

https://dspace.org/

Design of a disk-based file server involving the mapping of "pseudo-disks"

Hildyard, Mark William

2015-01-27
http://hdl.handle.net/10539/16783

Downloaded from a DSpace repository. https://dspace.org/

B.3.3 Common Software Information - *File*: COMINFO.RSC

```
/* ------------------------------------------------------------------------------------------ */

/* RESOURCE:  COMMON$INFO */
    /* File mapping Application Numbers to a directory name, plus access &
       other information */
  /*CONST*/
    Declare
      Non$Appl              literally   '0',
      Non$Disk              literally   'Non$Appl+1',
      Flat                  literally   'False',
      Immediate             literally   'True',
      Read                  literally   'False',
      Write                 literally   'True',
      Access$Sz             literally   '4',
      User$Appl             literally   '8000h',
      OS$Max                literally   '20',
      Any$OS                literally   'OS$Max',
      Size$OS$Entry         literally   'SIZE (OS$Array(0))',          /* 7 */
      Size$Comm$Entry       literally   'SIZE (Current$Common$Entry)', /* 31 */
      Group$Name$Size       literally   '9',
      Appl$Name$Size        literally   '9';

  /*TYPES*/
    Declare
      Type$File$Num         literally   'Word',
      OS$Entry$Type         literally   'STRUCTURE ( Appl$Num     Type$File$Num,
                                                    OS$Name (5)  Byte)',
      Access$Bytes          literally   '(Access$Sz)       Byte',
      Group$Name$Type       literally   '(Group$Name$Size)  Byte',
      Appl$Name$Type        literally   '(Appl$Name$Size)   Byte';

  /*VAR*/
    Declare
      Common$Info$Region    Region,
      OS$Array (OS$Max)     OS$Entry$Type,
      Large$OS              Byte,          /* Allows Max 256 Op Systs */
      Def$OS                Byte,          /* An Op Syst Num assigned during configuration */
      Access$Appl           Type$File$Num, /* File Num for the appl containing disks with access &
                                              availability info for users, read during configuration */
      Common$Info$File      Token,
      Large$Common          Word,
      First$Comm$Entry      Word,
      Current$Common$Entry  STRUCTURE
                              ( OS$Num              Byte,
                                Group$Name          Group$Name$Type,
                                Appl$Name           Appl$Name$Type,
                                Assigned            Boolean,
                                Appl$Connection     Token,
                                Qty                 Byte,
                                Max$Users           Byte,
                                Access              Access$Bytes,
                                All$Reqd            Boolean,
                                Load$Type           Boolean,
                                Track$Group$Size    Byte );


  /*PROCEDURE */ Common$Info_Init:Procedure;
```

```
    /*CONST*/
       Declare
         File            literally   '(11, ''COMMON$INFO'')',
         Qty$Buffers     literally   '2';
    /*VAR*/
       Declare
         Exception       IO$Exception,
         Bytes$Read      Word;

    /*BEGIN*/
       Common$Info$File : RQ$S$ATTACH$FILE (@File, @Exception);
       CALL RQ$S$OPEN (Common$Info$File, Reading, Qty$Buffers, @Exception);
       Bytes$Read : RQ$S$READ$MOVE (Common$Info$File, @Large$OS, 1, @Exception);
       Bytes$Read : RQ$S$READ$MOVE (Common$Info$File, @Large$Common, 2,  @Exception);
       Bytes$Read : RQ$S$READ$MOVE (Common$Info$File, @First$Com$Entry, 2, @Exception);
       Bytes$Read : RQ$S$READ$MOVE (Common$Info$File,@OS$Array,((Large$OS+1)*Size$OS$Entry),@Exception);
     END Common$Info_Init;


/*PROCEDURE */ Common$Info_Read: Procedure (Appl$Num, Status$Ptr);
                                 Declare Appl$Num    Type$File$Num,
                                         Status$Ptr  Pointer;

    /*CONST*/
       Declare
         Set             literally  '2',
         Position        literally  'First$Com$Entry + (Appl$Num-1) * Size$Com$Entry';
    /*VAR*/
       Declare
         Exception       IO$Exception,
         Bytes$Read      Word,
         Status          BASED Status$Ptr    Status$Type;

    /*BEGIN*/
       IF (Appl$Num > Max$Appl)
         THEN Status : Appl$Not$Exist;
         ELSE
           DO;
             /* Shift out the insignificant bits for an application number*/
             Appl$Num : SHR (Appl$Num,4);
             IF (Appl$Num > Large$Common)
               THEN Status : Appl$Not$Exist;
               ELSE
                 DO;
                   CALL RQ$S$SEEK (Common$Info$File, Set, Position, @Exception);
                   Bytes$Read : RQ$S$READ$MOVE (Common$Info$File, @Current$Common$Entry,
                                                Size$Com$Entry, @Exception);
                   IF ( Current$Common$Entry.Assigned = True)
                     THEN Status : Successful;
                     ELSE Status : Appl$Not$Exist;
                 END; /* Else 1*/
           END; /* Else 2*/
     END Common$Info_Read;


/*PROCEDURE */ Common$Info_Attach: Procedure (Status$Ptr);
                                   Declare  Status$Ptr  Pointer;
    /*CONST*/
       Declare
         Strings       literally  '3';   /* Number of strings sent */
```

```
/*VAR*/
   Declare
     Status       BASED Status$Ptr   Status$Type,
     Exception    IO$Exception,
     OS$Num       Byte              AT (@Current$Common$Entry.OS$Num),
     Group        Group$Name$Type   AT (@Current$Common$Entry.Group$Name),
     Appl         Appl$Name$Type    AT (@Current$Common$Entry.Appl$Name),
     Connection   Token             AT (@Current$Common$Entry.Appl$Connection);

/*BEGIN*/
   CALL HD$Attach_Att$Create$Dir ( @OS$Array(OS$Num).OS$Name, @Group, @Appl,
                 Strings,Common$Prefix, @Connection, @Exception);
   IF (Exception = RS$OK)
     THEN Status = Successful;
     ELSE Status = Disk$Attach$Error;
  END Common$Info_Attach;


/* END  COMMON$INFO RESOURCE */

/* ------------------------------------------------------------------------------------------ */
```

B.3.4 User Information - *File*: UINFO.RSC

```
/* ------------------------------------------------------------------------------------------ */

/* RESOURCE: USER$INFO */
    /*File mapping User numbers to Passwords etc */

  /*CONST*/
     Declare
       Non$User             literally       '0',
       Carte$Blanche        literally       '1',
       No$Access            literally       'OFFh',
       Pass$Size            literally       '7',
       Size$U$Entry         literally       'SIZE (Current$User$Entry)';  /* 11 */

  /*TYPE*/
     Declare
       Type$User$Num        literally       'Word',
       Pass$Type            literally       '(Pass$Size) Byte';  /* 6 Char String */

  /*VAR*/
     Declare
       User$Info$Region     Region,
       User$Info$File       Token,
       Large$User           Type$User$Num,
       First$User$Entry     Byte,
       Current$User$Entry   STRUCTURE
                              ( Password        Pass$Type,
                                Assigned        Boolean,
                                Access$Config   Byte,
                                Max$Qty         Byte,
                                Init$OS         Byte );


  /*PROCEDURE */ User$Info_Init: Procedure;
     /*CONST*/
        Declare
          File              literally       '(9, ''USER$INFO'')',
          Qty$Buffers       literally       '2';
     /*VAR*/
        Declare
          Exception         IO$Exception,
          Bytes$Read        Word;

     /*BEGIN*/
        User$Info$File : RQ$S$ATTACH$FILE (@File, @Exception);
        CALL RQ$S$OPEN (User$Info$File, Reading, Qty$Buffers, @Exception);
        Bytes$Read : RQ$S$READ$MOVE (User$Info$File, @Large$User, 2, @Exception);
        Bytes$Read : RQ$S$READ$MOVE (User$Info$File, @First$User$Entry, 1, @Exception);
        /* These Extended I/O System calls should all be successful. Possible errors
            are hard I/O & & insufficient memory */
      END User$Info_Init;


  /*PROCEDURE*/ User$Info_Read.   ocedure (User$Num, Password$Ptr, Status$Ptr);
                                Declare  User$Num      Type$User$Num,
                                         Password$Ptr  Pointer,
                                         Status$Ptr    Pointer;
     /*CONST*/
```

```
        Declare
          Set             literally  '2',
          Position        literally  '(First$User$Entry + (User$Num-1) * Size$U$Entry)';
      /*VAR*/
        Declare
          Exception       IO$Exception,
          Bytes$Read      Word,
          Password        BASED Password$Ptr   Pass$Type,
          Status          BASED Status$Ptr     Status$Type;

      /*BEGIN*/
        Status = User$Denied;
        IF ( (User$Num <= Large$User) AND (User$Num <> Non$User) )
        THEN
          DO;
            CALL RQ$S$SEEK (User$Info$File, Set, Position, @Exception);
            Bytes$Read = RQ$S$READ$MOVE (User$Info$File, @Current$User$Entry, Size$U$Entry, @Exception);
            IF (Current$User$Entry.Assigned = True)
              THEN IF ( String_Compare (Password$Ptr, @Current$User$Entry.Password) )
                     THEN Status = Successful;
          END;
      END User$Info_Read;

/* END  USER$INFO RESOURCE */

/* ------------------------------------------------------------------------------------------------ */
```

B.3.5 Opened Disks Tables - *File*: OP$TABLE.RSC

```
/* ------------------------------------------------------------------------------------------ */

/* RESOURCE:  OP$TABLE */
     /* Table of open disks & application packages */


  /*CONST*/
     Declare
       Max$Op$Appls          literally   '100',
       Max$Op$Disks          literally   '255',
       Non$Op$Appl           literally   'Max$Op$Appls',
       Non$Op$Disk           literally   'Max$Op$Disks',
       User$Appl$Num         literally   '0';

  /*TYPES*/
     Declare
       Type$Op$Appl$Num      literally   'Byte',
       Type$Op$Disk$Num      literally   'Byte',
       Atype                 literally   'Type$Op$Appl$Num',
       Dtype                 literally   'Type$Op$Disk$Num',
       Ptype                 literally   'Type$File$Num';

  /*STRUCTURED TYPES*/
     Declare
      /* Op$Appl$Entry
            : STRUCTURE
               ( Appl                Ptype,
                 Appl$Connection     Token,
                 OS$Num              Byte,
                 Qty                 Byte,
                 Max$Users           Byte,
                 Access              Access$Bytes,
                 All$Reqd            Boolean,
                 Load$Type           Boolean,
                 Track$Group$Size    Byte,
                 Qty$Open            Byte )     */

        Op$Appl$Entry literally
          'STRUCTURE (Appl Ptype, Appl$Connection Token, OS$Num Byte, Qty Byte, Max$Users Byte,
     Access Access$Bytes, All$Reqd Boolean, Load$Type Boolean, Track$Group$Size Byte, Qty$Open Byte ) ';

     Declare
       /* Op$Disk$Entry
            : STRUCTURE
               ( Disk                Ptype,
                 Op$Appl$Num         Atype,
                 Open$Count          Byte,
                 Read$Write          Byte,
                 Open$Connection     Token )   */

        Op$Disk$Entry literally
          'STRUCTURE ( Disk Ptype, Op$Appl$Num Atype, Open$Count Byte,
                       Read$Write Byte, Open$Connection Token ) ';

  /*VARS*/
     Declare
       Op$Table$Region             Region,
```

```
    First$Free$Appl              Atype,
    First$Free$Disk              Dtype,
    Large$Op$Appl                Atype,
    Large$Op$Disk                Dtype,
    Tot$Free$Disks               Dtype,
    Op$Appl$Map (Max$Op$Appls)   Op$Appl$Entry,
    Op$Disk$Map (Max$Op$Disks)   Op$Disk$Entry;


/*PROCEDURE */ Zero$Appl$Entry: Procedure (Op$Appl$Num);
                                Declare Op$Appl$Num Atype;
  /*VAR*/
     Declare
       Reqd$Entry$Ptr  Pointer,
       Reqd$Entry      BASED  Reqd$Entry$Ptr   Op$Appl$Entry;

  /*BEGIN*/
     /* WITH Op$Appl$Map(Op$Appl$Num) DO */
       Reqd$Entry$Ptr : @Op$Appl$Map(Op$Appl$Num);
         Reqd$Entry.Appl              : Non$Appl;
         Reqd$Entry.Appl$Connection   : Non$Token;
         Reqd$Entry.OS$Num            : Any$OS;
         Reqd$Entry.Qty               : 0;
         Reqd$Entry.Max$Users         : 0;
         CALL SETB (0, @Reqd$Entry.Access, Access$Sz);
         Reqd$Entry.All$Reqd          : False;
         Reqd$Entry.Loads$Type        : Flat;
         Reqd$Entry.Track$Group$Size  : 0;
         Reqd$Entry.Qty$Open          : 0;
   END Zero$Appl$Entry;


/*PROCEDURE */ Zero$Disk$Entry: Procedure (Op$Disk$Num);
                                Declare Op$Disk$Num Dtype;
  /*VAR*/
     Declare
       Reqd$Entry$Ptr  Pointer,
       Reqd$Entry      BASED  Reqd$Entry$Ptr  Op$Disk$Entry;

  /*BEGIN*/
     /* WITH Op$Appl$Map(Op$Appl$Num) DO */
       Reqd$Entry$Ptr : @Op$Disk$Map(Op$Disk$Num);
         Reqd$Entry.Disk          : Non$Disk;
         Reqd$Entry.Op$Appl$Num   : Non$Op$Appl;
         Reqd$Entry.Open$Count    : 0;
         Reqd$Entry.Read$Write    : Read;
         Reqd$Entry.Open$Connection  : Non$Token;
   END Zero$Disk$Entry;


/*PROCEDURE */ Op$Table_Init: Procedure;
  /*VAR*/
     Declare
       User$Entry  Op$Appl$Entry   AT (@Op$Appl$Map(User$Appl$Num)),
       A$Count     Atype,
       D$Count     Dtype;
  /*BEGIN*/
     /* Initialize first Appl Entry for User disk information */
       User$Entry.Appl            : User$Appl; /* Bit 15 must be set */
```

```
            User$Entry.Appl$Connection  : Non$Token;
            User$Entry.OS$Num           : Any$OS;
            User$Entry.Qty              : 0;
            User$Entry.Max$Users        : 1;
            CALL SETW (0, @User$Entry.Access, Access$Sz);
            User$Entry.All$Reqd         : False;
            User$Entry.Load$Type        : Flat;
            User$Entry.Track$Group$Size : 2;
            User$Entry.Qty$Open         : 1;

          /* Initialize rest of Application entries */
            DO A$Count = (User$Appl$Num + 1) TO (Max$Op$Appls - 1);
              CALL Zero$Appl$Entry(A$Count);
            END;
            First$Free$Appl = User$Appl$Num + 1;
            Large$Op$Appl = User$Appl$Num;

          /* Initialize Entries in Open Disk Map */
            DO D$Count = 0 TO (Max$Op$Disks - 1);
              CALL Zero$Disk$Entry(D$Count);
            END;
            First$Free$Disk = 0;
            Large$Op$Disk = 0;
            Tot$Free$Disks = Max$Op$Disks;
        END Op$Table_Init;


/*PROCEDURE */ Op$Table_Search$Appl: Procedure ( Reqd$Appl, Op$Appl$Num$Ptr, Status$Ptr );
                                     Declare  Reqd$Appl       Ptype,
                                              Op$Appl$Num$Ptr Pointer,
                                              Status$Ptr      Pointer;
    /*VAR*/
        Declare
          Op$Appl$Num      BASED  Op$Appl$Num$Ptr  Atype,
          Status           BASED  Status$Ptr       Status$Type;
        Declare
          Count            Atype;

    /*BEGIN*/
        Count = User$Appl$Num + 1;
        Status = NOT (Found);
        DO WHILE ( (Count <= Large$Op$Appl) AND (Status <> Found) );
          IF ( Op$Appl$Map(Count).Appl = Reqd$Appl )
            THEN DO;
                   Op$Appl$Num = Count;
                   Status = Found;
                 END;
          Count = Count + 1;
        END; /* While */
      END Op$Table_Search$Appl;


/*PROCEDURE */ Op$Table_Attach$Appl: Procedure (Appl$Num, Op$Appl$Num$Ptr, Status$Ptr);
                                     Declare  Appl$Num        Ptype,
                                              Op$Appl$Num$Ptr Pointer,
                                              Status$Ptr      Pointer;
    /*CONST*/
        Declare
          New$Appl      literally      'Current$Common$Entry';
    /*VAR*/
```

```
        Declare
          Op$Appl$Num   BASED Op$Appl$NumsPtr  Atype,
          Status        BASED Status$Ptr       Status$Type,
          Done          Boolean;

      /*BEGIN*/
        IF (First$Free$Appl = Non$Op$Appl)
          THEN Status = Too$Many$Appls;
          ELSE
            DO;
              Status = Successful;
              Op$Appl$Num = First$Free$Appl;
              /* Assign each member of the new Application */
              /* WITH Op$Appl$Map(Op$Appl$Num) DO */
                DO;
                  Declare
                    Reqd$Entry$Ptr Pointer,
                    Reqd$Entry     BASED Reqd$Entry$Ptr  Op$Appl$Entry;
                  Reqd$Entry$Ptr = @Op$Appl$Map(Op$Appl$Num);
                    Reqd$Entry.Appl             = Appl$Num;
                    Reqd$Entry.Appl$Connection  = New$Appl.Appl$Connection;
                    Reqd$Entry.OS$Num           = New$Appl.OS$Num;
                    Reqd$Entry.Qty              = New$Appl.Qty;
                    Reqd$Entry.Max$Users        = New$Appl.Max$Users;
                    CALL MOVB (@New$Appl.Access, @Reqd$Entry.Access, Access$Sz);
                    Reqd$Entry.All$Reqd         = New$Appl.All$Reqd;
                    Reqd$Entry.Load$Type        = New$Appl.Load$Type;
                    Reqd$Entry.Track$Group$Size = New$Appl.Track$Group$Size;
                END; /* With */

              /* Find the next free Open Application Entry */
                Done = False;
                DO WHILE ( (First$Free$Appl < Max$Op$Appls) AND (NOT Done) );
                  IF ( Op$Appl$Map(First$Free$Appl).Appl <> Non$Appl )
                    THEN First$Free$Appl = First$Free$Appl + 1;
                    ELSE Done = True;
                END;

              /* See if this is now the largest used entry */
                IF (Op$Appl$Num > Large$Op$Appl)
                  THEN Large$Op$Appl = Op$Appl$Num;

            END  /* Else Do */
      END Op$Table_Attach$Appl;


    /*PROCEDURE */ Op$Table_Remove$Appl: Procedure (Op$Appl$Num);
                                   Declare Op$Appl$Num Atype;
      /*VARS*/
        Declare
          Exception    IO$Exception;

      /*BEGIN*/
        /* Delete file connection & re-zero the application entry */
        CALL RQ$A$DELETE$CONNECTION (Op$Appl$Map(Op$Appl$Num).Appl$Connection, No$Mailbox, @Exception);
        CALL Zero$Appl$Entry (Op$Appl$Num);

        /* Update tracking variables */
        IF ( Op$Appl$Num < First$Free$Appl )
          THEN First$Free$Appl = Op$Appl$Num;
```

```
    IF ( Op$Appl$Num > Large$Op$Appl )
      THEN
        DO WHILE ( (Large$Op$Appl > 0) AND
                   (Op$Appl$Map(Large$Op$Appl).Appl = Non$Appl) );
          Large$Op$Appl = Large$Op$Appl - 1;
        END;
  END Op$Table_Remove$Appl;



/*PROCEDURE */ Op$Table_Search$Disk: Procedure (Reqd$Disk, Op$Disk$NumsPtr, Status$Ptr );
                                  Declare Reqd$Disk        Ptype,
                                          Op$Disk$NumsPtr  Pointer,
                                          Status$Ptr       Pointer;
  /*VAR*/
    Declare
      Op$Disk$Num   BASED Op$Disk$NumsPtr  Dtype,
      Status        BASED Status$Ptr       Status$Type;
    Declare
      Count         Dtype;

  /*BEGIN*/
    Count = 0;
    Status = NOT (Found);
    DO WHILE ( (Count <= Large$Op$Disk) AND (Status <> Found) );
      IF ( Op$Disk$Map(Count).Disk = Reqd$Disk )
        THEN DO;
               Op$Disk$Num = Count;
               Status = Found;
             END;
      Count = Count + 1;
    END; /* While */
  END Op$Table_Search$Disk;


/*PROCEDURE */ Op$Table_Attach$Disks: Procedure
            (Rev$Disk, Qty, Op$Appl$Num, Op$Disk$NumsPtr, Status$Ptr);
              Declare Rev$Disk          Ptype,
                      Qty               Byte,
                      Op$Appl$Num       Atype,
                      Op$Disk$NumsPtr   Pointer,
                      Status$Ptr        Pointer;
  /*VAR*/
    Declare
      Op$Disk$Num       BASED Op$Disk$NumsPtr  Dtype,
      Status            BASED Status$Ptr       Status$Type;
    Declare
      Count             Dtype,
      Free$Count        Byte,
      Done              Boolean;

  /*BEGIN*/
    IF (Tot$Free$Disks < Qty)
      THEN Status = Too$Many$Disks;
      ELSE
        DO;
          /* Search for free disk entries */
          Status = No$Disk$Space;
          Count = First$Free$Disk;
```

```
            Free$Count : 0;
            DO WHILE ((Count < Max$Op$Disks) AND (Status <> Successful));
              IF ( Op$Disk$Map(Count).Disk <> Non$Disk )
                THEN Free$Count : 0;
                ELSE
                  DO;
                    Free$Count : Free$Count + 1;
                    IF (Free$Count = Qty)
                      THEN Status : Successful;
                  END;
              Count : Count + 1;
            END; /* While */

            IF (Status : Successful)
              THEN
                DO;
                  /* Update the number of free disk entries & the largest entry used */
                    Tot$Free$Disks : Tot$Free$Disks - Qty;
                    IF ( (Count - 1) > Large$Op$Disk)
                      THEN Large$Op$Disk : (Count - 1);

                  /* Put the disk number & Appl entry in each entry */
                  Op$Disk$Num : Count - Qty;
                  DO Count : 0 TO (Qty - 1);
                    /* WITH Op$Disk$Map(Op$Disk$Num+Count) DO */
                    DO;
                      Declare
                        Current$Entry$Ptr  Pointer,
                        Current$Entry      BASED  Current$Entry$Ptr    Op$Disk$Entry;
                      Current$Entry$Ptr : @Op$Disk$Map(Op$Disk$Num+Count);
                        Current$Entry.Disk : (New$Disk + Count);
                        Current$Entry.Op$Appl$Num : Op$Appl$Num;
                    END; /* With */
                  END; /* Do Loop */

                  /* Find the first free Open Disk Entry */
                    Done : False;
                    DO WHILE ( (First$Free$Disk < Max$Op$Disks) AND (NOT Done) );
                      IF ( Op$Disk$Map(First$Free$Disk).Disk <> Non$Disk )
                        THEN First$Free$Disk : First$Free$Disk + 1;
                        ELSE Done : True;
                    END;

                END; /* If Then */
            END; /* Else Do */
        END Op$Table_Attach$Disks;


/*PROCEDURE */ Op$Table_Remove$Disks: Procedure (Op$Disk$Num, Qty);
                                    Declare Op$Disk$Num Dtype,
                                            Qty         Byte;
    /*VAR*/
        Declare
          Exception    IO$Exception,
          Count        Byte;

    /*BEGIN*/
        /* Delete file connections & re-zero the disk entries */
        DO Count : Op$Disk$Num TO (Op$Disk$Num + Qty - 1);
          /* The cache ought to have been flushed for this disk, before the disk is removed */
```

```
            CALL RQ$A$DELETE$CONNECTION (Op$Disk$Map(Count).Open$Connection, Nos$Mailbox, @Exception);
            CALL Zero$Disk$Entry (Count);
          END;

          /* Update tracking variables */
          Tot$Free$Disks = Tot$Free$Disks + Qty;
          IF ( Op$Disk$Num < First$Free$Disk )
            THEN First$Free$Disk = Op$Disk$Num;
          IF ( Large$Op$Disk = (Op$Disk$Num + Qty - 1) )
            THEN
              DO;
                Large$Op$Disk = Op$Disk$Num;
                DO WHILE ( (Large$Op$Disk > 0) AND (Op$Disk$Map(Large$Op$Disk).Disk = Non$Disk) );
                  Large$Op$Disk = Large$Op$Disk - 1;
                END;
              END;
        END Op$Table_Remove$Disks;


/*PROCEDURE */ Op$Table_Mark$Open: Procedure (Op$Disk$Num, Qty, Reads$Write, Status$Ptr );
                                   Declare Op$Disk$Num  Dtype,
                                           Qty          Byte,
                                           Reads$Write  Boolean,
                                           Status$Ptr   Pointer;
    /*VAR*/
        Declare
          Status   BASED Status$Ptr  Status$Type;
        Declare
          Op$Appl$Num Atype,
          Count       Dtype;

    /*BEGIN*/
        Status = Successful;
        Op$Appl$Num = Op$Disk$Map(Op$Disk$Num).Op$Appl$Num;

        /* Scan first to ensure each Opening is legitimate */
        Count = Op$Disk$Num;
        DO WHILE ((Status = Successful) AND (Count < (Op$Disk$Num + Qty)) );
          IF (Op$Disk$Map(Count).Open$Count > 0)
            THEN
              IF (Reads$Write = Write)
                THEN Status = Open$As$Read;
                ELSE
                  IF (Op$Disk$Map(Count).Reads$Write = Write)
                    THEN Status = Open$As$Write;
                    ELSE
                      IF ( Op$Disk$Map(Count).Open$Count >=
                           Op$Appl$Map(Op$Appl$Num).Max$Users )
                        THEN Status = Exceed$Opens;
          Count = Count + 1;
        END; /* While */

        /* If legitimate, mark open as requested */
        IF (Status = Successful)
          THEN
            DO Count = Op$Disk$Num TO (Op$Disk$Num + Qty - 1);
              IF (Op$Disk$M.  (Count).Open$Count = 0)
                THEN
                  DO;
                    Op$Disk$Map(Count).Reads$Write = Reads$Write;
```

```
                    Op$Appl$Map(Op$Appl$Num).Qty$Open : Op$Appl$Map(Op$Appl$Num).Qty$Open + 1;
                  END;

              Op$Disk$Map(Count).Open$Count : Op$Disk$Map(Count).Open$Count + 1;
            END; /* Do Loop */
      END Op$Table_Mark$Open;


/*FUNCTION */ Op$Table_Qty$Open: Procedure (Op$Disk$Num)   Byte;
                                 Declare Op$Disk$Num Dtype;
    /*BEGIN*/
       RETURN ( Op$Disk$Map(Op$Disk$Num).Open$Count );
     END Op$Table_Qty$Open;


/*PROCEDURE */ Op$Table_Mark$Closed: Procedure (Op$Disk$Num);
                                     Declare Op$Disk$Num Dtype;
    /*CONST*/
       Declare
         Num$Bits         literally  '1111B';
    /*VAR*/
       Declare
         Op$Appl$Num      Atype,
         D$Num            Byte,
         First$Disk       Dtype;

    /*BEGIN*/
       Op$Appl$Num : Op$Disk$Map(Op$Disk$Num).Op$Appl$Num;
         Declare
           Reqd$Disk     literally   'Op$Disk$Map(Op$Disk$Num)',
           Curr$Appl     literally   'Op$Appl$Map(Op$Appl$Num)';
             /* Reqd$Disk is the disk to be removed, while Curr$Appl is the application
                entry for the application to which this disk belongs */

       Reqd$Disk.Open$Count : Reqd$Disk.Open$Count - 1;
       IF (Reqd$Disk.Open$Count : 0)
         THEN
           IF ( NOT (Curr$Appl.All$Reqd) )
             THEN
               DO;
                 CALL Op$Table_Remove$Disks (Op$Disk$Num, 1);
                 Curr$Appl.Qty$Open : Curr$Appl.Qty$Open - 1;
                 IF (Curr$Appl.Qty$Open : 0)
                   THEN CALL Op$Table_Remove$Appl (Op$Appl$Num);
               END;
             ELSE
               DO;
                 Curr$Appl.Qty$Open : Curr$Appl.Qty$Open - 1;
                 IF (Curr$Appl.Qty$Open : 0)
                   THEN
                     DO;
                       /* Find first disk of the application */
                         D$Num : LOW (Reqd$Disk.Disk AND Num$Bits);
                         First$Disk : Op$Disk$Num - D$Num + 1;
                       /* Remove each disk of the application */
                         CALL Op$Table_Remove$Disks (First$Disk, Curr$Appl.Qty);
                       CALL Op$Table_Remove$Appl (Op$Appl$Num);

                     END; /* Then */
               END; /* Else */
```

```
    END Op$Table_Mark$Closed;


/*PROCEDURE */ Op$Table_Disks$Ready: Procedure (Op$Disk$Num, Qty, Status$Ptr);
                                     Declare Op$Disk$Num   Dtype,
                                             Qty           Byte,
                                             Status$Ptr    Pointer;
  /*VARS*/
    Declare
      Status   BASED Status$Ptr   Status$Type;
    Declare
      Count    Dtype;
  /*BEGIN*/
    Status : Ready;
    Count : Op$Disk$Num;
    DO WHILE ( (Status : Ready) AND (Count < (Op$Disk$Num + Qty)) );
      IF (Op$Disk$Map(Count).Open$Connection : Nons$Token)
        THEN Status : NOT(Ready);
        ELSE Count : Count + 1;
    END;
  END Op$Table_Disks$Ready;


/* END OP$TABLE RESOURCE */

/* ------------------------------------------------------------------------------------------ */
```

B.3.6 Cache Resource - *File*: CACHE.RSC

```
/* ------------------------------------------------------------------------------ */

/* RESOURCE:  PC Track Cache */

   /* The cache maintains the tracks of open 'disks' (files at the File Server). The actual cache
      entry is not a single track, but a group of tracks. The cache consists of a list of track-
      groups (TGL), for each possible open disk. Each track-group entry (TG) maintains a list (
      Track$List : TL ) of track-entries (T$Ent), which contain each track's location in memory.
      There are actually 2 tracks in each track-entry, since 2 PC-Tracks are required for a whole
      number of File-Server sectors. The number of tracks in a track-group depends on the disk,
      but is a multiple [illegible] Track-groups are also linked into 2 other lists: WL : Write$List,
      RL : Replaceme[illegible]nal disk-writes are performed in the background. */


/*CONST*/
   Declare
     Max$Tracks          literally   '20',
     Max$Groups          literally   '10',
     Num$Track           literally   'Max$Tracks',
     Num$Group           literally   'Max$Groups',
     Size$2Tracks        literally   '2400h';       /* 2 PC-Tracks (16 * 512-byte sectors) are
                                                        held in a track-entry */


/*TYPES*/
   Declare
     /* Dtype             literally   'Type$Op$Disk$Num'       Already declared */
     Type$Track$Num      literally   'Byte',              /* Depends on Cache size - Max$Tracks */
     Type$Track$Group$Num literally  'Byte',              /* Depends on Max$Groups */
     Track$Num           literally   'Type$Track$Num',
     Group$Num           literally   'Type$Track$Group$Num';

/*STRUCTURED TYPES*/
   /* SLR is for a Space List Rock, or first entry of a space list, LLR is for a
        Link List Rock, fp is for a forward pointer & bp for a backward pointer */

   Declare
     Track$Entry  literally
      'STRUCTURE  ( TL$fp Track$Num, Track$Pos Pointer, Present Boolean, Sect$Bits (3)  Byte )';


   /* Track$Group$Entry:  Description
        STRUCTURE
          ( Identity
              STRUCTURE
                ( Track$A       Type$PC_Track,          ( first & last PC-Tracks, which
                  Track$B       Type$PC_Track,                the group holds )
                  Disk          Dtype,            ( The disk to which the tracks belong )
                  Qty$Ops       Word,             ( The Number of operations still to be performed on the group )
                  Dirty         Boolean ),        ( The track-group has been written-to )
            Track$Group$List
              STRUCTURE
                ( fp            Group$Num),       ( Pointer to next group entry in the disk's TGL )
            Track$List
              STRUCTURE
                ( TL$LLR        Track$Num),       ( LLR or 1st track-entry in the group's TL )
            Replace$List
```

```
        STRUCTURE
          ( fp          Group$Num,       { Forward & backward pointers to track group
            bp          Group$Num ),                  entries in the Replacement List }
      Write$List
        STRUCTURE
          ( fp          Group$Num,       { Forward & backward pointers to track-group
            bp          Group$Num,                         entries in the Write List }
            Must$Write  Boolean,         { The disk will be updated, even if the
                                                   minimum time has not elapsed }
            Time$Stamp  Word )     ); */


Declare
  /* The declaration for a track-group entry type has been split into 3 parts,
      since otherwise the string is too long */

    Identity          literally   'Track$A   Type$PC_Track,      Track$B   Type$PC_Track,
                                  Disk       Dtype,              Qty$Ops   Word,
                                  Dirty      Boolean

    TGL$Info          literally   'TGL$fp    Group$Num',

    Track$List$Info   literally   'TL$LLR    Track$Num',

    Replace$List$Info literally   'RL$fp     Group$Num, RL$bp     Group$Num ',

    Write$List$Info   literally   'WL$fp     Group$Num, WL$bp     Group$Num,
                                  Must$Write Boolean,   Time$Stamp Word ',


  Track$Group$Entry   literally
    'STRUCTURE ( Identity, TGL$Info, Track$List$Info, Replace$List$Info, Write$List$Info ) ';


Declare
  Read$Msg$Structure  literally
    'STRUCTURE ( Qty Byte, Disk D$Type, Group Group$Num, Track$A Type$PC_Track, Track(1) Track$Num )';
                                                    /* Array Track is unbounded - cf: PLM-86 pg 8-2 */


/*VAR*/
  Declare
    TGL     (Max$Op$Disks)  STRUCTURE ( LLR Group$Num ),   /* Rocks for each open disk, marking the beginning of
                                                              the list of it's track-groups present in the cache */
    TG      (Max$Groups)    Track$Group$Entry,
    T$Ent   (Max$Tracks)    Track$Entry,
    TG$SLR                  Group$Num,          /* Space list rock for free track group entries */
    Free$Groups             Group$Num,
    T$Ent$SLR               Track$Num,          /* Space list rock for free track entries */
    Free$Tracks             Track$Num,          /* Total free track entries */

    WL$Front                Group$Num,          /* First & Last entries in Write$List */
    WL$Rear                 Group$Num,
    WL$Size                 Group$Num,
    RL$Front                Group$Num,          /* First & Last entries in Replacement$List */
    RL$Rear                 Group$Num,
    RL$Size                 Group$Num;

/*PROCEDURE */ Zero$Group$Entry: Procedure (Group);
                                Declare  Group  Group$Num;
```

```
    /*BEGIN*/
      /* Resets necessary variables in a track-group entry */
      TG(Group).Disk          : Non$Op$Disk;
      TG(Group).Qty$Ops       : 0;
      TG(Group).Dirty         : False;
      TG(Group).TL$LLR        : Non$Track;
      TG(Group).WL$fp         : Non$Group;
      TG(Group).WL$bp         : Non$Group;
      TG(Group).RL$fp         : Non$Group;
      TG(Group).RL$bp         : Non$Group;
      TG(Group).Must$Write    : False;
      TG(Group).Time$Stamp    : 0;
    END Zero$Groups$Entry;


/*PROCEDURE */ Zero$Track$$Entry: Procedure (Track);
                                  Declare  Track  Track$Num;
    /*BEGIN*/
      /* Resets necessary variables in a track entry */
      T$$Ent(Track).Present : False;
      CALL SETB (0, @T$$Ent(Track).Sect$Bits, 3);
    END Zero$Track$$Entry;


/*PROCEDURE */ WL$Insert$Rear: Procedure (Group);
                               Declare  Group  Group$Num;
    /*BEGIN*/
      /* Update the backward pointer of the previously rear entry */
      IF ( WL$Rear = Non$Group )  /* ie. If the list is empty */
        THEN WL$Front = Group;
        ELSE TG(WL$Rear).WL$bp = Group;
      TG(Group).WL$bp = Non$Group;
      TG(Group).WL$fp = WL$Rear;
      TG(Group).Time$Stamp = LOW (RQ$GET$TIME (@Exception) );
      WL$Rear = Group;
      WL$Size = WL$Size + 1;
    END WL$Insert$Rear;


/*PROCEDURE */ WL$Insert$Front: Procedure (Group);
                                Declare  Group  Group$Num;
    /*VAR*/
      Declare
        Pos             Group$Num,    /* Position before which the entry will be inserted */
        More$Must$Write  Boolean;

    /*BEGIN*/
      /* Find place in list, where to insert - behind those groups marked as 'must$write' */
      More$Must$Write = True;
      Pos = WL$Front;
      DO WHILE ( (Pos <> Non$Group) AND (More$Must$Write) );
        IF ( TG(Pos).Must$Write = False )
          THEN More$Must$Write = False;
          ELSE Pos = TG(Pos).WL$bp;
      END;

      /* Update the forward pointers, for the new entry & the
                  entry which it will be inserted ahead of */
      IF ( Pos = Non$Group )
        THEN  /* The position is at the rear of the list */
```

```
            DO;
              TG(Group).WL$fp = WL$Rear;
              WL$Rear = Group;
            END;
          ELSE
            DO;
              TG(Group).WL$fp = TG(Pos).WL$fp;
              TG(Pos).WL$fp = Group;
            END;

        /* Update the backward pointer for the entry which is ahead of the new entry */
        IF (WL$Front = Pos)
          THEN WL$Front = Group;  /* Position is at the front of the list */
          ELSE TG( TG(Group).WL$fp ).WL$bp = Group;

        TG(Group).WL$bp = Pos;
        TG(Group).Must$Write = True;
        WL$Size = WL$Size + 1;

      END WL$Insert$Front;


/*PROCEDURE */ WL$Remove: Procedure (Group);
                          Declare  Group  Group$Num;

      Declare
        Group$In$Writelist  literally
            '( (TG(Group).WL$bp <> Non$Group) OR (TG(Group).WL$fp <> Non$Group) )';

    /*BEGIN*/
      IF (Group$In$Writelist) THEN
        DO;
          /* Update the forward pointer of the previous track-group in the list */
          IF (WL$Rear = Group)
            THEN WL$Rear = TG(Group).WL$fp;    /* The entry to be removed is at the rear of the list */
            ELSE TG( TG(Group).WL$bp ).WL$fp = TG(Group).WL$fp;

          /* Update the backward pointer of the next track-group in the list */
          IF (WL$Front = Group)
            THEN WL$Front = TG(Group).WL$bp;    /* The entry to be removed is at the front of the list */
            ELSE TG( TG(Group).WL$fp ).WL$bp = TG(Group).WL$bp;

          TG(Group).WL$bp = Non$Group;
          TG(Group).WL$fp = Non$Group;
          TG(Group).Must$Write = False;
          WL$Size = WL$Size - 1;
        END;
    END WL$Remove;


/*PROCEDURE */ RL$Insert$Rear: Procedure (Group);
                               Declare  Group  Group$Num;
    /*BEGIN*/
      /* Update the backward pointer of the previously rear entry */
      IF ( RL$Rear = Non$Group )  /* ie. If the list is empty */
        THEN RL$Front = Group;
        ELSE TG(RL$Rear).RL$bp = Group;

      TG(Group).RL$bp = Non$Group;
      TG(Group).RL$fp = RL$Rear;
```

```
        RL$Rear : Group;
        RL$Size : RL$Size + 1;
      END RL$Insert$Rear;


/*PROCEDURE */ RL$Insert$Front: Procedure (Group);
                                Declare  Group  Group$Num;
    /*BEGIN*/
      /* Update the forward pointer of the previously front entry */
      IF ( RL$Front : Non$Group )  /* ie. If the list is empty */
        THEN RL$Rear : Group;
        ELSE TG(RL$Front).RL$fp : Group;

      TG(Group).RL$fp : Non$Group;
      TG(Group).RL$bp : RL$Front;
      RL$Front : Group;
      RL$Size : RL$Size + 1;
    END RL$Insert$Front;


/*PROCEDURE */ RL$Remove: Procedure (Group);
                          Declare  Group  Group$Num;

    Declare
      Group$In$Rep!$list  literally
          '( (TG(Group).RL$bp <> Non$Group) OR (TG(Group).RL$fp <> Non$Group) )';

    /*BEGIN*/
      IF Group$In$Rep!$list THEN
        DO;
          /* Update the forward pointer of the previous track-group in the list */
          IF (RL$Rear : Group)
            THEN RL$Rear : TG(Group).RL$fp;    /* The entry to be removed is at the rear of the list */
            ELSE TG( TG(Group).RL$bp ).RL$fp : TG(Group).RL$fp;

          /* Update the backward pointer of the next track-group in the list */
          IF (RL$Front : Group)
            THEN RL$Front : TG(Group).RL$bp;    /* The entry to be removed is at the front of the list */
            ELSE TG( TG(Group).RL$fp ).RL$bp : TG(Group).RL$bp;

          TG(Group).RL$bp : Non$Group;
          TG(Group).RL$fp : Non$Group;
          RL$Size : RL$Size - 1;
        END;
    END RL$Remove;


/*PROCEDURE */ RL$Dequeue : Procedure (Group$Ptr, Status$Ptr);
                           Declare  Group$Ptr  Pointer,
                                    Status$Ptr Pointer;
    /*VAR*/
      Declare
        Group    BASED Group$Ptr   Group$Num,
        Status   BASED Status$Ptr  Status$Type;

    /*BEGIN*/
      IF (RL$Size : 0)
        THEN Status : Empty;
        ELSE
          DO;
```

```
            Status = Successful;
            Group = RL$Front;
            CALL RL$Remove (Group);
          END;
      END RL$Dequeue;


/*PROCEDURE */ Tracks$Insert: Procedure (Group, Reqd$Qty$Ptr, Min$Qty, Status$Ptr);
                              Declare  Group         Group$Num,
                                       Reqd$Qty$Ptr  Pointer,
                                       Min$Qty       Byte,
                                       Status$Ptr    Pointer;

    /*CONST*/
        Declare
          Not$Response   literally '0';
    /*VAR*/
        Declare
          Count          Integer,
          Dumm$Track     Track$Num,
          Dumm$Group     Group$Num,
          Read$Msg$Size  Byte,
          Read$Msg$Seg   Token,
          Exception      IO$Exception;
        Declare
          Read$Msg       BASED Read$Msg$Seg     Read$Msg$Structure,
          Reqd$Qty       BASED Reqd$Qty$Ptr     Byte,
          Status         BASED Status$Ptr       Status$Type,

    /*BEGIN*/
        /* Release space for the desired number of track-entries */
        DO WHILE (Free$Tracks < Reqd$Qty);
          CALL RL$Dequeue (@Dumm$Group, Status$Ptr);
          IF (Status = Successful)
            THEN CALL Release$Group (Dumm$Group);
        END;

        /* Create memory segment to hold message for the Read Process, listing Tracks to read from disk */
        IF (Free$Tracks < Min$Reqd)
          THEN Status = Not$Cache$Space;
          ELSE DO;
                 IF (Free$Tracks < Reqd$Qty)
                   THEN Reqd$Qty = Free$Tracks;
                 Read$Msg$Size = (SIZE(Read$Msg) - 1) + (Reqd$Qty * SIZE(T$Rnt$SLR) );
                 Read$Msg$Seg  = RQ$CREATE$SEGMENT (Read$Msg$Size, @Exception);
                 IF (Exception <> E$OK)
                   THEN Status = Not$Mem$Space;
                   ELSE Status = Successful;

        IF (Status = Successful)
          THEN DO;
                 /* Construct first part of message */
                 Read$Msg.Qty     = Reqd$Qty;
                 Read$Msg.Disk    = TGL(Group).Disk;
                 Read$Msg.Group   = Group;
                 Read$Msg.Track$A = TGL(Group).Track$A;

                 /* Attach the first free Track-Entry to the Group's Track List, then step through the list
                     to find the new Space List Rock for Track Entries - Place each track in the Read Message */
                 TG(Group).TL$LLR  = T$Rnt$SLR;
```

```
                Read$Msg.Track(0) : T$Rnt$SLR;
                DO Count = 1 TO Reqd$Qty-1;
                  T$Rnt$SLR = T$Rnt( T$Rnt$SLR ).TL$fp;
                  Read$Msg.Track( Count ) = T$Rnt$SLR;
                END;
                Dumm$Track = T$Rnt$SLR;
                T$Rnt$SLR = T$Rnt( Dumm$Track ).TL$fp;
                Free$Tracks = Free$Tracks - Reqd$Qty;
                T$Rnt( Dumm$Track ).TL$fp = Non$Track;    /* Reset forward pointer for last track in the group */

                /* Send Message to Read Process */
                CALL RQ$SEND$MESSAGE (Read$Msg$Mbox, Read$Msg$Seg, Mo$Response, @Exception);
                IF (Exception = E$OK)
                  THEN TG(Group).Qty$Ops = Reqd$Qty;    /* This ensures that the tracks remain in the cache
                                                           until each has been successfully read from disk.
                                                           When marked present, the quantity of outstanding
                                                           operations is reduced. */
                  ELSE DO;
                       /* There is no memory space to enqueue the message */
                       Status = Mo$Mem$Space;
                       CALL RQ$DELETE$SEGMENT (Read$Msg$Seg, @Exception);
                     END;
              END;
        END Tracks$Insert;



/*PROCEDURE */ Group$Insert: Procedure (Disk, Track$A, New$Pos, Reqd$Qty, Min$Qty, Group$Ptr, Status$Ptr);
                           Declare  Disk       Dtype,
                                    Track$A    Type$PC_Track,
                                    New$Pos    Group$Num,
                                    Reqd$Qty   Byte,
                                    Min$Qty    Byte,
                                    Group$Ptr  Pointer,
                                    Status$Ptr Pointer;
  /*VAR*/
     Declare
       Group    BASED Group$Ptr    Group$Num,
       Status   BASED Status$Ptr   Status$Type;

  /*BEGIN*/
     /* If Necessary, & possible, release sufficient space for 1 track-group */
     IF (Free$Groups = 0)
       THEN DO;
            CALL RL$Dequeue (Group$Ptr, Status$Ptr);
            IF (Status = Successful)
              THEN CALL Release$Group (Group);
          END;

     IF (Free$Groups = 0)
       THEN Status = Mo$Cache$Space;
       ELSE DO;
            /* Remove a group from the track-group space list */
            Group  = TG$SLR;
            TG$SLR = TG(TG$SLR).TGL$fp;
            Free$Groups = Free$Groups - 1;

            /* Insert this in the required Track group list, after the new position */
            IF (New$Pos = Non$Group)
              THEN /* Insert at front of list */
```

```
                    DO;
                      TG(Group).TGL$fp : TGL(Disk).LLR;
                      TGL(Disk).LLR    : Group;
                    END;
                  ELSE DO;
                      TG(Group).TGL$fp : TG(New$Pos).fp;
                      TG(New$Pos).fp   : Group;
                    END;

                /* Insert the required number of track entries into the group */
                CALL Tracks$Insert (Group, @Reqd$Qty, Min$Qty, Status$Ptr);
                IF (Status <> Successful)
                  THEN CALL Release$Group (Group);
                  ELSE DO;
                       /* Update information in the new group */
                       TG(Group).Track$A : Track$A;
                       TG(Group).Track$B : Track$A + (Reqd$Qty * 2) - 1;
                       TG(Group).Disk    : Disk;
                     END;

              END;
    END Group$Insert;


/*PROCEDURE */ Release$Tracks: Procedure (Group);
                        Declare  Group  Group$Num;
  /*VAR*/
     Declare
       Track, Next$Track    Track$Num;

  /*BEGIN*/
     /* Release the track entries in the track-group's tracklist to the Track-Entry Spacelist*/
     Next$Track : TG(Group).TL$LLR;
     IF (Next$Track <> Hom$Track) /* i.e. if the group has tracks in it's tracklist */
       THEN DO;
            DO WHILE ( Next$Track <> Hom$Track );
              Track : Next$Track;
              Next$Track : Track.TL$fp;
              CALL Zero$Track$Entry (Track);
              Free$Tracks : Free$Tracks + 1;
            END;
            T$Ent(Track).TL$fp : T$Ent$SLR
            T$Ent$SLR : TG(Group).TL$LLR;
          END;
  END Release$Tracks;


/*PROCEDURE */ Release$Group: Procedure (Group);
                        Declare  Group  Group$Num;
  /*VAR*/
     Declare
       Dsk    literally   'TG(Group).Disk',    /* The disk to which the track group belongs */
       Prev   Group$Num;

  /*BEGIN*/
     /* Remove the group from its disk's track-group list - First find the preceding entry
                                                            and update it's forward pointer */
     Prev : TGL(Dsk).LLR;
     IF (Prev : Group)
```

```
        THEN TGL(Dsk).LLR = TG(Group).TGL$fp;       /* Group is first in list */
        ELSE
          DO;
            DO WHILE ( TG(Prev).TGL$fp <> Group );
              Prev = TG(Prev).TGL$fp;
            END;
            TG(Prev).TGL$fp = TG(Group).TGL$fp;
          END;

      /* Release the track entries in the track-group's tracklist */
      CALL Release$Tracks (Group);

      CALL Zero$Group$Entry (Group);
      TG(Group).TGL$fp = TG$SLR;        /* Insert at front of spacelist */
      TG$SLR = Group;
      Free$Groups = Free$Groups + 1;
    END Release$Group;


/*PROCEDURE */ Search$Track: Procedure (Disk, PC_Track, Group$Ptr, Track$Ptr, Status$Ptr);
                           Declare  Disk         Dtype,
                                    PC_Track     Type$PC_Track,
                                    Group$Ptr    Pointer,
                                    Track$Ptr    Pointer,
                                    Status$Ptr   Pointer;
  /*VAR*/
      Declare
        Count      Integer,
        Group      BASED Group$Ptr     Group$Num,
        Track      BASED Track$Ptr     Track$Num,
        Status     BASED Status$Ptr    Status$Type;

  /*BEGIN*/
      /* Scan the Track-group List of the required disk */
      Group = TGL(Disk).LLR;
      Status = Not$In$Cache;
      DO WHILE ( (Status <> Found) AND (Group <> Non$Group) );
        IF ( (TG(Group).Track$A <= PC_Track) AND (TG(Group).Track$B >= PC_Track)
          THEN Status = Found;
          ELSE Group = TG(Group).TGL$fp;
      END;

      IF (Status = Found)
        THEN   /* Traverse the Track-group's track-list & find the track-entry */
          DO;
            Count = ( PC_Track - TG(Group).Track$A ) / 2;  /* Each Track-entry has 2 PC-Tracks */
            Track = TG(Group).TL$LLR;
            DO Count = Count TO 1 BY (-1);
              Track = T$Ent(Track).TL$fp;
            END;
          END; /* Then */
    END Search$Track;


/*PROCEDURE */ TGL$Create$Entry: Procedure (Disk, PC_Track, Group$Ptr, Track$Ptr, Status$Ptr);
                               Declare  Disk         Dtype,
                                        PC_Track     Type$PC_Track,
                                        Group$Ptr    Pointer,
                                        Track$Ptr    Pointer,
```

```
                                    Status$Ptr   Pointer;

      /* The new entries track limits are evaluated based on the disk's track-group size.
         These limits are modified, so that track-groups do not overlap. At the same time
         the position for the new entry in the disk's TGL is found. Entries are released
         from the replacement list. Provided there are sufficient track-entries to satisfy
         a minimum value (which may be less than the desired quantity of tracks), then a
         new entry is inserted in the TGL */


/*CONST*/
    Declare
      Reqd$Group$Size  literally  'Op$Appl$Map( Op$Disk$Map(Disk).Op$Appl$Num ).Track$Group$Size';
                /* The group$size (in pairs of tracks) specified for the disk's application */
/*VAR*/
    Declare
      ( Track$A, Track$B, Closest )  Type$PC_Track,
      ( Scan$Group, New$Pos )        Group$Num,
      Reqd$Qty, Min$Reqd             Byte;       /* Refer to the number of track-entries required */
    Declare
      Group      BASED Group$Ptr     Group$Num,
      Track      BASED Track$Ptr     Track$Num,
      Status     BASED Status$Ptr    Status$Type;

/*EXEC*/
    /* Evaluate the desired track limits of the new group */
    Track$A = PC_Track - (PC_Track MOD 2);            /* Will be even */
    Track$B = Track$A + (2 * Reqd$Group$Size) - 1;    /* Will be odd  */
    IF (Track$B = PC_Track)
      THEN Track$B = Track$B + 2; /* Prefetching requires that at least one more track be present.
                          With a group size of one, this may involve increasing the group size */

    /* Ensure that track-groups will not overlap */
      /* Scan the TGL for the first track higher than Track$A */
        Scan$Group = TGL(Disk).LLR;
        New$Pos = Non$Group;
        Closest = Last$Track + 1;
        DO WHILE ( (Closest > Last$Track) AND (Scan$Group <> Non$Group) );
          IF ( TG(Scan$Group).Track$A > Track$A )
            THEN Closest = TG(Scan$Group).Track$A;
            ELSE DO;
                   New$Pos = Scan$Group;
                   Scan$Group = TG(Scan$Group).TGL$fp;
                 END;
        END; /* While */
      IF (Track$B >= Closest)
        THEN Track$B = Closest - 1;

    /* Find the desired & the minimum number of track entries needed */
    Reqd$Qty = ( (Track$B + 1) - Track$A ) / 2;
    Min$Reqd = 1 + (PC_Track MOD 2);       /* At least one track following the one required must be fetched */
    IF (Min$Reqd > Reqd$Qty)
      THEN Min$Reqd = Reqd$Qty;

    CALL Group$Insert ( Disk, Track$A, New$Pos, Reqd$Qty, Min$Qty, Group$Ptr, Status$Ptr );
    IF (Status = Successful)
      THEN Track = TG(Group).TL$LLR;

  END TGL$Create$Entry;
```

```
/*PROCEDURE */ Mark$Clean: Procedure (Group);
                        Declare  Group  Group$Num;
    /*CONSTS*/
       Declare
         Not$End$Track$List       literally  '(Track < Non$Track)',
         Test$All$Sectors$Clean   literally
             '( (T$Ent(Track).Sect$Bits(0) = 0) AND (T$Ent(Track).Sect$Bits(1) = 0)
                                                AND (T$Ent(Track).Sect$Bits(2) = 0) )';
    /*VAR*/
       Declare
         Track     Track$Num,
         Clean     Boolean;

    /*BEGIN*/
       Track = TG(Group).TL$LR;
       DO WHILE (Clean AND Not$End$Track$List);
         Clean = Test$All$Sectors$Clean;
         Track = T$Ent(Track).TL$fp;
       END; /* While */
       IF Clean
         THEN DO;
              TG(Group).Dirty     = False;
              TG(Group).Must$Write = False;
              TG(Group).Time$Stamp = 0;
              CALL WL$Remove (Group);
              IF ( TG(Group).Qty$Ops > 0 )
                 THEN CALL RL$Insert$Rear (Group);
            END; /* Then */
     END Mark$Clean;



/* CACHE OPERATORS - Available to be called by processes */


 /*PROCEDURE */ Cache_Init: Procedure;
     /*VAR*/
       Declare
         Disk       Dtype,
         Group      Group$Num,
         Track      Track$Num,
         Segment    Token,
         Exception  IO$Exception;

    /*BEGIN*/
       /* Initialize as empty, the disks track-group lists */
       DO Disk = 0 TO (Max$Ops$Disks - 1);
         TGL(Disk).LLR = Non$Group;
       END;

       /* Init & insert all group entries in the spacelist */
       TG$CLR = 0;
       DO Group = 0 TO (Max$Groups - 1);
         CALL Zero$Group$Entry (Group);
         TG(Group).TGL$fp = Group + 1;
       END;
       Free$Groups = Max$Groups;

       /* Allocate memory space for tracks, init track entries & insert all entries in the spacelist */
```

```
        T$Ent$SLR = 0;
        DO Track = 0 TO (Max$Tracks - 1);
          CALL Zero$Track$Entry (Track);
          Segment = RQ$CREATE$SEGMENT (Size$2Tracks, @Exception);
          T$Ent(Track).Track$Pos = Build$Ptr (0, Segment);
          T$Ent(Track).TL$fp     = Track + 1;
        END;
        Free$Tracks = Max$Tracks;
      END Cache_Init;


/*PROCEDURE */ Cache_Increase$Ops: Procedure (Group);
                                   Declare  Group  Group$Num;
    /*BEGIN*/
       TG(Group).Qty$Ops = TG(Group).Qty$Ops + 1;
       CALL RL$Remove (Group);
     END Cache_Increase$Ops;


/*PROCEDURE */ Cache_Reduce$Ops: Procedure (Group);
                                 Declare  Group  Group$Num;
    /*BEGIN*/
       TG(Group).Qty$Ops = TG(Group).Qty$Ops - 1;
       IF ( (TG(Group).Qty$Ops = 0) AND (TG(Group).Dirty = False) )
           THEN CALL RL$Insert$Rear (Group);
     END Cache_Reduce$Ops;


/*PROCEDURE */ Cache_Mark$Present: Procedure (Track, Group);
                                   Declare  Track  Track$Num,
                                            Group  Group$Num;
    /*BEGIN*/
       T$Ent(Track).Present = True;
       CALL Cache_Reduce$Ops (Group);
     END Cache_Mark$Present;


/*FUNCTION */ Cache_Query$Present: Procedure (Track)  Boolean;
                                   Declare  Track   Track$Num;
    /*BEGIN*/
       RETURN  (T$Ent(Track).Present);
     END Cache_Query$Present;


/*FUNCTION */ Cache_Fetch: Procedure (Disk, PC_Track, Group$Ptr, Track$Ptr, Track$Pos$Ptr)  Status$Type;
                           Declare   Disk      Disk$Num,
                                     PC_Track  Type$PC_Track,
                                     (Group$Ptr, Track$Ptr, Track$Pos$Ptr)  Pointer;
    /*VAR*/
       Declare
         Group       BASED  Group$Ptr       Group$Num,
         Track       BASED  Track$Ptr       Track$Num,
         Track$Pos   BASED  Track$Pos$Ptr   Pointer,
         Status      Status$Type;

    /*BEGIN*/
       CALL Search$Track (Disk, PC_Track, Group$Ptr, Track$Ptr, @Status);
       IF (Status = Found)
         THEN CALL RL$Remove (Group);
         ELSE CALL TGL$Create$Entry (Disk, PC_Track, Group$Ptr, Track$Ptr, @Status);
```

```
        IF (Status : Successful)
          THEN DO;
                CALL Cache_Increase$Ops (Group);
                Track$Pos : T$Ent(Track).Track$Pos;
                PC_Track : PC_Track + 1;
                /* If necessary, then prefetch next track */
                IF ( T$Ent(Track).TL$fp : Mon$Track )
                  THEN IF (PC_Track =< Last$Track)
                         THEN CALL Cache_Prefetch (Disk, (PC_Track + 1) );
              END;
        RETURN (Status);
      END Cache_Fetch;


/*PROCEDURE */ Cache_Prefetch: Procedure (Disk, PC_Track);
                               Declare   Disk      Disk$Num,
                                         PC_Track  Type$PC_Track;
    /*VAR*/
        Declare
          Group       Group$Num,
          Track       Track$Num,
          Track$Pos   Pointer,
          Status      Status$Type;

    /*BEGIN*/
        CALL Search$Track (Disk, PC_Track, @Group, @Track, @Status);
        IF (Status : Found)
          THEN DO;
                CALL RL$Remove (Group);
                IF ( ( TG(Group).Qty$Ops : 0) AND ( TG(Group).Dirty : False) )
                  THEN CALL RL$Insert$Rear (Group);
               END;
          ELSE CALL TGL$Create$Entry (Disk, PC_Track, @Group, @Track, @Status)
      END Cache_Prefetch;


/*FUNCTION */ Cache_WL$Ready: Procedure (Group$Ptr)   Boolean;
                              Declare  Group$Ptr Pointer;

        /* The front entry in the writelist is examined to determine if a write-to-disk must be
            performed. This depends on the size of the queue, the time elapsed since the track
            was updated, & whether the disk is to be removed from the cache - marked Must$Write */

    /*CONST*/
        Declare
          WL$Long       literally   '5',
          Wait$To$Write literally   '5',   /* 5 seconds delay */
          Time$Up       literally   'LOW (RO$GET$TIME (@Exception) ) - TG(Group).Time$Stamp';
    /*VAR*/
        Declare
          Write$Ready   Boolean,
          Exception     IO$Exception,
          Group         BASED Group$Ptr   Group$Num;

    /*BEGIN*/
        Write$Ready : False;
        IF (WL$Size <> 0)
          THEN DO;
                Group : WL$Front;
                IF ( TG(Group).Must$Write  OR  (WL$Size > WL$Long)  OR  (Time$Up > Wait$To$Write) )
```

```
                THEN Write$Ready = True;
             END; /* Then */
          RETURN Write$Ready;
       END Cache_WL$Ready;


/*FUNCTION */ Cache_Next$Write: Procedure  (Disk, Group$Ptr, PC_Track$Ptr, Track$Ptr, Sect$Bits$Ptr)   Boolean;
                               Declare   Disk       DkType,
                                        (Group$Ptr, PC_Track$Ptr, Track$Ptr, Sect$Bits$Ptr)  Pointer;

   /*CONST*/
      Declare
        Not$End$Group$List       literally  '(Group <> Non$Group)',
        Not$End$Track$List       literally  '(Track <> Non$Track)',
        Test$Dirty$Sector        literally  '( (T$Ent(Track).Sect$Bits(0) <> 0)
                                            OR (T$Ent(Track).Sect$Bits(1) <> 0)
                                            OR (T$Ent(Track).Sect$Bits(2) <> 0) )';

   /*VAR*/
      Declare
        Group           BASED  Group$Ptr          Group$Num,
        PC_Track        BASED  PC_Track$Ptr       Type$PC_Track,
        Track           BASED  Track$Ptr          Track$Num,
        Sect$Bits(3)    BASED  Sect$Bits$Ptr      Byte;
      Declare
        Count           Byte,
        Next$Found      Boolean;

   /*BEGIN*/
      Next$Found = False;
      IF (Group = Non$Group)
        THEN Group = TGL(Disk).Group$Num;

      DO WHILE ( (NOT (Next$Found)) AND (Not$End$Group$List) );
        IF (Track = Non$Track)
          THEN DO;
                 Track = TG(Group).TL$LLR;
                 PC_Track = TG(Group).Track$A;
               END;
          ELSE DO;
                 Track = T$Ent(Track).TL$fp;
                 PC_Track = PC_Track + 2;
               END;
        DO WHILE ( (Next$Found) AND (Not$End$Track$List) );
          Next$Found = Test$Dirty$Sector;
          IF (NOT Next$Found)
            THEN DO;
                   Track = T$Ent(Track).TL$fp;
                   PC_Track = PC_Track + 2;
                 END;
        END; /* While - Stepping through Tracks in a Group Entry */

        IF (NOT Next$Found)
          THEN DO;    /* A complete group has been scanned, without finding a dirty track */
                 CALL Mark$Clean(Group);
                 Group = TG(Group).TGL$fp;
               END; /* Then */
      END; /* While - Stepping through Groups in the disk's Group list */

      IF Next$Found
```

```
            THEN DO;   /* A dirty track was found and must be written */
                 DO Count = 0 TO 2;
                   Sect$Bit(Count) = TkSDir(Track).Sect$Bit(Count);
                   TkSDir(Track).Sect$Bit(Count) = 0;
                 END;
               END;

        RETURN (Rest$Found);
      END Cache_Rest$Write;


/*PROCEDURE */ Cache_Perform$Write: Procedure (Group, Track, Sector, SI_Sect$Pos);
                                  Declare  Group       Group$Num,
                                           Track       Track$Num,
                                           Sector      Type$Pc_Sect,
                                           SI_Sect$Pos  Pointer;
  /*VARS*/
    Declare
      Sect$Pos    Pointer;

  /*PROCEDURE */ Sect$Bit  Procedure
    /*DCL*/
      Declare
        Sect$Bits     Literally   'TkSDir(Track).Sect$Bits';
    /*VARS*/
      Declare
        (Reg$Byte, Reg$Bit)       Byte,
    /*EXEC*/
      Reg$Byte = Sector/8;
      Reg$Bit  = Sector MOD 8;
      Reg$Bit  = SHR( 0000$0000$1, Reg$Bit);
      Sect$Bits(Reg$Byte) = Sect$Bits(Reg$Byte) OR Reg$Bit;
    END Sect$Bit;

  /*EXEC*/
    /* Copy the sector from the sector pool, to its position in the track storage of the cache */
      Sect$Pos = TkSDir(Track).Track$Pos + (Sector * Sect$Size);
      CALL MOVE (SI_Sect$Pos, Sect$Pos, Sect$Size);
    /* Mark a bit in the Track Entry indicating that this sector is 'dirty' */
      CALL Sect$Bit;
    /* Update the Group Entry to which the track belongs */
      TSL(Group).Dirty = True;
      CALL MSL$Remover (Group);
      CALL MSL$Insert$Rear (Group);
      CALL Write$Stops (Group);
  END Cache_Perform$Write;


/*PROCEDURE */ Cache_Flush  Procedure (Disk)  Boolean;
                          Declare  Disk   Disk$Num;
  /*VARS*/
    Declare
      Group       Group$Num,
      Rest$Group  Group$Num,
      Flushed     Boolean;

  /*EXEC*/
    Group = TSL(Disk).LLR;
    Flushed = True;
    DO WHILE (Group <> Nons$Group);
```

```
            Next$Group = TG(Group).TGL$fp;
            IF ( NOT (TG(Group).Dirty) )
              THEN CALL Release$Group (Group);
              ELSE DO;
                     Flushed = False;
                     IF ( NOT (TG(Group).Must$Write) )
                       THEN DO;
                              TG(Group).Must$Write = True;
                              CALL WL$Remove(Group);
                              CALL WL$Insert$Front(Group);
                            END;
                   END; /* Else */
            Group = Next$Group;
          END; /* While */
          RETURN (Flushed);
        END Cache_Flush;


/* END  CACHE RESOURCE */

/* ------------------------------------------------------------------------------------------ */
```

B.3.7 Disk-Read Queue - *File:* DSK$RADQ.RSC

```
/* ------------------------------------------------------------------------------------------ */

/* RESOURCE: DISK$READ$Q */

  /*VAR*/
     Declare
       Read$Msg$Mbox        Token;

  /*PROCEDURE */ Disk$Read$Q_Init: Procedure;
     /*COMMENT*/
        Declare
          Mbox$Flags        literally       '1Dh';     /* Sets high performance queue to maximum
                                                          size (60), and the queue to FIFO */
     /*VAR*/
        Declare
          Exception         IO$Exception;
     /*BEGIN*/
        Read$Msg$Mbox : RQ$CREATE$MAILBOX (Mbox$Flags, @Exception);
       END Disk$Read$Q_Init;


  /* Other queue operations are performed by operating system calls, and are listed here */

    /*PROCEDURE RQ$SEND$MESSAGE: Procedure (Mbox, Msg$Segment, Response, Exception$Ptr);
                                 Declare (Mbox, Msg$Segment, Response)  Token,
                                         Exception$Ptr      Pointer;

        !!  Response should be zero, since no response is required !!
        !!  Msg$Segment, contains a token for the message segment & is added to the queue  !!  */


    /*FUNCTION  RQ$RECEIVE$MESSAGE: Procedure (Mbox, Wait$Time, Response$Ptr, Exception$Ptr)   Token;
                                    Declare  Mbox        Token,
                                             Wait$Time   Word,
                                            (Response$Ptr, Exception$Ptr)  Pointer;

        !! RETURNS the Token for the segment containing the message !!  */


/* END  DISK$READ$Q RESOURCE */

/* ------------------------------------------------------------------------------------------ */
```

B. 4 2ND-LEVEL PROCESSES

B.4.1 Initialization Process - *File*: INIT.PRS

```
/* ------------------------------------------------------------------------------------------ */

/* PROCESS: INITIALIZATION */
   /* Initialization includes, creating the Regions for resource access resolution, initializing
      the resources, reading system configuration information (which would be flexible) and
      initiating the concurrent tasks (except for the server tasks which are only created when
      the first request from a node is received). */

  /*PROCEDURE*/ Initialization: Procedure;

    /*CONST*/
       Declare
         PS$ID                literally    '10h,0',          /* Second & First byte: of word: ID : 10h */
         PS$Id$Entry          literally    '(1,0,1,0,PS$ID)',
         Region$Flags         literally    '1';              /* Priority based queuing of tasks */


    /*VAR*/
       Declare
         PS$User            Token,
         Exception          IO$Exception,
         Error$Count        Word,
         Queue$Prod$Task    Token;


    /*PROCEDURE */ Configure$PS: Procedure;
     /*CONST*/
        Declare
          File              literally      '(6, ''CONFIG'')',
          Qty$Buffers       literally      '1',
          Max$Perms$Files   literally      '20';
     /*VAR*/
        Declare
          Config$File       Token,
          Exception         IO$Exception,
          Bytes$Read        Word,
          Qty$Perms$Files   Byte,
          Count             Byte,
          File$Num          Type$File$Num,
          Qty               Byte,
          First             Type$Op$Disk$Num,
          Status            Status$Type;

     /*BEGIN*/
        Config$File : RQ$S$ATTACH$FILE (@File, @Exception);
        CALL RQ$S$OPEN (Config$File, Reading, Qty$Buffers, @Exception);

        Bytes$Read : RQ$S$READ$MOVE (Config$File, @Def$OS, 1, @Exception);
        Bytes$Read : RQ$S$READ$MOVE (Config$File, @Qty$Perms$Files, 1, @Exception);
        IF (Qty$Perms$Files > Max$Perms$Files)
          THEN Qty$Perms$Files : Max$Perms$Files;
        DO Count : 1 TO Qty$Perms$Files;
          Bytes$Read : RQ$S$READ$MOVE (Config$File, @File$Num, 2, @Exception);
```

```
        CALL Common$Disk$Open (File$Num, Immediate, Read, Any$OS, Carte$Blanche, @Qty, @First, @Status);
        IF (Status <> Successful)
          THEN Error$Count = Error$Count + 1;
      END; /* Do Loop */

      CALL RQ$S$DELETE$CONNECTION (Config$File, @Exception);
    END Configure$PS;


/*PROCEDURE */ Create$All$Tasks: Procedure;
  /*CONST*/
      Declare
        Dseg              literally   '0',      /* Assigns its own Data Segment */
        Stack$Ptr         literally   'Nil',    /* Nucleus must assign a stack  */
        Flags             literally   '0';      /* No floating point instructions */

      /* Queue Producing Process */
      Declare
        QP_Priority       literally   '210',    /* Lower than the server tasks */
        QP_Start          literally   '@Queue$Producer',
        QP_Stack$Size     literally   '2FFh';   /* 500 Bytes */

      /* Disk Read Process */
      Declare
        DR_Priority       literally   '210',    /* Lower than the server tasks */
        DR_Start          literally   '@Disk$Read',
        DR_Stack$Size     literally   '2FFh';   /* 500 Bytes */

      /* Disk Write Process */
      Declare
        DW_Priority       literally   '210',    /* Lower than the server tasks */
        DW_Start          literally   '@Disk$Write',
        DW_Stack$Size     literally   '2FFh';   /* 500 Bytes */

      /* Complete Sends Process */
      Declare
        CS_Priority       literally   '210',    /* Lower than the server tasks */
        CS_Start          literally   '@Complete$Sends',
        CS_Stack$Size     literally   '2FFh';   /* 500 Bytes */


    /*BEGIN*/
      Queue$Prods$Task     : RQ$CREATE$TASK (QP_Priority, QP_Start, Dseg, Stack$Ptr, QP_Stack$Size, Flags, @Except
      Disk$Reads$Task      : RQ$CREATE$TASK (DR_Priority, DR_Start, Dseg, Stack$Ptr, DR_Stack$Size, Flags, @Exception);
      Disk$Writes$Task     : RQ$CREATE$TASK (DW_Priority, DW_Start, Dseg, Stack$Ptr, DW_Stack$Size, Flags, @Exception);
      Complete$Sends$Task : RQ$CREATE$TASK (CS_Priority, CS_Start, Dseg, Stack$Ptr, CS_Stack$Size, Flags, @Exception);
    END Create$All$Tasks;


/*BEGIN  Initialization */
    /* Create protected regions for resources used by more than one task */
    HD$Attach$Region   : RQ$CREATE$REGION (Region$Flags, @Exception);
    Comm_~t$Info$Region : RQ$CREATE$REGION (Region$Flags, @Exception);
    User$Info$Region   : RQ$CREATE$REGION (Region$Flags, @Exception);
    Ops$Table$Region   : RQ$CREATE$REGION (Region$Flags, @Exception);
    SP$Region          : RQ$CREATE$REGION (Region$Flags, @Exception);
    MW$Out$Region      : RQ$CREATE$REGION (Region$Flags, @Exception);
    Mode$Rqsts$Region  : RQ$CREATE$REGION (Region$Flags, @Exception);
    Cache$Region       : RQ$CREATE$REGION (Region$Flags, @Exception);
```

```
        /* Initialize Resources - do not need to gain access, since no other tasks exist yet */
        CALL HD$Attach_Init;
        CALL Common$Info_Init;
        CALL User$Info_Init;
        CALL Op$Table_Init;
        CALL WW$Inp_Init;
        CALL SP_Init;
        CALL WW$Out_Init;
        CALL Node$Reqsts_Init;
        CALL Disk$Read$Q_Init;
        CALL Cache_Init;

        /* Read in the default Op. Syst. & open Common disks which are to be permanently available */
        CALL Configure$PS;

        /* Create tasks */
        CALL Create$All$Tasks;

    END Initialization;

/* END INITIALIZATION PROCESS */

/* ------------------------------------------------------------------------------------------ */
```

B.8.2 Queue-Producing Process : File: Q$PRODUC.PRS

```
/* PROCESS  QUEUE PRODUCER */
   /* Reads the Network input queue & redirects the requests to individual queues for each node */


   /*VAR*/
      Declare
        Node$Num$Semaphore  Token,
        New$Node$Num        Node$Type;  /* This variable is used to pass the node number to a new serving
                                           task. It's Access is controlled by the Node$Num Semaphore */


   /*PROCEDURE */ Queue$Producer: Procedure;

      /*CONST*/
         Declare
           Reqd$Queue       literally  'Node$Queues(Node)',
           Wait$Done        literally  '0FFFFh',
           Time$Len         literally  '0';  /* Allows other tasks to become active */

         /* For the Semaphore Creation */
         Declare
           Init$0           literally  '0',
           Max$1            literally  '1',
           FIFO             literally  '0';

         /* For creating a server task */
         Declare
           Server$Priority  literally  '200',
           Start            literally  '@Server',  /* Start of sever procedure */
           D$Seg            literally  '0',        /* Creates own Data Segment */
           S$Ptr            literally  'Nil',      /* Nucleus allocates Stack  */
           Stack$Size       literally  '$FFFh',    /* 1 Kbyte Stack */
           Flags            literally  '0';

      /*VAR*/
         Declare
           Status           Status$Type,
           Dummy            word,
           Exception        IO$Exception,
           Node             Node$Type;

      /*BEGIN*/
         Node$Num$Semaphore = RQ$CREATE$SEMAPHORE (Init$0, Max$1, FIFO, @Exception);

         DO Forever;

           /* Wait until there is at least one input from the Network */
           DO WHILE (NW$Inp_Queue$Size = 0);
              CALL RQ$SLEEP (Time$Len, @Exception);
           END; /* While */
           Node = NW$Inp_Queue(NW$Inp_Read$Pos).Node;  /* The node which sent the request */

           /* Examine the Node's queue to determine if there is already a server task
                associated with it.  If not then create a server task */
           IF (Reqd$Queue.Server = Non$Token)
             THEN
               DO;
```

```
                New$Node$Num = Node;
                Reqds$Queue.Server = RQ$CREATE$TASK (Server$Priority, Start, D$Seg, St$Ptr,
                                                     Stack$Size, Flags, @Exception);
                Dummy = RQ$RECEIVE$UNITS ( Node$Num$Semaphore, 1, Wait$Done, @Exception );
                /* Wait for new task to send the semaphore. Allows task access to its node number */
              END;

            /* Wait until there is at least one free entry in the Node Queues' spacelist,
                & then enqueue the request */
            Status = Full;
            DO WHILE (Status = Full);
              CALL RQ$RECEIVE$CONTROL ( Node$Rqsts$Region, @Exception);
                CALL Nodes$Rqsts_Enqueue (Node, @Status);
              CALL RQ$SEND$CONTROL (@Exception);   /* Release the Nodes' Request queues Resource */
              IF (Status = Full)
                THEN CALL RQ$SLEEP (Time$Len, @Exception);,
            END;

          END; /* Forever Loop */
        END Queue$Producer;

/* END QUEUE PRODUCING PROCESS */
```

B.4.3 Individual-Node Server Process

This process is performed by each task set up for a workstation. It contains as an internal resource, a map of the workstation's drives, with further decomposition into sub-processes handling each type of request.

Node-Server Process Body - File: SERVER.PRS

```
/* ------------------------------------------------------------------------------ ----- ----- */

/* PROCESS: SERVER */
   /* Code for each serving task for each active node */


   /*PROCEDURE */ Server: Procedure  REENTRANT;


    /*CONST*/
       Declare
         Max$Pn            literally   '7',
         Sign$On           literally   '2',
         Resp$Size         literally   'SIZE (Resp)',
         Time$Len          literally   '0';  /* Allows other t           control */

       /* Message functions - Track, Response or Command for '          IU */
         Declare
           Rqsted$Track    literally  '0',
           Response$to$Rqst literally  '1',
           Crunch          literally  '2',
           Set$Perm        literally  '3',
           Attach          literally  '4',
           Detach          literally  '5';

    /*VAR*/
       Declare
         Node                  Node$Type,
         Exception             Exception,
         Status                Status$Type,
         User$Connected        Boolean,
         Request               Request$Type,
         Response              Message$Type,
         Pn                    Byte   AT ( @Request(0) ),
         Current               STRUCTURE ( User$Num      Byte,
                                           Access$Config Byte,
                                           Max$Qty       Byte,
                                           OS$Num        Byte,
                                           User$File$Num Type$File$Num,
                                           Dir$Connect   Token );


   /*PROCEDURE */ Send$Node$Message: Procedure (Msg$Ptr, Size);
                                   Declare  Msg$Ptr  Pointer,
                                            Size     Byte;
      /*BEGIN*/
         Status : Full;
         DO WHILE (Status : Full);
           CALL RQ$RECEIVE$CONTROL (MW$Out$Region, @Exception);
```

```
                CALL NV$Out_Enqueue (Msg$Ptr, Size, Node, @Status);
                CALL RQ$SEND$CONTROL (@Exception);  /* Release Network Output Queue Resou  . */
                IF (Status <> Successful)
                  THEN CALL RQ$SLEEP (Time$Len, @Exception);
              END;
            END Send$Node$Message;


    /*RESOURCES*/
$Include (drive.RSC)

    /*SERVER PROCEDURES */
$Include (Serv$Proc.RTN)

    /*REQUEST INTERPRETING PROCEDURES*/
$Include (Rqst$Proc.RTN)


    /*BEGIN - Server Procedure */
        /* Get the node which the task will be serving - then release access of the variable */
        Node = New$Node$Num;
        CALL RQ$SEND$UNITS ( New$Node$Semaphore, 1, @Exception);

        /* Initialize the task with only the default Op.Syst connected to the drives */
        CALL Drive$Map_Init;
        CALL Zero$Current$Settings;
        CALL Attach$OS;

        DO Forever;
          /* Get request from input queue */
          Status = Empty;
          DO WHILE (Status = Empty);
            CALL RQ$RECEIVE$CONTROL (Node$Queues$Region, @Exception);
              CALL Node$Rqsts_Dequeue (Node, @Request, @Status);
            CALL RQ$SEND$CONTROL (@Exception);  /* Release Node Queues Resource */
            IF (Status <> Successful)
              THEN CALL RQ$SLEEP ( Time$Len, @Exception);
          END;

          IF ( Fn <= Max$Fn )
            THEN IF ( User$Connected OR ( Fn <= Sign$On ) )
              THEN
                DO  CASE Fn;
                  /* 0 Read */              CALL Rqst$Track;
                  /* 1 Write */             CALL Rqst$Sect;
                  /* 2 Sign$On */           CALL Rqst$Sign$On;
                  /* 3 Sign$Off */          CALL Rqst$Sign$Off;
                  /* 4 Attach$Common */     CALL Rqst$Common$Disk;
                  /* 5 Attach$User$Disk */  CALL Rqst$User$Disk;
                  /* 6 Detach */            CALL Rqst$Detach;
                  /* 7 Change$OS */         CALL Rqst$OS;
                END; /* Case Fn */
        END; /* Forever Loop */
    END SERVER;

/* END SERVER PROCESS */

/* ------------------------------------------------------------------------------------------ */
```

Drive-Map : File: DRIVE.RSC

```
/* ..... */

/* RESOURCE:  DRIVE$MAP */
   /* Maps the drives of a node to an open file (Op$Disk$Num)
      on the File Server */

  /*CONSTs*/
     Declare
       Max$Drives          literally     '26',
       Non$Drive           literally     '1',
       Drive$Type          literally     'Byte',
       Map$Size            literally     'SIZE (Map)';

  /*VARs*/
     Declare
       Perm$Drive$Limit          Drive$Type,
       Drive$Array (Max$Drives)  D$Type,
       MIU$Message               Message$Type;


  /*PROCEDURE */  Drive$Map_Init: Procedure;
     /*VARs*/
        Declare
          Count   Drive$Type;
     /*BEGINs*/
        DO Count = 0 TO (Max$Drives-1);
          Drive$Array(Count) = Non$Op$Disk;
        END;
        Perm$Drive$Limit = 0;
      END  Drive$Map_Init;


  /*PROCEDURE */  Drive$Map_Crunch: Procedure;
     /*VARs*/
        Declare
          Free$Drive      Drive$Type,
          Count           Drive$Type;
     /*BEGINs*/
        /* Find the first free drive */
        Count = 0;
        DO WHILE ( (Count < (Max$Drives-1) ) AND
                 ( (Drive$Array(Count) <> Non$Op$Disk) OR (Count = Non$Drive) ) );
          Count = Count + 1;
        END;  /* While */
        Free$Drive = Count;

        /* Shift disks in subsequent used drives to the first free drives */
        DO Count = (Free$Drive + 1) TO (Max$Drives - 1);
          IF ( (Drive$Array(Count) <> Non$Op$Disk) AND (Count <> Non$Drive) )
            THEN
             DO;
               Drive$Array(Free$Drive) = Drive$Array(Count);
               Drive$Array(Count) = Non$Op$Disk;
               Free$Drive = Free$Drive + 1;
               IF (Free$Drive = Non$Drive)
                 THEN Free$Drive = Free$Drive + 1;
             END;
```

```
      END; /* Do Loop */
    END Drive$Map_Crunch;


/*PROCEDURE */ Drive$Map_Set$Perm: Procedure;
  /*VARS*/
    Declare     Count   Drive$Type;
  /*BEGIN*/
    /* Finds the highest attached drive & sets itself & all drives
       less than it as permanent */
    Count = Max$Drives - 1;
    DO WHILE ( (Count >= 0) AND (Drive$Array(Count) = Non$Op$Disk) );
      Count = Count-1;
    END; /* While */
    Perm$Drive$Limit = Count + 1;
  END Drive$Map_Set$Perm;


/*PROCEDURE */ Drive$Map_Attach: Procedure (Op$Disk$Num, Qty, File$Num, Drive1$Ptr, Status$Ptr);
                                  Declare  Op$Disk$Num    Dtype,
                                           Qty            Byte,
                                           File$Num       Ptype,
                                           Drive1$Ptr     Pointer,
                                           Status$Ptr     Pointer;


  /*VARS*/
    Declare
      Drive1      BASED Drive1$Ptr  Drive$Type,
      Status      BASED Status$Ptr  Status$Type,
      Msg         STRUCTURE ( Command    Byte,
                              Drive      Drive$Type,
                              File$Num   Ptype,
                              Qty        Byte );
    Declare
      Attempts    Byte,
      Crunched    Boolean,
      Free$Count  Byte,
      Count       Drive$Type;

  /*BEGIN*/
    Status = No$Drive$Space;
    Attempts = 0;
    Crunched = False;

    DO WHILE ( (Status = Successful) AND (Attempts < 2) );

      /* The Drive$Array is scanned to find the required number of
         sequentially available drives. If this fails the Drive$Array
         is crunched and then another attempt is made */

      Attempts = Attempts + 1;
      Count = 0;
      Free$Count = 0;

      /* Scan for available drives */
      DO WHILE ( (Count < Max$Drives) AND (Status = Successful) );
        IF ( Drive$Array(Count) = Non$Op$Disk )
          THEN Free$Count = 0;
          ELSE
```

```
                IF ( Count < Non$Drive )
                  THEN
                    DO;
                      Free$Count = Free$Count + 1;
                      IF ( Free$Count = Qty )
                        THEN Status = Successful;
                    END;
                Count = Count + 1;
              END; /* While */

              IF ( ( NOT Crunched ) AND ( Status < Successful ) )
                THEN
                  DO;
                    Crunched = True;
                    CALL Drive$Map_Crunch;
                  END;

            END; /* While */

            IF (Status = Successful)
              THEN
                DO;
                  Drive1 = Count - Qty;
                  DO Count = 0 TO (Qty - 1);
                    Drive$Array(Drive1+Count) = (Op$Disk$Num + Count);
                  END;
                  /* Send message to MIU */
                  Msg.Command    = Attach;
                  Msg.Drive      = Drive1;
                  Msg.File$Num   = File$Num;
                  Msg.Qty        = Qty;
                  CALL Send$Node$Message (@Msg, Msg$Size);
                END;

          END Drive$Map_Attach;


/*PROCEDURE */ Drive$Map_Detach: Procedure ( Drive1, Qty );
                                 Declare  Drive1 Drive$Type,
                                          Qty    Byte;
    /*CONSTs*/
        Declare
          Time$Len        literally     '0';
    /*VARs*/
        Declare  Start    Drive$Type,
                 Count    Drive$Type,
                 Exception IO$Exception;

    /*BEGIN*/
        IF (Drive1 >= Perm$Drive$Limit)
          THEN Start = Drive1;
          ELSE Start = Perm$Drive$Limit;

        /* Flush each Disk from the cache if this is the only user. */
        /*   ZZTrap 3ZZ - may need to nest regions here  */
        Flushed = False;
        CALL RQ$RECEIVE$CONTROL (Op$Table$Region, @Exception);
        DO WHILE ( NOT(Flushed) );
          Flushed = True;
          DO Count = Start TO (Drive1 + (Qty - 1) );
```

```
            IF ( Op$Table_Qty$Open( Drive$Array(Count)) > 1)
              THEN DO;
                     CALL RQ$SEND$CONTROL (@Exception)
                      CALL RQ$RECEIVE$CONTROL (Cache$Region, @Exception);
                           Flushed = Flushed AND Cache_Flush ( Drive$Array(Count));
                      CALL RQ$SEND$CONTROL (@Exception);
                     CALL RQ$RECEIVE$CONTROL (Op$Table$Region, @Exception);
                   END;
            END; /* Do Loop */
            IF ( NOT(Flushed) )
              THEN DO;
                     CALL RQ$SEND$CONTROL (@Exception);
                     CALL RQ$SLEEP (Time$Len, @Exception);
                     CALL RQ$RECEIVE$CONTROL (Op$Table$Region, @Exception);
                   END;
          END; /*While*/

          /* Close the disk entries, once successfully flushed */
            DO Count = Start TO (Drive1 + (Qty - 1) );
              CALL Op$Table_Mark$Closed ( Drive$Array(Count) );
            END;
          CALL RQ$SEND$CONTROL (@Exception);
        END Drive$Map_Detach;


/* END DRIVE$MAP RESOURCE */

/* ..... */
```

3rd-Level Processes : File: RQSTPROC.PTN

```
/* ------------------------------------------------------------------ */
/* REQUEST INTERPRETING PROCEDURES - MAPS THE REQUEST & RESPONSE BYTE ARRAYS INTO
    VARIABLES & PERFORMS PROCEDURE CALLS WITH THESE PARAMETERS */


/*PROCEDURE */ Rqst$Track: Procedure;
   /*CONST*/
      Declare
        Time$Len  literally   '0';
   /*VAR*/
      Declare
        Track     Track$Num
        Exception IO$Exception;
      Declare
        Rqst      STRUCTURE ( Drive       Byte,
                              PC_Track    Type$PC_Track,
                              Head        Byte )      AT ( @Request(1) ), /* 3-Bytes */
        Resp      STRUCTURE ( Fn          Byte,
                              Cache$Group Group$Num,
                              Drive       Byte,
                              PC_Track    Type$PC_Track,
                              Head        Byte,
                              Track$Pos   Pointer )   AT ( @Response);
   /*BEGIN*/
      Resp.Fn       : Rqsted$Track;
      Resp.Drive    : Rqst.Drive;
      Resp.PC_Track : Rqst.PC_Track;
      Resp.Head     : Rqst.Head;
      Rqst.PC_Track : Rqst.PC_Track + Rqst.Head;
      CALL RQ$RECEIVE$CONTROL (Cache$Region, @Exception);
      DO WHILE ( Cache$Fetch (Drive$Map(Rqst.Drive), Rqst.PC_Track,
                              @Resp.Cache$Group, @Track, @Resp.Track$Pos) <> Found );
        CALL RQ$SEND$CONTROL (Cache$Region, @Exception);
        CALL RQ$SLEEP (Time$Len, @Exception);
        CALL RQ$RECEIVE$CONTROL (Cache$Region, @Exception);
      END; /* While */
      Resp.Track$Pos : Resp.Track$Pos + (Rqst.Head * Track$Size);
      DO WHILE ( Cache$Query$Present(Track) <> True );
        CALL RQ$SEND$CONTROL (Cache$Region, @Exception);
        CALL RQ$SLEEP (Time$Len, @Exception);
        CALL RQ$RECEIVE$CONTROL (Cache$Region, @Exception);
      END; /* While */
      /* Send Response to PC */
      CALL Send$Node$Message (@Resp, Resp$Size);
    END Rqst$Track;


/*PROCEDURE */ Rqst$Sect: Procedure;
   /*CONST*/
      Declare
        Time$Len  literally   '0';
   /*VAR*/
      Declare
        Status    Status$Type,
        Track     Track$Num,
```

```
            Group     Group$Num,
            Track$Pos Pointer,
            Exception IO$Exception;
          Declare
            Rqst      STRUCTURE ( Drive       Byte,
                                  PC_Track    Type$PC_Track,
                                  Head        Byte,
                                  PC_Sect     Type$PC_Sect,
                                  Sect        Sect$Num,
                                  Sect$Pos    Byte )       AT ( @Request(1) );  /* 5-Bytes */
        /*BEGIN*/
          Rqst.PC_Track = Rqst.PC_Track + Rqst.Head;
          CALL RQ$RECEIVE$CONTROL (Cache$Region, @Exception);
          DO WHILE ( Cache$Fetch (Drive$Map(Rqst.Drive), Rqst.PC_Track, @Group, @Track, @Track$Pos) <> Found );
            CALL RQ$SEND$CONTROL (Cache$Region, @Exception);
            CALL RQ$SLEEP (Time$Len, @Exception);
            CALL RQ$RECEIVE$CONTROL (Cache$Region, @Exception);
          END; /* While */
          DO WHILE ( Cache$Query$Present(Track) <> True );
            CALL RQ$SEND$CONTROL (Cache$Region, @Exception);
            CALL RQ$SLEEP (Time$Len, @Exception);
            CALL RQ$RECEIVE$CONTROL (Cache$Region, @Exception);
          END; /* While */
          Rqst.PC_Sect = Rqst.PC_Sect + (Rqst.Head * Sects$In$Track);
          CALL Cache$Perform$Write (Group, Track, Rqst.PC_Sect, Rqst.Sect$Pos);
          /* Get access to release sector to pool */
            CALL RQ$RECEIVE$CONTROL (SP$Region, @Exception);
              CALL SP_Release$Sect (Rqst.Sect);
            CALL RQ$SEND$CONTROL (SP$Region, @Exception);
        END Rqst$Sect;


/*PROCEDURE */ Rqst$Sign$On: Procedure;
    /*CONST*/
      Declare
        New$User  literally 'Current$User$Entry';
    /*VAR*/
      Declare
        Rqst      STRUCTURE ( User$Num    Word,
                              Password    Pass$Type )  AT ( @Request(1) ),  /* 9-Bytes */
        Resp      STRUCTURE ( Fn          Byte,
                              Status      Status$Type,
                              Access$Config Byte,
                              User$OS     Byte,
                              Max$U$Disks Byte )       AT ( @Response);

    /*BEGIN*/
      Resp.Fn = Response$To$Rqst;
      IF (User$Connected)
        THEN Resp.Status = Already$Signed$On;
        ELSE
          DO;
            CALL RQ$RECEIVE$CONTROL (User$Info$Region, @Exception);
              CALL User$Info_Read (Rqst.User$Num, @Rqst.Password, @Resp.Status);
                IF (Resp.Status = Successful)
                  THEN
                    DO;
                      Current.User$Num     = Rqst.User$Num;
                      Current.Access$Config = New$User.Access$Config;
                      Current.Max$Qty      = New$User.Max$Qty;
```

```
                    Current.OS$Num       : New$User.Init$OS;
                    Current.User$File$Num : SHL(Current.User$Num,4) + 8000h; /* Leftmost bit high
                          indicates a User file.  Right 4 bits zero, allow up to 15 user disks */
                    User$Connected : True;
                   END;
            CALL RQ$SEND$CONTROL (@Exception); /* Release User$Info Resource */

            IF (Resp.Status = Successful)
              THEN
                DO;
                  CALL Attach$OS; /* Attach the Op.Syst. & load the Config disk & User disks */
                  Resp.Access$Config = Current.Access$Config;
                  Resp.User$OS       = Current.OS$Num;
                  Resp.Max$U$Disks   = Current.Max$Qty;
                END;
            END; /* Else */

        /* Send Response */
        CALL Send$Node$Message (@Resp, Resp$Size);
      END Rqst$Sign$On;


/*PROCEDURE */ Rqst$Sign$Off: Procedure;
    /*BEGIN*/
      IF (User$Connected)
        THEN
          DO;
            IF (Current.Dir$Connect <> Non$Token)
              THEN CALL RQ$A$DELETE$CONNECTION (Current.Dir$Connect, No$Mailbox, @Exception);
            CALL Zero$Current$Settings;
            CALL Attach$OS;
          END;
      END Rqst$Sign$Off;


/*PROCEDURE */ Rqst$Common$Disk: Procedure;
    /*VAR*/
      Declare
        First$Track literally  '0',
        Rqst       STRUCTURE ( File$Num     Type$File$Num,
                               Load$Type    Boolean,
                               Read$Write   Boolean,     AT ( @Request(1) ),  /* 4-Bytes */
        Resp       STRUCTURE ( Fn           Byte,
                               Status       Status$Type,
                               Drive$1      Drive$Type,
                               Qty          Byte )       AT ( @Response);

    /*BEGIN*/
      Resp.Fn = Response$To$Rqst;
      CALL Common$Disk$Attach (Rqst.File$Num, Rqst.Load$Type, Rqst.Read$Write, Current.OS$Num,
                               Current.Access$Config, @Resp.Qty, @Resp.Drive$1, @Resp.Status);

      /* Prefetch disk1, from cache (if immediate load$type) */
      IF (Resp.Status = Successful)
        THEN IF (Rqst.Load$Type = Immediate)
               CALL Cache_Prefetch ( Drive$Array(Resp.Drive1), First$Track);

      /* Send Response to PC */
      CALL Send$Node$Message (@Resp, Resp$Size);
    END Rqst$Common$Disk;
```

```
/*PROCEDURE */ Rqst$User$Disk: Procedure;
   /*VARS*/
      Declare
        Rqst      STRUCTURE ( Disk          Byte,
                              Read$Write    Byte  )      AT ( @Request(1) ),  /* 2-Bytes */
        Resp      STRUCTURE ( Fn            Byte,
                              Status        Status$Type,
                              Drive         Drive$Type,   /* The drive of the last disk attached */
                              Qty$Done      Byte )       AT ( @Response);

   /*BEGIN*/
      Resp.Fn = Response$To$Rqst;
      CALL User$Disk$Attach (Rqst.Disk, Rqst.Read$Write, @Resp.Drive, @Resp.Qty$Done, @Resp.Status);
      /* Send Response to PC */
      CALL Send$Mode$Message (@Resp, Resp$Size);
    END Rqst$User$Disk;


/*PROCEDURE */ Rqst$Detach: Procedure;
   /*VARS*/
      Declare
        Rqst      STRUCTURE ( Drive$1       Drive$Type,
                              Qty           Byte )       AT ( @Request(1) );  /* 2-Bytes */
   /*BEGIN*/
      CALL Drive$Map_Detach (Rqst.Drive$1, Rqst.Qty);
    END Rqst$Detach;


/*PROCEDURE */ Rqst$OS: Procedure;
   /*VARS*/
      Declare
        Rqst      STRUCTURE ( OS$Num        Byte )       AT ( @Request(1) );  /* 1-Byte */
   /*BEGIN*/
      /* Check first to see that requested OS is not already attached */
      IF (Current.OS$Num <> Rqst.OS$Num)
        THEN
          DO;
            Current.OS$Num = Rqst.OS$Num;
            CALL Attach$OS; /* Config disk & user disks will also be attached if necessary,
                               & if an error occurs, the default OS is attached */
          END;
    END Rqst$OS;
```

```
/* ------------------------------------------------------------------- */
/* SERVER PROCEDURES - PERFORM OPENING & ATTACHING OF DISKS TO DRIVES */


/*PROCEDURE*/ Zero$Current$Settings: Procedure;
   /*BEGIN*/
      User$Connected        : False;
      Current.User$Num      : Non$User;
      Current.Access$Config : No$Access;
      Current.Max$Qty       : 0;
      Current.OS$Num        : Def$OS;
      Current.User$File$Num : 8000h;
      Current.Dir$Connect   : Non$Token;
    END Zero$Current$Settings;



/*PROCEDURE Common$Disk$Open - is defined external to the server process */



/*PROCEDURE */ User$Disk$Open: Procedure (File$Num, Read$Write, Ops$Disk$Num$Ptr, Status$Ptr);
                               Declare  File$Num          Type$File$Num,
                                        Read$Write        Boolean,
                                        Ops$Disk$Num$Ptr  Pointer,
                                        Status$Ptr        Pointer;
   /*VARS*/
      Declare
        Ops$Disk$Num   BASED Ops$Disk$Num$Ptr  Type$Ops$Disk$Num,
        Status         BASED Status$Ptr        Status$Type;


   /*PROCEDURE */ Mark$Open$And$Wait: Procedure;

      /* Mark the disk entry open & wait until the file is open, & the
          token is available for use. */

      /*CONST*/
         Declare  Time$Len0  literally    '0';

      /*BEGIN*/
         CALL Ops$Table_Marks$Open (Ops$Disk$Num, 1, Read$Write, Status$Ptr);
         IF (Status = Successful)
           THEN
             DO;
               Status = NOT (Ready);
               DO WHILE (Status <> Ready);
                  CALL Ops$Table_Disks$Ready (Ops$Disk$Num, 1, Status$Ptr);
                  IF (Status <> Ready)
                    THEN
                      DO;
                        /* Allow other tasks to continue */
                        CALL RQ$SEND$CONTROL (@Exception); /* Release Ops$Table Resource */
                        CALL RQ$SLEEP (Time$Len0, @Exception);
                        CALL RQ$RECEIVE$CONTROL (Ops$Table$Region, @Exception);
                      END;
```

```
            END; /* While - Files are open for use */
          END;
      END Mark$Open$And$Wait;


/*BEGIN*/
    CALL RQ$RECEIVE$CONTROL (Op$Table$Region, @Exception);
    CALL Op$Table_Search$Disk (File$Num, Op$Disk$Num$Ptr, Status$Ptr);
    IF (Status : Found)

      THEN /* Disk has already been used */
        DO;
          CALL Mark$Open$And$Wait;
          CALL RQ$SEND$CONTROL (@Exception); /* Release Op$Table Resource */
        END;

      ELSE /* Disk has not yet been used */
        DO;
          /* Create Entry in Op$Table Resource & Mark it as open */
          CALL Op$Table_Attach$Disks (File$Num, 1, User$Appl$Num, Op$Disk$Num$Ptr, Status$Ptr);
          IF (Status : Successful)
            THEN
              DO;
                CALL Op$Table_Mark$Open (Op$Disk$Num, 1, Read$Write, Status$Ptr);
                IF (Status <> Successful)
                  THEN /* Restore Op$Table */
                    CALL Op$Table_Remove$Disks (Op$Disk$Num, 1);
              END;
          CALL RQ$SEND$CONTROL (@Exception); /* Release Op$Table Resource */

          /* Open the files on the Hard Disk */
          IF (Status : Successful)
            THEN
              DO;
                /* Attach & Open files on the Hard Disk */
                CALL RQ$RECEIVE$CONTROL (HD$Attach$Region, @Exception);
                  CALL HD$Attach_Open$Create$File ( (File$Num MOD 16), Current.Dir$Connect,
                            Read$Write, @Op$Disk$Map(Op$Disk$Num).Open$Connection, @Exception);
                  IF (Exception <> E$OK)
                    THEN Status : Disk$Attach$Error;
                CALL RQ$SEND$CONTROL (@Exception); /* Release HD$Attach Resource */

                /* If an error occured then restore the Op$Table Resource */
                IF (Status <> Successful)
                  THEN
                    DO;
                      CALL RQ$RECEIVE$CONTROL (Op$Table$Region, @Exception);
                        CALL Op$Table_Mark$Closed (Op$Disk$Num);
                      CALL RQ$SEND$CONTROL (@Exception);
                    END;
              END; /* Then */
        END; /* Disk Entry Created */

    END User$Disk$Open;



/*PROCEDURE */ Common$Disk$Attach: Procedure (File$Num, Load$Type, Read$Write, OS$Num, Access$Config,
                                                                  Qty$Ptr, Drive$1$Ptr, Status$Ptr);
```

```
                                Declare  File$Num      Type$File$Num,
                                         Load$Type    Boolean,
                                         Read$Write   Boolean,
                                         OS$Num       Byte,
                                         Access$Config Byte,
                                         Qty$Ptr      Pointer,
                                         Drive$1$Ptr  Pointer,
                                         Status$Ptr   Pointer;
    /*VARs*/
       Declare
         Qty           BASED Qty$Ptr      Byte,
         Drive$1       BASED Drive$1$Ptr  Drive$Type,
         Status        BASED Status$Ptr   Status$Type;
       Declare
         Op$Disk$Num1  Type$Op$Disk$Num,
         Count         Type$Op$Disk$Num;

    /*BEGIN*/
       CALL Common$Disk$Open (File$Num, Load$Type, Read$Write, OS$Num, Access$Config,
                                                  @Qty, @Op$Disk$Num1, @Status);
       IF (Status = Successful)
         THEN
           DO;
             CALL Drive$Map_Attach (Op$Disk$Num1, Qty, File$Num, @Drive$1, @Status);
             IF (Status <> Successful)
               THEN
                 DO;
                   CALL RQ$RECEIVE$CONTROL (Op$Table$Region, @Exception);
                     DO Count = Op$Disk$Num1 TO (Op$Disk$Num1 + Qty);
                       CALL Op$Table_Mark$Closed (Count);
                     END;
                   CALL RQ$SEND$CONTROL (@Exception);
                 END; /* Then */
           END; /* Then */
    END Common$Disk$Attach;



/*PROCEDURE */ User$Disk$Attach: Procedure (Disk, Read$Write, Drive$Ptr, Qty$Done$Ptr, Status$Ptr);
                                Declare  Disk          Type$File$Num,
                                         Read$Write    Boolean,
                                         Drive$Ptr     Pointer,
                                         Qty$Done$Ptr  Pointer,
                                         Status$Ptr    Pointer;
    /*VARs*/
       Declare
         Drive         BASED Drive$Ptr     Drive$Type,
         Qty$Done      BASED Qty$Done$Ptr  Byte,
         Status        BASED Status$Ptr    Status$Type;
       Declare
         User$File     Type$File$Num,
         Qty           Byte,
         Op$Disk$Num   Type$Op$Disk$Num;

    /*BEGIN*/
       Qty$Done = 0;

       IF (Disk > Current.Max$Qty)
         THEN Status = Disk$Not$Exist;
         ELSE
```

```
            DO;
              Status = Successful;

              /* Set up loop for either a single disk or for all user disks */
              IF (Disk = 0)
                THEN /* Request is for all user disks not already connected */
                  DO;
                    User$File = Current.User$File$Num + 1;
                    Qty = Current.Max$Qty;
                  END;
                ELSE /* Request is for a specific disk */
                  DO;
                    User$File  = Current.User$File$Num + Disk;
                    Qty = 1;
                  END;

              DO User$File = User$File TO (User$File + Qty - 1);

                /* Attach entry to Op$Table & Open associated Disk file */
                CALL User$Disk$Open (User$File, Read$Write, @Op$Disk$Num, Status$Ptr);

                /* Attach to an available drive */
                IF (Status = Successful)
                  THEN
                    DO;
                      CALL Drive$Map_Attach (Op$Disk$Num, 1, User$File, @Drive, @Status);
                      IF (Status <> Successful)
                        THEN /* Restore Op$Table, no drives free */
                          DO;
                            CALL RQ$RECEIVE$CONTROL (Op$Table$Region, @Exception);
                            CALL Op$Table_Ma...closed (Op$Disk$Num);
                            CALL RQ$SEND$CONTROL (@Exception);
                          END;
                    END; /* Then */

                IF (Status = Successful)
                  THEN Qty$Done = Qty$Done + 1;
              END; /* Do Loop */

          END; /* Else */
        END User$Disk$Attach;



/*PROCEDURE */ Attach$OS: Procedure;
    /*VARS*/
        Declare
          Qty               Byte,
          First$Drive       Drive$Type,
          Status            Status$Type,
          U$Dir$Name (6)    Byte,          /* Name for the user directory */
          U$Num$String (5)  Byte;          /* 4-Digit String composed from the User Number */

    /*BEGIN*/
        IF (Current.Dir$Connect <> Non$Token)
          THEN CALL RQ$A$DELETE$CONNECTION (Current.Dir$Connect, No$Mailbox, @Exception);

        /* Detach ALL disks connected to drive */
        Perm$Drive$Limit = 0;
        CALL Drive$Map_Detach (0, Max$Drives);
```

```
/* Attach the required Operating System */
IF (Current.OS$Num > Large$OS)
  THEN Current.OS$Num = Def$OS;
CALL Common$Disk$Attach ( OS$Array(Current.OS$Num).Appl$Num, Immediate, Read, Current.OS$Num,
                                    Current.Access$Config, @Qty, @First$Drive, @Status);

/* If an error occured attach the Default Operating System */
IF (Status <> Successful)
  THEN
    DO;
      Current.OS$Num = Def$OS;
      CALL Common$Disk$Attach ( OS$Array(Current.OS$Num).Appl$Num, Immediate, Read,
                                Current.OS$Num, Carte$Blanche, @Qty, @First$Drive, @Status);
    END;

CALL Drive$Map_Set$Perm;
IF (User$Connected)
  THEN
    DO;
      /* Attach the disk of application information to the PC */
      CALL Common$Disk$Attach ( (Access$Appl + Current.Access$Config), Immediate, Read,
                               Current.OS$Num, Carte$Blanche, @Qty, @First$Drive, @Status);
      CALL Drive$Map_Set$Perm;
      IF (Current.Max$Qty > 0)
        THEN
          DO;
            /* User directory name : U + a 4-digit decimal (formed from the user Num) */
            U$Dir$Name(0) = Empty$String;
            CALL String_Add ( @U$Dir$Name, @('U'), 5);
            CALL String_Num (Current.User$Num, @U$Num$String, 4);
            CALL String_Add ( @U$Dir$Name, @U$Num$String, 5);

            /* Attach or create this directory */
            CALL RQ$RBCEIVE$CONTROL (HD$Attach$Region, @Exception);
              CALL HD$Attach_Att$Create$Dir ( @OS$Array(Current.OS$Num).OS$Name, @U$Dir$Name,
                       @(Empty$String), 2, Users$Prefix, @Current.Dir$Connect, @Exception);
            CALL RQ$SEND$CONTROL (@Exception);

            /* Attach user disks for write access. 0 specifies all disks to be loaded */
            CALL User$Disk$Attach (0, Write, @First$Drive, @Qty, @Status);
          END;
      END; /* Then */
END Attach$OS;
```

B.4.4 Disk-Read Process: *File:* DSKREAD.PRS

```
/* ------------------------------------------------------------------------------------------------------ */

/* PROCESS: DISK$READ */
   /* This Process obtains a read request from the Disk Read Mailbox, & reads the requ red
        data into the cache, one (PC)track at a time. As each track becomes available it
        marks it in the cache as present */

  /*PROCEDURE */ Disk$Read: Procedure;

      /*CONST*/
         Declare
           Wait$Done        literally    '0FFFFh';
           Set              literally    '2',
           File$Connection  literally    'Op$Disk$Map(Msg.Disk).Open$Connection',
           File$Pos         literally    '(Msg.PC_Track + Count) * Size$Cache$Track',
           Read$Pos         literally    'T$Ent(Msg.Track(Count)).Track$Pos';

      /*VAR*/
         /* Declared already in cache resource
             Read$Msg$Structure  literally
               'STRUCTURE ( Qty Byte, Disk D$Type, Group Group$Num, Track$A Type$PC_Track, Track(1) Track$Num )';
                                                                  Array Track is unbounded - cf: PLM-86 pg 8-2 */
         Declare
           Msg$Segment      Token,
           Msg              BASED Msg$Segment   Read$Msg$Structure;
         Declare
           Count            Byte,
           Response         Word,
           Bytes$Read       Word,
           Exception        IO$Exception;

      /*BEGIN*/
         DO Forever;
            Msg$Segment : RQ$RECEIVE$MESSAGE (Read$Msg$Mbox, Wait$Done, @Response, @Exception);
            DO Count : 0 TO (Msg.Qty - 1);
               CALL RQ$S$SEEK (File$Connection, Set, File$Pos, @Exception);
               Bytes$Read : RQ$S$READ$MOVE (File$Connection, Read$Pos, Size$2Tracks, @Exception);
               CALL RQ$RECEIVE$CONTROL ( Cache$Region, @Exception);
               CALL Cache_Mark$Present (Msg.Track(Count), Msg.Group);
               CALL RQ$SEND$CONTROL (@Exception);
            END; /* Do count */
            CALL RQ$DELETE$SEGMENT (Msg$Segment, @Exception);
         END; /* Forever Loop */
       END Disk$Read;

/* END DISK$READ PROCESS */

/* ------------------------------------------------------------------------------------------------------ */
```

B.4.5 Disk-Write Process - *File*: DSI$WRITE.PRS

```
/* ------------------------------------------------------------------------------------------ */

/* PROCESS: DISI$WRITE */
   /* This process waits until disk writing is required. It then loops, each time obtaining
      from the cache the next item to be written, together with its dirty (PC)sectors
      (Note - the PC sector gives the offset for the file image for that disk). The data
      is flushed to disk. */

   /*PROCEDURE */ Disk$Write: Procedure;

      /*CONST*/
         Declare
           Wait$Done        literally   'OFFFFh',
           Time$Len         literally   '0',   /* Allows other tasks to become active */
           Set              literally   '2',
           File$Connection  literally   'Op$Disk$Map(Disk).Open$Connection',
           File$Pos         literally   '(PC_Track * Size$Cache$Track) + (PC_Sect * Sect$Size)',
           Sect$Pos         literally   'T$Ent(Track).Track$Pos + (PC_Sect * Sect$Size)';

      /*VAR*/
         Declare
           Disk             D$type,
           Group            Group$Num,
           PC_Track         Type$PC_Track,
           Track            Track$Num,
           Sect$Bits (3)    Byte,
           Exception        IO$Exception;

      /*PROCEDURE */ Sect$Dirty: Procedure (Sector)  Boolean;
                               Declare  Sector  Type$PC_Sect;
         /*VAR*/
            Declare
              (Req$Byte, Req$Bit)       Byte;
         /*BEGIN*/
            Req$Byte = Sector/8;
            Req$Bit  = Sector MOD 8;
            Req$Bit  = SHL ( 0000$0001B, Req$Bit);
            RETURN (Sect$Bits(Req$Byte) AND Req$Bit);
          END Sect$Dirty;

      /*BEGIN*/
         DO Forever;
           CALL RQ$RECEIVE$CONTROL ( Cache$Region, @Exception);
           DO WHILE (NOT Cache_WL$Ready);
              CALL RQ$SEND$CONTROL (@Exception);
              CALL RQ$SLEEP (Time$Len, @Exception);
              CALL RQ$RECEIVE$CONTROL ( Cache$Region, @Exception);
           END; /* While */
           Disk  = TGL(Group).Disk;
           Track = Non$Track;
           Group = Non$Group;
           DO WHILE ( Cache_Next$Write (Disk, @Group, @PC_Track, @Track, @Sect$Bits) );
              CALL RQ$SEND$CONTROL (@Exception);
              DO PC_Sect = 0 TO (Sect$In$Cache$Track - 1);
                 IF Sect$Dirty(PC_Sect)
                    THEN DO;
                          CALL RQ$S$SEEK (File$Connection, Set, File$Pos, @Exception);
```

```
                    #Bytes$Written : RQ$S$WRITE$MOVE (File$Connection, Sect$Pos, Sect$$Size, @Exception);
                  END; /* Then */
              END; /* Do PC_Sect */
              CALL RQ$RECEIVE$CONTROL ( Cache$Region, @Exception);
            END; /* While */
            CALL RQ$SEND$CONTROL (@Exception);
          END; /* Forever Loop */
        END Disk$Write;

/* END SERVER PROCESS */

/* ------------------------------------------------------------------------------------------ */
```

B.4.5 Track-Send Completion Process - File: CPLSENDS.PRS

```
/* ------------------------------------------------------------------------------------------ */

/* PROCESS: COMPLETE$SENDS*/

  /* This process involves scanning the output queue resource for responses no longer required by the
     network processor, & performing outstanding operations on the Server's resources. For Read
     responses, the cache must hold the tracks at least until they have been transmitted, and so
     must be informed when such operation is complete. */


  /*PROCEDURE */ Complete$Sends: Procedure;

    /*CONST*/
       Declare
         Time$Len        literally   '0';  /* Allows other tasks to become active */
    /*VAR*/
       Declare
         Msg$Ptr         Pointer,
         Fn              BASED Msg$Ptr     Byte,
         Exception       IO$Exception;

    /*BEGIN*/
       DO Forever;
         /* Wait until the Network Processor has finished sending and processing message */
         CALL RQ$RECEIVE$CONTROL ( NW$Out$Region, @Exception);
         DO WHILE ( NOT (NW$Out_Done (@Msg$Ptr)) );
           CALL RQ$SEND$CONTROL (@Exception);
           CALL RQ$SLEEP (Time$Len, @Exception);
           CALL RQ$RECEIVE$CONTROL ( NW$Out$Region, @Exception);
         END; /* While */

         /* Perform actions required for each message type */
         CALL RQ$SEND$CONTROL (@Exception);
         DO CASE Fn;
           /* 0 Read */          DO;
                                   Declare
                                     Msg  BASED Msg$Ptr   STRUCTURE  (Fn      Byte,
                                                                      Group   Group$Num);
                                   CALL RQ$RECEIVE$CONTROL ( Cache$Region, @Exception);
                                   CALL Cache_Reduce$Ops (Msg.Group);
                                   CALL RQ$SEND$CONTROL (@Exception);
                                 END;
           /* All Others */      /* DO Nothing */
         END; /* Case */

         /* Release the message entry of the output queue resource, for re-use */
         CALL RQ$RECEIVE$CONTROL ( NW$Out$Region, @Exception);
         CALL NW$Out_Release;
         CALL RQ$SEND$CONTROL (@Exception);
       END; /* Forever Loop */
     END Complete$Sends;

/* END COMPLETE$SENDS PROCESS */

/* ------------------------------------------------------------------------------------------ */
```

B.5 SUB-PROCESSES COMMON TO 2ND LEVEL PROCESSES

These are processes which form sub-processes for more than one of the main (2nd-level) processes. There is only one such common sub-process - the *common-disk open process* - which is performed by the initialization and node-server processes. It executes operations on the hard disk, 'common-info' and 'opened-disks tables' resources, necessary to open a common or applications disk.

B.5 Disk-Open Process - *File*: COM$OPEN.RTN

```
/* ------------------------------------------------------------------------------------------------ */

/* PROCESS: COMMON$DISK$OPEN */

/* Opening of Common disks - Used during initialization (to ensure that well used files
    are permanently open) and by the server tasks. Accesses the following resources:
    Op$Table, Common$Info, HD$Attach and the Cache. If at any stage the status is not
    successful, the resources must be restored to their original conditions */


   /*PROCEDURE */ Common$Disk$Open: Procedure ( Rqst$Disk, Reqd$Load, Read$Write, Caller$OS,
                                                Access$Config, Qty$Ptr, First$D$Num$Ptr, Status$Ptr )  REENTRANT;
                                      Declare  Rqst$Disk          Type$File$Num,
                                               Reqd$Load          Boolean,
                                               Read$Write         Boolean,
                                               Caller$OS          Byte,
                                               Access$Config      Byte,
                                               Qty$Ptr            Pointer,
                                               First$D$Num$Ptr    Pointer,
                                               Status$Ptr         Pointer;

    /*CONST*/
       Declare  New$Appl   literally   'Current$Common$Entry';
    /*VAR*/
       Declare
         Qty              BASED Qty$Ptr          Byte,
         First$D$Num      BASED First$D$Num$Ptr  Type$Op$Disk$Num,
         Status           BASED Status$Ptr       Status$Type;
       Declare
         Rqst$Appl        Type$File$Num,
         Whole$Appl$Rqst  Boolean,
         Op$Appl$Num      Type$Op$Appl$Num,
         Reqd$Op$Appl$Ptr Pointer,
         Reqd$Appl        BASED Reqd$Op$Appl$Ptr    Op$Appl$Entry,
         Exception        IO$Exception,
         Count            Byte;


    /*PROCEDURE */ Query$Access: Procedure (Access$Ptr, Load$Type, OS$Num);
                                  Declare   Access$Ptr      Pointer,
                                  Load$Type       Boolean,
                                  OS$Num          Byte;

       /* Determines access to applications by examining the bits in the Access$Bytes string,
```

```
        which refer to the caller's access configuration. The first bit of the pair is for
        read & the second for write access. The callers Operating System & the type of
        load requested are also compared with those set for the application */


    /*VAR*/
       Declare
         Access      BASED Access$Ptr    Access$Bytes,
         Reqd$Byte         Byte,
         Reqd$Bit          Byte;

    /*BEGIN*/
       Status = Granted;
       /* The rightmost pair of bits in the access bytes refers to configuration-2 & so on right to
            left. Config-1 (Carte Blanche) is allowed both read & write access on all applications
            & requires no access bytes. Config-2 can be used for unofficial users. */

       IF (Access$Config <> Carte$Blanche)
         THEN
           DO;
             /* Examine the access bits for the users configuration */
             Reqd$Byte = Access ( (Access$Config - 2) / 4);
             Reqd$Bit  = ( (Access$Config - 2) MOD 4) * 2;
             IF (Read$Write = Write)
               THEN Reqd$Bit = Reqd$Bit + 1;
             IF ( (SHR(Reqd$Byte,Reqd$Bit) AND (0001B) ) <> 1 )
               THEN  IF (Read$Write = Read)
                       THEN Status = Read$Access$Denied;
                       ELSE Status = Write$Access$Denied;

             /* If the load$type for an appl is 'immediate' (protected), it
                 cannot be loaded using the 'flat' load by an ordinary user */
             IF (Status = Granted) THEN
               IF ((Reqd$Load <> Load$Type) AND (Load$Type = Immediate))
                 THEN Status = Wrong$Loader;

             /* Operating systems of user & the appl must match */
             IF (Status = Granted) THEN
               IF ( (OS$Num <> Any$OS) AND (OS$Num <> Caller$OS) )
                 THEN Status = Wrong$OS;

           END; /* Then */
      END Query$Access;



/*PROCEDURE */ Mark$Open$And$Wait: Procedure (Qty);
                                  Declare  Qty  Byte;

     /* Mark the disk entries open & wait until all the files
          are open, & their tokens are available for use. */

     /*CONST*/
        Declare  Time$Len    literally   '0';

     /*BEGIN*/
        CALL Ops$Table_Mark$Open (First$Ds$Num, Qty, Read$Write, Status$Ptr);
        IF (Status = Successful)
          THEN
            DO;
```

Author Hildyard Mark William
Name of thesis Design Of A Disk-based File Server Involving The Mapping Of "pseudo-disks". 1989

PUBLISHER:

University of the Witwatersrand, Johannesburg
©2013

LEGAL NOTICES:

Copyright Notice: All materials on the University of the Witwatersrand, Johannesburg Library website are protected by South African copyright law and may not be distributed, transmitted, displayed, or otherwise published in any format, without the prior written permission of the copyright owner.

Disclaimer and Terms of Use: Provided that you maintain all copyright and other notices contained therein, you may download material (one machine readable copy and one print copy per page) for your personal and/or educational non-commercial use only.

The University of the Witwatersrand, Johannesburg, is not responsible for any errors or omissions and excludes any and all liability for any errors in or omissions from the information on the Library website.