Operating Systems 2023/24
Assignment 1

Prof. Dr. Frank Bellosa
Peter Maucher, M.Sc.
Lukas Werling, M.Sc.

Submission Deadline: Monday, November 13nd, 2023 – 09:00

A new assignment will be published every week, right after the last one was due. It must be completed before its submission deadline.

The assignments must be filled out online in ILIAS. Handwritten solutions are no longer accepted. You will find the online version for each assignment in your tutorial's directory. **P-Questions** are programming assignments. Download the provided template from ILIAS. Do not fiddle with the compiler flags. Submission instructions can be found in the introductory section below. **After you have submitted your final solutions, make sure you finish the ILIAS test ("Test beenden").**

In this assignment you will get familiar with the C programming language.

T-Question 1.1: Basics

a. What is the most important piece of software running on a computer, the operating system or applications? Why? **1 T-pt**

b. Why is abstraction a central task of an operating system? **1 T-pt**

c. As computers have become cheap, today most devices (desktops, tablets, smartphones, etc.) that are running an operating system are used by a single user only. Why is protection still as important? **1 T-pt**

T-Question 1.2: C Basics

a. How many bits usually make up a byte? Which C type can fit exactly one byte? **1 T-pt**

b. What is the advantage of fixed-size integer types like `uint32_t`? **1 T-pt**

c. What is the size of the following struct (`sizeof(struct x)`)? Explain. **2 T-pt**

```
struct x { uint8_t a; int32_t b; uint64_t c; };
```

d. Now, what is the size of the following struct (`sizeof(struct y)`)? Explain. **2 T-pt**

```
struct y { uint64_t c; int32_t b; uint8_t a; };
```

e. Explain all functions of the unary `*` and `&` operators in C. **2 T-pt**

About the Programming Assignments

The following introductory words outline our expectations of your work and the requirements your solutions have to fit.

Write Readable Code

In your programming assignments, you are expected to write well-documented, readable code. There are a variety of reasons to strive for clear and readable code: Code that is understandable to others is a requirement for any real-world programmer, not to mention the fact that, after enough time, you will be in the shoes of one of the others when attempting to understand what you wrote in the past. Finally, clear, concise, well-commented code makes it easier to review your assignment! (This is especially important if you cannot get the assignment running. If you cannot figure out what is going on, how do you expect us to do it?)
There is no single right way to organize and document your code. It is not our intent to dictate a particular coding style for this class. The best way to learn about writing readable code is to read other people's code.
Here are some general tips for writing better code:

- Split large functions. If a function spans multiple pages, it is probably too long.
- Group related items together, whether they are variable declarations, lines of code, or functions.
- Use descriptive names for variables and procedures. Be consistent with this throughout the program.
- Comments should describe the programmer's intent, not the actual mechanics of the code. A comment which says "Find a free disk block" is much more informative than one that says "Find first non-zero element of array".

Write Compilable And Executable Code

Obscure code is bad, but uncompilable code is even worse. To increase your coding awareness, we expect you to use the GNU C compiler with some restrictions on warning-behavior as written in the makefiles. Do not change these flags!
If you are unable to write a fully working solution, at least make sure that your partial solution does compile, even though it might not produce the correct result. Document your intents and problems as comments in the source file to give your tutor a head start in understanding your code.

Groups

We assume (and recommend) that you will complete the assignment on your own. Feel free to discuss your solutions with your colleagues, but do not share code. Please inform your tutor if you still want to submit assignments as a group to avoid duplicate work.

Templates And Stubs

You will find templates for all programming assignments in ILIAS. Unpack them with the command `tar xf asstXX-templates.tar.gz`. The archives contain a directory for each individual task, wherein you can find several files:

<**task name**>**.h** A header file defining the function prototypes as listed in the assignment's description. You should not modify this.

<**task name**>**.c** Put your solution in here. Your tutor will receive this file only.

main.c Contains the entry point in the resulting program. While we provide trivial test cases, you should write your own test code here.

Makefile Call `make` to build your sources.

These templates should ease your work as well as ours, so don't change anything unless explicitly allowed.

Assignment Submission

To submit your solution, please upload your <**task name**>**.c** file in the online ILIAS assignment in the respective question. Do not change the name of this file. All other files are not part of your solution and should not be uploaded. They will be ignored.

About the Programming Exam

The problems in the programming exam will be of the following three categories:

1. Questions and programming tasks about basic C programming. *(C Basics)*
2. Programming tasks about using the POSIX API. *(POSIX)*
3. Programming tasks about implementing operating system components. *(OS)*

The programming assignments will be marked accordingly to help you practise for the exam. If you have trouble with C programmming, we strongly recommend doing at least the *C Basics* assignments!

P-Question 1.1: Pointers *(C Basics)*

Download the template **p1** for this assignment from ILIAS. You can only upload the file `pointers.c`.

a. Write a function that returns the rounded-towards-zero, arithmetic mean over all values in an array. The array is passed to the function via a pointer to the first element of the array and is bounded by the parameter `size` (if `size = 0`, set result to 0). **1 P-pt**

```
int average(int *arrayPointer, unsigned int size);
```

b. Write another function that returns the rounded-towards-zero, arithmetic mean over all values in an array. This time, the array elements do not contain the values themselves – they contain pointers to the actual values. The number of elements is given by the parameter `size` (again, set average $= 0$ if `size = 0`). **1 P-pt**

```
int averageIndirect(int **arrayPointer, unsigned int size);
```

P-Question 1.2: Printf *(C Basics)*

Download the template **p2** for this assignment from ILIAS. You can only upload the file `print.c`.

a. Write a function that prints a 64-bit signed integer and a null-terminated ASCII string with a single call to `printf`. The number and string should be separated by a whitespace. Each call to your function should print the output to a new line. For the integer use the platform-independent format from `inttypes.h`. **2 P-pt**

```
void print_line(int64_t number, char *string);
```

P-Question 1.3: String to Integer Conversion *(C Basics)*

Download the template **p3** for this assignment from ILIAS. You can only upload the file `parseint.c`.

a. Write a function that converts a single decimal digit in a char to an integer. You may use neither a library function nor a lookup table for this task. Return -1 if the given char is not a valid decimal digit. **1 P-pt**

```
int parseDecimalChar(char c);
```

b. Write a function that converts from an octal or decimal string to an integer. Reuse your function from above, but do not use any library function. Your function should recognize octal numbers through a leading zero. You may assume that the resulting integers fit into an int. Return -1 if the given string is not a valid octal or decimal number. **3 P-pt**

```
int parseInt(char *string);
```

P-Question 1.4: Pushing Bits Around *(C Basics)*

Download the template **p4** for this assignment from ILIAS. You can only upload the file `bits.c`.

a. Given a large array `A` (e.g., 1 MiB), write three functions `getN()`, `setN()` and `clrN()` that return, set, or clear the n'th bit (not byte!) in the array, respectively. The first bit (the bit with index 0) should be the least significant bit (LSB) of the first integer. **3 P-pt**

```
int getN(uint64_t *A, size_t n);
void setN(uint64_t *A, size_t n);
void clrN(uint64_t *A, size_t n);
```

b. Write a function that rotates the bits of the integer `i` by `n` bits to the right (MSB → LSB). Bits rotated "out" of the integer shall be rotated "in" on the other side. Keep in mind that `n` may be negative which shall rotate `i` to the left. **2 P-pt**

```
uint64_t rot(uint64_t i, int n);
```

Total:
11 T-pt
13 P-pt