Assignment 7: Cats! And x86...

Due by: May 4, 2018 before 10:00 pm
Collaboration: None
Grading: Packaging 10%, Style 10% (where applicable), Design 10% (where applicable), Performance 20% (where applicable), Correctness/Functionality 50% (where applicable)

Overview

The ninth and **final** assignment is **again** mostly about hacking 32-bit x86 (aka IA-32) assembly code using the standard gcc/gas toolchain under Linux. (Note: you **cannot** use Intel syntax; you **must** use AT&T syntax!)

Hint: If you find yourself wondering how to use a C function or a system call, you can use the `man` command to look up information about either. For example, to find out how the `putchar` function from the C standard library works, use `man 3 putchar`; to find out how the `read` system call works, use `man 2 read` and search for register conventions and the system call number.

Problem 1: It's a cat Jim... (40%)

The essence of the `cat` command in UNIX is that it copies characters read from standard input (usually the keyboard) to standard output (usually the terminal window). For example, here is a short interaction with `cat`:

```
$ cat
I typed this!
I typed this!
And this... :-)
And this... :-)
$
```

The first `I typed this!` line was actually typed into the terminal by the user; the second line is what the `cat` command prints in return. Same for the third and fourth lines. After the fourth line was printed, the user typed **CTRL-D** to signal that the input was over, which ended the `cat` program. Here is a simple C implementation of the `cat` program:

```
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    int c;
    while ((c = getchar()) != EOF) {
        putchar(c);
    }
    return EXIT_SUCCESS;
}
```

If we compile this program using `gcc cat.c -o cat -O0` we see that it works just like the "real" cat we studied above:

```
$ ./cat
I typed this!
I typed this!
And this... :-)
And this... :-)
$
```

Your task for this assignment is as follows: First use the `-save-temps` option for `gcc` to get the assembly source code `cat.s` generated by the C compiler for the `cat.c` program.

Then copy the source code to a new file `mcat.s` and "clean it up" as best as you can by **removing** all superfluous assembler directives and **commenting** the assembly program carefully to explain which parts of the C program ended up in which lines of the assembly program and how it all works. Do **not** rewrite the assembly program beyond these two steps and make sure that your cleaned-up `mcat.s` can be assembled into a working `mcat` program! The command to build `mcat` should be as follows:

```
$ gcc -o mcat mcat.s
```

(Of course if you're using a 64-bit system an `-m32` may be assumed as well.) Finally, check how fast your `mcat` program is when copying a larger file: use the command

```
time ./mcat </usr/share/dict/words >/dev/null
```

a few times and pay attention to the `real` time it displays on average. You'll need this information again below!

Problem 2: ...but not as we know it! (60%)

Now we'll leave the safety of the C standard library behind. Your **final** task is to take the `mcat.s` program from the previous problem and **rewrite it** to use **neither** the C library **nor** the C startup code! In other words, you'll have to rewrite `mcat.s` into a new version `qcat.s` that uses **only** the `read` and `write` **system calls**, nothing else. It should be possible to build your `qcat.s` as follows:

```
$ gcc -nostdlib -o qcat qcat.s
```

(Again, maybe a `-m32` is required.) Note that `read` and `write` work **very differently** from `getchar` and `putchar`, you can find the details in the man pages. For example you'll need to provide a **buffer** for `read` to read characters into from standard input; that buffer is then used by `write` as the data to write back to standard output. **Make sure you understand the values returned from `read` and `write`, otherwise you will never be able to get this program done!**

Finally make sure that your new `qcat` program is **at least as fast (or faster)** than the `mcat` program from Problem 1. If you first version of `qcat` is too slow, be sure to explain in your `README` how you made it faster.

Good luck!

Deliverables

Please turn in a gzip compressed tarball of your assignment; the filename should be HW7_<your_name>.tar.gz with name replaced by the first part of the email address you used to register on Piazza (so I would use HW7_kmemon1.tar.gz).

Include a README file that briefly explains what your programs do and contains any other notes you want us to check out before grading.

Grading

For reference, here is a short explanation of the grading criteria; some of the criteria don't apply to all problems, and not all of the criteria are used on all assignments. **Packaging** refers to the proper organization of the stuff you hand in, following the guidelines for Deliverables above. **Style** refers to either to programming style if we are talking about a programming problem (things like consistent indentation, appropriate identifiers, useful comments, generally simple, clean, readable code), or to the clarity and readability of your solution for a written problem. **Design** refers to proper modularization and the proper choice of algorithms and data structures for a programming problem, and the proper choice of abstractions for all problems. **Functionality** refers to your programs being able to do what they should according to the specification given above; if the specification is ambiguous and you had to make a certain assumption, defend that assumption in your `README` file.

If your programs cannot be built you will get no points whatsoever for programming problems. If your programs fail miserably even once, i.e. terminate with an exception of any kind or dump core, we will take off 10%; do proper error checking and handling! Finally, make sure to include your name and email address in *every* file you turn in (well, if there *is* a way to include that information, otherwise you can leave it out)!

Acknowledgment

This assignment was originally designed by Peter Fröhlich.