HOTSPOT INTERNALS: SAFEPOINTS, NULLPOINTERS AND STACKOVERFLOWS

Volker Simonis [Волькер Симонис], SAP / volker.simonis@gmail.com

http://www.sunstategearbox.com.au/wp-content/uploads/2014/09/gearbox.jpg

SIGNALS

- Asynchronous notifications sent to a process/thread
- Originate from 1970s Bell Labs Unix - now POSIX
- Quite heavy-weight operations
- Interferes with other programming models (e.g. C++ exceptions, threads)
- Nevertheless reliable, cross-platform (POSIX), useful..
- On Windows there's a similar mechanism called Structured/Vectored Exception Handling (SEH/VEH)

SIGNALS

- Many programmers are scared by signals
- Ever saw `SIGSEGV`, `SIGILL`, `SIGBUS`,.. ?
- They are usually associated with crashes and core files
- But they can be useful :-)

DEMO

NULL-POINTER CHECKS - C/C++

```
struct NullCheck {
  long x, y, z;
  long l0001;
};

void getField(NullCheck* n1, NullCheck* n2, NullCheck* n3, NullCheck* n4) {
  long tmp = n1->l0001;
  n1->l0001 = n2->l0001;
  n2->l0001 = n3->l0001;
  n3->l0001 = n4->l0001;
  n4->l0001 = tmp;
}

int main(int argc, char** argv) {
  NullCheck* n = (NullCheck*)0;
  getField(n, n, n, n);
}
```

NULL-POINTER CHECKS - C/C++

Unmanaged languages (e.g. C/C++) don't have Null-Pointer checks:

```
$ g++ NullCheck.cpp
$ ./a.out
Segmentation fault (core dumped)
$ objdump --disassemble --demangle
```

```
0000000000000000 <getField(NullCheck*, NullCheck*, NullCheck*, NullCheck*)>:
   0:   48 8b 47 18              mov    0x18(%rdi),%rax
   4:   4c 8b 46 18              mov    0x18(%rsi),%r8
   8:   4c 89 47 18              mov    %r8,0x18(%rdi)
   c:   48 8b 7a 18              mov    0x18(%rdx),%rdi
  10:   48 89 7e 18              mov    %rdi,0x18(%rsi)
  14:   48 8b 71 18              mov    0x18(%rcx),%rsi
  18:   48 89 72 18              mov    %rsi,0x18(%rdx)
  1c:   48 89 41 18              mov    %rax,0x18(%rcx)
```

NULL-POINTER CHECKS - HOTSPOT

Managed languages like Java guarantee Null-Pointer checks!

```
public class NullCheck {
  long 10000, 10001, 10002, 10003, 10004, 10005, 10006, 10007, 10008, 10009;
  //...
  long 10510, 10511, 10512, 10513, 10514, 10515, 10516, 10517, 10518, 10519;

  void getField_1(NullCheck n1, NullCheck n2, NullCheck n3, NullCheck n4) {
    long tmp = n1.10001;
    n1.10001 = n2.10001;
    n2.10001 = n3.10001;
    n3.10001 = n4.10001;
    n4.10001 = tmp;
  }
}
```

NULL-POINTER CHECKS - HOTSPOT

Managed languages like Java guarantee Null-Pointer checks!

```
public class NullCheck {
  long 10000, 10001, 10002, 10003, 10004, 10005, 10006, 10007, 10008, 10009;
  //...
  long 10510, 10511, 10512, 10513, 10514, 10515, 10516, 10517, 10518, 10519;

  void getField_2(NullCheck n1, NullCheck n2, NullCheck n3, NullCheck n4) {
    long tmp = n1.10512;
    n1.10512 = n2.10512;
    n2.10512 = n3.10512;
    n3.10512 = n4.10512;
    n4.10512 = tmp;
  }
}
```

NULL-POINTER CHECKS - HOTSPOT

```
void getField_2(NullCheck n1, NullCheck n2, NullCheck n3, NullCheck n4) {
  0x00007fe7a0b6f74c: test   %rsi,%rsi
  0x00007fe7a0b6f74f: je     0x00007fe7a0b6f7a4
  0x00007fe7a0b6f751: test   %rdx,%rdx
  0x00007fe7a0b6f754: je     0x00007fe7a0b6f7b5
  0x00007fe7a0b6f756: mov    0x1010(%rdx),%r10
  0x00007fe7a0b6f75d: mov    0x1010(%rsi),%r11
  0x00007fe7a0b6f764: mov    %r10,0x1010(%rsi)
  0x00007fe7a0b6f76b: test   %rcx,%rcx
  0x00007fe7a0b6f76e: je     0x00007fe7a0b6f7c5
  0x00007fe7a0b6f770: mov    0x1010(%rcx),%r10
  0x00007fe7a0b6f777: mov    %r10,0x1010(%rdx)
  0x00007fe7a0b6f77e: test   %r8,%r8
  0x00007fe7a0b6f781: je     0x00007fe7a0b6f7d5
  0x00007fe7a0b6f783: mov    0x1010(%r8),%r10
  0x00007fe7a0b6f78a: mov    %r10,0x1010(%rcx)
  0x00007fe7a0b6f791: mov    %r11,0x1010(%r8)
```

NULL-POINTER CHECKS - HOTSPOT

```
void getField_1(NullCheck n1, NullCheck n2, NullCheck n3, NullCheck n4) {



  0x00007facf0b6fccc: mov    0x18(%rsi),%r10
  0x00007facf0b6fcd0: mov    0x18(%rdx),%r11
  0x00007facf0b6fcd4: mov    %r11,0x18(%rsi)

  0x00007facf0b6fcd8: mov    0x18(%rcx),%r11
  0x00007facf0b6fcdc: mov    %r11,0x18(%rdx)

  0x00007facf0b6fce0: mov    0x18(%r8),%r11
  0x00007facf0b6fce4: mov    %r11,0x18(%rcx)
  0x00007facf0b6fce8: mov    %r10,0x18(%r8)
```

NULL-POINTER CHECKS - HOTSPOT

DEMO

NULL-POINTER CHECKS & COMPRESSED OOPS

- On 64-bit platforms pointers are 8-byte aligned
  - The three least-significant bits are redundant (i.e. zero)
- We can actually encode 32G within 32-bit..
  - ..by shifting right/left for encoding/decoding
- If (`Java-heap < 4G && max_heap_Addr < 4G`) ==> *Unscaled mode*
  - No encoding/decoding - oops fit into 32 bit
- If (`Java-heap < 32G && max_heap_Addr < 32G`) ==> *Zero-Based mode*
  - Shifting for encoding/decoding
- If (`Java-heap < 32G`) ==> *Heap-Based mode*
  - Shifting plus base subtraction/addition for encoding/decoding
- See https://wiki.openjdk.java.net/display/HotSpot/CompressedOops

NULL-POINTER CHECKS & COMPRESSED OOPS

```
public class NullCheck_CompOops {
  long l0001;
  NullCheck_CompOops nc;

  void getField_1(NullCheck_CompOops n1, NullCheck_CompOops n2) {
    long tmp = n1.nc.l0001;
    n1.l0001 = n2.l0001;
    n2.l0001 = tmp;
  }
}
```

NULL-POINTER CHECKS & COMPRESSED OOPS

DEMO

NULL-POINTER CHECKS - SUMMARY

- Null-pointer checks are done implicitly (if possible):
  - Not on all platforms (i.e. AIX can read from `0x0000`)
  - Not all field offsets (usually within `` `getconf PAGESIZE` ``)
- If there are too many NPE (controlled by `PerBytecodeTrapLimit`)
  - Methods are made "not-entrant" and..
  - ..recompiled with explicit checks instead
- Work together with Compressed Oops

But where's the benchmark?

https://github.com/shipilev/article-compress-me/blob/master/src/main/java/net/shipilev/ImplicitNullChecks.java

SAFEPOINTS

> *“ A point during program execution at which all GC roots are known and all heap object contents are consistent. ”*
>
> HotSpot Glossary of Terms

- HotSpot uses a *cooperative* suspension model
- All threads need to come to a safepoint quickly if required
  - Running interpreted: change interpreter dispatch table
  - Running JIT-compiled: read global safepoint polling page
  - Running in native (JNI): no need to stop
    - native code accesses oops trough handles
    - block when returning from JNI or when calling to Java

SAFEPOINTS

DEMO

ARRAY-OUT-OF-BOUNDS CHECKS

DEMO