Making Elfutils' libdw Thread-Safe

Srđan Milaković, Jonathon Anderson
Department of Computer Science
Rice University

Changes for Thread-Safety

Main Modifications

- Made Dwarf_Abbrev hash table thread-safe by adding atomics to mediate concurrent accesses
- Added Pthread-based thread-safety to libdw memory management

Minor modifications

- Made __libelf_version* thread-local, to work correctly when threaded

Dwarf_Abbrev Background

- Dwarf stores most of its information in tree-based structures
- Libdw caches .dwarf_abbrev section subtrees as Dwarf_Abbrev structures
- Each Dwarf_Abbrev is referenced by a unique abbreviation code, used as the key for the Dwarf_Abbrev hash table (shown here)
- Lookups are frequent when scanning Dwarf information; mutual exclusion would be slow
- We enhanced the original to work in parallel!

Dwarf_Abbrev Concurrent Hash Table

- **Based on original implementation, but with enhancements for concurrency**
- **Supports only insert and find, iteration is not needed for Dwarf_Abbrev**
- **Only one entry per key, keys are unique**
- **Only whole-table operation is resizing, other operations use per-entry atomics to mediate**
- **Threads waiting for a resize to complete can “help” by initializing and copying entries, mediated by atomic counters**

Libdw Memory Management

- Libdw allocates small structures in internal caches
- Using malloc/free directly is slow, so libdw uses a suballocator to manage memory
- Everything is freed upon dwarf_end: only have to deal with allocation
- Memory blocks are held in a singly-linked stack, top block is used for allocation if enough remains, otherwise a new block is pushed
- We enhanced the original!

Thread-safe Memory Management

- **Simple solution: use a separate allocation stack for every thread, free all on dwarf_end**
- **TLS wouldn’t allow dwarf_end to free, so have to use a more manual structure**
- **Use a shared atomic counter to allocate IDs to every new encountered thread (static TLS)**
- **If memory stacks array is not large enough, acquire a lock and resize (mediated via rwlock)**
- **Every thread uses the stack entry at its ID, not a performance bottleneck**

Performance Results

Timed Dyninst DWARF parsing on 32 threads
Input binary has 280KiB symbol table + 73.0MiB DWARF
Parallel speedup: 10.43x (1.10s, 11.51s in serial)
Serial slowdown: ~1.00x (11.51s, 11.48s before)
Graphical trace output (white is idle, color is work):