Exercise sheet 4 – OS architecture

Goals:

- Boot procedure and steps
- `systemd` usage
- User vs. kernel space

Exercise 4.1: Recap boot procedure (theoretical) *(no solution prop. provided)*

(a) Describe the difference (advantages/disadvantages) between BIOS and UEFI

Proposal for solution: See lecture slide UEFI (7) and read `https://www.marksei.com/bios-uefi-explained`

(b) What are the advantages of GPT over MBR?

Proposal for solution: Compare lecture slide MBR (8) and GPT (9).

(c) Is the BIOS compatible with the GPT?

Proposal for solution: No, if you have a BIOS, you can't boot with a GPT table. Source: `https://www.marksei.com/bios-uefi-explained/` -> What the BIOS can and can't do.

Exercise 4.2: systemd (theoretical) *(no solution prop. provided)*

(a) What is `systemd` doing?

Proposal for solution: Starts the user space processes on boot.

(b) What is a default target?

Proposal for solution: The target that is automatically started when systemd starts. Usually, it's redirects to `multi-user.target` on servers or to `graphical.target` on desktop systems.

(c) Can `systemd` start the kernel?

Proposal for solution: No, because the kernel starts `systemd` as the first process, and not vice versa.

(d) What is a `systemd` `<daemon>.service` file?

Proposal for solution: A service file describes how and when a service (daemon) should be started.

(e) Where are the `systemd` config files located on the system?

Proposal for solution: Usually in: `/etc/systemd/system/`

Exercise 4.3: systemd (practical)

(a) Run `OS_exercises/sheet_04_os_arch/systemd/installDaemon.sh` to install the daemon `demo_timer_daemon`. Every second, the `demo_timer_daemon` daemon writes the current time into its log file (`/var/log/demo_timer.log`).

Proposal for solution:

```
sudo OS_exercises/sheet_04_os_arch/systemd/installDaemon.sh
```

(b) Start the daemon.

Proposal for solution: `sudo service demo_timer_daemon start`
or:
`sudo systemctl start demo_timer_daemon`

(c) Check if the daemon is started. You can additionally check the log file with `tail -f /var/log/demo_timer.log`

Proposal for solution: `sudo service demo_timer_daemon status`
or:
`sudo systemctl status demo_timer_daemon`

(d) Stop the daemon.

Proposal for solution: `sudo service demo_timer_daemon stop`
or:
`sudo systemctl stop demo_timer_daemon`

(e) Activate the daemon for the `multi-user.target`.

Proposal for solution: `sudo systemctl enable demo_timer_daemon`

(f) Reboot the VM and check if the daemon is automatically started.

Proposal for solution: After the VM has rebooted:
`sudo service demo_timer_daemon status`
or:
`sudo systemctl status demo_timer_daemon`

(g) Deactivate the daemon for the `multi-user.target`.

Proposal for solution: `sudo systemctl disable demo_timer_daemon`

(h) Reboot the VM and check if the daemon is automatically started.

Proposal for solution: After the VM has rebooted:
`sudo service demo_timer_daemon status`
or:
`sudo systemctl status demo_timer_daemon`

Exercise 4.4: User vs. kernel space (theoretical) *(no solution prop. provided)*

(a) Can a process, running with root privileges, directly access the kernel space?

Proposal for solution: No, it's not possible. An SVC is required.

(b) How many processes has the kernel space?

Proposal for solution: Basically it's not distributed in different processes, therefore, it's one process. But the kernel also has some parallel mechanisms.

(c) Can a user space process directly access the memory of another process?

Proposal for solution: No, only to the assigned memory.

(d) Can a user space process directly communicate with a device?

Proposal for solution: No, it's not possible. An SVC is required for that.

(e) How can a user space process print something on a console/terminal (please consider the different spaces)?

Proposal for solution:

- User space: By using `printf()`, which internally performs an SVC.
- Kernel space: Can directly print something, but not with `printf()`, because the kernel is already in the kernel space.

(f) How is an SVC identified?

Proposal for solution: The SVC instruction takes a number as an argument. The number—called service number—is used by the kernel to identify which service is requested.