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
  - 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++ / ...?

Resource management in C

```c
int do_something()
{
    int fd_in, fd_out;
    char *buffer;

    if ((fd_in = open(.....)) == -1)
        return -1;
    if ((fd_out = open(....)) == -1)
        return -1; // !!!

    if ((buffer == malloc(...)) == NULL)
        return -1; // !!!

    // do something

    free(buffer);
    close(fd_out);
    close(fd_int);

    return 0;
}
```

Resource management in C

```
int do_something()
{
    int fd_in, fd_out;
    char *buffer;

    if ((fd_in = open(.....)) == -1)
        return -1;
    if ((fd_out = open(....)) == -1)
        close(fd_in);
        return -1; //

    if ((buffer == malloc(..)) == NULL)
       close(fd_out);
       close(fd_in);
        return -1; //

    // more resources ?!

    free(buffer);
    close(fd_out);
    close(fd_int);

    return 0;
}
```

Resource management in C

```c
int do_something()
{
    int fd_in, fd_out, ret = -1;;
    char *buffer;

    if ((fd_in = open(.....)) != 1) {
        if ((fd_out = open(....)) != -1) {
            if ((buffer == malloc(...)) != NULL) {
                // do something...
                // ret = 0;
                free(buffer);
            }
            close(fd_out);
        }
        close(fd_in);
    }

    return ret;
}
```

Resource management in C

```
int do_something()
{
    int fd_in, fd_out, ret = -1;
    char *buffer;

    if ((fd_in = open(.....)) == -1)
        goto err_fd_in;
    if ((fd_out = open(....)) == -1)
        goto err_fd_out;

    if ((buffer == malloc(...)) == NULL)
        goto err_buffer;

    // do something
    ret = -1;

    free(buffer);
err_fd_buffer:
    close(fd_out);
err_fd_out:
    close(fd_int);
err_fd_in:
    return ret;
}
```

Resource management in C++ (RAII)

```
int doSomething() // returns -1 or throws std::bad_alloc on errors
{
    ifstream inf(...);
    if (!inf.good()) return -1;

    ofstream outf(...);
    if (!outf.good()) return -1; // ifstream destructor releases resource

    Buffer buffer(...); // throws std::bad_alloc when a problem occurs

    // do something

    return 0;
}
```

Resource management in C++ (RAII)

```
void doSomething() // throws something on errors
{
    File inf(...);
    File outf(...); // throws IOException when open fails?
    Buffer buffer(...);

    // do something
}
```

So, why are RAII + exceptions not used all the time...

- need to go all the way...
- ... so people consider C++ exceptions problematic
- interop with c / "bad" libraries (need to wrap everything etc)

Resource management in C++ (RAII)

```
void doSomething() // throws something on errors
{
    File inf(...);
    File outf(...); // throws IOException when open fails?
    Buffer buffer(...);

    // do something
}
```

So, why are RAII + exceptions not used all the time...

- need to go all the way...
- ... so people consider C++ exceptions problematic
- interop with c / "bad" libraries (need to wrap everything etc)

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';
```