```
/* di.c RHS 7/15/89
*
*/


#define      INCL_DOS
#define      INCL_ERRORS

#include<os2.h>
#include<mt\stdio.h>
#include<mt\string.h>
#include<mt\stdlib.h>
#include<mt\ctype.h>
#include"errexit.h"

#define DICODE
#include"di.h"

#define      lastchar(str)                (str[strlen(str)-1])

void   diInit(PID *qowner, HQUEUE *qhandle);
void   adddriveletter(PCH *filespecs,USHORT driveno);
void     makefpath(char *org,char*result,char*currentpath,
                 USHORT currentdrive);
void     getdriveinfo(USHORT*currentdrive,char*currentpath,
                              USHORT *psize);
USHORT diallocseg(USHORT size,SEL*oursel,PID other,SEL*othersel):
void   convertptr(VOID **ptr,SEL newsel);

        /* DiMakeRequest
        This function creates or adds to an existing directory information
        request. A new request is created if the pointer (hptr) is set to              NULL. Note
that this is the pointer whose address is passed to the                function.     The
filespec is a full path and file specification with          optional wildcards. The attribute
parameter will control the files                that are found. */

void DiMakeRequest(PREQUESTHEADER *hptr, PCH filespec, USHORT att)
        {
  PREQUESTHEADER     header;
  PDIRINFORESULT       resultstru;
        SEL                     hselector,serversel;
        void far                *results;
        USHORT                  retval,size,psize = 0,setdir = FALSE;
        HQUEUE                  qhandle;
        PID                     qowner;
        PCH                     requestspec;
```

```
    char                    resultbuf[_MAX_DIR],*resultfspec;
    char                    currentpath[_MAX_PATH];
    USHORT                  currentdrive;

if(!(header = *hptr))
     DiInit(&qowner, &qhandle);

    /* Now gather the information to prepare the header:
     o      Get the current disk drive
     o      Get the length of the current directory path
     o      Get the size of the request arguments (the requestspec)
     o      Allocate the header segment
     o      Make it giveable to the server process */

getdriveinfo(&currentdrive,currentpath,&psize);

    if(!header)                             /* If no header is  alloc'd */
      {
                                                /* Allocate header segment */
    if(retval = diallocseg(size = sizeof(REQUESTHEADER),&hselector,
                           qowner,&serversel))
     error_exit(retval,"diallocseg");

      header = MAKEP(hselector,0);        /* Make header pointer */
      *hptr = header;
      header->serverhsel = serversel;   /* Allocate work segment */
    if(retval = diallocseg(MAXSEGSIZE,&header->rselector,qowner,
                                &serversel))
     error_exit(retval,"diallocseg");

     results = MAKEP(header->rselector,0); /* Make pointer */

     header->RAMsem = 0L;                    /* Initialize semaphore */
header->resultptr = MAKEP(serversel,0); /* Set pointer to work area */
     header->serverwsel = serversel;        /* Selector to work area*/
     header->numRequests = 0;               /* Set number of requests */

    requestspec = (PCH)results;             /* Set to work area */
     header->currentdir = requestspec;      /* Set pointer to it */
     *requestspec++ = (char)(currentdrive + 'A' - 1);
                                                /* Add drive letter*/
      strcpy(requestspec,":\\");            /* And ':\'*/
      requestspec += 2;                     /* Move pointer past them */

    DosQCurDir(currentdrive,requestspec,&psize);
                                                /* Add current directory */
```

```
  requestspec += (strlen(requestspec)+1);   /* Move pointer past dir*/
   header->qowner = qowner;
   header->qhandle = qhandle;
        }

   else
        {
   requestspec = header->requestspec;
   qowner = header->qowner;
   qhandle = header->qhandle;

   size = header->size + sizeof(DIRINFORESULT);
   hselector = SELECTOROF(header);

   if(retval = DosReallocSeg(size,hselector)) /* Resize segment */
        error_exit(retval,"DosReallocSeg");
        }

         /* Resultstru always points to the next available structure */
resultstru = &header->resultArray[header->numRequests];
   strupr(filespec);                          /* Set arg to upper case */
memset(resultstru,0,sizeof(DIRINFORESULT)); /* Clear structure */
   resultstru->attributes = att;                    /* Set attributes */
                                                    /* Get full path filespec */
makefpath(filespec,resultbuf,currentpath,currentdrive);
 resultfspec = strrchr(resultbuf,'\\');  /* Find last backslash */

if(strcmp(resultbuf,header->currentdir))  /* If not in currentdir */
   {
  strcpy(requestspec,resultbuf);                /* Copy path */
 resultstru->currentdir = requestspec;       /* Set pointer */
 requestspec += (strlen(requestspec)+1);
                                                  /* Set pointer to next spot */
   }
 else                                           /* Use default directory */
  resultstru->currentdir = header->currentdir; /* Set pointer */

strcpy((char *)requestspec,resultfspec);  /* Copy the filespec */
resultstru->filespec = requestspec;   /* Set a pointer to it */
requestspec += (strlen(requestspec)+1);   /* set to next position */

header->size = size;
header->numRequests++;
header->requestspec = requestspec;
 }
```

```
void DiSendRequest(PREQUESTHEADER hdr)
        {
        PCH *resptr,*newptr;
        USHORT      offset,retval,i;
        SEL    newsel,serverwsel = SELECTOROF(hdr->resultptr);
        PBYTE sheader = MAKEP(hdr->serverhsel,0);   /* Make pointer */

                /* Adjust resultptr to point to available space   */
        resptr = MAKEP(hdr->rselector,0);              /* Create ptr to result */
        offset = (hdr->requestspec - (PCH)resptr);         /* Get offset to use */
        hdr->resultptr = MAKEP(serverwsel,offset);       /* Reset pointer */

                /* Write request to the queue */
        if(retval = DosWriteQueue(hdr->qhandle,0,hdr->size,(PBYTE)sheader,0))
          error_exit(retval,"DosWriteQueue");  /* Wait for server to finish */

        DosSemSetWait(&hdr->RAMsem,SEM_INDEFINITE_WAIT); /* Get new segment */
        if(retval = DosAllocSeg(hdr->resultsize,&newsel,SEG_NONSHARED))
          error_exit(retval,"DosAllocSeg");

         newptr = MAKEP(newsel,0);                    /* Make pointer to segment */
        memmove(newptr,resptr,hdr->resultsize);           /* Copy from old segment */
        convertptr(&hdr->currentdir,newsel);        /* Convert pointers */
          convertptr(&hdr->requestspec,newsel);

        for( i = 0; i < hdr->numRequests; i++)
           {
        convertptr(&hdr->resultArray[i].filespec,newsel);
        if(SELECTOROF(hdr->resultArray[i].currentdir) == hdr->rselector)
        convertptr(&hdr->resultArray[i].currentdir,newsel);
        convertptr(&hdr->resultArray[i].firstfile,newsel);
        convertptr(&hdr->resultArray[i].nextfile,newsel);
           }

        DosFreeSeg(hdr->rselector);           /* Free the old segment */
        hdr->rselector = newsel;                      /* Set for new selector */
         }

void DiGetNumResults(PREQUESTHEADER header,USHORT *numresults,
                             USHORT *numrequests)
        {
   *numresults = header->totalresults;
   *numrequests = header->numRequests;
        }
```

```
void DiDestroyRequest(PREQUESTHEADER *header)
         {
        USHORT retval;
        PREQUESTHEADER hdr = *header;

        if(retval = DosFreeSeg(hdr->rselector))         /* Free work segment */
          error_exit(retval,"DosFreeSeg");
        if(retval = DosFreeSeg(SELECTOROF(hdr)))        /* Free header segment */
          error_exit(retval,"DosFreeSeg");

        *header = NULL;                                 /* Set pointer to NULL *\
         }

char *DiGetResultFspec(PDIRINFORESULT result)
        {
   static char *p = "*.*";

   if(result->errorval == DIREXPANDED)
                return p;
   return result->currentdir;
   return result->filespec;
        }

char *DiGetResultDir(PDIRINFORESULT result)
        {
   static char dirbuf[80];

   if(result->errorval == DIREXPANDED)
                {
         strcpy(dirbuf,result->currentdir);
     strcat(dirbuf,"\\");
         strcat(dirbuf,result->filespec);
         return dirbuf;
                }

   return result->currentdir;
        }

void DiGetResultHdl(PREQUESTHEADER header,USHORT requestnum,USHORT *num,
PDIRINFORESULT *resulthdl)
        {
  *resulthdl = &header->resultArray[requestnum];
  *num = (*resulthdl)->numfound;
        }

void DiGetFirstResult(PDIRINFORESULT result, char *buffer)
```

```
        {
  result->nextfile = result->firstfile;
  DiGetNextResult(result,buffer);
        }

void DiGetNextResult(PDIRINFORESULT result, char *buffer)
        {
  if(!result->nextfile)
          {
         *buffer = '\0';
         return;
          }

  strcpy(buffer,result->nextfile->achName);
  result->nextfile =
  (PFILEFINDBUF)(&(result->nextfile->cchName)+
                                result->nextfile->cchName+2);
        }

void DiGetFirstResultPtr(PDIRINFORESULT result, PFILEFINDBUF *ptr)
        {
   result->nextfile = result->firstfile;
   DiGetNextResultPtr(result,ptr);
        }

void DiGetNextResultPtr(PDIRINFORESULT result, PFILEFINDBUF *ptr)
        {
   if(!result->nextfile)
            {
          *ptr = NULL;
          return;
           }

        *ptr = result->nextfile;
        result->nextfile =
        (PFILEFINDBUF)(&(result->nextfile->cchName)+
                                result->nextfile->cchName+2);
        }

void DiBuildResultTbl(PREQUESTHEADER header, PFILEFINDBUF **table)
        {
   SEL tablesel;
   USHORT retval, i;
   PFILEFINDBUF f,*temp;

  if (retval = DosAllocSeg((header->totalresults*sizeof(PFILEFINDBUF)),
```

```
                                        &tablesel, SEG_NONSHARED))
        error_exit(retval,"DosAllocSeg");

        temp = MAKEP(tablesel,0);
        f = header->resultArray[0].firstfile;

        for( i = 0; i < header->totalresults; i++)
                {
          temp[i] = f;
          f = (PFILEFINDBUF) (&(f->cchName)+f->cchName+2);
                }

       *table = temp;
        }

void diInit(PID *qowner, HQUEUE *qhandle)
        {
  USHORT   retval;

/* Try to open the queue to the directory server */
  if(!(retval = DosOpenQueue(qowner,qhandle,DIRINFOQNAME)))
        return;
  if(retval != ERROR_QUE_NAME_NOT_EXIST)
        error_exit(retval,"DosOpenQueue");
        else
        error_exit(retval,
                        "DosOpenQueue - Server probably hasn't opened queue");
        }

void convertptr(VOID**ptr,SEL newsel)
        {
  USHORT offset = OFFSETOF(*ptr);
  *ptr = MAKEP(newsel,offset);
        }

/* Assumes that the globals, currentpath and currentdrive are properly
  initialized */
void makefpath(char *org,char *result,char *currentpath,
                        USHORT currentdrive)
        {
  char drive[_MAX_DRIVE], dir[_MAX_DIR], fname[_MAX_FNAME],
            ext[_MAX_EXT];
  char currdir[_MAX_DIR], *backup, *outdir;
  USHORT driveno,cdirsize = _MAX_DIR-1, retval;
  strupr(org);                                  /* uppercase the path */
  _splitpath(org,drive,dir,fname,ext);          /* get path components */
```

```
  /* If we have a full path from the user at this point, we don't have to do
anything more-whatever they ask for, they get. If we don't have a full path, we will
need to get the fullpath to the current directory of that drive, and then
reconcile the working directory or
         parent directory to end up with the "real" path to the file. However,
         if the path is on the same drive, we can use currentdir and save the the
time of a DosQCurDir call. */

    if(!(*drive))                              /* If no drive letter */
          {
         driveno = currentdrive;
         *drive = (char)(currentdrive+'A'-1);
         strcpy(&drive[1],":");
          }
        else
         driveno = (*drive-'A'+1);

        if(*dir != '\\')                       /* If not a full path */
          {
         if(driveno != currentdrive)
               {
           *currdir = '\\';
         if(retval = DosQCurDir(driveno,&currdir[1],&cdirsize))
               error_exit(retval,
                          "DosQCurDir - probably invalid drive or directory");
           }

         else
               strcpy(currdir,&currentpath[2]);

        if(lastchar(currdir) != '\\')
          strcat(currdir,"\\");

        strcat(currdir,dir);
        strcpy(dir,currdir);
      }

    /* dir now has full path to the filespec, reconcile and restore */
   while(backup = strstr(dir,"\\..\\"))  /* Remove any "\..\" */
          {
    for(outdir = backup-1; (*outdir != '\\') && (outdir > dir);
               outdir--);
                                                        /* Now outdir is '\' dest */
         backup += 3;       /* now backup is on the source '\'    */
         strcpy(outdir,backup);
```

```
        }

  while(backup = strstr(dir,"\\.\\"))           /* Remove any "\.\" */
        {
        outdir = backup;
        backup += 2;
        strcpy(outdir,backup);
        }

       _makepath(result,drive,dir,fname,ext);     /* Put it all back togeth */
       }

void getdriveinfo(USHORT *currentdrive,char *currentpath,USHORT *psize)
       {
       ULONG       drivemap;
       USHORT    retval;

       DosQCurDisk(currentdrive,&drivemap);    /* Get current drive number */
       *currentpath = (char)(*currentdrive+'A'-1);
       strcpy(&currentpath[1],":\\");

       *psize = _MAX_PATH;
       if(retval = DosQCurDir(*currentdrive,&currentpath[3],psize))
         error_exit(retval,"DosQCurDir");

       DosQCurDisk(currentdrive,&drivemap);    /* Get current drive number */
       *psize = 0;
       DosQCurDir(*currentdrive,NULL,psize);     /* Get size of currdir path */
       }

USHORT diallocseg(USHORT size, SEL *oursel,PID other,SEL *othersel)
       {
  USHORT retval = 0;
                                                        /* And shareable by server */
  if(!(retval = DosAllocSeg(size,oursel,SEG_GIVEABLE)))
       retval = DosGiveSeg(*oursel,other,othersel);

  return retval;
       }
```