TECHNISCHE UNIVERSITÄT DRESDEN

Center for Information Services and High Performance Computing (ZIH)

Memory Debugging

HRSK Practical on Debugging, 03.04.2009

Zellescher Weg 12

Willers-Bau A106

Tel. +49 351 - 463 - 31945

Matthias Lieber (matthias.lieber@tu-dresden.de)

Tobias Hilbrich (tobias.hilbrich@zih.tu-dresden.de)

ZIH
Center for Information Services & High Performance Computing

Content

- Introduction
- Tools
  - Valgrind
  - DUMA
- Demo
- Exercise

Memory Debugging

- Segmentation faults sometimes happen far behind the incorrect code
- Memory debuggers help to find the real cause of memory bugs
- Detect memory management bugs
  - Access non-allocated memory
  - Access memory out off allocated bounds
  - Memory leaks – when pointers to allocated areas get lost forever
  - etc.
- Different approaches
  - **Valgrind**: Simulation of the program run in a virtual machine which accurately observes memory operations
  - Libraries like **ElectricFence**, **DMalloc**, and **DUMA**: Replace memory management functions through own versions

Memory Debugging with Valgrind

- Valgrind detects:
  - Use of uninitialized memory
  - Access free'd memory
  - Access memory out off allocated bounds
  - Access inappropriate areas on the stack
  - Memory leaks
  - Mismatched use of malloc and free (C, Fortran), new and delete (C++)
  - Wrong use of memcpy() and related functions
- Available on Deimos via
  - **module load valgrind**
- Simply run program under Valgrind:
  - **valgrind ./myprog**
- More Information: http://www.valgrind.org

Memory Debugging with Valgrind

```
mlieber@deimos103:~/myprog
mlieber@deimos103:~/myprog> module load valgrind
Valgrind Memory Debugger version 3.3.1 loaded
mlieber@deimos103:~/myprog> gcc -g myprog.c -o myprog
mlieber@deimos103:~/myprog> valgrind ./myprog
==27183== Memcheck, a memory error detector.
==27183== Copyright (C) 2002-2007, and GNU GPL'd, by Julian Seward et al.
==27183== Using LibVEX rev 1854, a library for dynamic binary translation.
==27183== Copyright (C) 2004-2007, and GNU GPL'd, by OpenWorks LLP.
==27183== Using valgrind-3.3.1, a dynamic binary instrumentation framework.
==27183== Copyright (C) 2000-2007, and GNU GPL'd, by Julian Seward et al.
==27183== For more details, rerun with: -v
==27183==
==27183== Use of uninitialised value of size 8
==27183==    at 0x400615: read_file (myprog.c:13)
==27183==    by 0x400688: main (myprog.c:29)
==27183==
==27183== Use of uninitialised value of size 8
==27183==    at 0x400636: read_file (myprog.c:17)
==27183==    by 0x400688: main (myprog.c:29)
```

Memory Debugging with DUMA

- DUMA detects:
  - Access memory out off allocated bounds
  - Using a free'd memory
  - Mismatched use of malloc (C), new and delete (C++)
- Triggers a crash when an error occurs – use debugger for analysis
- Available on Mars and Deimos via
  - **module load duma**
- Run program under DUMA
  - **duma ./myprog**
- Or link the DUMA library
  - **icc –g –O0 –o myprog myprog.c –lduma**
  - **gdb ./myprog**

Demo with Valgrind and DUMA

- exampleC-06.c
  - Run without memory debugger
  - Run with Valgrind
  - Run with DUMA
  - Run with DUMA + GDB

Exercise: Find the Bug!

- Login on Deimos to run Valgrind
  - Same procedure as for Mars, replace the host name through **deimos.hrsk.tu-dresden.de**
- DUMA works on Deimos and Mars
- Run **exampleC-06.c** or **exampleF-06.F90**
  - Run without memory debugger
  - Run with Valgrind (Deimos)
  - Run with DUMA
  - Run with DUMA + GDB
- Run **exampleC-07.c** or **exampleF-07.F90**
  - Run without a memory debugger
  - Run with Valgrind (Deimos)
    - Hint: try Valgrind option **--leak-check=full**

Exercise: Solution

- **exampleC-06.c** or **exampleF-06.F90**
  - Program does not crash
  - C version produces wrong results
  - Fortran version seems to be correct
  - Wrong upper bound in the boundary update loop in **heatBoundary**
  - Program may crash with other compilers or on other machines

- **exampleC-07.c** or **exampleF-07.F90**
  - Memory Leak: **thetanew** is allocated in **heatTimestep**, but not deallocated
    - Simple Solution: deallocate thetanew at the end of heatTimestep
  - Valgrind reports the line where thetanew was allocated and the number lost blocks (641 for C) and lost bytes (2,594,768 for C)

Optional Demo + Exercise: Memory Debugging with DDT

- Run **exampleC-06.c** or **exampleF-06.F90** under DDT:
  - Click Advanced... in the session start-up window
  - Select *Enable Memory Debugging* in the session window
  - Click *Settings...*
    - Select *Medium* level
    - Select *Add guard pages*
  - Run