Programovanie v operačných systémoch

04 - Memory

Jozef Šiška

Department of Applied Informatics
Comenius University in Bratislava

2020/2021

Memory management

Memory management

- Allocation
  - kernel: brk, mmap
  - C/C++: malloc, realloc, free, mmap, new, delete
- "Management"
  - pairing alloc/release, memory leaks
  - ownership, passing between functions etc. (size?)
  - dangling pointers
  - `std::unique_ptr`, `std::shared_ptr`
- Reference counting
  - RAII, immediate release, cycles?
  - implicit sharing, COW
- Garbage collection
  - when will it happen? price of detection?

Reference counting

- `std::shared_ptr`
- immediate "release", RAII similar to other resources
- cheap / fast (at least relatively: large object "trees" can take a while to release, which can be noticable in realtime apps)
- slight space (refcount/control block) / speed (inc/dec) overhead
- memory access - one or two dereferences
- synchronization (atomic refcount)
- cycles!
- breaking cycles: weak references
  - can become dangling
  - reference "zeroing"
    - keep track of both weak and strong references
    - when "strong" refcount becomes zero, data is released and weak references can't be used anymore to access data
    - `std::shared_ptr` + `std::weakt_ptr`

Garbage collection

- doesn’t combine nicely with management of other resources... (Java `finalize()`)
- "unpredictable", performance...

- reference counting + cycle detection (Python)
- tracing - find objects not reachable from "root" objects

Memory leaks

So... how to avoid memory leaks in C / bad C++ / ...?

- release reources before each `return`
- `goto` solutions
- with exceptions in C++, everything is a possible `return`!
- valgrind (memcheck)
- (and other tools...)

... and is Java really safe?

- "hidden" references: registering listeners, observers,...

Copy on write (COW)

- reference counting on steroids
- cheap pass by value even for very large objects
- don't make copies when not needed
- shared data - every data class is basically a shared (refcounted) pointer
- Copy on change
  - C++: const vs non-const methods
  - when refcount > 1
- might not be always possible/feasible (`std::string` in C++11?)
- unpredictable/unintuitive complexity/efficiency

  ```
  String s2 = s1; s2[0] = 'a';
  ```

Small string/object optimisation

- Conatiner classes allocate data buffers on heap
- Usually need `data` pointer, `capacity` and `size`
  `sizeof(void*) + sizeof(size_t) + sizeof(size_t) == 24;`
- If the data is small, this is a "waste" (24 bytes on stack to point to 2 byte string on heap)
- Sacrify one bit from size/capacity (or maybe pointer) as a flag that the fields contains actual data