CS 501
Debugging

Nate Nystrom
September 19, 1999

The Debugging Process

1. Bug reports should contain a test case, output, and the version number of the software.

2. Reproduce the bug using the same version the customer used.

3. Find the root cause of the bug.

4. Check if the bug still occurs with the latest version. If it does, fix it.

5. If it doesn't, make sure the bug is not just masked by other changes to the software.

6. Add test cases used to reproduce the bug to the regression test suite.

7. Keep records!

Debuggers

- Debuggers are tools which examine the state of a running program.
- Common debuggers: `adb`, `dbx`, `gdb`, `kdb`, `wdb`, `xdb`
- Microsoft Visual Studio has a built-in debugger
- This talk will focus on `gdb` because it's widely available.

GDB

- Command-line oriented, but will soon have a text-based UI. Text UI is available on HP-UX now (HP did the work).

- `xxgdb` is an X-based front-end.

- There is a TCL/TK based front-end also: `tkgdb?`

- To run, type: `gdb` *executable-file*

- Can attach to a running process with:
  `gdb` *executable-file process-id*

- Can examine a core file with:
  `gdb` *executable-file core-file*

How to use GDB

```
$ gdb himom
[...]
This GDB was configured as "i386-redhat-linux"...
(gdb) r
Starting program: /home/nystrom/501/gdb/himom/himom
Hi, Mom!

Program exited normally.
(gdb) list
1       #include <stdio.h>
2
3       int main()
4       {
5           printf("Hi, Mom!\n");
6           return 0;
7       }
```

Breakpoints

Can stop the program at any line or in any function with *b*.

```
$ gdb himom
(gdb) b main
Breakpoint 1 at 0x80483d3: file himom.c, line 5.
(gdb) r
Starting program: /home/nystrom/501/gdb/himom/himom

Breakpoint 1, main () at himom.c:5
5           printf("Hi, Mom!\n");
(gdb) b himom.c:6
Breakpoint 2 at 0x80483e0: file himom.c, line 6.
(gdb) c
Continuing.
Hi, Mom!

Breakpoint 2, main () at himom.c:6
6           return 0;
(gdb) fini
Run tell exit from #0  main () at himom.c:6
[...]
```

Examining the state of the program

- Printing variables:

  `p i`

  `p a[i]`

  `p obj->count`

- Stack trace:

```
(gdb) bt
#0  main () at himom.c:5
#1  0x4002fcb3 in __libc_start_main [...]
```

- Changing the stack frame:

  `up`, `down`, `frame` n

- Disassembly:

  `disas` *function_name*

Other useful commands

- Modifying variables in the program you're debugging:

  `set list = list->next`

- Calling functions in the program you're debugging:

  `set print_control_flow_graph()`

  `p get_list_size(list)`

- Stepping:

  `next` - next line in program, jumping over calls

  `step` - next line in program, entering calls

- Conditional breakpoints:

  `cond 1 strcmp(filename, "input7.dat") == 0`

- Help: `help`

Unexpected Aborts (in Unix)

If your software core dumps. Check what signal caused the core dump to get an indication of what the problem is.

- SIGBUS - likely a dereference of a null pointer
- SIGSEGV - likely a dereference of a bogus pointer, or sometimes an invalid write to code space, or a bad branch to data space
- SIGFPE - division by zero

Fire up the debugger to find out why it crashed:

`gdb` *executable-file core-file*

Blame the compiler

Sometimes software crashes in optimized code, but not in debug code.

The tendency is to blame the optimizer and de-optimize the function or the file where the bug occurred.

Most often the problem is in the code and is just exposed by the optimizer, typically an unititialized global variable.

Of course, sometimes it really is an optimizer bug. In that case, please submit a bug to the compiler vendor with a nice short test program.