M7 – Memory Checkers

CS 136L F23 – LEC 9

Yiqing Irene Huang, Qianqiu Zhang

UNIVERSITY OF WATERLOO | FACULTY OF MATHEMATICS
DAVID R. CHERITON SCHOOL OF COMPUTER SCIENCE

Disclaimer

- The following slides were not presented page by page in class.
- They are my own study notes to share with students.
- In the lab session, we will cover key points, do small demos and give hints on commonly seen errors

Main Points

Use memory checkers to identify memory errors.

- Valgrind directly operates on the executable
  - Supports both clang and gcc compiled code, works better with gcc
  - memcheck tool

- AddressSanitizer injects instrumentation at compile time
  - Part of clang, gcc version 4.8 and up also supports it.

Memory Errors

1. Using uninitialized memory
2. Dereferencing/accessing a NULL pointer or an invalid address.
3. Buffer overflow
4. Using stack memory after function has returned
5. Memory Leak
6. Accessing dynamic memory beyond the range of memory allocated
7. Accessing dynamic memory that has already been deallocated
8. Trying to deallocate memory that is no-longer/not ours to deallocate

Lab Thresholds

Question	Description	# of Tests	Pass	Complete
Q1	Allocating a node	4	1	3
Q2	Deallocating a node	4	1	3
Q3	Printing a list	4	1	3
Q4	Computing list length	4	1	3

Valgrind

```
gcc -Wall -O0 -g <files-in-the-program-come-here>
valgrind -s --leak-check=full --track-origins=yes ./executable-name
```

AddressSanitizer

```
clang -Wall -O0 -g -fsanitize=address <files-in-the-program-come-here>
./executable-name
```

Bug Reporter

```
/u2/cs136l/pub/lab7/bug-reporter
```

How to name the executable?
What is a.out?

Introduction

- Valgrind

```
gcc -Wall -g -O0 test.c -o test
valgrind ./test
```

```c
// File test.c
#include <stdio.h>

int main(void) {
  printf("Hello World!\n");
  return 0;
}
```

- AddressSanitizer

```
clang -O1 -fsanitize=address -fno-omit-frame-pointer -Wall -g test.c -o test
./test
```

Which optimization level does the module recommend to use when using Address Sanitizer?

Uninitialized Memory

- The `-Wall` option is required to identify the error both for clang and gcc at compile time
- Clang: `-fsanitize=address` at compile time
- Valgrind will capture the memory error at runtime.
  - Use `-track-origins=yes` option for details
- AddressSantizer will not report runtime error
  - `-O0` gives consistent output
  - `-O1` gives random output

```c
// File test1.c
#include <stdio.h>

int main(void){
  int i;
  printf("%d",i);
  return 0;
}
```

Conditional Jump Over Uninitialized Values

- The `-Wall` option is required to identify the error for gcc
- clang cannot capture this error at compile time
- Valgrind will capture the memory error at runtime.
  - Use `-track-origins=yes` option for details
- AddressSantizer will not report runtime error
- MemorySanitizer will report runtime error

`-fsanitize=memory`

```
1 // File test2.c
2 int main(int argc, char **argv) {
3   int arr[2];
4   if (arr[argc != 1])
5     return 1;
6   else
7     return 0;
8 }
```

NULL Address

Invalid Write

```c
// File test3.c
#include <stdio.h>

int main(void){
  int *p = NULL;
  *p = 5;
  printf("%d\n",*p);
}
```

Segmentation fault both for clang and gcc

Invalid Read

```c
// File test4.c
#include <stdio.h>

int main(void){
  int *p = NULL;
  int x = *p;
  printf("%d\n",x);
}
```

Segmentation fault for gcc
Only SEGV with –O0 for clang

Buffer Overflow

```c
//File: buffer1.c
#include <stdio.h>

void print_element(int *arr, int index){
    printf("%d",arr[index]);
}
int main(void) {
  int a[10];
  print_element(a,10);
}
```

```c
#include <stdio.h>
int main(void){
  int a = 5;
  printf("%d\n",*(&a + 1));
}
```

- Valgrind cannot detect buffer overflow
- AddressSanitizer can detect it with `-O0` only

Stack Use After Run

- Valgrind cannot detect the error.
- AddressSanitizer can detect the error with special flags

```c
// File stack0.c
#include <stdlib.h>
#include <stdio.h>

char *get_hello(){
  char str[20] = "Hello World!";
  char *toRet = str;
  return toRet;
}
int main (){
  printf("%s\n",get_hello());
  return 0;
}
```

```
clang -fsanitize=address -Wall -g stack0.c -o stack0
./stack0
??}9EV
./stack0
?!?:V

ASAN_OPTIONS=detect_stack_use_after_return=1 ./stack0
=================================================================
==5169==ERROR: AddressSanitizer: stack-use-after-return or
--- output truncated
```

Stack Use After Run – Cont'd

```c
// File stack0.c
#include <stdlib.h>
#include <stdio.h>

char *get_hello(){
  char str[20] = "Hello World!";
  return &str;
}
int main (){
  printf("%s\n",get_hello());
  return 0;
}
```

```c
// File stack0.c
#include <stdlib.h>
#include <stdio.h>

char *get_hello(){
  char str[20] = "Hello World!";
  char *toRet = str;
  return toRet;
}
int main (){
  printf("%s\n",get_hello());
  return 0;
}
```

- The `-Wall` will detect it both for gcc and clang at compile time

```c
//file stack1.c
#include <stdio.h>

struct vec {
  int x;
  int y;
};

// get_vec(_x,_y) is a helper for creating a
//   vec object with the specified values
struct vec *get_vec(int _x, int _y){
    struct vec myVec;
    struct vec *p = &myVec;
    p->x = _x;
    p->y = _y;
    return p;
}

int main(void){
    struct vec *p1 = get_vec(1,2);
    struct vec *p2 = get_vec(5,10);

    printf("%d,%d\n", p1->x, p1->y);
    printf("%d,%d\n", p2->x, p2->y);

    return 0;
}
```

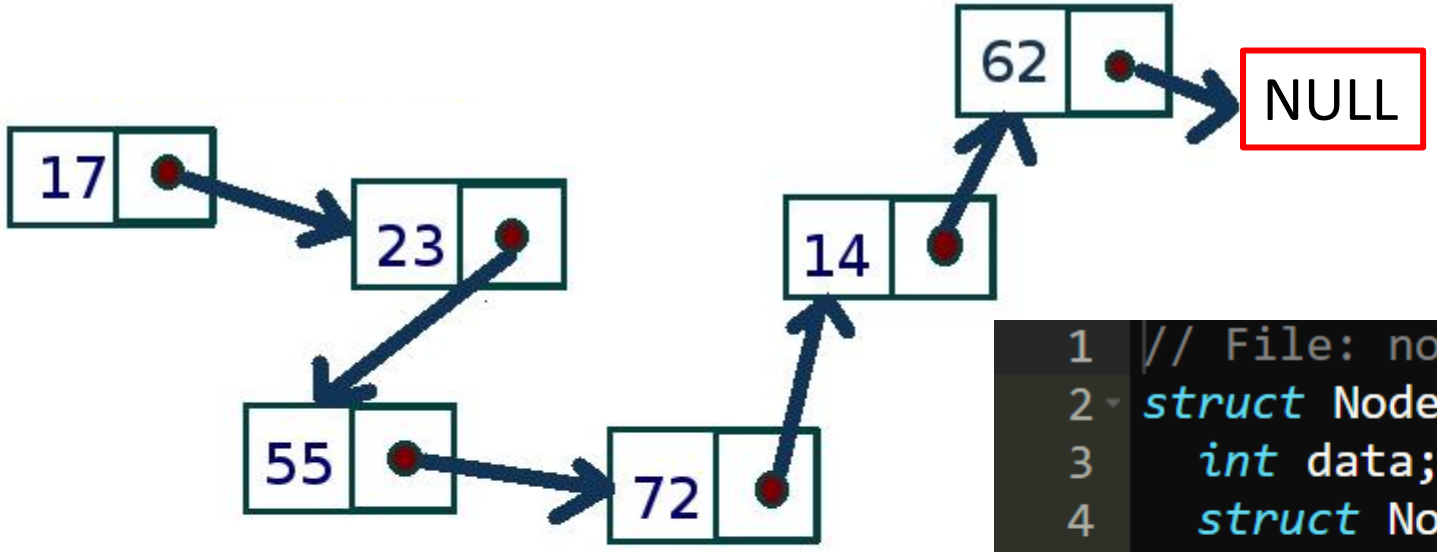
Stack Use After Run – Cont'd

- Both valgrind and AddressSantizer will catch the error

Heap Memory Errors

- Memory leak
  - malloc without the matching free()
- Incorrect use of malloced memory
  - Access memory that is not allocated. Allocate X bytes, access X+n (n>0) bytes
- Incorrect free
  - Free a pointer that is not returned by malloc
- Premature free
  - After free, access the memory

Linked Structure




```
1  // File: node.h
2  struct Node {
3    int data;
4    struct Node *next;
5  };
6
7  struct Node *list_creator(int length);
8  void list_printer(struct Node *list);
9
```

Acknowledgement

- Slides by courtesy of Carmen Bruni and Dave Tompkins
- Demo notes from Nomair Naeem
- Demo lectures by Carmen Bruni, Dave Tompkins, and Nomair Naeem

References

- CS 136L edX notes at https://online.cs.uwaterloo.ca/