Automatic Memory Management

Godmar Back

Virginia Tech

November 8, 2022

Automatic Memory Management

Rationale

Explicit memory management (via, e.g. `malloc()` and `free()`) is prone to errors. All modern languages provide forms of automatic memory management, also called "implicit memory management."

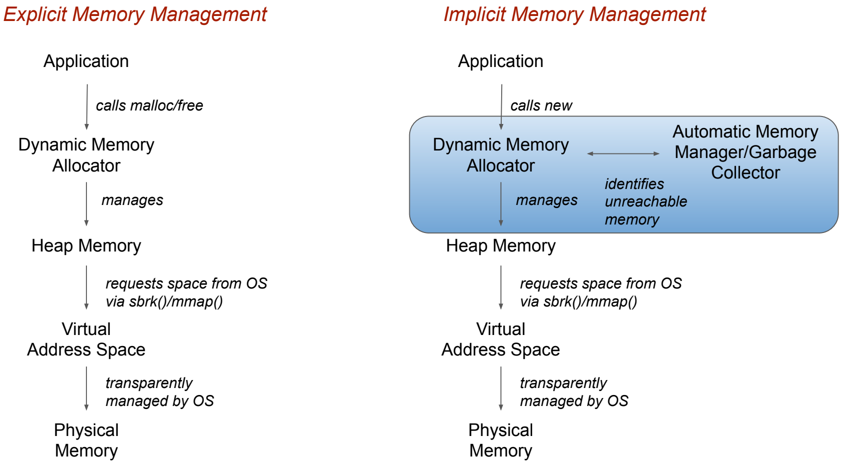
- Manual (explicit) memory management is difficult, many errors are possible
  - Free memory too early, risk use-after-free errors
  - Free too late (or forget to free (*)), risk memory leaks
- Requires principled design that identifies ownership and lifetimes of objects
- Complicates design of APIs

Will study

- Garbage Collection: Principles, Implementation, and Tuning
- Reference-counting approaches
- Related Programming Issues: Leaks, Churn, and Bloat

VIRGINIA TECH.

Explicit vs. Implicit Memory Management



VIRGINIA TECH

Garbage Collection

Key Idea

Identify those objects that the program may be accessing in the future. Keep them, reclaim the rest.

- Invented in 1960 by McCarthy for LISP [1]
- Assumption: well-defined programs cannot legally access objects to which they do not have pointers/references
  - Assumes no pointer $\leftrightarrow$ integer conversion
- Objects that can be accessed are said to be reachable
- We do not know if program will access any reachable object in the future
  - Those that won't be accessed are said to be leaked
- Essential abstraction: reachability graph

VIRGINIA TECH

Reachability Graph: Java Example

```java
class B {
  int x, y;
  B(int x, int y) {
    this.x = x;
    this.y = y;
  }
}

public class A {
    static A S;
    B f;
    public static void main(String[] args) {
      S = new A();
      A local = new A();
      B b = new B(1, 2);
      set(local, b);
      b = null;
      local = null;
    }
    static void set(A t, B b) {
      t.f = b;
    }
}
```

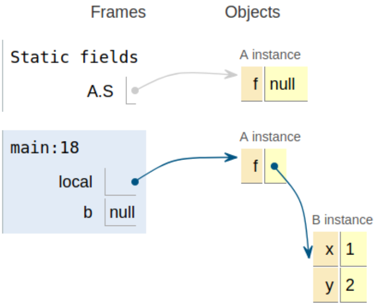


Figure 1: Reachability graph after setting `b = null`. Made with http://pythontutor.com/java.html

Mark and Sweep Garbage Collection

- Identify roots, e.g., in Java
  - Static fields
  - Local variables of in-progress method calls of all threads
  - JVM Internal roots
- Traverse the entire heap via, e.g. DFS, "mark"ing all reachable objects
- Reclaim ("sweep") all objects not marked

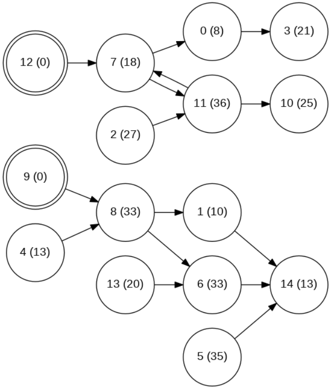


Figure 2: Reachability Graph

References

[1] John McCarthy.
Recursive functions of symbolic expressions and their computation by machine, part I.
Communications of the ACM, 3(4):184–195, 1960.

VIRGINIA TECH