANDOM: 0x3fffff6821
AT_EXECFN: /bin/uname
Linux localhost 4.15.0-00049-ga3b1e7a-dirty #11 Thu Nov 8 20:30:26 CET 2018 riscv64 riscv64 riscv64 GNU/Linux
```

¹ <https://elixir.bootlin.com/linux/latest/source/include/uapi/linux/auxvec.h#L10>

² <https://bellard.org/jslinux/>

³ <https://man7.org/linux/man-pages/man3/getauxval.3.html>

⁴ <https://static.lwn.net/images/2012/auxvec.png>

command not found

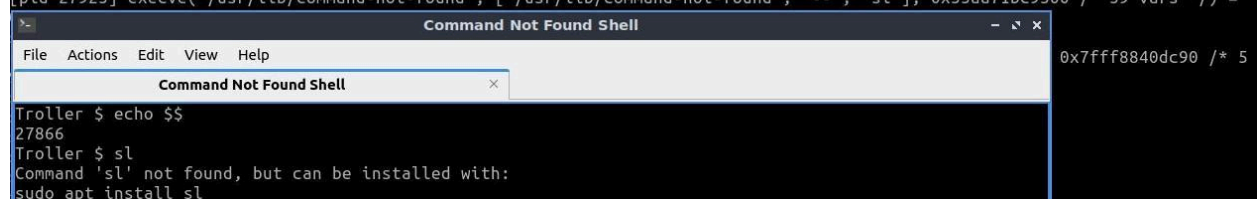
Have you ever asked yourself what happens when you see “command not found” on bash? This writeup is not going to talk about that and not about the flow which determines if a command is found or not (that is a topic for a different write up ;-).

I am going to focus my discussion on what happens in an environment based on bash + Ubuntu (version 22.04). I guess you at least once wrote “sl” instead of “ls” and you got a message “Command 'sl' not found, but can be installed with: sudo apt install sl” - how did bash know that there is such a package that could be installed? - as shown in the screenshot below

Overall, the magic happens with the python script “/usr/lib/command-not-found” which is executed when a bash does not find a command - as shown in the screenshot below. This feature is based on an sqlite database that has a connection between command and packages, it is sorted in “/var/lib/command-not-found/commands.db”.

Lastly, there is a nice website <https://command-not-found.com/> which allows you to search for a command and get a list of different ways of installing it (for different Linux distributions/Windows/MacOS/Docker/etc).

```
Troller$ sudo strace -f -e execve -p 27866
strace: Process 27866 attached
strace: Process 27924 attached
strace: Process 27925 attached
[pid 27925] execve("/usr/lib/command-not-found", ["/usr/lib/command-not-found", "-.", "sl"], 0x55aa71bc9300 /* 59 vars */) =
0x7fff8840dc90 /* 5
```



The screenshot shows a terminal window titled "Command Not Found Shell". The terminal output is as follows:

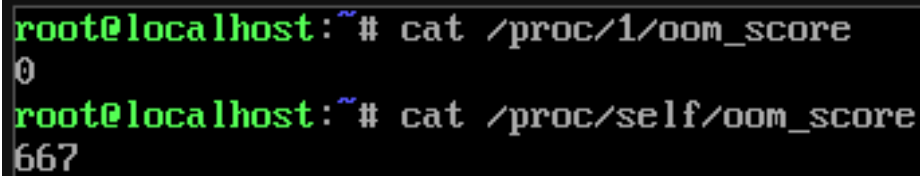
```
Troller $ echo $$
27866
Troller $ sl
Command 'sl' not found, but can be installed with:
sudo apt install sl
```

Out-of-Memory Killer (OOM killer)

The Linux kernel has a mechanism called “out-of-memory killer” (aka OOM killer) which is used to recover memory on a system. The OOM killer allows killing a single task (called also oom victim) while that task will terminate in a reasonable time and thus free up memory.

When OOM killer does its job we can find indications about that by searching the logs (like /var/log/messages and grepping for “Killed”). If you want to configure the “OOM killer”⁵.

It is important to understand that the OOM killer chooses between processes based on the “oom_score”. If you want to see the value for a specific process we can just read “/proc/[PID]/oom_score” - as shown in the screenshot below. If we want to alter the score we can do it using “/proc/[PID]/oom_score_adj” - as shown also in the screenshot below. The valid range is from 0 (never kill) to 1000 (always kill), the lower the value is the lower is the probability the process will be killed⁶.



```
root@localhost:~ # cat /proc/1/oom_score
0
root@localhost:~ # cat /proc/self/oom_score
667
```

⁵ <https://www.oracle.com/technical-resources/articles/it-infrastructure/dev-oom-killer.html>

⁶ <https://man7.org/linux/man-pages/man5/proc.5.html>

Why doesn't "ltrace" work on new versions of Ubuntu?

Many folks have asked me about that, so I have decided to write a short answer about it. Two well known command line tools on Linux which can help with dynamic analysis are "strace" and "ltrace". "strace" allows tracing of system calls ("man 2 syscalls") and signals ("man 7 signal"). I am not going to focus on "strace" in this writeup, you can read more about it using "man strace". On the other hand, "ltrace" allows the tracing of dynamic library calls and signals received by the traced process ("man 1 ltrace"). Moreover, it can also trace syscalls (like "strace") if you are using the "-S" flag.

If you have tried using "ltrace" in the new versions of Ubuntu you probably saw that the library calls are not shown (you can verify it using "ltrace `which ls`"). In order to demonstrate that I have created a small c program - as you can see in the screenshot below ("code.c").

First, if we compile "code.c" and run it using "ltrace" we don't get any information about a library call (see in the screenshot below). Second, if we compile "code.c" with "-z lazy" we can see when running the executable with "ltrace" we do get information about the library functions. So what is the difference between the two?

"ltrace" (and "strace") works by inserting a breakpoint⁷ in the PLT for the relevant symbol (that is library function) we want to trace. So because by default the binaries are not compiled with "lazy loading" of symbols they are resolved when the application starts and thus the breakpoints set by "ltrace" are not triggered (and we don't see any library calls in the output - as shown in the screenshot below). Also, you can read more about the internals of "ltrace" here - <https://www.kernel.org/doc/ols/2007/ols2007v1-pages-41-52.pdf>

```
Troller # cat code.c
#include <stdio.h>

void main()
{
    printf("\n*****!!!!!!!\n");
}
Troller # gcc code.c -o code
Troller # ./code

*****!!!!!!!
Troller # ltrace ./code

*****!!!!!!!
+++ exited (status 24) +++
Troller # gcc -z lazy code.c -o ./code
Troller # ./code

*****!!!!!!!
Troller # ltrace ./code
puts("\n*****!!!!!!!")
*****!!!!!!!
)
+++ exited (status 24) +++
Troller #
```

⁷<https://medium.com/@boutnaru/have-you-ever-asked-yourself-how-breakpoints-work-c72dd8619538>

vDSO (Virtual Dynamic Shared Object)

vDSO is a shared library that the kernel maps into the memory address space of every user-mode application. It is not something that developers need to think about due to the fact it is used by the C library⁸.

Overall, the reason for even having vDSO is the fact they are specific system calls that are used frequently by user-mode applications. Due to the time/cost of the context-switching between user-mode and kernel-mode in order to execute a syscall it might impact the overall performance of an application.

Thus, vDSO provides “virtual syscalls” due to the need for optimizing system calls implementations. The solution needed to not require libc to track CPU capabilities and or the kernel version. Let us take for example x86, which has two ways of invoking a syscall: “int 0x80” or “sysenter”. The “sysenter” option is faster, due to the fact we don’t need to through the IDT (Interrupt Descriptor Table). The problem is it is supported for CPUs newer than Pentium II and for kernel versions greater than 2.6⁹.

If you vDSO the implementation of the syscall interface is defined by the kernel in the following manner. A set of CPU instructions formatted as ELF is mapped to the end of the user-mode address space of all processes - as shown in the screenshot below. When libc needs to execute a syscall it checks for the presence of vDSO and if it is relevant for the specific syscalls the implementation in vDSO is going to be used - as shown in the screenshot below¹⁰.

Moreover, for the case of “virtual syscalls” (which are also part of vDSO) there is a frame mapped as two different pages. One in kernel space which is static/”readable & writeable” and the second one in user space which is marked as “read-only”. Two examples for that are the syscalls “getpid()” (which is a static data example) and “gettimeofday()” (which is a dynamic read-write example).

Also, as part of the kernel compilation process the vDSO code is compiled and linked. We can most of the time find it using the following command “find arch/\$ARCH/ -name '*vdso*.so*' -o -name '*gate*.so*'”¹¹

If we want to enable/disable vDSO we can set “/proc/sys/vm/vdso_enable” to 1/0 respectively¹². Lastly, a benchmark of different syscalls using different implementations is shown below.

⁸ <https://man7.org/linux/man-pages/man7/vdso.7.html>

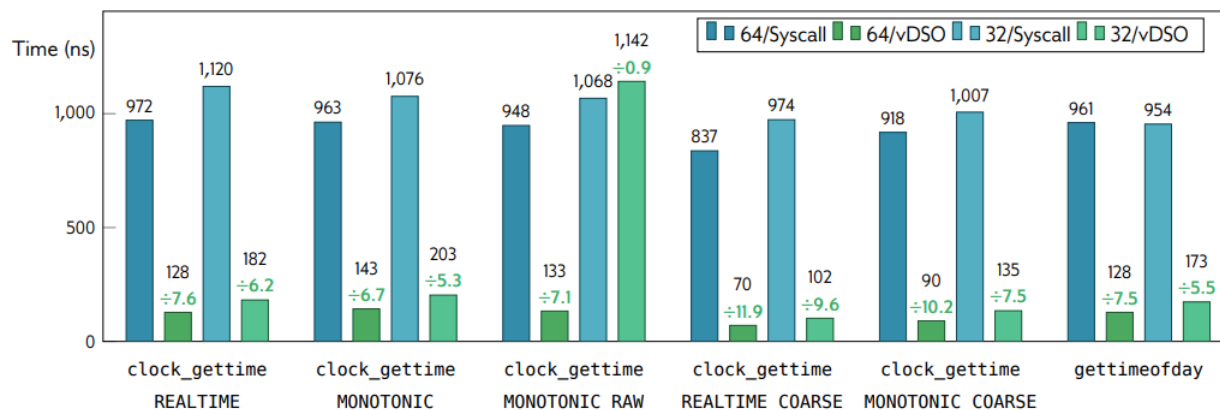
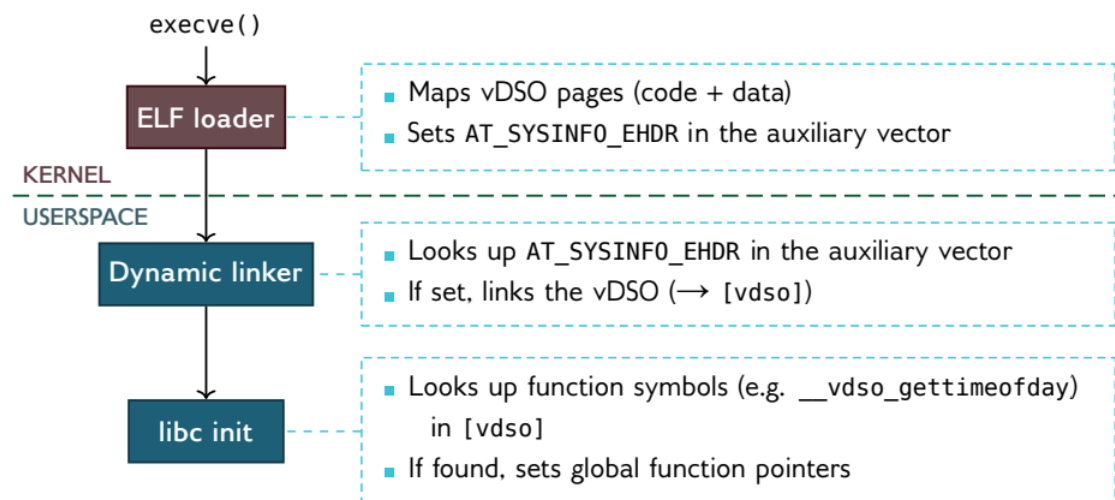
⁹ <https://linux-kernel-labs.github.io/refs/heads/master/so2/lec2-syscalls.html>

¹⁰ <https://hackmd.io/@sysprog/linux-vdso>

¹¹ <https://manpages.ubuntu.com/manpages/xenial/man7/vdso.7.html>

¹² <https://talk.maemo.org/showthread.php?t=32696>

Kernel and userspace setup



Calling syscalls from Python

Have you ever wanted a quick way to call a syscall (even if it is not exposed by libc)? There is a quick way of doing that using “ctypes” in Python.

We can do it using the “syscall” exported function by libc (check out ‘man 2 syscall’¹³ for more information). By calling that function we can call any syscall by passing its number and parameters.

How do we know what the number of the syscall is? We can just check <https://filippo.io/linux-syscall-table/>. What about the parameters? We can just go to the source code which is pointed in any entry of a syscall (from the previous link) or we can just use man (using the following pattern - ‘man 2 {NameOfSyscall}’, for example ‘man 2 getpid’).

Let us see an example, we will use the syscall getpid(), which does not get any arguments. Also, the number of the syscall is 39 (on x64 Linux). You can check the screenshot below for the full example. By the way, the example was made with https://www.tutorialspoint.com/linux_terminal_online.php and online Linux terminal (kernel 3.10).

```
$ python3
Python 3.8.6 (default, Jan 29 2021, 17:38:16)
[GCC 8.4.1 20200928 (Red Hat 8.4.1-1)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import ctypes
>>> libc=ctypes.CDLL(None)
>>> libc.syscall(39)
47617
```

¹³ <https://man7.org/linux/man-pages/man2/syscall.2.html>

Syscalls' Naming Rule: What if a syscall's name starts with "f"?

Due to the large number of syscalls, there are some naming rules used in order to help in understanding the operation performed by each of them. Let me go over some of them to give more clarity.

If we have a syscall "<syscall_name>" so we could also have "f<syscall_name>" which means that "f<syscall_name>" does the same operation as "<syscall_name>" but on a file referenced by an fd (file descriptor). Some examples are ("chown", "fchown") and ("stat", "fstat"). It is important to understand that not every syscall which starts with "f" is part of such a pair, look at "fsync()" as an example, however in this case the prefix still denoting the input of the syscall is an fd. There are also examples in which the "f" prefix does not even refer to an fd like in the case of "fork()", it is just part of the syscall name.

Syscalls' Naming Rule: What if a syscall's name starts with "l"?

I want to talk about those syscalls starting with "l". If we have a syscall "<syscall_name>" so we could also have "l<syscall_name>" which means that "l<syscall_name>" does the same operation as "<syscall_name>" but in case of a symbolic link given as input the information is retrieved about the link itself and not the file that the link refers to (for example "getxattr" and "lgetxattr". Moreover, not every syscall that starts with "l" falls in this category (think about "listen").

I think the last rule is the most confusing one because there are cases in which the "l" prefix is not part of the original name of the syscall and is not relevant to any type of links. Let us look at "lseek", the reason for having the prefix is to emphasize that the offset is given as long as opposed to the old "seek" syscall.

RCU (Read Copy Update)

Because there are multiple kernel threads (check it out using `ps -ef | grep rcu` - the output of the command is included in the screenshot at the end of the post) which are based on RCU (and other parts of the kernel) . I have decided to write a short explanation about it.

RCU is a sync mechanism which avoids the use of locking primitives in case multiple execution flows that read and write specific elements. Those elements are most of the times linked by pointers and are part of a specific data structure such as: has tables, binary trees, linked lists and more.

The main idea behind RCU is to break down the update phase into two different steps: “reclamation” and “removal” - let’s detail those phases. In the “removal” phase we remove/unlink/delete a reference to an element in a data structure (can be also in case of replacing an element with a new one). That phase can be done concurrently with other readers. It is safe due to the fact that modern CPUs ensure that readers will see the new or the old data but not partially updated. In the “reclamation” step the goal is to free the element from the data structure during the removal process. Because of that this step can disrupt a reader which references that specific element. Thus, this step must start only after all readers don’t hold a reference to the element we want to remove.

Due to the nature of the two steps an updater can finish the “removal” step immediately and defer the “reclamation” for the time all the active during this phase will complete (it can be done in various ways such as blocking or registering a callback).

RCU is used in cases where read performance is crucial but can bear the tradeoff of using more memory/space. Let’s go over a sequence of an update to a data structure in place using RCU. First, we just create a new data structure. Second, we copy the old data structure into the new one (don’t forget to save the pointer to the old data structure). Third, alter the new/copied data structure. Fourth, update the global pointer to reference the new data structure. Fifth, sleep until the kernel is sure they are no more readers using the old data structure (called also grace period, in Linux we can use `synchronize_rcu()`)¹⁴.

In summary, RCU is probably the most common “lock-free” technique for shared data structures. It is lock-free for any number of readers. There are implementations also for single-writer and even multi-writers (However, it is out of scope for now). Of course, RCU also has problems and it is not designed for cases in which there are update-only scenarios (it is better for “mostly-read” and “few-writes”) - More about that in a future writeup.

¹⁴ <https://elixir.bootlin.com/linux/latest/source/kernel/rcu/tree.c#L3796>

root	3	2	0	07:15	?	00:00:00	[rcu_gp]
root	4	2	0	07:15	?	00:00:00	[rcu_par_gp]
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	13	2	0	07:15	?	00:00:02	[rcu_sched]

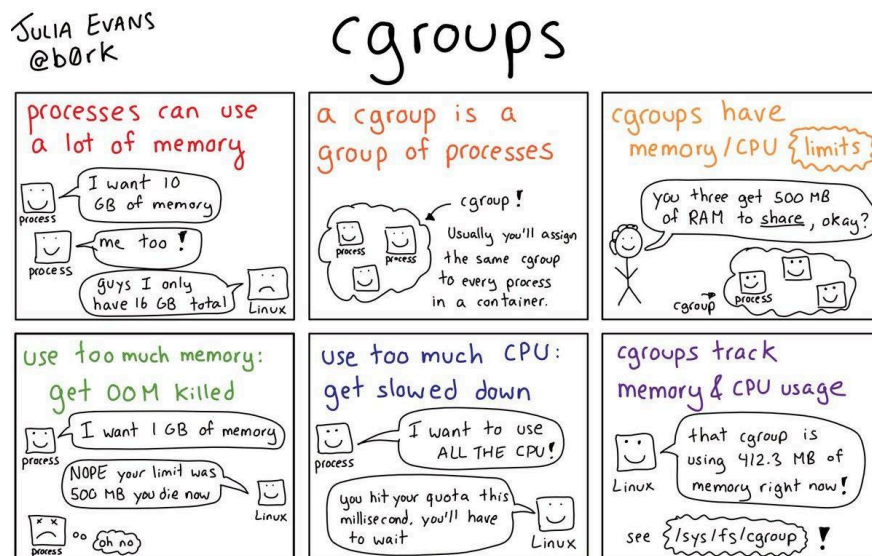
cgroups (Control Groups)

“Control Groups” (aka cgroups) is a Linux kernel feature that organizes processes into hierarchical groups. Based on those groups we can limit and monitor different types of OS resources. Among those resources are: disk I/O usage, network usage, memory usage, CPU usage and more (<https://man7.org/linux/man-pages/man7/cgroups.7.html>). cgroups are one of the building blocks used for creating containers (which include other stuff like namespaces, capabilities and overlay filesystems).

The cgroups functionality has been merged into the Linux kernel since version 2.6.24 (released in January 2008). Overall, cgroups provide the following features: resource limiting (as explained above), prioritization (some process groups can have larger shares of resources), control (freezing group of processes) and accounting¹⁵.

Moreover, there are two versions of cgroups. cgroups v1 was created by Paul Menage and Rohit Seth. cgroups v2 was redesigned and rewritten by Tejun Heo¹⁶. The documentation for cgroups v2 first appeared in the Linux kernel 4.5 release on March 2016¹⁷.

I will write on the differences between the two versions and how to use them in the upcoming writeups. A nice explanation regarding the concept of cgroups is shown in the image below¹⁸. By the way, since kernel 4.19 OOM killer¹⁹ is aware of cgroups, which means the OS can kill a cgroup as a single unit.



¹⁵ <https://docs.kernel.org/admin-guide/cgroup-v1/cgroups.html>

¹⁶ <https://www.wikiwand.com/en/Cgroups>

¹⁷ <https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/diff/Documentation/cgroup-v2.txt?id=v4.5&id2=v4.4>

¹⁸ <https://twitter.com/b0rk/status/1214341831049252870>

¹⁹ <https://medium.com/@boutnaru/linux-out-of-memory-killer-oom-killer-bb2523da15fc>

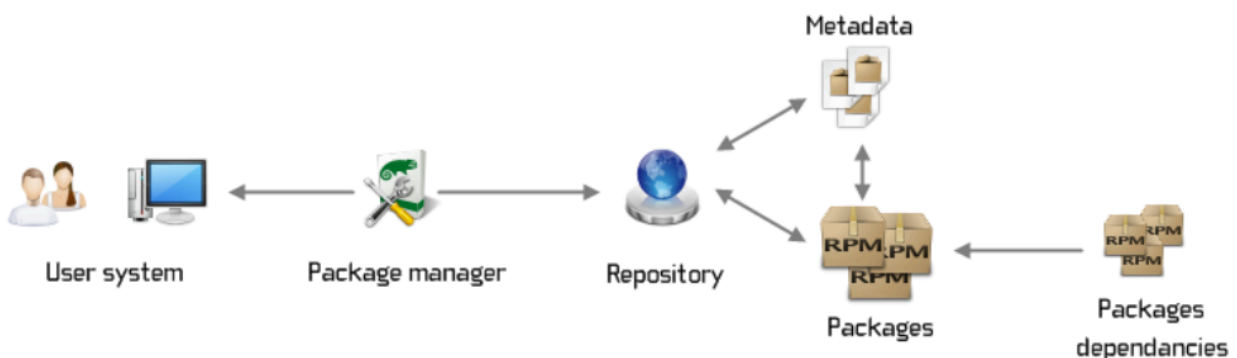
Package Managers

Package manager (aka “Package Management System”) is a set of software components which are responsible for tracking what software artifacts (executables, scripts, shared libraries and more). Packages are defined as a bundle of software artifacts that can be installed/removed as a group (<https://www.debian.org/doc/manuals/aptitude/pr01s02.en.html>).

Thus, we can say that a package manager automates the installation/upgrading/removing of computer programs from a system in a consistent manner. Moreover, package managers often manage a database that includes the dependencies between the software artifacts and version information in order to avoid conflicts (https://en.wikipedia.org/wiki/Package_manager).

Basically, there are different categories of package managers. The most common are: OS package managers (like dpkg, apk, rpm, dnf, pacman and more - part are only frontends as we will describe in the future) and runtime package managers focused on specific programming languages (like maven, npm, PyPi, NuGet, Composer and more). Each package manager can also have its own package file format (more on those in future writeups). Moreover, package managers can have different front-ends CLI based or GUI based. Those package managers can also support downloading software artifacts from different repositories (<https://devopedia.org/package-manager>).

Overall, package managers can store different metadata for each package like: list of files, version, authorship, licenses, targeted architecture and checksums. An overview of the package management flow is shown in the diagram below (<https://developerexperience.io/articles/package-management>).



What is an ELF (Executable and Linkable Format) ?

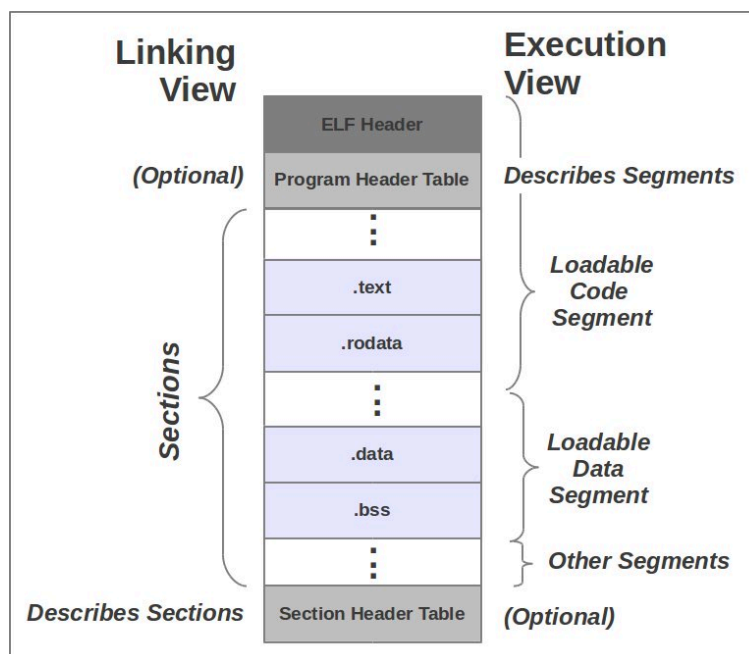
Every generic/standard operating system has a binary format for its user mode executables/libraries, kernel code and more. Windows has PE (Portable Executable), OSX has MachO and Linux has ELF. We are going to start with ELF (I promise to go over the others also).

In Linux ELF among the others (but not limited to) for executables, kernel models, shared libraries, core dumps and object files. Although, Linux does not mandates an extension for files ELF files may have an extension of *.bin, *.elf, *.ko, *.so, *.mod, *.o, *.bin and more (it could also be without an extension).

Moreover, today ELF is a common executable format for a variety of operating systems (and not only Linux) like: QNX, Android, VxWorks, OpenBSD, NetBSD, FreeBSD, Fuchsia, BeOS. Also, it is used in different platforms such as: Wii, Dreamcast and Playstation Portable.

ELF, might include 3 types of headers: ELF header (which is mandatory), program headers and sections header . The appearance of the last two is based on the goal of the file: Is it for linking only? Is it execution only? Both? (More on the difference between the two in the next chapters). You can see the different layouts of ELF in the image below²⁰.

In the next parts we will go over each header in more detail. By the way, a great source for more information about ELF is man ("man 5 elf").



²⁰ <https://i.stack.imgur.com/RMV0g.png>

The ELF (Executable and Linkable Format) Header

Now we are going to start with the ELF header. Total size of the header is 32 bytes. The header starts with the magic “ELF” (0x7f 0x45 0x4c 0x46).

From the information contained in the header we can answer the following questions: Is the file 32 or 64 bit? Does the file store data in big or little endian? What is the ELF version? The type of the file (executable, relocatable, shared library, etc)? What is the target CPU? What is the address of the entry point? What is the size of the other headers (program/section)? - and more.

If we want to parse the header of a specific ELF file we can use the command “readelf” (we are going to use it across all of the next parts to parse ELF files). In order to show the header of an ELF file we can run “readelf -h {PATH_TO_ELF_FILE}”. In the image below we can see the ELF header of “ls”. The image was taken from an online Arch Linux in a browser (copy.sh).

```
root@localhost:~# readelf -h `which ls`
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                                ELF32
  Data:                                      2's complement, little endian
  Version:                               1 (current)
  OS/ABI:                                UNIX - System V
  ABI Version:                           0
  Type:                                  DYN (Position-Independent Executable file)
  Machine:                               Intel 80386
  Version:                               0x1
  Entry point address:                   0x3c60
  Start of program headers:              52 (bytes into file)
  Start of section headers:              156320 (bytes into file)
  Flags:                                  0x0
  Size of this header:                    52 (bytes)
  Size of program headers:                32 (bytes)
  Number of program headers:              12
  Size of section headers:                40 (bytes)
  Number of section headers:              28
  Section header string table index:      27
```

File System Hierarchy in Linux

As it turns out, there is a standard which is a reference that describes the conventions used by Unix/Linux systems for the organization and layout of their filesystem. This standard was created about 28 years ago (14 Feb 1994) and the last version (3.0) was published 7 years ago (3 Jun 2015). If you want to go over the specification for more details use the following link - <https://refspecs.linuxfoundation.org/fhs.shtml>.

We are going to give a short description for each directory (a detailed description for some of them will be in a dedicated write-up). We are going to list all the directories based on a lexicographic order. All the examples that I am going to share are based on a VM running Ubuntu 22.04.1 (below is a screenshot showing the directories for that VM). So let the fun begin ;-)

“/”, is the root directory of the entire system (the start of everything).

“/bin”, basic command mostly binaries (there are also scripts like `zgrep`) that are needed for every user. Examples are: `ls`, `ip` and `id`.

“/boot”, contains files needed for boot like the kernel (`vmlinuz`), `initrd` and boot loader configuration (like for `grub`). It may also contain metadata information about the kernel build process like the config that was used (A detailed writeup is going to be shared about “/boot” in the future).

“/dev”, device files, for now you should think about it as an interface to a device driver which is located on a filesystem (more on that in the future). Examples are: `/dev/null`, `/dev/zero` and `/dev/random`.

“/etc”, contains configuration about the system or an installed application. Examples are: `/etc/adduser.conf` (configuration file for `adduser` and `addgroup` commands) and `/etc/sudo.conf`.

“/home”, is the default location of the users’ home directory (it can be modified in `/etc/passwd` per user). The directory might contain personal settings of the user, saved files by the user and more. Example is `.bash_history` which is a hidden file that contains the historical commands entered by the user (while using the bash shell).

“/lib”, contains libraries needed by binaries mostly (but not limited to) in “/bin” and “/sbin”. On 64 bit systems we can also have “lib64”.

“/media”, used as a mount point for removable media (like CD-ROMs and USBs).

“/mnt”, can be used for temporary mounted filesystems.

“/opt”, should include applications installed by the user as add-ons (in reality not all of the addons are installed there).

“/lost+found”, this directory holds files that have been deleted or lost during a file operation. It means that we have an inode for those files, however we don’t have a filename on disk for them (think about cases of kernel panic or an unplanned shutdown). It is handled by tools like `fsck` - more on that is a future writeup.

“/proc”, it is a pseudo filesystem which enables retrieval of information about kernel data structures from user space using file operations, for example “ps” reads information for there to build the process list. Due to the fact it is a crucial part of Linux I am going to dedicate an entire writeup about it.

“/root”, it's the default home directory of the root account.

“/run”, it is used for runtime data like: running daemons, logged users and more. It should be erased/initialized every time on reboot/boot.

“/sbin”, similar to “/bin” but contains system binaries like: lsmod, init (in Ubuntu by the way it is a link to systemd) and dhclient.

“/srv”, contains information which is published by the system to the outside world using FTP/web server/other.

“/sys”, also a pseudo filesystem (similar to /proc) which exports information about hardware devices, device drivers and kernel subsystems. It can also allow configuration of different subsystems (like tracing for ftrace). I will cover it separately in more detail in the near future.

“/tmp”, the goal of the directory is to contain temporary files. Most of the time the content is not saved between reboots. Remember that there is also “/var/tmp”.

“/usr”, it is referred to by multiple names “User Programs” or “User System Resources”. It has several subdirectories containing binaries, libs, doc files and also can contain source code. Historically, it was meant to be read-only and shared between FHS-compliant hosts²¹. Due to the nature of its complexity today and the large amount of files it contains we will go over it also in a different writeup.

“/var”, aka variable files. It contains files which by design are going to change during the normal operation of the system (think about spool files, logs and more). More on this directory in the future.

It is important to note that those are not all the directories and subdirectories included on a clean Linux installation, but the major ones I have decided to start with

```
lrwxr-xr-x 3 root root 4096 Sep 16 22:02 boot
lrwxr-xr-x 19 root root 4120 Sep 18 05:47 dev
lrwxr-xr-x 135 root root 12288 Sep 16 22:02 etc
lrwxr-xr-x 3 root root 4096 Jul 29 07:13 home
lrwxrwxrwx 1 root root 7 Apr 19 06:02 lib -> usr/lib
lrwxrwxrwx 1 root root 9 Apr 19 06:02 lib32 -> usr/lib32
lrwxrwxrwx 1 root root 9 Apr 19 06:02 lib64 -> usr/lib64
lrwxrwxrwx 1 root root 10 Apr 19 06:02 libx32 -> usr/libx32
lrwx----- 2 root root 16384 Jul 29 07:10 lost+found
lrwxr-xr-x 2 root root 4096 Apr 19 06:02 media
lrwxr-xr-x 2 root root 4096 Apr 19 06:02 mnt
lrwxr-xr-x 2 root root 4096 Apr 19 06:02 opt
lr-xr-xr-x 273 root root 0 Sep 9 12:31 proc
lrwx----- 6 root root 4096 Sep 12 18:40 root
lrwxr-xr-x 29 root root 840 Sep 18 00:54 run
lrwxr-xr-x 9 root root 4096 Apr 19 06:08 snap
lrwxr-xr-x 2 root root 4096 Apr 19 06:02 srv
lr-xr-xr-x 13 root root 0 Sep 9 12:31 sys
lrwxrwxrwt 19 root root 4096 Sep 18 07:20 tmp
lrwxr-xr-x 14 root root 4096 Apr 19 06:02 usr
lrwxr-xr-x 14 root root 4096 Apr 19 06:07 var
```

²¹ <https://tldp.org/LDP/Linux-Filesystem-Hierarchy/html/usr.html>

/boot/config-\$(uname-r)

“/boot/config-\$(uname-r)” is a text file that contains a configuration (feature/options) that the kernel was compiled with. The “uname -r” is replaced by the kernel release²². It is important to understand that the file is only needed for the compilation phase and not for loading the kernel, so it can be removed or even altered by a root user and therefore not reflect the specific configuration that was used. Overall, any time one of the following “make menuconfig”/“make xconfig”, “make localconfig”, “make oldconfig”, “make XXX_defconfig” or other “make XXXconfig” creates a “.config” file. This file is not erased (unless using “make mrproper”). Also, many distributions are copying that file to “/boot”²³.

The build system will read the configuration file and use it to generate the kernel by compiling the relevant source code. By using the configuration file we can customize the Linux kernel to your needs²⁴. The configuration file is based on key values - as shown in the screenshot below²⁵. Using the configuration we can enable/disable features like sound/networking/USB support as we can see with the “CONFIG_MMU=y” in the screenshot below²⁶. Also, we can adjust a specific value of features like the “CONFIG_ARCH_MMAP_RND_BITS_MIN=28”²⁷.

Moreover, in case of kernel modules we can add/remove modules and decide if we want to compile them into the kernel itself or as a separate “.ko” file. In case the setting is “y” it means to compile inside the kernel, “m” means as a separate file and “n” means not to compile²⁸. Thus, if “CONFIG_DRM_TTM=m” then the “TTM memory manager subsystem” is going to be compiled outside of the kernel²⁹. If “ttm” is loaded it will be shown in the output of “lsmod”³⁰.

```
1 #
2 # Automatically generated file; DO NOT EDIT.
3 # Linux/x86 4.15.0-117-generic Kernel Configuration
4 #
5 CONFIG_64BIT=y
6 CONFIG_X86_64=y
7 CONFIG_X86=y
8 CONFIG_INSTRUCTION_DECODER=y
9 CONFIG_OUTPUT_FORMAT="elf64-x86-64"
10 CONFIG_ARCH_DEFCONFIG="arch/x86/configs/x86_64_defconfig"
11 CONFIG_LOCKDEP_SUPPORT=y
12 CONFIG_STACKTRACE_SUPPORT=y
13 CONFIG_MMU=y
14 CONFIG_ARCH_MMAP_RND_BITS_MIN=28
15 CONFIG_ARCH_MMAP_RND_BITS_MAX=32
16 CONFIG_ARCH_MMAP_RND_COMPAT_BITS_MIN=8
17 CONFIG_ARCH_MMAP_RND_COMPAT_BITS_MAX=16
18 CONFIG_NEED_DMA_MAP_STATE=y
19 CONFIG_NEED_SG_DMA_LENGTH=y
20 CONFIG_GENERIC_ISA_DMA=y
21 CONFIG_GENERIC_BUG=y
22 CONFIG_GENERIC_BUG_RELATIVE_POINTERS=y
23 CONFIG_GENERIC_HWEIGHT=y
24 CONFIG_ARCH_MAY_HAVE_PC_FDC=y
25 CONFIG_RWSEM_XCHGADD_ALGORITHM=y
26 CONFIG_GENERIC_CALIBRATE_DELAY=y
27 CONFIG_ARCH_HAS_CPU_RELAX=y
28 CONFIG_ARCH_HAS_CACHE_LINE_SIZE=y
29 CONFIG_ARCH_HAS_FILTER_PGPROT=y
30 CONFIG_HAVE_SETUP_PER_CPU_AREA=y
31 CONFIG_NEED_PER_CPU_SMP_FIRST_CHUNK=y
/boot/config-4.15.0-117-generic 9621 /j --8%-
```

²² <https://linux.die.net/man/1/uname>

²³ <https://unix.stackexchange.com/questions/123026/where-kernel-configuration-file-is-stored>

²⁴ <https://linuxconfig.org/in-depth-howto-on-linux-kernel-configuration>

²⁵ https://blog.csdn.net/weixin_43644245/article/details/121578858

²⁶ <https://elixir.bootlin.com/linux/v6.4.11/source/arch/um/Kconfig#L36>

²⁷ <https://elixir.bootlin.com/linux/v6.4.11/source/arch/x86/Kconfig#L322>

²⁸ <https://stackoverflow.com/questions/14587251/understanding-boot-config-file>

²⁹ https://github.com/torvalds/linux/blob/master/drivers/gpu/drm/ttm/ttm_module.c

³⁰ <https://man7.org/linux/man-pages/man8/lsmod.8.html>

/proc/config.gz

Since kernel version 2.6 the configuration options that were used to build the current running kernel are exposed using procfs in the following path “/proc/config.gz”. The format of the content is the same as the .config file which is copied by different distribution to “/boot”³¹.

Overall, as opposed to the “.config” file the data of “config.gz” is compressed. Due to that, if we want to view its content we can use `zcat`³² or `zgrep`³³ which allow reading/searching inside compressed files. As shown in the screenshot below (taken from `copy.sh`).

Lastly, in order for “config.gz” to be supported and exported by “/proc” the kernel needs to be built with “CONFIG_IKCONFIG_PROC” enabled³⁴ - as also shown in the screenshot below. We can also go over the creation of the “/proc” entry³⁵ and the function that returns the data when reading that entry³⁶.

```
root@localhost:~# file /proc/config.gz
/proc/config.gz: gzip compressed data, max compression, from Unix, original size modulo 2^32 265269
root@localhost:~# zcat /proc/config.gz | head -25
#
# Automatically generated file; DO NOT EDIT.
# Linux/x86_64 5.19.7-arch1 Kernel Configuration
#
CONFIG_CC_VERSION_TEXT="gcc (GCC) 12.2.0"
CONFIG_CC_IS_GCC=y
CONFIG_GCC_VERSION=120200
CONFIG_CLANG_VERSION=0
CONFIG_AS_IS_GNU=y
CONFIG_AS_VERSION=23900
CONFIG_LD_IS_BFD=y
CONFIG_LD_VERSION=23900
CONFIG_LLD_VERSION=0
CONFIG_CC_CAN_LINK=y
CONFIG_CC_CAN_LINK_STATIC=y
CONFIG_CC_HAS_ASM_GOTO=y
CONFIG_CC_HAS_ASM_GOTO_OUTPUT=y
CONFIG_CC_HAS_ASM_GOTO_TIED_OUTPUT=y
CONFIG_CC_HAS_ASM_INLINE=y
CONFIG_CC_HAS_NO_PROFILE_FN_ATTR=y
CONFIG_PAHOLE_VERSION=123
CONFIG_IRQ_WORK=y
CONFIG_BUILDTIME_TABLE_SORT=y
CONFIG_THREAD_INFO_IN_TASK=y

root@localhost:~# zcat /proc/config.gz | grep IKCONFIG_PROC
CONFIG_IKCONFIG_PROC=y
```

³¹ <https://medium.com/@boutnaru/the-linux-concept-journey-boot-config-uname-r-6a4dd16048c4>

³² <https://linux.die.net/man/1/zcat>

³³ <https://linux.die.net/man/1/zgrep>

³⁴ <https://elixir.bootlin.com/linux/v6.5/source/kernel/configs.c#L35>

³⁵ <https://elixir.bootlin.com/linux/v6.5/source/kernel/configs.c#L60>

³⁶ <https://elixir.bootlin.com/linux/v6.5/source/kernel/configs.c#L41>

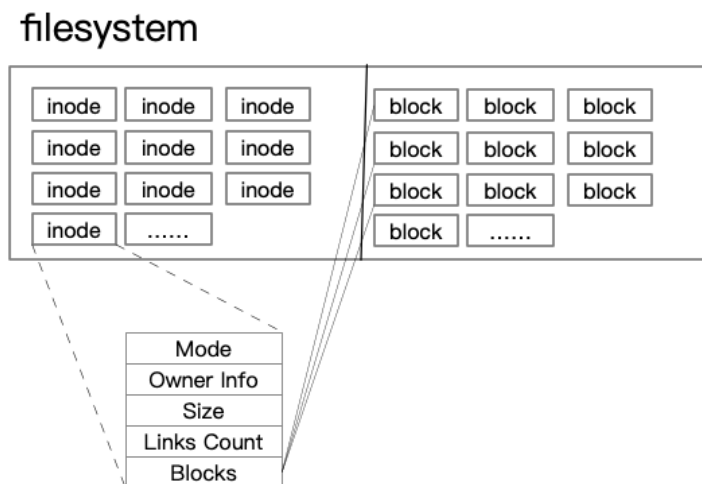
What is an inode?

An inode (aka index node) is a data structure used by Unix/Linux like filesystems in order to describe a filesystem object. Such an object could be a file or a directory. Every inode stores pointers to the disk blocks locations of the object's data and metadata³⁷. An illustration of that is shown below³⁸.

Overall, the metadata contained in an inode is: file type (regular file/directory/symbolic link/block special file/character special file/etc), permissions, owner id, group id, size, last accessed time, last modified time, change time and number of hard links³⁹.

By using inodes the filesystem tracks all files/directories saved on disk. Also, by using inodes we can read any specific byte in the data of a file very effectively. We can see the number of total inodes per mounted filesystem using the command “df -i”⁴⁰. Also, we can see the inode of a file/directory and other metadata of the file using the command “ls -li”⁴¹ or “stat”⁴². By the way, the “stat” command can use different syscalls (depending on the filesystem and the specific version) like “stat”⁴³, “lstat”⁴⁴ or “statx”⁴⁵.

Lastly, you can check out “struct inode” in the source code of the Linux kernel⁴⁶. Not all the points/links are directly connected to the data blocks, however I will elaborate on that in a future writeup.



³⁷ <https://www.bluematador.com/blog/what-is-an-inode-and-what-are-they-used-for>

³⁸ <https://www.sobyte.net/post/2022-05/linux-inode/>

³⁹ <https://www.stackscale.com/blog/inodes-linux/>

⁴⁰ <https://linux.die.net/man/1/df>

⁴¹ <https://man7.org/linux/man-pages/man1/ls.1.html>

⁴² <https://linux.die.net/man/1/stat>

⁴³ <https://linux.die.net/man/2/stat>

⁴⁴ <https://linux.die.net/man/2/lstat>

⁴⁵ <https://man7.org/linux/man-pages/man2/statx.2.html>

⁴⁶ <https://elixir.bootlin.com/linux/v6.4.2/source/include/linux/fs.h#L612>

Why is removing a file not dependent on the file's permissions?

Something which is not always understood correctly by Linux users is the fact that removing a file is not dependent on the permissions of the file itself. As you can see in the screenshot below even if a user has full permission (read+write+execute) it can't remove a file. By the way, removing a file is done by using the "unlink" syscall⁴⁷ or the "unlinkat" syscall⁴⁸.

The reason for that is because the data that states a file belongs to a directory is saved as part of the directory itself. We can think about a directory as a "special file" whose data is the name and the inode⁴⁹ numbers of the files that are part of that specific directory.

Thus, if we add write permissions to the directory even if the user has no permissions to the file ("chmod 000") the file can be removed (from the directory) - as shown in the screenshot below.

```
[troller@localhost test]$ ls -lah ./troller
-rwxrwxrwx 1 root root 8 Jul 11 2023 ./troller
[troller@localhost test]$ cat ./troller
Troller
[troller@localhost test]$ rm ./troller
rm: cannot remove './troller': Permission denied
[troller@localhost test]$ exit
exit
root@localhost:/tmp/test# chmod o+w /tmp/test
root@localhost:/tmp/test# chmod 000 ./troller
root@localhost:/tmp/test# su troller
[troller@localhost test]$ ls -lah ./troller
----- 1 root root 8 Jul 11 2023 ./troller
[troller@localhost test]$ cat ./troller
cat: ./troller: Permission denied
[troller@localhost test]$ rm ./troller
rm: remove write-protected regular file './troller'? y
[troller@localhost test]$ ls -lah ./troller
ls: cannot access './troller': No such file or directory
```

⁴⁷ <https://linux.die.net/man/2/unlink>

⁴⁸ <https://linux.die.net/man/2/unlinkat>

⁴⁹ <https://medium.com/@boutnaru/linux-what-is-an-inode-7ba47a519940>

VFS (Virtual File System)

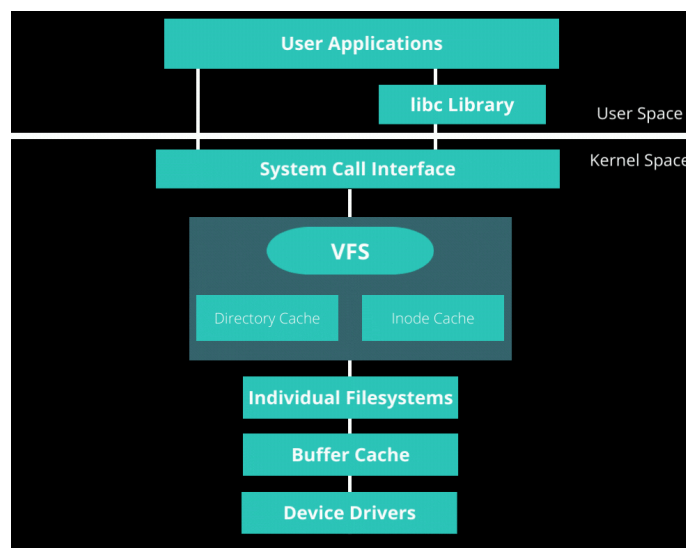
VFS (Virtual File System, aka Virtual File Switch) is a software component of Linux which is responsible for the filesystem interface between the user-mode and kernel mode. Using it allows the kernel to provide an abstraction layer that makes implementation of different filesystems very easy⁵⁰.

Overall, VFS is masking the implementation details of a specific filesystem behind generic system calls (open/read/write/close/etc), which are mostly exposed to user-mode application by some wrappers in libc - as shown in the diagram below⁵¹.

Moreover, we can say that the main goal of VFS is to allow user-mode applications to access different filesystems (think about NTFS, FAT, etc.) in the same way. There are four main objects in VFS: superblock, dentries, inodes and files⁵².

Thus, “inode”⁵³ is what the kernel uses to keep track of files. Because a file can have several names there are “dentries” (“directory entries”) which represent pathnames. Also, due to the fact a couple of processes can have the same file opened (for read/write) there is a “file” structure that holds the information for each one (such as the cursor position). The “superblock” structure holds data which is needed for performing actions on the filesystem - more details about all of those and more (like mounting) are going to be published in the near future.

Lastly, there are also other relevant data structures that I will post on in the near future (“filesystem”, “vfsmount”, “nameidata” and “address_space”).



⁵⁰ <https://www.kernel.org/doc/html/next/filesystems/vfs.html>

⁵¹ <https://www.starlab.io/blog/introduction-to-the-linux-virtual-filesystem-vfs-part-i-a-high-level-tour>

⁵² <https://www.win.tue.nl/~aeb/linux/lk/lk-8.html>

⁵³ <https://medium.com/@boutnaru/linux-what-is-an-inode-7ba47a519940>

tmpfs (Temporary Filesystem)

“tmpfs” is a filesystem that saves all of its files in virtual memory. By using it none of the files created on it are saved to the system’s hard drive. Thus, if we unmount a tmpfs mounting point every file which is stored there is lost. tmpfs holds all of the data into the kernel internal caches⁵⁴. By the way, it used to be called “shm fs”⁵⁵.

Moreover, tmpfs is able to swap space if needed (it can also leverage “Transparent Huge Pages”), it will fill up until it reaches the maximum limit of the filesystem - as shown in the screenshot below. tmpfs supports both POSIX ACLs and extended attributes⁵⁶. Overall, if we want to use tmpfs we can use the following command: “mount -t tmpfs tmpfs [LOCATION]”. We can also set a size using “-o size=[REQUESTED_SIZE]” - as shown in the screenshot below.

Lastly, there are different directories which are based on “tmpfs” like: “/run” and “/dev/shm” (more on them in future writeups). To add support for “tmpfs” we should enable “CONFIG_TMPFS” when building the Linux kernel⁵⁷. We can see the implementation as part of the Linux’s kernel source code⁵⁸.

```
root@localhost:/tmp# mount | grep troller
root@localhost:/tmp# mount -t tmpfs tmpfs -o size=1M /tmp/troller
root@localhost:/tmp# mount | grep troller
tmpfs on /tmp/troller type tmpfs (rw,relatime,size=1024k)
root@localhost:/tmp# dd if=/dev/zero of=/tmp/troller/tmp bs=512 count=999999999
dd: error writing '/tmp/troller/tmp': No space left on device
2049+0 records in
2048+0 records out
1048576 bytes (1.0 MB, 1.0 MiB) copied, 0.0801 s, 13.1 MB/s
root@localhost:/tmp# echo test > /tmp/troller/test
-bash: echo: write error: No space left on device
```

⁵⁴ <https://www.kernel.org/doc/html/latest/filesystems/tmpfs.html>

⁵⁵ <https://cateee.net/lkddb/web-lkddb/TMPFS.html>

⁵⁶ <https://man7.org/linux/man-pages/man5/tmpfs.5.html>

⁵⁷ <https://cateee.net/lkddb/web-lkddb/TMPFS.html>

⁵⁸ <https://elixir.bootlin.com/linux/v6.6-rc1/source/mm/shmem.c#L133>

ramfs (Random Access Memory Filesystem)

“ramfs” is a filesystem that exports the Linux caching mechanism (page cache/dentry cache) as a dynamically resizable RAM based filesystem. The data is saved in RAM only and there is no backing store for it⁵⁹.

Thus, if we unmount a “ramfs” mounting point every file which is stored there is lost - as shown in the screenshot below. By the way, the trick is that files written to “ramfs” allocate dentries and page cache as usual, but because they are not written they are never marked as being available for freeing⁶⁰.

Moreover, with “ramfs” we can keep on writing until we fill-up the entire physical memory. Due to that, it is recommended that only root users will be able to write to a mounting point which is based on “ramfs”. The differences between “ramfs” and “tmpfs”⁶¹ is that “tmpfs” is limited in size and can also be swapped⁶².

Lastly, we can go over the implementation of “ramfs” as part of Linux's kernel source code⁶³. There are two implementations, one in case of an MMU⁶⁴ and one in case there is no MMU⁶⁵. A good example for using “ramfs” is “initramfs”.

```
root@localhost:/tmp# mount | grep troller
root@localhost:/tmp# mount -t ramfs ramfs /tmp/troller
root@localhost:/tmp# mount | grep troller
ramfs on /tmp/troller type ramfs (rw,relatime)
root@localhost:/tmp# df -h /tmp/troller
Filesystem      Size  Used Avail Use% Mounted on
ramfs            0      0      0  -  /tmp/troller
root@localhost:/tmp# free -h
              total        used        free      shared  buff/cache   available
Mem:           479Mi        15Mi        447Mi          0Ki        16Mi        451Mi
Swap:            0B           0B           0B
root@localhost:/tmp# dd if=/dev/random of=/tmp/troller/file bs=512 count=99999
99999+0 records in
99999+0 records out
51199488 bytes (51 MB, 49 MiB) copied, 11.0403 s, 4.6 MB/s
root@localhost:/tmp# ls -lah /tmp/troller/file
-rw-r--r-- 1 root root 49M Nov  7 05:25 /tmp/troller/file
root@localhost:/tmp# free -h
              total        used        free      shared  buff/cache   available
Mem:           479Mi        15Mi        398Mi          0Ki         65Mi        402Mi
Swap:            0B           0B           0B
root@localhost:/tmp# umount /tmp/troller/
root@localhost:/tmp# free -h
              total        used        free      shared  buff/cache   available
Mem:           479Mi        15Mi        447Mi          0Ki        16Mi        451Mi
Swap:            0B           0B           0B
```

⁵⁹ <https://docs.kernel.org/filesystems/ramfs-rootfs-initramfs.html>

⁶⁰ <https://lwn.net/Articles/157676/>

⁶¹ <https://medium.com/@boutnaru/the-linux-concept-journey-tmpfs-temporary-filesystem-886b61a545a0>

⁶² <https://wiki.debian.org/ramfs>

⁶³ <https://elixir.bootlin.com/linux/v6.5.3/source/fs/ramfs>

⁶⁴ <https://elixir.bootlin.com/linux/v6.5.3/source/fs/ramfs/file-mmu.c>

⁶⁵ <https://elixir.bootlin.com/linux/v6.5.3/source/fs/ramfs/file-nommu.c>

⁶⁸ <https://www.expertsmind.com/questions/describe-the-buddy-system-of-memory-allocation-3019462.aspx>

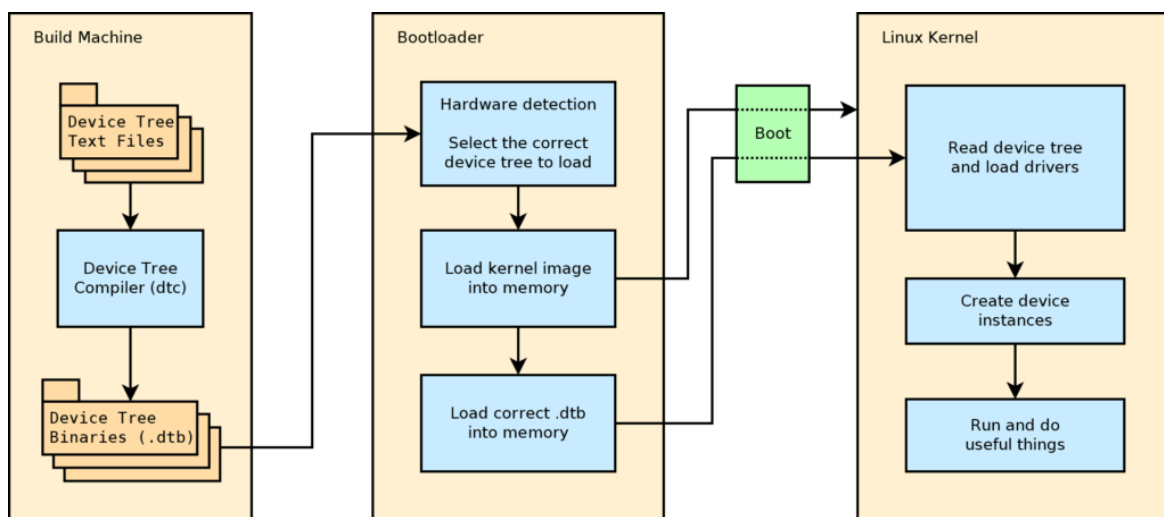
DeviceTree

Overall, a “Device Tree” is a mechanism for describing non-discoverable hardware. In the past this type of information was hard-coded as part of the source code (and later part of the binary/firmware). So we can say a “DeviceTree” is a data structure for describing hardware. By the way, there is a specification for that called “The Devicetree Specification”⁶⁹.

Moreover, “DeviceTree” allows the kernel to manage different components such as CPU, memory, buses and other. The specification detailed above contains the data format used, which is internally a tree of named nodes and properties (key-value based). Device trees can be in binary format (“*.dtb”) or text based (“*.dts”) for easing the management and editing⁷⁰. The “*.dtb” files are loaded by the bootloader and passed to the kernel - as shown in the diagram below⁷¹.

Thus, we can find in the Linux kernel source code “*.dts” files in the directories of some of the architectures, with the following pattern “/arch/[ARCHITECTURE]/boot/dts”. Two examples are “arm64”⁷² and “nios2”⁷³. In order to transform “*.dts” files to “*.tdb” we can use the “dtc” which is the “Device Tree Compiler”⁷⁴. We can also go over the Makefiles responsible for that if we want⁷⁵.

Lastly, there are also “*.dtsi” files that include files (like we have in c/c++). We can use also YAML as the format of “*.dts” files in order to describe the different hardware components⁷⁶.



⁶⁹ <https://www.devicetree.org/specifications/>

⁷⁰ <https://en.wikipedia.org/wiki/Devicetree>

⁷¹ <https://vocal.com/resources/development/what-is-linux-device-tree/>

⁷² <https://elixir.bootlin.com/linux/v6.5.4/source/arch/arm64/boot/dts>

⁷³ <https://elixir.bootlin.com/linux/v6.5.4/source/arch/nios2/boot/dts>

⁷⁴ <https://manpages.ubuntu.com/manpages/xenial/man1/dtc.1.html>

⁷⁵ <https://elixir.bootlin.com/linux/v6.5.4/source/arch/arm64/boot/dts/Makefile>

⁷⁶ <https://www.konsulko.com/yaml-and-device-tree>

How can we recover a deleted executable of a running application?

In contrast to what you may think in case an executable file is deleted there are cases in which we can recover it very easily. One of them is in case there is at least one instance of an application running which is based on that executable.

Moreover, if the file is deleted we will see an indication for that in “/proc/[PID]/maps”, the string pattern “(deleted)” is added in that case - as shown in the screenshot below. Thus, we can recover the file by copying it from the location “/proc/[PID]/exe” - as also shown in the screenshot below.

Lastly, this can be used for forensics/security purposes or just in case we deleted the executable by mistake (just remember not to stop/kill the application). By the way, we can use a similar trick for other files which are not the main image/executable (but that is for future writeups).

```
Troller $ cat ./troller.c
#include <stdio.h>
#include <unistd.h>

void main()
{
    printf("Troller...\n");
    sleep(2222);
}

Troller $ gcc troller.c -o troller
Troller $ sha256sum ./troller
780c5ac33b7f2a803ea0a1592ec4cfad5c943581bf0a6d5cae04ce38fc55ada3  ./troller
Troller $ ./troller &
[1] 19628
Troller $ Troller...

Troller $ cat /proc/19628/maps | head -2
557ce0ac2000-557ce0ac3000 r--p 00000000 fd:00 672196 /tmp/troller/troller
557ce0ac3000-557ce0ac4000 r-xp 00001000 fd:00 672196 /tmp/troller/troller
Troller $ rm ./troller
Troller $ cat /proc/19628/maps | head -2
557ce0ac2000-557ce0ac3000 r--p 00000000 fd:00 672196 /tmp/troller/troller (deleted)
557ce0ac3000-557ce0ac4000 r-xp 00001000 fd:00 672196 /tmp/troller/troller (deleted)
Troller $ cp /proc/19628/exe ./troller_from_proc
Troller $ sha256sum ./troller_from_proc
780c5ac33b7f2a803ea0a1592ec4cfad5c943581bf0a6d5cae04ce38fc55ada3  ./troller_from_proc
```

Process Group

Overall, a “Process Group” is a collection of processes which can be managed/handled together by the operating system (one example of that is signal management). Each “Process Group” has an identifier (PGID) and a “leader” which is the process that has created it. The PGID is equal to the PID of the group leader⁷⁷.

Moreover, we can use `getpgid/getpgrp` to get PGID or `setpgid/setpgrp` to set it⁷⁸. The `setpgrp/getpgrp` is a System-V API which `setpgid/getpgid` is the POSIX API⁷⁹. The POSIX one is the preferred one. Thus, if we check `grep.app` we can see that “`setpgid`”⁸⁰ has more than 29 times more results than “`setpgrp`”⁸¹.

Lastly, any time we execute a command in a shell or a pipeline of commands we create a “Process Group” (it is sometimes called a job in the shell). As we can see in the screenshot below the number of the “Process Group” changes for each execution, and for the last one it gets incremented by one only even though we execute 5 processes (cause it is the same pipeline).

```
root@localhost:~# cat print_pgid.py
import ctypes
libc=ctypes.CDLL(None)
print(libc.getpgid(libc.getpid())) #we can also pass 0 to getpgid(), instead of using getpid()

root@localhost:~# python ./print_pgid.py
1281
root@localhost:~# python ./print_pgid.py
1282
root@localhost:~# ps | ps | ps | ps | python ./print_pgid.py
1283
```

⁷⁷ <https://biriukov.dev/docs/fd-pipe-session-terminal/3-process-groups-jobs-and-sessions/>

⁷⁸ <https://man7.org/linux/man-pages/man2/setpgid.2.html>

⁷⁹ <https://unix.stackexchange.com/questions/404054/how-is-a-process-group-id-set>

⁸⁰ <https://grep.app/search?q=%20setpgid%28>

⁸¹ <https://grep.app/search?q=%20setpgrp%28>

<Major:Minor> Numbers

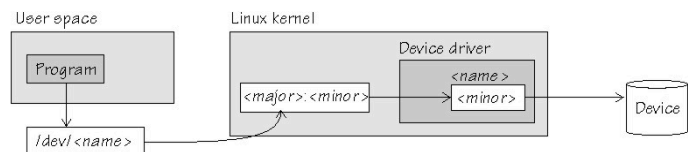
In the Linux kernel devices (character/block device) are represented as a pair of numbers (<major>:<minor>). There are some major/minor numbers which are reserved while others are assigned dynamically. A major number can be shared between multiple device drivers⁸².

Overall, device files are located in “/dev” and they allow accessing the device from user-mode. Those files are “connected” to the device using the major and minor number - as shown in the diagram below on the left side⁸³. We can see them using “ls -l” - as shown in the shell output in the screenshot below (taken from copy.sh). The major number identifies the drivers associated with the device, while the minor number is used only by the driver which gets the number without the kernel using it⁸⁴.

Moreover, the kernel code that can assign the name and the major number for a specific device char device can use the “register_chrdev” function⁸⁵. In case of a block device can use the macro “register_blkdev”⁸⁶.

Lastly, we can use the “mknod”⁸⁷ command to create a block/char device file by providing a name, major and minor numbers. We can also list the major numbers of the currently registered devices with their names⁸⁸ - as shown in the output of the screenshot below.

```
root@localhost:~# ls -la /dev/null /dev/random /dev/urandom /dev/zero
crw-rw-rw- 1 root root 1, 3 Nov  7 02:50 /dev/null
crw-rw-rw- 1 root root 1, 8 Nov  7 02:50 /dev/random
crw-rw-rw- 1 root root 1, 9 Nov  7 02:50 /dev/urandom
crw-rw-rw- 1 root root 1, 5 Nov  7 02:50 /dev/zero
root@localhost:~# cat /proc/devices | head -10
Character devices:
1 mem
4 /dev/vc/0
4 tty
4 ttyS
5 /dev/tty
5 /dev/console
5 /dev/ptmx
7 ucs
10 misc
```



⁸² <https://www.ibm.com/docs/en/linux-on-systems?topic=hdaa-device-nodes-numbers>

⁸³ <https://www.ibm.com/docs/en/linuxonibm/com.ibm.linux.z.ufdd/lxnode.jpg>

⁸⁴ <https://www.oreilly.com/library/view/linux-device-drivers/0596000081/ch03s02.html>

⁸⁵ <https://elixir.bootlin.com/linux/v6.6.1/source/include/linux/fs.h#L2535>

⁸⁶ <https://elixir.bootlin.com/linux/v6.6.1/source/include/linux/blkdev.h#L809>

⁸⁷ <https://man7.org/linux/man-pages/man1/mknod.1.html>

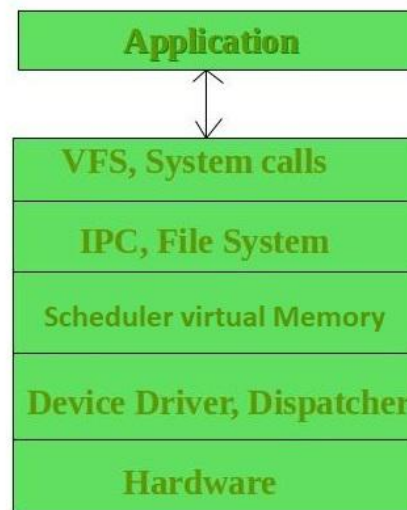
⁸⁸ <https://man7.org/linux/man-pages/man5/procfs.5.html>

Monolithic Kernel

Monolithic kernel is an operating system architecture in which the entire OS code is executed in kernel mode. Examples of operating systems which use this design are: Linux, BSD, SunOS, AIX, MS-DOS, OpenVMS, HP-UX, TempleOS, z/TPF, XTS-400, Solaris, FreeBSD and MULTICS⁸⁹.

Thus, the kernel provides CPU scheduling, file management, memory management, IPC and more (as opposed to microkernel) - as shown in the diagram below. There are a couple of advantages when using this design like: we can use a single static binary for the kernel, we can access all the capabilities of the kernel using system calls and we can get a single execution flow in a single address space⁹⁰.

Moreover, probably the biggest disadvantage of a monolithic kernel design is that in case of a service failure the entire system fails. A monolithic kernel can be extended (without recompilation and linking) only if it is modular. Linux it is done using loadable kernel modules, the support for that is defined by enabling “CONFIG_MODULES” which compiling the kernel⁹¹. Lastly, in kernel version 6.7.4 “CONFIG_MODULES” is referenced in 158 files across the kernel source code files⁹².



Monolithic Kernel

⁸⁹ https://en.wikipedia.org/wiki/Monolithic_kernel

⁹⁰ <https://www.geeksforgeeks.org/monolithic-kernel-and-key-differences-from-microkernel/>

⁹¹ <https://elixir.bootlin.com/linux/latest/source/Makefile#L1106>

⁹² https://elixir.bootlin.com/linux/latest/A/ident/CONFIG_MODULES

Loadable Kernel Module (LKM)

A loadable kernel module (LKM) allows us to add code to the Linux kernel without the need of recompiling and linking the kernel binary. This is used for different use cases such as (but not limited to) filesystem drivers and device drivers⁹³.

Overall, when compiling the Linux kernel we can decide if we want to incorporate a specific kernel module as part of the kernel itself (also called built-in kernel modules - more details on them in future writeups) or as a separate “*.ko” (kernel object) file. In case the kernel is already compiled we only have the option of creating a kernel object file which can be later loaded by using the “insmod” utility⁹⁴.

Moreover, a kernel object file has the same type as an ordinary object file (before it is linked and can be executed in user mode) - as shown in the screenshot below. This type is called relocatable and is defined as “ET_REL”⁹⁵.

Lastly, we can dynamically check the list of the actively loaded kernel modules using the “lsmod”⁹⁶ utility - as shown in the screenshot below. “lsmod” basically parses “/proc/modules”⁹⁷.

```
Troller$ cat troller.c
#include <stdio.h>

void main()
{
    printf("Tr0LeR\n");
}
Troller$ gcc -c troller.c -o troller
Troller$ file ./troller
./troller: ELF 64-bit LSB relocatable, x86-64, version 1 (SYSV), not stripped
Troller$ readelf -h ./troller | head -8
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                           ELF64
  Data:                               2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                        0
  Type:                               REL (Relocatable file)
Troller$ lsmod | head -2
Module                  Size      Used by
tls                     0
Troller$ modinfo tls | head -1
filename:               /lib/modules/...-generic/kernel/net/tls/tls.ko
Troller$ file /lib/modules/...-generic/kernel/net/tls/tls.ko
/lib/modules/...-generic/kernel/net/tls/tls.ko: ELF 64-bit LSB relocatable, x86-64, version 1 (SYSV), BuildID[sha1]=..., not stripped
Troller$ readelf -h /lib/modules/...-generic/kernel/net/tls/tls.ko | head -8
readelf: Warning: ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                           ELF64
  Data:                               2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                        0
  Type:                               REL (Relocatable file)
```

⁹³ <https://tldp.org/HOWTO/Module-HOWTO/x73.html>

⁹⁴ <https://man7.org/linux/man-pages/man8/insmod.8.html>

⁹⁵ <https://man7.org/linux/man-pages/man5/elf.5.html>

⁹⁶ <https://man7.org/linux/man-pages/man8/lsmod.8.html>

⁹⁷ <https://man7.org/linux/man-pages/man5/proc.5.html>

Builtin Kernel Modules

In general, we can compile a kernel module⁹⁸ as a separate “*.ko” file or include it as part of the kernel itself. If the kernel module is included as part of the kernel’s image it is referred to as a builtin kernel module.

Overall, we can check this configuration regarding our compiled kernel by leveraging “/proc/config.gz”⁹⁹ or “/boot/config-\$(uname-r)”¹⁰⁰ in case they exist. If there are built-in kernel modules compiled into the kernel, it is not sufficient to use “lsmod” for identifying them and we need to use other techniques like searching for specific symbols in “/proc/kallsyms”¹⁰¹ or using “modinfo”¹⁰² - as shown in the screenshot below.

Lastly, “built-in kernel modules” can’t be unloaded as opposed to “loadable kernel modules”. However, we can be sure that they are loaded and we don’t need to take care of additional files which are needed as in the case of “loadable kernel modules”¹⁰³. Due to that (and more), “built-in kernel modules” are great to ensure essential functionality that must be available at all times¹⁰⁴.

```
Troller $ mount | grep "/"
/dev/mapper/ on / type ext4 (rw,relatime)
Troller $ lsmod | grep ext4
Troller $ cat /proc/kallsyms | grep ext4 | wc -l
2626
Troller $ cat /proc/kallsyms | grep ext4 | head -2
0000000000000000 t __uncore_umask_ext4_show
0000000000000000 t ext4_has_free_clusters
Troller $ modinfo ext4
name: ext4
filename: (builtin)
softdep: pre: crc32c
license: GPL
file: fs/ext4/ext4
description: Fourth Extended Filesystem
author: Remy Card, Stephen Tweedie, Andrew Morton, Andreas Dilger, Theodore Ts'o and others
alias: fs-ext4
alias: ext3
alias: fs-ext3
alias: ext2
alias: fs-ext2
Troller $
```

⁹⁸ <https://medium.com/@boutnaru/the-linux-concept-journey-loadable-kernel-module-lkm-5eaa4db346a1>

⁹⁹ <https://medium.com/@boutnaru/the-linux-concept-journey-proc-config-gz-34c4086e0207>

¹⁰⁰ <https://medium.com/@boutnaru/the-linux-concept-journey-boot-config-uname-r-6a4dd16048c4>

¹⁰¹ <https://www.unix.com/man-page/redhat/8/kallsyms/>

¹⁰² <https://linux.die.net/man/8/modinfo>

¹⁰³ <https://stackoverflow.com/questions/22929065/difference-between-linux-loadable-and-built-in-modules>

¹⁰⁴ <https://shape.host/resources/kernel-modules-vs-built-in-making-the-right-choice>

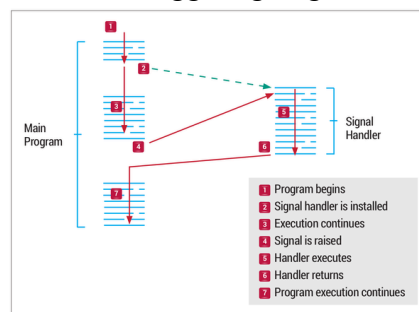
Signals

Signals are asynchronous events sent to a task (process), signals are numbered and their names in Linux start with “SIG” . A couple of examples are: “SIGINT” (generated when Control+C is pressed), “SIGALARM” (generated when the timer set by an alarm is fired), “SIGSTOP” (tells Linux to pause a process execution), “SIGCONT” (tells Linux to resume the execution of a process), “SIGSEGV” (generated in case of a segmentation fault) and “SIGKILL” (when sent to a process causes it to be terminated). When a signal occurs one of three things can happen: the signal is ignored, the signal is caught and handled using a registered function and letting the default action of the signal to happen¹⁰⁵.

Moreover, the default action could be: ignore, terminate, terminate and core dump, stop/pause the process or resume the stop/paused process (they are also called “Signal Dispositions”. Signals can be used as an IPC (Inter Process Communication) mechanism. In case we want to alter the disposition of a signal we can use the “signal”¹⁰⁶ syscall or the “sigaction”¹⁰⁷ syscall - the first is the preferred way.

Overall, for sending a signal by using its PID we can use the “kill”¹⁰⁸ system call. We can also use the “pidfd_send_signal” syscall for sending a signal by using a PID file descriptor¹⁰⁹. In order to send a signal to a specific thread of a process we can use the “tgkill”¹¹⁰ syscall. For more syscalls/library calls related to signals I suggest going over the signal man page¹¹¹.

Lastly, when a signal is received the relevant signal handler is called - as shown below¹¹². The number representing a signal ranges from 1-31, there are also “real time signals” which range from 34-64¹¹³. For further information I suggest going over the relevant source code as part of the Linux kernel¹¹⁴.



¹⁰⁵ <https://faculty.cs.niu.edu/~hutchins/csci480/signals.htm>

¹⁰⁶ <https://man7.org/linux/man-pages/man2/signal.2.html>

¹⁰⁷ <https://man7.org/linux/man-pages/man2/sigaction.2.html>

¹⁰⁸ <https://man7.org/linux/man-pages/man2/kill.2.html>

¹⁰⁹ https://man7.org/linux/man-pages/man2/pidfd_send_signal.2.html

¹¹⁰ <https://man7.org/linux/man-pages/man2/tgkill.2.html>

¹¹¹ <https://man7.org/linux/man-pages/man7/signal.7.html>

¹¹² <https://devopedia.org/linux-signals>

¹¹³ https://www-uxsup.csx.cam.ac.uk/courses/moved_Building/signals.pdf

¹¹⁴ <https://elixir.bootlin.com/linux/v6.7/source/kernel/signal.c>

Real Time Signals

In general, “Real Time Signals” (aka realtime signals or rtsignals) are similar to normal/original signals¹¹⁵ in the sense they have signal numbers and can be dealt with using the original signal functions. Since kernel 2.2, Linux started supporting real-time signals; their range is defined using the SIGRTMIN and SIGRTMAX macros. Linux supports 33-different real-time signals (ranging from 32-64). However, the range of available real-time signals varies according to the glibc threading implementation (this variation can occur at run time according to the available kernel and glibc), due to that programs should never refer them using hard-coded numbers¹¹⁶.

Overall, there are some differences between the original signals and real-time signals. Among them is the fact that real-time signals have multiple instances of individual real-time signals that can be queued (as opposed to the original signals which are merged when sent to a process that has a signal pending). Also, real-time signals can carry additional data with the signal number¹¹⁷.

Moreover, regarding the delivery priority lower number signals are given priority over higher number signals. For Linux this is true across the types of all signals, due to that the original signals have higher priority than all real-time signals. By the way, POSIX says that priority of real-time versus ordinary signals is unspecified, and only the real-time signals have a specified priority relative to each other¹¹⁸.

Lastly, for handling real-time signals and receiving the additional data we need to use the “sigaction” system call¹¹⁹ or library call¹²⁰ and not “signal” system call¹²¹. For sending a signal we should use the “sigqueue” syscall¹²² or the “sigqueue” library call¹²³ and not the “kill” syscall¹²⁴ or “the kill” library¹²⁵ - as shown below¹²⁶.

```
#include <signal.h>

union sigval {
    int    sival_int;
    void *sival_ptr;
};

int sigqueue (pid_t pid, int sig, const union sigval value);
```

¹¹⁵ <https://medium.com/@boutnaru/the-linux-concept-journey-signals-d1f37a9d2854>

¹¹⁶ <https://man7.org/linux/man-pages/man7/signal.7.html>

¹¹⁷ <https://www.nxp.com/docs/en/white-paper/CWLNxRTOSWP.pdf>

¹¹⁸ <https://davmac.org/davpage/linux/rtsignals.html>

¹¹⁹ <https://linux.die.net/man/2/sigaction>

¹²⁰ <https://linux.die.net/man/3/sigaction>

¹²¹ <https://man7.org/linux/man-pages/man2/signal.2.html>

¹²² <https://linux.die.net/man/2/sigqueue>

¹²³ <https://linux.die.net/man/3/sigqueue>

¹²⁴ <https://linux.die.net/man/2/kill>

¹²⁵ <https://linux.die.net/man/3/kill>

¹²⁶ <https://www.softprayog.in/programming/posix-real-time-signals-in-linux>

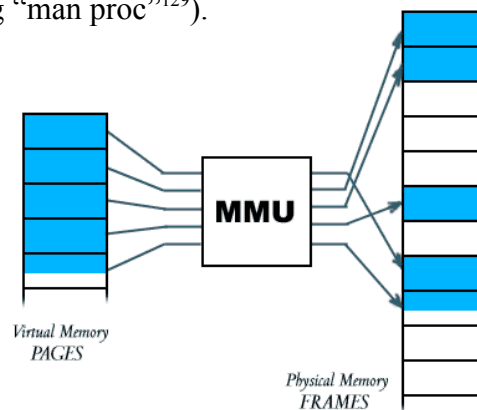
Memory Management - Introduction

Linux's memory management subsystem is surprisingly responsible for managing the system held in the system. It does that by using several components: virtual memory and demand paging, userspace memory allocator and an allocator for kernel structures.

Virtual memory is a memory management technique used to create an illusion for a running program that it has the entire memory address space only for itself with no need to coordinate the address used with other running programs. Two basic concepts we should get to know are frames (sometimes called physical pages/page frames) and pages (sometimes called virtual pages).

First, we are going to divide physical memory (total available memory the system has, we can see it using `cat /proc/meminfo | grep -i memtotal`) into blocks with the same size. We call them "frames", their size is a power of 2 (for now we are going to use the magic number of 4K more on that in future writeup). Second, we divide the virtual memory address space into blocks with the same size called "pages". The maximum range of the address space depends on the CPU and the OS (more on that in a future writeup). The size of the "pages" is also a power of 2, for simplicity the size of "pages" equals those of "frames". We can see that size from AUXV¹²⁷.

For those who know NAT/PAT (Network Address Translation/Port Address Translation) it is similar to the process done by the memory management subsystem. We take a "Virtual Address" which resides in a specific "page" and it is translated into a "Physical Address" which resides in a specific "frame". The component responsible for that is the MMU (Memory Management Unit), for now we are going to talk about hardware based only MMUs (we will cover soft-MMU in the future). The translation is based on tables created and managed by the OS ("page tables") and the MMU is responsible for using them for the translation and the enforcement of different checks (defined by page bit-like validity) - see the illustration below¹²⁸. In order to configure the memory management subsystem we can use the `/proc/sys/vm` interface (for more information about that I suggest reading `"man proc"`¹²⁹).



¹²⁷ <https://medium.com/@boutnaru/linux-the-auxiliary-vector-auxv-cba527871b50>

¹²⁸ http://dysphoria.net/OperatingSystems1/4_paging.html

¹²⁹ <https://man7.org/linux/man-pages/man5/proc.5.html>

Hard Link

As mentioned in previous writeup, an inode is data structure used by Unix/Linux like filesystems in order to describe a filesystem object¹³⁰. Thus, each hard link has the same inode value as the original file it points to - as shown in the screenshot below. When removing a hard link it just reduces the “link count” (think about it as a reference count), but it does not affect other links (the file is removed only when the count reaches “0”). Each hard link has a different file name and if the size of the content of one link changes then all the hard links file sizes are updated¹³¹.

Overall, we have a couple of limits regarding hard links which includes the fact we can create a hard link only for regular files (not including special files or directories). Also, we can not use a hard link to point to a file in a different filesystem¹³² - as shown in the screenshot below.

Lastly, we can use the “ln” command line utility in order to create hard links¹³³ - as shown in the screenshot below. It is based on the “link”/“linkat” system call which is described as “make a new name for a file”¹³⁴.

```
root@localhost:/tmp# echo tRoLleR > troller
root@localhost:/tmp# ln /tmp/troller /run/troller_link
ln: failed to create hard link '/run/troller_link' => '/tmp/troller': Invalid cross-device link
root@localhost:/tmp# mkdir troller_dir
root@localhost:/tmp# ln /tmp/troller /tmp/troller_dir/troller_link
root@localhost:/tmp# cat /tmp/troller_dir/troller_link
tRoLleR
root@localhost:/tmp# stat /tmp/troller
  File: /tmp/troller
  Size: 8          Blocks: 1          IO Block: 131072 regular file
Device: 0,23   Inode: 98020      Links: 2
Access: (0644/-rw-r--r--)  Uid: (  0/   root)   Gid: (  0/   root)
Access: 202    02:56:47.000000000 +0000
Modify: 202    06:29:17.000000000 +0000
Change: 202    06:29:17.000000000 +0000
 Birth: -
root@localhost:/tmp# ls -li /tmp/troller /tmp/troller_dir/troller_link
98020 -rw-r--r-- 2 root root 8 /tmp/troller
98020 -rw-r--r-- 2 root root 8 /tmp/troller_dir/troller_link
```

¹³⁰ <https://medium.com/@boutnaru/linux-what-is-an-inode-7ba47a519940>

¹³¹ <https://www.geeksforgeeks.org/soft-hard-links-unix/linux/>

¹³² <https://www.redhat.com/sysadmin/linking-linux-explained>

¹³³ <https://man7.org/linux/man-pages/man1/ln.1.html>

¹³⁴ <https://man7.org/linux/man-pages/man2/link.2.html>

Soft Link

As opposed to a hard link¹³⁵ which points to an inode, a soft link (aka symbolic link) points to another directory/file by name. Thus, soft links do not have the limitation of hard links in the sense of not being able to point to directories¹³⁶. By the way, we can use the “ln” command with the “-s” in order to create a soft link¹³⁷ - as shown in the screenshot below.

Moreover, we can use soft links to point to a file/directory in a different file system, having a different inode, different permissions and size. Because a soft link is not a mirror (as in the case of a hard link) the size of the file are the number of bytes needed to hold the name of the file/directory¹³⁸. The name of the file/directory can be based on a relative/full path - as shown in the screenshot below.

Lastly, soft links have their own limitations (we can't have only pros ;-). In case the name of the original file/directory (the target of the soft link) is changed it “breaks” the pointer/link - as shown in the screenshot below. Also, changing the permissions of the original file/directory does not affect the soft link. The creation of a soft link is based on the “link”/“linkat” system call which is described as “make a new name for a file”¹³⁹.

```
root@localhost:/tmp/troller# ls
file1
root@localhost:/tmp/troller# ln -s ./file1 ./file2
root@localhost:/tmp/troller# ls -l
total 1
-rw-r--r-- 1 root root 02:51 file1
l----- 1 root root file2 -> ./file1
root@localhost:/tmp/troller# mv file1 file
root@localhost:/tmp/troller# ls -l
total 1
-rw-r--r-- 1 root root file
l----- 1 root root file2 -> ./file1
root@localhost:/tmp/troller# cat ./file2
cat: ./file2: No such file or directory
root@localhost:/tmp/troller# ln -s /tmp/troller/file
file file2
root@localhost:/tmp/troller# ln -s /tmp/troller/file ./file3
root@localhost:/tmp/troller# ls -l
total 2
-rw-r--r-- 1 root root file
l----- 1 root root file2 -> ./file1
l----- 1 root root file3 -> /tmp/troller/file
```

¹³⁵ <https://medium.com/@boutnaru/the-linux-concept-journey-hard-link-f3e9b3d6b8c4>

¹³⁶ <https://kodekloud.com/blog/linux-create-and-manage-soft-links/>

¹³⁷ <https://man7.org/linux/man-pages/man1/ln.1.html>

¹³⁸ <https://www.redhat.com/sysadmin/soft-links-linux>

¹³⁹ <https://man7.org/linux/man-pages/man2/link.2.html>

BusyBox

BusyBox sees itself as the “The Swiss Army Knife of Embedded Linux”. It is a software component that combines tiny versions of many Unix utilities into a single binary. By doing so it can replace most utilities like GNU shellutils and fileutils. It is important to know that BusyBox has fewer options than the full-featured GNU utilities¹⁴⁰.

Moreover, BusyBox supports about 400 commands such as: “ls”, “ln”, “mv”, “mkdir”, “more” and “grep”. Also, it is an open source (GPL) project¹⁴¹. We can execute any command supported by BusyBox in the following way: “busybox <command>” - as shown in the screenshot below. The screenshot was taken from a “Buildroot Linux”¹⁴². We can download the source code of BusyBox for the official BusyBox website¹⁴³. There are also porting of BusyBox to other operating systems like Android¹⁴⁴.

Lastly, we can also execute BusyBox commands without calling the busybox executable directly. This works due to the fact that for every command supported by BusyBox a symbolic link with the name of the command is created. The symbolic link points to the BusyBox executable - as shown in the screenshot below. This causes the “cmdline”, which holds the complete command line for the process (unless it is a zombie process) to hold the name of the symbolic link and the arguments passed¹⁴⁵.

```
troller% busybox pwd
/tmp/troller
troller% busybox ls
troller1  troller2  troller3
troller% pwd
/tmp/troller
troller% ls
troller1  troller2  troller3
troller% ls -l `which ls`
lrwxrwxrwx   1 root    root      7 Jul 17  2020 /bin/ls -> busybox
troller% ls -l $(which pwd)
lrwxrwxrwx   1 root    root      7 Jul 17  2020 /bin/pwd -> busybox
troller% _
```

¹⁴⁰ <https://busybox.net/about.html>

¹⁴¹ <https://opensource.com/article/21/8/what-busybox>

¹⁴² <https://copy.sh/v86/?profile=buildroot>

¹⁴³ <https://busybox.net/downloads/>

¹⁴⁴ <https://github.com/meefik/busybox>

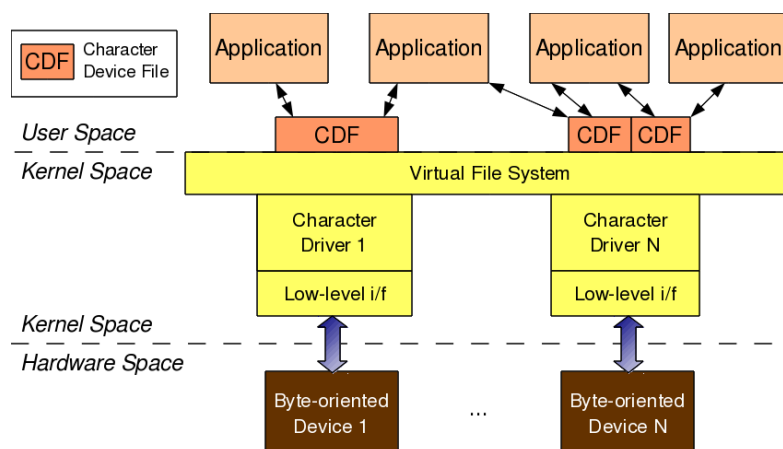
¹⁴⁵ <https://man7.org/linux/man-pages/man5/proc.5.html>

Character Devices

In Unix/Linux hardware devices are accessed using device files (located in “/dev”). A character device is used in case of slow devices (like sound card/joystick/keyboard/serial ports), which usually manage a small amount of data. Operations on those devices are performed sequentially byte by byte¹⁴⁶.

Moreover, as with every device also character devices have a major number and a minor number¹⁴⁷. We can think of the major number identifying the driver and the minor number identifying each physical device handled by the driver. Thus, we can say we have four main entities: the application, a character device file, a character device driver and a character device - as shown in the diagram below¹⁴⁸.

Lastly, in order to add a character device driver we need to register it with the kernel. This can be done by leveraging the “register_chrdev” function which is part of the “include/linux/fs.h” header file¹⁴⁹. In the case of kernel version “6.9.7” there are 46 files which reference that function¹⁵⁰.



¹⁴⁶ https://linux-kernel-labs.github.io/refs/heads/master/labs/device_drivers.html

¹⁴⁷ <https://medium.com/@boutnaru/the-linux-concept-journey-major-minor-numbers-56abe372482e>

¹⁴⁸ <https://sysplay.in/blog/linux-device-drivers/2013/05/linux-character-drivers/>

¹⁴⁹ <https://elixir.bootlin.com/linux/v6.9.7/source/include/linux/fs.h#L2746>

¹⁵⁰ https://elixir.bootlin.com/linux/v6.9.7/A/ident/register_chrdev

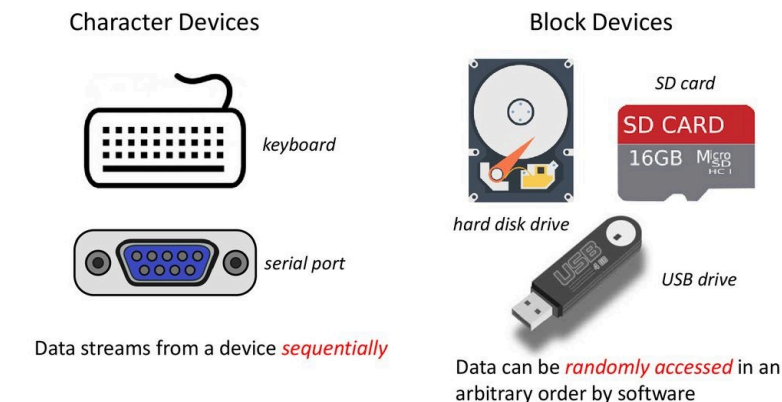
Block Devices

Block devices provide the ability to randomly access data which is organized in fixed-size blocks. Examples of such devices are: RAM disks, CD-ROM drives and hard drives. The speed of block devices is in general higher than those of character devices¹⁵¹. Another difference is that a character device has a single current position. However, in the case of a block device we need to be able to move to any random position for accessing/writing data¹⁵². We can see a comparison (with examples) between character devices and block devices in the diagram below¹⁵³.

Moreover, as with every device, block devices have a major number and a minor number¹⁵⁴. We can think of the major number identifying the driver and the minor number identifying each physical device handled by the driver. Although there is a difference between block devices they have some common abstractions like: data is typically buffered/cached on reads (an even writes if supported) and data is mostly organized as files and directories for ease of use by the user¹⁵⁵.

Lastly, in order to add a character device driver we need to register it with the kernel. This can be done by leveraging the “register_blkdev” macro which is part of the “include/linux/fs.h” header file¹⁵⁶. In the case of kernel version “6.9.7” there are 33 files which reference that macro¹⁵⁷.

Character vs. Block Devices



¹⁵¹ <https://medium.com/@boutnaru/the-linux-concept-journey-character-devices-0c75aa70ceb2>

¹⁵² https://linux-kernel-labs.github.io/refs/heads/master/labs/block_device_drivers.html

¹⁵³ <https://www.cs1training.com/block-device-vs-character-devices-in-linux-os/>

¹⁵⁴ <https://medium.com/@boutnaru/the-linux-concept-journey-major-minor-numbers-56abe372482e>

¹⁵⁵ <https://www.codingame.com/playgrounds/2135/linux-filesystems-101---block-devices/about-block-devices>

¹⁵⁶ <https://elixir.bootlin.com/linux/v6.9.7/source/include/linux/blkdev.h#L809>

¹⁵⁷ https://elixir.bootlin.com/linux/v6.9.7/A/ident/register_blkdev

Unnamed Pipe (Anonymous Pipe)

In general a Linux pipe allows commands to send output of one program to a different one. Thus, the term “Piping” means redirecting standard input/output/error of one process to another. In order to create a pipe (unnamed pipe) we can use the “|” character. For example “Command-(i)” | “Command-(i+1)...|..”Command-(i+n)” - as shown in the screenshot below¹⁵⁸.

Overall, the “unnamed pipe” that is created (like by the command shown above) can’t be accessed from a different session. Those types of pipes are created temporarily and deleted after the execution of the command. The pipe can be created using the pipe/pipe system call¹⁵⁹.

Lastly, by using an unnamed pipe we get a unidirectionality of the stream. Also, each shell has its own way of getting the status of each command in case of a pipeline, because by default only the status of the last command is saved to “\$?”. For example “bash” has the “\$PIPESTATUS” environment variable¹⁶⁰.

```
root@localhost:~# (sleep 1337 | sleep 1338 )&
[1] 1333
root@localhost:~# pidof sleep
1335 1334
root@localhost:~# ls -l /proc/1334/fd
total 0
lrwx----- 1 root root 64 Nov  7 03:01 0 -> /dev/tty1
l-wx----- 1 root root 64 Nov  7 03:01 1 -> 'pipe:[13457]'
lrwx----- 1 root root 64 Nov  7 03:01 2 -> /dev/tty1
root@localhost:~# ls -l /proc/1335/fd
total 0
lr-x----- 1 root root 64 Nov  7 03:01 0 -> 'pipe:[13457]'
lrwx----- 1 root root 64 Nov  7 03:01 1 -> /dev/tty1
lrwx----- 1 root root 64 Nov  7 03:01 2 -> /dev/tty1
```

¹⁵⁸ <https://opensource.com/article/18/8/introduction-pipes-linux>

¹⁵⁹ <https://man7.org/linux/man-pages/man2/pipe.2.html>

¹⁶⁰ <https://www.baeldung.com/linux/anonymous-named-pipes>

Linux File Types

As we know the philosophy of Linux is that “Everything is a file”. However, not all files are created equally. We have 7 different file types: directory, regular file, named pipe, socket, symbolic link, block device file and character device file¹⁶¹ - more information about each type in future writeups.

Overall, we can identify the type of a file using the “ls” utility using the “-l” argument¹⁶². The first character in each line identifies the type of the file - as described in the table below¹⁶³.

Lastly, we can display only specific file types by filtering the output of “ls” using “grep”¹⁶⁴. For example if we want to see all regular files in the current directory we can use the following one-liner: “ls -la | grep ^-”¹⁶⁵.

- rw-----	: Regular File
d rw-r-xr-x	: Directory File
l rw-rw-rw-	: Link File
C rw-rw----	: Character Device File
S rw-rw-rw-	: Socket File
p rw-----	: Named Pipe File
b rw-rw----	: Block Device File

How to Identify
File Types
on Linux

¹⁶¹ <https://linuxconfig.org/identifying-file-types-in-linux>

¹⁶² <https://man7.org/linux/man-pages/man1/ls.1.html>

¹⁶³ <https://www.2daygeek.com/wp-content/uploads/2019/01/find-identify-file-types-in-linux-4.png>

¹⁶⁴ <https://man7.org/linux/man-pages/man1/grep.1.html>

¹⁶⁵ <https://www.2daygeek.com/find-identify-file-types-in-linux/>

Regular File

As we know the philosophy of Linux is that “Everything is a file”. However, not all files are created equally. As you know there are seven different file types used in Linux¹⁶⁶. The following writeup is going to focus on regular files, those that when using “ls -l”¹⁶⁷ are marked with “-” - as shown in the screenshot below (taken using <https://copy.sh/v86/?profile=archlinux>).

Overall, it is important to understand that a type of a “regular file” is from the perspective of the operating system and it does not describe the format of the file itself. We can check the format of a file using the “file” command¹⁶⁸. For example a “Regular File” can be an image file (such as PNG, gif, JPEG or BMP), a text file, an executable (like ELF or PE), archive files (like RAR and ZIP) and more - as shown in the screenshot below.

Lastly, probably the easiest way to create a regular file is by using the “touch” command¹⁶⁹. By the way, we can create a regular file in any directory¹⁷⁰. For that to happen we just need to have write permissions to the directory itself. Also, the mounted filesystem should not be read-only.

```
root@localhost:~# ls -l /etc/passwd
-rw-r--r-- 1 root root 1159 Nov  6 18:44 /etc/passwd
root@localhost:~# file /etc/passwd
/etc/passwd: ASCII text
root@localhost:~# ls -l /usr/bin/passwd
-rwxr-xr-x 1 root root 62116 Sep  9 2020 /usr/bin/passwd
root@localhost:~# file /usr/bin/passwd
/usr/bin/passwd: setuid ELF 32-bit LSB pie executable, Intel 80386, version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux.so.2, BuildID[sha1]=6f854a4aa3c4532fa9842c724f58c6de50377a14, for GNU/Linux 3.2.0, stripped
```

¹⁶⁶ <https://medium.com/@boutnaru/the-linux-concept-journey-linux-file-types-4cb622887331>

¹⁶⁷ <https://man7.org/linux/man-pages/man1/ls.1.html>

¹⁶⁸ <https://man7.org/linux/man-pages/man1/file.1.html>

¹⁶⁹ <https://man7.org/linux/man-pages/man1/touch.1.html>

¹⁷⁰ <https://www.geeksforgeeks.org/how-to-find-out-file-types-in-linux/>

Directory File

As you know there are seven different file types used in Linux¹⁷¹. Among them we have a “directory” file type, we can think about it as a “file” that holds in its content file names and there representing inode numbers¹⁷².

Thus, using the “rm” utility¹⁷³ basically removes the file from the directory and does not delete it (until the reference count equals “0”). Also, because of that removing a file does not require any permissions on the file itself, it requires having “write permissions” to the directory containing the file.

Lastly, in order to create a new directory we can use the “mkdir” utility¹⁷⁴ in order to create a new directory or the “rmdir” utility¹⁷⁵ in order to remove a directory - as shown in the screenshot below (taken using <https://copy.sh/v86/?profile=archlinux>). A directory is marked with a “d” as the first character in the output of the “ls -l” command¹⁷⁶ - also shown in the screenshot below.

```
root@localhost:/tmp/troller# type mkdir
mkdir is hashed (/usr/bin/mkdir)
root@localhost:/tmp/troller# file `which mkdir`
/usr/bin/mkdir: ELF 32-bit LSB pie executable, Intel 80386, version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux.so.2, BuildID[sha1]=f86e3978c22e994928b83471ee5db2e046cb6f3c, for GNU/Linux 4.4.0, stripped
root@localhost:/tmp/troller# mkdir dir
root@localhost:/tmp/troller# ls -l
total 1
drwxr-xr-x 2 root root 4096 Jan 10 10:10 dir
root@localhost:/tmp/troller# type rmdir
rmdir is /usr/bin/rmdir
root@localhost:/tmp/troller# file `which rmdir`
/usr/bin/rmdir: ELF 32-bit LSB pie executable, Intel 80386, version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux.so.2, BuildID[sha1]=a7999caabdd4a599a216ae82227b6477e038786, for GNU/Linux 4.4.0, stripped
root@localhost:/tmp/troller# rmdir ./dir/
root@localhost:/tmp/troller# ls
```

¹⁷¹ <https://medium.com/@boutnaru/the-linux-concept-journey-linux-file-types-4cb622887331>

¹⁷² <https://medium.com/@boutnaru/linux-what-is-an-inode-7ba47a519940>

¹⁷³ <https://linux.die.net/man/1/rm>

¹⁷⁴ <https://man7.org/linux/man-pages/man1/mkdir.1.html>

¹⁷⁵ <https://man7.org/linux/man-pages/man1/rmdir.1.html>

¹⁷⁶ <https://man7.org/linux/man-pages/man1/ls.1.html>

Link File (aka Symbolic Link)

As you know there are seven different file types used in Linux¹⁷⁷. Among them we have a “link” file type (aka “symbolic link”), which is used for pointing to other files¹⁷⁸. When using “ls -l”¹⁷⁹ link files are marked with “l” in the output - as shown in the screenshot below.

Overall, we can use the “ln” command¹⁸⁰ to make links between files. It supports both creating “hard links”¹⁸¹ and “symbolic links”. In order to create a “link file” we need to use the “-s” switch - as shown in the screenshot below.

Lastly, as opposed to “hard links” that can’t be created to a directory a “symbolic link” can point to a directory - as shown in the screenshot below. By the way, “symbolic links” are also called “soft links”¹⁸².

```
root@localhost:/tmp/troller# ls -l
total 1
drwxr-xr-x 2 root root 0 [redacted] dir
-rw-r--r-- 1 root root 0 [redacted] file
root@localhost:/tmp/troller# ln -s file file_link
root@localhost:/tmp/troller# ls -l
total 2
drwxr-xr-x 2 root root 0 [redacted] dir
-rw-r--r-- 1 root root 0 [redacted] file
l----- 1 root root 0 [redacted] file_link -> file
root@localhost:/tmp/troller# ln -s dir dir_link
root@localhost:/tmp/troller# ls -l
total 2
drwxr-xr-x 2 root root 0 [redacted] dir
l----- 1 root root 0 [redacted] dir_link -> dir
-rw-r--r-- 1 root root 0 [redacted] file
l----- 1 root root 0 [redacted] file_link -> file
root@localhost:/tmp/troller# _
```

¹⁷⁷ <https://medium.com/@boutnaru/the-linux-concept-journey-linux-file-types-4cb622887331>

¹⁷⁸ <https://www.freecodecamp.org/news/linux-ln-how-to-create-a-symbolic-link-in-linux-example-bash-command/>

¹⁷⁹ <https://man7.org/linux/man-pages/man1/ls.1.html>

¹⁸⁰ <https://www.man7.org/linux/man-pages/man1/ln.1.html>

¹⁸¹ <https://medium.com/@boutnaru/the-linux-concept-journey-hard-link-f3e9b3d6b8c4>

¹⁸² <https://www.geeksforgeeks.org/soft-hard-links-unixlinux/>

Socket File

A socket file is one of the file types supported under Linux¹⁸³. The main goal of a socket file is to pass information between applications. For convenient applications which create socket files use the “.socket”/”*.sock” suffix for the file name. Examples for such file names are: acpid.socket and docker.sock¹⁸⁴.

Overall, we can use the “AF_UNIX”/”AF_LOCAL” socket family for communicating between processes on the same machine. This type of socket is known as “Unix Domain Sockets”. This type of sockets can be unnamed or to be bound to a file-system path. We can use both “SOCK_STREAM” (for stream oriented socket) or “SOCK_DGRAM” (for data-gram oriented). Since Linux 2.6.4 we can use “SOCK_SEQPACKET” for a sequenced-packet socket, that delivers messages in the order that they were sent¹⁸⁵.

Lastly, when we create such a socket a “socket file” is created - as shown in the screenshot below. Due to the fact it is a socket we can use any of the socket API functions on¹⁸⁶ - it is a big advantage as opposed to using regular files as an IPC. Also, it is important to understand that even if “Unix Domain Sockets” are designed for local communication only, if we manage to create our socket file on a remote device (like by using NFS/SMB) we can pass information between remote processes - it can also be done using regular files however it is not the preferred method for that.

```
root@localhost:~# ls -la /tmp/troller/
total 1
drwxr-xr-x 2 root root 0 2024 .
drwxrwxrwt 5 root root 51 18:35 ..
root@localhost:~# python3 -c "import socket; server=socket.socket(socket.AF_UNIX,socket.SOCK_STREAM); server.bind('/tmp/troller/troller.sock'); server.listen(1)"
root@localhost:~# ls -la /tmp/troller/
total 2
drwxr-xr-x 2 root root 0 2024 .
drwxrwxrwt 5 root root 51 18:35 ..
srwxr-xr-x 1 root root 0 2024 troller.sock
root@localhost:~#
```

¹⁸³ <https://medium.com/@boutnaru/the-linux-concept-journey-linux-file-types-4cb622887331>

¹⁸⁴ https://www.bogotobogo.com/Linux/linux_File_Types.php

¹⁸⁵ <https://man7.org/linux/man-pages/man7/unix.7.html>

¹⁸⁶ <https://www.baeldung.com/linux/python-unix-sockets>