Valgrind and some Linux tools

CptS 223 - Spring 2017 - Aaron Crandall

Today’s Agenda

- Announcements
- Thing of the day
- Valgrind - a lifesaver (as I’m finding out)
- Other tools

Announcements

- Costco’s IS/IT group is recruiting on campus today:
  - 5pm in Sloan 169
  - IT group is about 1550 people in Seattle
  - Company has 225k total employees & $123 billion in annual revenue
  - Also runs an IT architecture competition & will present it today

Presentation by Costco Team

Thing of the day

Didn’t manage to find one that really grabbed me.

Quiz time!

bit.ly/2yfXuQa

Valgrind

- http://valgrind.org/
- Valgrind is an instrumentation framework for building dynamic analysis tools. There are Valgrind tools that can automatically detect many memory management and threading bugs, and profile your programs in detail.
- Memory leaks, pointer issues, oh my!
- Often part of testing frameworks for projects
- Can run on programs that you didn’t compile. :-)

Definitions to be aware of

- Memory leak vs. Memory error
  - Leak: allocated memory that is not freed during program run
    - Will be a problem as you write bigger and longer running programs
  - Error: BADNESS:
    - Reading uninitialized memory
    - Writing past the end of allocated memory (buffer over/underflow)
    - Accessing freed memory (Segfault)
    - Other
- Valgrind can help find both of these types of errors

Memory leaks

- Finding Memory Leaks
  - Memory leaks are among the most difficult bugs to detect because they don't cause any outward problems until you've run out of memory and your call to malloc suddenly fails.
  - Even one mistake can be costly if your program runs for long enough
- valgrind --tool=memcheck [program_name]

Memory loss happens when...

- If allocs != frees...

```
% valgrind --tool=memcheck program_name
...
=18515== malloc/free: in use at exit: 0 bytes in 0 blocks.
==18515== malloc/free: 1 allocs, 1 frees, 10 bytes allocated.
==18515== For a detailed leak analysis,  rerun with: --leak-check=yes
```

- So, you run with --leak-check=yes (and compile with -g on gcc/g++)

```
==2330== 100 bytes in 1 blocks are definitely lost in loss record 1 of 1
==2330==    at 0x1B900DD0: malloc (vg_replace_malloc.c:131)
==2330==    by 0x804840F: main (example1.c:5)
```

Invalid Pointer Use

- Where does this write to? →

```
char *x = malloc(10);
x[10] = 'a';
```

```
==9814==  Invalid write of size 1
==9814==    at 0x804841E: main (example2.c:6)
==9814==  Address 0x1BA3607A is 0 bytes after a block of size 10 alloc'd
==9814==    at 0x1B900DD0: malloc (vg_replace_malloc.c:131)
==9814==    by 0x804840F: main (example2.c:5)
```

Yes, that’s bad unless you’re actually *going* for a buffer overflow attack.

Detecting USE of Uninitialized Variables

What's the big deal with this?

```
==17943== Conditional jump or move depends on uninitialised value(s)
==17943==    at 0x804840A: main (example3.c:6)
```

Double Free - Invalid Free

- If you free the same memory twice, valgrind will grock it

Want to know more?

http://valgrind.org/docs/manual/quick-start.html

http://heeris.id.au/2016/valgrind-gdb/

https://web.stanford.edu/class/cs107/guide_valgrind.html

Linux Commands

wc
tail / head
grep
sed & awk
less
pstree

git

- This is a big topic and a wonderful tool once you learn the ins and outs of it!
- I use git often these days
- EECS has a git server for your use
  - Should be storing all programs there
  - *will* be using it later to collaborate on projects

A bit more about Git - Branching & Collaboration with other people

- Learning to work as a group in Git is invaluable
- It’s one of my big 6 from companies for grads:

1) C/C++ or a similar language
2) Scripting language of choice: Python/PERL/Ruby, etc
3) Communication skills: written & verbal
4) Tech tools (one or more of): Docker, Web services, mobile dev
5) Testing and test writing: why and how to
6) Version control for group dev: Git

General notion of how it works:
We don’t all just work on master!

- Clone
- Branch
- Dev
- Test
- Merge from origin
- Pull request
- Pull request review
- Merge to Origin

- Get the source (be clean)
- Make your sandbox to work
- Do your work
- Test it! Ensure it works well
- Fetch origin to get yourself up to date
- Make a pull request merge back to origin
- Others review your code
- Complete the merge

Stable vs. Unstable Branches

- Most projects use master to deploy the “stable” branch to users
- There’s normally one or more unstable branches for major feature dev
- Other branches for nightly builds are possible
- Code is promoted from nightly -> unstable -> master with reviews