COMS W4115
Programming Languages and Translators
Lecture 18: Run-time Environments
April 3, 2013

Lecture Outline

1. Activation trees
2. Calling sequences
3. Access to local arrays
4. Heap memory management

1. Activation Trees

- Consider the following C program:

```
int x = 2;

void f(int n) {
  static int x = 1;
  g(n);
  x--;
}

void g(int m) {
  int y = m-1;
  if (y > 0) {
    f(y);
    x--;
    g(y);
  }
}

main() {
  g(x);
  return 0;
}
```

- The activation tree for this program is

```
   main()
     |
    g(2)
    / \
 f(1) g(1)
  |
 g(1)
```

2. Calling Sequences

- Procedure calls are implemented by calling sequences, code that allocates an activation record on the control stack and enters information into its fields.
- A return sequence is code invoked after the call to restore the state of the machine so the calling procedure can continue its execution after the call.
- The code in a calling sequence is usually divided between the calling procedure (the "caller") and the procedure it calls (the "callee").
- When a procedure p calls a procedure q, we might do something like the following:
  - Evaluate the parameters of q and store them in a new AR for q.
  - Store the frame pointer (fp) for p as the control link in the AR for q.
  - Update fp to point to the AR of q.
  - Store the return address in the AR for q.
  - Jump to the code for q.
  - Have q allocate and initialize its local data and temporaries on the stack.
- When q exits:

    - Copy the fp of the AR for q into sp (the top of stack pointer).
    - Load the control link of the AR for q into the fp.
    - Jump to the return address.
    - Change the stack pointer to pop the parameters of q off the run-time stack.
- Contrast the run-time stack during the first and second calls to `g(1)` in the program in section (1).

3. Allocating Space for Arrays

- Fixed-size arrays

```
void f(int x) {
  int a;
  int b[4];
  int c;
  ...
}
```

- Space for the array b can be allocated on the runtime stack between the space for a and c.
- Variable-size arrays

```
void f(int x) {
  int a;
  int b[n];
  int c;
  ...
}
```

- A pointer to the space for b can be allocated on the runtime stack between the space for a and c so variables remain a constant offset from the frame pointer. The actual space for b can be allocated after c.

4. Heap Memory Management

- The runtime heap is used for data objects whose lifetimes may exist long after the procedure that created them.
- The heap memory manager is the subsystem that allocates and deallocates space within the heap.
- Garbage collection is the process of finding memory within the heap that is no longer used by the program and can therefore be reallocated to house other data objects. In some programming languages like C allocatation and deallocation needs to be done manually using library functions like `malloc` and `free`. In other languages like Java it is the garbage collector that deallocates memory.
- There are many different garbage collection algorithms that vary in performance metrics such as execution time, space usage, pause time, and program locality.
- Mark-and-sweep garbage collection
    - A basic mark-and-sweep (M&S) garbage collection algorithm is straightforward to implement and works well for languages like C because it does not move data objects during garbage collection.
    - A M&S collector firsts visits and marks all reachable data objects in the heap, setting the reached-bit of each object to 1.
    - It then sweeps the entire heap, freeing up unreachable objects.
    - See Algorithm 7.12 (ALSU, p. 471) for details.

5. Practice Problem

- Consider the following program written in a hypothetical statically scoped language that allows nested functions. In this program, main calls f which calls g2 which calls h which calls g1.

```
function main() {
  int i;

  function f() {
    int a, b, c;

    function g1() {
      int a, d;

      a = b + c;              // point 1
```

```
    }; // end of g1

    function g2(int x) {
      int b, e;

      function h() {
        int c, e;

        g1();
        e = b + a;              // point 2

      }; // end of h

      h();
      a = d + e;                // point 3

    }; // end of g2

    g2(1);

  }; //end of f

  // execution of main begins here

  f();

}; // end of main
```

a. Suppose we have activation records with the following fields:
   Parameters
   Control Link
   Access Link
   Return Address
   Local data

If function p is nested immediately within function q, then the access link in any AR for p points to the most recent AR for q.

Show the activation records on the run-time stack when execution first arrives at point 1 in the program above.

- To which declarations are the references to variables a, b, c at position 1?
- To which declarations are the references to variables a, b, e at position 2?
- To which declarations are the references to variables a, d, e at position 3?

6. Reading

- ALSU, Sections 7.1, 7.2, 7.6.1
- K. C. Louden, Compiler Construction, PWS Publishing, 1997.

---

aho@cs.columbia.edu