```
1      mvms:  procedure options(main) ;
2
3         /*************************************************************/
4         /*     Purpose:  This is a simulation program. This program */
5         /*               simulates a process scheduler as it        */
6         /*               performs its tasks for a mock up           */
7         /*               operation system called mvms (mock up      */
8         /*               virtual memory system).  This program      */
9         /*               will simulate the OS functions of job      */
10        /*               scheduling by readding input commands      */
11        /*               which will instruct this program.          */
12        /*                                                          */
13        /*       Input:  sysin  - file sysctr.dat                   */
14        /*                                                          */
15        /*      Output:  sysout - file dump.dat                     */
16        /*               systat - file summary.dat                  */
17        /*               syserr - file error.dat                    */
18        /*                                                          */
19        /*      Author:  Garry R. Daly                              */
20        /*                                                          */
21        /*************************************************************/
22
23      /* Define the Preprocessor variables */
24
25    %replace $MAXTICK by 1638;
26    %replace $TICKMS by 20;
27    %replace $TICKMAX by 4;
28
29    %replace $NPRI by 4;
30    %replace $NWAIT by 12;
31    %replace $MAXTICK by 99999;
32
33    %replace $QLIMIT by 3;
34
35    %replace $IDMAX by 3;
36    %replace $PTOKMAX by 1;
37    %replace $ETOKMAX by 5;
38    %replace $WAITMAX by 2;
39    %replace $WLIMIT by 1;
40
41    %replace $RLIMIT by 1;
42    %replace $RDYSERV by 400;
43    %replace $MAXUPGE by 2;
44
45    %replace $QUAN1 by 320;
46    %replace $QUAN2 by 160;
47    %replace $QUAN3 by 100;
48    %replace $QUAN4 by 60;
49
50          dcl   null                builtin ,
51                length              builtin ;
52
```

```
53        dcl   sysin              file input stream ,
54              sysout             file print ,
55              systat             file print ,
56              syserr             file print ;
57
58        dcl   input_string       char(80)      varying ,
59              command            char(5)       varying ,
60              pcbptr             pointer ;
61
62        dcl   i                  fixed bin(15) ,
63              pos_in_string      fixed bin(15) ,
64              tot_jobs           fixed bin(15) init (0) ,
65              tot_time           fixed bin(15) init (0) ,
66              err_count          fixed bin(15) init (0) ,
67              tot_act_time       fixed bin(15) init (0) ,
68              old_count          fixed bin(15) ;
69
70        dcl   request            char(7)       init ('request') ,
71              release            char(7)       init ('release') ;
72
73        dcl   bad_cmd            char(80)      init
74                                 ('Error:  Invalid command found: ') ;
75
76        dcl   early_stop         char(80)      init
77                        ('Error:  Program terminated prematurely: ') ;
78
79        dcl 1 holdq($NPRI) ,
80              2 front_ptr        pointer ,
81              2 back_ptr         pointer ,
82              2 no_of_entries    fixed bin(15) ;
83
84        dcl 1 readyq($NPRI) ,
85              2 front_ptr        pointer ,
86              2 back_ptr         pointer ,
87              2 no_of_entries    fixed bin(15) ;
88
89        dcl 1 runq($NPRI) ,
90              2 front_ptr        pointer ,
91              2 back_ptr         pointer ,
92              2 no_of_entries    fixed bin(15) ;
93
94        dcl 1 waitq($NWAIT) ,
95              2 front_ptr        pointer ,
96              2 back_ptr         pointer ,
97              2 no_of_entries    fixed bin(15) ;
98
99        dcl 1 doneq ,
100             2 front_ptr        pointer ,
101             2 back_ptr         pointer ,
102             2 no_of_entries    fixed bin(15) ;
103
104       /* initialize the data structures */
```

```
105
106         do i=1 to $NPRI ;
107
108           holdq(i).front_ptr      = null() ;
109           holdq(i).back_ptr       = null() ;
110           holdq(i).no_of_entries  = 0      ;
111
112           readyq(i).front_ptr     = null() ;
113           readyq(i).back_ptr      = null() ;
114           readyq(i).no_of_entries = 0      ;
115
116           runq(i).front_ptr       = null() ;
117           runq(i).back_ptr        = null() ;
118           runq(i).no_of_entries   = 0      ;
119
120         end ;
121
122         do i=1 to $NWAIT ;
123
124           waitq(i).front_ptr      = null() ;
125           waitq(i).back_ptr       = null() ;
126           waitq(i).no_of_entries  = 0      ;
127
128         end ;
129
130         pcbptr = null() ;
131
132         doneq.front_ptr           = null() ;
133         doneq.back_ptr            = null() ;
134         doneq.no_of_entries       = 0      ;
135
136         /* open the files and assign names to them. */
137
138       /**/ open file (debug) title ('debug.dat') ;
139       /**/ dcl debug print file ;
140         open file (sysin)  title ('sysctr.dat')  ;
141         open file (sysout) title ('dump.dat')    ;
142         open file (systat) title ('summary.dat') ;
143         open file (syserr) title ('error.dat')   ;
144
145         on endfile (sysin)
146           begin ;
147             call inerr(early_stop, ' ', err_count) ;
148             stop ;
149           end ;
150
151         /* read an input line */
152
153         get file (sysin) edit (input_string) (a(80)) ;
154
155         /* An 80 character line has just been read.  From this */
156         /* line (input_line), parse the command within it.     */
```

```
157         /* The command will be the first token of the input    */
158         /* line.                                               */
159
160         call gtoken(1, input_string, command, pos_in_string) ;
161
162     /**/ dcl 1 msgs ,
163     /**/      2 text char(30) init ('Just read a card') ;
164     /**/ dcl 1 cardin ,
165     /**/  2 cards char(80) varying ;
166     /**/ put file (debug) skip(2) edit (msgs) (a) ;
167     /**/ cardin.cards = input_string ;
168     /**/ put file (debug) skip(1) edit (cardin) (a) ;
169         /* Loop continuously while there are command tokens.   */
170
171         do while (command ^= 'HALT') ;
172           select ;
173
174             /* Validate the input command */
175
176             when (command = 'JOB')
177               do ;
178                 old_count = err_count ;
179                 call jobber(holdq, input_string, pos_in_string,
180                             err_count, tot_jobs) ;
181                 if old_count = err_count
182                   then do ;
183
184                     /* call the long term scheduler */
185
186                     call lgsched(holdq, readyq, runq) ;
187
188                    /* call the short term scheduler */
189
190                     call priorck(holdq, readyq, runq) ;
191                   end ;
192               end ;
193
194             when (command = 'TICK')
195               call clock(input_string, pos_in_string, err_count,
196                          holdq, readyq, runq, waitq, doneq,
197                          tot_time, tot_act_time) ;
198
199             when (command = 'REQ' | command = 'REL')
200               do ;
201
202                 /* This program is set up to handle future growth  */
203                 /* of sharing resources and having more then 1 job */
204                 /* active on the cpu at one time.                  */
205
206                 if command = 'REQ'
207                   then call resrc(request, input_string, pos_in_string,
208                                   err_count, readyq, holdq, waitq,
```

```
                                runq) ;

                  else call resrc(release, input_string, pos_in_string,
                                err_count, readyq, holdq, waitq,
                                runq) ;
              end ;

            when (command = 'DUMP')
              call dumper(tot_time, runq, readyq, holdq,
                          waitq, doneq) ;

            otherwise
              call inerr(bad_cmd, input_string, err_count) ;

          end ;   /* end of select */

          get file (sysin) edit (input_string) (a(80)) ;
    /**/ put file (debug) skip(2) edit (msgs) (a) ;
    /**/ cardin.cards = input_string ;
    /**/ put file (debug) skip(1) edit (cardin) (a) ;
          call gtoken(1, input_string, command, pos_in_string) ;

        end ;   /* end do while */

        /* Print the final statistics */

        call sumprt(doneq, tot_jobs, tot_time, tot_act_time,
                    err_count) ;

        stop ;
```

```
gtoken:  procedure(search_start_pos, input_string, token_ret,
                   token_end_pos) ;

   /*************************************************************/
   /* This procedure searches an input string from a search     */
   /* start position for a token which is terminated by a blank. */
   /* This procedure will return the token and the position     */
   /* within the input string where the token terminated.       */
   /*************************************************************/

   dcl   substr             builtin ,
         length             builtin ;

   dcl   search_start_pos   fixed bin(15) ,
         input_string       char(80)      varying ,
         token_ret          char(*)       varying ,
         token_end_pos      fixed bin(15) ;

   dcl   beginning          fixed bin(15) ,
         ending             fixed bin(15) ,
         str_length         fixed bin(15) ;

   /* Initialize return values */

   token_ret = ''    ;
   token_end_pos = 0 ;

   /* Find the start of the token -- throw away blanks */

   str_length = length(input_string) ;
   if search_start_pos > str_length
     then return ;   /* error */

   beginning = search_start_pos ;
   do while (substr(input_string, beginning, 1) = ' ') ;
     beginning = beginning + 1 ;
     if beginning > str_length
       then return ;   /* error */
   end ;

   /* Now that the beginning is known, find the end of the */
   /* string.                                              */

   ending = beginning ;
   do while ((substr(input_string, ending, 1) ^= ' ') &
             (ending < str_length)) ;
     ending = ending + 1 ;
   end ;

   /* Check if the ending character is truly a blank. */
   /* If it is, then you found the end of the token.  */
   /* If it is not, then your at the end of the       */
```

```
294         /* input string and have the last token.           */
295
296         if substr(input_string, ending, 1) = ' '
297           then token_end_pos = ending - 1 ;  /* complete token */
298           else token_end_pos = ending ;      /* incomplete token */
299
300         token_ret = substr(input_string,beginning,
301                            token_end_pos - beginning + 1) ;
302       end gtoken ;
303
304
305
```

```
charnumb:  procedure(input_string, ret_type, ret_bin, ret_dec) ;

   /*********************************************************/
   /* This function takes a character string and converts it   */
   /* to either a fixed bin(7) number or a fixed dec(6)        */
   /* depending on how the ret_type flag is set.  If the flag  */
   /* is set to '1'b then binary is returned, otherwise a      */
   /* fixed decimal number is returned.                        */
   /*********************************************************/

   dcl   length             builtin ,
         substr             builtin ;

   dcl   numbers(0:9)       char(1)        init ('0', '1', '2', '3',
                                                '4', '5', '6', '7',
                                                '8', '9');

   dcl   input_string       char(80)       varying,
         i                  fixed bin(7) ,
         j                  fixed bin(7) ,
         k                  fixed dec(6) ;

   dcl   exp_bin            fixed bin(7) ,
         exp_dec            fixed dec(6) ,
         ret_bin            fixed bin(7) ,
         ret_dec            fixed dec(6) ,
         ret_type           bit(1)       ,
         one_char           char(1) ;

   ret_bin = 0 ;
   ret_dec = 0 ;

   if ret_type              /* if ret_type = '1'b */
     then do ;              /* return binary      */

       exp_bin  = length(input_string) - 1 ;

       do i=1 to length(input_string) ;
         one_char = substr(input_string, i, 1) ;
         do j=0 to 9 while (one_char ^= numbers(j)) ;
         end ;

         ret_bin = ret_bin + (j * ((10)**(exp_bin))) ;
       end ;
     end ;

     else do ;

       exp_dec  = length(input_string) - 1 ;

       do i=1 to length(input_string) ;
         one_char = substr(input_string, i, 1) ;
```

```
358             do k=0 to 9 while (one_char ^= numbers(k)) ;
359             end ;
360
361             ret_dec = ret_dec + (k * ((10)**(exp_dec))) ;
362             exp_dec = exp_dec - 1 ;
363           end ;
364         end ;
365
366     end charnumb ;
367
368
```

```
369     inerr:  procedure(message, input_string, err_count) ;
370
371        /*********************************************************/
372        /* This procedure writes out error messages to an error  */
373        /* file and updates the error count.                     */
374        /*********************************************************/
375
376        dcl   message            char(80),
377              input_string       char(80)     varying ,
378              err_count          fixed bin(15) ;
379
380        dcl   syserr             print file ;
381
382        dcl 1 header ,
383              2 fill1            char(22)      init (' ') ,
384              2 msg              char(36)      init
385                              ('MVMS Operating System Error Listing') ,
386              2 fill2            char(22)      init (' ') ;
387
388        dcl 1 output ,
389              2 out_msg          char(80) ;
390
391        if err_count = 0
392          then put file (syserr) skip(1) edit (header) (a) ;
393
394        err_count = err_count + 1 ;
395
396        output.out_msg = message ;
397        put file (syserr) skip(2) edit (output) (a) ;
398        output.out_msg = input_string ;
399        put file (syserr) skip(1) edit (output) (a) ;
400
401     end inerr ;
402
403
```

```
jobber:  procedure(holdq, input_string, pos_in_string, err_count,
                   tot_jobs) ;

    /*********************************************************/
    /* This procedure creates a pcb for new job entering the */
    /* operating system and assigns this pcb to the holdq.   */
    /*********************************************************/

    dcl   length             builtin ,
          verify             builtin ,
          substr             builtin ;

    dcl   tot_jobs           fixed bin(15) ,
          back               bit(1)        init ('0'b) ;

    dcl   input_string       char(80)      varying ,
          pos_in_string      fixed bin(15) ,
          err_count          fixed bin(15) ;

    dcl   token_ret          char(80)      varying,
          token_end_pos      fixed bin(15) ;

    dcl   jobid              char(3) ,
          priority           fixed bin(7) ,
          est_cpu_tm         fixed dec(6) ;

    dcl   want_bin           bit(1)        init ('1'b) ,
          want_dec           bit(1)        init ('0'b) ;

    dcl   ret_bin            fixed bin(7) ,
          ret_dec            fixed dec(6) ;

    dcl   digits             char(10)       init ('0123456789') ;

    dcl   tmp_ptr            pointer ;

    dcl 1 holdq($NPRI) ,
          2 front_ptr        pointer ,
          2 back_ptr         pointer ,
          2 no_of_entries    fixed bin(15) ;

    dcl 1 pcb                based(pcbptr) ,
          2 job_id           char(3) ,
          2 state            char(8)       varying ,
          2 init_pri         fixed bin(7) ,
          2 curr_pri         fixed bin(7) ,
          2 init_tm          fixed dec(6) ,
          2 rem_tm           fixed dec(6) ,
          2 turn_tm          fixed dec(6) ,
          2 act_tm           fixed dec(6) ,
          2 ready_tm         fixed dec(6) ,
          2 next_pcb         pointer ;
```

```
456
457         dcl   job_err1           char(80)       init
458                             ('Error:  No job id found in input string') ;
459
460         dcl   job_err2           char(80)       init
461                ('warning:  job id too long truncation has occurred') ;
462
463         dcl   pri_err1           char(80)       init
464                           ('Error:  No priority found in input string') ;
465
466         dcl   pri_err2           char(80)       init
467                                      ('Error:  Priority too long') ;
468
469         dcl   pri_err3           char(80)       init
470                                       ('Error:  Priority not numeric') ;
471
472         dcl   pri_err4           char(80)       init
473                                      ('Error:  Priority out of range') ;
474
475         dcl   est_err1           char(80)       init
476               ('Error:  Estimated cpu time not found in input string') ;
477
478         dcl   est_err2           char(80)       init
479                                 ('Error:  Estimated cpu time too long') ;
480
481         dcl   est_err3           char(80)       init
482                                ('Error:  Estimated cpu time not numeric') ;
483
484         dcl   est_err4           char(80)       init
485                             ('Error:  Estimated cpu time out of range') ;
486
487         /* Retrieve the job id, priority and estimated cpu time */
488         /* tokens from the input string.                        */
489
490         /* Job id token. */
491         pos_in_string = pos_in_string + 1 ;   /* First spot after token. */
492         call gtoken (pos_in_string, input_string, token_ret,
493                      token_end_pos) ;
494         if length(token_ret) < 1      /* No token found. */
495           then do ;
496             call inerr(job_err1, input_string, err_count) ;
497             return ;
498           end;
499
500         if length(token_ret) > $IDMAX      /* Token too long */
501           then call inerr(job_err2, input_string, err_count) ;
502
503         jobid = substr(token_ret, 1, $IDMAX) ;
504
505         /* Priority token. */
506
507         pos_in_string = token_end_pos + 1 ;   /* First spot after token */
```

```
508         call gtoken(pos_in_string, input_string, token_ret,
509                     token_end_pos) ;
510
511         if length(token_ret) < 1     /* No token found. */
512           then do ;
513             call inerr(pri_err1, input_string, err_count) ;
514             return ;
515           end ;
516
517         if length(token_ret) > $PTOKMAX   /* Token too long. */
518           then do ;
519             call inerr(pri_err2, input_string, err_count) ;
520             return ;
521           end ;
522
523         /* Convert token from character to numeric and check */
524         /* its range.                                        */
525
526         if verify(token_ret, digits) > 0
527           then do ;                       /* Token not numeric. */
528             call inerr(pri_err3, input_string, err_count) ;
529             return ;
530           end ;
531
532         call charnumb(token_ret, want_bin, priority, ret_dec) ;
533
534         if priority > $NPRI | priority < 1   /* Priority exceeds maximum.
535   */
536           then do ;
537             call inerr(pri_err4, input_string, err_count) ;
538             return ;
539           end ;
540
541         /* Estimated cpu token. */
542
543         pos_in_string = token_end_pos + 1 ;   /* First spot after token. */
544         call gtoken(pos_in_string, input_string, token_ret,
545                     token_end_pos) ;
546
547         if length(token_ret) < 1     /* No token found */
548           then do ;
549             call inerr(est_err1, input_string, err_count) ;
550             return ;
551           end ;
552
553         if length(token_ret) > $ETOKMAX     /* Token too long. */
554           then do ;
555             call inerr(est_err2, input_string, err_count) ;
556             return ;
557           end ;
558
559         /* Convert token from character to numeric and check */
```

```
560         /* its range.                                          */
561
562         if verify(token_ret, digits) > 0     /* Token not numeric. */
563           then do ;
564             call inerr(est_err3, input_string, err_count) ;
565             return ;
566           end ;
567
568         call charnumb(token_ret, want_dec, ret_bin, est_cpu_tm) ;
569         if est_cpu_tm > $MAXTICK | est_cpu_tm < 0
570           then do ;
571             call inerr(est_err4, input_string, err_count) ;
572             return ;
573           end ;
574
575         /* Allocate a pcb seeing we have good data. */
576
577         allocate pcb ;
578         pcbptr->pcb.job_id   = jobid    ;
579         pcbptr->pcb.state    = 'HOLD'   ;
580         pcbptr->pcb.init_pri = priority ;
581         pcbptr->pcb.curr_pri = pcbptr->pcb.init_pri ;
582         pcbptr->pcb.init_tm  = est_cpu_tm ;
583         pcbptr->pcb.rem_tm   = pcbptr->pcb.init_tm ;
584         pcbptr->pcb.turn_tm  = 0 ;
585         pcbptr->pcb.act_tm   = 0 ;
586         pcbptr->pcb.ready_tm = 0 ;
587
588         /* Now assign the pcb to the hold queue. */
589
590         call isrtq(holdq(priority), back, pcbptr) ;
591
592         tot_jobs = tot_jobs + 1 ;
593
594       end jobber ;
595
596
```

```
lgsched:  procedure(holdq, readyq, runq) ;

   /*************************************************************/
   /* This procedure is the long term scheduler.  Its primary   */
   /* function is to examine the hold queue for processes       */
   /* waiting to execute and move them from the hold queue to   */
   /* the ready queue so that they can be selected to be run    */
   /* by the short term scheduler.                              */
   /*                                                           */
   /* This procedure will transfer a job from the hold queue    */
   /* to the appropriate ready queue dictated by the job's      */
   /* priority.  If the queue is full, the job is placed in     */
   /* the next higher priority queue.  If this queue is also    */
   /* full, the job is not transferred.                         */
   /*************************************************************/


   dcl 1 holdq($NPRI) ,
         2 front            pointer       ,
         2 back             pointer       ,
         2 no_of_entries    fixed bin(15) ;

   dcl 1 readyq($NPRI) ,
         2 front_ptr        pointer       ,
         2 back_ptr         pointer       ,
         2 no_of_entries    fixed bin(15) ;

   dcl 1 runq($NPRI) ,
         2 front_ptr        pointer       ,
         2 back_ptr         pointer       ,
         2 no_of_entries    fixed bin(15) ;

   dcl   pcbptr             pointer       ,
         tmp_ptr            pointer       ,
         priority           fixed bin(7)  ;

   dcl 1 pcb                based(pcbptr) ,
         2 job_id           char(3)       ,
         2 state            char(8)       varying ,
         2 init_pri         fixed bin(7) ,
         2 curr_pri         fixed bin(7) ,
         2 init_tm          fixed dec(6) ,
         2 rem_tm           fixed dec(6) ,
         2 turn_tm          fixed dec(6) ,
         2 act_tm           fixed dec(6) ,
         2 ready_tm         fixed dec(6) ,
         2 next_pcb         pointer       ;

   dcl   front              bit(1)        init ('1'b) ,
         back               bit(1)        init ('0'b) ;

   priority = $NPRI ;
```

```
649
650         /* Loop until priority is not zero. */
651
652         do while (priority > 0) ;
653
654           if holdq(priority).no_of_entries > 0
655             then do ;
656
657               /* There are jobs in the holdq at this priority. */
658               /* Now see if there is room in the readyq.       */
659
660               if readyq(priority).no_of_entries < $QLIMIT
661                 then do ;
662
663                   /* Move the job in the front of the holdq */
664                   /* to the back of the readyq.             */
665
666                   holdq(priority).front_ptr->pcb.state = 'READY' ;
667                   call dletq(holdq(priority), front, tmp_ptr) ;
668                   call isrtq(readyq(priority), back, tmp_ptr) ;
669                   call priorck(holdq, readyq, runq) ;
670                 end ;     /* End holdq -> readyq */
671
672                 else if priority < $NPRI
673                   then do ;
674
675                     /* Here we get a chance to check a higher         */
676                     /* priority queue for an open slot.  If a slot is */
677                     /* available, we move the lower priority job into */
678                     /* this higher priority readyq. We will bump the  */
679                     /* priority.                                      */
680
681                     if readyq(priority+1).no_of_entries < $QLIMIT
682                       then do ;          /* Move job to higher readyq */
683                         holdq(priority).front_ptr->pcb.curr_pri =
684                                                       priority + 1 ;
685                         holdq(priority).front_ptr->pcb.ready_tm = 0    ;
686                         holdq(priority).front_ptr->pcb.state = 'READY'  ;
687                         call dletq(holdq(priority), front, tmp_ptr) ;
688                         call isrtq(readyq(priority+1), back, tmp_ptr) ;
689                         call priorck(holdq, readyq, runq) ;
690                       end;      /* end holdq+1 -> ready+1 */
691                       else priority = priority - 1 ;
692
693                   end ;         /* End 2nd chance */
694
695                   else priority = priority - 1 ;
696             end ;               /* End holdq has entries */
697
698             else priority = priority - 1 ;
699
700         end ;                   /* End do while */
```

```
701
702       end lgsched ;
703
704
```

```
705     priorck:  procedure (holdq, readyq, runq) ;
706
707        /*************************************************************/
708        /* This procedure is the MVMS OS short term scheduler.       */
709        /* This procedure will check the runq priority versus the    */
710        /* readyq in order to attempt to preempt jobs running with   */
711        /* a lower priority then one waiting in the readyq.  If the  */
712        /* cpu has idle capacity, jobs will be submitted to the      */
713        /* running state.                                            */
714        /*************************************************************/
715
716        dcl 1 holdq($NPRI) ,
717              2 front_ptr        pointer       ,
718              2 back_ptr         pointer       ,
719              2 no_of_entries    fixed bin(15) ;
720
721        dcl     tmp_ptr          pointer       ,
722                pcbptr           pointer       ;
723
724        dcl 1 readyq($NPRI) ,
725              2 front_ptr        pointer       ,
726              2 back_ptr         pointer       ,
727              2 no_of_entries    fixed bin(15) ;
728
729        dcl 1 runq($NPRI) ,
730              2 front_ptr        pointer       ,
731              2 back_ptr         pointer       ,
732              2 no_of_entries    fixed bin(15) ;
733
734        dcl 1 waitq($NWAIT) ,
735              2 front_ptr        pointer       ,
736              2 back_ptr         pointer       ,
737              2 no_of_entries    fixed bin(15) ;
738
739        dcl   i                  fixed bin(7)  ,
740              act_jobs_found     fixed bin(15) ,
741              run_jobs_found     fixed bin(15) ;
742
743        dcl   ready_pri          fixed bin(7) ,
744              run_pri            fixed bin(7)  ,
745              use_runq           bit(1)         init('1'b) ;
746
747        dcl 1 pcb                based(pcbptr) ,
748              2 job_id           char(3)       ,
749              2 state            char(8)       varying,
750              2 init_pri         fixed bin(7) ,
751              2 curr_pri         fixed bin(7) ,
752              2 init_tm          fixed dec(6) ,
753              2 rem_tm           fixed dec(6) ,
754              2 turn_tm          fixed dec(6) ,
755              2 act_tm           fixed dec(6) ,
756              2 ready_tm         fixed dec(6) ,
```

```
      2 next_pcb         pointer       ;

dcl   front              bit(1)        init ('1'b) ,
      back               bit(1)        init ('0'b) ;

/* Determine if the cpu has idle capacity. */

act_jobs_found = 0 ;
do i=1 to $NPRI ;
  act_jobs_found  = act_jobs_found + runq(i).no_of_entries ;
end ;

/* Now active_jobs has the number of active jobs running. */
/* If this value is less than $RLIMIT (run limit), then   */
/* we have idle capacity.                                 */

/* Loop through the runq and readyq until a readyq job is */
/* found with a priority higher than a job in the runq.   */
/* Once a match is found, preempt the runq job.           */

/* One for one preemption */

run_jobs_found = 0 ;         /* Flag to stop search early */
ready_pri = $NPRI     ;
run_pri   = $NPRI - 1 ;

do while (ready_pri > 1 & run_jobs_found < $RLIMIT) ;

  if readyq(ready_pri).no_of_entries > 0
    &  runq(run_pri).no_of_entries > 0

    then do ;

      /* Both queues must have entries & the readyq          */
      /* must be a higher priority in order to preempt.      */
      /* Move runq job to readyq.  Swap readyq job to        */
      /* holdq, if necessary.                                */

      call preempt(runq(run_pri).front_ptr, run_pri, run_pri,
                   use_runq, holdq, readyq, runq, waitq) ;

      /* Delete job from readyq since it is now running. */

      call dletq(readyq(ready_pri), front, tmp_ptr) ;

      /* Insert higher readyq job into runq. */

      tmp_ptr->pcb.ready_tm = 0 ;
      tmp_ptr->pcb.act_tm   = 0 ;
      tmp_ptr->pcb.state    = 'RUNNING' ;
      call isrtq(runq(ready_pri), back, tmp_ptr) ;
```

```
809             run_jobs_found = run_jobs_found + 1  ;
810
811           end ;      /* End then do */
812
813           else do ;
814                     /* Decrement priorities no preemption yet . */
815             run_pri   = run_pri   - 1 ;
816             if run_pri < 1
817               then do ;
818                 ready_pri = ready_pri - 1 ;
819                 run_pri   = ready_pri - 1 ;
820               end ;
821           end ;
822
823       end ;          /* End do while */
824
825       /* If the cpu has idle capacity, insert jobs from the readyq */
826       /* to the runq.                                               */
827
828       ready_pri = $NPRI ;
829       do while (act_jobs_found < $RLIMIT & ready_pri > 0) ;
830
831         if readyq(ready_pri).no_of_entries > 0
832           then do ;
833
834             /* Delete from readyq. */
835
836             call dletq(readyq(ready_pri), front, tmp_ptr) ;
837
838             /* Transfer readyq job to runq. */
839
840             tmp_ptr->pcb.state    = 'RUNNING' ;
841             tmp_ptr->pcb.act_tm   = 0 ;
842             tmp_ptr->pcb.ready_tm = 0 ;
843             call isrtq(runq(ready_pri), back, tmp_ptr) ;
844             act_jobs_found = act_jobs_found + 1 ;
845
846           end ;
847
848           else ready_pri = ready_pri - 1 ;
849
850
851       end ;          /* End do while */
852
853   end priorck ;
854
855
```

```
preempt:  procedure(pcbptr, pres_loc, fut_loc, q_type,
                    holdq, readyq, runq, waitq) ;

   /*************************************************************/
   /* This procedure preempts jobs running from either the runq  */
   /* or the waitq.  The jobs are moved to the readyq.  Four     */
   /* parameters are required by this procedure along with the   */
   /* holdq, readyq, and the runq or waitq.                      */
   /* The pcbptr is the address of the job at the front of the q */
   /* which is being moved to the readyq.  The pres_loc is the   */
   /* present priority of the job in the q.  The fut_loc is the  */
   /* new priority of the job in the q moving to the readyq.     */
   /* The q_type is the location of the pcb in the q  .          */
   /* This location is the priority in a runq, but in the waitq, */
   /* this is the device number location in the queue.  The      */
   /* fut_loc is higher when a job when moving out of the waitq. */
   /* The fut_loc is lower when a jobs quantum has expired.      */
   /* The fut_loc is same as the pres_pri when the queues are    */
   /* being checked by the short term scheduler.                 */
   /*************************************************************/

   dcl 1 pcb                based(pcbptr) ,
         2 job_id           char(3) ,
         2 state            char(8)       varying ,
         2 init_pri         fixed bin(7) ,
         2 curr_pri         fixed bin(7) ,
         2 init_tm          fixed dec(6) ,
         2 rem_tm           fixed dec(6) ,
         2 turn_tm          fixed dec(6) ,
         2 act_tm           fixed dec(6) ,
         2 ready_tm         fixed dec(6) ,
         2 next_pcb         pointer       ;

   dcl   front              bit(1)        init('1'b) ,
         back               bit(1)        init('0'b) ;

   dcl 1 holdq($NPRI),
         2 front_ptr        pointer       ,
         2 back_ptr         pointer       ,
         2 no_of_entries    fixed bin(15) ;

   dcl 1 readyq($NPRI) ,
         2 front_ptr        pointer       ,
         2 back_ptr         pointer       ,
         2 no_of_entries    fixed bin(15) ;

   dcl 1 runq($NPRI) ,
         2 front_ptr        pointer       ,
         2 back_ptr         pointer       ,
         2 no_of_entries    fixed bin(15) ;

   dcl 1 waitq($NWAIT) ,
```

```
908             2 front_ptr        pointer       ,
909             2 back_ptr         pointer       ,
910             2 no_of_entries    fixed bin(15) ;
911
912         dcl   tmp_ptr          pointer       ,
913               last_pcbptr      pointer       ;
914
915         dcl   pcbptr           pointer       ,
916               q_type           bit(1)         ,
917               pres_loc         fixed bin(7)  ,
918               fut_loc          fixed bin(7)  ;
919
920     /**/ dcl jj fixed bin(7) ;
921     /**/ dcl 1 errmsg1 ,
922     /**/     2 msg char (50) init('     * * ENTRY in preempt');
923     /**/ put file (debug) skip edit (errmsg1) (a) ;
924     /**/ dcl 1 errmsg2 ,
925     /**/ 2 msg char(50) init ('leaving preempt') ;
926     /**/ do jj=1 to $NPRI ;
927     /**/  call debugger(jj, holdq(jj), 'hold') ;
928     /**/ end ;
929     /**/ do jj=1 to $NPRI ;
930     /**/ call debugger(jj, readyq(jj), 'readyq') ;
931     /**/ end ;
932     /**/ do jj=1 to $NPRI ;
933     /**/ call debugger(jj, runq(jj), 'runq') ;
934     /**/ end ;
935     /**/ do jj=1 to $NWAIT ;
936     /**/ call debugger(jj, waitq(jj), 'waitq') ;
937     /**/ end ;
938
939        /* Check if future priority is lower than initial */
940        /* priority.  If it is, adjust it to be pres_pri. */
941
942        if fut_loc < pcbptr->pcb.init_pri
943          then fut_loc = pcbptr->pcb.init_pri ;
944
945        if readyq(fut_loc).no_of_entries = $QLIMIT
946          then do ;
947
948            /* In this preemption case, the readyq is full so a */
949            /* job will have to transfer back to the holdq.     */
950
951            /* readyq to holdq */
952
953            readyq(fut_loc).back_ptr->pcb.state = 'HOLD' ;
954            call dletq(readyq(fut_loc), back, tmp_ptr) ;
955            call isrtq(holdq(fut_loc), front, tmp_ptr) ;
956
957          end ;
958
959         /* Determine which queue you will be dealing with */
```

```
960
961         if q_type        /* if q_type = '1'b */
962           then do ;      /*      RUNQ ONLY   */
963
964             /* Find the pcb in the runq and move it to the readyq */
965             /* also update pcb status.                            */
966
967             pcbptr->pcb.ready_tm = 0       ;
968             pcbptr->pcb.state    = 'READY' ;
969             pcbptr->pcb.curr_pri = fut_loc ;
970
971             /* Find pcb in the runq. */
972
973             if pcbptr = runq(pres_loc).front_ptr
974               then do ;
975
976                 /* Front of queue. */
977
978                 call dletq(runq(pres_loc), front, tmp_ptr) ;
979                 call isrtq(readyq(fut_loc), back, tmp_ptr) ;
980
981               end ;
982
983               else if pcbptr = runq(pres_loc).back_ptr
984                 then do ;
985
986                   /* Back of queue. */
987
988                   call dletq(runq(pres_loc), back, tmp_ptr) ;
989                   call isrtq(readyq(fut_loc), back, tmp_ptr) ;
990
991                 end ;
992
993                 else do ;
994
995                   /* Middle of queue. */
996
997                   /* Find the pcb in the queue. */
998
999                   tmp_ptr = runq(pres_loc).front_ptr ;
000
001                   do while (tmp_ptr ^= null() & tmp_ptr ^= pcbptr) ;
002                     last_pcbptr = tmp_ptr ;
003                     tmp_ptr = tmp_ptr->pcb.next_pcb ;
004                   end ;
005
006                   /* Adjust the link list */
007
008                   last_pcbptr->pcb.next_pcb = pcbptr->pcb.next_pcb ;
009                   runq(pres_loc).no_of_entries =
010                                    runq(pres_loc).no_of_entries - 1 ;
011                   pcbptr->pcb.next_pcb = null() ;
```

```
012               tmp_ptr = pcbptr ;
013               call isrtq(readyq(fut_loc), back, tmp_ptr) ;
014
015             end ;
016         end ;          /* end runq preemption */
017
018         else do ;      /*      WAITQ ONLY   */
019
020           /* Find the pcb in the waitq and move it to the readyq */
021           /* also update pcb status.                             */
022
023           pcbptr->pcb.ready_tm = 0       ;
024           pcbptr->pcb.state    = 'READY' ;
025           pcbptr->pcb.curr_pri = fut_loc ;
026
027           /* Find pcb in the waitq. */
028
029           if pcbptr = waitq(pres_loc).front_ptr
030             then do ;
031
032               /* Front of queue. */
033
034               call dletq(waitq(pres_loc), front, tmp_ptr) ;
035               call isrtq(readyq(fut_loc), back, tmp_ptr) ;
036
037             end ;
038
039             else if pcbptr = waitq(pres_loc).back_ptr
040               then do ;
041
042                 /* Back of queue. */
043
044                 call dletq(waitq(pres_loc), back, tmp_ptr) ;
045                 call isrtq(readyq(fut_loc), back, tmp_ptr) ;
046
047               end ;
048
049               else do ;
050
051                 /* Middle of queue. */
052
053                 /* Find the pcb in the queue. */
054
055                 tmp_ptr = waitq(pres_loc).front_ptr ;
056
057                 do while (tmp_ptr ^= null() & tmp_ptr ^= pcbptr) ;
058                   last_pcbptr = tmp_ptr ;
059                   tmp_ptr = tmp_ptr->pcb.next_pcb ;
060                 end ;
061
062                 /* Adjust the link list */
063
```

```
064                last_pcbptr->pcb.next_pcb = pcbptr->pcb.next_pcb ;
065                waitq(pres_loc).no_of_entries =
066                                waitq(pres_loc).no_of_entries - 1 ;
067                pcbptr->pcb.next_pcb = null() ;
068                tmp_ptr = pcbptr ;
069                call isrtq(readyq(fut_loc), back, tmp_ptr) ;
070
071             end ;
072       end ;          /* end waitq preemption */
073    /**/ put file (debug) skip edit (errmsg2) (a) ;
074    /**/ do jj=1 to $NPRI ;
075    /**/  call debugger(jj, holdq(jj), 'hold') ;
076    /**/ end ;
077    /**/ do jj=1 to $NPRI ;
078    /**/ call debugger(jj, readyq(jj), 'readyq') ;
079    /**/ end ;
080    /**/ do jj=1 to $NPRI ;
081    /**/ call debugger(jj, runq(jj), 'runq') ;
082    /**/ end ;
083
084     end preempt ;
085
086
087
```

```
088     isrtq:  procedure(queue, pos_flag, pcbptr) ;
089
090        /*********************************************************/
091        /* This procedure will insert a member from an applicable */
092        /* queue.  This procedure will insert in the front of the */
093        /* queue when the pos_flag is set to '1'b (true) and will */
094        /* insert at the back of the queue when the pos_flag is   */
095        /* set to '0'b (false).                                   */
096        /*********************************************************/
097
098        dcl 1 queue ,
099              2 front_ptr        pointer       ,
100              2 back_ptr         pointer       ,
101              2 no_of_entries    fixed bin(15) ;
102
103        dcl   pos_flag           bit(1)   ,
104              pcbptr             pointer  ;
105
106        dcl   null               builtin  ;
107
108        dcl 1 pcb                based(pcbptr) ,
109              2 job_id           char(3)       ,
110              2 state            char(8)       varying ,
111              2 init_pri         fixed bin(7) ,
112              2 curr_pri         fixed bin(7) ,
113              2 init_tm          fixed dec(6) ,
114              2 rem_tm           fixed dec(6) ,
115              2 turn_tm          fixed dec(6) ,
116              2 act_tm           fixed dec(6) ,
117              2 ready_tm         fixed dec(6) ,
118              2 next_pcb         pointer       ;
119
120        /* Check the insertion order. */
121
122        if pos_flag        /* If pos_flag = '1'b */
123          then do ;        /* insert in front    */
124            if queue.no_of_entries = 0
125              then do ;                              /* Empty queue */
126                queue.front_ptr      = pcbptr ;
127                queue.back_ptr       = pcbptr ;
128                queue.no_of_entries  = 1      ;
129                pcbptr->pcb.next_pcb = null() ;
130
131              end ;
132
133              else do ;                              /* Adjust the links */
134                pcbptr->pcb.next_pcb = queue.front_ptr ;
135                queue.front_ptr      = pcbptr ;
136                queue.no_of_entries  = queue.no_of_entries + 1 ;
137              end ;
138
139          end ;       /* End insert front */
```

```
140
141         else do ;          /* pos_flag is set to '0'b */
142                            /* insert at back          */
143
144           if queue.no_of_entries = 0          /* Empty queue */
145             then do ;
146               queue.front_ptr      = pcbptr ;
147               queue.back_ptr       = pcbptr ;
148               queue.no_of_entries  = 1      ;
149               pcbptr->pcb.next_pcb = null() ;
150             end ;
151
152             else do ;    /* Adjust pcb links */
153               queue.back_ptr->pcb.next_pcb = pcbptr ;
154               queue.back_ptr               = pcbptr ;
155               queue.no_of_entries  = queue.no_of_entries + 1 ;
156               pcbptr->pcb.next_pcb = null() ;
157             end ;
158
159         end ;        /* End back insert */
160
161     end isrtq ;
162
163
```

```
dletq:  procedure(queue, pos_flag, pcbptr) ;

   /*************************************************************/
   /* This procedure will delete a member from an applicable    */
   /* queue.  This procedure will delete from the front of the  */
   /* queue when the pos_flag is set to '1'b (true) and will    */
   /* delete from the back of the queue when the pos_flag is    */
   /* set to '0'b (false).                                      */
   /* The pcbptr variable is set to the pcb which is deleted.   */
   /*************************************************************/

   dcl 1 queue ,
         2 front_ptr        pointer       ,
         2 back_ptr         pointer       ,
         2 no_of_entries    fixed bin(15) ;

   dcl   pos_flag           bit(1)   ,
         pcbptr             pointer  ,
         tmp_ptr            pointer  ;

   dcl   null               builtin  ;

   dcl 1 pcb                based(pcbptr) ,
         2 job_id           char(3)       ,
         2 state            char(8)       varying ,
         2 init_pri         fixed bin(7) ,
         2 curr_pri         fixed bin(7) ,
         2 init_tm          fixed dec(6) ,
         2 rem_tm           fixed dec(6) ,
         2 turn_tm          fixed dec(6) ,
         2 act_tm           fixed dec(6) ,
         2 ready_tm         fixed dec(6) ,
         2 next_pcb         pointer       ;

   /* Check the deletion order. */

   if pos_flag        /* If pos_flag = '1'b */
     then do ;        /* delete from front  */

        /* Set the pcbptr to reflect the pointer to the */
        /* pcb you are deleting.                        */

        pcbptr              = queue.front_ptr ;

        /* Check if there is only one member */

        if queue.front_ptr = queue.back_ptr     /* Only member */
          then do ;
            queue.front_ptr      = null() ;
            queue.back_ptr       = null() ;
            queue.no_of_entries  = 0      ;
```

```
216             end ;
217
218             else do ;    /* Adjust pcb links */
219               tmp_ptr = queue.front_ptr->pcb.next_pcb ;  /* New front */
220               queue.front_ptr = tmp_ptr ;
221               queue.no_of_entries = queue.no_of_entries - 1 ;
222             end ;   /* End adjust pcb links */
223
224           pcbptr->pcb.next_pcb = null() ;
225
226         end ;       /* End front delete */
227
228         else do ;        /* Pos_flag set to '0'b */
229                          /* delete from back     */
230
231           pcbptr = queue.back_ptr ;
232           if queue.back_ptr = queue.front_ptr     /* Only member */
233             then do ;
234               queue.front_ptr     = null() ;
235               queue.back_ptr      = null() ;
236               queue.no_of_entries = 0      ;
237             end ;
238
239           else do ;      /* Adjust pcb links */
240
241             /* Traverse the queue from the front until a link  */
242             /* points to the last member (back_ptr address).   */
243
244             tmp_ptr = queue.front_ptr ;
245             do while (tmp_ptr->pcb.next_pcb ^= queue.back_ptr) ;
246               tmp_ptr = tmp_ptr->pcb.next_pcb ;
247             end ;
248
249             /* Now tmp_ptr points to the new back pointer. */
250
251             tmp_ptr->pcb.next_pcb = null()  ;
252             queue.back_ptr        = tmp_ptr ;
253             queue.no_of_entries   = queue.no_of_entries - 1 ;
254
255           end ;   /* End adjust pcb links */
256
257         pcbptr->pcb.next_pcb = null() ;
258
259       end ;       /* End back delete      */
260
261     end dletq ;
262
263
```

```
264     clock:  procedure(input_string, pos_in_string, err_count,
265                       holdq, readyq, runq, waitq, doneq,
266                       tot_time, tot_act_time) ;
267
268        /*************************************************************/
269        /* This procedure updates the clock of the cpu.  This procedure */
270        /* will simulate the passage of time in the operating system.   */
271        /* Several things will happen due to the passage of time.       */
272        /* First all times are updated.  Secondly, the services of the  */
273        /* short term scheduler are invoked.  This causes movement of   */
274        /* jobs from the queues.  Thirdly, if a job exceeds its quantum */
275        /* limit, it is bumped out of the runq.                         */
276        /*************************************************************/
277
278     /**/ dcl jj fixed bin(7) ;
279     /**/ dcl 1 errmsg1 ,
280     /**/     2 msg char (50) init('     * * ENTRY in clock');
281     /**/ put file (debug) skip edit (errmsg1) (a) ;
282     /**/ dcl 1 errmsg2 ,
283     /**/ 2 msg char(50) init ('ready to leave clock') ;
284     /**/ do jj=1 to $NPRI ;
285     /**/  call debugger(jj, holdq(jj), 'hold') ;
286     /**/ end ;
287     /**/ do jj=1 to $NPRI ;
288     /**/ call debugger(jj, readyq(jj), 'readyq') ;
289     /**/ end ;
290     /**/ do jj=1 to $NPRI ;
291     /**/ call debugger(jj, runq(jj), 'runq') ;
292     /**/ end ;
293     /**/ do jj=1 to $NWAIT ;
294     /**/ call debugger(jj, waitq(jj), 'waitq') ;
295     /**/ end ;
296     /**/ call debugger(0, doneq, 'doneq') ;
297        dcl   length              builtin ,
298              verify              builtin ,
299              min                 builtin ,
300              null                builtin ;
301
302        dcl   pos_in_string       fixed bin(15) ,
303              input_string        char(80)      varying ,
304              err_count           fixed bin(15) ;
305
306        dcl   token_ret           char(80)      varying ,
307              token_end_pos       fixed bin(15) ,
308              ticker              fixed dec(6) ,
309              no_of_ticks         fixed dec(6) ,
310              ms                  fixed dec(6) ;
311
312        dcl   digits              char(10)      init('0123456789') ;
313
314        dcl   ret_bin             fixed bin(7) ,
315              use_runq            bit(1)        init ('1'b) ,
```

```
316             want_dec            bit(1)        init ('0'b) ;
317
318       dcl   tmp_ptr             pointer ,
319             next_ptr            pointer ;
320
321       dcl   tot_time            fixed bin(15) ,
322             tot_act_time        fixed bin(15) ;
323
324       dcl   front               bit(1)        init('1'b) ,
325             back                bit(1)        init('0'b) ;
326
327       dcl   quantum($NPRI)      fixed bin(15) init($QUAN1 ,
328                                                    $QUAN2 ,
329                                                    $QUAN3 ,
330                                                    $QUAN4 ) ;
331
332       dcl    new_pri            fixed bin(7)  ,
333              act_jobs_found     fixed bin(15) ,
334              i                  fixed bin(15) ;
335
336       dcl 1 holdq($NPRI) ,
337             2 front_ptr         pointer       ,
338             2 back_ptr          pointer       ,
339             2 no_of_entries     fixed bin(15) ;
340
341       dcl 1 readyq($NPRI) ,
342             2 front_ptr         pointer       ,
343             2 back_ptr          pointer       ,
344             2 no_of_entries     fixed bin(15) ;
345
346       dcl 1 runq($NPRI) ,
347             2 front_ptr         pointer       ,
348             2 back_ptr          pointer       ,
349             2 no_of_entries     fixed bin(15) ;
350
351       dcl 1 waitq($NWAIT) ,
352             2 front_ptr         pointer       ,
353             2 back_ptr          pointer       ,
354             2 no_of_entries     fixed bin(15) ;
355
356       dcl 1 tmpq($NPRI) ,
357             2 front_ptr         pointer       ,
358             2 back_ptr          pointer       ,
359             2 no_of_entries     fixed bin(15) ;
360
361       dcl 1 doneq ,
362             2 front_ptr         pointer       ,
363             2 back_ptr          pointer       ,
364             2 no_of_entries     fixed bin(15) ;
365
366       dcl 1 pcb                based(pcbptr) ,
367             2 job_id            char(3)       ,
```

```
368           2 state            char(8)       varying,
369           2 init_pri         fixed bin(7) ,
370           2 curr_pri         fixed bin(7) ,
371           2 init_tm          fixed dec(6) ,
372           2 rem_tm           fixed dec(6) ,
373           2 turn_tm          fixed dec(6) ,
374           2 act_tm           fixed dec(6) ,
375           2 ready_tm         fixed dec(6) ,
376           2 next_pcb         pointer       ;
377
378     dcl    tick_err1          char(80)      init
379                   ('Error:  No tick number found in input string') ;
380
381     dcl    tick_err2          char(80)      init
382                                   ('Error:  Tick number too long') ;
383
384     dcl    tick_err3          char(80)      init
385                                   ('Error:  Tick number not numeric') ;
386
387     dcl    tick_err4          char(80)      init
388                                   ('Error:  Tick number out or range') ;
389
390     /* Retrieve the time passage integer for which the clock   */
391     /* has just ticked away.  This integer * $TICK time will   */
392     /* represent the actual milliseconds of time which have    */
393     /* passed by .                                             */
394
395     /* Tick number. */
396
397     pos_in_string = pos_in_string + 1 ;   /* First spot after token */
398     call gtoken(pos_in_string, input_string, token_ret,
399                 token_end_pos) ;
400     if length(token_ret) < 1     /* No token found. */
401       then do ;
402         call inerr(tick_err1, input_string, err_count) ;
403         return ;
404       end ;
405
406     if length(token_ret) > $TICKMAX
407       then do ;                  /* Token too long. */
408         call inerr(tick_err2, input_string, err_count) ;
409         return ;
410       end ;
411
412     /* Conert token from character to numeric */
413     /* and check its range.                   */
414
415     if verify(token_ret, digits) > 0
416       then do ;                        /* Token not numeric. */
417         call inerr(tick_err3, input_string, err_count) ;
418         return ;
419       end ;
```

```
420
421         call charnumb(token_ret, want_dec, ret_bin, no_of_ticks) ;
422
423         if no_of_ticks < 1 | no_of_ticks > $MAXTICK
424           then do ;
425             call inerr(tick_err4, input_string, err_count) ;
426             return ;
427           end ;
428
429         /* Loop from 1 to the tick number updating pcb times,  */
430         /* bumping jobs, etc. as necessary.  In short, perform */
431         /* the functions of the scheduler.                     */
432
433         ms = $TICKMS ;
434
435         do ticker=1 to no_of_ticks ;
436       /**/ dcl 1  mymsg ,
437       /**/ 2 msg char(20) init ('* * TIC - TOCK * *') ;
438       /**/ put file (debug) skip edit (mymsg) (a) ;
439
440           /* Update the CPU active time if a job is now active */
441           /* on the CPU.                                       */
442
443           act_jobs_found = 0 ;
444           do i=1 to $NPRI ;
445             if runq(i).no_of_entries > 0
446               then act_jobs_found = act_jobs_found + 1 ;
447           end ;
448
449           if act_jobs_found > 0
450             then
451               tot_act_time = tot_act_time + ms ;
452
453           /* Update the total elapsed time. */
454
455           tot_time = tot_time + ms ;
456
457           /* Update holdq, readyq, runq pcb's. */
458
459           do i=$NPRI to 1 by -1 ;
460
461             /* Update turn around time for the holdq */
462
463             if holdq(i).no_of_entries > 0
464               then do ;                          /* Entries found. */
465                 tmp_ptr = holdq(i).front_ptr ;
466                 do while (tmp_ptr ^= null());
467                   tmp_ptr->pcb.turn_tm = tmp_ptr->pcb.turn_tm + ms ;
468                   tmp_ptr = tmp_ptr->pcb.next_pcb ;
469                 end ;
470               end ;
471
```

```
472           /* Update turn-around time and ready time for */
473           /* the readyq.                                */
474           if readyq(i).no_of_entries > 0
475             then do ;                          /* Entries found */
476               tmp_ptr = readyq(i).front_ptr ;
477               do while (tmp_ptr ^= null()) ;
478                 tmp_ptr->pcb.turn_tm = tmp_ptr->pcb.turn_tm + ms ;
479                 tmp_ptr->pcb.ready_tm = tmp_ptr->pcb.ready_tm + ms ;
480                 tmp_ptr = tmp_ptr->pcb.next_pcb ;
481               end ;
482             end ;
483
484           /* Update the turn around time, the remaining time and */
485           /* the active time for the runq.                       */
486
487           if runq(i).no_of_entries > 0
488             then do ;                          /* Entries found */
489               tmp_ptr = runq(i).front_ptr ;
490               do while (tmp_ptr ^= null()) ;
491                 tmp_ptr->pcb.rem_tm    = tmp_ptr->pcb.rem_tm  - ms ;
492                 tmp_ptr->pcb.act_tm    = tmp_ptr->pcb.act_tm  + ms ;
493                 tmp_ptr->pcb.turn_tm   = tmp_ptr->pcb.turn_tm + ms ;
494                 if tmp_ptr->pcb.rem_tm < 0
495                   then do
496
497                     /* The active job completed before the full */
498                     /* ms increment, so adjust the turn-around  */
499                     /* time to truly reflect the completion     */
500                     /* time.                                    */
501
502                     tmp_ptr->pcb.turn_tm = tmp_ptr->pcb.turn_tm -
503                                            tmp_ptr->pcb.rem_tm  ;
504                     tmp_ptr->pcb.rem_tm = 0 ;
505                   end ;       /* End rem_tm < 0 */
506
507                 tmp_ptr = tmp_ptr->pcb.next_pcb ;
508               end ;           /* End do while */
509             end ;             /* End then do  */
510
511         end ;  /* End holdq, readyq, runq pcb update */
512
513         /* Update the waitq pcb's turn-around time. */
514
515         do i=1 to $NWAIT ;
516           if waitq(i).no_of_entries > 0
517             then do ;                    /* Queue has jobs */
518               tmp_ptr = waitq(i).front_ptr ;
519               do while (tmp_ptr ^= null()) ;
520                 tmp_ptr->pcb.turn_tm = tmp_ptr->pcb.turn_tm + ms ;
521                 tmp_ptr = tmp_ptr->pcb.next_pcb ;
522               end ;      /* End do while */
523             end ;        /* End then do  */
```

```
524         end ;            /* End waitq do */
525
526         /* Start checking the runq if any jobs have completed. */
527         /* Copy each completed job to the doneq and the other  */
528         /* jobs to the tmpq.                                   */
529
530         /* Initialize the structure */
531
532         do i=1 to $NPRI ;
533           tmpq(i).front_ptr     = null() ;
534           tmpq(i).back_ptr      = null() ;
535           tmpq(i).no_of_entries = 0      ;
536         end ;
537
538         do i=1 to $NPRI ;
539           if runq(i).no_of_entries > 0   /* runq entries */
540             then do ;
541               pcbptr = runq(i).front_ptr ;
542               do while (pcbptr ^= null()) ;   /* loop thru links */
543
544                 tmp_ptr = pcbptr->pcb.next_pcb ;
545                 if pcbptr->pcb.rem_tm = 0
546                   then do ;
547
548                     /* update the pcb status to complete */
549
550                     pcbptr->pcb.state    = 'COMPLETE' ;
551                     pcbptr->pcb.ready_tm = tot_time   ;
552                     call isrtq(doneq, back, pcbptr) ;
553                   end ;
554
555                 else call isrtq(tmpq(i), back, pcbptr) ;
556                pcbptr = tmp_ptr ;
557               end ;     /* end do while */
558             end ;       /* end then do  */
559         end ;   /* end i loop */
560
561         /* Copy the tmpq to the runq structure */
562
563         do i=1 to $NPRI ;
564           runq(i).front_ptr     = tmpq(i).front_ptr     ;
565           runq(i).back_ptr      = tmpq(i).back_ptr      ;
566           runq(i).no_of_entries = tmpq(i).no_of_entries ;
567         end ;
568
569         /* Now check if the quantum time has expired for  */
570         /* the jobs running on the cpu.  If it has, bump  */
571         /* the active jobs out of the runq.                */
572
573         do i=1 to $NPRI ;
574           if runq(i).no_of_entries > 0
575             then do ;
```

```
576             pcbptr   = runq(i).front_ptr ;
577             do while (pcbptr ^= null()) ;
578               next_ptr = pcbptr->pcb.next_pcb ;
579
580               /* Check if quantum limit has been hit or exceeded. */
581               /* if so, bump the runq job.                        */
582
583               if pcbptr->pcb.act_tm >= quantum(i)
584                 then call preempt(pcbptr, pcbptr->pcb.curr_pri ,
585                                   pcbptr->pcb.curr_pri - 1 ,
586                                   use_runq ,
587                                   holdq, readyq, runq, waitq) ;
588
589               pcbptr = next_ptr ;
590             end ;            /* End do while */
591           end ;              /* End then do  */
592
593         end ;                /* End i loop   */
594
595         /* Check the readyq for jobs not receiving service */
596         /* for $RDYSERV time.  If this is the case, bump   */
597         /* their priority.                                 */
598
599         /* Initialize the structure */
600
601         do i=1 to $NPRI ;
602           tmpq(i).front_ptr     = null() ;
603           tmpq(i).back_ptr      = null() ;
604           tmpq(i).no_of_entries = 0      ;
605         end ;
606
607         do i=1 to $NPRI ;
608           if readyq(i).no_of_entries > 0
609             then do ;
610               pcbptr = readyq(i).front_ptr ;
611               do while (pcbptr ^= null()) ;
612                 tmp_ptr = pcbptr->pcb.next_pcb ;
613                 if pcbptr->pcb.ready_tm >= $RDYSERV
614                   then do ;              /* move to higher q */
615                     new_pri = i + min($MAXUPGE, ($NPRI -
616                                       pcbptr->pcb.curr_pri)) ;
617
618                     /* Grant the higher q if the new priority */
619                     /* is less then $NPRI and the q limit     */
620                     /* has not been exceeded.                 */
621
622                     if new_pri > $NPRI
623                       then new_pri = $NPRI ;
624
625                     if tmpq(new_pri).no_of_entries < $QLIMIT
626                       then do ;
627                         if new_pri > pcbptr->pcb.curr_pri
```

```
628                       then do ;
629                         pcbptr->pcb.curr_pri = new_pri ;
630                         pcbptr->pcb.ready_tm = 0       ;
631                       end ;
632                     call isrtq(tmpq(new_pri), back, pcbptr) ;
633                   end ;
634
635                   /* The queue is full at the +2 priority */
636                   /* increase.  Try the +1 increase if    */
637                   /* increase is greater then the current */
638                   /* priority.                            */
639
640                   else if tmpq(new_pri - 1).no_of_entries
641                           < $QLIMIT
642                           & new_pri - 1 > pcbptr->pcb.curr_pri
643
644                     then do ;
645                       pcbptr->pcb.curr_pri = new_pri - 1 ;
646                       pcbptr->pcb.ready_tm = 0 ;
647                       call isrtq(tmpq(new_pri - 1), back, pcbptr) ;
648                     end ;
649
650                     else call isrtq(tmpq(new_pri), back, pcbptr) ;
651
652               end ;   /* end then do ready_tm > $RDYSERV */
653
654               else call isrtq(tmpq(i), back, pcbptr) ;
655
656             pcbptr = tmp_ptr ;   /* next pcb in link list */
657
658           end ;   /* end do while */
659
660         end ;   /* end entries > 0 */
661      end ;   /* end i loop */
662      /* Copy the tmpq to the readyq structure */
663
664      do i=1 to $NPRI ;
665        readyq(i).front_ptr     = tmpq(i).front_ptr     ;
666        readyq(i).back_ptr      = tmpq(i).back_ptr      ;
667        readyq(i).no_of_entries = tmpq(i).no_of_entries ;
668      end ;
669
670
671    /* Perform the functions of the short term scheduler.    */
672    /* Invoke the priorck procedure which will schedule and/ */
673    /* or preempt jobs from the runq.                        */
674
675    call priorck(holdq, readyq, runq) ;
676
677      /* Now invoke the long term scheduler. */
678
679       call lgsched(holdq, readyq, runq ) ;
```

```
680
681         end ;    /* End ticker loop */
682
683     /**/ put file (debug) skip edit (errmsg2) (a) ;
684     /**/ do jj=1 to $NPRI ;
685     /**/  call debugger(jj, holdq(jj), 'hold') ;
686     /**/ end ;
687     /**/ do jj=1 to $NPRI ;
688     /**/ call debugger(jj, readyq(jj), 'readyq') ;
689     /**/ end ;
690     /**/ do jj=1 to $NPRI ;
691     /**/ call debugger(jj, runq(jj), 'runq') ;
692     /**/ end ;
693     /**/ do jj=1 to $NWAIT ;
694     /**/ call debugger(jj, waitq(jj), 'waitq') ;
695     /**/ end ;
696     /**/ call debugger(0, doneq, 'doneq') ;
697      end clock ;
698
699
```

```
resrc:  procedure(transaction, input_string, pos_in_string,
                  err_count, readyq, holdq, waitq, runq) ;

   /*************************************************************/
   /* This procedure will request or release resources for given */
   /* processes.  Currently we support only 1 active job and no  */
   /* resource sharing.   This procedure can be enhanced to      */
   /* support multiple resource handling.                        */
   /*************************************************************/

   dcl   null              builtin ,
         length            builtin ,
         verify            builtin ;

   dcl   transaction       char(7) ,
         input_string      char(80)      varying ,
         pos_in_string     fixed bin(15) ,
         err_count         fixed bin(15) ;

   dcl   i                 fixed bin(15) ,
         dev_no            fixed bin(7) ;

   dcl   want_bin          bit(1)        init ('1'b) ,
         use_waitq         bit(1)        init ('0'b) ,
         ret_dec           fixed dec(6)  ;

   dcl   token_ret         char(80)      varying ,
         token_end_pos     fixed bin(15) ;

   dcl   front             bit(1)        init('1'b) ,
         back              bit(1)        init('0'b) ;

   dcl   run_pcbptr        pointer ,
         wait_pcbptr       pointer ,
         found_pcbptr      pointer ,
         curr_ptr          pointer ,
         last_ptr          pointer ,
         pcbptr            pointer ;

   dcl   request           char(7)       init ('request') ,
         release           char(7)       init ('release') ;

   dcl   digits            char(10)      init ('0123456789') ;

   dcl 1 waitq($NWAIT) ,
         2 front_ptr       pointer ,
         2 back_ptr        pointer ,
         2 no_of_entries   fixed bin(15) ;

   dcl 1 runq($NPRI) ,
         2 front_ptr       pointer ,
         2 back_ptr        pointer ,
```

```
752           2 no_of_entries    fixed bin(15) ;
753
754       dcl 1 readyq($NPRI) ,
755           2 front_ptr        pointer ,
756           2 back_ptr         pointer ,
757           2 no_of_entries    fixed bin(15) ;
758
759       dcl 1 holdq($NPRI) ,
760           2 front_ptr        pointer ,
761           2 back_ptr         pointer ,
762           2 no_of_entries    fixed bin(15) ;
763
764       dcl   dev_err1           char(80)      init
765                ('Error:  No device number found in input string') ;
766
767       dcl   dev_err2           char(80)      init
768                                 ('Error:  Device number too long') ;
769
770       dcl   dev_err3           char(80)      init
771                               ('Error:  Device number not numeric') ;
772
773       dcl   dev_err4           char(80)      init
774                             ('Error:  Device number out of range') ;
775
776       dcl   dev_err5           char(80)      init
777                             ('Error:  Requested device is in use') ;
778
779       dcl   dev_err6           char(80)      init
780                         ('Error:  Releasing device is not in use') ;
781
782       dcl   run_err1           char(80)      init
783                        ('Error:  Requesting process is not active') ;
784
785       dcl   run_err2           char(80)      init
786                     ('Error:  Releasing process has no I/O device') ;
787
788       dcl 1 pcb                based(pcbptr) ,
789           2 job_id             char(3) ,
790           2 state              char(8)       varying ,
791           2 init_pri           fixed bin(7)  ,
792           2 curr_pri           fixed bin(7)  ,
793           2 init_tm            fixed dec(6)  ,
794           2 rem_tm             fixed dec(6)  ,
795           2 turn_tm            fixed dec(6)  ,
796           2 act_tm             fixed dec(6)  ,
797           2 ready_tm           fixed dec(6)  ,
798           2 next_pcb           pointer ;
799
800       /* retrieve the device number */
801
802       pos_in_string = pos_in_string + 1 ;
803       call gtoken(pos_in_string, input_string, token_ret,
```

```
804                token_end_pos) ;
805         if length(token_ret) < 1
806           then do ;     /* no token found */
807             call inerr(dev_err1, input_string, err_count) ;
808             return ;
809           end ;
810
811         if length(token_ret) > $WAITMAX   /* Token too long */
812           then do ;
813             call inerr(dev_err2, input_string, err_count) ;
814             return ;
815           end ;
816
817         /* Convert token from character to numeric and */
818         /* check its range.                            */
819
820         if verify(token_ret, digits) > 0
821           then do ;                      /* Not numeric */
822             call inerr(dev_err3, input_string, err_count) ;
823             return ;
824           end ;
825
826         call charnumb(token_ret, want_bin, dev_no, ret_dec) ;
827
828         if dev_no > $NWAIT | dev_no < 1
829           then do ;
830             call inerr(dev_err4, input_string, err_count) ;
831             return ;
832           end ;
833
834         select ;
835
836           when (transaction = request)
837             do ;
838
839               /* Check if requested device is available */
840
841               if waitq(dev_no).no_of_entries = $WLIMIT
842                 then do ;       /* Full queue */
843                   call inerr(dev_err5, input_string, err_count) ;
844                   return ;
845                 end ;
846
847               /* Find the requesting pcbptr from runq.  Normally  */
848               /* in a multi task os, this information is attached */
849               /* to the request.                                  */
850
851               run_pcbptr = null() ;
852               do i=1 to $NPRI while (run_pcbptr = null() ) ;
853                 if runq(i).no_of_entries > 0
854                   then run_pcbptr = runq(i).front_ptr ;
855               end ;
```

```
856
857             /* Check to determine if requesting run_pcbptr */
858             /* is active on the cpu.                       */
859
860             found_pcbptr = null() ;
861             do i=1 to $NPRI while (found_pcbptr = null()) ;
862               if runq(i).no_of_entries > 0
863                 then do ;
864                   if runq(i).front_ptr = run_pcbptr
865                     then found_pcbptr = run_pcbptr ; /* Front */
866
867                     else if runq(i).back_ptr = run_pcbptr
868                       then found_pcbptr      = run_pcbptr  ; /* Back */
869
870                       else do ;    /* In middle */
871
872                         /* Find the pcb within the link list. */
873
874                         curr_ptr = runq(i).front_ptr ;
875                         last_ptr = curr_ptr ;
876                         do while (curr_ptr ^= null()) ;
877                           if curr_ptr = run_pcbptr
878                             then do ;    /* Found in link list */
879                               curr_ptr     = null()    ;
880                               found_pcbptr = run_pcbptr ;
881                             end ;
882                             else do ;    /* No match */
883                               last_ptr = curr_ptr ;
884                               curr_ptr = curr_ptr->pcb.next_pcb ;
885                             end ;
886                         end ;     /* End do while */
887
888                       end ;       /* End else do  */
889
890                   end ;    /* End entries > 0 */
891
892               end ;        /* End i loop      */
893
894               /* Check if the requesting pcbptr was found */
895
896               if found_pcbptr ^= run_pcbptr
897                 then do ;
898                   call inerr(run_err1, input_string, err_count) ;
899                   return ;
900                 end ;
901
902               /* Move the running pcbptr from the runq to the */
903               /* waitq.                                       */
904
905               run_pcbptr->pcb.state = 'WAITING' ;
906               if run_pcbptr = runq(i-1).front_ptr
907                 then do ;       /* Front */
```

```
908               call dletq(runq(i-1), front, curr_ptr) ;
909               call isrtq(waitq(dev_no), back, curr_ptr) ;
910             end ;
911
912             else if run_pcbptr = runq(i-1).back_ptr
913               then do ;     /* Back */
914                 call dletq(runq(i-1), back, curr_ptr) ;
915                 call isrtq(waitq(dev_no), back, curr_ptr) ;
916               end ;
917
918               else do ;     /* Middle */
919                 last_ptr->pcb.next_pcb  = run_pcbptr->pcb.next_pcb ;
920                 runq(i-1).no_of_entries = runq(i-1).no_of_entries -1 ;
921                 curr_ptr = run_pcbptr ;
922                 call isrtq(waitq(dev_no), back, curr_ptr) ;
923               end ;
924
925         /* Schedule the cpu for the next job from the readyq */
926
927         call priorck(holdq, readyq, runq) ;
928
929     end ;     /* End select request command. */
930
931   when (transaction = release)
932     do ;
933
934       /* Check to be certain device is in use. */
935
936       if waitq(dev_no).no_of_entries < 1
937         then do ;
938           call inerr(dev_err6, input_string, err_count) ;
939           return ;
940         end ;
941
942       /* Read the released pcbptr.  Normally this comes  */
943       /* attached to the release request.  Since we have */
944       /* only 1 per device, we will pick it up from the  */
945       /* waitq.                                          */
946
947       wait_pcbptr = waitq(dev_no).front_ptr ;
948
949       /* Now check the waitq for this pcbptr.  This do */
950       /* loop is really unecessary now, but will work  */
951       /* when we have multiple device sharing          */
952
953       found_pcbptr = null() ;
954       do i=1 to $NWAIT while (found_pcbptr = null()) ;
955         if waitq(i).no_of_entries > 0
956           then do ;
957             if waitq(i).front_ptr = wait_pcbptr
958               then found_pcbptr   = wait_pcbptr ;  /* Front */
959
```

```
960                     else if waitq(i).back_ptr = wait_pcbptr
961                       then found_pcbptr       = wait_pcbptr ;
962
963                       else do ;
964                         curr_ptr = waitq(i).front_ptr ;
965                         do while (curr_ptr ^= null()) ;
966                           if curr_ptr = wait_pcbptr
967                             then do ;
968                               curr_ptr     = null() ;
969                               found_pcbptr = wait_pcbptr ;
970                             end ;
971
972                             else do ;
973                               last_ptr = curr_ptr ;
974                               curr_ptr = curr_ptr->pcb.next_pcb ;
975                             end ;
976                         end ;   /* end do while */
977                       end ;     /* end else do  */
978                 end ;     /* end entries > 0 */
979             end ;     /* end i loop */
980
981             /* Now check that you found the pcb */
982
983             if found_pcbptr ^= wait_pcbptr
984               then do ;
985                 call inerr(run_err2, input_string, err_count) ;
986                 return ;
987               end ;
988
989             /* Now remove the pcb from the waitq to the readyq. */
990             /* Call preempt to do this.  Pass the waitq as the  */
991             /* runq, preempt wont know the difference.          */
992
993             call preempt(wait_pcbptr, wait_pcbptr->pcb.curr_pri,
994                          wait_pcbptr->pcb.curr_pri + 1,
995                          use_waitq, holdq, readyq, runq, waitq) ;
996
997           end ;   /* end do - release */
998       end ;   /* end of the select */
999
1000    end resrc ;
1001
1002
```

```
003     dumper:  procedure (tot_time, runq, readyq, holdq, waitq, doneq) ;
004
005        /*********************************************************/
006        /* This procedure takes a snapshot of the operating system */
007        /* by dumping the pcb's in all of the queues.              */
008        /*********************************************************/
009
010        dcl   tot_time           fixed bin(15) ,
011              dump_numb          fixed bin(15) static init (0) ,
012              i                  fixed bin(7)  ;
013
014        dcl   prt_hdr            char(14) ,
015              curr_ptr           pointer  ,
016              pcbptr             pointer  ,
017              header_flag        bit(1)   ,
018              prt_cnt            fixed bin(15) ;
019
020        dcl   null               builtin ;
021
022        dcl 1 prt_ln ,
023              2 header           char(14)     ,
024              2 job_id           char(3)      ,
025              2 blanks1          char(4)      ,
026              2 state            char(8)      ,
027              2 blanks2          char(1)      ,
028              2 init_pri         pic 'zz9'    ,
029              2 blanks3          char(8)      ,
030              2 curr_pri         pic 'zz9'    ,
031              2 blanks4          char(4)      ,
032              2 rem_tm           pic 'zzzzz9' ,
033              2 blanks5          char(3)      ,
034              2 turn_tm          pic 'zzzzz9' ,
035              2 blanks6          char(3)      ,
036              2 ready_tm         pic 'zzzzz9' ;
037
038        dcl 1 pcb                based(pcbptr) ,
039              2 job_id           char(3) ,
040              2 state            char(8)       varying ,
041              2 init_pri         fixed bin(7) ,
042              2 curr_pri         fixed bin(7) ,
043              2 init_tm          fixed dec(6) ,
044              2 rem_tm           fixed dec(6) ,
045              2 turn_tm          fixed dec(6) ,
046              2 act_tm           fixed dec(6) ,
047              2 ready_tm         fixed dec(6) ,
048              2 next_pcb         pointer ;
049
050        dcl 1 no_complete ,
051              2 blanks           char(8)       init (' ')    ,
052              2 text             char(4)       init ('none') ;
053
054        dcl 1 no_process ,
```

```
055           2 header           char(14) ;
056
057       dcl 1 stars ,
058           2 filler           char(74)      init('***************'
059                                              || '***************'
060                                              || '***************'
061                                              || '***************'
062                                              || '***************') ;
063
064       dcl 1 dump_ln ,
065           2 filler           char(8)       init('*** DUMP') ,
066           2 dump_no          pic 'zzzzz9'  ,
067           2 filler2          char(3)       init(' at') ,
068           2 time             pic 'zzzzz9'  ,
069           2 filler3          char(7)       init(' ms ***') ;
070
071       dcl 1 hdr_ln ,
072           2 filler1          char(13)      init(' ') ,
073           2 filler2          char(8)       init('jobid   ')   ,
074           2 filler3          char(8)       init('state   ')   ,
075           2 filler4          char(10)      init('initpri   ') ,
076           2 filler5          char(10)      init('currpri   ') ,
077           2 filler6          char(8)       init('remtm   ')   ,
078           2 filler7          char(9)       init('turntm   ')  ,
079           2 filler8          char(7)       init('readytm')    ;
080
081       dcl 1 end_hdr ,
082           2 text1              char(36)      init
083                          ('Jobs complete as of, elapsed time = ') ,
084           2 clock_tm           pic 'zzzzz9' ,
085           2 text2              char(4)       init (' ms:') ;
086
087       dcl 1 end_ln ,
088           2 filler           char(8)       init(' ') ,
089           2 fields(11)       char(6) ;
090
091       dcl 1 banner ,
092            2 filler1         char(23)      init(' ') ,
093            2 text            char(28)      init
094                                  ('MVMS OPERATING SYSTEM STATUS') ;
095
096       dcl 1 runq($NPRI) ,
097           2 front_ptr        pointer ,
098           2 back_ptr         pointer ,
099           2 no_of_entries    fixed bin(15) ;
100
101       dcl 1 readyq($NPRI) ,
102           2 front_ptr        pointer ,
103           2 back_ptr         pointer ,
104           2 no_of_entries    fixed bin(15) ;
105
106       dcl 1 holdq($NPRI) ,
```

```
107             2 front_ptr        pointer ,
108             2 back_ptr         pointer ,
109             2 no_of_entries    fixed bin(15) ;
110
111       dcl 1 waitq($NWAIT) ,
112             2 front_ptr        pointer ,
113             2 back_ptr         pointer ,
114             2 no_of_entries    fixed bin(15) ;
115
116       dcl 1 doneq ,
117             2 front_ptr        pointer ,
118             2 back_ptr         pointer ,
119             2 no_of_entries    fixed bin(15) ;
120
121       /* If this is the first dump, print the banner */
122
123       if dump_numb = 0
124         then do ;
125           put file (sysout) page ;
126           put file (sysout) skip(2) edit (banner) (a) ;
127           put file (sysout) skip(1) ;
128         end ;
129
130       /* Print the headings */
131
132       dump_numb = dump_numb + 1 ;
133       put file (sysout) skip(2) edit (stars) (a) ;
134       dump_ln.dump_no = dump_numb ;
135       dump_ln.time    = tot_time  ;
136       put file (sysout) skip(1) edit (dump_ln) (a) ;
137       put file (sysout) skip(2) edit (hdr_ln ) (a) ;
138
139       /* Print the active running jobs */
140
141       header_flag = '1'b ;
142       do i=1 to $NPRI ;
143         if runq(i).no_of_entries > 0
144           then do ;
145             curr_ptr = runq(i).front_ptr ;
146             do while (curr_ptr ^= null()) ;
147               if header_flag
148                 then do ;   /* first time */
149                   prt_hdr = 'CPU process' ;
150                   call initprt(prt_hdr, curr_ptr, prt_ln) ;
151                   prt_hdr = ' ' ;
152                   header_flag = '0'b ;
153                 end ;
154
155               else call initprt(prt_hdr, curr_ptr, prt_ln) ;
156               curr_ptr = curr_ptr->pcb.next_pcb ;
157
158             end ;
```

```
159             end ;    /* end entries > 0 */
160           end ;   /* end i loop */
161
162           /* now check if anything printed */
163
164           if header_flag      /* nothing printed */
165             then do ;
166               no_process.header = 'CPU process' ;
167               put file (sysout) skip(1) edit (no_process) (a) ;
168             end ;
169
170           /* print the ready queue */
171
172           header_flag = '1'b ;
173           do i=1 to $NPRI ;
174             if readyq(i).no_of_entries > 0
175               then do ;
176                 curr_ptr = readyq(i).front_ptr ;
177                 do while (curr_ptr ^= null()) ;
178                   if header_flag
179                     then do ;   /* first time */
180                       prt_hdr = 'ready queue' ;
181                       call initprt(prt_hdr, curr_ptr, prt_ln) ;
182                       prt_hdr = ' ' ;
183                       header_flag = '0'b ;
184                     end ;
185
186                   else call initprt(prt_hdr, curr_ptr, prt_ln) ;
187                 curr_ptr = curr_ptr->pcb.next_pcb ;
188               end ;
189             end ;     /* end entries > 0 */
190           end ;   /* end i loop */
191
192           /* check if anything printed */
193
194           if header_flag    /* nothing printed */
195             then do ;
196               no_process.header = 'ready queue' ;
197               put file (sysout) skip(1) edit (no_process) (a) ;
198             end ;
199
200           /* Print the holdq */
201
202           header_flag = '1'b ;
203           do i=1 to $NPRI ;
204             if holdq(i).no_of_entries > 0
205               then do ;
206                 curr_ptr = holdq(i).front_ptr ;
207                 do while (curr_ptr ^= null()) ;
208                   if header_flag        /* first time */
209                     then do ;
210                       prt_hdr = 'hold queue' ;
```

```
211                call initprt(prt_hdr, curr_ptr, prt_ln) ;
212                prt_hdr = ' ' ;
213                header_flag = '0'b ;
214              end ;
215
216              else call initprt(prt_hdr, curr_ptr, prt_ln) ;
217            curr_ptr = curr_ptr->pcb.next_pcb ;
218          end ;
219        end ;   /* end entries > 0 */
220      end ;   /* end i loop */
221
222      /* Check if anything printed */
223
224      if header_flag   /* nothing printed */
225        then do ;
226          no_process.header = 'hold queue' ;
227          put file (sysout) skip(1) edit (no_process) (a) ;
228        end ;
229
230      /* Print the I/O queue */
231
232      header_flag = '1'b ;
233      do i=1 to $NWAIT  ;
234        if waitq(i).no_of_entries > 0
235          then do ;
236            curr_ptr = waitq(i).front_ptr ;
237            do while (curr_ptr ^= null()) ;
238              if header_flag   /* first time */
239                then do ;
240                  prt_hdr = 'i/o queue' ;
241                  call initprt(prt_hdr, curr_ptr, prt_ln) ;
242                  prt_hdr = ' ' ;
243                  header_flag = '0'b ;
244                end ;
245
246              else call initprt(prt_hdr, curr_ptr, prt_ln) ;
247            curr_ptr = curr_ptr->pcb.next_pcb ;
248          end ;
249        end ;     /* end entries > 0 */
250      end ;   /* end i loop */
251
252      /* check if anything printed */
253
254      if header_flag   /* nothing printed */
255        then do ;
256          no_process.header = 'i/o queue' ;
257          put file (sysout) skip(1) edit (no_process) (a) ;
258        end ;
259
260      /* print the completed jobs */
261
262      header_flag = '1'b ;
```

```
263         prt_cnt = 0 ;
264         end_hdr.clock_tm = tot_time ;
265         put file (sysout) skip (1) edit (end_hdr) (a) ;
266         if doneq.no_of_entries > 0
267           then do ;
268             curr_ptr = doneq.front_ptr ;
269             do while (curr_ptr ^= null()) ;
270               prt_cnt = prt_cnt + 1 ;
271               end_ln.fields(prt_cnt) = curr_ptr->pcb.job_id ;
272               header_flag = '0'b ;
273               if prt_cnt > 10
274                 then do ;
275                   prt_cnt = 0 ;
276                   put file (sysout) skip(1) edit (end_ln) (a) ;
277                 end ;
278               curr_ptr = curr_ptr->pcb.next_pcb ;
279             end ;     /* end do while */
280           end ;
281
282         /* Check if anything printed */
283
284         if header_flag      /* nothing printed */
285           then do ;
286             put file (sysout) skip(1) edit (no_complete) (a) ;
287           end ;
288
289           else if prt_cnt > 0  &  prt_cnt < 11
290             then do ;
291               do i=prt_cnt + 1 to 11 ;
292                 end_ln.fields(i) = ' ' ;
293               end ;
294
295               put file (sysout) skip(1) edit (end_ln) (a) ;
296             end ;
297
298       end dumper ;
299
300
```

```
301     initprt:  procedure(prt_hdr, pcbptr, prt_ln) ;
302
303        /*****************************************/
304        /* This procedure will initialize the prt_ln */
305        /* structure and print the structure.        */
306        /*****************************************/
307
308        dcl   prt_hdr            char(14) ,
309              pcbptr             pointer  ;
310
311        dcl 1 pcb                based(pcbptr) ,
312              2 job_id           char(3) ,
313              2 state            char(8)       varying ,
314              2 init_pri         fixed bin(7) ,
315              2 curr_pri         fixed bin(7) ,
316              2 init_tm          fixed dec(6) ,
317              2 rem_tm           fixed dec(6) ,
318              2 turn_tm          fixed dec(6) ,
319              2 act_tm           fixed dec(6) ,
320              2 ready_tm         fixed dec(6) ,
321              2 next_pcb         pointer ;
322
323        dcl 1 prt_ln ,
324              2 header           char(14) ,
325              2 job_id           char(3) ,
326              2 blanks1          char(4) ,
327              2 state            char(8) ,
328              2 blanks2          char(1) ,
329              2 init_pri         pic 'zz9' ,
330              2 blanks3          char(8) ,
331              2 curr_pri         pic 'zz9' ,
332              2 blanks4          char(4) ,
333              2 rem_tm           pic 'zzzzz9' ,
334              2 blanks5          char(3) ,
335              2 turn_tm          pic 'zzzzz9' ,
336              2 blanks6          char(3) ,
337              2 ready_tm         pic 'zzzzz9' ;
338
339        /* fill the structure */
340
341        prt_ln.header   = prt_hdr ;
342        prt_ln.job_id   = pcbptr->pcb.job_id ;
343        prt_ln.blanks1  = ' ' ;
344        prt_ln.state    = pcbptr->pcb.state ;
345        prt_ln.blanks2  = ' ' ;
346        prt_ln.init_pri = pcbptr->pcb.init_pri ;
347        prt_ln.blanks3  = ' ' ;
348        prt_ln.curr_pri = pcbptr->pcb.curr_pri ;
349        prt_ln.blanks4  = ' ' ;
350        prt_ln.rem_tm   = pcbptr->pcb.rem_tm ;
351        prt_ln.blanks5  = ' ' ;
352        prt_ln.turn_tm  = pcbptr->pcb.turn_tm ;
```

```
353         prt_ln.blanks6  = ' ' ;
354         prt_ln.ready_tm = pcbptr->pcb.ready_tm ;
355
356         put file (sysout) skip(1) edit (prt_ln) (a) ;
357
358      end initprt ;
359
360
```

```
361     sumprt:  procedure(doneq, tot_jobs, tot_time,
362                        tot_act_time, err_count) ;
363
364        /*************************************************/
365        /* This procedure will print the final statistics */
366        /* of the MVMS operating system                   */
367        /*************************************************/
368
369        dcl   round              builtin ,
370              decimal            builtin ,
371              null               builtin ;
372
373        dcl 1 doneq ,
374              2 front_ptr        pointer ,
375              2 back_ptr         pointer ,
376              2 no_of_entries    fixed bin(15) ;
377
378        dcl 1 pcb                based(pcbptr) ,
379              2 job_id           char(3) ,
380              2 state            char(8)       varying ,
381              2 init_pri         fixed bin(7) ,
382              2 curr_pri         fixed bin(7) ,
383              2 init_tm          fixed dec(6) ,
384              2 rem_tm           fixed dec(6) ,
385              2 turn_tm          fixed dec(6) ,
386              2 act_tm           fixed dec(6) ,
387              2 ready_tm         fixed dec(6) ,
388              2 next_pcb         pointer ;
389
390        dcl   tot_jobs           fixed bin(15) ,
391              tot_time           fixed bin(15) ,
392              tot_act_time       fixed bin(15) ,
393              err_count          fixed bin(15) ,
394              total_turn_tm      fixed dec(9) ;
395
396        dcl   pcbptr             pointer ;
397
398        dcl 1 sum_hdr ,
399              2 blanks           char(24)      init (' ') ,
400              2 text             char(50) ;
401
402        dcl 1 sum_ent ,
403              2 blanks           char(8)       init (' ') ,
404              2 text             char(24)      init
405                                 ('Number of jobs entered: ') ,
406              2 jobs_in          pic 'zz9' ;
407
408        dcl 1 sum_com ,
409              2 blanks           char(8)       init (' ') ,
410              2 text             char(26)      init
411                                 ('Number of jobs completed: ') ,
412              2 jobs_com         pic 'zz9' ;
```

```
413
414         dcl 1 sum_elapse ,
415               2 blanks           char(8)       init (' ') ,
416               2 text1            char(20)      init
417                                  ('Total elapsed time: ') ,
418               2 tot_elapse       pic 'zzzzz9' ,
419               2 text2            char(3)       init (' ms') ;
420
421         dcl 1 sum_cpu ,
422               2 blanks           char(8)      init ( ' ') ,
423               2 text1            char(16)     init
424                                               ('Total CPU time: ') ,
425               2 tot_cpu          pic 'zzzz9' ,
426               2 text2            char(3)      init (' ms') ;
427
428         dcl 1 sum_tot_turn ,
429               2 blanks           char(8)       init (' ') ,
430               2 text1            char(44)      init
431                 ('Total turnaround time (for completed jobs): ') ,
432               2 tot_turn         pic 'zzzzzzzz9' ,
433               2 text2            char(3)       init (' ms') ;
434
435         dcl 1 sum_avg_turn ,
436               2 blanks           char(8)       init (' ') ,
437               2 text1            char(25)      init
438                                  ('Average turnaround time: ') ,
439               2 tot_turn         pic 'zzzzzzz9v.9' ,
440               2 text2            char(3)       init (' ms') ;
441
442         dcl 1 sum_err ,
443               2 blanks           char(8)       init (' ') ,
444               2 text             char(18)      init
445                                  ('Number of errors: ') ,
446               2 tot_err          pic 'zz9' ;
447
448         dcl 1 sum_thru ,
449               2 blanks           char(8)       init (' ') ,
450               2 text1            char(12)      init ('Throughput:') ,
451               2 tot_thru         pic 'zzzzzzz9v.9' ,
452               2 text2            char(9)       init (' jobs/sec') ;
453
454         dcl 1 sum_util ,
455               2 blanks           char(8)       init (' ') ,
456               2 text             char(17)      init
457                                  ('CPU utilization') ,
458               2 tot_util         pic 'zzzzzzz9v.9' ,
459               2 text2            char(2)       init (' %') ;
460
461         dcl 1 com_hdr1 ,
462               2 blanks1          char(11)     init (' ') ,
463               2 text1            char(7)      init ('ELAPSED') ,
464               2 blanks2          char(20)     init (' ') ,
```

```
465             2 text2             char(10)      init ('TURNAROUND') ,
466             2 blanks3           char(4)       init (' ') ,
467             2 text3             char(7)       init ('INITIAL') ,
468             2 blanks4           char(5)       init (' ') ,
469             2 text4             char(7)       init ('CURRENT') ;
470
471       dcl 1 com_hdr2 ,
472             2 text1             char(6)       init ('JOB ID') ,
473             2 blanks1           char(5)       init (' ') ,
474             2 text2             char(9)       init ('TIME (ms)') ,
475             2 blanks2           char(5)       init (' ') ,
476             2 text3             char(8)       init ('CPU TIME') ,
477             2 blanks3           char(5)       init (' ') ,
478             2 text4             char(9)       init ('TIME (ms)') ,
479             2 blanks4           char(5)       init (' ') ,
480             2 text5             char(8)       init ('PRIORITY') ,
481             2 blanks5           char(4)       init (' ') ,
482             2 text6             char(8)       init ('PRIORITY') ;
483
484       dcl 1 com_out ,
485             2 blanks1           char(1)       init (' ') ,
486             2 job_id            char(3) ,
487             2 blanks2           char(7)       init (' ') ,
488             2 elapse_tm         pic 'zzzzz9' ,
489             2 blanks3           char(8)       init (' ') ,
490             2 cpu_tm            pic 'zzzzz9' ,
491             2 blanks4           char(7)       init (' ') ,
492             2 turn_tm           pic 'zzzzz9' ,
493             2 blanks5           char(10)      init (' ') ,
494             2 init_pri          pic 'z9' ,
495             2 blanks6           char(10)      init (' ') ,
496             2 curr_pri          pic 'z9' ;
497
498       /* Print the summary statistics */
499
500       put file (systat) page ;
501       sum_hdr.text = 'MVMS SUMMARY REPORT' ;
502       put file (systat) skip(2) edit (sum_hdr) (a) ;
503       sum_ent.jobs_in = tot_jobs ;
504       put file (systat) skip(2) edit (sum_ent) (a) ;
505       sum_com.jobs_com = doneq.no_of_entries ;
506       put file (systat) skip(1) edit (sum_com) (a) ;
507       sum_err.tot_err = err_count ;
508       put file (systat) skip (1) edit (sum_err) (a) ;
509       sum_elapse.tot_elapse = tot_time ;
510       put file (systat) skip (1) edit (sum_elapse) (a) ;
511       sum_cpu.tot_cpu = tot_act_time ;
512       put file (systat) skip (1) edit (sum_cpu) (a) ;
513
514       /* Calculate the total turnaround time */
515
516       total_turn_tm = 0 ;
```

```
517         pcbptr = doneq.front_ptr ;
518         do while (pcbptr ^= null()) ;
519           total_turn_tm = total_turn_tm + pcbptr->pcb.turn_tm ;
520           pcbptr = pcbptr->pcb.next_pcb ;
521         end ;
522
523         sum_tot_turn.tot_turn = total_turn_tm ;
524         put file (systat) skip (1) edit (sum_tot_turn) (a) ;
525
526         if total_turn_tm <= 0 | doneq.no_of_entries <= 0
527           then do ;
528             sum_avg_turn.tot_turn = 0 ;
529             sum_thru.tot_thru     = 0 ;
530           end ;
531
532           else do ;
533             sum_avg_turn.tot_turn = round((total_turn_tm /
534                         (decimal(doneq.no_of_entries, 9, 3))), 1) ;
535             sum_thru.tot_thru = round(((1000 *
536                                   decimal(doneq.no_of_entries, 9, 3))
537                                                / total_turn_tm), 1) ;
538           end ;
539
540         put file (systat) skip(1) edit (sum_avg_turn) (a) ;
541         put file (systat) skip(1) edit (sum_thru) (a) ;
542
543         if tot_time <= 0 | tot_act_time <= 0
544           then sum_util.tot_util = 0 ;
545           else sum_util.tot_util = round(((decimal(tot_act_time, 9, 3)
546                             / (decimal(tot_time, 9, 3))) * 100), 1) ;
547
548         put file (systat) skip(1) edit (sum_util) (a) ;
549
550         sum_hdr.text = 'COMPLETED JOBS REPORT' ;
551         put file (systat) skip(4) edit (sum_hdr)  (a) ;
552         put file (systat) skip ;
553         put file (systat) skip(1) edit (com_hdr1) (a) ;
554         put file (systat) skip(1) edit (com_hdr2) (a) ;
555         put file (systat) skip ;
556
557         /* Print the doneq */
558
559         pcbptr = doneq.front_ptr ;
560         do while (pcbptr ^= null()) ;
561           com_out.job_id    = pcbptr->pcb.job_id     ;
562           com_out.elapse_tm = pcbptr->pcb.ready_tm   ;
563           com_out.cpu_tm    = pcbptr->pcb.init_tm    ;
564           com_out.turn_tm   = pcbptr->pcb.turn_tm    ;
565           com_out.init_pri  = pcbptr->pcb.init_pri   ;
566           com_out.curr_pri  = pcbptr->pcb.curr_pri   ;
567           put file (systat) skip(1) edit (com_out) (a) ;
568           pcbptr = pcbptr->pcb.next_pcb ;
```

```
569         end ;
570
571     end sumprt ;
572
573
```

```
574     debugger:  procedure(prior_in, queue, qname_in) ;
575
576        dcl   prior_in  fixed bin(7) ,
577              null      builtin ,
578              pcb_cnt   fixed bin(15) ,
579              qname_in  char(10) ,
580              pcbptr    pointer ;
581
582
583         dcl 1 queue ,
584               2 front_ptr        pointer ,
585               2 back_ptr         pointer ,
586               2 no_of_entries    fixed bin(15) ;
587
588         dcl 1 pcb                based(pcbptr) ,
589               2 job_id           char(3) ,
590               2 state            char(8)       varying ,
591               2 init_pri         fixed bin(7) ,
592               2 curr_pri         fixed bin(7) ,
593               2 init_tm          fixed dec(6) ,
594               2 rem_tm           fixed dec(6) ,
595               2 turn_tm          fixed dec(6) ,
596               2 act_tm           fixed dec(6) ,
597               2 ready_tm         fixed dec(6) ,
598               2 next_pcb         pointer ;
599
600        dcl 1 q_prt,
601              2 text1             char(14) init ('STATUS of *** ') ,
602              2 qname             char(12) ,
603              2 text2             char(18) init ('* with priority =  '),
604              2 prior            pic 'zz9' ;
605
606        dcl 1 pcb_prt1 ,
607              2 blanks1           char(5) init(' ') ,
608              2 text1             char(20) init('Number of entries = ') ,
609              2 ent               pic 'zz9' ;
610
611        dcl 1 pcb_prt2 ,
612              2 blanks1           char(10) init(' ') ,
613              2 text1             char(17) init('Address of PCB = '),
614              2 addr_of           pic 'zz9' ;
615
616        dcl 1 pcb_prt3 ,
617              2 blanks            char(10) init(' ') ,
618              2 text1             char(9) init('Job id = ') ,
619              2 job_id            char(3) ;
620
621        dcl 1 pcb_prt4 ,
622              2 blanks            char(10) init (' ') ,
623              2 text1             char(8) init('State = ') ,
624              2 state             char(8) ;
625
```

```
626         dcl 1 pcb_prt5 ,
627               2 blanks1          char(10) init(' ') ,
628               2 text1            char(19) init('Initial priority = ') ,
629               2 init_pri         pic 'zz9' ;
630
631         dcl 1 pcb_prt6 ,
632                2 blanks1         char(10) init(' ') ,
633                2 text1           char(19) init('Current priority = ') ,
634                2 curr_pri        pic 'zz9' ;
635
636         dcl 1 pcb_prt7 ,
637               2 blanks           char(10) init(' ') ,
638               2 text1            char(17) init('Remaining time = ') ,
639               2 rem_tm           pic 'zzzzz9' ;
640
641         dcl 1 pcb_prt8 ,
642               2 blanks1         char(10) init(' ') ,
643               2 text1           char(18) init('Turnaround time = ') ,
644               2 turn_tm         pic 'zzzzz9' ;
645
646         dcl 1 pcb_prt9 ,
647               2 blanks1         char(10) init(' ') ,
648               2 text1           char(14) init('Active time = ') ,
649               2 act_tm          pic 'zzzzz9' ;
650
651         dcl 1 pcb_prt10 ,
652               2 blanks1        char(10) init(' ') ,
653               2 text1          char(13) init('Ready time = ') ,
654               2 ready_tm       pic 'zzzzz9' ;
655
656         dcl 1 pcb_prt11 ,
657               2 blanks1        char(10) init(' ') ,
658               2 text1          char(19) init('Next PCB address = ' ) ,
659               2 next_pcb       pic 'zz9' ;
660
661         dcl 1 pcb_null ,
662               2 blanks1        char(10) init(' ') ,
663               2 text1          char(25) init('Next PCB address = NULL') ;
664
665         pcb_cnt = 0 ;
666         pcbptr = queue.front_ptr ;
667         q_prt.qname = qname_in ;
668         q_prt.prior = prior_in ;
669         put file (debug) skip ;
670         put file (debug) skip(2) edit (q_prt) (a) ;
671         pcb_prt1.ent = queue.no_of_entries ;
672         put file (debug) skip edit (pcb_prt1) (a) ;
673       do while (pcbptr ^= null());
674         pcb_cnt = pcb_cnt + 1 ;
675         pcb_prt2.addr_of = pcb_cnt ;
676         put file (debug) skip;
677         put file (debug) skip edit (pcb_prt2) (a) ;
```

```
678         pcb_prt3.job_id = pcbptr->pcb.job_id ;
679         put file (debug) skip edit (pcb_prt3) (a) ;
680         pcb_prt4.state = pcbptr->pcb.state ;
681         put file (debug) skip edit (pcb_prt4) (a) ;
682         pcb_prt5.init_pri = pcbptr->pcb.init_pri ;
683         put file (debug) skip edit (pcb_prt5) (a) ;
684         pcb_prt6.curr_pri = pcbptr->pcb.curr_pri ;
685         put file (debug) skip edit (pcb_prt6) (a) ;
686         pcb_prt7.rem_tm = pcbptr->pcb.rem_tm ;
687         put file (debug) skip edit (pcb_prt7) (a) ;
688         pcb_prt8.turn_tm = pcbptr->pcb.turn_tm ;
689         put file (debug) skip edit (pcb_prt8) (a) ;
690         pcb_prt9.act_tm = pcbptr->pcb.act_tm ;
691         put file (debug) skip edit (pcb_prt9) (a) ;
692         pcb_prt10.ready_tm = pcbptr->pcb.ready_tm ;
693         put file (debug) skip edit (pcb_prt10) (a) ;
694         if pcbptr->pcb.next_pcb = null()
695           then put file (debug) skip(2) edit (pcb_null) (a) ;
696           else do ;
697             pcb_prt11.next_pcb = pcb_cnt + 1 ;
698             put file (debug) skip(2) edit (pcb_prt11) (a) ;
699          end ;
700         pcbptr = pcbptr->pcb.next_pcb ;
701       end ;
702
703     end debugger ;
704
705     end mvms ;
```