

601.220 Intermediate Programming

Valgrind

Outline

- The `valgrind` tool for tracking memory usage

valgrind: a tool for tracking memory usage

- valgrind is an easy-to-use tool for finding memory usage mistakes
 - invalid memory accesses: e.g. array index out of bounds
 - memory leaks: failure to free dynamically-allocated memory

valgrind: a tool for tracking memory usage

- To use:
 - As when using gdb to debug, compile program with `-g`
 - Run program using valgrind:

```
valgrind --leak-check=full ./myFile <arg1> <arg2> ...
```

- See also <http://valgrind.org/docs/manual/QuickStart.html>

Example using valgrind

```
// example.c:
#include <stdio.h>

int main() {
    printf(" *** My program's output ***\n");
    return 0;
}

$ gcc -std=c99 -Wall -Wextra -pedantic example.c -o example -g
```

Running our example code using valgrind: no issues reported

```
$ valgrind --leak-check=full ./example
*** My program's output ***
==25901== Memcheck, a memory error detector
==25901== Copyright (C) 2002-2017, and GNU GPLd, by Julian Seward et al.
==25901== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==25901== Command: ./example
==25901==
==25901==
==25901== HEAP SUMMARY:
==25901==      in use at exit: 0 bytes in 0 blocks
==25901==    total heap usage: 1 allocs, 1 frees, 4,096 bytes allocated
==25901==
==25901== All heap blocks were freed -- no leaks are possible
==25901==
==25901== For counts of detected and suppressed errors, rerun with: -v
==25901== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Running code using valgrind

- Your program's output is interleaved with valgrind messages
- Kinds of issues that get flagged:
 - Invalid reads or writes: attempts to dereference pointers to memory that is not yours
 - Memory leaks: failing to deallocate a block of memory that you allocated. (See valgrind's HEAP SUMMARY section)

Example with memory usage issues

```
// stringCopy.c:
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
char * string_copy(const char *orig) {
    char *fresh = malloc(strlen(orig));
    assert(fresh != NULL); //check that malloc succeeded
    strcpy(fresh, orig);
    return fresh;
}
int main() {
    char *hello_copy = string_copy("hello");
    assert(hello_copy != NULL);
    printf("%s\n", hello_copy);
    return 0;
}
```

```
$ gcc -std=c99 -Wall -Wextra -pedantic stringCopy.c -o stringCopy -g
```


valgrind output

```
$ valgrind --leak-check=full ./stringCopy
hello
==21672== Memcheck, a memory error detector
==21672== Copyright (C) 2002-2017, and GNU GPLd, by Julian Seward et al.
==21672== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==21672== Command: ./stringCopy
==21672==
==21672== Invalid write of size 1
==21672==    at 0x4C2FC9D: strcpy (vg_replace_strmem.c:510)
==21672==    by 0x40065D: string_copy (stringCopy.c:8)
==21672==    by 0x400675: main (stringCopy.c:12)
==21672== Address 0x51ef045 is 0 bytes after a block of size 5 allocd
==21672==    at 0x4C2CB6B: malloc (vg_replace_malloc.c:299)
==21672==    by 0x400626: string_copy (stringCopy.c:6)
==21672==    by 0x400675: main (stringCopy.c:12)
==21672==
```

valgrind output, continued

```
==21672== Invalid read of size 1
==21672==    at 0x4C2FBD4: __strlen_sse2 (vg_replace_strmem.c:460)
==21672==    by 0x4EA7D81: puts (in /usr/lib64/libc-2.26.so)
==21672==    by 0x4006A5: main (stringCopy.c:14)
==21672== Address 0x51ef045 is 0 bytes after a block of size 5 allocd
==21672==    at 0x4C2CB6B: malloc (vg_replace_malloc.c:299)
==21672==    by 0x400626: string_copy (stringCopy.c:6)
==21672==    by 0x400675: main (stringCopy.c:12)
==21672==
```

valgrind output, continued once more

```
==21672== HEAP SUMMARY:
==21672==      in use at exit: 5 bytes in 1 blocks
==21672==    total heap usage: 2 allocs, 1 frees, 4,101 bytes allocated
==21672==
==21672== 5 bytes in 1 blocks are definitely lost in loss record 1 of 1
==21672==      at 0x4C2CB6B: malloc (vg_replace_malloc.c:299)
==21672==      by 0x400626: string_copy (stringCopy.c:6)
==21672==      by 0x400675: main (stringCopy.c:12)
==21672==
==21672== LEAK SUMMARY:
==21672==    definitely lost: 5 bytes in 1 blocks
==21672==    indirectly lost: 0 bytes in 0 blocks
==21672==    possibly lost: 0 bytes in 0 blocks
==21672==    still reachable: 0 bytes in 0 blocks
==21672==          suppressed: 0 bytes in 0 blocks
==21672==
==21672== For counts of detected and suppressed errors, rerun with: -v
==21672== ERROR SUMMARY: 3 errors from 3 contexts (suppressed: 0 from 0)
```

Example with memory usage issues

- So what was wrong?
 - An invalid write
 - An invalid read
 - A block of memory that wasn't freed
- But when run on `ugradx`, this program didn't crash! In fact, it seemed to work!
 - `valgrind` is really useful for finding problematic code!

Fixing those memory usage issues

```
// stringCopyFixed.c:
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
char * string_copy(const char *orig) {
    char *fresh = malloc(strlen(orig) + 1); //FIX 1: make space for '\0'
    assert(fresh != NULL);
    strcpy(fresh, orig);
    return fresh;
}
int main() {
    char *hello_copy = string_copy("hello");
    assert(hello_copy != NULL);
    printf("%s\n", hello_copy);
    free(hello_copy); //FIX 2: free the memory that function malloc-ed
    return 0;
}
```

\$ gcc -std=c99 -Wall -Wextra -pedantic stringCopyFixed.c -o stringCopyFixed -g

valgrind output for fixed version

```
$ valgrind --leak-check=full ./stringCopyFixed
hello
==33155== Memcheck, a memory error detector
==33155== Copyright (C) 2002-2017, and GNU GPLd, by Julian Seward et al.
==33155== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==33155== Command: ./stringCopyFixed
==33155==
==33155==
==33155== HEAP SUMMARY:
==33155==       in use at exit: 0 bytes in 0 blocks
==33155==     total heap usage: 2 allocs, 2 frees, 4,102 bytes allocated
==33155==
==33155== All heap blocks were freed -- no leaks are possible
==33155==
==33155== For counts of detected and suppressed errors, rerun with: -v
==33155== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Checkpoint Poll!

Which of the following three functions has issues with pointers?

```
int * fun1(void) {  
    int x= 10;  
    return (&x);  
}  
  
int * fun2(void) {  
    int * px;  
    *px= 10;  
    return px;  
}  
  
int * fun3(void) {  
    int *px;  
    px = (int *) malloc (sizeof(int));  
    *px= 10;  
    return px;  
}
```

A. fun1

B. fun2

C. fun1 and fun2

D. fun1, fun2 and fun3

E. fun1 and fun3