02335 Operating Systems, Fall 2024
Instructions for Assignment 1:
Reintroduction to C — v.1

September 2, 2024

1 Introduction

In this assignment you will implement a simple program that stores numbers based on commands received from the command line via standard in (stdin). You are free to solve this problem in any fashion you like, using any data structures you wish to implement.

The purpose of this assignment is to refresh your C programming skills and to introduce you to system call semantics. You may be familiar with printf and other similar functions provided by the C library. These functions interact with the operating system kernel, in this case Unix, via its system call interface. In this assignment you must use the Unix read and write system calls directly.

In addition, it is required that you use dynamically allocated memory to manage your number storage system. Not only is it a technical requirement, but you have no knowledge of how many numbers you will have to store prior to execution. Therefore a statically sized collection is not sufficient. Again, what data structures you use is entirely up to you.

As part of this assignment, you will have to establish a working development environment for the course. In general the assignments will assume that you are programming for a Linux environment, although you may probably also use any Posix-compliant environment such as MacOS.

2 Learning objectives

During this assignment you will be working towards the following learning objectives:

- You can explain how the compiler, assembler and linker are used to create executables.
- You can apply standard programming methodologies and tools such as test-driven development, build systems, debuggers.
- You can explain the role of each component of a compilation tool-chain used in system programming and how the components interact.

- Given reference material, you can implement C programs with pointers, pointer arithmetic, arrays, structs, memory management and low level I/O.
- You can explain in your own words what the system call interface to an operating system is, and develop C applications that make use of it.

3 Reference material

When reading literature, it is helpful to remember that the AMD64 architecture is known under many synonyms, including AMD64, x86-64, x86_64, EM64T, IA-32e and Intel 64.

During the work on the assignment you will program in C. Before starting working you may consult:

- Sections 1 - 1.3, 1.5 - 1.5.3, and 1.8 in Tanenbaum's book.
- ANSI C for Programmers on UNIX Systems by T. Love, `http://www.inference.org.uk/teaching/comput/C/teaching_C/teaching_C.html`

  An introduction to C for programmers already knowing other languages, e.g. Java.
- C tutorial, `https://www.tutorialspoint.com/cprogramming/index.htm`

Depending on your background, you might find the following literature helpful:

- UNIX Tutorial for Beginners, `http://www.ee.surrey.ac.uk/Teaching/Unix/`

  Simple UNIX tutorial. Follow this if you are not comfortable with UNIX command line tools.
- Manual pages for system calls. On a Unix-system these are bound by the command `man 2 <topic>`, where `2` refers to section 2 of the manual pages covering system calls.

  The man pages may also be found various places online, e.g. at `https://www.man7.org/linux/man-pages/index.html`
- The C Programming Language, by Brian Kernighan and Dennis Ritchie.

  Classic book on C, still widely used. Optional.

During the work on the assignment you will use several tools. You might find the following manuals helpful:

- GNU Binutils `http://www.gnu.org/software/binutils/`

  This is a large collection of tools for manipulating different executable binary files. We are primarily using ld:

  ld: `http://sourceware.org/binutils/docs/ld/index.html`

  Please note that Binutils refers to the AMD64 architecture as x86_64.
- GCC `http://gcc.gnu.org/`

  GCC is a collection of compilers.

- GNU Make `http://www.gnu.org/software/make/manual/`

  make is a tool that can help you coordinate compilations of larger programs.

- A Make tutorial `https://www.tutorialspoint.com/makefile/index.htm`
- GDB `https://sourceware.org/gdb/current/onlinedocs/gdb/`

  A debugger allowing you to step through the code as you are executing and print out values.

- Bash `https://www.gnu.org/software/bash/manual/bash.html`

  A common shell (command interpreter).

4 Programming Environment

For all the assignments, you are supposed to be programming for a Unix environment. This may in particular be a Linux environment, but any Posix-compliant system (like MacOS) should be feasible.

On DTU Learn you will find instructions for establishing and testing a Unix environment on your platform.

5 Source code tree

For this assignment, you are provided with a source code file `assignment_1.tgz` which is organized into various `.c` and `.h` files and a Makefile for building your program.

The files your are given in this assignment are:

- `io.h`

  Contains the prototypes and definitions needed to read and write without using `<stdio.h>`. This file should not be changed.

- `io.c`

  Contains skeleton implementations of the functions defined in `io.h`. In Task 1, you must provide proper implementations of these functions. You are not allowed to use the standard i/o library `<stdio.h>`.

- `io_demo.c`

  This is a program that is used to demonstrate and test the result of Task 1. You need not modify this program.

- `main.c`

  This is a skeleton program for Task 2. You must extend this in order to implement the command interpreter as described.

  Note that you are not allowed to use `<stdio.h>` in this program either.

- `Makefile`

  A default makefile for running and testing your program. You may add new files to the file list(s), but otherwise it should not be changed.

- `test.sh`

  A simple bash script for testing the command interpreter. You should extend this with some extra tests.

6 Working on the assignment

6.1 Task 0: Testing the environment

Establish a Unix environment on your platform and test it as described on DTU Learn. You do not have to hand in a report section for this task.

6.2 Task 1: I/O Implementation

You should start by implementing the functions specified in `io.h`. These resemble functions in the standard i/o library `<stdio.h>`, but you are not allowed to use this library for the current assignment.

Instead you should use the system calls for reading and writing directly. These are found in the `<unistd.h>` library. In particular, you must use the basic `read` and `write` calls.

Study how these calls work and use them to implement the reading and writing functions. Be sure to handle the return values properly.

In order to test your implementation, you should run the program `io_demo.c`. This may be done by the command `make run-demo`.

Reporting for Task 1

In the report section for this task, you should address the following points:

1. Give a brief explanation of the various components of the makefile.
2. Explain briefly how you have implemented the i/o functions.
3. Assume that you enter the following sequence of characters to the demo program:

   ```
   abcqls
   ```

   before hitting the return button. What happens? How do you explain this?

   State an input string for which such behaviour would be very unfortunate.

   Suggest how the behaviour of the demo program could be made more safe.

6.3 Task 2: Command Interpreter

In this assignment you have to read commands from stdin and perform actions based on them. The commands operate upon an integer counter and a collection of integers. Commands are single characters with the following meanings.

- `'a'`: Add the current counter value to the collection, then increment the counter.
- `'b'`: Do nothing except increment the counter.
- `'c'`: Remove the most recently added element (if any) from the collection and increment the counter.

Anything else (including end-of-file): Stop processing commands, print the current collection elements in the order they were added, and exit. Initially the counter will be set to 0 and the collection will be empty, as shown in Figure 1.

```
Count:  0
Collection:
```

Figure 1: Initial state

As an example, the string `"abbabaq"` would produce the collection shown in Figure 2. Note that we use the `'q'` character to terminate command processing, but any character other than `'a'`, `'b'` and `'c'` would have the same effect.

```
Count:  6
Collection:  0 3 5
```

Figure 2: After `"abbabaq"`

The string `"abccbaabcq"` will produce the collection shown in Figure 3.

```
Count:  9
Collection:  5
```

Figure 3: After `"abccbaabcq"`

The commands must be read using your implementation of `io.h` from Task 1. You will need to create and maintain all necessary data structures yourself. How you implement the collection is entirely up to you but keep in mind that you do not know the size of the collection at execution time. Some form of dynamic memory allocation using `malloc` and `free` from `<stdlib.h>` will be required. This can be accomplished in a number of different ways one of which is to use a doubly linked list.

When the command processing is stopped, you are required to print out the contents of your collection to stdout. Again, this must be done using functions from `io.h`.

In particular, when your program encounters a character other than `'a'`, `'b'`, or `'c'`, you should print your collection, in entry order, and then terminate by returning from `main`. The format for printing the collection is as a comma delimited list of integers. The list must be ended with a semicolon mark (`';'`) and a new line character. The output of the three examples above would thus be, repectively:

```
;
0,3,5;
5;
```

Reporting for Task 2

In the report section for this task, you should address the following points:

1. Explain briefly how you have implemented the collection and illustrate this with one or more figures.
2. Explain what happens when `make test` is run.
3. Add at least two more test cases to the test script `test.sh` and describe these.

7 Hand-in and reporting

On Assignment 1 on DTU Learn, you must hand in the following:

- A file named `group_`*`NN`*`_assign_1.tgz` containing all your solution files for tasks 1 and 2 as well as a proper `Makefile`. In particular, it must be possible to build your executables using `make` without further arguments. As suggested by the suffix, the format of the file must be a gzipped tar-archive.
- A file named `group_`*`NN`*`_assign_1.pdf` with a report for the assignment. See below.

In the above file names, *NN* is your (two-digit) group number, e.g. `group_05_assign_1.tgz` .

7.1 Report requirements and rules

The report may be written in Danish or English. It must be a pdf file in A4 format but otherwise the layout is free.

The report should have a front page with the following information:

- **Who:** Group no. + name and study no. of each participant
- **What:** Assignment name and no.
- **Where:** University, department, course number and course name
- **When:** Date of hand-in

In the report, each task should be addressed in order and the reporting points properly answered.

There is no page limit, but the report should be kept concise and to the point. You may use code snippets to illustrate your text.

References to used material and solutions may be given as footnotes or by a reference list. Any use of generative AI must be declared: What dit you use it for and how?

By handing in a report with all your names, all members of the group implicitly confirm that all work has been done by the group itself and that each member has made a significant contribution to the work.

Also note:

- Any collaboration with other groups on smaller parts of the assignments must be declared and clearly identified. Collaboration on major parts is not acceptable.
- Any undeclared use or adaption of others' work is *strictly prohibited*

Good Luck!