Operating Systems, Spring 2017
Instructions for Databar Exercise 4:
Cache Performance and System Calls

September 21, 2017

1 Introduction

In this databar exercise, you will explore cache performance and how program behavior affect cache performance. You will also start moving towards the operating system matter by working at the operating system interface.

You will report on this exercise as part of the first report. You will there write one and a half page about the design of your program and your experiences and findings. You will also have to attach your source code. Furthermore, you will have to address the questions asked in this exercise. NB: You will write more in the first report than the outcome of this exercise. The full set of items to address in the first report will be available on September 25th.

Read through this document in entirely before starting working on the exercises!

2 Learning objectives

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

- You can explain in your own words the following terms: cache.
- You can draw a figure showing the memory hierarchy of a modern computer with the processor, cache levels, memory and storage.
- You can demonstrate with pseudo code how the memory access pattern of a program can greatly influence cache performance.
- 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 toolchain used in system programming and how the components interact.

3 Reports and rules

DTU has a zero tolerance policy on cheating and plagiarism. This also extends to the reports and indeed all your work. For example, to copy text passages or source code from someone else without clearly and properly citing your source is considered plagiarism. See the study hand book for further detail.

4 Reference material

Before carrying out the exercise read section 1.7 in Dan Negrut, "Primer: Elements of Processor Architecture. The Hardware/Software Interplay". You can find a link on DTU Inside. If you find section 1.7 difficult, you should study all of chapter 1 up to and including section 1.7.

You also need to study the following sections in Tanenbaum&Bos: introduction to chapter 1, 1.3, 1.5 and 1.6.

5 Observing cache performance

Download `cachetest.c` from DTU Inside into the VirtualBox machine.

Study the code. What does it do? What does it print?

Compile and run the code by going to the directory where you downloaded `cachetest.c` and type:

```
gcc -O0 cachetest.c
```

```
./a.out
```

Why do you see the numbers you see? Relate the numbers to the memory hierarchy. Use the output to figure out the memory hierarchy of your laptop. For example, how many and how large caches does your machine have and why? You may need to run the program multiple times if you see odd results.

How do you need to write your programs to suit the memory hierarchy?

Check online how many, and how large, caches your processor have. Do your results match the online data?

5.1 Improving performance

Rewrite `cachetest.c` so that the `x` array is accessed the same number of times as before but in a way which reduce the average memory access time.

5.2 reporting

You report your work in the first report with one page. You need to answer all questions above.

6 Linux system calls

We will now return to our program we wrote last week. We were then free to use the standard C library. However, this week we will program directly towards the Linux system call interface.

The system call interface sits below the standard C library and is relatively basic in nature. This means that it is a little bit more difficult to work with.

On DTU Inside, you will find a tar ball with functions that interact directly with the Linux read and write system calls and you must use those in this assignment. Beyond these you are allowed to use the `malloc` and `free` standard C library functions. You are not allowed to use any other library functions.

We provide:

- io.c

  Contains the two functions `syscall_read` and `syscall_write` that wrap the Linux read and write system calls. It is worth the time to understand how these work.

- io.h

  Contains the prototypes and declarations for the read and write system call functions.

NOTE: The write system call takes a pointer to an array of ASCII characters, or chars, to print. This means that if you want to print something other than an array of chars you must first convert it. For example, you must implement a method for printing integers yourself.

Rewrite your code from last week to now not rely on the standard C library as outlined above.

6.1 reporting

You report your work in the first report with half a page. You describe your work, your solution and your experiences. You deliver also your source code.

7 License