Valgrind

Jeroen Engelberts - jeroene@sara.nl
Consultant Supercomputing

Layout Presentation

- What is Valgrind?
- Why Should I Use Valgrind
- Difference Valgrind And GDB
- Some Examples
- Questions & Answers

What is Valgrind – for us?

- Valgrind is a “program-execution monitoring framework”
- Valgrind comes with many tools, the tool you will use most often is the *memcheck* tool
- Memcheck will detect and report the following types of memory errors:
  - Use of uninitialized memory
  - Reading/writing to memory after it has been freed
  - Reading/writing off the end of malloc’d blocks
  - Reading/writing inappropriate areas on the stack
  - Overlapping src and dest pointers in memcpy() and related functions.
  - others

What is Valgrind – historically?

- Valgrind is the gate to Valhalla, realm for those fallen in combat
- This realm is ruled by Odin, or Wodan
- Why is this important? On a day like today, Wednesday?

- In English (and Dutch), we still honour four Germanic Gods
- Tiwaz, Wodan, Thor and Frigg have their own day of the week
- I think it's apt to discuss Wodan's portal on his “own” day :-)

Why should I use Valgrind?

- Valgrind will tell you about tough to find bugs early!
- Valgrind is very thorough.
  - You may be tempted to think that Valgrind is too picky, since your program may seem to work even when valgrind complains. It is the author's experience that fixing *ALL* Valgrind complaints will save you time in the long run.
- Is there a downside to using Valgrind?
  - Valgrind is kind-of like a virtual x86 interpreter. So your program will run 10 to 30 times slower than normal.

Difference Valgrind and GDB

- What is the difference between Valgrind and GDB?
  - GDB is a debugger, Valgrind is a memory checker (among other things).
  - Valgrind won't let you step interactively through a program.
  - GDB doesn't check for use of uninitialized values, or over/underflowing dynamic memory.
  - Both GDB and Valgrind will show you the line number of a segfault
  - Valgrind often shows the cause of a segfault, too
  - Often bugs are found and fixed faster using Valgrind than GDB

Valgrind produces too much output

- Often Valgrind produces so much output that the important sections disappear off the top of the terminal. Here are three suggestions to cope:
  - Use Shift-PageUp and Shift-PageDown to scroll up and down.
  - If you are using xterm, enable the scrollbar by clicking Ctrl-Middle mouse button and select “Enable Scrollbar”. Use the middle mouse button to select the ScrollBar.
  - Re-direct the output to a file, and view the file with **less**

    valgrind ... > out 2>&1

Example 1

- A small program with two problems:

```
#include <stdlib.h>

void f(void)
{
        int* x = malloc(10 * sizeof(int));
        x[10] = 0;          // problem 1: heap block overrun
}                           // problem 2: memory leak -- x not freed

int main(void)
{
        f();
        return 0;
}
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"a.c" 13L, 228C                                                1,1           All
```

Example 1

- Compilation, no errors – Valgrind finds both problems:

```
valgrind-tut $ gcc -O0 -g -Wall -o a.x a.c
valgrind-tut $ valgrind ./a.x
==13754== Memcheck, a memory error detector
==13754== Copyright (C) 2002-2011, and GNU GPL'd, by Julian Seward et al.
==13754== Using Valgrind-3.7.0 and LibVEX; rerun with -h for copyright info
==13754== Command: ./a.x
==13754==
==13754== Invalid write of size 4
==13754==    at 0x80483FF: f (a.c:6)
==13754==    by 0x8048411: main (a.c:11)
==13754==  Address 0x41f1050 is 0 bytes after a block of size 40 alloc'd
==13754==    at 0x402BE68: malloc (in /usr/lib/valgrind/vgpreload_memcheck-x86-linux.so)
==13754==    by 0x80483F5: f (a.c:5)
==13754==    by 0x8048411: main (a.c:11)
==13754==
==13754==
==13754== HEAP SUMMARY:
==13754==     in use at exit: 40 bytes in 1 blocks
==13754==   total heap usage: 1 allocs, 0 frees, 40 bytes allocated
==13754==
==13754== LEAK SUMMARY:
==13754==    definitely lost: 40 bytes in 1 blocks
==13754==    indirectly lost: 0 bytes in 0 blocks
==13754==      possibly lost: 0 bytes in 0 blocks
==13754==    still reachable: 0 bytes in 0 blocks
==13754==         suppressed: 0 bytes in 0 blocks
==13754== Rerun with --leak-check=full to see details of leaked memory
==13754==
==13754== For counts of detected and suppressed errors, rerun with: -v
==13754== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
valgrind-tut $
```

Example 1

- OK, let's fix both problems:

```
#include <stdlib.h>

void f(void)
{
        int* x = malloc(11 * sizeof(int));
        x[10] = 0;        // solution 1: zero based 10th element = 11
        free(x);
}                         // solution 2: array x is freed upon leaving f

int main(void)
{
        f();
        return 0;
}
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"a.c" 14L, 255C                                                1,1           All
```

Example 1

- Problems solved! – Valgrind is happy :-)

```
valgrind-tut $ gcc -O0 -g -Wall -o a.x a.c
valgrind-tut $ valgrind ./a.x
==13826== Memcheck, a memory error detector
==13826== Copyright (C) 2002-2011, and GNU GPL'd, by Julian Seward et al.
==13826== Using Valgrind-3.7.0 and LibVEX; rerun with -h for copyright info
==13826== Command: ./a.x
==13826==
==13826==
==13826== HEAP SUMMARY:
==13826==     in use at exit: 0 bytes in 0 blocks
==13826==   total heap usage: 1 allocs, 1 frees, 44 bytes allocated
==13826==
==13826== All heap blocks were freed -- no leaks are possible
==13826==
==13826== For counts of detected and suppressed errors, rerun with: -v
==13826== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
valgrind-tut $
```

Example 2

- Again a program with two problems. Do you see them?

```
#include <stdlib.h>

int main()
{
        int **ia;
        int ix1, ix2;
        ia = malloc( 20 * sizeof(int*) );
        for( ix1 = 0; ix1 < 20; ix1++ )
                ia[ix1] = malloc( 10 * sizeof(int) );
        for( ix1 = 0; ix1 < 10; ix1++ )
                for( ix2 = 0; ix2 < 20; ix2++ )
                        ia[ix1][ix2] = 10;
        return 0;
}
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"index.c" 14L, 270C                                           1,1           All
```

Example 2

- Our compiler doesn't, but Valgrind sees them both:

```
valgrind-tut $ gcc -O0 -g -Wall -o index.x index.c
valgrind-tut $ valgrind ./index.x
==14063== Memcheck, a memory error detector
==14063== Copyright (C) 2002-2011, and GNU GPL'd, by Julian Seward et al.
==14063== Using Valgrind-3.7.0 and LibVEX; rerun with -h for copyright info
==14063== Command: ./index.x
==14063==
==14063== Invalid write of size 4
==14063==    at 0x8048459: main (index.c:12)
==14063==  Address 0x41f10d0 is 0 bytes after a block of size 40 alloc'd
==14063==    at 0x402BE68: malloc (in /usr/lib/valgrind/vgpreload_memcheck-x86-linux.so)
==14063==    by 0x8048420: main (index.c:9)
==14063==
==14063==
==14063== HEAP SUMMARY:
==14063==     in use at exit: 880 bytes in 21 blocks
==14063==   total heap usage: 21 allocs, 0 frees, 880 bytes allocated
==14063==
==14063== LEAK SUMMARY:
==14063==    definitely lost: 80 bytes in 1 blocks
==14063==    indirectly lost: 800 bytes in 20 blocks
==14063==      possibly lost: 0 bytes in 0 blocks
==14063==    still reachable: 0 bytes in 0 blocks
==14063==         suppressed: 0 bytes in 0 blocks
==14063== Rerun with --leak-check=full to see details of leaked memory
==14063==
==14063== For counts of detected and suppressed errors, rerun with: -v
==14063== ERROR SUMMARY: 100 errors from 1 contexts (suppressed: 0 from 0)
valgrind-tut $
```

Example 2

- Plugged in both solutions – will this do?

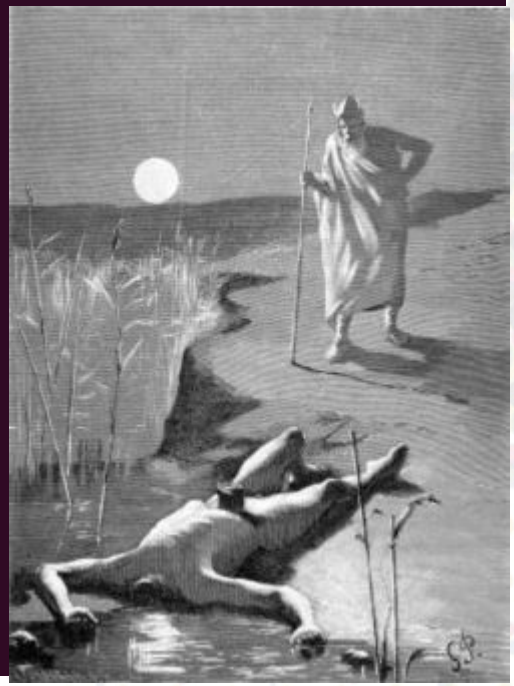
```
#include <stdlib.h>

int main()
{
        int **ia;
        int ix1, ix2;
        ia = malloc( 20 * sizeof(int*) );
        for( ix1 = 0; ix1 < 20; ix1++ )
                ia[ix1] = malloc( 10 * sizeof(int) );
        for( ix1 = 0; ix1 < 20; ix1++ )          // solution 1: loop lengths ix1 and ix2 swapped
                for( ix2 = 0; ix2 < 10; ix2++ )
                        ia[ix1][ix2] = 10;
        free(ia);                                // solution 2: free the memory allocated for ia
        return 0;
}
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"index.c" 15L, 391C                                                    1,1           All
```

Example 2

- The severed head of Mímir tells us there is still something wrong...

```
valgrind-tut $ gcc -O0 -g -Wall -o index.x index.c
valgrind-tut $ valgrind ./index.x
==14170== Memcheck, a memory error detector
==14170== Copyright (C) 2002-2011, and GNU GPL'd, by Julian Seward et al.
==14170== Using Valgrind-3.7.0 and LibVEX; rerun with -h for copyright info
==14170== Command: ./index.x
==14170==
==14170==
==14170== HEAP SUMMARY:
==14170==     in use at exit: 800 bytes in 20 blocks
==14170==   total heap usage: 21 allocs, 1 frees, 880 bytes allocated
==14170==
==14170== LEAK SUMMARY:
==14170==    definitely lost: 800 bytes in 20 blocks
==14170==    indirectly lost: 0 bytes in 0 blocks
==14170==      possibly lost: 0 bytes in 0 blocks
==14170==    still reachable: 0 bytes in 0 blocks
==14170==         suppressed: 0 bytes in 0 blocks
==14170== Rerun with --leak-check=full to see details of leaked memory
==14170==
==14170== For counts of detected and suppressed errors, rerun with: -v
==14170== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
valgrind-tut $
```



Example 2

- OK, let's fix the third problem:

```c
#include <stdlib.h>

int main()
{
        int **ia;
        int ix1, ix2;
        ia = malloc( 20 * sizeof(int*) );
        for( ix1 = 0; ix1 < 20; ix1++ )
                ia[ix1] = malloc( 10 * sizeof(int) );
        for( ix1 = 0; ix1 < 20; ix1++ )          // solution 1: loop lengths ix1 and ix2 swapped
                for( ix2 = 0; ix2 < 10; ix2++ )
                        ia[ix1][ix2] = 10;
        for( ix1 = 0; ix1 < 20; ix1++ )
                free(ia[ix1]);                   // solution 3: free all pointers indexed by ix1 - thanks Mimir
        free(ia);                                // solution 2: free the memory allocated for ia
        return 0;
}
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"index.c" 17L, 522C                                                    1,1           All
```

Example 2

- Our program is free of leaks and indices out of bounds!

```
valgrind-tut $ gcc -O0 -g -Wall -o index.x index.c
valgrind-tut $ valgrind ./index.x
==14308== Memcheck, a memory error detector
==14308== Copyright (C) 2002-2011, and GNU GPL'd, by Julian Seward et al.
==14308== Using Valgrind-3.7.0 and LibVEX; rerun with -h for copyright info
==14308== Command: ./index.x
==14308==
==14308==
==14308== HEAP SUMMARY:
==14308==     in use at exit: 0 bytes in 0 blocks
==14308==   total heap usage: 21 allocs, 21 frees, 880 bytes allocated
==14308==
==14308== All heap blocks were freed -- no leaks are possible
==14308==
==14308== For counts of detected and suppressed errors, rerun with: -v
==14308== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
valgrind-tut $
```

Questions & Answers