

# Debugging

ECE453/CS447/SE465

Bob Zhang

March 11<sup>th</sup> and 12<sup>th</sup> 2010

Valgrind

# Introduction

- Valgrind is a programming tool for:
  - Memory debugging
    - Memory leaks and buffer overflows
    - Allocation and deallocation of dynamic memory
  - Memory leak detection
    - Unable to release memory it has acquired
  - `valgrind --leak-check=yes myprog arg1 arg2`
    - Memcheck (default tool)

# Locating Memory Leaks with Valgrind

- example1.c

1. #include <stdlib.h>

- 2.

3. int main()

4. {

5. char \*x = malloc(100);

6. return 0;

7. }

Problem: x is not freed

```
% valgrind --leak-check=yes example1
```

```
==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)
```

# Locating Invalid Pointer use with Valgrind

- example2.c

```
1. #include <stdlib.h>
2.
3. int main()
4. {
5.   char *x = malloc(10);
6.   x[10] = 'a';
7.   return 0;
8. }
```

**Problem:** Trying to access a location past the end of the array

```
% valgrind --leak-check=yes example2
==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)
```

# Detecting the use of Uninitialized Variables

- example3.c

```
1. #include <stdio.h>
2.
3. int main()
4. {
5.     int x;
6.     if(x == 0)
7.     {
8.         printf("X is zero");
9.     }
10. return 0;
11. }
```

**Problem: x is uninitialized**

```
% valgrind --leak-check=yes example3
```

```
==17943== Conditional jump or move depends on uninitialized value(s)
```

```
==17943== at 0x804840A: main (example3.c:6)
```

# What Valgrind Won't Detect?

- example4.c

1. #include <stdio.h>

- 2.

3. int static[5];

4. int main(void)

5. {

6. int stack[5];

7. static[5] = 0;

8. stack[5] = 0;

9. return 0;

10. }

Problem: Inability to detect bounds errors in the use of static or stack allocated data

# Resources

- Valgrind home page <http://valgrind.org/>
- Valgrind live debugging examples <http://www.youtube.com/watch?v=7xJuBqhlChE>
- Projects using Valgrind <http://valgrind.org/gallery/users.html>

# Debugging with Record Replay

# What is Debugging with Record Replay?

- Debug recordings of programs running in virtual machines
- Find, diagnose and fix bugs that are not easily reproduced
  - Non-deterministic bugs
  - Bugs that can only be reproduced with a complex environment
  - Memory corruption bugs

# Tools Needed for Debugging with Record Replay

- Microsoft Visual Studio 2005 or above
- VMware Workstation 6.5 or above
  - Windows XP Professional or above .iso

# Debugging with Record Reply Demo

- Introducing Record Reply <http://www.blip.tv/file/1051146/>
- Debugging with Record Replay <http://www.blip.tv/file/1051171/>
- Virtual Machine-Based Replay Debugging (gets techie at time 35:15 where they talk about the implementation of Record Replay)  
<http://www.youtube.com/watch?v=RvMlihjqlhY>

# Resources

- VMware Workstation <http://www.vmware.com/products/workstation/index.html>
- Replay debugging blog <http://www.replaydebugging.com/>
- Contact developer E Lewis <http://www.elewis.net/>

# Debugging Facilities in O'Caml

# The Debugger (ocamldebug)

- During program execution, a counter is incremented at each event encountered (*current time*)
- Using counter we can...
  - step 0
  - run/reverse
  - step/backstep
  - goto *time*

# Example

- `uncaught.ml`

```
let l = ref [];;
```

```
let add_address name address = l := (name, address) :: !l;;
```

```
let find_address name = List.assoc name !l;;
```

```
add_address "JOHN" "Beckhamcourt";;
```

```
print_string(find_address "JNONH"); print_newline();;
```

```
Fatal error: exception Not_found
```

# Finding the Cause of the Exception

```
ocamlc -g uncaught.ml  
ocamldebug a.out
```

```
(ocd) r  
Loading program... done.  
Time : 12  
Program end.  
Uncaught exception: Not_found  
(ocd)
```

```
(ocd) b  
Time      : 11 - pc : 15500 - module List  
143      [] -> <|b|>raise Not_found
```

```
(ocd) bt  
#0 Pc : 15500 List char 3562  
#1 Pc : 19128 Uncaught char 221
```

The function that calls it is in module Uncaught, file uncaught.ml char 221:

- `print_string (find_address "JNOHN"); print_newline ();;`

# Another Approach

```
(ocd) break @Uncaught 9
```

```
(ocd) g 0
```

```
Time : 0
```

```
Beginning of program.
```

```
(ocd) r
```

```
Time : 6 - pc : 19112 - module Uncaught
```

```
Breakpoint : 1
```

```
9 add "JOHN" "Beckhamcourt"<|a|>;
```

```
(ocd) s
```

```
Time : 7 - pc : 19012 - module Uncaught
```

```
5 let find_address name = <|b|>List.assoc name !;;
```

```
(ocd) p name
```

```
name : string = "JNOHN"
```

```
(ocd) p !!
```

```
$1 : (string * string) list = ["JOHN", "Beckhamcourt"]
```

# Resources

- O'Caml Language <http://caml.inria.fr/>
- O'Caml plug-in for Eclipse <http://www.algo-prog.info/ocaide/>
- Chapter 16 The debugger (ocamldebug)  
<http://caml.inria.fr/pub/docs/manual-ocaml/manual030.html>