CSE 3302 Notes 5: Memory Management

(Last updated 10/27/15 12:36 PM)

References:

Gabbrielli-Martini: 5
Wirth: 4

5.1. Implementing "Simple" Subprograms

Historical FORTRAN and COBOL

Simple =

- no recursion
- static allocation
- minimal call-by-value
- call-by-reference for anything non-trivial
- If program starts, it will have enough memory

Gabbrielli, figure 5.1 - one activation record per subprogram

Layout of activation record (a.r., stack frame), register conventions mandated by vendor

Who saves registers? - caller or callee?

5.2. Implementing Subprograms with Run-Time Stack

PL/0

Calling Sequence:

Caller

```
cal l,a:  call procedure a (absolute address) at level l

    cal: begin {generate new block mark}
           s[t + 1] := base(l);        { static link for nested procs }
           s[t + 2] := b;              { dynamic link to caller proc }
           s[t + 3] := p;              { return address }
           b := t + 1;                 { new base of stack frame
           p := a                      { new program counter }
         end;
```

Dynamic link is needed for C; Pascal and PL/0 also need static (i.e. lexical scope) link for immediate containing procedure.

Called

`jmp` over any instructions for any nested procedures

```
int 0,a  :  increment t-register by a  { a includes 3 slots for cal }

      int: t := t + a;   { allocate stack frame including sl, dl, ra }
```

Return Sequence

Called

```
opr 0,a  :  execute operation a (=0 here)

      opr: case a of {operator}
           0: begin {return}
                 t := b - 1;             { discard stack frame }
                 p := s[t + 3];          { return to caller }
                 b := s[t + 2];          { old base address }
              end;
```

Caller does nothing special

Pascal-S

Interpreter addressing is done using *display* array to avoid cost of `base` function.

5.3. Referencing Stack-Dynamic Local Variables for Nested Subprograms

A nested subprogram may potentially reference local variables for subprograms that contain it. Two difficulties (that also interact):

1. Calls among nested subprograms at same level.
2. Recursion

For a given variable (under lexical/static scoping), the *most recent* invocation of each containing subprogram is the one whose activation record is needed.

(The outermost scope is level 0. Nested scopes have increasing level numbers.)

There is exactly one activation record *per level* in the (ascending) *static chain*.

PL/0

Assumption - non-local data is rarely accessed, simple solution is sufficient

Referencing is based on:

Compiler computes level *difference* to put into `lod`, `sto`, and `cal` instructions
`cal` instruction sets saved static link (`s[t + 1]`) to point at next level a.r.

Level difference is used with loop to follow static links:

```
instruction = packed record
                 f: fct;                {function code}
                 l: 0..levmax;          {level}
                 a: {0..amax} integer   {displacement address}
               end;

function base(l: integer): integer;
  var b1: integer;
  begin
    b1 := b; {find base l levels down}
    while l > 0 do
      begin
        b1 := s[b1];
        l := l - 1
      end;
    base := b1
 end {base};
```

To help with optimizing code, compilers may use a sequence of indirections.

```
      http://ranger.uta.edu/~weems/NOTES3302/NEWNOTES/NOTES05/simple.pl0
   0 var a;              0   jmp  0   27    These 3 jmp's get compiled in block
   1                                        with "gen(jmp,0,0)", but are patched
   1 procedure b;        1   jmp  0   19    when the start addresses are known.
   1                                        (Just before "statement" is called in
   1   var c,d;                             block.)
   2
   2   procedure e;      2   jmp  0    3
   2
   2     var f,g,h;
   3
   3     begin           3   int   0     6 code for e
   4       a:=1;         4   lit   0     1
                         5   sto   2     5 a
   6       c:=2;         6   lit   0     2
                         7   sto   1     3 c
   8       d:=3;         8   lit   0     3
                         9   sto   1     4 d
  10       f:=4;        10   lit   0     4
                        11   sto   0     3 f
  12       g:=5;        12   lit   0     5
                        13   sto   0     4 g
  14       h:=6;        14   lit   0     6
                        15   sto   0     5 h
  16       call e;      16   cal   1     3 e
  17       call b       17   cal   2     1 b
  18     end;           18   opr   0     0 return
  19
  19   begin            19   int   0     5 code for b
  20     a:=7;          20   lit   0     7
                        21   sto   1     5 a
  22     c:=8;          22   lit   0     8
                        23   sto   0     3 c
  24     call e;        24   cal   0     3 e
  25     call b         25   cal   1    19 b
  26   end;             26   opr   0     0 return
  27
  27 begin              27   int   0     6 code for unnamed driver
  28   a:=9;            28   lit   0     9
                        29   sto   0     5 a
  30   if 0=1 then      30   lit   0     0
                        31   lit   0     1
                        32   opr   0     8 =
  33     call b         33   jpc   0    35
                        34   cal   0    19 b
  35 end.               35   opr   0     0 return
start pl/0
end pl/0
```

http://ranger.uta.edu/~weems/NOTES3302/NEWNOTES/NOTES05/args.pl0

```
    0 var aout,result2;
    1
    1 procedure a(ain);
    2
    2   var bout;
    2
    2   procedure b(bin);
    3
    3     var cout;
    3
    3     procedure c(cin);
    4
    4       begin
    5         cout:=cin+8;
    9         result2:=ain+bin+cin+8
   15       end;
    4   int   0     4 code for c
    5   lod   0     3 cin
    6   lit   0     8
    7   opr   0     2 +
    8   sto   1     4 cout
    9   lod   2     3 ain
   10   lod   1     3 bin
   11   opr   0     2 +
   12   lod   0     3 cin
   13   opr   0     2 +
   14   lit   0     8
   15   opr   0     2 +
   16   sto   3     6 result2
   17   opr   0     0 return
   18     begin
   19       call c(4);
   23       bout:=bin+cout
   25     end;
   18   int   0     5 code for b
   19   int   0     3 push args(s) for c
   20   lit   0     4
   21   int   0    -4 -(3+number of args)
   22   cal   0     4 c
   23   lod   0     3 bin
   24   lod   0     4 cout
   25   opr   0     2 +
   26   sto   1     4 bout
   27   opr   0     0 return
   28
   28   begin
   29     call b(2);
   33     aout:=ain+bout
   35   end;
   28   int   0     5 code for a
   29   int   0     3 push args(s) for b
   30   lit   0     2
   31   int   0    -4 -(3+number of args)
   32   cal   0    18 b
   33   lod   0     3 ain
   34   lod   0     4 bout
   35   opr   0     2 +
   36   sto   1     5 aout
   37   opr   0     0 return
   38
   38 begin
   39 call a(1);
   43 out:=aout;
   45 out:=result2
   46 end.
   38   int   0     7 code for unnamed driver
   39   int   0     3 push args(s) for a
   40   lit   0     1
   41   int   0    -4 -(3+number of args)
   42   cal   0    28 a
   43   lod   0     5 aout
   44   wro   0     0
   45   lod   0     6 result2
   46   wro   0     0
   47   opr   0     0 return
```

```
b=1 p=39      initial
   7        0 result2
   6        0 aout
   5  -999999 cvy
   4  -999999 cvx
   3        0 ret adr
   2        0 d.l.
   1        0 s.l.
b=8 p=29      after call to a
  12        0 bout
  11        1 ain
  10       43 ret adr
   9        1 d.l.
   8        1 s.l.
   7        0 result2
   6        0 aout
   5  -999999 cvy
   4  -999999 cvx
   3        0 ret adr
   2        0 d.l.
   1        0 s.l.
b=13 p=19     after call to b
  17        0 cout
  16        2 bin
  15       33 ret adr
  14        8 d.l.
  13        8 s.l.
  12        0 bout
  11        1 ain
  10       43 ret adr
   9        1 d.l.
   8        1 s.l.
   7        0 result2
   6        0 aout
   5  -999999 cvy
   4  -999999 cvx
   3        0 ret adr
   2        0 d.l.
   1        0 s.l.
b=18 p=5      after call to c
  21        4 cin
  20       23 ret adr
  19       13 d.l.
  18       13 s.l.
  17        0 cout
  16        2 bin
  15       33 ret adr
  14        8 d.l.
  13        8 s.l.
  12        0 bout
  11        1 ain
  10       43 ret adr
   9        1 d.l.
   8        1 s.l.
   7        0 result2
   6        0 aout
   5  -999999 cvy
   4  -999999 cvx
   3        0 ret adr
   2        0 d.l.
   1        0 s.l.
start pl/0
15
15
end pl
```

http://ranger.uta.edu/~weems/NOTES3302/NEWNOTES/NOTES05/rtsExample.pl0

```
 0 procedure iloop(i);
 2
 2   procedure jloop(j);
 3
 3     procedure kloop(k);
 4
 4       begin
 5         k:=k+1;
 9         out:=i+j+k;
15         if k<2 then
18           call kloop(k);
23         if k=2 then
26           call jloop(j)
31       end;
 4   int   0    4 kloop code
 5   lod   0    3 k
 6   lit   0    1
 7   opr   0    2 +
 8   sto   0    3 k
 9   lod   2    3 i
10   lod   1    3 j
11   opr   0    2 +
12   lod   0    3 k
13   opr   0    2 +
14   wro   0    0
15   lod   0    3 k
16   lit   0    2
17   opr   0   10 <
18   jpc   0   23
19   int   0    3
20   lod   0    3 k
21   int   0   -4
22   cal   1    4 kloop
23   lod   0    3 k
24   lit   0    2
25   opr   0    8 =
26   jpc   0   31
27   int   0    3
28   lod   1    3 j
29   int   0   -4
30   cal   2    2 jloop
31   opr   0    0 return
32
32     begin
33       j:=j+10;
37       if j<30 then
40         call kloop(0);
45       if j=30 then
48         call iloop(i)
53     end;
32   int   0    4 jloop code
33   lod   0    3 j
34   lit   0   10
35   opr   0    2 +
36   sto   0    3 j
37   lod   0    3 j
38   lit   0   30
39   opr   0   10 <
40   jpc   0   45
```

```
41   int   0    3
42   lit   0    0
43   int   0   -4
44   cal   0    4 kloop
45   lod   0    3 j
46   lit   0   30
47   opr   0    8 =
48   jpc   0   53
49   int   0    3
50   lod   1    3 i
51   int   0   -4
52   cal   2    1 iloop
53   opr   0    0 return
54
54   begin
55     i:=i+100;
59     if i<300 then
62       call jloop(0)
67   end;
54   int   0    4 iloop code
55   lod   0    3 i
56   lit   0  100
57   opr   0    2 +
58   sto   0    3 i
59   lod   0    3 i
60   lit   0  300
61   opr   0   10 <
62   jpc   0   67
63   int   0    3
64   lit   0    0
65   int   0   -4
66   cal   0   32 jloop
67   opr   0    0 return
68
68 begin
69   call iloop(0)
73 end.
68   int   0    5
69   int   0    3
70   lit   0    0
71   int   0   -4
72   cal   0   54 iloop
73   opr   0    0 return
b=62 p=15 (after out:=i+j+k;)
 65       2 k ********
 64      23 ret adr
 63      58 d.l.
 62      54 s.l.
 61       1 k
 60      45 ret adr
 59      54 d.l.
 58      54 s.l.
 57      20 j ********
 56      31 ret adr
 55      50 d.l.
 54      38 s.l.
 53       2 k
 52      23 ret adr
 51      46 d.l.
```

```
 50       42 s.l.
 49        1 k
 48       45 ret adr
 47       42 d.l.
 46       42 s.l.
 45       10 j
 44       67 ret adr
 43       38 d.l.
 42       38 s.l.
 41      200 i ********
 40       53 ret adr
 39       34 d.l.
 38        1 s.l.
 37       30 j
 36       31 ret adr
 35       30 d.l.
 34        6 s.l.
 33        2 k
 32       23 ret adr
 31       26 d.l.
 30       22 s.l.
 29        1 k
 28       45 ret adr
 27       22 d.l.
 26       22 s.l.
 25       20 j
 24       31 ret adr
 23       18 d.l.
 22        6 s.l.
 21        2 k
 20       23 ret adr
 19       14 d.l.
 18       10 s.l.
 17        1 k
 16       45 ret adr
 15       10 d.l.
 14       10 s.l.
 13       10 j
 12       67 ret adr
 11        6 d.l.
 10        6 s.l.
  9      100 i
  8       73 ret adr
  7        1 d.l.
  6        1 s.l.
  5  -999999 cvy
  4  -999999 cvx
  3        0 ret adr
  2        0 d.l.
  1        0 s.l.
start pl/0
111
112
121
122
211
212
221
222
```

5.4. Heap(-Dynamic) Allocation

Most flexible "temporally"

(Historical) Pascal implementation - same space as stack,

Buddy systems - maintain available blocks of size 2^k for various k
(aside - use Fibonacci sequence with arbitrary starting pair to approach 1.618 ratio)

One or many free lists?

Many - Start with list of closest adequate size, continuing through lists for larger sizes until non-empty list is found

One - ordered or unordered by size?, first fit or best fit?

Fragmentation

External - unallocated space between allocated blocks
Internal - extra space inside allocated block

Compaction is a possible when pointer flexibility is restricted for purposes of garbage collection.

5.5. Implementing Display as Alternative to Traversing Static Chain

Concepts:

Makes no assumption regarding locality of references

Instructions include *absolute* level and offset

No static chain traversals. . . since number of scope levels is small, each access references the *display*.

Trivial implementations use expensive approach of rebuilding the entire display for each call (just use the `base` loop).

(Caches on modern machines lessen the performance advantage of displays)

Gabbrielli

Every call saves and modifies one slot of the display.

Every return restores one slot of the display.

`pl0.display.3.js` at `http://ranger.uta.edu/~weems/NOTES3302/LAB1FALL14/` has details - see `base`, `cal` and return processing for differences

`http://ranger.uta.edu/~weems/NOTES3302/NEWNOTES/NOTES05/notes05.display.pl0`

Pascal-S aside (section 4 of report)

Every call saves and modifies one slot of the display.

Returns on callee side do not restore any display slots.

A nested caller for an outer procedure will restore as many display slots as levels - after return