Computer System Performance Modelling

Learning MAP Execs by Example

The purpose of this handout is to provide you with a very brief introduction to the use of MAP Execs. While rudimentary (and, some would say, "assembly-like"), MAP Execs provide a full power programming language within MAP. They can be very useful for doing repetitive studies over ranges of parameter values for example. Detailed information on the use of MAP Execs can be found in Chapter 7 of the User's Guide, and Chapters 3 and 4 and Appendix A of the Reference Manual.

MAP Execs can be created either within MAP, using the make_exec command, or with an arbitrary editor. Since no editting is possible during the creation of an exec within the make_exec command, the former approach is only useful for very short execs that can be entered without errors. In either case, the exec is stored in a file such as xxxxx.mapexec where xxxxx is the name by which the exec is to be known. If the file is stored in the directory from which you invoke MAP, or your home directory, or your sub-directory called "map-files", then it should be found when you invoke the xxxxx command within MAP.

The exec language includes all regular MAP commands plus integer and real variables, arrays (TABLE), assignment (SET), branch (GOTO), and conditional branches (IF). Also, execs can have character string parameters. MAP Execs can call other MAP Execs (although parameter passing can get tricky in that case).

Some standard MAP Execs, to which you all have access, are stored in the directory

/homes/kcs/map-files

The big ones include (1) INITDEVS, which encodes knowledge of specific device types that can be elicited from MAP by the "model" and "tdefine" commands, (2) demo, which goes through a simple demo of MAP, and (3) graph, which is the exed you used (unknowingly) in assignment #1 to produce graphs.

A few small ones are included below to illustrate how the exec language can be used in conjunction with standard MAP commands to create more powerful specialized procedures.

(a). quit
This exec simply terminates a MAP session by executing the exit command and confirming it:

```
------------------
quit.mapexec
------------------
exit
y
```

(b). select <cen>
This exec hides the output for all centers except the one specified by the parameter:

```
--------------------
select.mapexec
--------------------
{ Parameter: center name }
{ Turns output off for every center }
{  except the selected one. }
cen every output cen off
cen &p1 output cen on
```

(c). threshold <f>
This execs hides the output for all centers at which the utilization is below the threshold set by the parameter:

```
--------------------
threshold.mapexec
--------------------
{ Parameter: threshold utilization }
{ Turns output on for only those centers }
{ with utilization greater than the threshold. }
      CENTER EVERY
      OUTPUT CENTER OFF
      CLASS EVERY
      FOR ICENTER 1 &NCENTER
```

```
            IF GE &UTIL &P1 THEN
                OUTPUT CENTER ON
            ENDIF
        ENDFOR
```

(d). memwait <cls>
This exec calculates and prints the average time spent by customers of the specified class waiting for admission to their domain (ie, to obtain memory).

```
-------------------
memwait.mapexec
-------------------
class &p1
set f1 &response-(&insystem/&throughput)
writeln &f1
```

(e). calc <expr>
This exec calculates and prints the value of a real-valued expression:

```
----------------
calc.mapexec
---------------
{ Parameter: a real-valued expression }
{ Returns the value of the expression }
   set f1 &p1
   msgln &f1
```

(f). matchxto <adjclass> <targclass>
This exec adjusts the workload intensity of one class to match its throughput with that of another class, doing essentially binary search:

```
------------------
matchxto.mapexec
------------------
     { matchxto <adjclass> <targclass> }
     perf
     class &p1
     if ne &type 1 notbatch
     writeln error:  cant adjust a batch class.
     return
:notbatch
     class &p2
     set f1 &throughput
     class &p2
     writeln adjclass: &f1 targclass: &throughput
     if eq &type 2 terminal
     set f3 (&throughput-&f1)/&f1
     arrival &f1
     goto test
:terminal
     set f2 (&wi/&f1)-&response-&think
     set f3 &f2/&think
     think &f2+
:test
     if gt &f3 .01 again
     if lt &f3 -.01 again
     perf
```

(g). birthdeath

This exec solves the generalized birth-death equations for the proportion of time spent in each state in a single resource model. The exec prompts the user for (i) the value of N (most number of customers), and (ii) the arrival (birth) rates and the completion (death) rates in each state.

[NOTE: This exec uses for loops.]

```
------------------
birthdeath.mapexec
------------------
{ birthdeath -- solves a birth-death model }
{     (At most 50 states are allowed.) }
    set prompt n_max?
    set i1 &token
    set i2 &i1-1
    writeln enter birth rates for states 0 to &i2 :
  for i9 1 &i1
    set prompt arate?
    set index1 &i9
    set table1 &token
  endfor

    writeln enter death rates for states 1 to &i1 :
  for i9 1 &i1
    set prompt srate?
    set index2 &i9
    set table2 &token
  endfor

    set index3 1
    set table3 1
    set f8 1
    set f9 1
  for i9 1 &i1
    set index1 &i9
    set index2 &i9
    set index3 &i9+1
    set f9 (&table1/&table2)*
    set table3 &f9
    set f8 &f9+
  endfor

       set f6 0
       set f7 0
  for i9 1 &i1+1
       set index3 &i9
       set index2 &i9
       set table3 &f8/
       set f7 &table3+
       set f6 &table3*&table2+
       set i8 &i9-1
       writeln state: &i8 prob: &table3
  endfor

       writeln totprob: &f7
       if lt &nclass 1 return
       set iclass 1
       set f5 (&wi/&f6)-&think
       writeln throughput = &f6 ; response time = &f5
:return
       return
```

(h). emva

This exec solves a single class (terminal type) queueing network model using exact Mean Value Analysis. It prompts for the required parameter values.

[NOTE: This exec uses for loops and if-then-else statements.]

```
------------------
emva.mapexec
------------------
{ emva -- Exact MVA solver for a single terminal class }
{   Variable usage:                    }
{        i1  -  number of centers      }
{        i2  -  class population       }
{        f3  -  think time             }
{        i4  -  center index           }
{        i5  -  population index       }
{        f1  -  response time          }
{        f2  -  throughput             }
{        f9  -  calculation variable   }
{    table1  -  center loadings        }
{    table2  -  queue lengths          }
{    table3  -  residence times        }
    set iclass 1
    tablize
    set i1 &ncenter
    set i2 &wi
    set f3 &think
    set i7 0  { 1 ==> details; 0 ==> none }
    for i4 1 &i1
        set index2 &i4
        set table2 0
    endfor
    format t 8 a 12 a 12 a 12 endformat
    writeln pop tput resp
    for i5 1 &i2
       set f1 0
       for i4 1 &i1
          set index2 &i4
          set f9 &table2
          set f9 1+
          set index1 &i4
          set f9 &table1*
          set index3 &i4
          set table3 &f9
          if ne &i7 0 then
             format a 5 i 4 t 12 a 5 f 10 5 endformat
             writeln cen &i4 resid &f9
          endif
          set f1 &f9+
       endfor
       set f2 &i5/(&f1+&f3)
       if ne &i7 0 then
          format a 8 f 10 5 t 24 a 8 f 10 5 endformat
          writeln resp &f1 xput &f2
       endif
       for i4 1 &i1
          set index2 &i4
          set index3 &i4
          set table2 &table3*&f2
          if ne &i7 0 then
             format a 5 i 4 t 12 a 5 f 10 5 endformat
             writeln cen &i4 qlen &table2
          endif
       endfor
       format i 12 f 12 5 f 12 5 endformat
```

```
      writeln &i5 &f2 &f1
   endfor
   format endformat
```

(i). amva

This exec solves a single class (terminal type) queueing network model using approximate Mean Value Analysis. It prompts for the required parameter values.

[NOTE: This exec uses for loops and if-then-else statements.]

```
------------------
amva.mapexec
------------------
{ amva -- Approximate MVA solver for a single terminal class }
{   Variable usage:                    }
{        i1  -  number of centers      }
{        i2  -  class population       }
{        f3  -  think time             }
{        i4  -  center index           }
{        i5  -  population index       }
{        f1  -  response time          }
{        f2  -  throughput             }
{        f9  -  calculation variable   }
{    table1  -  center loadings        }
{    table2  -  queue lengths          }
{    table3  -  residence times        }
   set iclass 1
   tablize
   set i1 &ncenter
   set i2 &wi
   set f3 &think
   set i7 0  { 1 ==> details; 0 ==> none }
   for i4 1 &i1
      set index2 &i4
      set table2 &i2/&i1
   endfor
   format t 8 a 12 a 12 a 12 endformat
   writeln iter tput resp
   for i5 1 12
      set f1 0
      for i4 1 &i1
         set index2 &i4
         set f9 &table2
         set f9 ((&i2-1)/&i2)*
         set f9 1+
         set index1 &i4
         set f9 &table1*
         set index3 &i4
         set table3 &f9
         if ne &i7 0 then
            format a 5 i 4 t 12 a 5 f 10 5 endformat
            writeln cen &i4 resid &f9
         endif
         set f1 &f9+
      endfor
      set f2 &i2/(&f1+&f3)
      if ne &i7 0 then
         format a 8 f 10 5 t 24 a 8 f 10 5 endformat
         writeln resp &f1 xput &f2
      endif
      for i4 1 &i1
         set index2 &i4
         set index3 &i4
         set table2 &table3*&f2
         if ne &i7 0 then
```

```
         format a 5 i 4 t 12 a 5 f 10 5 endformat
         writeln cen &i4 qlen &table2
      endif
   endfor
   format i 12 f 12 5 f 12 5 endformat
   writeln &i5 &f2 &f1
endfor
format endformat
```