Memory Management

YSC3208: Programming Language Design & Implementation

Răzvan Voicu

Week 12, April 3-7, 2017

1. Heap Memory
   - Memory Allocation for Programs
   - A Heap Memory Model

2. Heap Management Techniques

Resources of computing in dVM

- Time: accounted for by the number of executed instructions
- Memory: not well represented yet, instructions freely construct “things”

Questions

- Does a given data structure have to be stored forever?
- Will a given program run out of memory?
- Can we design a virtual machine that makes effective use of the available memory? Allowing memory to be easily re-cycled.

Memory Allocation for Programs

- Static allocation
- Stack allocation
- Heap alloation

Static Allocation

- Assign fixed memory location for every identifier
- Limitations
  - The size of each data structure must be known at compile-time. For example, arrays whose size depends on function parameters are not possible.
  - Recursive functions are not possible, because each recursive call needs its own copy of parameters and local variables.
  - Data structures such as closures cannot be created dynamically.

Stack Allocation

- Keep track of information on function invocations on runtime stack
- Recursion possible
- Size of locals can depend on arguments
- Remaining shortcomings:
    - Difficult to manipulate recursive data structures
    - Only objects with known compile time size can be returned from functions

Heap Allocation

- Data structures may be allocated and deallocated in any order
- Complex pointer structures will evolve at runtime
- Management of allocated memory becomes an issue

A Heap Memory Model

- Nodes: stack frames, operand stacks, environments etc
- Edges: references between nodes
- Labels on edges
- Nodes can point to primitive values

Formal Model of Heap

A heap is a pair (V, E), where

$$E \subseteq \{(v, f, w) | v \in V, f \in \mathbf{LS} + \mathbf{Int}, w \in V + \mathbf{PV}\}$$

Edge (v, f, w) has source v, label f and target w.
Edges are functional in the first two arguments.

Issues

- How realistic is this graph view of a heap?
- Once a node is created, will it have to be stored until the end of the program execution?

Memory Management

- $V = V_{useful} \cup V_{useless}$
- Is there an algorithm to compute V_{useful} and $V_{useless}$?
- No, undecidable! :-(
- Idea: Approximate V_{useful} and $V_{useless}$ by
  $V_{live} \supseteq V_{useful}$
  $V_{dead} \subseteq V_{useless}$

1. Heap Memory

2. Heap Management Techniques
    - Reference Counting
    - Mark-Sweep Garbage Collection
    - Copying Garbage Collection
    - Assessment

Reference Counting

$$V_{dead} = \{w \in V \mid \text{ there is no } f \in L, v \in V, \text{ s.t. } (v, f, w) \in E\}$$

Every update operation identifies all new elements of V_{dead} and makes them available for future *newnode* operations.

Freelist for Keeping Track of Free Memory

- Heap memory divided into blocks
- Each block has a header with meta-info
  - A field contains the length of the block, so that the next block can be found
- Some blocks are free, some are occupied
- Free blocks are chained into a "free list", via another meta-field
- Allocation: Find a free block large enough to host the requested size
  - Divide the block into two: one will hold the data, the other remains free
  - Update the free list accordingly
- Deallocation: add the deallocated block into the free list
  - Possibly coalesce adjacent free blocks

Advantages of Reference Counting

- Incrementality
- Locality
- Immediate reuse

Disadavantages of Reference Counting

- Runtime overhead
- Cannot reclaim cyclic data structures

Mark-Sweep Garbage Collectors

When `newnode()` runs out of memory, a garbage collector computes a set V_{dead} and reclaims the memory its elements were occupying.

Liveness

$$\frac{\exists f.(v_1, f, v_2) \in E}{v_1 \longrightarrow v_2}$$

$$\frac{}{v \longrightarrow^* v} \qquad \frac{v_1 \longrightarrow v_2}{v_1 \longrightarrow^* v_2} \qquad \frac{v_1 \longrightarrow^* v_2 \qquad v_2 \longrightarrow^* v_3}{v_1 \longrightarrow^* v_3}$$

Liveness (continued)

The set V_{live} of a machine in state $(os, pc, e, rs, (V, E))$ is now defined as follows:

$$V_{live} = \{v \in V \mid r \longrightarrow^* v,\ \text{where } r \in \{os, e, rs\}\}$$

$\{os, e, rs\}$ are called roots.

Idea

Visit all nodes in V_{live} and `MARK` them.
Visit every node in the heap and free every `UNMARKED` node.

Copying Garbage Collection - Idea

- Use only half of the available memory for allocating nodes
- Once this half is filled up, copy the live memory contained in the first half to the second half
- Reverse the roles of the halves and continue.

Historical Background

- Pioneered by LISP implementations
- Reference counting: Gelernter et al 1960, Collins 1960, used in Smalltalk, and Modula-2+
- Mark-sweep: McCarthy 1960, widely used, e.g. JVM
- Minsky 1963, Cheney 1970, widely used in functional and logic programming

Explicit Heap Deallocation

```
var p : ^t
p := nil

new(p)

dispose(p)
```

Space Leak

```
new(p);
p := nil
```

Dangling Reference

```
a.s := p;
dispose(p)
```

Memory Management in Software Systems

Space leaks can occur even in systems with automatic memory management.

Large systems implement their own memory management (e.g. Adobe Photoshop).