Time Lord's adventures: abusing time on Linux systems

Daniels Heincis
12.08.2025.
WHY

About me

- Daniels Heincis
- Cybersecurity educator, pentester, programmer, CTF organizer, physicist

Contents

- How time work
- How can we manipulate time
- What are the consequences of time manipulation

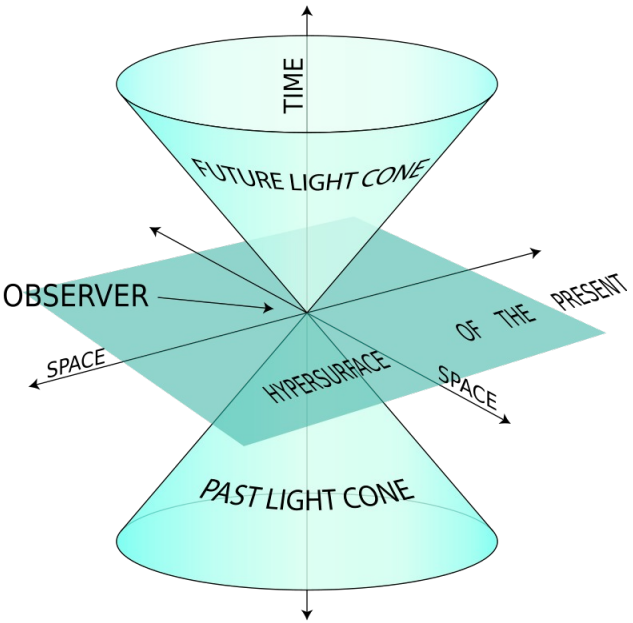
How time works

Time dilation equation

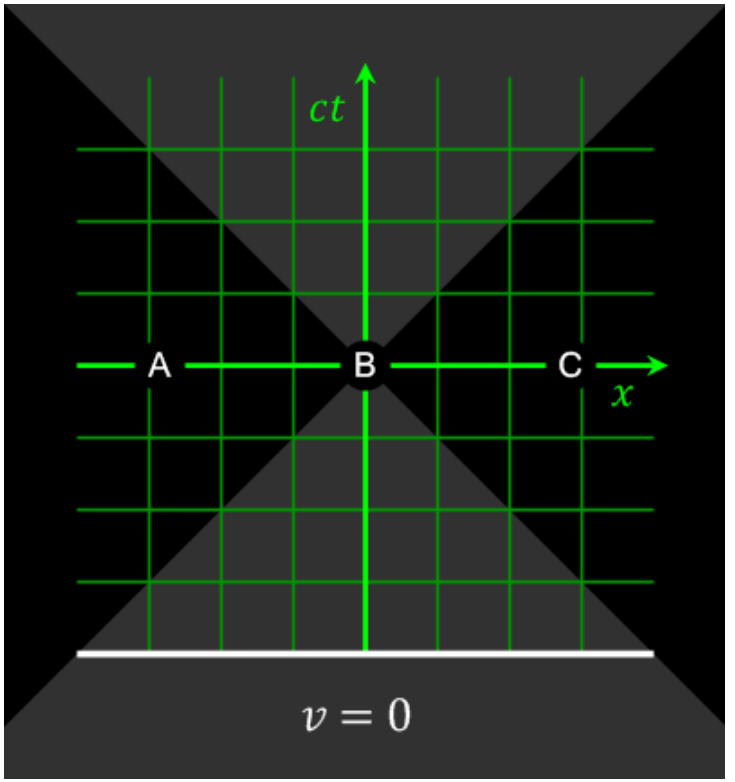
$$\Delta t' = \frac{\Delta t}{\sqrt{1 - \frac{v^2}{c^2}}} = \gamma \Delta t$$

EFE

$$R_{\mu\nu} - \frac{1}{2} R g_{\mu\nu} + \Lambda g_{\mu\nu} = \kappa T_{\mu\nu}.$$



By SVG version: K. Aainsqatsi at en.wikipediaOriginal PNG version: Stib at en.wikipedia - Transferred from en.wikipedia to Commons.(Original text: self-made), CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=2210907



By User:Acdx - Self-made, based on Image:Relativity_of_Simultaneity.svg, source code: en:User:Acdx/Relativity_of_Simultaneity_Animation, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=5560059

How time works **on Linux**

- Hardware clock (Real Time Clock (RTC) or CMOS clock)
- System clock
- Some network time protocols or other time syncing mechanisms

Hardware clock

- Stores persistent time
- Battery-backed, survives reboots
- UTC by convention, usually
- Read at boot time
- Can drift without correction

System clock (1)

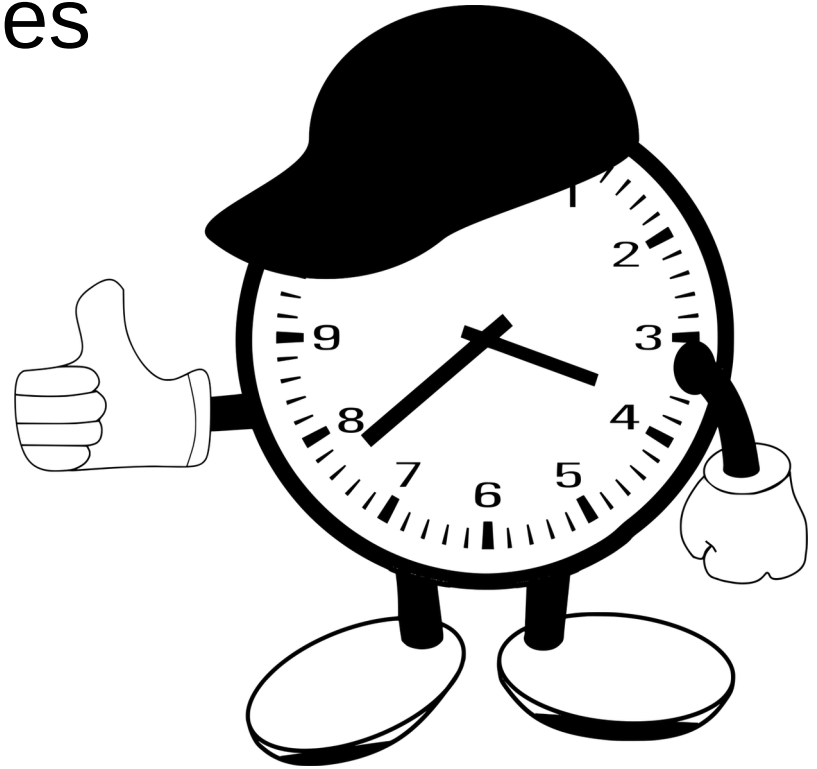
- Starts at boot time
- Initialized from hardware clock
- Some parts of it lives in RAM some in files
- Used by all processes

System clock (2)

- Can be modified by user
- Adjusted by NTP/chronyd
- Core to scheduling

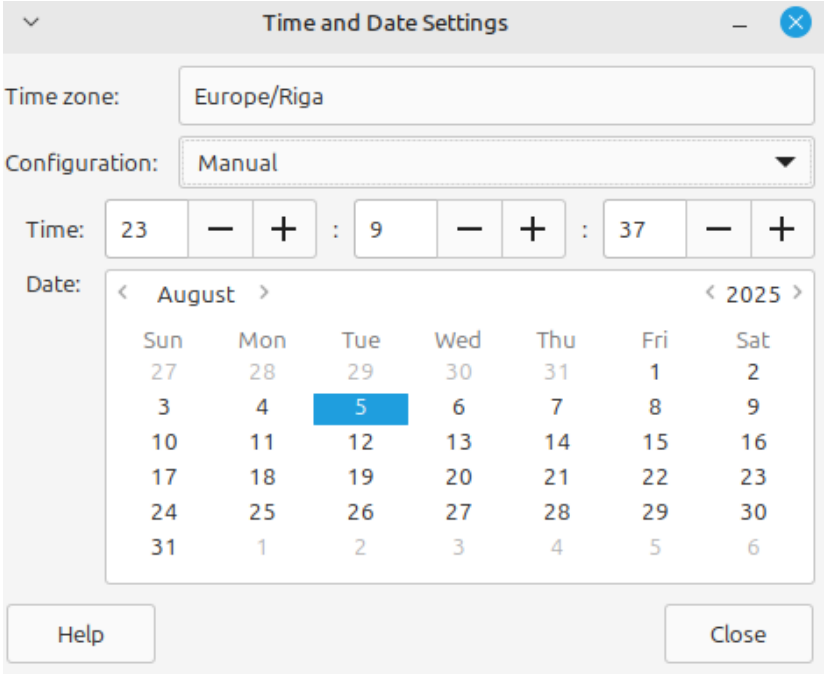
How can we modify system time?

- Disable NTP, hwclock sync, and any other time synchronization services



How can we modify system time?

- Use system utility “Time and Date”



How can we modify system time?

- Use timedatectl

```
user@terminal:~$ timedatectl
               Local time: Tue 2025-08-05 23:23:02 EEST
           Universal time: Tue 2025-08-05 20:23:02 UTC
                 RTC time: Tue 2025-08-05 20:23:02
                Time zone: Europe/Riga (EEST, +0300)
System clock synchronized: yes
              NTP service: inactive
          RTC in local TZ: no
```

How can we modify system time?

- libfaketime/faketime

```
user@terminal:~$ faketime

Usage: faketime [switches] <timestamp> <program with arguments>

This will run the specified 'program' with the given 'arguments'.
The program will be tricked into seeing the given 'timestamp' as its starting date and time.
The clock will continue to run from this timestamp. Please see the manpage (man faketime)
for advanced options, such as stopping the wall clock and make it run faster or slower.

The optional switches are:
  -m                 : Use the multi-threaded version of libfaketime
  -f                 : Use the advanced timestamp specification format (see manpage)
  --exclude-monotonic : Prevent monotonic clock from drifting (not the raw monotonic one)
  -p PID             : Pretend that the program's process ID is PID
  --date-prog PROG   : Use specified GNU-compatible implementation of 'date' program
```

How can we modify system time?

- Bashing

Time freeze

Local JS timer workaround

System time travel consequences

- Scheduled task disruption
- sudo session timeout bypass
- Kerberos / Active Directory login DOS
- Licensing system issues
- Broken 2FA / TOTP codes
- SSL/TLS certificate validation issues
- Log tampering
- Bypassing **some** delays

But what about ALL delays?

CLOCK_MONOTONIC

A nonsettable system-wide clock that represents monotonic time since—as described by POSIX—"some unspecified point in the past". On Linux, that point corresponds to the number of seconds that the system has been running since it was booted.

The **CLOCK_MONOTONIC** clock is not affected by discontinuous jumps in the system time (e.g., if the system administrator manually changes the clock), but is affected by frequency adjustments. This clock does not count time that the system is suspended. All **CLOCK_MONOTONIC** variants guarantee that the time returned by consecutive calls will not go backwards, but successive calls may—depending on the architecture—return identical (not-increased) time values.

Monotonic time

- Always moves forward*, never rewinds
- Immune to time jumps
- Used for time intervals, not timestamps
- Starts ticking at system boot
- Not **directly** observable in user space

Who uses which clock?

System time

- File timestamps (mtime, ctime)
- Logs (journalctl, syslog)
- TLS/SSL certificate checks
- Scheduled tasks (cron, at)

Monotonic time

- sleep, usleep, nanosleep
- Systemd services (WatchdogSec, TimeoutStartSec)
- Rust/Go timers, C++ std::chrono::steady_clock
- Some benchmarking libs

Can we manipulate monotonic clock?

Of course, it’s Linux, we can do whatever we want

How?

- Kernel module patch
- Hypervisor/VM-based time manipulation
- Syscall interception

Next steps and future research

- Read and understand Linux kernel
- Write some patches
- Write some automatization scripts
- Look into time keeping in quantum computers

Sources

- https://man7.org/linux/man-pages/man2/clock_gettime.2.html
- https://www.kernel.org/doc/Documentation/timers/timekeeping.txt
- https://classes.engineering.wustl.edu/cse422/code_pointers/06_a_brief_guide_to_time_in_linux.html
- https://wiki.archlinux.org/title/System_time
- https://man7.org/linux/man-pages/man2/clock_gettime.2.html
- https://www.man7.org/linux/man-pages/man7/time_namespaces.7.html

Slides and code used in demos

Thank you for attention!
Questions?