Precise detection of memory leaks

Jonas Maebe, Michiel Ronsse, Koen De Bosschere

Ghent University, ELIS Department, Belgium

presented by Jaroslav Ševčík
Overview

- Memory leak – failure to release memory.
- Focus on **physical memory leaks**
  - …memory block is physically lost
  - Does not occur in Java
- Logical memory leaks
  - …memory block referenced but never used.
- Uses dynamic instrumentation
Analysis results

Output:

List of memory leaks including:

- Occurrence count
- Allocation site
- Information about the last reference
  - Where it was created
  - Where it was lost
Detection

- Based on reference counting
- Uses a dynamic instrumentation framework
  - Intercepts all memory-related calls and write operations
- Keeps track of:
  - List of all allocated and freed memory blocks
  - List of all references in memory
  - … including stack traces.
- If last reference to block lost
  … potential (why?) leak is recorded
Memory leak verification

Why is it **potential**?

- E.g. reference in registers.

⇒ verification (when the same leak detected or at the end):
  - Was it freed in the meantime?
  - Was it used?
    ...then someone had to touch the block, hence
    - There must be a reference in a register or
    - Hidden reference via pointer arithmetics.

Check succeeds ⇒ the leak is deemed permanent.
Imprecisions

False positives:

- Returning the last reference in a register.
- References via pointer arithmetics.
- Overwriting only part of a pointer or writing byte by byte.

False negatives:

- Random memory content = address of a block . . .  
  . . .  C can’t use reflection to tell them apart
- Leaked cycles.
Evaluation

Tested on

- *Vim* and *lynx* – no recurring memory leak.
- *Fortran parallel transformer* – memory leak discovered (50000+ lines of C++ code).

Performance impacts:

- Slowdown factor 200-300
- Memory consumption factor - approximately 2
Conclusion

- The most important features are:
  - Dynamic instrumentation
  - Information about the last reference
- Lot of C-related technicalities.
- Not very applicable to Java
- Surprisingly low memory overhead . . .
- . . .and surprisingly slow