DM510 Operating Systems
Spring 2026

Exercise Sheet 1

Complete before tutorial on Thursday, February 12th

In the lecture on Monday, February 9th, you should have received a Raspberry Pi Zero. If you have not, you can pick one up in the Tutorial. Follow the instructions on https://larsrohwedder.com/teaching/dm510-26/pi and get SSH access to it. This is important for the future programming projects.

Learning goals

Be able to

- differenciate between kernel, system programs, and user programs
- explain the functioning of interrupts
- explain the purpose of kernel and user mode
- list different categories of operating system services and system programs and examples
- state different computing environments and their individual challenges
- explain role of source files, header files, object files, compiler and linker
- write and understand simple Makefiles

Chapter 1

Exercise 1. Explain what an interrupt is and name three use cases.

Exercise 2. A system clock is a hardware device that can be instructed to send an interrupt to the CPU after a specified number of CPU cycles. How can a system clock be used by the kernel to prevent user processes from blocking the CPU by executing indefinitely?

Exercise 3. Which of the following instructions should be privileged?

a. Mask interrupts

b. Issue a trap instruction

c. Switch between user and kernel mode

d. Write to I/O device's buffer

e. Load a word from main memory into a register

Exercise 4. Discuss special requirements and challenges for the operating system arising in the following use-cases. Make connection to terminology from the lecture if possible.

- Industrial robot in a production line
- System that trains LLM models
- Smartwatch

Chapter 2

Exercise 5. For each of the following Linux command line tools, describe what they do and assign them to one categories of system programs or services mentioned in the lecture. To find out about a command line tool, say, `dmesg`, open its manual page either using `man dmesg` on a Linux command line or via the web version https://man7.org/linux/man-pages/man1/dmesg.1.html.

- `dmesg, ls, vi, make, kill, killall, cd, top, sh, cat, ps, mv`

Exercise 6. What are advantages of implementing operating system functionality in system programs rather than in the kernel and vice versa. What is a microkernel and what is a monolithic kernel?

Exercise 7. A multi-boot system is a computer where the user can choose one of several operating systems to run. Which software is responsible for implementing multi-boot?

C Programming

The following exercises assume that you have access to a Linux system. You can for example use your Raspberry Pi.

Exercise 8. Create a file `bits.c`, which contains implementations of the following functions:

- `int set_bit(int* word, int bit)`
- `int unset_bit(int* word, int bit)`
- `int toggle_bit(int* word, int bit)`

These functions should modify the integer pointed to by `word` by changing the bit at position `bit` to 1, 0, or the opposite of its current value. The function should return 0 if successful and -1 when `bit` is out of bounds.
Discuss why `word` is passed by reference (pointer) and not by value.

Exercise 9. Create a header file `bits.h` for the function definitions of the previous exercise. Create a main file `main.c`, which has a main function that tests the bit functions on some values and prints out the result. Compile each source file to an object file using `gcc -c bits.c` and `gcc -c main.c`. Link together the object files using `gcc -o main bits.o main.o` (or your favorite alternative C compiler). Discuss the role of:

- source files, header files, object files, executables

Exercise 10. `make` is a widely used build system in Unix based operating systems. A special file with the name `Makefile` specifies how to build a project. It has the following syntax

```
target: dependence1 dependence2 ...
        build command 1
        build command 2
        ...
```

Note that the commands must be preceeded by a tab character and not spaces. A `Makefile` can have several build targets. `target` is usually a file that needs to be built. `dependenceX` is a file that is required to build `target`. If the dependence is itself a build target then it will be built first. `make` will avoid building targets if possible. This is determined by the timestamps of the files: if dependencies are older than the target, it does not need to be rebuilt. See https://www.gnu.org/software/make/manual/make.html for more details and examples.

a. Create a file called `Makefile` containing the build instructions from the previous exercises. Remove all object and executable files and rebuild the executable using the command `make`.

b. Change `main.c`, but not `bits.c`. Execute `make` again, observe and interpret what is being rebuilt.

c. Discuss the advantages of `make` over using a shell script (a file containing a series of commands to be executed in the terminal) for building a project.