PA1: Address Spaces and Resource Usage Monitoring

CMPSC 473, FALL 2017

Due: September 7, 2017, 11:59:59pm

1 Goals

We hope that PA0 will serve to refresh your understanding of (or make you learn in case you lack it): (i) logging into a Linux machine (`ssh, VPN, 2FA`) and using basic Linux shell and commands, (ii) compiling a C program (`gcc, make`), creating/linking against libraries, (iii) debugging (`gdb`), (iv) working with a code repository (`github`), and (v) using Linux man pages (the `man` command). All of these are good programming and experimental analysis practices that we would like you to follow throughout this course (and beyond it).

2 Getting Started

After accepting the invitational link to the Github Classroom @PSU473, you will be given access to your own private repository and another repository named "PA0" containing all the files needed for completing PA0. When you open this repository, you will find 3 folders, named `prog1`, ..., `prog3`. These folders contain the files described in Section 3. On the web-page of this repository, you will find a link to download it. To download copy the link and type on the command line:

```
$ git clone <link_of_the_repository>
```

You will find additional information on using GitHub and other important documents uploaded in a separate document on canvas.

As mentioned in class, you may do the bulk of your work on any Linux (virtual) machine of your choosing. However, the results in your report must be carried out on CSE department's Linux-based teaching machines. These machines are named `cse-p208instxx.cse.p` (where `xx` is a machine number). The reason for asking you to report results on these machines is to have relative consistency/uniformity in your measurements.

3 Description of Tasks

1. **Debugging refresher:** The program `prog1.c` calls a recursive function which has a local and a dynamically allocated variable. this program will crash due a bug we

have introduced into it. Use the `Makefile` that we have provided to compile the program. Execute it. The program will exit with an error printed on the console. You are to compile the program with 32 bit and 64 bit options and carry out the following tasks separately for each:

   (a) Observe and report the differences in the following for the 32 bit and 64 bit executables: (i) size of compiled code, (ii) size of code during run time, (iii) size of linked libraries.

   (b) Use `gdb` to find the program statement that caused the error. See some tips on `gdb` in the Appendix if needed.

   (c) Explain the cause of this error.

2. **More debugging:** Consider the program `prog2.c`. It calls a recursive function which has a local and a dynamically allocated variable. Like the last time, this program will crash due to a bug that we have introduced in it. Use the provided `Makefile` to compile the program. Again, create both a 32 bit and a 64 bit executable. For each of these, execute it. Upon executing, you will see an error on the console before the program terminates. You are to carry out the following tasks:

   (a) Observe and report the differences in the following for the 32 bit and 64 bit executables: (i) size of compiled code, (ii) size of code during run time, (iii) size of linked libraries.

   (b) Use `valgrind` to find the cause of the error including the program statement causing it. For this, simply run `valgrind prog2` on the command line.

   (c) How is this error different than the one for `prog1`?

3. **And some more:** The program `prog3.c` may seem to be error-free. But when executing under `valgrind`, you will see many errors. You are to perform the following tasks:

   (a) Describe the cause and nature of these errors. How would you fix them?

4 Submission and Grading

You will submit all your source files, Makefiles, and READMEs (the last to convey something non-trivial we may need to know to use your code). You are to submit a report answering all the questions posed above into your github repo.

Appendix

We offer some useful hints here.

- **Quick notes on `gdb`:**

1. To run a program `prog1` under `gdb`, simply execute
   `$ gdb prog1`
2. While running under `gdb`'s control, you can add breakpoints in the program to ease the debugging process. To add a breakpoint, type
   `$ break <linenumber>`
3. To run the code type
   `$ r`
4. To continue running the program after a breakpoint is hit, type
   `$ c`
5. To inspect the stack using `gdb`, type
   `$ backtrace` or
   `$ backtrace full` (to display contents of local variables)
6. To get information about individual frames, type
   `$ info frame <frame number>`
   E.g., if you want to see information about frame 5 (assuming your program has made 6 recursive function calls, since frame number starts from 0), then the command would look like
   `$ info frame 5`
7. To get size of a frame, subtract frame addresses of two consecutive frames.

- To compile the code using 32/64 bit options, add the `-m<architecture>` flag to the compile command in the Makefile. E.g., to compile with the 32 bit option:
  `$ gcc -g -m32 prog.c -o prog`

- To find the size of an executable (including its code vs. data segments), consider using the `size` command. Look at its man pages.

- To find the size of code during run time, type the following while the code is in execution:
  `$ pmap PID | grep "total"`
  To see memory allocated to each section of the process, type
  `$ pmap PID`