```
1
2   #include <stdio.h>
3   #include <string.h>
4   #include <sys/fcntl.h>
5   #define FAIL -1
6   #define PAGESIZE 65
7   #define BASEBAUD 4800L
8   #define BASETHRU 2300L
9   #define QSIZE 1001
10  #define PSIZE 256
11  #define DATESZ 11
12  #define TIMESZ 6
13  #define XIDSZ 9
14  #define X25SZ 4
15  #define BLKSZ 9
16  #define COMPSZ 9
17  #define OFFSZ 9
18
19  #define PORTP1 "\nEnter the number of Primary Ports : "
20  #define PORTP2 "\nEnter the number of Secondary Ports (0 for none): "
21  #define BAUDP  "\nEnter the Baud rate for the %d Ports : "
22  #define BADAT1 "ERROR:  Invalid input at line %ld - No %s was found -\n"
23  #define BADAT2 "        Record was skipped as program continued...\n\n"
24  #define BADAT3 "ERROR:  Invalid input at line %ld - Invalid %s, found [%s] -\n"
25  #define BADAT4 "ERROR:  Invalid input at line %ld - %s exceeds max size of %d -\n"
26  #define BADAT5 "ERROR:  Invalid input at line %ld - Invalid transfer rate %f -\n"
27  #define BADAT6 "        Program used 1 block/min and continued...\n\n"
28  #define FATAL1 "\nFATAL ERROR:  Unsorted data detected at line %ld -\n"
29  #define FATAL2 "               Program can not continue...\n"
30  #define LIMBWM "\nDo you wish to limit the BWM size (y/n) ? "
31  #define BWMSIZ "\nEnter the MAXIMUM BWM size in bytes: "
32
33  struct act_q
34  {
35      char   in_use;
36      char   pad[3];
37      long   baud;
38      float  rate_factor;
39      float  dflt_xfer_min;
```

```
40      float  xfer_min;
41      float  blks_left;
42      long   hrs_wait;
43      long   min_wait;
44      long   hrs_act;
45      long   min_act;
46      long   x25_ret;
47      long   hrs_idle;
48      long   min_idle;
49      struct act_q *next;
50
51  } ;
52  struct act_q *act_curblk, *act_prevblk;
53
54  struct
55  {
56      long   act_cnt;
57      struct act_q *start;
58      struct act_q *end;
59
60  }  act_head;
61
62  struct wait_q
63  {
64      float  blks_left;
65      long   hrs_wait;
66      long   min_wait;
67      long   wrk_hrs;
68      long   wrk_min;
69      long   x25_ret;
70      struct wait_q *next;
71
72  } ;
73  struct wait_q *wait_curblk, *wait_prevblk;
74
75  struct
76  {
77      long   wait_cnt;
78      struct wait_q *start;
```

```
79      struct wait_q *end;
80
81  }  wait_head;
82
83  struct
84  {
85      char  beg_date[DATESZ];        /* 01/01/1991 */
86      char  beg_time[TIMESZ];        /* 00:00      */
87      char  beg_xid[XIDSZ];          /* 00:00      */
88      char  end_time[TIMESZ];        /* 00:00      */
89      char  time_in_q[TIMESZ];       /* 00:00      */
90      char  time_xfer[TIMESZ];       /* 00:00      */
91      char  x25_ret[X25SZ];
92      char  blocks[BLKSZ];
93      char  comp_id[COMPSZ];
94      char  office_id[OFFSZ];
95
96  } data_q[2];
97
98  struct
99  {
100     long  month;
101     long  date;
102     long  year;
103     long  beg_hrs;
104     long  beg_min;
105     long  wrk_hrs;
106     long  wrk_min;
107     float wrk_blk;
108     long  x25_ret;
109
110 } request[2];
111
112 struct
113 {
114     long  no_requests;
115     long  hrs_act;
116     long  min_act;
117     long  hrs_wait;
```

```
118       long  min_wait;
119
120   } req_q[QSIZE];
121
122   struct
123   {
124       long  no_requests;
125       long  hrs_wait;
126       long  min_wait;
127
128   } que_q[QSIZE];
129
130   long  mon_cnt;
131   long  day_cnt;
132   long  year_cnt;
133
134   long  hour_cnt;
135   long  min_cnt;
136
137   long  pgm_hrs;
138   long  pgm_min;
139
140   long  record_cnt;
141   float x25err_cnt;
142
143   int   ports;
144   int   ports1;
145   int   ports2;
146
147   long  baud;
148   long  baud1;
149   long  baud2;
150
151   char  pgm_start[40];
152
153   /* BOF - main */
154
155   int
156   main()
```

```
{
     FILE  *fopen();
     FILE  *fn_opn();
     FILE  *reqfp;

     int    get_num();
     int    vfy_chk();
     int    fclose();
     void   prt_page();
     void   que_unload();
     void   que_load();
     void   que_prt();

     extern char *malloc();
     extern char *strtok();
     extern void  free();
     extern void  rewind();
     extern void  exit();
     extern long  atol();

     extern int errno;

     char   buffer1[256];
     char   buffer2[256];
     char   save_buf[256];
     char   tok_buf[80];

     char   last_buf[40];
     char   this_buf[40];

     long   days_month[13];

     char  *rlnptr;
     char  *token_ptr;

     float  rfact;
     float  rfact1;
     float  rfact2;

```

```
196         float  xfer;
197         float  blkmin1;
198         float  blkmin2;
199         float  bwmlimit;
200
201         long   time_incr;
202         long   temp;
203
204         int    rc;
205         int    i;
206         int    pos;
207         int    curpos;
208         int    nxtpos;
209         int    line_cnt;
210         int    page_cnt;
211
212         float ave_ser;
213         float ave_act;
214         float ave_wait;
215         float ave_idl;
216
217         float ind_act;
218         float ind_que;
219         float ind_per;
220
221         float  tot_idl_min;
222         float  tot_idl_hrs;
223         float  tot_wait_min;
224         float  tot_wait_hrs;
225         float  tot_act_min;
226         float  tot_act_hrs;
227         float  tot_req_hrs;
228
229         float  tot_req;
230         float  tot_wait_req;
231         float  wait_per;
232         float  tot_xfer;
233
234         double ave_blk;
```

```
235         float  thrupt1;
236         float  thrupt2;
237
238
239         /*****************************************/
240         /* Initialize the number of days in a year */
241         /*****************************************/
242
243         days_month[1]  = 31L;
244         days_month[2]  = 28L;
245         days_month[3]  = 31L;
246         days_month[4]  = 30L;
247         days_month[5]  = 31L;
248         days_month[6]  = 30L;
249         days_month[7]  = 31L;
250         days_month[8]  = 31L;
251         days_month[9]  = 30L;
252         days_month[10] = 31L;
253         days_month[11] = 30L;
254         days_month[12] = 30L;
255
256         /*************************************/
257         /* Start the time at 0 minutes 0 hours */
258         /*************************************/
259
260         mon_cnt    = 0L;
261         day_cnt    = 0L;
262         year_cnt   = 0L;
263
264         hour_cnt   = 0L;
265         min_cnt    = 0L;
266
267         pgm_hrs    = 0L;
268         pgm_min    = 0L;
269
270         tot_xfer   = 0.0;
271         x25err_cnt = 0.0;
272
273         /************************/
```

```
274         /* Initialize the data_q */
275         /************************/
276
277         for (i=0; i<2; i++)
278         {
279             (void) sprintf(data_q[i].beg_date, "%d", i);
280             (void) sprintf(data_q[i].beg_time, "%d", i);
281             (void) sprintf(data_q[i].beg_xid, "%d", i);
282             (void) sprintf(data_q[i].end_time, "%d", i);
283             (void) sprintf(data_q[i].time_in_q, "%d", i);
284             (void) sprintf(data_q[i].time_xfer, "%d", i);
285             (void) sprintf(data_q[i].x25_ret, "%d", i);
286             (void) sprintf(data_q[i].blocks, "%d", i);
287             (void) sprintf(data_q[i].comp_id, "%d", i);
288             (void) sprintf(data_q[i].office_id, "%d", i);
289         }
290
291         /* Assign an address to the the token_ptr */
292
293         token_ptr = tok_buf;
294
295         /*****************************************/
296         /* Prompt user for primary port assignment */
297         /*****************************************/
298
299         ports1 = 0;
300         while (1)
301         {
302             rc = get_num(PORTP1, buffer1);
303             if (rc != FAIL)
304             {
305                 ports1 = atoi(buffer1);
306                 if (ports1 > 0)
307                 {
308                     break;
309                 }
310             }
311         }
312
```

```
313         (void) sprintf(buffer2, BAUDP, ports1);
314
315         rfact1 = 0.0;
316         while (1)
317         {
318             (void) get_num(buffer2, buffer1);
319             baud1 = atoi(buffer1);
320             switch (baud1)
321             {
322                 case 4800L:
323                 {
324                     rfact1 = (float) baud1 / (float) BASEBAUD;
325                     break;
326                 }
327                 case 9600L:
328                 {
329                     rfact1 = 1.30;
330                     break;
331                 }
332                 case 9601L:
333                 {
334                     rfact1 = 9600.00 / (float) BASEBAUD;
335                     break;
336                 }
337                 case 28000L:
338                 {
339                     rfact1 = (float) baud1 / (float) BASEBAUD;
340                     break;
341                 }
342             }
343             if (rfact1 > 0.0)
344                 break;
345         }
346
347         /*******************************************/
348         /* Prompt user for secondary port assignment */
349         /*******************************************/
350
351         rfact2 = 1.0;
```

```
352         while(1)
353         {
354             rc = get_num(PORTP2, buffer1);
355             if (rc != FAIL)
356             {
357                 ports2 = atoi(buffer1);
358                 if (ports2 >= 0)
359                 {
360                     break;
361                 }
362             }
363         }
364
365         if (ports2 > 0)
366         {
367             (void) sprintf(buffer2, BAUDP, ports2);
368
369             rfact2 = 0.0;
370             baud2  = 0L;
371             while (1)
372             {
373                 (void) get_num(buffer2, buffer1);
374                 baud2 = atoi(buffer1);
375                 switch (baud2)
376                 {
377                     case 4800L:
378                     {
379                         rfact2 = (float) baud2 / (float) BASEBAUD;
380                         break;
381                     }
382                     case 9600L:
383                     {
384                         rfact2 = 1.30;
385                         break;
386                     }
387                     case 9601L:
388                     {
389                         rfact2 = 9600.00 / (float) BASEBAUD;
390                         break;
```

```
            }
            case 28000L:
            {
                rfact2 = (float) baud2 / (float) BASEBAUD;
                break;
            }
        }
        if (rfact2 > 0.0)
            break;

    }   /* end of while */
}

ports = ports1 + ports2;


/*************************/
/* OPEN the request file */
/*************************/

reqfp = fn_opn("\nEnter filename containing the requests: ",
                  buffer1, "r");


(void) fprintf(stderr, "\n\tREADING and ANALYZING data...\n");

/************************************************/
/* READ the request file and determine the average */
/* base throughput and average block size.         */
/************************************************/

/*********************************/
/* 1st PASS - READ the 1st record */
/*********************************/

record_cnt = 1L;
tot_req    = 0.0;

rlnptr = fgets(buffer1, 256, reqfp);
```

```
430         if (rlnptr == NULL)
431         {
432             (void) fprintf(stderr,"ERROR:  1st PASS - retrieving request errno=%d file\n",errno);
433             exit(-1);
434 
435         }
436 
437         /************************************************/
438         /* 1st PASS - Collect the data from the 1st record */
439         /************************************************/
440 
441         /* Transfer the data from buffer1 to the data_q array */
442 
443         (void) strcpy(save_buf, buffer1);
444 
445         /***********************************************/
446         /* Extract the data from the buffer and verify */
447         /* the data as good data.                      */
448         /***********************************************/
449 
450         curpos = 0;
451         if ((rc = vfy_chk(buffer1, curpos, save_buf )) == FAIL)
452         {
453             (void) fprintf(stderr,"\nERROR:  1st PASS - PROGRAM ABORTING due to FATAL error \n");
454             exit(-1);
455         }
456 
457         ave_blk = (double) 0;
458         thrupt1 = 0.0;
459         thrupt2 = 0.0;
460 
461         nxtpos  = 1;
462         while (1)
463         {
464 
465             /*******************************/
466             /* 1st PASS - READ next request */
467             /*******************************/
468 
```

```
469         record_cnt = record_cnt + 1L;
470         rlnptr = fgets(buffer2, 256, reqfp);
471
472         if (rlnptr == NULL)
473             break;
474
475         (void) strcpy(save_buf, buffer2);
476
477         /*********************************************/
478         /* Extract the data from the buffer and verify */
479         /* the data as good data.                      */
480         /*********************************************/
481
482         if ((rc = vfy_chk(buffer2, nxtpos, save_buf )) == FAIL)
483             continue;
484
485         if ((strcmp(data_q[curpos].beg_xid, data_q[nxtpos].beg_xid)) == 0)
486         {
487             request[curpos].wrk_blk = request[curpos].wrk_blk +
488                                       request[nxtpos].wrk_blk;
489
490             request[curpos].wrk_min = request[curpos].wrk_min +
491                                       request[nxtpos].wrk_min;
492             temp                    = request[curpos].wrk_min / 60L;
493             request[curpos].wrk_min = request[curpos].wrk_min -
494                                       (temp * 60L);
495             request[curpos].wrk_hrs = request[curpos].wrk_hrs +
496                                       request[nxtpos].wrk_hrs +
497                                       temp;
498
499             continue;
500         }
501
502
503         /*************************************************/
504         /* 1st PASS - REQUEST is complete - Process request */
505         /*************************************************/
506
507         /* Calculate ave_blk, thruput1 and thruput2 */
```

```
508
509             xfer     =   ( (float) request[curpos].wrk_hrs * 60.00 )
510                      +     (float) request[curpos].wrk_min;
511             if ((xfer > 0.0) &&  (request[curpos].x25_ret == 0L))
512             {
513                 xfer     = request[curpos].wrk_blk / xfer;
514                 thrupt1  = thrupt1 + (xfer * rfact1);
515                 if (ports2 > 0)
516                     thrupt2  = thrupt2 + (xfer * rfact2);
517
518                 ave_blk  = ave_blk + (double) request[curpos].wrk_blk;
519
520                 tot_req  = tot_req + 1.0;
521             }
522
523             /*****************************************************/
524             /* 1st PASS - Rearrange the indexes for the next read */
525             /*****************************************************/
526
527             pos    = curpos;
528             curpos = nxtpos;
529             nxtpos = pos;
530
531         }
532
533         /* Calculate the last the last ave_blk, thruput1 and thruput2 */
534
535         xfer     =   ( (float) request[0].wrk_hrs * 60.00 )
536                  +     (float) request[0].wrk_min;
537         if ((xfer > 0.0) &&  (request[0].x25_ret == 0L))
538         {
539             xfer     = request[0].wrk_blk / xfer;
540             thrupt1  = thrupt1 + (xfer * rfact1);
541             if (ports2 > 0)
542                 thrupt2  = thrupt2 + (xfer * rfact2);
543
544             ave_blk  = ave_blk + (double) request[0].wrk_blk;
545
546             tot_req  = tot_req + 1.0;
```

```
547       }
548
549
550       /*************************/
551       /* Calculate the averages */
552       /*************************/
553
554       ave_blk = ave_blk / (double) tot_req;
555       thrupt1 = thrupt1 /  tot_req;
556       blkmin1 = thrupt1;
557
558       if (ports2 > 0)
559       {
560           thrupt2 = thrupt2 / (double) tot_req;
561           blkmin2 = thrupt2;
562       }
563   /*
564   fprintf(stderr,"\n blkmin1=%f   blkmin2=%f \n", blkmin1,blkmin2);
565   fprintf(stderr,"rfact1=%f   rfact2=%f   ports=%d   ports1=%d   ports2=%d\n",
566   rfact1,rfact2,ports,ports1,ports2);
567   */
568
569       /* Rewind the file to the beginning */
570
571       rewind(reqfp);
572
573       (void) fprintf(stderr, "\n\tThere were %8.0f requests with X25 return codes of 0\n",
574                              tot_req);
575       (void) fprintf(stderr, "\tThe average block size is: %8.0f blocks\n",
576                              (float) ave_blk);
577
578
579       /*************************/
580       /* Initialize the Queues */
581       /*************************/
582
583       baud  = baud1;
584       xfer  = blkmin1;
585       rfact = rfact1;
```

```
586
587         /* Initialize the head tables */
588
589         act_head.act_cnt   = 0L;
590         act_head.start     = NULL;
591         act_head.end       = NULL;
592
593         wait_head.wait_cnt = 0L;
594         wait_head.start    = NULL;
595         wait_head.end      = NULL;
596
597         /*****************************************/
598         /* Allocate and initialize the active queue */
599         /*****************************************/
600
601         for (i=1; i<=ports; i++)
602         {
603             if (i > ports1)
604             {
605                 baud  = baud2;
606                 rfact = rfact2;
607                 xfer  = blkmin2;
608             }
609
610             act_curblk = (struct act_q *) malloc(sizeof(struct act_q));
611             if (act_curblk == NULL)
612             {
613                 (void) fprintf(stderr, "Error: malloc has failed to allocate an act_q member\n");
614                 exit(-1);
615             }
616             if (i == 1)
617             {
618                 act_head.start    = act_curblk;
619             }
620             else
621             {
622                 act_prevblk->next = act_curblk;
623             }
624
```

```
625         /* Set the previous block pointer to the current allocated block */
626         /* Set the next block pointer to NULL.                           */
627
628         act_prevblk              = act_curblk;
629         act_head.end             = act_curblk;
630         act_curblk->next         = NULL;
631
632         act_curblk->in_use       =  'N';
633         act_curblk->baud         =  baud;
634         act_curblk->rate_factor  =  rfact;
635         act_curblk->dflt_xfer_min =  xfer;
636         act_curblk->xfer_min     =  xfer;
637         act_curblk->blks_left    =  0.0;
638         act_curblk->hrs_act      =  0L;
639         act_curblk->min_act      =  0L;
640         act_curblk->hrs_wait     =  0L;
641         act_curblk->min_wait     =  0L;
642         act_curblk->x25_ret      =  0L;
643         act_curblk->hrs_idle     =  0L;
644         act_curblk->min_idle     =  0L;
645     }
646
647     for (i=1; i<QSIZE; i++)
648     {
649         /* request queue */
650         req_q[i].no_requests =  0L;
651         req_q[i].hrs_act     =  0L;
652         req_q[i].min_act     =  0L;
653         req_q[i].hrs_wait    =  0L;
654         req_q[i].min_wait    =  0L;
655
656         /* queue time */
657         que_q[i].no_requests =  0L;
658         que_q[i].hrs_wait    =  0L;
659         que_q[i].min_wait    =  0L;
660     }
661
662     /*****************************************************/
663     /* Allow an option for the user to limit the BWM size */
```

```
664         /*****************************************************/
665
666         bwmlimit = 0.0;
667         while (1)
668         {
669             (void) fprintf(stderr, LIMBWM);
670             if ((gets(buffer1)) != NULL)
671             {
672                 if ((buffer1[0] == 'y') || (buffer1[0] == 'Y'))
673                 {
674                     (void) fprintf(stderr, "\n");
675                     rc = get_num(BWMSIZ, buffer1);
676                     if (rc != FAIL)
677                     {
678                         temp = atol(buffer1);
679                         if (temp >= 0L)
680                         {
681                             bwmlimit = (float) temp;
682                             break;
683                         }
684                     }
685                 }
686                 else
687                 {
688                     if ((buffer1[0] == 'n') || (buffer1[0] == 'N'))
689                         break;
690                 }
691             }
692             else (void) fprintf(stderr, "\n");
693         }
694
695         (void) fprintf(stderr, "\n\n\tQUEUE Simulation EXECUTING...\n");
696
697         /*********************************/
698         /* 2nd PASS - READ the 1st record */
699         /*********************************/
700
701         record_cnt = 1;
702         rlnptr = fgets(buffer1, 256, reqfp);
```

```
703         if (rlnptr == NULL)
704         {
705             (void) fprintf(stderr,"ERROR:  2nd PASS - retrieving request errno=%d file\n",errno);
706             exit(-1);
707         }
708 
709         /************************************************/
710         /* 2nd PASS - Collect the data from the 1st record */
711         /************************************************/
712 
713         /* Transfer the data from buffer1 to the data_q array */
714 
715         (void) strcpy(save_buf, buffer1);
716 
717         /************************************************/
718         /* Extract the data from the buffer and verify */
719         /* the data as good data.                       */
720         /************************************************/
721 
722         curpos = 0;
723         if ((rc = vfy_chk(buffer1, curpos, save_buf )) == FAIL)
724         {
725             (void) fprintf(stderr,"\nERROR:  2nd PASS - PROGRAM ABORTING due to FATAL error \n");
726             exit(-1);
727         }
728 
729         nxtpos = 1;
730         while (rlnptr != NULL)
731         {
732 
733             /*******************************/
734             /* 2nd PASS - READ next request */
735             /*******************************/
736 
737             record_cnt = record_cnt + 1;
738 
739             rlnptr = fgets(buffer2, 256, reqfp);
740 
741             if (rlnptr == NULL)
```

```
742         {
743             break;
744         }
745
746         (void) strcpy(save_buf, buffer2);
747
748         /*********************************************/
749         /* Extract the data from the buffer and verify */
750         /* the data as good data.                      */
751         /*********************************************/
752
753         if ((rc = vfy_chk(buffer2, nxtpos, save_buf )) == FAIL)
754             continue;
755
756         /* Is this request a continuation from the last one? */
757         /* (i.e. passwords are the same).                    */
758
759         if ((strcmp(data_q[curpos].beg_xid, data_q[nxtpos].beg_xid)) == 0)
760         {
761             request[curpos].wrk_blk = request[curpos].wrk_blk +
762                                       request[nxtpos].wrk_blk;
763
764             request[curpos].wrk_min = request[curpos].wrk_min +
765                                       request[nxtpos].wrk_min;
766             temp                    = request[curpos].wrk_min / 60L;
767             request[curpos].wrk_min = request[curpos].wrk_min -
768                                       (temp * 60L);
769             request[curpos].wrk_hrs = request[curpos].wrk_hrs +
770                                       request[nxtpos].wrk_hrs +
771                                       temp;
772
773             continue;
774         }
775
776         /*************************************************/
777         /* 2nd PASS - REQUEST is complete - Process request */
778         /*************************************************/
779
780         /* Save the current date and time so we can tell if a minute */
```

```
781          /* has elapsed between each request (i.e. so we can add a    */
782          /* minute or add zero minutes to time_elapsed.               */
783
784          (void) sprintf(this_buf, "%s %s",
785                                   data_q[curpos].beg_date,
786                                   data_q[curpos].beg_time);
787
788          /*****************************/
789          /* Loop through each request */
790          /*****************************/
791
792
793  /*
794  printf("REQUEST [curpos]   blocks=%f   Transfer time=%ld:%ld\n", request[curpos].wrk_blk,
795  request[curpos].wrk_hrs,request[curpos].wrk_min);
796  */
797          while (1)
798          {
799
800              /* Increment the time by either 0 or 1 minutes */
801
802              if (mon_cnt == 0L)
803              {
804                  mon_cnt   = request[curpos].month;
805                  day_cnt   = request[curpos].date;
806                  year_cnt  = request[curpos].year;
807
808                  hour_cnt  = request[curpos].beg_hrs;
809                  min_cnt   = request[curpos].beg_min;
810
811                  (void) strcpy(pgm_start, this_buf);
812                  (void) strcpy(last_buf, this_buf);
813              }
814
815              if ((strcmp(last_buf, this_buf)) == 0)
816              {
817                  time_incr = 0L;
818              }
819              else
```

```
820             {
821                 time_incr = 1L;
822             }
823
824             if (time_incr == 1L)
825             {
826                 pgm_min  = pgm_min  + 1L;
827                 if (pgm_min == 60L)
828                 {
829                     pgm_hrs = pgm_hrs + 1L;
830                     pgm_min = 0L;
831                 }
832
833                 min_cnt  = min_cnt  + 1L;
834                 if (min_cnt == 60L)
835                 {
836                     hour_cnt = hour_cnt + 1L;
837                     min_cnt = 0L;
838                 }
839                 if (hour_cnt == 24L)
840                 {
841                     day_cnt  = day_cnt + 1L;
842                     hour_cnt = 0L;
843
844                 }
845                 if (day_cnt > days_month[mon_cnt])
846                 {
847                     mon_cnt = mon_cnt + 1L;
848                     day_cnt = 1L;
849 /*
850 fprintf(stderr,"\n*** NEW DAY ***\n");
851 fprintf(stderr,"record_cnt = %ld\n",record_cnt);
852 fprintf(stderr,"\n\n\n*** MAIN:  time = %ld hour(s) %ld minutes ***\n", hour_cnt, min_cnt);
853
854 fprintf(stderr,"    MAIN:  Jobs ACTIVE=%ld   Jobs WAITING=%ld\n\n", act_head.act_cnt,
855 wait_head.wait_cnt);
856 */
857                 }
858                 if (mon_cnt > 12L)
```

```
                {
                    year_cnt = year_cnt + 1L;
                    mon_cnt = 1L;
                }
            }

/*
if (record_cnt < 10)
{
fprintf(stderr,"\nmain:  %ld/%ld/%ld %ld:%ld\n", mon_cnt, day_cnt, year_cnt, hour_cnt, min_cnt);

fprintf(stderr,"       *** REQUEST TO PROCESS ***\nmain:  request[curpos] %ld / %ld / %ld
%ld:%ld\n",
request[curpos].month,request[curpos].date,request[curpos].year,request[curpos].beg_hrs,request[cur
pos].beg_min);

fprintf(stderr,"main:  SYSTEM     %ld / %ld / %ld    %ld:%ld\n",
mon_cnt,day_cnt,year_cnt,hour_cnt,min_cnt);

fprintf(stderr,"main:  request[curpos]  wrk_hrs=%ld wrk_min=%ld\n",
request[curpos].wrk_hrs,request[curpos].wrk_min);

fprintf(stderr,"main:  request[curpos]  JOB byte size=%f    x25_ret=%ld\n\n",
request[curpos].wrk_blk,request[curpos].x25_ret);

fprintf(stderr,"\t       *** NEXT REQUEST ***\n\tmain:  request[nxtpos] mon=%ld date=%ld year=%ld
beg_hrs=%ld\n",
request[nxtpos].month,request[nxtpos].date,request[nxtpos].year,request[nxtpos].beg_hrs);

fprintf(stderr,"\tmain:  request[nxtpos] beg_min=%ld wrk_hrs=%ld wrk_min=%ld\n",
request[nxtpos].beg_min,request[nxtpos].wrk_hrs,request[nxtpos].wrk_min);

fprintf(stderr,"\tmain:  request[nxtpos] wrk_blk=%f x25_ret=%ld\n\n",
request[nxtpos].wrk_blk,request[nxtpos].x25_ret);
}
*/

            /*********************************/
            /* Service OLD previous requests */
```

```
898             /*********************************/
899
900             if (time_incr > 0L)
901                 (void) que_unload(&tot_xfer, save_buf);
902
903             /*************************************************/
904             /* Service a NEW request only if the time is  */
905             /* the current time.                             */
906             /*************************************************/
907
908             if ( (request[curpos].month   == mon_cnt ) &&
909                  (request[curpos].date    == day_cnt ) &&
910                  (request[curpos].year    == year_cnt) &&
911                  (request[curpos].beg_hrs == hour_cnt) &&
912                  (request[curpos].beg_min == min_cnt ) )
913             {
914                if (bwmlimit > 0.0)
915                {
916                    /*******************************/
917                    /* Assign the maximum bwm size */
918                    /* if limit is exceeded        */
919                    /*******************************/
920
921                    if (request[curpos].wrk_blk > bwmlimit)
922                        request[curpos].wrk_blk = bwmlimit;
923                }
924
925                (void) que_load(request[curpos].x25_ret,
926                                request[curpos].wrk_blk,
927                                request[curpos].wrk_hrs,
928                                request[curpos].wrk_min,
929                                save_buf);
930                break;
931             }
932             else
933             {
934                 if ( (request[curpos].month   <= mon_cnt ) &&
935                      (request[curpos].date    <= day_cnt ) &&
936                      (request[curpos].year    <= year_cnt) &&
```

```
                    (request[curpos].beg_hrs <= hour_cnt) &&
                    (request[curpos].beg_min <= min_cnt ) )
                {
               /*************************/
               /* Unsorted data detected */
               /*************************/

                    (void) fprintf(stderr, FATAL1, record_cnt);
                    (void) fprintf(stderr,"%s\n",save_buf);
                    (void) fprintf(stderr, FATAL2);

                    exit(-1);
                }
            }
/*
{
static int iii = 0;
if ((wait_head.wait_cnt > 0) && (iii < 5))
{
iii++;
fprintf(stderr,"\n\n\n*** MAIN:  time = %ld hour(s) %ld minutes ***\n", hour_cnt, min_cnt);

fprintf(stderr,"    MAIN:  Jobs ACTIVE=%ld   Jobs WAITING=%ld\n\n", act_head.act_cnt,
wait_head.wait_cnt);
(void) que_prt();
}
}
*/
        }   /* end while(1) - loop until service a request */


        /*************************************************/
        /* Copy the current buffer to being the last buffer */
        /*************************************************/

        (void) sprintf(last_buf, "%s %s", data_q[curpos].beg_date,
                                          data_q[curpos].beg_time);

        /*************************************************/
```

```
976         /* 2nd PASS - Rearrange the indexes for the next read */
977         /******************************************************/
978
979         pos    = curpos;
980         curpos = nxtpos;
981         nxtpos = pos;
982
983
984       }   /* end of while */
985
986       /******************************************************/
987       /* No more data left to read - unload existing queues */
988       /******************************************************/
989
990       time_incr = 1L;
991       while (act_head.act_cnt > 0L)
992       {
993           /* Increment the minute and hour count */
994
995           pgm_min  = pgm_min  + 1L;
996           if (pgm_min == 60L)
997           {
998               pgm_hrs = pgm_hrs + 1L;
999               pgm_min = 0L;
000           }
001
002           min_cnt  = min_cnt  + 1L;
003           if (min_cnt == 60L)
004           {
005               hour_cnt = hour_cnt + 1L;
006               min_cnt  = 0L;
007           }
008           if (hour_cnt == 24L)
009           {
010               day_cnt  = day_cnt + 1L;
011               hour_cnt = 0L;
012
013           }
014           if (day_cnt > days_month[mon_cnt])
```

```
015             {
016                 mon_cnt = mon_cnt + 1L;
017                 day_cnt = 1L;
018             }
019             if (mon_cnt > 12L)
020             {
021                 year_cnt = year_cnt + 1L;
022                 mon_cnt = 1L;
023             }
024
025             (void) que_unload(&tot_xfer, save_buf);
026         }
027
028         /* Print the statistics for this job */
029
030         tot_req      = 0.0;
031         tot_act_hrs  = 0.0;
032         tot_act_min  = 0.0;
033         tot_wait_hrs = 0.0;
034         tot_wait_min = 0.0;
035
036         for (i=1; i<QSIZE; i++)
037         {
038             if (req_q[i].no_requests > 0L)
039             {
040                 tot_req      = tot_req      + (float) req_q[i].no_requests;
041
042                 tot_act_hrs  = tot_act_hrs  + (float) req_q[i].hrs_act    ;
043                 tot_act_min  = tot_act_min  + (float) req_q[i].min_act    ;
044
045                 tot_wait_hrs = tot_wait_hrs + (float) req_q[i].hrs_wait   ;
046                 tot_wait_min = tot_wait_min + (float) req_q[i].min_wait   ;
047             }
048         }
049
050         tot_idl_hrs = 0.0;
051         tot_idl_min = 0.0;
052
053         act_curblk = act_head.start;
```

```
054       while (act_curblk != NULL)
055       {
056           tot_idl_hrs = tot_idl_hrs + (float) act_curblk->hrs_idle;
057           tot_idl_min = tot_idl_min + (float) act_curblk->min_idle;
058           act_curblk  = act_curblk->next;
059       }
060   /*
061   fprintf(stderr,"main:  BEFORE tot_act_hrs=%f, tot_wait_hrs=%f, tot_idl_hrs=%f \n",tot_act_hrs,
062   tot_wait_hrs, tot_idl_hrs);
063   */
064       /* Calculate active, wait and idle time in hours */
065
066       tot_act_hrs  = tot_act_hrs  + (tot_act_min  / 60.00);
067       tot_wait_hrs = tot_wait_hrs + (tot_wait_min / 60.00);
068       tot_idl_hrs  = tot_idl_hrs  + (tot_idl_min  / 60.00);
069       pgm_hrs      = pgm_hrs      + (pgm_min      / 60.00);
070
071       tot_req_hrs  = tot_act_hrs  + tot_wait_hrs;
072
073       ave_ser      = tot_req_hrs  / tot_req;
074       ave_act      = tot_act_hrs  / tot_req;
075
076       if (tot_wait_hrs > 0.0)
077       {
078           ave_wait     = tot_wait_hrs / tot_req;
079       }
080       else
081       {
082           ave_wait     = 0.0;
083       }
084
085       ave_idl = ((tot_idl_hrs / (float) ports ) / pgm_hrs) * 100.00;
086
087       /* Calculate the average transfer rate */
088
089       tot_xfer = (tot_xfer / tot_req) * 60.00;
090   /*
091   fprintf(stderr,"main:  AFTER tot_act_hrs=%f, tot_wait_hrs=%f, tot_idl_hrs=%f \n",tot_act_hrs,
092   tot_wait_hrs, tot_idl_hrs);
```

```
093
094  fprintf(stderr,"main:  pgm_hrs=%f, tot_req_hrs=%f \n",pgm_hrs, tot_req_hrs);
095
096  fprintf(stderr,"main:  mon_cnt=%ld / day_cnt=%ld / year_cnt=%ld   hour_cnt=%ld:min_cnt=%ld
097  \n",mon_cnt,day_cnt,year_cnt,hour_cnt,min_cnt);
098
099  fprintf(stderr,"main:  ave_ser=%f, ave_act=%f , ave_waqit=%f, ave_idl=%ld\n",ave_ser,
100  ave_act,ave_wait,ave_idl);
101  (void) que_prt();
102  */
103
104       /*********************/
105       /* PRINT THE RESULTS */
106       /*********************/
107
108       page_cnt = 0;
109       line_cnt = 0;
110
111       /* Convert blocks/min to blocks/hour */
112
113       thrupt1 = thrupt1 * 60.00;
114       if (ports2 > 0)
115           thrupt2 = thrupt2 * 60.00;
116
117       /*****************************************/
118       /* PRINT QUEUE TIME DISTRIBUTION - REPORT 1 */
119       /*****************************************/
120
121       (void) strcpy(tok_buf, "                                QUEUE TIME DISTRIBUTION");
122       (void) prt_page(1, 1, tok_buf, tot_xfer, &line_cnt, &page_cnt,
123                       thrupt1, thrupt2, (float) ave_blk, bwmlimit);
124
125       tot_wait_req = 0.00;
126       for (i=1; i<QSIZE; i++)
127       {
128           if (que_q[i].no_requests > 0L)
129           {
130               if (line_cnt >= (PAGESIZE - 1))
131               {
```

```
132                 (void) printf("                    ==========    ==========    ==========\n\n");
133                 (void) prt_page(1, 1, tok_buf, tot_xfer,
134                                 &line_cnt, &page_cnt, thrupt1, thrupt1,
135                                 (float) ave_blk, bwmlimit);
136             }
137 
138             ind_per = ((float)que_q[i].no_requests/(float)tot_req) * 100.00;
139             tot_wait_req = tot_wait_req + (float) que_q[i].no_requests;
140 
141             if (i == (QSIZE - 1))
142             {
143                 (void) printf("                % 8ld+      % 8ld       %9.1f\n",
144                                 i,
145                                 que_q[i].no_requests,
146                                 ind_per);
147                 line_cnt++;
148             }
149             else
150             {
151                 (void) printf("                % 8ld       % 8ld       %9.1f\n",
152                                 i,
153                                 que_q[i].no_requests,
154                                 ind_per);
155                 line_cnt++;
156             }
157         }
158     }
159 
160 
161     if ((tot_req == 0.0) || (tot_wait_req == 0.0))
162     {
163         wait_per = 0.00;
164     }
165     else
166     {
167         wait_per = (tot_wait_req / tot_req) * 100.00;
168     }
169 
170     if ((line_cnt + 19) >= (PAGESIZE - 1))
```

```
171         {
172             (void) printf("
```

```
\n");
        (void) prt_page(1, 0, tok_buf, tot_xfer, &line_cnt, &page_cnt,
                        thrupt1, thrupt2, (float) ave_blk, bwmlimit);
    }
    else
    {
        (void) printf("                    ==========    ==========    =============\n\n");
        line_cnt = line_cnt + 2;
    }

    (void) printf("TOTAL QUEUED REQUESTS:         %9.0f\n\n", tot_wait_req);
    line_cnt = line_cnt + 2;

    (void) printf("TOTAL SUCCESSFUL REQUESTS:     %9.0f\n\n", tot_req);
    line_cnt = line_cnt + 2;

    (void) printf("TOTAL UNSUCCESSFUL REQUESTS:   %9.0f\n\n", x25err_cnt);
    line_cnt = line_cnt + 2;

    (void) printf("TOTAL QUEUED PERCENTAGE        %9.2f %%\n\n", wait_per);
    line_cnt = line_cnt + 2;

    (void) printf("AVERAGE SERVICE TIME:          %9.2f hours\n\n", ave_ser);
    line_cnt = line_cnt + 2;

    (void) printf("AVERAGE ACTIVE  TIME:          %9.2f hours\n\n", ave_act);
    line_cnt = line_cnt + 2;

    (void) printf("AVERAGE QUEUE   TIME:          %9.2f hours\n\n", ave_wait);
    line_cnt = line_cnt + 2;

    (void) printf("IDLE PORT CONDITION:           %9.2f %%\n", ave_idl);
    line_cnt = line_cnt + 1;

    if (line_cnt < PAGESIZE)
    {
        (void) printf("
```

```
210   \n");
211        }
212
213
214        /****************************************/
215        /* PRINT QUEUE TIME ANALYSIS - REPORT 2 */
216        /****************************************/
217
218        line_cnt = 0;
219        (void) strcpy(tok_buf, "                                QUEUE TIME ANALYSIS");
220        (void) prt_page(1, 1, tok_buf, tot_xfer, &line_cnt, &page_cnt,
221                        thrupt1, thrupt2, (float) ave_blk, bwmlimit);
222
223        ind_que = tot_req - tot_wait_req;
224        ind_per = ((ind_que) / (float)tot_req) * 100.00;
225        (void) printf("                        0          % 8.0f       % 9.1f\n",
226                        ind_que, ind_per);
227        line_cnt++;
228
229        ind_que = (float)que_q[1].no_requests ;
230        ind_per = ((ind_que) / (float)tot_req) * 100.00;
231        (void) printf("                        1          % 8.0f       % 9.1f\n",
232                        ind_que, ind_per);
233        line_cnt++;
234
235        ind_que = (float)que_q[2].no_requests  + (float)que_q[3].no_requests;
236        ind_per = ((ind_que) / (float) tot_req) * 100.00;
237        (void) printf("                        2 -  4     % 8.0f       % 9.1f\n",
238                        ind_que, ind_per);
239        line_cnt++;
240
241        ind_que = (float)que_q[4].no_requests + (float)que_q[5].no_requests;
242        ind_per = ((ind_que) / (float) tot_req) * 100.00;
243        (void) printf("                        4 -  6     % 8.0f       % 9.1f\n",
244                        ind_que, ind_per);
245        line_cnt++;
246
247        ind_que = (float)que_q[6].no_requests + (float)que_q[7].no_requests;
248        ind_per = ((ind_que) / (float) tot_req) * 100.00;
```

```
249         (void) printf("                      6 -  8      % 8.0f       % 9.1f\n",
250                       ind_que, ind_per);
251         line_cnt++;
252
253         ind_que = (float)que_q[8].no_requests  + (float)que_q[9].no_requests +
254                   (float)que_q[10].no_requests + (float)que_q[11].no_requests;
255         ind_per = ((ind_que) / (float) tot_req) * 100.00;
256         (void) printf("                       8 - 12      % 8.0f       % 9.1f\n",
257                       ind_que, ind_per);
258         line_cnt++;
259
260         ind_que = (float)que_q[12].no_requests + (float)que_q[13].no_requests +
261                   (float)que_q[14].no_requests + (float)que_q[15].no_requests;
262         ind_per = ((ind_que) / (float) tot_req) * 100.00;
263         (void) printf("                      12 - 16      % 8.0f       % 9.1f\n",
264                       ind_que, ind_per);
265         line_cnt++;
266
267         ind_que = (float)que_q[16].no_requests + (float)que_q[17].no_requests +
268                   (float)que_q[18].no_requests + (float)que_q[19].no_requests;
269         ind_per = ((ind_que) / (float) tot_req) * 100.00;
270         (void) printf("                      16 - 20      % 8.0f       % 9.1f\n",
271                       ind_que, ind_per);
272         line_cnt++;
273
274         ind_que = (float)que_q[20].no_requests + (float)que_q[21].no_requests +
275                   (float)que_q[22].no_requests + (float)que_q[23].no_requests;
276         ind_per = ((ind_que) / (float) tot_req) * 100.00;
277         (void) printf("                      20 - 24      % 8.0f       % 9.1f\n",
278                       ind_que, ind_per);
279         line_cnt++;
280
281         ind_que = 0.0;
282         for (i=24; i<QSIZE; i++)
283             ind_que = ind_que + (float) que_q[i].no_requests;
284
285         ind_per = ((ind_que) / (float) tot_req) * 100.00;
286         (void) printf("                      24 +         % 8.0f       % 9.1f\n",
287                       ind_que, ind_per);
```

```
288         line_cnt++;
289
290         if ((line_cnt + 19) >= (PAGESIZE - 1))
291         {
292             (void) printf("
```

```
\n");
          (void) prt_page(1, 0, tok_buf, tot_xfer, &line_cnt, &page_cnt,
                          thrupt1, thrupt2, (float) ave_blk, bwmlimit);
      }
      else
      {
          (void) printf("                    ==========    ==========    ============\n\n");
          line_cnt = line_cnt + 2;
      }

      (void) printf("TOTAL QUEUED REQUESTS:           %9.0f\n\n", tot_wait_req);
      line_cnt = line_cnt + 2;

      (void) printf("TOTAL SUCCESSFUL REQUESTS:       %9.0f\n\n", tot_req);
      line_cnt = line_cnt + 2;

      (void) printf("TOTAL UNSUCCESSFUL REQUESTS:     %9.0f\n\n", x25err_cnt);
      line_cnt = line_cnt + 2;

      (void) printf("TOTAL QUEUED PERCENTAGE          %9.2f %%\n\n", wait_per);
      line_cnt = line_cnt + 2;

      (void) printf("AVERAGE SERVICE TIME:            %9.2f hours\n\n", ave_ser);
      line_cnt = line_cnt + 2;

      (void) printf("AVERAGE ACTIVE  TIME:            %9.2f hours\n\n", ave_act);
      line_cnt = line_cnt + 2;

      (void) printf("AVERAGE QUEUE   TIME:            %9.2f hours\n\n", ave_wait);
      line_cnt = line_cnt + 2;

      (void) printf("IDLE PORT CONDITION:             %9.2f %%\n", ave_idl);
      line_cnt = line_cnt + 1;


      if (line_cnt < PAGESIZE)
      {
          (void) printf("
```

```
\n");
     }


     /*****************************************************/
     /* PRINT REQUEST TOTAL TIME DISTRIBUTION - REPORT 3 */
     /*****************************************************/

     line_cnt = 0;
     (void) strcpy(tok_buf, " ");
     (void) prt_page(2, 1, tok_buf, tot_xfer, &line_cnt, &page_cnt,
                     thrupt1, thrupt2, (float) ave_blk, bwmlimit);
     for (i=1; i<QSIZE; i++)
     {
         if (req_q[i].no_requests > 0L)
         {
             if (line_cnt >= (PAGESIZE - 1))
             {
                 (void) printf("
```

```
\n");
                (void) printf("  ============    ===========    ============    =========
=============\n\n");
                (void) prt_page(2, 1, tok_buf, tot_xfer,
                                &line_cnt, &page_cnt, thrupt1, thrupt2,
                                (float) ave_blk, bwmlimit);
            }

            ind_act = ((float) req_q[i].min_act/60.0) + req_q[i].hrs_act;
            ind_act =  ind_act / (float) req_q[i].no_requests;

            ind_que = ((float) req_q[i].min_wait/60.0) + req_q[i].hrs_wait;
            ind_que =  ind_que / (float) req_q[i].no_requests;

            ind_per = ((float)req_q[i].no_requests/(float)tot_req) * 100.00;
            if (i == (QSIZE - 1))
            {
                (void) printf("      %4ld+       %9.1f      %9.1f         %8ld      %9.1f\n",
                           i,
                       ind_act,
                       ind_que,
                          req_q[i].no_requests,
                          ind_per);
                line_cnt = line_cnt + 1;
            }
            else
            {
                (void) printf("      %4ld       %9.1f      %9.1f         %8ld      %9.1f\n",
                                i,
                            ind_act,
                            ind_que,
                               req_q[i].no_requests,
                               ind_per);
                line_cnt = line_cnt + 1;
            }
        }
    }

    if ((line_cnt + 19) >= PAGESIZE)
```

```
389         {
390             (void) printf("
```

```
\n");
         (void) prt_page(1, 0, tok_buf, tot_xfer, &line_cnt, &page_cnt,
                         thrupt1, thrupt2, (float) ave_blk, bwmlimit);
     }
     else
     {
         (void) printf("  ============    ===========    ============    =========
=============\n\n");
         line_cnt = line_cnt + 2;
     }

     (void) printf("                 TOTAL SUCCESSFUL REQUESTS:    %9.0f\n\n", tot_req);
     line_cnt = line_cnt + 2;

     (void) printf("                 TOTAL UNSUCCESSFUL REQUESTS:  %9.0f\n\n", x25err_cnt);
     line_cnt = line_cnt + 2;

     (void) printf("                 TOTAL QUEUED REQUESTS:        %9.0f\n\n", tot_wait_req);
     line_cnt = line_cnt + 2;

     (void) printf("                 TOTAL QUEUED PERCENTAGE:      %9.2f %%\n\n", wait_per);
     line_cnt = line_cnt + 2;

     (void) printf("                 AVERAGE SERVICE TIME:         %9.2f hours\n\n", ave_ser);
     line_cnt = line_cnt + 2;

     (void) printf("                 AVERAGE ACTIVE  TIME:         %9.2f hours\n\n", ave_act);
     line_cnt = line_cnt + 2;

     (void) printf("                 AVERAGE QUEUE   TIME:         %9.2f hours\n\n", ave_wait);
     line_cnt = line_cnt + 2;

     (void) printf("                 IDLE PORT CONDITION:          %9.2f %%\n", ave_idl);
     line_cnt = line_cnt + 1;

     if (line_cnt < PAGESIZE)
     {
         (void) printf("
```

```
\n");
     }
     line_cnt = 0;

     (void) fclose(reqfp);
     return(0);

}

/* EOF - main */


/* BOF - get_num */

int
get_num(msg, buffer)
char *msg;
char *buffer;
{
     short i;

     while(1)
     {
         (void) fprintf(stderr, msg);
         if ((gets(buffer)) != NULL)
         {
             for (i=0; i<strlen(buffer); i++)
             {
                 if ((buffer[i] < '0') || (buffer[i] > '9'))
                 {
                     return(FAIL);
                 }
             }
             if (i  == strlen(buffer))
             {
                 break;
             }

         }
```

```
468           else
469           {
470               return (FAIL);
471           }
472
473       }
474       return(i);
475   }
476
477   /* EOF - get_num */
478
479   /* BOF - fn_opn */
480
481   FILE
482   *fn_opn(msg, buffer, mode)
483   char *msg;
484   char *buffer;
485   char *mode;
486   {
487
488       FILE *fp;
489
490       while(1)
491       {
492           if (strlen(msg) == 0)
493           {
494               fp = fopen(buffer, mode);
495               return(fp);
496           }
497
498           (void) fprintf(stderr, msg);
499           if ((gets(buffer)) != NULL)
500           {
501               if ((fp = fopen(buffer, mode)) != NULL)
502               {
503                   return(fp);
504               }
505           }
506
```

```
507
508       }
509   }
510
511   /* EOF - opn_fn */
512
513   /* BOF - que_unload */
514
515   void
516   que_unload(tot_xfer, save_buf)
517   float *tot_xfer;
518   char  *save_buf;
519   {
520       long  min_time;
521       long  hrs_time;
522       long  tot_time;
523       long  temp1;
524       long  temp2;
525       float t_float;
526
527       /*****************************************************************/
528       /* Move the jobs finished from the active queue to the request queue */
529       /*****************************************************************/
530
531       act_curblk = act_head.start;
532       while (act_curblk != NULL)
533       {
534           if (act_curblk->in_use == 'Y')
535           {
536               act_curblk->min_act   = act_curblk->min_act     + 1L;
537            if (act_curblk->min_act == 60L)
538               {
539                   act_curblk->min_act = 0;
540                   act_curblk->hrs_act = act_curblk->hrs_act + 1L;
541               }
542
543               act_curblk->blks_left = act_curblk->blks_left -
544                                       act_curblk->xfer_min;
545
```

```
546             if (act_curblk->blks_left <= 0.0)
547             {
548                 /* Job finished - move to the request queue */
549
550                 act_curblk->in_use = 'N';
551                 act_head.act_cnt   = act_head.act_cnt - 1L;
552
553                 /*******************************************************/
554                 /* Determine if this was a successful X25 request      */
555                 /*******************************************************/
556
557                 if (act_curblk->x25_ret != 0L)
558                 {
559
560                     /* Throw away the results - x25 error the          */
561                     /* request was put in the active queue as a        */
562                     /* place holder only.                              */
563
564                     x25err_cnt = x25err_cnt + (float) 1;
565
566                     if (act_curblk->next == NULL)
567                     {
568                         break;
569                     }
570                     else
571                     {
572                         act_curblk = act_curblk->next;
573                         continue;
574                     }
575
576                 }
577
578                 /* Convert the minutes to hours to store the data in */
579                 /* the request queue via hours.  Any fractional      */
580                 /* minutes are bumped up into the next queue.        */
581
582                 min_time = act_curblk->min_act + act_curblk->min_wait;
583                 hrs_time = act_curblk->hrs_act + act_curblk->hrs_wait;
584
```

```
585                 /* Determine how many hours in the the combined minutes */
586
587                 tot_time = min_time / 60L;
588                 temp1     = 60L * tot_time;
589                 if (temp1 < min_time)
590                 {
591                     tot_time = tot_time + 1L;
592                 }
593
594                 /* Add the total hours in to tot_time */
595
596                 tot_time = tot_time + hrs_time;
597
598                 if (tot_time >= (long) QSIZE )
599                     tot_time = (long) (QSIZE - 1);
600
601                 /************************/
602                 /* Move the ACTIVE time */
603                 /************************/
604
605                 req_q[tot_time].no_requests = req_q[tot_time].no_requests
606                                             + 1L;
607                 req_q[tot_time].hrs_act     = req_q[tot_time].hrs_act +
608                                               act_curblk->hrs_act;
609                 req_q[tot_time].min_act     = req_q[tot_time].min_act +
610                                               act_curblk->min_act;
611                 *tot_xfer                   = *tot_xfer
612                                             + act_curblk->xfer_min;
613
614                 /* Adjust the minutes to be 59 minutes or less */
615
616                 if (req_q[tot_time].min_act >= 60L)
617                 {
618                    temp1 = req_q[tot_time].min_act / 60L;
619                    temp2 = temp1 * 60L;
620                    req_q[tot_time].min_act = req_q[tot_time].min_act -
621                                                 temp2;
622                    req_q[tot_time].hrs_act = req_q[tot_time].hrs_act +
623                                                 temp1;
```

```
624                }
625
626                /*****************************************/
627                /* Move the WAIT time to the request queue */
628                /*****************************************/
629
630                req_q[tot_time].hrs_wait    = req_q[tot_time].hrs_wait +
631                                              act_curblk->hrs_wait;
632                req_q[tot_time].min_wait    = req_q[tot_time].min_wait +
633                                              act_curblk->min_wait;
634
635                /* Adjust the minutes to be 59 minutes or less */
636
637                if (req_q[tot_time].min_wait >= 60L)
638                {
639                   temp1 = req_q[tot_time].min_wait / 60L;
640                   temp2 = temp1 * 60L;
641                   req_q[tot_time].min_wait = req_q[tot_time].min_wait -
642                                                 temp2;
643                   req_q[tot_time].hrs_wait = req_q[tot_time].hrs_wait +
644                                                 temp1;
645                }
646
647                /*************************************************/
648                /* Store the wait time in seperate array to measure */
649                /* customer queue time.                          */
650                /*                                               */
651                /*      Move the WAIT time to the queue queue    */
652                /*************************************************/
653
654                   tot_time = act_curblk->hrs_wait;
655
656                   if (act_curblk->min_wait > 0L)
657                    tot_time = tot_time + 1L;
658
659                if (tot_time >= (long) QSIZE)
660                    tot_time = (long) (QSIZE - 1);
661
662                que_q[tot_time].no_requests = que_q[tot_time].no_requests
```

```
                                          + 1L;
            que_q[tot_time].hrs_wait    = que_q[tot_time].hrs_wait +
                                          act_curblk->hrs_wait;
            que_q[tot_time].min_wait    = que_q[tot_time].min_wait +
                                          act_curblk->min_wait;

            /* Adjust the minutes to be 59 minutes or less */

            if (que_q[tot_time].min_wait >= 60L)
            {
               temp1 = que_q[tot_time].min_wait / 60L;
               temp2 = temp1 * 60L;
               que_q[tot_time].min_wait = que_q[tot_time].min_wait -
                                             temp2;
               que_q[tot_time].hrs_wait = que_q[tot_time].hrs_wait +
                                             temp1;
            }
        }
    }
    else
    {
        /* This queue is idle */

        act_curblk->min_idle = act_curblk->min_idle + 1L;

        if (act_curblk->min_idle >= 60L)
        {
            act_curblk->min_idle = 0L;
            act_curblk->hrs_idle = act_curblk->hrs_idle + 1L;
        }
    }

    /* Get next active queue member */

    act_curblk = act_curblk->next;
}

/*****************************************************/
/* Increment the wait time for jobs in the wait queue */
```

```
/*******************************************************/

if (wait_head.wait_cnt > 0L)
{
    wait_curblk = wait_head.start;
    while (wait_curblk != NULL)
    {
        /* Adjust minutes to be 59 minutes or less */

        wait_curblk->min_wait = wait_curblk->min_wait + 1L;
        if (wait_curblk->min_wait >= 60)
        {
            wait_curblk->hrs_wait = wait_curblk->hrs_wait + 1L;
            wait_curblk->min_wait = 0L;
        }

        wait_curblk = wait_curblk->next;

    }   /* end of while wait queue entries */

    /*********************************************************/
    /* Move the jobs in the wait queue to the active queue     */
    /* ONLY if ports are available.                            */
    /*********************************************************/

    if (act_head.act_cnt < (long) ports)
    {
        act_curblk  = act_head.start;
        while (act_curblk != NULL)
        {
            if (act_curblk->in_use == 'N')
            {
                act_curblk->in_use    = 'Y';
                wait_curblk           = wait_head.start;

                act_curblk->blks_left = wait_curblk->blks_left;
                act_curblk->hrs_wait  = wait_curblk->hrs_wait ;
                act_curblk->min_wait  = wait_curblk->min_wait ;
                act_curblk->x25_ret   = wait_curblk->x25_ret;
```

```
741
742                     /* Calculate the transfer rate for this request */
743
744                     if ((wait_curblk->x25_ret == 0L) &&
745                        ((wait_curblk->wrk_hrs + wait_curblk->wrk_min) > 0L))
746                     {
747                         /* 1. Convert hours to minutes.             */
748                         /* 2. Divide the blocks by total minutes.   */
749                         /* 3. Multiply rate by rate factor.         */
750
751                         t_float = ((float) wait_curblk->wrk_hrs * 60.00)
752                                 +  (float) wait_curblk->wrk_min;
753 /*
754 printf("\tque_unload: %f blocks took %ld:%ld == %f minutes\n", wait_curblk->blks_left, wait_curblk-
755 >wrk_hrs, wait_curblk->wrk_min, t_float);
756 */
757
758                         t_float  = wait_curblk->blks_left / t_float;
759 /*
760 printf("\tque_unload:  xfer BEFORE %f RATE adjusment = %f\n",act_curblk->rate_factor,t_float);
761 */
762                         act_curblk->xfer_min = t_float *
763                                                act_curblk->rate_factor;
764                         if (act_curblk->xfer_min <= 0.0)
765                         {
766                             (void) fprintf(stderr, BADAT5, record_cnt,
767                                            act_curblk->xfer_min);
768                             (void) fprintf(stderr,"%s\n",save_buf);
769                             (void) fprintf(stderr, BADAT6);
770
771                             act_curblk->xfer_min = 1.0;
772                         }
773                     }
774                     else
775                     {
776                         act_curblk->xfer_min = act_curblk->dflt_xfer_min;
777                     }
778
779 /*
```

```
printf("\tque_unload: Transfer rate: %ld  Record count: %ld\n\n", act_curblk->xfer_min,
record_cnt);
*/
                    act_head.act_cnt      = act_head.act_cnt + 1L;

                    /*****************************/
                    /* Re-adjust the wait queue */
                    /*****************************/

                    wait_curblk     = wait_head.start;
                    wait_head.start = wait_curblk->next;
                    (void) free(wait_curblk);

                    wait_head.wait_cnt = wait_head.wait_cnt - 1L;
                }
                if (wait_head.wait_cnt <= 0L)
                    break;

                act_curblk = act_curblk->next;
            }

         }   /* end if act_head.act_cnt < ports */

     }   /* end if wait_head.wait_cnt > 0 */
}

/* EOF - que_unload */

/* BOF - que_load */

void
que_load(x25ret, block, whrs, wmin, save_buf)
long  x25ret;
float block;
long  whrs;
long  wmin;
char *save_buf;
{
     int    req_todo;
```

```
819       float  temp;
820       int    i;
821
822       /* Determine the average size of the bwms to assign to queues */
823
824       req_todo = 1;
825   /*
826   fprintf(stderr,"que_load:   BWM size = %ld   BWM requests = %d\n",block, req_todo);
827   */
828       /*******************************************************/
829       /* Attempt to put the request first in the active queue */
830       /*******************************************************/
831
832       if (act_head.act_cnt < (long) ports)
833       {
834           act_curblk = act_head.start;
835           while (act_curblk != NULL)
836           {
837               if (act_curblk->in_use == 'N')
838               {
839                   /* Move the request to the active queue */
840
841                   act_curblk->in_use    = 'Y';
842                   act_curblk->blks_left = block;
843                   act_curblk->x25_ret   = x25ret;
844                   act_curblk->hrs_wait  = 0L;
845                   act_curblk->min_wait  = 0L;
846                   act_curblk->hrs_act   = 0L;
847                   act_curblk->min_act   = 0L;
848
849                   /* Calculate the transfer rate for this request */
850
851                   if ((x25ret == 0L) && (whrs + wmin > 0L))
852                   {
853                       /* 1. Convert hours to minutes.             */
854                       /* 2. Divide the blocks by total minutes.   */
855                       /* 3. Multiply rate by rate factor.         */
856
857                       temp = ((float) whrs * 60.00) + (float) wmin;
```

```
/*
printf("\tque_load: %ld blocks took %ld:%ld == %f minutes\n",block,whrs,wmin,temp);
*/
                    temp = block /temp;
/*
printf("\tque_load:  xfer BEFORE %f RATE adjusment = %f\n",act_curblk->rate_factor,temp);
*/
                    act_curblk->xfer_min = temp *
                                           act_curblk->rate_factor;
                    if (act_curblk->xfer_min <= 0.0)
                    {
                        (void) fprintf(stderr, BADAT5, record_cnt,
                                               act_curblk->xfer_min);
                        (void) fprintf(stderr,"%s\n",save_buf);
                        (void) fprintf(stderr, BADAT6);

                        act_curblk->xfer_min = 1.0;
                    }
                }
                else
                {
                    act_curblk->xfer_min = act_curblk->dflt_xfer_min;
                }
/*
printf("\tque_load:  Transfer rate: %ld  Record count: %f\n", act_curblk->xfer_min, record_cnt);
*/

                req_todo          = req_todo - 1;
                act_head.act_cnt  = act_head.act_cnt + 1L;
            }

            /* Break out of the loop if no more requests */

            if (req_todo <= 0)
                break;

            act_curblk = act_curblk->next;
        }

```

```
897       }   /* end if act_head.act_cnt < ports */
898
899       /*********************************************************/
900       /* If more requests then idle ports, place the request in */
901       /* the wait queue.                                       */
902       /*********************************************************/
903
904   /*
905   fprintf(stderr,"que_load:   BWM size = %f   TODO requests = %d\n",block, req_todo);
906   */
907       if (req_todo > 0)
908       {
909           wait_prevblk = wait_head.end;
910           for (i=1; i<=req_todo; i++)
911           {
912               wait_curblk = (struct wait_q *) malloc(sizeof(struct wait_q));
913               if (wait_curblk == NULL)
914               {
915                   (void) fprintf(stderr, "Error: malloc has failed to allocate an wait_q member\n");
916                   exit(-1);
917               }
918               if (wait_head.wait_cnt == 0L)
919               {
920                   wait_head.start = wait_curblk;
921               }
922               else
923               {
924                   wait_prevblk->next = wait_curblk;
925               }
926
927               /* Set the previous block pointer to the current allocated */
928               /* block.  Set the next block pointer to NULL.             */
929
930               wait_prevblk          = wait_curblk;
931               wait_head.end         = wait_curblk;
932
933               wait_curblk->blks_left = block   ;
934               wait_curblk->x25_ret   = x25ret  ;
935               wait_curblk->hrs_wait  = 0L      ;
```

```
936                 wait_curblk->min_wait  = 0L      ;
937                 wait_curblk->wrk_hrs   = whrs    ;
938                 wait_curblk->wrk_min   = wmin    ;
939                 wait_curblk->x25_ret   = x25ret  ;
940
941                 wait_curblk->next      = NULL    ;
942                 wait_head.wait_cnt     = wait_head.wait_cnt + 1L;
943
944             }
945         }
946
947     }
948
949     /* EOF - que_load */
950
951     /* BOF - que_prt */
952
953     void
954     que_prt()
955     {
956         int i;
957         int flag;
958
959         flag=0;
960
961
962         (void) fprintf(stderr,"\n\tACTIVE QUEUE\n");
963         if (act_head.act_cnt <= 0L)
964         {
965             (void) fprintf(stderr,"\t ** none **\n");
966         }
967         else
968         {
969             act_curblk = act_head.start;
970             i=0;
971             while (act_curblk != NULL)
972             {
973                 i = i + 1;
974                 if (act_curblk->in_use == 'Y')
```

```
975             {
976                 (void) fprintf(stderr,"act_q[%d].blks_left = %f\n",i, act_curblk->blks_left);
977                 (void) fprintf(stderr,"act_q[%d].hrs_wait  = %ld\n",i, act_curblk->hrs_wait);
978                 (void) fprintf(stderr,"act_q[%d].min_wait  = %ld\n",i, act_curblk->min_wait);
979                 (void) fprintf(stderr,"act_q[%d].hrs_act   = %ld\n",i, act_curblk->hrs_act);
980                 (void) fprintf(stderr,"act_q[%d].min_act   = %ld\n",i, act_curblk->min_act);
981                 (void) fprintf(stderr,"act_q[%d].hrs_idle  = %ld\n",i,act_curblk->hrs_idle);
982                 (void) fprintf(stderr,"act_q[%d].min_idle  = %ld\n\n",i,act_curblk->min_idle);
983                 (void) fprintf(stderr,"act_q[%d].x25_ret   = %ld\n\n",i,act_curblk->x25_ret);
984             }
985             else
986             {
987                 (void) fprintf(stderr,"act_q[%d].hrs_idle  = %ld\n",i,act_curblk->hrs_idle);
988                 (void) fprintf(stderr,"act_q[%d].min_idle  = %ld\n\n",i,act_curblk->min_idle);
989             }
990 
991             act_curblk = act_curblk->next;
992         }
993     }
994 
995     (void) fprintf(stderr,"\n\tWAIT QUEUE\n");
996     if (wait_head.wait_cnt <= 0L)
997     {
998         (void) fprintf(stderr,"\t ** none **\n");
999     }
000     else
001     {
002         wait_curblk = wait_head.start;
003         i = 0;
004         while (wait_curblk != NULL)
005         {
006             i = i + 1;
007             (void) fprintf(stderr,"wait_q[%d].blks_left = %f\n",i, wait_curblk->blks_left);
008             (void) fprintf(stderr,"wait_q[%d].hrs_wait  = %ld\n",i, wait_curblk->hrs_wait);
009             (void) fprintf(stderr,"wait_q[%d].min_wait  = %ld\n\n",i, wait_curblk->min_wait);
010             (void) fprintf(stderr,"wait_q[%d].x25_ret   = %ld\n\n",i, wait_curblk->x25_ret);
011             wait_curblk = wait_curblk->next;
012         }
013     }
```

```
014
015       (void) fprintf(stderr,"\n\tREQUEST QUEUE\n");
016       for (i=1; i<QSIZE; i++)
017       {
018           if (req_q[i].no_requests > 0L)
019           {
020               if (flag == 0)
021               {
022                    flag=1;
023               }
024               (void) fprintf(stderr,"req_q[%d].no_requests = %ld\n",i, req_q[i].no_requests);
025               (void) fprintf(stderr,"req_q[%d].hrs_act     = %ld\n",i, req_q[i].hrs_act);
026               (void) fprintf(stderr,"req_q[%d].min_act     = %ld\n",i, req_q[i].min_act);
027               (void) fprintf(stderr,"req_q[%d].hrs_wait    = %ld\n",i, req_q[i].hrs_wait);
028               (void) fprintf(stderr,"req_q[%d].min_wait    = %ld\n\n",i, req_q[i].min_wait);
029           }
030       }
031
032       if (flag == 0)
033           (void) fprintf(stderr,"\t ** none **\n\n");
034   }
035   /* EOF - que_prt */
036
037   /* BOF - vfy_chk */
038
039   int
040   vfy_chk(buffer, pos, save_buf)
041   char  *buffer;
042   int    pos;
043   char  *save_buf;
044   {
045       extern char   *strtok();
046
047       char   temp_buf[256];
048       char   tok_buf[80];
049       char  *token_ptr;
050       int    retc;
051
052       int    i;
```

```
053         int    j;
054
055         /***********************************/
056         /* Collect the data from the buffer */
057         /***********************************/
058
059         /* Set up a while loop to avoid a goto */
060
061         token_ptr = tok_buf;
062         retc      = FAIL;
063         while (1)
064         {
065
066             if ((token_ptr = strtok(buffer, " ")) == NULL)
067             {
068                 (void) fprintf(stderr, BADAT1, record_cnt, "beg_date");
069                 (void) fprintf(stderr,"%s\n",save_buf);
070                 (void) fprintf(stderr, BADAT2);
071                 break;
072             }
073             if (strlen(token_ptr) > (DATESZ - 1))
074             {
075                 (void) fprintf(stderr, BADAT4, record_cnt, "beg_date");
076                 (void) fprintf(stderr,"%s\n",save_buf);
077                 (void) fprintf(stderr, BADAT2);
078                 break;
079             }
080             (void) strcpy(data_q[pos].beg_date, token_ptr);
081
082             if ((token_ptr = strtok((char *) NULL, " ")) == NULL)
083             {
084                 (void) fprintf(stderr, BADAT1, record_cnt, "beg_time");
085                 (void) fprintf(stderr,"%s\n",save_buf);
086                 (void) fprintf(stderr, BADAT2);
087                 break;
088             }
089             if (strlen(token_ptr) > (TIMESZ - 1))
090             {
091                 (void) fprintf(stderr, BADAT4, record_cnt, "beg_time");
```

```
092             (void) fprintf(stderr,"%s\n",save_buf);
093             (void) fprintf(stderr, BADAT2);
094             break;
095         }
096         (void) strcpy(data_q[pos].beg_time, token_ptr);
097
098         if ((token_ptr = strtok((char *) NULL, " ")) == NULL)
099         {
100             (void) fprintf(stderr, BADAT1, record_cnt, "beg_xid");
101             (void) fprintf(stderr,"%s\n",save_buf);
102             (void) fprintf(stderr, BADAT2);
103             break;
104         }
105         if (strlen(token_ptr) > (XIDSZ - 1))
106         {
107             (void) fprintf(stderr, BADAT4, record_cnt, "beg_xid");
108             (void) fprintf(stderr,"%s\n",save_buf);
109             (void) fprintf(stderr, BADAT2);
110             break;
111         }
112         (void) strcpy(data_q[pos].beg_xid, token_ptr);
113
114         if ((token_ptr = strtok((char *) NULL, " ")) == NULL)
115         {
116             (void) fprintf(stderr, BADAT1, record_cnt, "end_time");
117             (void) fprintf(stderr,"%s\n",save_buf);
118             (void) fprintf(stderr, BADAT2);
119             break;
120         }
121         if (strlen(token_ptr) > (TIMESZ - 1))
122         {
123             (void) fprintf(stderr, BADAT4, record_cnt, "end_time");
124             (void) fprintf(stderr,"%s\n",save_buf);
125             (void) fprintf(stderr, BADAT2);
126             break;
127         }
128         (void) strcpy(data_q[pos].end_time, token_ptr);
129
130         if ((token_ptr = strtok((char *) NULL, " ")) == NULL)
```

```
131             {
132                 (void) fprintf(stderr, BADAT1, record_cnt, "time_in_q");
133                 (void) fprintf(stderr,"%s\n",save_buf);
134                 (void) fprintf(stderr, BADAT2);
135                 break;
136             }
137             if (strlen(token_ptr) > (TIMESZ - 1))
138             {
139                 (void) fprintf(stderr, BADAT4, record_cnt, "time_in_q");
140                 (void) fprintf(stderr,"%s\n",save_buf);
141                 (void) fprintf(stderr, BADAT2);
142                 break;
143             }
144             (void) strcpy(data_q[pos].time_in_q, token_ptr);
145 
146             if ((token_ptr = strtok((char *) NULL, " ")) == NULL)
147             {
148                 (void) fprintf(stderr, BADAT1, record_cnt, "time_xfer");
149                 (void) fprintf(stderr,"%s\n",save_buf);
150                 (void) fprintf(stderr, BADAT2);
151                 break;
152             }
153             if (strlen(token_ptr) > (TIMESZ - 1))
154             {
155                 (void) fprintf(stderr, BADAT4, record_cnt, "time_xfer");
156                 (void) fprintf(stderr,"%s\n",save_buf);
157                 (void) fprintf(stderr, BADAT2);
158                 break;
159             }
160             (void) strcpy(data_q[pos].time_xfer, token_ptr);
161 
162             if ((token_ptr = strtok((char *) NULL, " ")) == NULL)
163             {
164                 (void) fprintf(stderr, BADAT1, record_cnt, "x25_ret");
165                 (void) fprintf(stderr,"%s\n",save_buf);
166                 (void) fprintf(stderr, BADAT2);
167                 break;
168             }
169             if (strlen(token_ptr) > (X25SZ - 1))
```

```
170             {
171                 (void) fprintf(stderr, BADAT4, record_cnt, "x25_ret");
172                 (void) fprintf(stderr,"%s\n",save_buf);
173                 (void) fprintf(stderr, BADAT2);
174                 break;
175             }
176             (void) strcpy(data_q[pos].x25_ret, token_ptr);
177 
178             if ((token_ptr = strtok((char *) NULL, " ")) == NULL)
179             {
180                 (void) fprintf(stderr, BADAT1, record_cnt, "blocks");
181                 (void) fprintf(stderr,"%s\n",save_buf);
182                 (void) fprintf(stderr, BADAT2);
183                 break;
184             }
185             if (strlen(token_ptr) > (BLKSZ - 1))
186             {
187                 (void) fprintf(stderr, BADAT4, record_cnt, "blocks");
188                 (void) fprintf(stderr,"%s\n",save_buf);
189                 (void) fprintf(stderr, BADAT2);
190                 break;
191             }
192             (void) strcpy(data_q[pos].blocks, token_ptr);
193 
194             if ((token_ptr = strtok((char *) NULL, " ")) == NULL)
195             {
196                 (void) fprintf(stderr, BADAT1, record_cnt, "comp_id");
197                 (void) fprintf(stderr,"%s\n",save_buf);
198                 (void) fprintf(stderr, BADAT2);
199                 break;
200             }
201             if (strlen(token_ptr) > (COMPSZ - 1))
202             {
203                 (void) fprintf(stderr, BADAT4, record_cnt, "comp_id");
204                 (void) fprintf(stderr,"%s\n",save_buf);
205                 (void) fprintf(stderr, BADAT2);
206                 break;
207             }
208             (void) strcpy(data_q[pos].comp_id, token_ptr);
```

```
209
210         if ((token_ptr = strtok((char *) NULL, " ")) == NULL)
211         {
212             (void) fprintf(stderr, BADAT1, record_cnt, "office_id");
213             (void) fprintf(stderr,"%s\n",save_buf);
214             (void) fprintf(stderr, BADAT2);
215             break;
216         }
217         if (strlen(token_ptr) > (OFFSZ - 1))
218         {
219             (void) fprintf(stderr, BADAT4, record_cnt, "office_id");
220             (void) fprintf(stderr,"%s\n",save_buf);
221             (void) fprintf(stderr, BADAT2);
222             break;
223         }
224         (void) strcpy(data_q[pos].office_id, token_ptr);
225
226
227         /*********************************************************/
228         /* Map the character data strings to numeric data fields */
229         /*********************************************************/
230
231         /* Get the begin month and date */
232
233         tok_buf[0] = '\0';
234
235                     /* month */
236         for (i=0; data_q[pos].beg_date[i] != '/'; i++)
237         {
238             temp_buf[i] = data_q[pos].beg_date[i];
239             if ((temp_buf[i]) < '0' || (temp_buf[i] > '9'))
240                 (void) strcpy(tok_buf, "ERROR");
241         }
242         temp_buf[i] = '\0';
243
244         if (strlen(tok_buf) > 0)
245         {
246             (void) fprintf(stderr, BADAT3, record_cnt, "month", temp_buf);
247             (void) fprintf(stderr,"%s\n",save_buf);
```

```
248             (void) fprintf(stderr, BADAT2);
249             break;
250         }
251 
252         request[pos].month = atol(temp_buf);
253 
254                     /* date */
255         for (j=i+1, i=0; data_q[pos].beg_date[j] != '/'; j++)
256         {
257             temp_buf[i] = data_q[pos].beg_date[j];
258             if ((temp_buf[i]) < '0' || (temp_buf[i] > '9'))
259                 (void) strcpy(tok_buf, "ERROR");
260             i++;
261         }
262         temp_buf[i] = '\0';
263 
264         if (strlen(tok_buf) > 0)
265         {
266             (void) fprintf(stderr, BADAT3, record_cnt, "date", temp_buf);
267             (void) fprintf(stderr,"%s\n",save_buf);
268             (void) fprintf(stderr, BADAT2);
269             break;
270         }
271 
272         request[pos].date  = atol(temp_buf);
273 
274                     /* year */
275         for (i=j+1, j=0; data_q[pos].beg_date[i] != '\0'; i++)
276         {
277             temp_buf[j] = data_q[pos].beg_date[i];
278             if ((temp_buf[j]) < '0' || (temp_buf[j] > '9'))
279                 (void) strcpy(tok_buf, "ERROR");
280             j++;
281         }
282         temp_buf[j] = '\0';
283 
284         if (strlen(tok_buf) > 0)
285         {
286             (void) fprintf(stderr, BADAT3, record_cnt, "year", temp_buf);
```

```
287             (void) fprintf(stderr,"%s\n",save_buf);
288             (void) fprintf(stderr, BADAT2);
289             break;
290         }
291 
292         request[pos].year  = atol(temp_buf);
293 
294         /* Get the time in the queues hours and minutes */
295 
296                     /* beg_hrs */
297         for (i=0; data_q[pos].beg_time[i] != ':'; i++)
298         {
299             temp_buf[i] = data_q[pos].beg_time[i];
300             if ((temp_buf[i]) < '0' || (temp_buf[i] > '9'))
301                 (void) strcpy(tok_buf, "ERROR");
302         }
303         temp_buf[i] = '\0';
304 
305         if (strlen(tok_buf) > 0)
306         {
307             (void) fprintf(stderr, BADAT3, record_cnt, "beg_hrs", temp_buf);
308             (void) fprintf(stderr,"%s\n",save_buf);
309             (void) fprintf(stderr, BADAT2);
310             break;
311         }
312 
313         request[pos].beg_hrs = atol(temp_buf);
314 
315                     /* beg_min */
316         for (j=i+1, i=0; data_q[pos].beg_time[j] != '\0'; j++)
317         {
318             temp_buf[i] = data_q[pos].beg_time[j];
319             if ((temp_buf[i]) < '0' || (temp_buf[i] > '9'))
320                 (void) strcpy(tok_buf, "ERROR");
321             i++;
322         }
323         temp_buf[i] = '\0';
324 
325         if (strlen(tok_buf) > 0)
```

```
{
    (void) fprintf(stderr, BADAT3, record_cnt, "beg_min", temp_buf);
    (void) fprintf(stderr,"%s\n",save_buf);
    (void) fprintf(stderr, BADAT2);
    break;
}

request[pos].beg_min = atol(temp_buf);

            /* wrk_hrs */
for (i=0; data_q[pos].time_xfer[i] != ':'; i++)
{
    temp_buf[i] = data_q[pos].time_xfer[i];
    if ((temp_buf[i]) < '0' || (temp_buf[i] > '9'))
        (void) strcpy(tok_buf, "ERROR");
}
temp_buf[i] = '\0';

if (strlen(tok_buf) > 0)
{
    (void) fprintf(stderr, BADAT3, record_cnt, "month", temp_buf);
    (void) fprintf(stderr,"%s\n",save_buf);
    (void) fprintf(stderr, BADAT2);
    break;
}

request[pos].wrk_hrs = atol(temp_buf);

            /* wrk_min */
for (j=i+1, i=0; data_q[pos].time_xfer[j] != '\0'; j++)
{
    temp_buf[i] = data_q[pos].time_xfer[j];
    if ((temp_buf[i]) < '0' || (temp_buf[i] > '9'))
        (void) strcpy(tok_buf, "ERROR");
    i++;
}
temp_buf[i] = '\0';

if (strlen(tok_buf) > 0)
```

```
365         {
366             (void) fprintf(stderr, BADAT3, record_cnt, "wrk_min", temp_buf);
367             (void) fprintf(stderr,"%s\n",save_buf);
368             (void) fprintf(stderr, BADAT2);
369             break;
370         }
371
372         request[pos].wrk_min = atol(temp_buf);
373
374                     /* wrk_blk */
375         for (i=0; data_q[pos].blocks[i] != '\0'; i++)
376         {
377             temp_buf[i] = data_q[pos].blocks[i];
378             if ((temp_buf[i]) < '0' || (temp_buf[i] > '9'))
379                 (void) strcpy(tok_buf, "ERROR");
380         }
381         temp_buf[i] = '\0';
382
383         if (strlen(tok_buf) > 0)
384         {
385             (void) fprintf(stderr, BADAT3, record_cnt, "wrk_blk", temp_buf);
386             (void) fprintf(stderr,"%s\n",save_buf);
387             (void) fprintf(stderr, BADAT2);
388             break;
389         }
390
391         request[pos].wrk_blk = (float) atol(temp_buf);
392
393                     /* x25_ret */
394         for (i=0; data_q[pos].x25_ret[i] != '\0'; i++)
395         {
396             temp_buf[i] = data_q[pos].x25_ret[i];
397             if ((temp_buf[i]) < '0' || (temp_buf[i] > '9'))
398                 (void) strcpy(tok_buf, "ERROR");
399         }
400         temp_buf[i] = '\0';
401
402         if (strlen(tok_buf) > 0)
403         {
```

```
404                 (void) fprintf(stderr, BADAT3, record_cnt, "x25_ret", temp_buf);
405                 (void) fprintf(stderr,"%s\n",save_buf);
406                 (void) fprintf(stderr, BADAT2);
407                 break;
408             }
409
410             request[pos].x25_ret = atol(temp_buf);
411
412             /* Break out of the while loop */
413
414             retc = 0;
415             break;
416
417         }  /* end of while */
418
419         /* Return with the return code: 0 = success,  -1 = fail */
420         return(retc);
421     }
422
423     /* EOF - vfy_chk */
424
425
426     /* BOF - prt_page */
427
428     void
429     prt_page(hdr_type, det_prt, caption, tot_xfer, line_cnt, page_cnt,
430              xfer1, xfer2, ave_blk, limit)
431     int    hdr_type;
432     int    det_prt;
433     char  *caption;
434     float  tot_xfer;
435     int   *line_cnt;
436     int   *page_cnt;
437     float  xfer1;
438     float  xfer2;
439     float  ave_blk;
440     float  limit;
441     {
442
```

```
443       if (hdr_type == 1)
444       {
445           *page_cnt = *page_cnt + 1;
446
447           (void) printf("\n\n");
448           *line_cnt = 2;
449
450           (void) printf("                          *** SCANS-III STATISTICS ***                Page
451   %3d\n\n", *page_cnt);
452           *line_cnt = *line_cnt + 2;
453
454           (void) printf("%s\n\n",caption);
455           *line_cnt = *line_cnt + 2;
456
457           if (ports2 > 0)
458           {
459               (void) printf("                   Baud Rate: %4d           Primary Ports: %3d\n",
460   baud1, ports1);
461               *line_cnt = *line_cnt + 1;
462
463               (void) printf("                   Default Transfer Rate: %9.1f Blocks/Hour\n\n",
464   xfer1);
465               *line_cnt = *line_cnt + 2;
466
467               (void) printf("                   Baud Rate: %4d         Secondary Ports: %3d\n",
468   baud2, ports2);
469               *line_cnt = *line_cnt + 1;
470
471               (void) printf("                   Default Transfer Rate: %9.1f Blocks/Hour\n\n",
472   xfer2);
473               *line_cnt = *line_cnt + 2;
474           }
475           else
476           {
477               (void) printf("                   Baud Rate: %4d                  Ports: %3d\n",
478   baud, ports);
479               *line_cnt = *line_cnt + 1;
480
```

```
481             (void) printf("                  Default Transfer Rate: %9.1f Blocks/Hour\n\n",
482  xfer1);
483             *line_cnt = *line_cnt + 2;
484         }
485
486         (void) printf("                  Average Block Size:    %9.1f Blocks\n", ave_blk);
487         *line_cnt = *line_cnt + 1;
488
489         if (limit > 0.0)
490         {
491             (void) printf("                  SU Size Limitation:    %9.1f Blocks\n", ave_blk);
492             *line_cnt = *line_cnt + 1;
493         }
494
495         (void) printf("                  Average Transfer Time: %9.1f Blocks/Hour\n\n",
496  tot_xfer);
497         *line_cnt = *line_cnt + 2;
498
499         (void) printf("                  Elapsed Time:       % 6ld Hours % 2ld Minutes\n\n",
500                 pgm_hrs, pgm_min);
501         *line_cnt = *line_cnt + 2;
502
503         (void) printf("                      %s - %02ld/%02ld/%04ld %02d:%02d\n\n",
504                pgm_start, mon_cnt, day_cnt, year_cnt, hour_cnt, min_cnt);
505         *line_cnt = *line_cnt + 2;
506
507         if (det_prt == 1)
508         {
509             (void) printf("                    Total        Requests     Percentage of\n");
510             *line_cnt = *line_cnt + 1;
511
512             (void) printf("                  Queue Time      Queued          Total\n");
513             *line_cnt = *line_cnt + 1;
514
515             (void) printf("                  ==========    ==========    =============\n");
516             *line_cnt = *line_cnt + 1;
517         }
518
519
```

```
        return;
    }   /* end hdr_type == 1 */

    if (hdr_type == 2)
    {
        *page_cnt = *page_cnt + 1;

        (void) printf("\n\n");
        *line_cnt = 2;

        (void) printf("                          *** SCANS-III STATISTICS ***                Page
%3d\n\n", *page_cnt);
        *line_cnt = *line_cnt + 2;

        (void) printf("                              TOTAL DELIVERY TIME \n\n");
        *line_cnt = *line_cnt + 2;

        if (ports2 > 0)
        {
            (void) printf("                  Baud Rate: %4d           Primary Ports: %3d\n",
baud1, ports1);
            *line_cnt = *line_cnt + 1;

            (void) printf("                  Default Transfer Rate: %9.1f Blocks/Hour\n\n",
xfer1);
            *line_cnt = *line_cnt + 2;

            (void) printf("                  Baud Rate: %4d         Secondary Ports: %3d\n",
baud2, ports2);
            *line_cnt = *line_cnt + 1;

            (void) printf("                  Default Transfer Rate: %9.1f Blocks/Hour\n\n",
xfer2);
            *line_cnt = *line_cnt + 2;

        }
        else
        {
```

```
            (void) printf("                  Baud Rate: %4d                  Ports: %3d\n",
baud, ports);
            *line_cnt = *line_cnt + 1;

            (void) printf("                  Default Transfer Rate: %9.1f Blocks/Hour\n\n",
xfer1);
            *line_cnt = *line_cnt + 2;
        }

        (void) printf("                  Average Block Size:    %9.1f Blocks\n", ave_blk);
        *line_cnt = *line_cnt + 1;

        if (limit > 0.0)
        {
            (void) printf("                  SU Size Limitation:    %9.1f Blocks\n", ave_blk);
            *line_cnt = *line_cnt + 1;
        }

        (void) printf("                  Average Transfer Time: %9.1f Blocks/Hour\n\n",
tot_xfer);
        *line_cnt = *line_cnt + 2;

        (void) printf("                  Elapsed Time:       % 6ld Hours % 2ld Minutes\n\n",
                pgm_hrs, pgm_min);
        *line_cnt = *line_cnt + 2;

        (void) printf("                      %s - %02ld/%02ld/%04ld %02d:%02d\n\n",
               pgm_start, mon_cnt, day_cnt, year_cnt, hour_cnt, min_cnt);
        *line_cnt = *line_cnt + 2;

        if (det_prt == 1)
        {

            (void) printf("    Total          Average         Average      Requests
Percentage of\n");

            *line_cnt = *line_cnt + 1;

```

```
596                 (void) printf("  Request Time    Active Time     Queue Time     Completed
597   Total\n");
598
599                 *line_cnt = *line_cnt + 1;
600
601                 (void) printf("  ============    ===========    ============    =========
602   =============\n");
603
604                 *line_cnt = *line_cnt + 1;
605
606           }
607
608           return;
609        }   /* end hdr_type == 2 */
610
611   }
612
613   /* EOF - prt_page */
```