ECE471: Embedded Systems – Homework 3

Linux Assembly and Code Density

Due: Friday, 24 September 2021, 11:00am EDT

1. Use your Raspberry-Pi to work on this project.

   - Download the code from:
     `http://web.eece.maine.edu/~vweaver/classes/ece471/ece471_hw3_code.tar.gz`
     and copy it to the Raspberry-Pi.
   - Uncompress/unpack it with the command `tar -xzvf ece471_hw3_code.tar.gz`
   - Change into the ece471_hw3_code directory `cd ece471_hw3_code`
   - Put all answers to questions into the included text `README` file. This will automatically be bundled up with your submission.

2. **Modify the `exit_asm.s` file to return the value 42. (1 point total)**

   (a) Modify `exit_asm.s`

   (b) Be sure any code comments are accurate!

   (c) Run `make` to generate an updated version

   (d) To test, run `./exit_asm` followed by `echo $?` which will show you the last program's exit status.

   (e) Some reminders about Linux GNU assembler (`as`) syntax:
   `.equ IDENTIFIER,value` sets a macro replacement, like
   `#define IDENTIFIER value` would in C
   You can use @ to specify a comment, like // in C
   You prefix a constant value with #
   (to move the number 5 into a register you would do `mov r0,#5`)

   (f) Reminders about the Linux kernel ARM syscall EABI:
   Arguments go in `r0` to `r6`
   System Call Number goes in `r7`
   Use `swi 0x0` to trigger a system call.

3. **Investigate the code density of `integer_print` (3 points total)**

   (a) Run `make` if you haven't already, which should generate the `integer_print` and `integer_print.thumb2` executables.

   (b) Look at the source code (the same file is used for both): `print_integer.c`. The algorithm used was described in class.

   (c) Find the size of the `print_integer()` function in the ARM32 executable and record it in the README

      i. To disassemble the code, run the command
      `objdump --disassemble-all ./integer_print | less`
      the | less at the end says to feed the output of the call to the program less, which lets you scroll backwards and see all the output.

ii. Find the `print_integer()` function. You could look manually, but the best way is to use the '/' slash character to start a search and then search for `print_integer`. The first location you find is the call from main() to the function, press '/' again to find the actual function.

iii. You should find something like the following:

```
0001041c <print_integer>:
   1041c:       e59fc064        ldr     ip, [pc, #100]  ; 10488 <print_integer+0x6
   10420:       e52de004        push    {lr}            ; (str lr, [sp, #-4]!)
   ...
```

The first column is the address in memory where this code lives, the next is the raw machine code for the instruction, the next is the decoded assembly language, and the last is the disassembler giving "helpful" hints about what's going on.

iv. You can see in this first case all the instructions are 32-bit hex values.

v. Calculate the length of this function and note it in the README. You can calculate length by scrolling down to where the function ends, (probably at something like `__libc_csu_init`) getting the memory address (the first column), then subtract from that the address of the start of the function.

(d) Now go back and do the same for the THUMB2 `print_integer.thumb2` executable too. (strip it too) Record the size in the README.

(e) Answer in README: Does the machine language look different for THUMB2 than ARM32? How?

4. **C vs Assembly code density: (2 points total)**
Put the answer to these in the README.

(a) Compare the size of the ARM32 `integer_print` executable and the THUMB2 `integer_print.thumb2` executables. (list both sizes)
You can get filesize with `ls -l` (that's a lowercase L) You will want to run the `strip` command on the executables first (i.e. `strip integer_print`) or your results might be unexpected.

(b) The `integer_print_static` file is also generated. This has the C library "statically linked" (included in the executable instead of dynamically loading the system copy of the C library at runtime). How does the size of the static version compare to the others?

(c) When you ran `make` it also compiled a pure assembly language version of the integer print code: `integer_print_asm`. What is the size of that file?

(d) Given these results, Which language might you use in space constrained system? Why?

(e) Which code to you think is easier to write, the C or assembly one?

5. **Use `gdb` to track down the source of a segfault. (2 points total)**

(a) When you ran `make` it should have built a program called `crash`

(b) Run that program. It should crash with a `Segmentation fault` error.

(c) Use the `gdb` debugger to find the source of the error.

(d) Run `gdb ./crash`

(e) When it comes up to a prompt, type `run` and press enter to run it until it crashes.

(f) It should tell you it crashed, then tell you what line of code caused the crash. Put the line that caused the crash in the `README`

(g) You can do various other things here, such as run `bt` to get a backtrace, which shows you which functions were called to get you to this error. You can run `info regis` to see the current register values.

(h) Run `disassem` to get a disassembly of the function causing problems. There should be an arrow pointing to the problem code. Cut and paste this line into the `README`

(i) In the end, what was the cause of the error in this program? (again, put this in the `README`

6. **Something cool: (1 point total)**
Modify `integer_print.extra.c` (which starts out the same as `integer_print.c` and do one of the following:

- Easy: modify so instead of printing a hard-coded value in `main()`, it reads a value from the prompt. To read from the console into an integer in C you would use something like `scanf("%d",&value);`
- Moderate: make the file print the integer in hexadecimal instead of decimal. (modify the existing algorithm, don't use `printf()`)
- Hard: modify `integer_print_asm.s` and make it print hexadecimal

7. **Linux Command Line Exploration (1 point total)**

Try out the `cal` program. This prints a calendar, by default the current month. You can also `cal 2016` or `cal 12 2016`. Beware not to do `cal 16` as that will give you year 16, not 2016.

(a) Run `cal 9 1752` Is there a bug here? Can you explain what is happening?

8. **Submitting your work**

- Run `make submit` which will create `hw3_submit.tar.gz` containing the various files. You can verify the contents with `tar -tzvf hw3_submit.tar.gz`
- e-mail the `hw3_submit.tar.gz` file to me by the homework deadline. Be sure to send the proper file!