Tools supporting programming in C/C++ for Linux

- Tomasz Powchowicz

- Sanitizer
  - To sanitise = to make something completely clean and free from bugs.
  - Compiler instrumentation
    all read/write
    function enter/exit
    read zones around stack and global variables
  - Run-time library
    malloc replacement
    read zones around every allocation
    intercepts all synchronisations

- AddressSanitizer (aka ASan)
  - gcc(4.8), clang(3.1)
  - Detects:
    Dangling pointer (use-after-free)
    Out-of-bounds accesses to heap, stack and globals
    Use-after-return
    Double-free, invalid free
  - Program slowdown, memory overhead
    cpu 2x
    memory 3x-5x

- AddressSanitizer heap-use-after-free
  - Hard to detect without auxiliary tools.
  - Dangling pointer dereference scenario.
  - Preventive action: clear ptr to nullptr after delete.
  - Could be prevented by using std::shared_ptr or std::unique_ptr.
  - $ g++ -fsanitize=address

```
int main(int argc, char** argv) {
  int* array = new int[100];
  delete [] array;
  return array[0];
}
```

• AddressSanitizer heap-use-after-free

```
==20704==ERROR: AddressSanitizer: heap-use-after-free on address 0x61400000fe40 at pc 0x0000004007fd
READ of size 4 at 0x61400000fe40 thread T0
    #0 0x4007fc in main (/home/adbuni/test+0x4007fc)
    #1 0x7f04d58aba3f in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x20a3f)
    #2 0x4006c8 in _start (/home/adbuni/test+0x4006c8)

0x61400000fe40 is located 0 bytes inside of 400-byte region [0x61400000fe40,0x61400000ffd0)
freed by thread T0 here:
    #0 0x7f04d5cef02a in operator delete[](void*) (/usr/lib/x86_64-linux-gnu/libasan.so.2+0x9a02a)
    #1 0x4007c5 in main (/home/adbuni/test+0x4007c5)
    #2 0x7f04d58aba3f in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x20a3f)

previously allocated by thread T0 here:
    #0 0x7f04d5ceea32 in operator new[](unsigned long) (/usr/lib/x86_64-linux-gnu/libasan...)
    #1 0x4007ae in main (/home/adbuni/test+0x4007ae)

Shadow bytes around the buggy address:
  0x0c287fff9fb0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
=>0x0c287fff9fc0: fa fa fa fa fa fa fa fa[fd]fd fd fd fd fd fd fd
  0x0c287fff9fd0: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd
  0x0c287fff9fe0: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd
  0x0c287fff9ff0: fd fd fd fd fd fd fd fd fd fd fa fa fa fa fa fa
  0x0c287fffa000: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  Heap left redzone:       fa
  Freed heap region:       fd
```

- AddressSanitizer heap-buffer-overflow
  - Consequences: data corruption or unexpected behaviour by any process.
  - Faster reproduction: execution bit (separate data from the code)
  - Faster reproduction: randomization

```
int main(int argc, char** argv) {
  int* array = new int[100];
  array[0] = 0;
  int res = array[100];
  delete [] array;
  return res;
}
```

• AddressSanitizer heap-buffer-overflow

```
==1913==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x61400000ffd4 at pc 0x000000400894
READ of size 4 at 0x61400000ffd4 thread T0
    #0 0x400893 in main (/home/adbuni/test+0x400893)
    #1 0x7f8a83bc0a3f in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x20a3f)
    #2 0x400718 in _start (/home/adbuni/test+0x400718)

0x61400000ffd4 is located 4 bytes to the right of 400-byte region [0x61400000fe40,0x61400000ffd0)
allocated by thread T0 here:
    #0 0x7f8a84003a32 in operator new[](unsigned long) (/usr/lib/x86_64-linux-gnu/libasan.so...)
    #1 0x4007fe in main (/home/adbuni/test+0x4007fe)
    #2 0x7f8a83bc0a3f in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x20a3f)

Shadow bytes around the buggy address:
  0x0c287fff9fb0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c287fff9fc0: fa fa fa fa fa fa fa fa 00 00 00 00 00 00 00 00
  0x0c287fff9fd0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c287fff9fe0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x0c287fff9ff0: 00 00 00 00 00 00 00 00 00 00[fa]fa fa fa fa fa
  0x0c287fffa000: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Heap left redzone:     fa
  Heap right redzone:    fb
```

- AddressSanitizer stack-buffer-overflow
  - Write access could damage data on other frames.
  - A program could be executed in an unpredicted way.
  - Security vulnerability: stack smashing
  - Stack canaries

```
int main(int argc, char** argv) {
  int stack_array[100];
  stack_array[1] = 0;
  return stack_array[100];
}
```

• AddressSanitizer stack-buffer-overflow

```
==16470==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x7fff58c12d94 at pc 0x000000400916
READ of size 4 at 0x7fff58c12d94 thread T0
    #0 0x400915 in main (/home/adbuni/test+0x400915)
    #1 0x7f33f817aa3f in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x20a3f)
    #2 0x400758 in _start (/home/adbuni/test+0x400758)

Address 0x7fff58c12d94 is located in stack of thread T0 at offset 436 in frame
    #0 0x400835 in main (/home/adbuni/test+0x400835)

  This frame has 1 object(s):
    [32, 432) 'stack_array' <== Memory access at offset 436 overflows this variable

Shadow bytes around the buggy address:
  0x10006b17a560: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x10006b17a570: 00 00 00 00 00 00 00 00 00 00 00 00 f1 f1 f1 f1
  0x10006b17a580: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x10006b17a590: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x10006b17a5a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x10006b17a5b0: 00 00[f4]f4 f3 f3 f3 f3 00 00 00 00 00 00 00 00
  0x10006b17a5c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:              00
  Stack left redzone:       f1
  Stack right redzone:      f3
  Stack partial redzone:    f4
```

- AddressSanitizer global-buffer-overflow
  - Could damage data around the global buffer .
  - Due to program error vulnerable data could be accessed.

```
int global_array[100] = {-1};
int main(int argc, char** argv) {
  return global_array[100];
}
```

• AddressSanitizer global-buffer-overflow

```
==5124==ERROR: AddressSanitizer: global-buffer-overflow on address 0x000000601214 at pc 0x00000040080c
READ of size 4 at 0x000000601214 thread T0
    #0 0x40080b in main (/home/adbuni/test+0x40080b)
    #1 0x7fc875207a3f in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x20a3f)
    #2 0x4006e8 in _start (/home/adbuni/test+0x4006e8)

0x000000601214 is located 4 bytes to the right of global variable 'global_array' defined in
'./global_buffer_overflow.cpp:1:5' (0x601080) of size 400

Shadow bytes around the buggy address:
  0x0000800b81f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0000800b8200: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0000800b8210: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0000800b8220: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0000800b8230: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x0000800b8240: 00 00[f9]f9 f9 f9 f9 f9 00 00 00 00 00 00 00 00
  0x0000800b8250: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0000800b8260: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0000800b8270: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0000800b8280: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0000800b8290: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Global redzone:        f9
```

- UndefinedBehaviorSanitizer
  - A program could work well on one architecture, but on the other one the result could be undefined.

    Detects:
  - Using misaligned or null pointer
  - Signed integer overflow
  - Overflow conversion to, from, or between floating-point types

```
int main(int argc, char** argv) {
  unsigned int x = 1;
  x = x << 33;
  return x;
}
```

- UndefinedBehaviorSanitizer

```
$ g++ -fsanitize=undefined -o test ./undefined_behaviour.cpp

./undefined_behaviour.cpp: In function ‘int main(int, char**)’:

./undefined_behaviour.cpp:4:12: warning: left shift count >=
width of type [-Wshift-count-overflow]
   x = x << 33;
            ^


$ ./test
undefined_behaviour.cpp:4:9: runtime error: shift exponent 33 is
too large for 32-bit type 'unsigned int'
```

- ThreadSanitizer

  - clang -O1 -g -fsanitize=thread -fno-omit-frame-pointer

  - Detects:
    data races

  - Program slowdown, memory overhead
    cpu 5x-15x
    memory 5x-10x

- ThreadSanitizer

```
1:  #include <pthread.h>
2:  int Global;
3:  void* Thread1(void* x) {
4:   Global = 42;
5:   return x;
6:  }
7:  int main() {
8:   pthread_t t;
9:    pthread_create(&t, NULL, Thread1, NULL);
10:   Global = 43;
11:  pthread_join(t, NULL);
12:  return Global;
13: }
```

• ThreadSanitizer

```
$ clang++ -fsanitize=thread -g -O1 -o test ./race.cpp

WARNING: ThreadSanitizer: data race (pid=22670)
  Write of size 4 at 0x0000014b48a0 by thread T1:
    #0 Thread1(void*) /home/adbuni/./race.cpp:4 (test+0x0000004b6047)

  Previous write of size 4 at 0x0000014b48a0 by main thread:
    #0 main /home/adbuni/./race.cpp:10 (test+0x0000004b608e)

  Location is global '<null>' of size 0 at 0x000000000000 (test+0x0000014b48a0)

  Thread T1 (tid=22677, running) created by main thread at:
    #0 pthread_create <null> (test+0x000000453cb1)
    #1 main /home/adbuni/./race.cpp:9 (test+0x0000004b6084)

SUMMARY: ThreadSanitizer: data race /home/adbuni/./race.cpp:4 Thread1(void*)
```

- gdb core

```
$ ulimit -c unlimited
$ compile with “-g” option

$ ./crash
Aborted (core dumped)

$ gdb  ./crash
(gdb) core ./core
(gdb) bt
#0  0x00007f18314e3cc9 in __GI_raise (sig=sig@entry=6)
at ../nptl/sysdeps/unix/sysv/linux/raise.c:56
#1  0x00007f18314e70d8 in __GI_abort () at abort.c:89
#2  0x0000000000400536 in main ()
```

```
#include <stdlib.h>
int main() {
  abort();
  return 0;
}
```

- gdb examine callstack

```
#0  0x00007f18314e3cc9 in __GI_raise (sig=sig@entry=6)
at ../nptl/sysdeps/unix/sysv/linux/raise.c:56
#1  0x00007f18314e70d8 in __GI_abort () at abort.c:89
#2  0x0000000000400536 in main ()
(gdb) f 0
(gdb) info args
sig = 6
(gdb) info locals
resultvar = 0
pid = 1816
selftid = 1816
```

Signal	Value	Action	Comment
SIGHUP	1	Term	Hangup detected on controlling terminal or death of controlling process
SIGINT	2	Term	Interrupt from keyboard
SIGQUIT	3	Core	Quit from keyboard
SIGILL	4	Core	Illegal Instruction
SIGABRT	6	Core	Abort signal from abort(3)
SIGFPE	8	Core	Floating point exception
SIGKILL	9	Term	Kill signal
SIGSEGV	11	Core	Invalid memory reference

- gdb examine memory

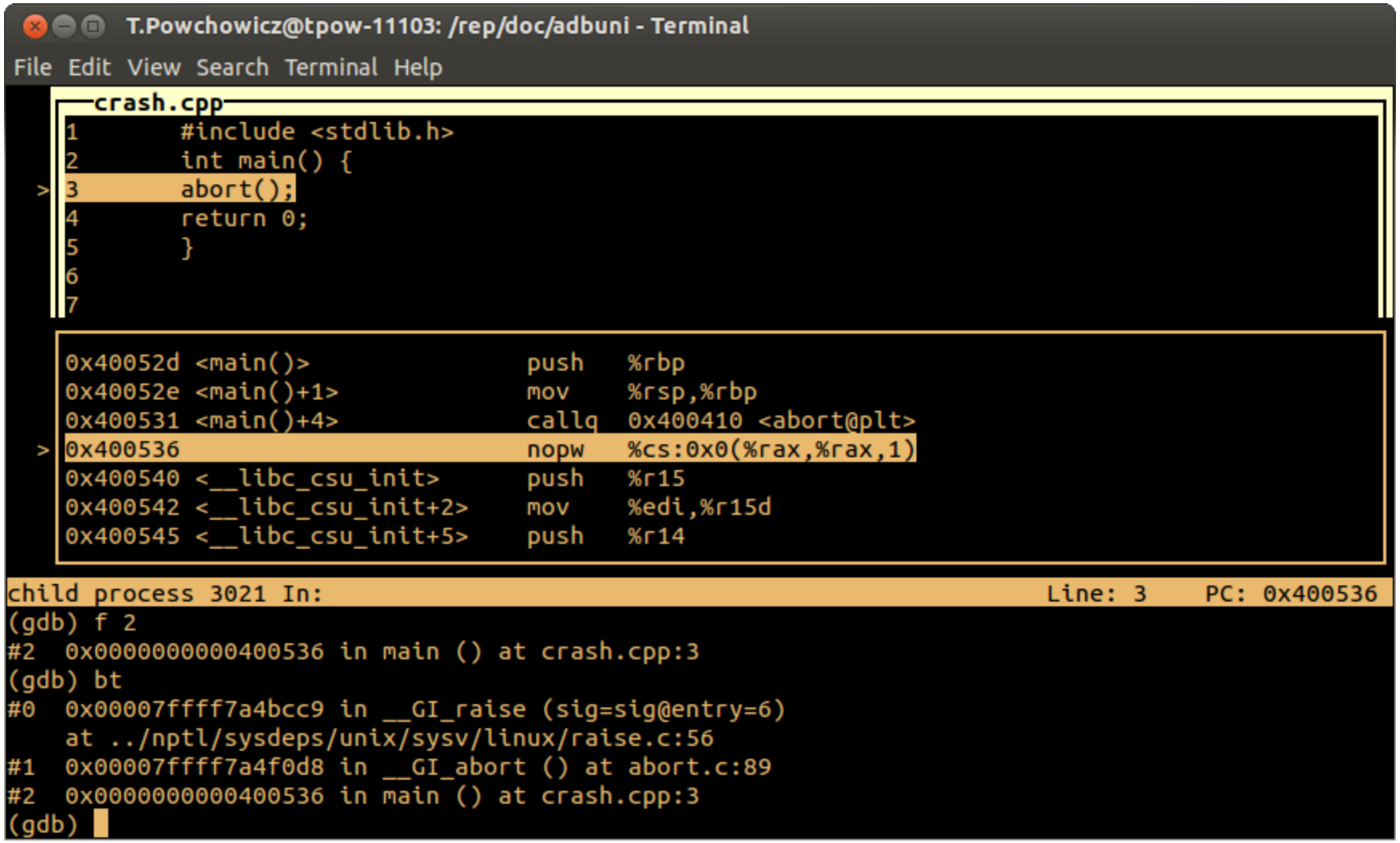
```
(gdb) f 1
#1  0x00007f18314e70d8 in __GI_abort () at abort.c:89
(gdb) p act
$5 = {
  __sigaction_handler = {
    sa_handler = 0x0,
    sa_sigaction = 0x0
  }, ...
}
(gdb) p &act
$6 = (struct sigaction *) 0x7ffc7dc8a630
(gdb) x/32x &act
0x7ffc7dc8a630: 0x00000000 0x00000000 0x00000000 0x00000000
0x7ffc7dc8a640: 0x7dc8a6e0 0x00007ffc 0x31881557 0x00007f18
```

- gdb TUI

```
ctrl + x a

ctrl + x 1
ctrl + x 2
ctrl + x o

<PgUp>
<PgDn>
<Up>
<Down>
<Left>
<Right>
```

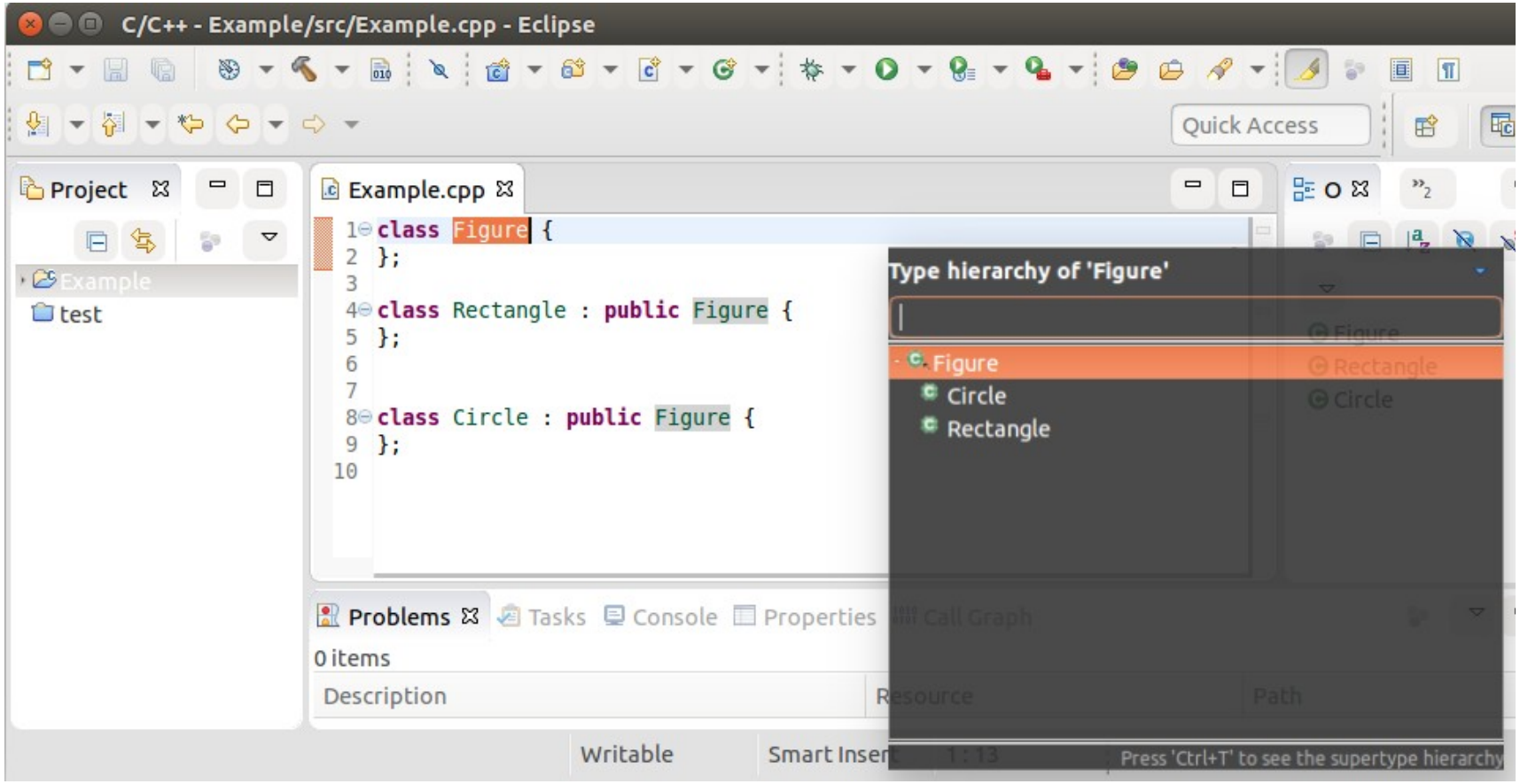


- GLIBCXX_DEBUG

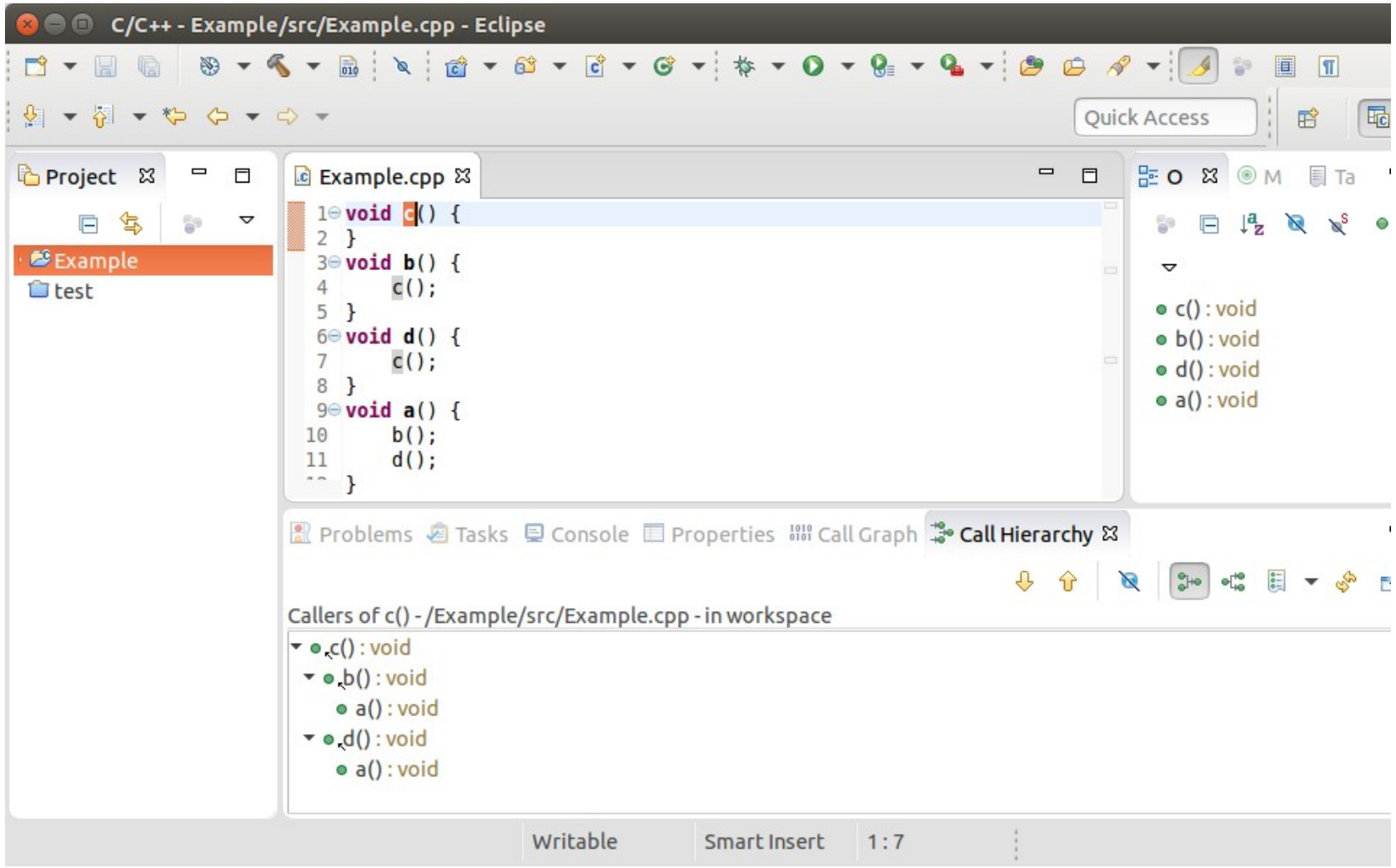
```
g++ stl.cpp -D_GLIBCXX_DEBUG -o stl
./stl
/usr/.../vector:346:error:
attempt to subscript container with
out-of-bounds
index 0, but container only holds
elements.
Objects involved in the operation:
sequence "this" @ 0x0x7ffcd6067da0
{
  type =
  NSt7__debug6vectorIiSaIiEEE;
  }
```

```
#include <vector>
#include <cstdio>
int main() {
    std::vector<int> v;
    v.reserve(10);
    printf("%i\n", v[0]);
    return 0;
}
```

- IDE: Eclipse CDT
  - Ctrl + k Finds the next occurrence
  - Ctrl + Shift + k Finds the previous occurrence
  - Ctrl + Shift + t
    Open Type
  - Since Mars
    incremental
    indexer
    is working fast
    on big projects.



- IDE: Eclipse CDT
  - Ctrl + Shift + h Open Type In Hierarchy
  - Alt + Shift + R in place refactor
  - Ctrl + Tab jumps between header and its implementation
  - Ctrl + left click follow selected
  - Ctrl + i correct indication



- Google Test (gtest, gmock)

```
#include <gtest/gtest.h>
unsigned int fib(unsigned int n) {
  if (n <= 0) {
    return 0;
  }
  unsigned int a = 1, b = 1;
  for (int i = 2; i <= n; i += 1) {
    unsigned int next = a + b;
    a = b;
    b = next;
  }
  return b;
}
```

```
TEST(FibonacciTest, testIterative)
{
  ASSERT_EQ(1, fib(1));
  ASSERT_EQ(2, fib(2));
  ASSERT_EQ(3, fib(3));
  ASSERT_EQ(5, fib(4));
}

TEST(FibonacciTest, testZero) {
  ASSERT_NE(1, fib(0));
}

int main(int argc, char **argv) {
  ::testing::InitGoogleTest(&argc,
    argv);
  return RUN_ALL_TESTS();
}
```

- Google Test (gtest, gmock)
  - What is the difference between gtest and gtest_main?

```
$./FibonacciTests

[==========] Running 2 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 2 tests from FibonacciTest
[ RUN      ] FibonacciTest.testIterative
[       OK ] FibonacciTest.testIterative (0 ms)
[ RUN      ] FibonacciTest.testZero
[       OK ] FibonacciTest.testZero (0 ms)
[----------] 2 tests from FibonacciTest (0 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test case ran. (0 ms total)
[  PASSED  ] 2 tests.
```

- Google Test (gtest, gmock)

```
[==========] Running 2 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 2 tests from FibonacciTest
[ RUN      ] FibonacciTest.testIterative
[       OK ] FibonacciTest.testIterative (0 ms)
[ RUN      ] FibonacciTest.testZero
./FibonacciTests.cpp:24: Failure
Expected: (1) != (fib(0)), actual: 1 vs 1
[  FAILED  ] FibonacciTest.testZero (0 ms)
[----------] 2 tests from FibonacciTest (0 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test case ran. (0 ms total)
[  PASSED  ] 1 test.
[  FAILED  ] 1 test, listed below:
[  FAILED  ] FibonacciTest.testZero

 1 FAILED TEST
```

- ccache, distcc
  - ccache - speeds up recompilation by caching previous compilation results and reusing it.

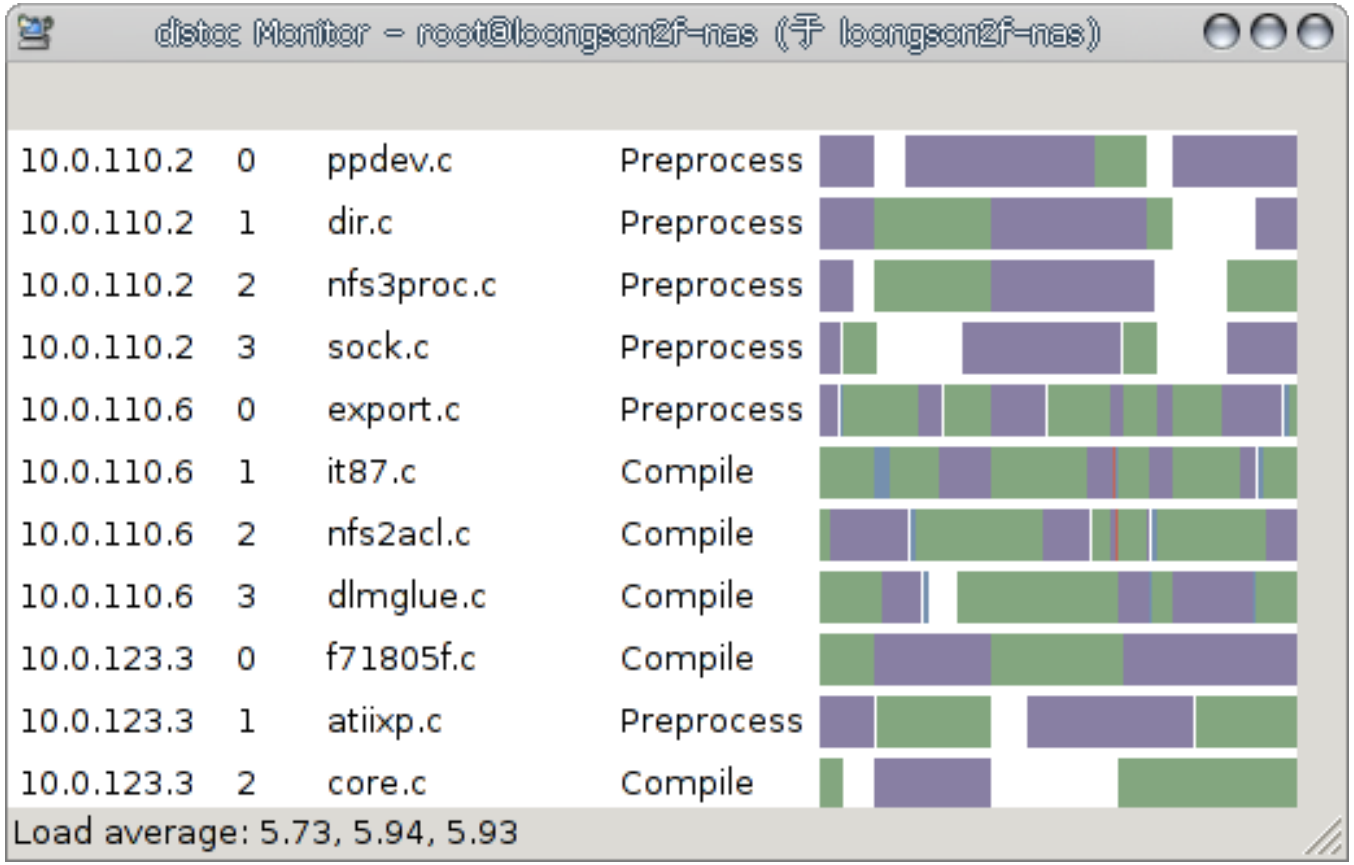
    export CC="ccache gcc"

    export CXX="ccache g++"

  - dictcc – distrubute compiling tasks across a network to hosts.

    make -jX

    Server: distccd
    Client: distcc



- Google cpplint
  - Style & correctness checker for C++ files. Implements what Google considers to be the best practices in C++ coding. Mostly relies on regular expressions.
  - Google C++ Style Guide http://google.github.io/styleguide/cppguide.html
  - The format for error messages is: File:Line Error

```
Add #include <string> for string  [build/include_what_you_use] [4]
Lines should be <= 120 characters long [whitespace/line_length [2]
Single-parameter constructors should be marked explicit.
[runtime/explicit] [5]
"public:" should be preceded by a blank line
[whitespace/blank_line] [3]
```

- Resources
  - http://www.eclipse.org/cdt/
  - http://darkdust.net/files/GDB%20Cheat%20Sheet.pdf
  - https://gcc.gnu.org/onlinedocs/libstdc++/manual/debug_mode_using.html
  - https://ccache.samba.org/
  - https://github.com/distcc/distcc
  - https://github.com/google/styleguide/tree/gh-pages/cpplint

- References
  - http://clang.llvm.org/docs/index.html
  - https://apps.ubuntu.com/cat/applications/quantal/distccmon-gnome/

Thank you

adbglobal.com