

# Programovanie v operačných systémoch

## 04 - Memory

Jozef Šiška



Department of Applied Informatics  
Comenius University in Bratislava

2016/2017



- 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
- Reference counting
  - RAI, immediate release, cycles?
  - implicit sharing, COW
- Garbage collection
  - when will it happen? price of detection?

# Reference counting

- `std::shared_ptr`
- immediate "release", RAI 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
- 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::weak_ptr`

- doesn't combine nicely with management of other resources...  
(cf Java `finalize()`)
- "unpredictable", performance...
- reference counting + cycle detection (Python)
- tracing - find objects not reachable from "root" objects

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

- 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';`