Friday, September 6 2019

15-441: Computer Networks

Recitation 2

P1 lead TAs: Mingran Yang, Alex Bainbridge

Agenda

1. Project 1 Checkpoint 2
2. git - version control and other things
3. Debugging Tips
4. Intro to gdb and valgrind + class exercices
5. Q&A

Project 1: Quick reminder

CP	Grade	Deadline
1	15% 25%	Sep 6 Sep 10
2	50% 75%	Sep 20 Sep 27
3	35%	Sep 27

15441

15641

15-441
15-641

Ckpt 2 is harder than Cpkt 1
Do not wait until the last day!

Checkpoint 2 *Due Sep 20/27, 2019*

What you need to do (writeup - pg 6, sec 4 + RFC 2616):

1. Respond to properly formatted HTTP HEAD and GET requests
2. Support five HTTP 1.1 error codes (default: 400)
    i. 400 (Bad Request)
    ii. 404 (Not Found)
    iii. 408 (Timeout)
    iv. 501 (Unsupported Method)
    v. 505 (Wrong HTTP version).
    vi. Optional: 418 (I'm a Teapot)
3. Handle concurrent connections using select()
4. Handle pipelined requests.

Git

- Git is a version-control system to track changes to your code
- It contains features that makes merging code from different authors easy (though we won't be discussing this for P1)
- Commands:
  - `git init` Initialize a new Git repository locally
  - `git clone <url>` Clone a remote repository located at *<url>*
  - `git add example.txt` Add changes to example.txt to staging area (i.e. pre-commit area)
  - `git add -A` Add changes to all files (including new files) in your repository to staging area
  - `git commit -m "Hello"` Place whatever is in the staging area into a commit
  - `git push origin master` Push most recent commit to online repository (such as one on Github)

Git: advanced

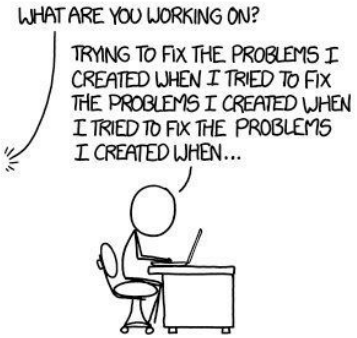
- Ignore certain files with a .gitignore (helps your git repo not become too large). Make sure you don't ignore important files (e.g. C files, header files, Makefile)

```
# Ignore foo.txt
foo.txt

# Ignore bar directory except bar/hello.c
bar/*
!/bar/hello.c

# Ignore all SQL files
*.sql
```

- Git tags allow you to mark a certain commit of the Git repo. A git tag does not know of any commits that occur after the tag is created. Tags are used for marking version releases (e.g. v2.0.1)

```
git tag -a checkpoint-<num>-m<message>[<commit hash>]
```

- Find out more about Git in [ref 5]



Debugging Tips

- Enable all warnings when compiling with gcc (**-Wall/ -Werror** … etc) [ref 1]
- Simple but most of the time efficient method: **print** statements (especially since your server is single threaded)
- Debugger for C: **gdb** [ref 2]
- Use **valgrind** to check for memory leaks [ref 4]
- Utilize and expand provided tests. NB: in ckpt2, you can start testing with your own browser

gdb: Introduction/Reminder

What is gdb?

GNU debugger (gdb) is a debugger for C. It uses a command line interface. It can help you get information about the following:

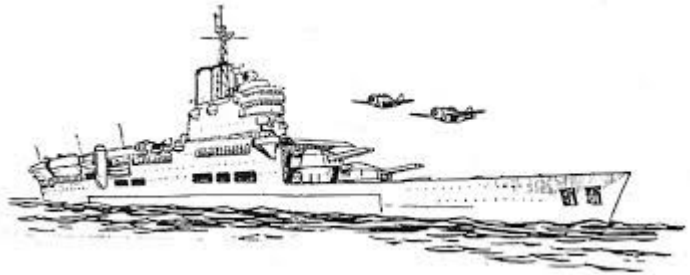
- If a SEG FAULT happened, then what statement did the program crash on?
- If an error occurs while executing a function, what line of the program contains the call to that function, and what are the arguments?
- What are the values of program variables at a particular point during execution of the program?
- What is the result of a particular expression in a program?

gdb: Getting started

- **Compiling**:
  To prepare your program for debugging with gdb, you must compile it with the `-g` flag
- **Entering and Quitting gdb:**
  gdb executable
  quit
- **Basic commands** (more in [ref 3]):
  - run — starts running the program in gdb
  - bt (backtrace) — figure out where the program was when it crashed (very useful for segfaults)
  - break — spot in your program where you would like to temporarily stop execution
    - break *function*
    - break *filename: linenumber*
  - step — continue to next source line
  - continue — continue to next breakpoint
  - p/d i — print variable *i* as a signed int
  - p/x i — print variable *i* in hex
  - p/d y[i] — print the ith element of *y* as a signed int

Battleship Exercise

- Download battleship.c from
  https://github.com/inespot/15441-recitation2
- This is a simplified version of a 1-player Battleship game
- Using gdb, find the 4 bugs in it



Battleship Bugs

1. Lines 10 & 11: **<=** should be **<**
2. Line 48: Need to populate board with hidden_water
3. Lines 43 & 46: Should be ***sizeof(int *)*** and ***sizeof(int)*** respectively
4. Line 53: Need to initialize ***i*** to ***0***

valgrind: Introduction/Reminder

What is valgrind?

Provides a number of debugging and profiling tools that help you make your programs faster and more correct.

The most popular of these tools is called *Memcheck*. It can detect many memory-related errors that are common in C programs and that can lead to crashes and unpredictable behaviour.

How to detect memory leaks with valgrind?

>> valgrind --tool=memcheck --leak-check=yes --show-reachable=yes myprog arg1 arg2

NB: Your program will run much slower than normal, and use a lot more memory.

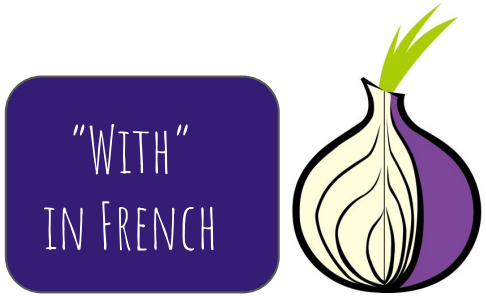
What a memory leak output will look like:

```
==19182== 40 bytes in 1 blocks are definitely lost in loss record 1 of 1
==19182==    at 0x1B8FF5CD: malloc (vg_replace_malloc.c:130)
==19182==    by 0x8048385: f (a.c:5)
==19182==    by 0x80483AB: main (a.c:11)
```

Vector exercice

- Download vector.c from
  https://github.com/inespot/15441-recitation2

- Use valgrind to find the 5 bugs



Vector Bugs

1. In ***main()***, you need to give ***x*** a value
2. In ***main()***, you need to free ***x*** after calling ***VectorSet()***
3. In ***main()***, you need to call ***VectorFree(x)*** before exiting
4. In ***VectorCreate()***, you need to set ***v***'s length before returning
5. In ***ResizeArray()***, you need to free ***arry*** before returning

Q & A

References

[1] **GCC Warning options**: https://gcc.gnu.org/onlinedocs/gcc/Warning-Options.html

[2] **gdb manual**: https://ftp.gnu.org/old-gnu/Manuals/gdb/html_node/gdb_toc.html

[3] **gdb Cheatsheet:** https://darkdust.net/files/GDB%20Cheat%20Sheet.pdf

[4] **Valgrind manual:** http://valgrind.org/docs/manual/manual.html

[5] **Git Handbook:** https://guides.github.com/introduction/git-handbook/