CSE 333 – SECTION 2

Programming Tools - gdb, valgrind

Questions, Comments, Concerns

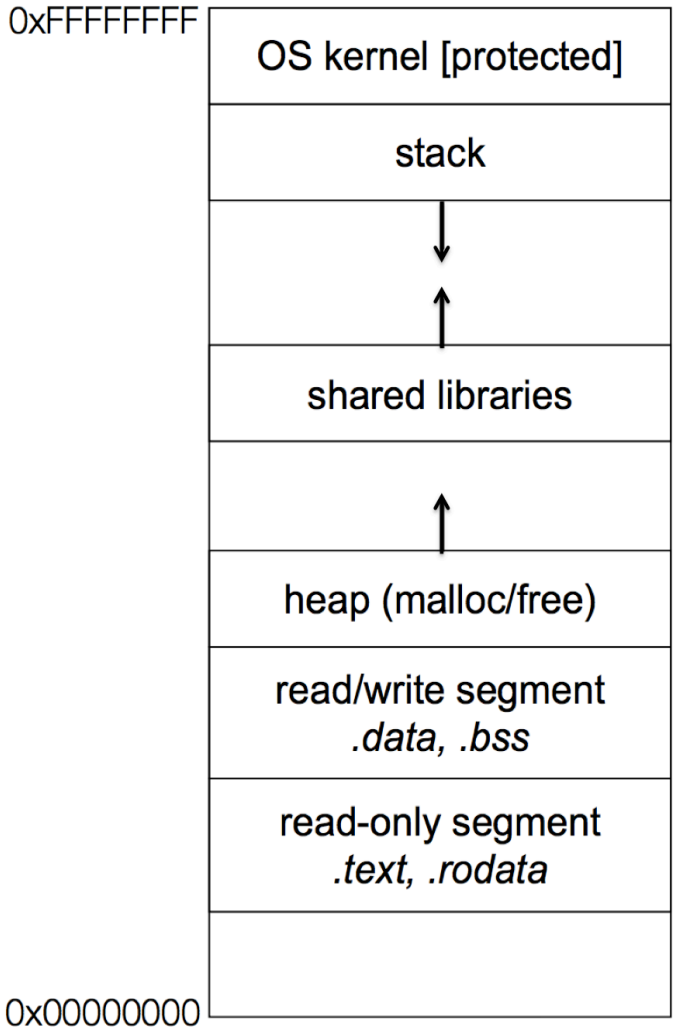
- Do you have any?
- Exercises going ok?
- Lectures make sense?
- Homework 1 – START EARLY!

Common Exercise Errors

- Comments
  - Program Comments – Author, copyright, problem description at the top
  - Function Comments – Near the prototype/declaration, not near the definition
- `clint` or `cpplint` errors
- Valgrind errors

Memory Management

- Heap
  - Large pool of unused memory
  - malloc() allocates chunks of this memory
  - free() deallocates memory and reclaims space
- Stack and stack frame
  - Stores temporary/local variables
  - Each function has its own stack frame
- Lifetime on heap vs. Lifetime on stack



Segmentation fault

- Causes of segmentation fault
  - Dereferencing uninitialized pointer
  - Null pointer
  - A previously freed pointer
  - Accessing end of an array
  - …
- gdb (GNU Debugger) is a debugging tool
- Very useful in tracking undefined behavior

reverse.c demo + code fix

Memory Errors

- Use of uninitialized memory
- Reading/writing memory after it has been freed – Dangling pointers
- Reading/writing to the end of malloc'd blocks
- Reading/writing to inappropriate areas on the stack
- Memory leaks where pointers to malloc'd blocks are lost
- Mismatched use of malloc/new/new[] vs free/delete/delete[]

Valgrind is your friend!!

Some buggy code

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

3.  //Returns an array containing [n, n+1, ... , m-1, m]. If n>m, then the
4.  //array returned is []. If an error occurs, NULL is returned.

5.  int *RangeArray(int n, int m) {
6.    int length = m-n+1;
7.
8.    //Heap allocate the array needed to return
9.
10.    int *array = (int*) malloc(sizeof(int)*length);
11.
12.    //Initialize the elements
13.    for(int i=0;i<=length; i++)
14.      array[i] = i+n;

15.    return array;
16.  }

17.  //Accepts two integers as arguments
18.  int main(int argc, char *argv[]) {
19.    if(argc != 3) return EXIT_FAILURE;
20.    int n = atoi(argv[1]), m = atoi(argv[2]);  //Parse cmd-line args
21.    int *nums = RangeArray(n,m);

22.    //Print the resulting array
23.    for(int i=0; i<= (m-n+1); i++)
24.      printf("%d", nums[i]);
25.    puts("");

26.    return EXIT_SUCCESS;
27.  }
```

Valgrind output

```
==22891== Command: ./warmup 1 10
==22891==
==22891== Invalid write of size 4
==22891== at 0x400616: RangeArray (warmup.c:14)
==22891== by 0x400683: main (warmup.c:22)
==22891== Address 0x51d2068 is 0 bytes after a block of size 40 alloc'd
==22891== at 0x4C2A93D: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==22891== by 0x4005EC: RangeArray (warmup.c:10)
==22891== by 0x400683: main (warmup.c:22)
==22891==
==22891== Invalid read of size 4
==22891== at 0x4006A5: main (warmup.c:26)
==22891== Address 0x51d2068 is 0 bytes after a block of size 40 alloc'd
==22891== at 0x4C2A93D: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==22891== by 0x4005EC: RangeArray (warmup.c:10)
==22891== by 0x400683: main (warmup.c:22)
==22891==
1 2 3 4 5 6 7 8 9 10 11
==22891==
==22891== HEAP SUMMARY:
==22891== in use at exit: 40 bytes in 1 blocks
==22891== total heap usage: 1 allocs, 0 frees, 40 bytes allocated
==22891==
==22891== 40 bytes in 1 blocks are definitely lost in loss record 1 of 1
==22891== at 0x4C2A93D: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==22891== by 0x4005EC: RangeArray (warmup.c:10)
==22891== by 0x400683: main (warmup.c:22)
==22891==
==22891== LEAK SUMMARY:
==22891== definitely lost: 40 bytes in 1 blocks
==22891== indirectly lost: 0 bytes in 0 blocks
==22891== possibly lost: 0 bytes in 0 blocks
==22891== still reachable: 0 bytes in 0 blocks
==22891== suppressed: 0 bytes in 0 blocks
==22891==
==22891== For counts of detected and suppressed errors, rerun with: -v
==22891== ERROR SUMMARY: 3 errors from 3 contexts (suppressed: 3 from 3)
```

Demo: leaky code

leaky.c demo + code fix

Section exercise

- Handouts.
- Work with a partner, if you wish.
- Look at the expandable vector code in imsobuggy.c.
- First, try to find all the bugs by inspection.
- Then try to use Valgrind on the same code.
- Code is located at http://courses.cs.washington.edu/courses/cse333/18au/sections/sec2-code/