In the same fashion as above, the new service routine would probably be of the format shown below.

```
/* The new service routine */

void NewService (void)
{

    /* Pick up the request from the receiveWs queue */


    /* Process it and place the response in the sendWs buffer */

    /* Set the response type as well */
}
```

Do not forget to declare NewService in the "declaration of routines" section in DB.C.

4. Insert a line in the appropriate place in the LoadDatabaseFiles routine in DB.C that will copy the new database file to RAM. An example is shown below.

```
ccode += CopyFileToRAM (Name("AP_BATCH.DBF"),
                        (void far *)&g_apBatch,
                        &size);
```

In the same way, using the examples of the new service given above, the code to be inserted in the LoadDatabaseFiles routine would be:

```
ccode += CopyFileToRAM (Name("NEW_FILE.DBF"),
                        (void far *)&g_newFile,
                        &size);
```

This code should be inserted in the same place where all the other examples of the CopyFileToRAM function is to be found.

5. Insert a case statement in the ServiceTheRequest routine (DB.C). An existing case statement is shown below.

```
case MENU_RECORD:

    GetMenu();
    break;
```

The new case statement would be:

```
case NEW_DB_RECORD:

    NewService();
    break;
```

6. Compile DB.C.

> If the file does not have to be loaded into the memory, then you can ignore step four and possibly step two as well.

B4.2 Function Reference

This section presents information on the Library. Please refer to the documentation entitled "Information on the VM" for some details on the services available and the format of the communication buffers.

B4.2.1 CheckForCommand

Checks a specified environmental block to see whether there is enough space to add a new variable.

Syntax

```
int CheckForCommand(unsigned int pSeg, char *envVar, char *newVar);
```

pSeg	(Input) Address of environmental block.
envVar	(Input) Environmental variable that check is being performed for.
newVar	(Input) New value that the environmental variable should be set to.

Return Values

0	Success
1	Space in the environment cannot accomodate new environmental variable.

Remarks

This routine obtains the maximum size that the specified environmental block can be expanded to from the Memory Control Block for the environment and also the current size of the environmental block and thus determines whether there is enough space to insert a new variable.

B4.2.2 DBinit

DBinit initialises parameters necessary for communication with a VM database server.

Syntax

```
int  DBinit (int *dbSocket,
             DBTABLE_struct *table,
             int *numDBservers,
             SENDDB_struct *sendDb);
```

dbSocket	(Output) Returns the socket number that the VM database server uses to communicate with.
table	(Output) A table containing all available VM database servers upto a maximum of four.
numDBservers	(Output) Total number of VM database servers that have been located.
sendDb	(Input/Output) Sets the request number in the sendDB buffer to zero.

It is easiest and most convenient for programming purposes if all the parameters supplied to DBinit are global variables.

Return Values

0	Success
1	No VM database servers could be located.

Remarks

DBinit sets the socket number to communicate with the VM database server, locates all available VM database servers and initialises the number of requests for services to zero. Each time the PC requests for service, the request number is incremented.

This function uses the function LocateDBServer.

B4.2.3 DBRequest

Requests a particular service from a VM database server.

Syntax

```
BYTE DBRequest (RECEIVEDB_struct *receiveDb,
                SENDDB_struct *sendDb,
                BYTE *returnDBflag,
                DBTABLE_struct *table,
                int dbSocket,
                int dbTablepos,
                void (far *ESRAddress)(),
                IPXHeader *ipxSendDb,
                IPXHeader *ipxReceiveDb,
                ECB *ECBSendDb,
                ECB *ECBReceiveDb);
```

receiveDb	(Output) Points to receive buffer.
sendDb	(Input) Points to send buffer.
returnDBflag	(Output) Flag indicating DB responded.
table	(Input) Points to table of DB servers.
dbSocket	(Input) VM database server communication socket #.
dbTablepos	(Input) Position in table of DB servers.
ESRAddress	(Input) Address of ESR routine. This is an interrupt service routine which is activated whenever something is received from the VM database server.
ipxSendDb	(Input) The buffer containing the header for the packets sent to the VM database server.
ipxReceiveDb	(Input) The buffer containing the header for the packets received from the VM database server.
ECBSendDb	(Input/Output) The Event Control Block for packets sent to the VM database server.
ECBReceiveDb	(Input/Output) The Event Control Block for packets received from the VM database server.

It is easiest and most convenient for programming purposes if all the parameters supplied to DBRequest are global variables.

Return Values

0 Success

0xC1 could not set the send ECB

0xCC No response from the specified VM database server.

Other values dependent on the error generated by the specific request.

Remarks

This routine sends the request to a VM database server (specified by dbTablePos in the list of database servers) and waits for the response from the server. If a response is not received within a time-out period, the request is repeated until the maximum number of retries is reached. The information to be sent to the database is found in the buffer sendDb and the information received from the server is in the buffer receiveDb.

This function calls the following functions in this library:

- DiffFloatTime
- Set_EcbReceiveDb
- Set_EcbSendDb

B4.2.4 DiffFloatTime

Obtains the floating point difference between two times.

Syntax

```
float DiffFloatTime (struct time *last, struct time *first);
```

last	(Input) A structure of type time that contains the last value.
first	(Input) A stucture of type time that contains the first value where first ≤ last.

Return Values

Returns the difference between last and first.

Remarks

This routine is used to obtain the floating point difference between two times accurate to a hundredth of a second. This routine calls FloatTime.

B4.2.5 FloatTime

Converts a time to a floating point value in seconds.

Syntax

```
float FloatTime (struct time *start);
```

start (Input) A structure of type time that contains a valid time.

Return Values

Returns the floating point representation of the time in seconds (the fractional part is in hundredths of a second).

Remarks

This function is useful when a time that is accurate to a hundredth of a second is required.

B4.2.6 LocateDBServer

Locates available VM database servers

Syntax

```
BYTE LocateDBServer (DBTABLE_struct *table);
```

table (Output) Points to table of VM database servers.

Return Values

0 No VM database servers have been located.

Any other value represents the number of servers that have been located.

Remarks

This function locates and fills up a table of available VM database servers up to a maximum specified by MAX_DB_TABLE_POSITIONS. The address of the table is passed as a parameter to the function LocateDBServer.

B4.2.7 OpenSAC

Opens the SAC file for the current session

Syntax

```
FILE *OpenSAC (char *mode);
```

mode	(Input) Mode in which SAC file should be opened. The mode is further described in the C manual under the fopen function.

Return Values

If unable to open the SAC file, returns NULL.

On success returns a pointer (of type FILE) to the opened SAC file.

Remarks

This routine determines the name of the SAC file from the connection number to the primary file server and opens the file in the directory pointed to by the TEMP environmental variable according to the mode specified.

B4.2.8 Set_EcbReceiveDb

Fills up an event control block in preparation to receive a packet from the VM database server.

Syntax

```
void Set_EcbReceiveDb (ECB *ECBReceiveDb,
                       int dbSocket,
                       IPXHeader *ipxReceiveDb,
                       RECEIVEDB_struct *receiveDb,
                       void (far *ESRAddress)());
```

ECBReceiveDb	(Input/Output) The Event Control Block for packets received from the VM database server.
dbSocket	(Input) VM database server communication socket #.
ipxReceiveDb	(Input) The buffer containing the header for the packets received from the VM database server.
receiveDb	(Input) Points to receive buffer.
ESRAddress	(Input) Address of ESR routine. This is an interrupt service routine which is activated whenever something is received from the VM database server.

Return Values

None.

Remarks

This routine fills up the event control block for the packets received from the VM database server.

B4.2.9 Set_EcbSendDb

Fills up an event control block in preparation to send a packet to the VM database server.

Syntax

```
int  Set_EcbSendDb (ECB *ECBSendDb,
                    int dbSocket,
                    IPXHeader *ipxSendDb,
                    SENDDB_struct *SendDb,
                    DBTABLE_struct *table,
                    int dbTablepos);
```

ECBSendDb	(Input/Output) The Event Control Block for packets received from the VM database server.
dbSocket	(Input) VM database server communication socket #.
ipxSendDb	(Input) The buffer containing the header for the packets sent to the VM database server.
sendDb	(Input) Points to send buffer.
table	(Input) Points to a valid table of DB servers.
dbTablepos	(Input) Valid position in table of DB servers.

Return Values

0 Success.

1 Could not properly set the send ECB.

Remarks

This routine fills up the event control block for the packets sent to the VM database server.

B4.2.10 SetLevelEnv

Sets an environmental variable in a particular environmental block.

Syntax

```
void SetLevelEnv (unsigned int far *env, char *envVar, char *newVar);
```

env	(Input) Address of the environmental block that is to be modified.
envVar	(Input) Environmental variable that is to be modified.
newVar	(Input) The value that the modified environmental variable is to be set to.

Return Values

None.

Remarks

This routine sets an environmental variable envVar to a new value newVar. If envVar does not exist, then envVar is inserted into the environment. The environment that is modified is located at the address given by env.

B4.2.11 SetParentEnv

Sets an environmental variable in all environmental blocks up to the main parent.

Syntax

```
void SetParentEnv (char *envVar, char *newVar);
```

envVar	(Input) Environmental variable that is to be modified.
newVar	(Input) The value that the modified environmental variable is to be set to.

Return Values

None.

Remarks

This routine sets an environmental variable envVar to a new value newVar. If envVar does not exist, then envVar is inserted into the environment. All environments up to the main parent given in the SAC file is modified in this fashion. This function uses:

- SetLevelEnv
- CheckForCommand

B4.2.12 ScramblePassword

Converts a string into a string that is 125 characters long.

Syntax

```
void ScramblePassword (char *plainText, char *c);
```

plainText	(Input) Original string.
c	(Output) Encrypted string.

Return Values

None.

Remarks

This routine uses a mathematical transformation to convert a string of arbitrary length to a string that is 125 characters long. This routine is mostly used to convert plain text passwords for the VM group login names to a password that is recognised by the file servers.

B4.4 UVM.H

```
/*
```

File Name: UVM.H

```
This is the header file for the VM source programs.

S.J. Isaac

10th June 1993

*/

#define DB_SERVER_TYPE          0x9999
#define DB_SERVER_SOCKET        0x4848
#define SERVER_NAME_LENGTH      3
#define USER_NAME_LENGTH        9
#define MENU_NAME_LENGTH        20
#define APPLICATION_NAME_LENGTH 20

#define REQUEST_SIZE            20
#define RESPONSE_SIZE           500
#define MENU_ITEMS              20
#define MENU                    'M'
#define APPLICATION             'A'

#define USER_RECORD             1
#define MENU_RECORD             2
#define APPLICATION_RECORD      3
#define PASSWORD_RECORD         4
#define LOAD_DATABASE           5

#define STUDENT                 0x01
#define LECTURER                0x02
#define MANAGER                 0x80
#define SUPERUSER               0xFF

#define VM_GROUP_NAME_LENGTH    15
#define VM_USER_NAME_LENGTH     15
#define VM_PASSWORD_LENGTH      128
#define NUM_OF_ACCESS_GROUPS    9

#define STANDARD_BATCH_FILE     "H:\\SUVM.BAT"
#define SAC_PATH_ON_C_DRIVE     "C:\\TEMP\\"
#define SAC_PATH_ON_H_DRIVE     "H:\\"


// database

#define MAX_RETRY               3
#define MAX_DB_TABLE_POSITIONS  4


typedef struct
    {
    char primaryServer[SERVER_NAME_LENGTH];
    char userName[USER_NAME_LENGTH];
    char loginTime[7];
    char dAC[26];
    unsigned int parentPSPseg;
    } SAC_struct;


typedef struct
    {
    int    dummy1[11];
    unsigned int pspseg;                // psp segment
    int    dummy2[10];
    unsigned int envseg;                // environment segment
    } PSP_struct;
```

```
typedef struct
    {
    char    end;                                // 0x4d==notend
    unsigned int pspseg;                        // psp segment
    unsigned int size;                          // size of controlled block
    char    reserve[11];
    } MCB_struct;


typedef struct
    {
    char name[48];
    BYTE address[12];
    } DBTABLE_struct;

typedef struct          // Information buffer to send to the database
    {
    BYTE requestType;
    WORD requestNumber;
    BYTE retry;
    char data[REQUEST_SIZE];
    } SENDDB_struct;

typedef struct          // Information buffer to receive from the database
    {
    BYTE responseType;
    WORD requestNumber;
    char data[RESPONSE_SIZE];
    } RECEIVEDB_struct;



//_____________________________________________________________________________
//                  Declaration of Routines

void SetParentEnv (char *envVar, char *newVar);

void SetLevelEnv (unsigned int far *env, char *envVar, char *newVar);

FILE *OpenSAC (char *mode);

int CheckForCommand(unsigned int pSeg, char *envVar, char *newVar);

int  DBinit (int *dbSocket,
             DBTABLE_struct *table,
             int *numDBservers,
             SENDDB_struct *sendDb);

int  Set_EcbSendDb (ECB *ECBSendDb,
                    int dbSocket,
                    IPXHeader *ipxSendDb,
                    SENDDB_struct *SendDb,
                    DBTABLE_struct *table,
                    int dbTablepos);

void Set_EcbReceiveDb (ECB *ECBReceiveDb,
                       int dbSocket,
                       IPXHeader *ipxReceiveDb,
                       RECEIVEDB_struct *receiveDb,
                       void (far *ESRAddress)());

BYTE DBRequest (RECEIVEDB_struct *receiveDb, // points to receive buffer
                SENDDB_struct *sendDb,       // points to send buffer
                BYTE *returnDBflag,          // flag indicating DB responded
                DBTABLE_struct *table,       // points to table of DB servers
                int dbSocket,                // DB communication socket #
                int dbTablepos,              // position in table of DB serv
                void (far *ESRAddress)(),    // address of ESR routine
                IPXHeader *ipxSendDb,
                IPXHeader *ipxReceiveDb,
                ECB *ECBSendDb,
                ECB *ECBReceiveDb);

BYTE LocateDBServer (DBTABLE_struct *table);

void ScramblePassword (char *plainText, char *c);

float FloatTime (struct time *start);

float DiffFloatTime (struct time *last, struct time *first);
```

B4.5 LIB.VM

```
/*
```

File name: LIB.VM

```
    This file contains some standard routines for the VM programs

    By S. J. Isaac
    date: 25 April 1993
*/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <dos.h>
#include <nit.h>
#include <nxt.h>

#include "h:\c\r\h\uvm.h"




//_____________________________________________________________________________
//                    SetParentEnv

void SetParentEnv (char *envVar, char *newVar)
{
  FILE              *sacFilePtr;
  SAC_struct        *sacInfo;
  long              offset;
  unsigned int      mainParentPSPseg;
  MCB_struct far    *mcbptr;
  PSP_struct          *ps;
  unsigned int far *points[20];
  unsigned int        p,mcbseg,count;                 // segments
  int               ccode;

    count=0;


    // Open SAC file

    if ((sacFilePtr = OpenSAC ("r+b")) == NULL){
        puts ("Could not open SAC file  ...TERMINATING!");
        fclose (sacFilePtr);
        exit (1);
    }


    // Read main parent's PSP from SAC file

    sacInfo = 0;
    offset = (long)((char *)&(sacInfo->parentPSPseg) - (char *)sacInfo);
    fseek (sacFilePtr, offset, SEEK_SET);
    fread ((void *)&mainParentPSPseg, sizeof (unsigned int), 1, sacFilePtr);
    fclose (sacFilePtr);


    // go to memory control block of main parent

    mcbptr = (MK_FP(p = mainParentPSPseg - 1,0));


    // examine the memory arena(s)

    while(*(char*)(MK_FP(p,0))==0x4d) { // 0x4d if not last, 0x5a if last
        if(mcbseg=(mcbptr->pspseg)) {    // psp segment of controlled memory
            ps=MK_FP(mcbseg,0);
            if(ps->envseg-p==1) {
                if(mcbseg!=_psp) {
```

```
                    // Check whether Command.com

                    ccode = CheckForCommand (ps->envseg, envVar, newVar);
                    if (ccode == 0){


                        // store environment pointer

                        points[count++]=MK_FP(ps->envseg,0);
                    }
                }
            }
        }
        p+=(mcbptr->size+1);
        mcbptr=MK_FP(p,0);
    }


    while(count--) {                    // set environment
        SetLevelEnv (points[count], envVar, newVar);
    }
}



//_____________________________________________________________________________
//                  SetLevelEnv

void SetLevelEnv (unsigned int far *env, char *envVar, char *newVar)
{
  char far *envBlockAddr;
  char far *envEnd;
  char far *envString;
  char far *varAddr;

  int envVarLen;
  int newVarLen;
  int len;

    newVarLen = strlen (newVar);

    envBlockAddr = (char *) env;


    // Determine the end of the environment block
    // The environment block ends with the 0x0000 word


    for (envEnd = envBlockAddr; 0 != *(unsigned int far *)envEnd; envEnd++);


    //Determine the starting address of the required environment variable

    envVarLen = strlen (envVar);
    envString = envBlockAddr;
    varAddr = NULL;
    do {
        if (strnicmp(envString, envVar, envVarLen) != 0){
            len = strlen(envString);
            envString += (len+1);
        }
        else {
            varAddr = envString;
            break;
        }
    }while (envString < envEnd);


    //Append to environment if the variable does not exist

    if (varAddr == NULL){
        if (newVar[0] != '\0'){
            strcpy(envString, envVar);
            strcat(envString, newVar);
            *(envString + envVarLen + (newVarLen + 1)) = '\0';
        }
        return;
    }
```

```
    //Insert in the environment

    len = strlen(envString);
    envString += (len+1);
    if (envString >= envEnd){     //If envVar is the last in the environment

        if (newVar[0] != '\0'){
            strcpy(varAddr + envVarLen, newVar);
            *(varAddr + envVarLen + (newVarLen + 1)) = '\0';
        }
        else{      // reset
            *(varAddr + 0) = '\0';
            *(varAddr + 1) = '\0';
        }
    }
    else {    //If envVar is not the last in the environment

        len = envEnd - envString + 2;
        if (newVar[0] != '\0'){
            memmove ((varAddr + newVarLen + envVarLen + 1), envString, len);
            strcpy(varAddr + envVarLen, newVar);
        }
        else  //reset
            memmove (varAddr, envString, len);
    }

    return;
}


//_____________________________________________________________________________
//                    OpenSAC

FILE *OpenSAC (char *mode)
{
  FILE *sac;
  int  ccode;
  WORD objType;
  char objName[48];
  long objID;
  BYTE loginTime[7];
  BYTE logTime[7];
  int  connectionNumber;
  char sacName[8];
  char sacExtension[4];
  char sacFile[50];
  long offset;
  char *tempEnvVar;


    // set primary server as preferred server

    SetPreferredConnectionID(GetPrimaryConnectionID());
    connectionNumber = GetConnectionNumber();
    ccode = GetConnectionInformation (connectionNumber,
                                      objName,
                                      &objType,
                                      &objID,
                                      loginTime);
    if (ccode != 0){
        printf("\nError %i in GetConnectionInformation()", ccode);
        exit(1);
    }


    // determine the server attachment count file name

    itoa (connectionNumber, sacExtension, 10);
    strcpy (sacName, "SAC.");
    strcat (sacName, sacExtension);


    // open server attachment count file or return if it cannot be found

    if ((tempEnvVar=getenv ("TEMP")) != NULL) {
        strcpy (sacFile, tempEnvVar);
        if (tempEnvVar[strlen(tempEnvVar)-1] != '\\')
```

```
            strcat (sacFile, "\\");
        strcat (sacFile, sacName);
        if ((sac=fopen(sacFile, mode)) == NULL) {
            printf ("Unable to open sacFile on %s", tempEnvVar);
            return sac;
        }
    }else return (sac = NULL);


    // Check whether this is the proper file for this session

    offset = SERVER_NAME_LENGTH + USER_NAME_LENGTH;
    fseek (sac, offset, SEEK_SET);
    fread ((void *)logTime, 7, 1, sac);
    if (memcmp (logTime, loginTime, 7) != 0) {
        sac = NULL;
        fclose (sac);
    }

    return sac;
}



//_____________________________________________________________________________
//                    CheckForCommand

int CheckForCommand(unsigned int pSeg,char *envVar, char *newVar)
{
  char far *p;
  unsigned int far *p1;
  unsigned int len, count;
  MCB_struct far *p2;


    p1=MK_FP(pSeg, 0x00);
    p2=MK_FP(pSeg-1, 0x00);


    for (p = (char far *)p1, count=0; *(int *)p; p++, count++);

    len = (p2->size)*16;
    if (len > count + strlen(envVar) + strlen(newVar) + 2) return 0;
    else return 1;
}



//_____________________________________________________________________________
//                    DBinit
/*
 return values
 1 = no database servers
 0 = success
*/

int  DBinit (int *dbSocket,
             DBTABLE_struct *table,
             int *numDBservers,
             SENDDB_struct *sendDb)
{

    *dbSocket = IntSwap ( (int) DB_SERVER_SOCKET );


    // obtain the address of the database

    *numDBservers = LocateDBServer(table);
    if (*numDBservers == 0)         return (1);  // no DB servers!


    // initialise database request number

    sendDb->requestNumber = 0;

    return (0); // success
}
```

```
//_____________________________________________________________________
//                    Set_EcbSendDb
/*
 return values
 0 = success
 1 = could not properly set the Send to database ECB
*/

int  Set_EcbSendDb (ECB *ECBSendDb,
                    int dbSocket,
                    IPXHeader *ipxSendDb,
                    SENDDB_struct *sendDb,
                    DBTABLE_struct *table,
                    int dbTablePos)

{
  int ccode, transportTime;


    // fill the ipxheader for g_sendDb

    memcpy((void *)&(ipxSendDb->destination),
           table[dbTablePos].address, 12);
    ipxSendDb->packetType = 4;


    // fill_ecbSendDb ();

    memset (ECBSendDb, 0, sizeof(ECB));
    ECBSendDb->ESRAddress = NULL;
    ECBSendDb->socketNumber = dbSocket;
    ECBSendDb->fragmentCount = 2;
    ECBSendDb->fragmentDescriptor[0].address = ipxSendDb;
    ECBSendDb->fragmentDescriptor[0].size = sizeof(IPXHeader);
    ECBSendDb->fragmentDescriptor[1].address = sendDb;
    ECBSendDb->fragmentDescriptor[1].size = sizeof(SENDDB_struct);

    ccode = IPXGetLocalTarget (table[dbTablePos].address,
                               ECBSendDb->immediateAddress,
                               &transportTime);

    if (ccode) return (1);     // fail
    else return (0);           // success
}



//_____________________________________________________________________
//                    Set_EcbReceiveDb

void Set_EcbReceiveDb (ECB *ECBReceiveDb,
                       int dbSocket,
                       IPXHeader *ipxReceiveDb,
                       RECEIVEDB_struct *receiveDb,
                       void (far *ESRAddress)())
{
    memset (ECBReceiveDb, 0, sizeof(ECB));
    ECBReceiveDb->ESRAddress = ESRAddress;
    ECBReceiveDb->socketNumber = dbSocket;
    ECBReceiveDb->fragmentCount = 2;
    ECBReceiveDb->fragmentDescriptor[0].address = ipxReceiveDb;
    ECBReceiveDb->fragmentDescriptor[0].size = sizeof(IPXHeader);
    ECBReceiveDb->fragmentDescriptor[1].address = receiveDb;
    ECBReceiveDb->fragmentDescriptor[1].size = sizeof(RECEIVEDB_struct);
}



//_____________________________________________________________________
//                    DBRequest
/*
    return values

 0xC1 = Could not set the Send ECB
 0xCC = No response from the specified database server

 0    = Success
 other values dependent on the error generated by specific request
*/
```

```
BYTE DBRequest (RECEIVEDB_struct *receiveDb, // points to receive buffer
                SENDDB_struct *sendDb,       // points to send buffer
                BYTE *returnDBflag,          // flag indicating DB responded
                DBTABLE_struct *table,       // points to table of DB servers
                int dbSocket,                // DB communication socket #
                int dbTablePos,              // position in table of DB serv
                void (far *ESRAddress)(),    // address of ESR routine
                IPXHeader *ipxSendDb,
                IPXHeader *ipxReceiveDb,
                ECB *ECBSendDb,
                ECB *ECBReceiveDb)
{
  int         ccode;
  struct time start, now;
  float       time_out = 1.5;
  int         c;


    // set other variables

    sendDb->requestNumber++;
    *returnDBflag = 0;


    // open the workstation socket to communicate with the database

    ccode = IPXOpenSocket ( (BYTE *) &dbSocket, (BYTE) SHORT_LIVED );
    if ( ccode ) printf("IPXOpenSocket error = %i\n", ccode);


    // Set receive and send ECBs

    Set_EcbReceiveDb (ECBReceiveDb,
                      dbSocket,
                      ipxReceiveDb,
                      receiveDb,
                      ESRAddress);

    ccode = Set_EcbSendDb  (ECBSendDb,
                            dbSocket,
                            ipxSendDb,
                            sendDb,
                            table,
                            dbTablePos);
    if (ccode){
        IPXCloseSocket (dbSocket);
        return (0xC1);
    }



    // Start listening for a response from the database

    IPXListenForPacket( ECBReceiveDb );


    // send and wait for database to reply


    for (sendDb->retry=0; sendDb->retry<MAX_RETRY; sendDb->retry++){

        IPXSendPacket (ECBSendDb);

        for (gettime(&start), gettime(&now);
             time_out > DiffFloatTime(&now, &start);
             gettime(&now) )  {

            if (*returnDBflag){   // Has the database responded?

                *returnDBflag = 0;
                IPXCloseSocket (dbSocket);
                return (receiveDb->responseType);
            }

            IPXRelinquishControl();
        }

    }
```

```
    IPXCloseSocket (dbSocket);
    return (0xCC);   //No response from database, so failed
}


//_____________________________________________________________________________
//                    LocateDBServer
/*
 return values
 The number of database servers that have been located
 0 = No database servers have been located
*/

BYTE LocateDBServer (DBTABLE_struct *table)
{
  int  oCode, pCode;
  long objectID = -1;
  char objectName[48];
  WORD objectType;
  char objectHasProperties;
  char objectFlag;
  char objectSecurity;
  BYTE propertyValue[128];
  BYTE moreSegments;
  BYTE propertyFlags;
  WORD dbServerType;
  BYTE numberOfServers;

    dbServerType =  (WORD) IntSwap ((int)DB_SERVER_TYPE);
    for (objectID = -1, oCode = 0, numberOfServers = 0;
        ( !oCode && (numberOfServers < MAX_DB_TABLE_POSITIONS) ); ){
        oCode = ScanBinderyObject ("*",
                                   dbServerType,
                                   &objectID,
                                   objectName,
                                   &objectType,
                                   &objectHasProperties,
                                   &objectFlag,
                                   &objectSecurity);
        if (oCode == 0){  // a database server has been located
            pCode = ReadPropertyValue (objectName,
                                       dbServerType,
                                       "NET_ADDRESS",
                                       1,
                                       propertyValue,
                                       &moreSegments,
                                       &propertyFlags);
            if (pCode == 0){  // The NET_ADDRESS has been obtained
                strcpy (table[numberOfServers].name, objectName);
                memcpy (table[numberOfServers].address,
                        propertyValue, 12);
                numberOfServers++;
            }
        }
    }

    return numberOfServers;
}


//_____________________________________________________________________________
//                    ScramblePassword

void ScramblePassword(char *plainText, char *c)
{
  size_t pLength;
  char p[130];
  char k[130];
  int i;

    strcpy(p, "zc");
    strcpy(k, "kg");

    strncat(p, plainText, 127);
    pLength = strlen(p);

    for (i = 2; i < pLength; i++){
        k[i] = abs((((p[i]*p[i-1])/(p[i-2]+1)) + (k[i-1]+k[i-2])) % 26) + 65;
```

```
    }
    for (;i <127;i++){
        k[i] = abs((k[i-1] * k[i-2] / (k[i-3]+4)) % 26) + 65;
    }
    k[127] = '\0';
    strcpy (c, &k[2]);
}


//_____________________________________________________________________________
//                    FloatTime

float FloatTime (struct time *start)
{
  float sec;

    sec = (((start->ti_hour * 60)+start->ti_min) * 60)+start->ti_sec +
          (0.01*start->ti_hund);
    return sec;
}


//_____________________________________________________________________________
//                    DiffFloatTime

float DiffFloatTime (struct time *last, struct time *first)
{
  float a, b, diff;

    a = FloatTime (first);
    b = FloatTime (last);
    diff = b - a;
    return diff;
}
```

APPENDIX C1

C D B . C

CDB.C is the source file for the program that creates the Virtual Machine (VM) database. The database files that are presently modified by the program (directly) are: Ap_acces.dbf, Ap_batch.dbf, Password.dbf and Menu.dbf. C-Scape Interface Management routines are used to create a windowing environment (with mouse and keyboard support) that enables the database to be easily modified.

C1.1 Description of Code

C1.1.1 Function — main()

The function main does not return any values or receive any parameters. The pseudocode is presented below.

```
void main (void)
{
    while (the user does not want to quit the program){
        depending on the user's choice open the relevant database files
        and present the data entry screen for the choice.
    }
}
```

C1.1.2 Function — update_record_number()

This function is used to set new record numbers in the appropriate column of a scrolling list editor (sled) after an insert or a delete operation has been carried out.

C1.1.3 Function — Reset_Record()

This function is used to reset a record displayed in a sled.

C1.1.4 Functions Concerned with the Application Database

1. init_Ap_access_sled_var(): This function is used to fill sled variables from the file Ap_acces.dbf. If the file Ap_acces.dbf does not exist, then an empty sled is presented.
2. file_Ap_access_sled_var(): This function is used to place sled variables into the file Ap_acces.dbf. It is usually called when the changes made to the database are required to be saved.
3. Ap_access_scr(): This function presents the screen to the user and allows the user to make changes to the database.
4. Similar functions as in 1 to 3 exist for the Ap_batch.dbf file.

C1.1.5 Function — Menu_scr()

This function presents the menu database to the administrator. It is in essence a menu builder. Three windows will be visible on the screen. These are: the menu editor (with command buttons), the applications list and the menu list. To minimise errors in building the menu, the administrator chooses items from either the applications list or the menu list to insert in the menu record. The pseudocode is presented below.

```
int Menu_scr (void)
{
    open menu.dbf file

    build menu editor window
    build application list window
    build menu list window

    ascribe functionality to the command buttons and make the menu editor active
}
```

C1.1.6 Function — MenuEditor()

This function makes the menu editor window active.

C1.1.7 Function — AppInitVar()

This function builds the application list window

C1.1.8 Function — MenuInitVar()

This function updates the menu list window and is called whenever a change is made to a menu title or when a menu is added or deleted from the list.

C1.1.9 Function — ins_del_record()

This function is called whenever a record in the menu.dbf file is inserted or deleted.

C1.1.10 Function — SetMenuSedVar()

This function is called by the function MenuEditor(). The purpose of SetMenuSedVar() is to update the fields in the Menu Editor window with current values for the indicated menu record.

C1.1.11 Function — SetMenuFileRec()

This function is called by the function MenuEditor(). The purpose of SetMenuFileRec() is to update the fields a particular record in the menu.dbf file (especially after changes have been made to that record).

C1.1.12 Function — Password_scr()

This function allows the administrator to modify the password.dbf file.

C1.1.13 Special Functions

The special functions are used in the windows to interpret key presses and mouse clicks. Each time a key is pressed (or a mouse is clicked), control is passed to these special functions and afterwards returned to the function responsible for that window.

spc_MySled() is used by the functions: Ap_access_scr(), Ap_batch_scr() and User_scr(). **spc_List()** is used in both the application list and menu list windows. **spc_MyFields()** is used in the menu editor window.

C1.2 Program Listing

```
/*
                                File name: CDB.C

 Compiler Model LARGE.
 C++ options turned to "always use C++ compiler".
 The include directory should also contain the location of the cscape
 include files.
 In Options/Compiler/Code Generation the "treat enums as int" should be
 turned on.

 Project File should include the following files:
       cdb.c
       tplowl.lib
       tplcscap.lib


This is the source code for the application to create or modify the
VM database.

By S.J. Isaac

Last Revised: 23 January 1993

*/

#include <stdio.h>
#include <conio.h>
#include <io.h>
#include <string.h>
#include <dir.h>
#include "cscape.h"
#include "popdecl.h"
#include "ostdlib.h"
#include "msys.h"
#include "sled.h"
#include "scancode.h"

#include "c:\h\uvm.h"


#define MAX_APPLICATIONS  1000  // max. no. of applications in Ap_batch.ndx
#define EMPTY_VALUE  0x0000     // value of the empty records in Ap_batch.ndx



typedef unsigned char BYTE;
typedef unsigned int WORD;



//_____________________________________________________________________________
//               Global variable declaration

FILE *Ap_access_fptr, *Ap_batch_fptr, *User_fptr, *Menu_fptr;
int  selection;
char tmpname[13];
size_t g_menuSize;
char tType[MENU_ITEMS];
char tName[MENU_ITEMS][20];
WORD tID[20];
WORD tMenuID;
WORD tGoto;
WORD tResetAppID;
char tMenuName[20];
int  g_totMenuRecs = 0;
int  g_menuTitleChanged;
int  g_inApBatch = 0;
sed_type  medSed, mSed, aSed;
sed_type batchSled;
menu_type mMenu;


//===  Structure of the Records in the files

struct Ap_acces
```

```
    {
    char AppName[20];
    BYTE Access;
    BYTE OS;
    WORD Hardware;
    } Ap_acces;

struct Ap_batch
    {
    char Server[3];
    WORD AppID;
    char BatchFile[13];
    } Ap_batch;

struct User
    {
    char Server[3];
    char UserID[9];
    char Name[31];
    BYTE Access;
    } User;

struct Menu_choice
    {
    char type;
    char ItemName[20];
    WORD ID;
    } Menu_choice;

struct Menu
    {
    char MenuName[20];
    struct Menu_choice item[MENU_ITEMS];
    } Menu;


//=========  PopUp Menu structures

static char *filesMenu[] = {
    "1. Ap_acces.dbf",
    "2. Ap_batch.dbf",
    "3. User.dbf",
    "4. Menu.dbf",
    "5. Password.dbf",
    NULL
    };

static char *save_on_exit[] = {
    "Save then EXIT",
    "EXIT",
    NULL
    };


//_____________________________________________________________________________
//                Declaration of routines

void display_help (void);
boolean spc_MySled(sed_type sled, int scancode);
int update_record_number(sed_type sled, int rec);
int Reset_Record(sed_type sled,int rec);

int init_Ap_acces_sled_var(sed_type sled);
int file_Ap_acces_sled_var(sed_type sled);
int Ap_acces_scr (void);

int init_Ap_batch_sled_var(sed_type sled);
int file_Ap_batch_sled_var(sed_type sled);
int Ap_batch_scr(void);

int init_User_sled_var(sed_type sled);
int file_User_sled_var(sed_type sled);
int User_scr(void);

int Menu_scr(void);
int MenuEditor(sed_type sed, int rec);
void AppInitVar (void);
void MenuInitVar (void);
```

```
void MMenuPrintf (int rec);

int ins_del_record (long rec, int insert);
boolean spc_MyFields (sed_type sed, int scancode);
boolean spc_List (sed_type sed, int scancode);
void SetMenuSedVar (int rec);
void SetMenuFileRec (int rec);

int Password_scr(void);



//_____________________________________________________________________________

void main (void)
{
  int done;
  char buf[256];

    disp_Init(def_ModeText, FNULL);
    strcpy(buf,"\nCREATE VM DATABASE PROGRAM\n");
    strcat(buf,"\nby S.J. Isaac");
    strcat(buf,"\nVer 1.0");
    strcat(buf,"\nDate: 15/Jan/93\n");
    pop_Message(buf,-1,-1,-1,-1,0x3f,bd_2);
    hard_InitMouse();
    sedwin_ClassInit();

    pop_Message(NULL,-1,-1,-1,-1,0x3f,bd_2);

    while (selection = pop_Menu ("FILES MENU", filesMenu,
                                 -1,-1,6,18, 0x1f, 0, bd_box)){


        switch ( selection ) {

            case 1:     //=======================  Ap_acces.dbf

                if( (Ap_access_fptr=fopen("Ap_acces.dbf","r+b"))==NULL){
                    if( (Ap_access_fptr=fopen("Ap_acces.dbf","w+b"))==NULL){
                        perror("cannot open Ap_acces.dbf");
                        done = 0;
                    }else done =1;
                }else done = 1;

                if (done){
                    Ap_acces_scr ();
                    clrscr();

                    fclose(Ap_access_fptr);
                }

                break;

            case 2:     //=======================  Ap_batch.dbf

                if( (Ap_batch_fptr=fopen("Ap_batch.dbf","r+b"))==NULL){
                    if( (Ap_batch_fptr=fopen("Ap_batch.dbf","w+b"))==NULL){
                        perror("cannot open Ap_batch.dbf");
                        done = 0;
                    }else done =1;
                }else done = 1;

                if (done){
                    Ap_batch_scr ();
                    clrscr();

                    fclose(Ap_batch_fptr);
                }

                break;

            case 3:     //=======================  User.dbf

                if( (User_fptr=fopen("User.dbf","r+b"))==NULL){
                    if( (User_fptr=fopen("User.dbf","w+b"))==NULL){
                        perror("cannot open User.dbf");
                        done = 0;
```

```
                }else done = 1;
            }else done = 1;

            if (done){
                User_scr ();
                clrscr();

                fclose(User_fptr);
            }

            break;

        case 4:     //=======================  Menu.dbf

            if( (Menu_fptr=fopen("Menu.dbf","r+b"))==NULL){
                if( (Menu_fptr=fopen("Menu.dbf","w+b"))==NULL){
                    perror("cannot open Menu.dbf");
                    done = 0;
                }else done = 1;
            }else done = 1;

            if (done){
                Menu_scr ();
                textbackground(0x70);
                clrscr();

                fclose(Menu_fptr);
            }

            break;

        case 5:     //=======================  Password.dbf

            Password_scr();
            clrscr();

            break;
    }

  }

  disp_Close();
  exit(0);


}



//_____________________________________________________________________________
//                  displays the database help commands for each screen

void display_help (void)
{
  char *text ={"\n          TAB:\tmove right\
               \n    SHIFT+TAB:\tmove left\
               \n           UP:\tmove up\
               \n         DOWN:\tmove down\
               \n\
               \n         PGUP:\tPage Up\
               \n         PGDN:\tPage Down\
               \n    CTRL+PGUP:\tFirst Record\
               \n    CTRL+PGDN:\tLast Record\
               \n\
               \n GREY + or -:\tIns/Del Record\
               \n           F5:\tReset Record\
               \n           F9:\tAppend Record\
               \n\
               \n           F1:\tHelp\
               \n          ESC:\tQuit\n\
              "};


    pop_View("HELP",text,-1,-1,-1,-1,0x74,0,bd_box);

}
```

```
//_____________________________________________________________________
//                    The Special Function for the Sled

boolean spc_MySled(sed_type sled, int scancode)
{
  int row, max_rows, choice, i;
  mev_struct mev;

    switch (scancode) {

    default:
        return(spc_Sled (sled, scancode));

    case ESC:       //==  exits the database display
        choice = pop_Menu ("", save_on_exit,-1,-1,3,18,0x4f,0,bd_2);
        if (choice){
            if(choice==1)  sed_SetBaton(sled, ESC);
            else sed_SetBaton(sled, SED_ABORT);
            sed_ToggleExit(sled);
        }
        break;

    case TAB:       //==  moves one field to the right

    case ENTER:
        /* try to go to the next field */
        if (sed_IncField(sled) == SED_STUCK
            &&
            sled_Remap(sled, 1) == SED_MOVED) {

            sed_LeftField(sled);
        }
        break;

    case SHFT_TAB: //==  moves one field to the left
        /* try to go to the prev field */
        if (sed_DecField(sled) == SED_STUCK
            &&
            sled_Remap(sled, -1) == SED_MOVED) {

            sed_RightField(sled);
        }
        break;

    case UP:        //==  moves one field up
        if (sed_UpField(sled) == SED_STUCK)
            sled_Remap(sled, -1);
        break;

    case DOWN:      //==  moves one field down
        if (sed_DownField(sled) == SED_STUCK)
            sled_Remap(sled, 1);
        break;

    case PGUP:      //==  moves up one page in the display
        if (sled_Remap(sled, -sed_GetHeight(sled)) != SED_MOVED)
            for(;sed_UpField(sled) != SED_STUCK;);
        break;

    case PGDN:      //==  moves down one page in the display
        if (sled_Remap(sled, sed_GetHeight(sled)) != SED_MOVED)
            for(;sed_DownField(sled) != SED_STUCK;);
        break;

    case FN5:       //==  Resets the record at the cursor position
        row = sled_GetRow(sled);
        sled_Remap(sled,0);
        Reset_Record(sled,row);
        sled_Repaint(sled);
        break;

    case GREYPLUS: //==  inserts a row at the cursor position
        row = sled_GetRow(sled);
        sled_InsertRows(sled, row, 1);
        update_record_number(sled, row);
        sled_Repaint(sled);
        break;

    case GREYMINUS://==  deletes a row at the cursor position
        row = sled_GetRow(sled);
```

```
        sled_DeleteRows(sled, row, 1);
        update_record_number(sled, row);
        sled_Repaint(sled);
        break;

    case FN9:       //==  appends a row
        max_rows = sled_GetColSize(sled);
        row = sled_GetRow(sled);
        sled_InsertRows(sled,max_rows,1);
        sled_Remap(sled, max_rows-row);
        sled_SetColVar(sled,0, max_rows, (VOID *)&max_rows);
        Reset_Record(sled,max_rows);
        sed_GotoGridField(sled,max_rows,1);
        sled_Repaint(sled);
        break;

    case CTRL_PGUP://==  moves to the first field in the database
        row = sled_GetRow(sled);
        sled_Remap(sled, -row);
        sed_GotoFirstField(sled);
        sled_Repaint(sled);
        break;

    case CTRL_PGDN://==  moves to the last field in the database
        max_rows = sled_GetColSize(sled);
        row = sled_GetRow(sled);
        sled_Remap(sled, max_rows-row);
        sed_GotoLastField(sled);
        sled_Repaint(sled);
        break;

    case FN1:
        display_help();
        break;

    case MOU_CLICK:
        win_MouseCurrEvent(&mev);
        //check if right mouse button pressed
        if (mev_IsButton3Down(&mev)) i = 1;
        if (mev_IsButton2Down(&mev)) i = 1;
        if (i != 1) return (FALSE);

    case FN2:
        row = sled_GetRow(sled);
        if (g_inApBatch == 1){
            sled_SetColVar (sled, 2, row, (VOID *)&Menu_choice.ID);
            sled_Repaint(sled);
        }
        break;


    }

    return(TRUE);
}


//_____________________________________________________________________
//          Set New Record Numbers after an Insert or Delete Operation

int update_record_number(sed_type sled, int rec)
{
  int row, max_rows, tmp;

    max_rows = sled_GetColSize(sled);

    for (row=rec; row < max_rows; row++){
        if (selection==1){
            tmp =row+1;
            sled_SetColVar(sled,0, row, (VOID *)&tmp);
        }
        else sled_SetColVar(sled,0, row, (VOID *)&row);
    }
    return 1;
}


//_____________________________________________________________________
```

```
int Reset_Record(sed_type sled,int rec)
{
  int tmp;

    sled_DeleteRows(sled,rec,1);
    sled_InsertRows(sled,rec,1);
    if (selection==1){
        tmp =rec+1;
        sled_SetColVar(sled,0, rec, (void *)&tmp);
    }
    else sled_SetColVar(sled,0,rec,(VOID *)&rec);

    return 1;
}


//_____________________________________________________________________

int init_Ap_acces_sled_var(sed_type sled)
{
  int recno, tmp=1;
  size_t size;

    size = sizeof(struct Ap_acces);

    sled_SetColVar(sled,0,0,(VOID *)&tmp); //set (0,0) to AppID=1

    fseek (Ap_access_fptr, 0, SEEK_SET);
    fread((VOID *)&Ap_acces, size, 1, Ap_access_fptr);
    for (recno=0;!feof(Ap_access_fptr);recno++){
        tmp = recno+1;
        sled_SetColVar(sled,0,recno,(VOID *)&tmp);
        sled_SetColVar(sled,1,recno,(VOID *)Ap_acces.AppName);
        tmp = (int)(Ap_acces.Access);
        sled_SetColVar(sled,2,recno,(VOID *)&tmp);
        tmp = (int)(Ap_acces.OS);
        sled_SetColVar(sled,3,recno,(VOID *)&tmp);
        sled_SetColVar(sled,4,recno,(VOID *)&(Ap_acces.Hardware));
        fread((VOID *)&Ap_acces, size, 1, Ap_access_fptr);
    }
    return 1;
}



//_____________________________________________________________________

int file_Ap_acces_sled_var(sed_type sled)
{
  int row, max_rows;
  size_t size;

    size = sizeof(struct Ap_acces);
    max_rows = sled_GetColSize(sled);
    fseek (Ap_access_fptr, 0, SEEK_SET);
    for (row=0; row<max_rows; row++){
        strcpy (Ap_acces.AppName, (char *)sled_GetColVar (sled, 1, row));
        Ap_acces.Access = *((BYTE *)sled_GetColVar (sled, 2, row));
        Ap_acces.OS = *((BYTE *)sled_GetColVar (sled, 3, row));
        Ap_acces.Hardware = *((WORD *)sled_GetColVar (sled, 4, row));
        fwrite ((VOID *)&Ap_acces, size, 1, Ap_access_fptr);
    }
    chsize (fileno(Ap_access_fptr), ftell(Ap_access_fptr));
    return 1;
}



//_____________________________________________________________________

int Ap_acces_scr(void)
{
    menu_type   menu;
    sed_type    sed;
    int         ret;
```

```
    menu = menu_Open();

    menu_Printf(menu, "@p[0,0]@fpd[####]", NULL, &hex_funcs,
                "Application ID - Hex");
    menu_Printf(menu, "@p[0,5]@fd[@[19,#]]", NULL, &string_funcs,
                "Application Name");
    menu_Printf(menu, "@p[0,26]@fd[##]", NULL, &hex_funcs,
                "Access rights - Hex");
    menu_Printf(menu, "@p[0,30]@fd[##]", NULL, &hex_funcs,
                "Operating System - Hex");
    menu_Printf(menu, "@p[0,33]@fd[####]", NULL, &hex_funcs,
                "Hardware - Hex");

    menu_Flush(menu);

    sed = sled_Open(menu, 23, spc_MySled);
    sed_SetColors(sed, 0x31, 0x13, 0x0c);

    sed_SetBorder(sed, bd_mouse);
    sed_SetBorderFeature(sed, 0);
    sed_SetBorderTitle(sed, "Ap_acces.dbf");
    sed_SetPosition(sed, 0, 0);
    sed_SetWidth(sed, 60);
    sed_SetMouse(sed, sedmou_Track);


    init_Ap_acces_sled_var(sed);

    sed_Repaint(sed);
    ret = sed_Go(sed);
    if (ret == ESC) file_Ap_acces_sled_var(sed);

    sed_Close(sed);
    return(ret);
}



//_____________________________________________________________________________

int init_Ap_batch_sled_var(sed_type sled)
{
  int recno;
  size_t size;

    size = sizeof(struct Ap_batch);
    fseek (Ap_batch_fptr, 0, SEEK_SET);
    fread((void *)&Ap_batch, size, 1, Ap_batch_fptr);
    for (recno=0;!feof(Ap_batch_fptr);recno++){
        sled_SetColVar(sled,0,recno,(void *)&recno);
        sled_SetColVar(sled,1,recno,(void *)Ap_batch.Server);
        sled_SetColVar(sled,2,recno,(void *)&(Ap_batch.AppID));
        sled_SetColVar(sled,3,recno,(void *)Ap_batch.BatchFile);
        fread((void *)&Ap_batch, size, 1, Ap_batch_fptr);
    }
    return 1;
}



//_____________________________________________________________________________

int file_Ap_batch_sled_var(sed_type sled)
{
  int row, max_rows;
  size_t size;

    size = sizeof(struct Ap_batch);
    max_rows = sled_GetColSize(sled);
    fseek (Ap_batch_fptr, 0, SEEK_SET);
    for (row=0; row<max_rows; row++){
        strcpy (Ap_batch.Server, (char *)sled_GetColVar (sled, 1, row));
        Ap_batch.AppID = *((WORD *)sled_GetColVar (sled, 2, row));
        strcpy (Ap_batch.BatchFile, (char *)sled_GetColVar (sled, 3, row));
        fwrite ((void *)&Ap_batch, size, 1, Ap_batch_fptr);
    }
    chsize (fileno(Ap_batch_fptr), ftell(Ap_batch_fptr));
    return 1;
```

```
}


//_____________________________________________________________________________

int Ap_batch_scr(void)
{
    menu_type   menu;
    sed_type    batchSled;
    int         ret, i;

    g_inApBatch = 1;

    AppInitVar();
    sed_SetSpecial (aSed, FNULL /*spc_batchApList*/ );

    menu = menu_Open();

    menu_Printf(menu, "@p[0,0]@fpd[#####]", NULL, &pint_funcs,
                "Record Number");
    menu_Printf(menu, "@p[0,7]@fd3[##]", NULL, &string_funcs,
                "Resident Server",NULL,"U");
    menu_Printf(menu, "@p[0,11]@fd[####]", NULL, &hex_funcs,
                "Application ID");
    menu_Printf(menu, "@p[0,17]@fd3[@[12,#]]", NULL, &string_funcs,
                "Batch File Name",NULL,"U");

    menu_Flush(menu);

    batchSled = sled_Open(menu, 23, spc_MySled);
    sed_SetColors(batchSled, 0x31, 0x13, 0x0c);

    sed_SetBorder(batchSled, bd_mouse);
    sed_SetBorderFeature(batchSled, 0);
    sed_SetBorderTitle(batchSled, "Ap_batch.dbf");
    sed_SetPosition(batchSled, 0, 0);
    sed_SetWidth(batchSled, 35);
    sed_SetMouse(batchSled, sedmou_Track);

    init_Ap_batch_sled_var(batchSled);

    sed_Repaint(batchSled);
    i = 0;
    do {
        ret = sed_Go(batchSled);
        if (ret == ESC){
            file_Ap_batch_sled_var(batchSled);
            i = 1;
        }
        else if (ret == SED_ABORT) i = 1;
             else Menu_choice.ID = ret;
    }while (i != 1);

    g_inApBatch = 0;

    fclose(Ap_access_fptr);
    sed_Close(aSed);
    sed_Close(batchSled);
    return(ret);
}



//_____________________________________________________________________________

int init_User_sled_var(sed_type sled)
{
  int recno, tmp;
  size_t size;

    size = sizeof(struct User);
    fseek (User_fptr, 0, SEEK_SET);
    fread((void *)&User, size, 1, User_fptr);
    for (recno=0;!feof(User_fptr);recno++){
        sled_SetColVar(sled,0,recno,(void *)&recno);
        sled_SetColVar(sled,1,recno,(void *)User.Server);
        sled_SetColVar(sled,2,recno,(void *)User.UserID);
        sled_SetColVar(sled,3,recno,(void *)User.Name);
```

```
        tmp = (int)(User.Access);
        sled_SetColVar(sled,4,recno,(void *)&tmp);
        fread((void *)&User, size, 1, User_fptr);
    }
    return 1;
}



//_____________________________________________________________________

int file_User_sled_var(sed_type sled)
{
  int row, max_rows;
  size_t size;

    size = sizeof(struct User);
    max_rows = sled_GetColSize(sled);
    fseek (User_fptr, 0, SEEK_SET);
    for (row=0; row<max_rows; row++){
        strcpy (User.Server, (char *)sled_GetColVar (sled, 1, row));
        strcpy (User.UserID, (char *)sled_GetColVar (sled, 2, row));
        strcpy (User.Name, (char *)sled_GetColVar (sled, 3, row));
        User.Access = *((BYTE *)sled_GetColVar (sled, 4, row));
        fwrite ((void *)&User, size, 1, User_fptr);
    }
    chsize (fileno(User_fptr), ftell(User_fptr));
    return 1;
}



//_____________________________________________________________________

int User_scr(void)
{
    menu_type   menu;
    sed_type    sed;
    int         ret;

    menu = menu_Open();

    menu_Printf(menu, "@p[0,0]@fpd[#####]", NULL, &pint_funcs,
                "Record Number");
    menu_Printf(menu, "@p[0,7]@fd3[##]", NULL, &string_funcs,
                "Resident Server",NULL,"U");
    menu_Printf(menu, "@p[0,11]@fd3[@[8,#]]", NULL, &string_funcs,
                "UserID",NULL,"U");
    menu_Printf(menu, "@p[0,21]@fd3[@[30,#]]", NULL, &string_funcs,
                "User's Name",NULL,"P");
    menu_Printf(menu, "@p[0,53]@fd[##]", NULL, &hex_funcs,
                "Access Rights - in Hex");

    menu_Flush(menu);

    sed = sled_Open(menu, 23, spc_MySled);
    sed_SetColors(sed, 0x31, 0x13, 0x0c);

    sed_SetBorder(sed, bd_mouse);
    sed_SetBorderFeature(sed, 0);
    sed_SetBorderTitle(sed, "User.dbf");
    sed_SetPosition(sed, 0, 0);
    sed_SetWidth(sed, 60);
    sed_SetMouse(sed, sedmou_Track);

    init_User_sled_var(sed);

    sed_Repaint(sed);
    ret = sed_Go(sed);

    if (ret == ESC) file_User_sled_var(sed);

    sed_Close(sed);
    return(ret);
}


//_____________________________________________________________________
```

```
int Menu_scr(void)
{
  menu_type    medMenu;
  int          medValue;
  int       i, recno, c;


    if( (Menu_fptr=fopen("Menu.dbf","r+b"))==NULL){
        if( (Menu_fptr=fopen("Menu.dbf","w+b"))==NULL){
            perror("cannot open Menu.dbf");
            exit(1);
        }
    }


    // Initialisations ======================================================

    g_menuSize = sizeof (struct Menu);
    fseek (Menu_fptr, 0, SEEK_END);
    g_totMenuRecs = ftell(Menu_fptr) / g_menuSize;
    if (g_totMenuRecs == 0){
        memset((void *) &Menu, 0, g_menuSize);
        fwrite((void *) &Menu, g_menuSize, 1, Menu_fptr);
        g_totMenuRecs++;
    }


    // MenuEditor inits ----------------------------------
    medMenu = menu_Open();

    // print separator bar and numbers

    for (i=0; i<23; i++)    menu_Printf(medMenu,"@p[%d,28]°", i);
    menu_Printf(medMenu, "@p[1,0]@[28,f]¹");
    for (i=2; i<22; i++)    menu_Printf(medMenu,"@p[%d,0]%d", i, i-1);

    // print menuID and menu title

    menu_Printf(medMenu, "@p[0,2]@fpd[##]", &tMenuID, &hex_funcs,
                "MenuID");
    menu_Printf(medMenu, "@p[0,5]@fd[@[19,#]]", tMenuName, &string_funcs,
                "Menu Name");

    // print menu choices - fields

    for (i = 2; i < 22; i++){
        menu_Printf(medMenu, "@p[%d,3]@fpd3[#]", i,
                    &tType[i-2], &char_funcs, "Type", "A,M", "U");
        menu_Printf(medMenu, "@p[%d,5]@fd[@[19,#]]", i,
                    tName[i-2], &string_funcs, "Name");
        menu_Printf(medMenu, "@p[%d,25]@fpd[##]", i,
                    &tID[i-2], &hex_funcs, "ID");
    }

    // print command buttons

    menu_Printf(medMenu, "@p[0, 30]@fw17d[      NEXT       ]",
                NULL, &menu_funcs, "Get Next Menu");
    menu_Printf(medMenu, "@p[2, 30]@fw17d[    PREVIOUS     ]",
                NULL, &menu_funcs, "Get Previous Menu");
    menu_Printf(medMenu, "@p[4, 30]@fw17d[    APP LIST     ]",
                NULL, &menu_funcs, "Show Application Listing");
    menu_Printf(medMenu, "@p[6, 30]@fw17d[    MENU LIST    ]",
                NULL, &menu_funcs, "Show Menu Listing");
    menu_Printf(medMenu, "@p[8, 30]@fw17d[    GOTO: ##     ]",
                &tGoto, &hex_funcs,
                "Enter menu ID to go to");
    menu_Printf(medMenu, "@p[10,30]@fw17d[     APPEND      ]",
                NULL, &menu_funcs, "Append a record");
    menu_Printf(medMenu, "@p[12,30]@fw17d[   RESET MENU    ]",
                NULL, &menu_funcs,
                "Reset this menu and all references to it");
    menu_Printf(medMenu, "@p[14,30]@fw17d[   RESET ITEM    ]",
                NULL, &menu_funcs,
                "Resets the paste buffer for the items");
    menu_Printf(medMenu, "@p[16,30]@fw17d[ RESET appID: ## ]",
                &tResetAppID, &hex_funcs,
                "Resets all choices with this appID");
    menu_Printf(medMenu, "@p[20,30]@fw17d[     DELETE      ]",
```

```
            NULL, &menu_funcs, "Delete a menu at current position");
menu_Printf(medMenu, "@p[22,30]@fw17d[     INSERT      ]",
            NULL, &menu_funcs, "Insert a menu at current position");

menu_Flush(medMenu);

medSed = sed_Open(medMenu);
sed_SetColors(medSed, 0x30, 0x17, 0x4b);

sed_SetBorder(medSed, bd_prompt);
sed_SetBorderColor(medSed, 0x1b);
sed_SetBorderTitle(medSed, "Menu Editor");
sed_SetPosition(medSed, 0, 0);
sed_SetHeight(medSed, 23);
sed_SetWidth(medSed, 48);
sed_SetMouse(medSed, sedmou_Click);
sed_SetBorderFeature (medSed, BD_MOVE | BD_RESIZE | BD_TOP);
sed_SetSpecial(medSed, spc_MyFields);
sed_SetShadow (medSed, 1);
sed_SetShadowAttr (medSed, 0x08);
// End of MenuEditor inits ----------------------------

AppInitVar();

// MenuList inits -------------------------------------
mMenu = menu_Open();

fseek (Menu_fptr, 0, SEEK_SET);
fread((void *)&Menu, g_menuSize, 1, Menu_fptr);
for (recno=0;!feof(Menu_fptr);recno++){
    MMenuPrintf (recno);
    fread((void *)&Menu, g_menuSize, 1, Menu_fptr);
}

menu_Flush(mMenu);

mSed = sed_Open(mMenu);
sed_SetColors(mSed, 0x21, 0x21, 0x40);

sed_SetBorder(mSed, bd_mouse);
sed_SetBorderTitle(mSed, "Menu List");
sed_SetPosition(mSed, 14, 50);
sed_SetWidth(mSed, 28);
sed_SetHeight (mSed, 9);
sed_SetMouse(mSed, sedmou_Track);
sed_SetBorderFeature (mSed, BD_MOVE | BD_RESIZE | BD_TOP);
sed_SetSpecial(mSed, spc_List);
sed_SetShadow (mSed, 1);
sed_SetShadowAttr (mSed, 0x08);
sed_Repaint(mSed);
// End of MenuList inits ------------------------------


// End of Initialisations ===============================================

recno = 0;
while ((medValue = MenuEditor(medSed, recno)) != SED_ABORT){
    switch(medValue){
        case 63:  //Next
            recno++;
            if (recno > (g_totMenuRecs - 1))
                recno = g_totMenuRecs - 1;
            break;

        case 64:  //Previous
            recno--;
            if (recno < 0) recno = 0;
            break;


        case 67:  //Goto Menu
            recno = tGoto;
            if (recno > (g_totMenuRecs - 1))
                recno = g_totMenuRecs - 1;
            break;

        case 68:  //Append
            fseek (Menu_fptr, 0, SEEK_END);
            memset((void *) &Menu, 0, g_menuSize);
            fwrite((void *) &Menu, g_menuSize, 1, Menu_fptr);
```

```
            g_totMenuRecs++;
            recno = g_totMenuRecs - 1;
            MMenuPrintf (recno);
            MenuInitVar();
            sed_Repaint(mSed);
            break;

        case 69:  //Reset
            for (i=0; i<g_totMenuRecs; i++){
                if (i == recno){
                    fseek (Menu_fptr, i * g_menuSize, SEEK_SET);
                    memset((void *) &Menu, 0, g_menuSize);
                    fwrite((void *) &Menu, g_menuSize, 1, Menu_fptr);
                }
                else{
                    fseek (Menu_fptr, i * g_menuSize, SEEK_SET);
                    fread((void *) &Menu, g_menuSize, 1, Menu_fptr);
                    for (c=0; c<MENU_ITEMS; c++){
                        if (Menu.item[c].type == 'M'){
                            if (Menu.item[c].ID == recno){
                                memset((void *) &Menu.item[c], 0,
                                       sizeof (struct Menu_choice) );
                                fseek (Menu_fptr, i * g_menuSize,
                                       SEEK_SET);
                                fwrite((void *) &Menu, g_menuSize,
                                       1, Menu_fptr);
                            }
                        }
                    }
                }
            }
            MenuInitVar();
            sed_Repaint(mSed);
            break;

        case 70:  //Reset item
            memset ((void *)&Menu_choice, 0, sizeof (struct Menu_choice));
            break;

        case 71:  //Reset AppID
            for (i=0; i<g_totMenuRecs; i++){
                fseek (Menu_fptr, i * g_menuSize, SEEK_SET);
                fread((void *) &Menu, g_menuSize, 1, Menu_fptr);
                for (c=0; c<MENU_ITEMS; c++){
                    if (Menu.item[c].type == 'A'){
                        if (Menu.item[c].ID == tResetAppID){
                            memset((void *) &Menu.item[c], 0,
                                   sizeof (struct Menu_choice) );
                            fseek (Menu_fptr, i * g_menuSize,
                                   SEEK_SET);
                            fwrite((void *) &Menu, g_menuSize,
                                   1, Menu_fptr);
                        }
                    }
                }

            }
            break;

        case 72:  //Delete
            ins_del_record (recno, 0);
            g_totMenuRecs--;
            if (recno > (g_totMenuRecs - 1))
                recno = g_totMenuRecs - 1;
            sed_DeleteRows (mSed, recno, 1);
            MenuInitVar();
            sed_Repaint(mSed);
            break;

        case 73:  //Insert
            ins_del_record (recno, 1);
            g_totMenuRecs++;
            sed_InsertRows (mSed, recno, 1);
            MMenuPrintf(recno);
            MenuInitVar();
            sed_Repaint(mSed);
            break;

    }
}
```

```
    fclose(Menu_fptr);
    fclose(Ap_access_fptr);
    sed_Close(medSed);
    sed_Close(mSed);
    sed_Close(aSed);

    return(0);
}

//---------------------------------------------------------------------------

int MenuEditor(sed_type sed, int rec)
{
  int ret;

    SetMenuSedVar (rec);

    sed_Repaint(sed);
    ret = sed_Go(sed);

    SetMenuFileRec (rec);

    return(ret);
}


//---------------------------------------------------------------------------

void AppInitVar (void)
{
  int recno, tmp = 1;
  size_t size;
  menu_type aMenu;

    size = sizeof(struct Ap_acces);

    if( (Ap_access_fptr=fopen("Ap_acces.dbf","r+b"))==NULL){
        if( (Ap_access_fptr=fopen("Ap_acces.dbf","w+b"))==NULL){
            perror("cannot open Ap_acces.dbf");
            exit(1);
        }
    }

    aMenu = menu_Open();

    fseek (Ap_access_fptr, 0, SEEK_SET);
    fread ((VOID *)&Ap_acces, size, 1, Ap_access_fptr);
    for (recno = 0; !feof (Ap_access_fptr); recno++){
        tmp = recno+1;
        menu_Printf (aMenu, "@p[%u,0]@f[%02X %-19.19s]", recno,
                     NULL, &menu_funcs, tmp, Ap_acces.AppName);
        fread ((VOID *)&Ap_acces, size, 1, Ap_access_fptr);
    }

    menu_Flush (aMenu);

    aSed = sed_Open (aMenu);
    sed_SetColors (aSed, 0x5b, 0x5b, 0x74);

    sed_SetBorder (aSed, bd_mouse);
    sed_SetBorderColor (aSed, 0x53);
    sed_SetBorderTitle (aSed, "Applications List");
    sed_SetPosition (aSed, 0, 50);
    sed_SetWidth (aSed, 28);
    sed_SetHeight (aSed, 11);
    sed_SetMouse (aSed, sedmou_Track);
    sed_SetBorderFeature (aSed, BD_MOVE | BD_RESIZE | BD_TOP);
    sed_SetSpecial (aSed, spc_List);
    sed_SetShadow (aSed, 1);
    sed_SetShadowAttr (aSed, 0x08);

    sed_Repaint(aSed);
}


//---------------------------------------------------------------------------

void MenuInitVar (void)
{
```

```
  int recno, fieldNo, mergeLength;
  char buf[24];
  char *merge;

    fseek (Menu_fptr, 0, SEEK_SET);
    fread((void *)&Menu, g_menuSize, 1, Menu_fptr);
    for (recno=0;!feof(Menu_fptr);recno++){
        fieldNo = sed_GetGridField (mSed, recno, 0);
        merge = sed_GetMerge (mSed, fieldNo);
        mergeLength = sed_GetMergeLen (mSed, fieldNo);
        sprintf (buf, "%02X %-19.19s", recno, Menu.MenuName);
        strncpy (merge, buf, mergeLength);
        fread((void *)&Menu, g_menuSize, 1, Menu_fptr);
    }
}



//---------------------------------------------------------------------------

void MMenuPrintf (int rec)
{
    menu_Printf (mMenu, "@p[%u,0]@f[%02X %-19.19s]", rec,
                 NULL, &gmenu_funcs, rec, Menu.MenuName);
}



//---------------------------------------------------------------------------

int ins_del_record (long rec, int insert)
{
  FILE *tempfp;
  long count, totrec;

    //Open temporary file for insert or delete operation
    tmpnam(tmpname);
    if ((tempfp = fopen(tmpname, "w+b"))==NULL){
        printf("Temporary file could not be created\n");
        return 0;
    }

    //Determine the total number of records in the file
    fseek (Menu_fptr, 0, SEEK_END);
    totrec = ftell(Menu_fptr) / g_menuSize;
    g_totMenuRecs = totrec;

    //put relevant contents after 'insert' or 'delete' into the temp file
    fseek(Menu_fptr, 0L, SEEK_SET);
    for (count = 0; count < totrec; count++){
        if (insert && (count == rec)){
            memset((void *) &Menu, 0, g_menuSize);
            fwrite((void *) &Menu, g_menuSize, 1, tempfp);
        }
        fread((void *) &Menu, g_menuSize, 1, Menu_fptr);
        if (insert || (count != rec)){
            fwrite((void *) &Menu, g_menuSize, 1, tempfp);
        }
    }

    //Close both files to allow for 'delete file' and 'rename file' operation
    fclose(tempfp);
    fclose(Menu_fptr);

    //Delete Menu.dbf
    if (remove("Menu.dbf") != 0){
        perror("unable to delete Menu.dbf file");
        printf("\ntemp file created with name: %s", tmpname);
        exit(1);
    }

    //Rename temporary file as Menu.dbf
    if (rename(tmpname, "Menu.dbf") != 0){
        printf("\ntemp file created with name: %s", tmpname);
        perror("unable to rename temp file");
        exit(1);
    }
```

```
    //Open the new Menu.dbf
    if( (Menu_fptr = fopen("Menu.dbf", "r+b")) == NULL){
        perror("cannot re-open Menu.dbf after delete operation");
        exit(1);
    }


    return 1;
}


//---------------------------------------------------------------------------

boolean spc_MyFields (sed_type sed, int scancode)
{
  int i,c, fieldNo, mergeLength;
  mev_struct mev;
  char buf[24], *merge;

    fieldNo =  sed_GetFieldNo(sed);
    if (fieldNo == 1) g_menuTitleChanged = 1;
    else{
        if (g_menuTitleChanged == 1){
            fieldNo = sed_GetGridField (mSed, tMenuID, 0);

            merge = sed_GetMerge (mSed, fieldNo);
            mergeLength = sed_GetMergeLen (mSed, fieldNo);
            strcpy (tMenuName, sed_GetMerge (sed, 1));
            sprintf (buf, "%02X %-19.19s", tMenuID, tMenuName);
            strncpy (merge, buf, mergeLength);
            sed_Repaint (mSed);
            g_menuTitleChanged = 0;
        }
    }

    switch (scancode){

        default: return (FALSE);

        case ESC:
            sed_SetBaton (sed, SED_ABORT);
            sed_ToggleExit (sed);
            break;

        case ENTER:
            if (fieldNo > 61){
                if (fieldNo == 64){
                    sed_SetNextWin(sed, aSed);
                    sed_ToggleExit (sed);
                    return (TRUE);
                }
                if (fieldNo == 65){
                    sed_SetNextWin(sed, mSed);
                    sed_ToggleExit (sed);
                    return (TRUE);
                }

                sed_SetBaton (sed, (fieldNo + 1));
                sed_ToggleExit (sed);
            }
            else sed_IncField (sed);
            break;

        case UP:
            if (sed_DecField (sed) == SED_STUCK)
                sed_GotoLastField (sed);
            break;

        case DOWN:
            if (sed_IncField (sed) == SED_STUCK)
                sed_GotoFirstField (sed);
            break;

        case PGDN:
            sed_SetBaton (sed, 63);
            sed_ToggleExit (sed);
            break;

        case PGUP:
```

```
            sed_SetBaton (sed, 64);
            sed_ToggleExit (sed);
            break;

        case MOU_CLICK:
            win_MouseCurrEvent(&mev);
            //check if right mouse button pressed
            if (mev_IsButton3Down(&mev)) i = 1;
            if (mev_IsButton2Down(&mev)) i = 1;
            if (i != 1){
                if (fieldNo == 64){
                    sed_SetNextWin(sed, aSed);
                    sed_ToggleExit (sed);
                    return (TRUE);
                }
                if (fieldNo == 65){
                    sed_SetNextWin(sed, mSed);
                    sed_ToggleExit (sed);
                    return (TRUE);
                }
                return (FALSE);
            }

        case FN2:
            if ( (fieldNo > 1) && (fieldNo < 62) ){
                c = (fieldNo-2)/3;
                tType[c] = Menu_choice.type;
                tID[c] = Menu_choice.ID;
                strcpy (tName[c], Menu_choice.ItemName);
                i = (c * 3) + 2;
                for (c=0; c<3; c++, i++) sed_RepaintField(sed, i);
            }
            break;

    }
    return (TRUE);
}


//---------------------------------------------------------------------------

boolean spc_List (sed_type sed, int scancode)
{
  int fieldNo;
  char *mergeString;

    switch (scancode){


        case ENTER:
            fieldNo = sed_GetFieldNo(sed);
            mergeString = sed_GetCurrMerge (sed);
            strcpy (Menu_choice.ItemName, mergeString + 3);
            if (sed == mSed){
                Menu_choice.type = 'M';
                Menu_choice.ID = sed_GetGridRow (sed, fieldNo);
            }
            else{
                Menu_choice.type = 'A';
                Menu_choice.ID = fieldNo + 1;
            }
            sed_SetNextWin(sed, medSed);
            sed_ToggleExit(sed);

            break;

        case MOU_CLICK:
            fieldNo = sed_GetFieldNo(sed);
            mergeString = sed_GetCurrMerge (sed);
            strcpy (Menu_choice.ItemName, mergeString + 3);
            if (sed == mSed){
                Menu_choice.type = 'M';
                Menu_choice.ID = sed_GetGridRow (sed, fieldNo);
            }
            else{
                Menu_choice.type = 'A';
                Menu_choice.ID = fieldNo + 1;
            }
```

```
            break;

    }

    return (FALSE);
}


//---------------------------------------------------------------------------

void SetMenuSedVar (int rec)
{
int i;

    fseek (Menu_fptr, rec * g_menuSize, SEEK_SET);
    fread((void *)&Menu, g_menuSize, 1, Menu_fptr);

    tMenuID = rec;
    strcpy(tMenuName, Menu.MenuName);
    tGoto = rec;
    tResetAppID = 0;

    for (i = 0; i < 20; i++){
        tType[i] = Menu.item[i].type;
        strcpy(tName[i], Menu.item[i].ItemName);
        tID[i] = Menu.item[i].ID;
    }
}


//---------------------------------------------------------------------------

void SetMenuFileRec (int rec)
{
int i;

    fseek (Menu_fptr, rec * g_menuSize, SEEK_SET);

    strcpy(Menu.MenuName, tMenuName);
    for (i = 0; i < 20; i++){
        Menu.item[i].type = tType[i];
        strcpy(Menu.item[i].ItemName, tName[i]);
        Menu.item[i].ID = tID[i];
    }

    fwrite((void *)&Menu, g_menuSize, 1, Menu_fptr);

}


//---------------------------------------------------------------------------

int Password_scr(void)
{
    menu_type    menu;
    sed_type    sed;
    int         ret, i, c;
    int            pAccess[NUM_OF_ACCESS_GROUPS];
    FILE        *pass_fptr;
    size_t      size;
    struct buf  {
                char group[VM_GROUP_NAME_LENGTH];
                char user[VM_USER_NAME_LENGTH];
                char password[VM_PASSWORD_LENGTH];
                }b[NUM_OF_ACCESS_GROUPS];

    //Open password file
    i = 0;
    if( (pass_fptr=fopen("Password.dbf","r+t"))==NULL){
        i = 1;
        if( (pass_fptr=fopen("Password.dbf","w+t"))==NULL){
            perror("cannot open Password.dbf");
            return 0;
        }
    }

    //Initialise screen variables
```

```
    size = sizeof (struct buf) * NUM_OF_ACCESS_GROUPS;
    memset ((void *)&b, 0, size);
    if (i ==0)    fread((void *)&b, size, 1, pass_fptr);

    menu = menu_Open();
    //Set up fields in the screen
    for (c = 0; c < NUM_OF_ACCESS_GROUPS; c++, i = c * 3){
        pAccess[c] = c;
        menu_Printf (menu, "@p[%u,3]Access:@p[%u,11]@fp[##]", i, i,
                     &pAccess[c], &hex_funcs);
        menu_Printf (menu, "@p[%u,17]Group:@p[%u,24]@f[@[%u,#]]", i, i,
                     b[c].group, &string_funcs, VM_GROUP_NAME_LENGTH-1);
        menu_Printf (menu, "@p[%u,48]User:@p[%u,54]@f[@[%u,#]]", i, i,
                     b[c].user, &string_funcs, VM_USER_NAME_LENGTH-1);
        menu_Printf (menu, "@p[%u,0]@f[@[%u,#]]", i+1,
                     b[c].password, &string_funcs, VM_PASSWORD_LENGTH-1);
        menu_Printf (menu, "@p[%u,0]@a[0x1c]@[%u,1]@a[0x17]", i+2,
                     VM_PASSWORD_LENGTH);
    }
    pAccess[NUM_OF_ACCESS_GROUPS-1] = 0xff;

    menu_Flush(menu);

    sed = sed_Open(menu);
    sed_SetColors(sed, 0x30, 0x17, 0x4f);

    sed_SetBorder(sed, bd_mouse2);
    sed_SetBorderColor(sed, 0x13);
    sed_SetBorderTitle(sed, "Password Screen");
    sed_SetPosition(sed, 1, 0);
    sed_SetHeight(sed, 20);
    sed_SetWidth(sed, 78);
    sed_SetMouse(sed, sedmou_Track);

    sed_Repaint(sed);
    ret = sed_Go(sed);

    if (ret == 0){
        c = pop_Menu ("", save_on_exit,-1,-1,3,18,0x4f,0,bd_2);
        if (c ==1){
            fseek (pass_fptr, 0, SEEK_SET);
            fwrite ((void *)&b, size, 1, pass_fptr);
        }
    }

    sed_Close(sed);
    fclose (pass_fptr);

    return(ret);
}
```

APPENDIX C2

B A T C H . C

BATCH.C is the source file for the program that sorts the Virtual Machine (VM) database file AP_BATCH.DBF and creates an index file called AP_BATCH.NDX. The file is sorted on the application ID firstly and then on the server name. The program is designed for a maximum of 1000 applications duplicated on a maximum of six file servers. The index file keeps track of the position of an application in the sorted AP_BATCH.DBF file and how many consecutive records have information on that application. The record number of the index file (starting from one) indicates the application ID.

C2.1 Program Listing

```
/*
                                File name: BATCH.C

 Compiler Model COMPACT
 C++ options turned to "CPP extension only"


 Project File should include the following files:
    batch.c

    This is the source code for the application to sort the database
    Ap_batch.dbf according firstly to AppID and then according to
    server name.


    By S.J. Isaac

    Last Revised: 27 April 1993
*/


#include <stdio.h>
#include <string.h>
#include <stdlib.h>


#define MAX_APPLICATIONS 1000   // Max of 1000 applications
#define EMPTY_VALUE      0x00
#define MAX_ENTRY        6000   // Max. 6000 entries because of
                                // 64K array size limitation


typedef unsigned int WORD;



//_____________________________________________________________________________
//                  Global variable declaration

struct Ap_batch_record
    {
    char Server[3];
    WORD AppID;
    int  Rec;
    } Ap_batch[MAX_ENTRY];  //====== Max. 1000 applications
                            //====== duplicated on a max of 6 servers



//_____________________________________________________________________________
//                  Declaration of routines

int sort_function( const void *a, const void *b);
void Ap_batch_index(int total);



//_____________________________________________________________________________

void main (void)
{
  FILE *Ap_batch_fptr, *tmp;
  int count, max_record;
  char tempname[13];
  size_t size;
  struct record
        {
        char Server[3];
        WORD AppID;
        char BatchFile[13];
        } record;               // this is the storage area for one record
                                //    of the AP_BATCH database
```

```
// Open AP_BATCH.DBF for reading

printf("\nstatus - opening ap_batch.dbf");
if ( (Ap_batch_fptr = fopen("Ap_batch.dbf","rb")) == NULL){
    perror("cannot open Ap_batch.dbf");
    exit(1);
}

size = sizeof (struct record);


// place salient information in the array Ap_batch[]

printf("\nstatus - setting up array Ap_batch");
fread((void *)&record,size,1,Ap_batch_fptr);
for (count=0; (count<6000) && (!feof(Ap_batch_fptr));count++){
    strcpy(Ap_batch[count].Server, record.Server);
    Ap_batch[count].AppID = record.AppID;
    Ap_batch[count].Rec = count;
    fread((void *)&record,size,1,Ap_batch_fptr);
}
max_record = count;  //===== number of valid records in the array



// sort on AppID and then server name

printf("\nstatus - sorting");
qsort ( (void *)&Ap_batch[0], (size_t)max_record,
        sizeof (struct Ap_batch_record), sort_function);



// create a temporary file to hold sorted records

printf("\nstatus - opening temporary file");
tmpnam(tempname);
if ( (tmp=fopen(tempname,"w+b"))==NULL){
    perror("temporary file could not be created");
    exit(1);
}


// place sorted records in temporary file

printf("\nstatus - placing sorted records in temporary file");
for (count=0; count < max_record; count++){
    fseek(Ap_batch_fptr, (long)(size*Ap_batch[count].Rec), SEEK_SET);
    fread( (void *)&record, size, 1, Ap_batch_fptr);
    fwrite( (void *)&record, size, 1, tmp);
}

fclose(tmp);
fclose(Ap_batch_fptr);


// delete old database file

printf("\nstatus - deleting old ap_batch.dbf file");
if (remove("Ap_batch.dbf") != 0){
    perror("unable to delete Ap_batch.dbf file");
    printf("\ntemp file created with name: %s",tempname);
    exit(1);
}


// rename temporary file as AP_BATCH.DBF

printf("\nstatus - renaming temporary file as ap_batch.dbf");
if (rename(tempname,"Ap_batch.dbf") != 0){
    perror("unable to rename temp file");
    printf("\ntemp file created with name: %s",tempname);
    exit(1);
}


// set up ap_batch index
```

```
    printf("\nstatus - setting up ap_batch index");
    Ap_batch_index(max_record);

}


//_____________________________________________________________________

int sort_function( const void *a, const void *b)
{
  struct Ap_batch_record *a1,*b1;
    a1=(struct Ap_batch_record *)a;
    b1=(struct Ap_batch_record *)b;


    // sort on AppID

    if (a1->AppID > b1->AppID)  return 1;
    if (a1->AppID < b1->AppID)  return -1;


    // if same AppID then sort on server name

    if (a1->AppID == b1->AppID) return (strcmp(a1->Server, b1->Server));

    return 0;
}


//_____________________________________________________________________

void Ap_batch_index(int total)
{
  FILE *Ap_batchndx_fptr;
  int recpos, numrecs;
  WORD ID;
  size_t size;

  struct Ap_batch_index
        {
        int  record_position;
        int     number_of_records;
        } Ap_batch_index;


    // Open AP_BATCH.NDX for writing

    if ( (Ap_batchndx_fptr = fopen("Ap_batch.ndx","wb")) == NULL){
        perror("cannot open Ap_batch.ndx");
        exit(1);
    }


    // Initialize AP_BATCH.NDX with EMPTY_VALUE

    size = sizeof (struct Ap_batch_index);

    for(recpos=0, ID=1; recpos < total; recpos++, ID++){


        // fill with EMPTY_VALUE those AppIDs that are not on any server

        for (;(Ap_batch[recpos].AppID != ID)&&(ID < MAX_APPLICATIONS);ID++){
            memset((void *)&Ap_batch_index, EMPTY_VALUE, size);
            fwrite((void *)&Ap_batch_index, size, 1, Ap_batchndx_fptr);
        }
        if (ID >= MAX_APPLICATIONS){
            puts("application ID has exceeded max. num of applications of");
            printf ("%i in index file",MAX_APPLICATIONS);
            exit(1);
        }

        Ap_batch_index.record_position = recpos;


        // count how many records have the same AppID

        recpos++;
```

```
        for (numrecs=1; (Ap_batch[recpos].AppID == ID) && (recpos < total);
             recpos++,numrecs++);
        recpos--;

        Ap_batch_index.number_of_records = numrecs;
        fwrite ((void *)&Ap_batch_index, size, 1, Ap_batchndx_fptr);
    }

    fclose(Ap_batchndx_fptr);
    printf("\nfinished setting up Ap_batch index");
}
```

APPENDIX C3

D B . C

DB.C is the source file for the Virtual Machine (VM) database server program. When starting the database server, the name of a file server can be supplied as well. If the file server name is supplied then the database server program will attempt to load the database files from the specified file server. (In this case, the password for a user called VM_DATABASE on the specified file server will be asked for.) After loading the database files into memory, the server program will log out of the file server. If the file server name is not supplied then the database server will look for the files in the directory from which the program was started. On starting up, the program will also ask for a password (eight characters maximum) to terminate the database server program.

To help describe the program please consider Figure C3.1. When the database server receives a request, it is placed on a queue and serviced in a first-come-first-served (FIFO) basis. The four types of requests that are currently defined are also shown in Figure C3.1. When servicing a request, the server program passes control to the relevant module responsible for answering the request type. Finally, the response is sent to the requesting PC.

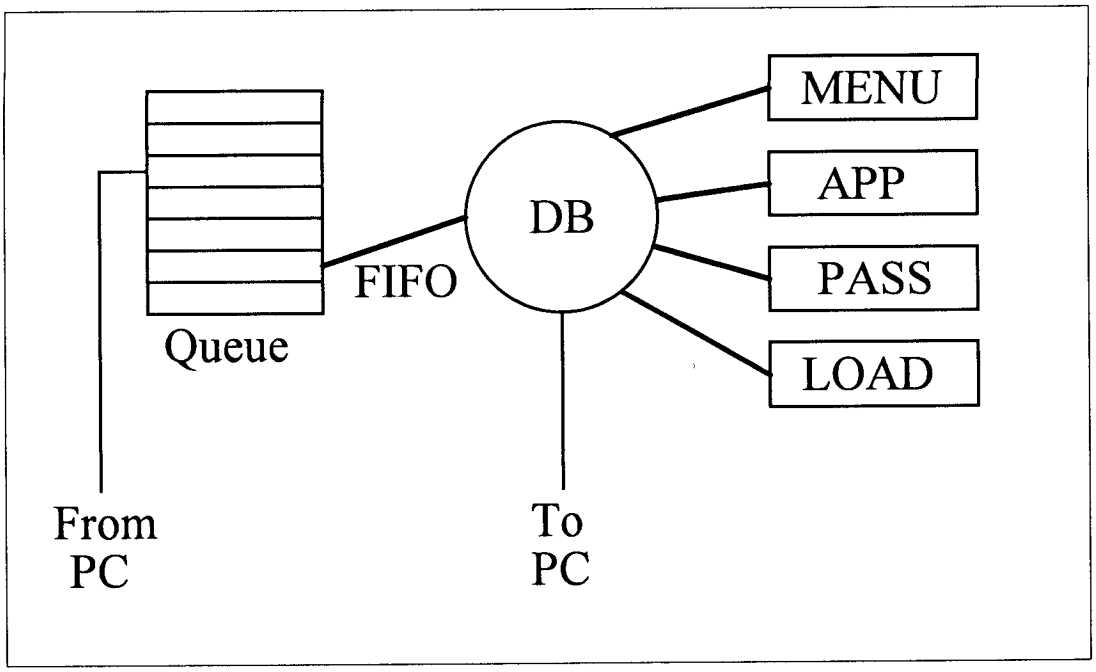

Figure C3.1: VM Database Server Program

C3.1 Description of the Code

The code that composes the DB.C program is described under this section. The description starts with an explanation of the main() function and then some important functions called by main(). The significant parts of the code is presented in pseudocode format.

The following significant flags (as global variables) are used in the program:

```
BYTE g_requestQueuePtr = 0;      /* shows the position in the request queue that
                                    will be filled by the next request          */
BYTE g_serviceQueuePtr = 0;      /* shows the position in the service queue that
                                    is being serviced                           */
BYTE g_queueCount = 0;           /* number of items in the request queue */
int  g_firstLoadingFlag = 0;     /* 1 = not first loading  */
int  g_successfulLoadFlag = 0;   /* 1 = not successful */
int  g_loadDBFlag = 0;           /* 1 = load new database   */
BYTE g_ptr;                      /* utility global pointer */
int  g_listenForPacket = 1;      /* 1 = server will receive requests */
int  g_terminateFlag = 0;        /* 1 = server should be terminated */
int  g_login;                    /* 1 = problem with login */
```

C3.1.1 Function — main()

The main() function receives an argument from the command line specifying the name of the server where the VM database files to be loaded are located. The usage and the explanation of the arguments is given in the function DisplayUsage(). The function main() does not return any values at all.

The pseudocode that describes the operation of main() is presented below.

```
void main (int argc, char *argv[])
{
    Load database files into memory

    Advertise the presence of this server on the network every 60 seconds;

    // Start Server Process

    while (1){

        if ( Something in the queue){
            service request and send reply to requesting PC
        }
        else {
            if (load database flag has been set) {
                load new database
            }
            if (terminate flag has been set) break;
        }
        if (any key on the keyboard has been pressed){
            check for correct password within specified time-out value
            if (correct password){
                set terminate flag
            }
        }
    }
}
```

C3.1.2 Function — InitialiseDBServerDetails()

This function does not receive any parameters or return any values.The pseudocode that describes the operation of InitialiseDBServerDetails() is presented below.

```
void InitialiseDBServerDetails (void)
{
    Make database server's name from its network address

    Set the type and socket number and open the socket
}
```

C3.1.3 Function — LoadDatabaseFiles()

This function does not receive any parameters or return any values.The pseudocode that describes the operation of LoadDatabaseFiles() is presented below.

```
void LoadDatabaseFiles (void)
{
    Login to File server if necessary
    Copy database files into memory
    if (error occurs while loading files){
        if (first time to load database) exit (1);

        set a flag to indicate an unsuccessful loading
    }

}
```

C3.1.4 Function — BeginAdvert()

This function does not receive any parameters or return any values.The pseudocode that describes the operation of BeginAdvert() is presented below.

```
void BeginAdvert (void)
{
    if (g_loadDBFlag == 0) // i.e. don't load new database
        Advertise service with dynamically allocated socket number
    else
        Advertise service with previously allocated socket number
}
```

C3.1.5 Function — EndAdvert()

This function does not receive any parameters or return any values.The pseudocode that describes the operation of EndAdvert() is presented below.

```
void EndAdvert (void)
{
    Shut down the service advertising
    Close the advertising socket
}
```

C3.1.6 Functions — Set_EcbSendWs() and Set_EcbReceiveWs()

These function prepares the Event Control Block for the send to and receive from PC operations.

C3.1.7 Function — SendToWs()

This function does not return any values. It receives one parameter; the position in the queue that has been serviced. The pseudocode that describes the operation of SendToWs() is presented below.

```
void SendToWs (void)
{
    Set the address to which the information should be sent
    Set the address of the first bridge (if any) the information has to cross
    Do the actual sending
}
```

C3.1.8 Function — Esr_ReceiveWs()

This function is an interrupt service routine that is called whenever the PC running the DB program receives something from a PC. Its function is to update the received queue count and pointer.

Esr_ReceiveWs() does not receive any parameters or return any values.

The pseudocode that describes the operation of Esr_ReceiveWs() is presented below.

```
void far Esr_ReceiveWs (void)
{
    Setup the DS register
    /* because the following conditions are true
       when this routine is called:
            • A far pointer to the ECB is in the
              ES:SI register pair
            • The DS register does not point to the
              program's data area
    */

    increment queue count and request queue position

    if (queue full or listeneing to be disabled)
        return
    else
        listen for next request to be stored in the next queue position
}
```

C3.1.9 Function — ServiceTheRequest()

This function does not receive any parameters. The return value indicates whether any information should be sent to the PC (0 = do not send). The pseudocode that describes the operation of ServiceTheRequest() is presented below.

```
void ServiceTheRequest (void)
{
    Set the request number to be sent to the PC
    if (the request is to load a new database){
        set g_fileServerName and g_password
        g_loadDBFlag = 1;
        g_listenForPacket = 0;
        g_ptr = g_serviceQueuePtr; // for future use in SendToWs()
        close the advertising socket
        return 0;
    }
    else perform other request
    if (request could not be processed) set response type to 0xFF
    return 1;
}
```

C3.1.10 Function — GetMenu()

This function does not receive any parameters or return any values. It is called by the ServiceTheRequest() routine. The pseudocode that describes the operation of GetMenu() is presented below.

```
void GetMenu (void)
{
    if (requested menuID is valid){
        get appropriate menu and set the response type to 0
    }
    else set response type to 1
}
```

C3.1.11 Function — GetApplication()

This function does not receive any parameters or return any values. It is called by the ServiceTheRequest() routine. The pseudocode that describes the operation of GetApplication() is presented below.

```
void GetApplication (void)
{
    if (requested appID is invalid) {
        set response type to 1 if appID is less than available appIDs
        set response type to 2 if appID is greater than available appIDs
        return;
    }
    get the number of records and the starting position containing the requested
    information in the global array containing information about batch files from
    the global array containing index information.

    get the access rights for the requested application
}
```

C3.1.12 Function — GetPassword()

This function does not receive any parameters or return any values. It is called by the ServiceTheRequest() routine. The requested password is obtained from the request buffer and a number of passwords from (and including) the requested password is placed in the response buffer. The pseudocode that describes the operation of GetPassword() is presented below.

```
void GetPassword (void)
{
  int passStep; //number of passwords (and related information) to return

    Get the requested VM group
    if (the number of remaining groups after requested one < passStep) {
        get remaining password information
    }
    else get passStep number of passwords from requested VM group
}
```

C3.1.13 Function — CopyFileToRAM()

This function copies a disk file into the RAM of PC and it receives three parameters:

diskFile	the name of the disk file to be copied into RAM
tBuf	the buffer in RAM that the file is to be copied into
size	the size of the disk file

If the operation is successful the function returns zero (else, it returns one).

C3.1.14 Function — CheckPass()

This routine checks a supplied password to ensure that only authorised persons terminate the VM database server. The password has to be supplied within five seconds.

C3.2 Program Listing

```
/*
```

File name: DB.C

```
Compiler Model COMPACT
C++ options turned to "CPP extension only"


Project File should include the following files:
    db.c
    netw.c
    cnit.lib
```

```
    Database software that coordinates with the workstation software to
    present a virtual machine environment for the workstation

    By S.J. Isaac

    Last Revised: 18 May 1993
```

```
*/


#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>
#include <conio.h>
#include <string.h>
#include <time.h>
#include <sap.h>
#include <nit.h>

#include "h:\c\r\h\netw.h"
#include "h:\c\r\h\uvm.h"


#define MAX_QUEUE 25

//#define VMDEBUG


//_____________________________________________________________________________
//                        Global variable declaration

struct
    {
    BYTE requestType;
    WORD requestNumber;
    BYTE retry;
    char data[REQUEST_SIZE];
    }g_receiveWs[MAX_QUEUE];

struct
    {
    BYTE responseType;
    WORD requestNumber;
    char data[RESPONSE_SIZE];
    }g_sendWs;

struct pass
    {
    char group[VM_GROUP_NAME_LENGTH];
    char name[VM_USER_NAME_LENGTH];
    char password[VM_PASSWORD_LENGTH];
    }g_pass[NUM_OF_ACCESS_GROUPS];

struct g_menuChoice
    {
    BYTE type;
    char name[MENU_NAME_LENGTH];
```

```
    WORD ID;
    };

struct menu      // menu database to be loaded from disk
    {
    char name[MENU_NAME_LENGTH];
    struct g_menuChoice choice[MENU_ITEMS];
    }far *g_menu;

struct apBatchNdx
    {
    int recPos;
    int numRecs;
    }far *g_apBatchNdx;

struct apBatch
    {
    char server[SERVER_NAME_LENGTH];
    WORD appID;
    char batchFile[13];
    }far *g_apBatch;

struct apAccess
    {
    char appName[APPLICATION_NAME_LENGTH];
    BYTE access;
    BYTE OS;
    WORD hardware;
    }far *g_apAccess;


ECB         g_ecbSendWs;
ECB         g_ecbReceiveWs[MAX_QUEUE];
IPXHeader   g_ipxheaderSendWs;
IPXHeader   g_ipxheaderReceiveWs[MAX_QUEUE];


BYTE g_advertisingSocket[2];
BYTE g_requestQueuePtr = 0;
BYTE g_serviceQueuePtr = 0;
BYTE g_queueCount = 0;
WORD g_dbServerSocket;
char g_dbServerName[13];
int  g_dbServerType;
char g_fileServerName[SERVER_NAME_LENGTH] = {""};
char g_serverDrive[3] = {""};
int  g_firstLoadingFlag = 0;
int  g_successfulLoadFlag = 0;
int  g_loadDBFlag = 0;
char g_diskFile[17];
char g_password[9];
char g_exitPassword[9];
BYTE g_ptr;
int  g_listenForPacket = 1;
int  g_terminateFlag = 0;
int  g_login;
int  g_maxMenuID;
int  g_maxAppID;


//_____________________________________________________________________________
//                        Declaration of routines

void InitialiseDBServerDetails (void);
void LoadDatabaseFiles (void);
int  ServiceTheRequest(void);
void SendToWs (BYTE ptr);
void Set_EcbSendWs (void);
void Set_EcbReceiveWs (void);
void far Esr_ReceiveWs (void);
void BeginAdvert (void);
void EndAdvert (void);
void GetMenu (void);
void GetApplication (void);
void GetPassword (void);
void DisplayUsage (void);
int  CopyFileToRAM (char *fileName, void far **tBuf, size_t *size);
char *Name (char *filename);
```

```
int  LogFS (void);
int  CheckPass (void);
void MakeServerName (char *sName, char *add);
void InitialiseCommunication (void);



//_____________________________________________________________________

void main (int argc, char *argv[])
{
  int ccode;

    puts ("");

    if (argc > 2) DisplayUsage();


    // Check whether files should be loaded from a server

    if (argc == 2){
        if (strcmp(argv[1], "?") == 0)  DisplayUsage();
        strcpy (g_fileServerName, argv[1]);
        strupr (g_fileServerName);
        strcpy (g_password, getpass("File server's password: "));
        strupr (g_password);
    }


    // Initialise System

    InitialiseDBServerDetails ();
    LoadDatabaseFiles();

    strcpy (g_exitPassword, getpass("Password to exit database server: "));
    strupr (g_exitPassword);

    BeginAdvert();
    InitialiseCommunication();


    // Start Server Process

    while (1){

        IPXRelinquishControl();

        #ifdef VMDEBUG
            printf ("In loop: Count=%x ",g_queueCount);
            printf ("sLoad=%x ", g_successfulLoadFlag);
            printf ("listen=%x ", g_listenForPacket);
            printf ("loadDB=%x ", g_loadDBFlag);
            printf ("terminate=%x ", g_terminateFlag);
            printf ("fLoad=%x\n", g_firstLoadingFlag);
        #endif

        if ( g_queueCount ){

            // Something in the queue, service it.

            if (ServiceTheRequest())    SendToWs (g_serviceQueuePtr);

            g_serviceQueuePtr = (g_serviceQueuePtr + 1)%MAX_QUEUE;
            g_queueCount--;

            if ( g_queueCount == (MAX_QUEUE-1) )
                IPXListenForPacket ( &g_ecbReceiveWs[ g_requestQueuePtr ] );

        }
        else {


            // Nothing in queue. Anything else to do?

            if (g_loadDBFlag == 1){


                // Load new database files from server
```

```
                LoadDatabaseFiles();
                if (g_login == 1) g_sendWs.responseType = 2;
                else g_sendWs.responseType = g_successfulLoadFlag;
                g_sendWs.requestNumber = g_receiveWs[g_ptr].requestNumber;
                SendToWs (g_ptr);
                if (g_successfulLoadFlag == 1) {
                    EndAdvert ();
                    IPXCloseSocket( g_dbServerSocket );
                    puts ("\nExiting ...");
                    exit (1);
                }
                else {
                    g_loadDBFlag = 0;
                    g_listenForPacket = 1;
                    puts ("Reopening advertising socket");
                    ccode = IPXOpenSocket (g_advertisingSocket,
                                           (BYTE) SHORT_LIVED );
                    if ( ccode ){
                        printf("IPXOpenSocket error = %i\n", ccode);
                    }

                    puts ("Now advertising services ...");
                    InitialiseCommunication();
                }
            }

            if (g_terminateFlag == 1) break;
        }


        // Check for key press

        if (g_terminateFlag == 0){
            if (CheckPass () == 0){
                EndAdvert ();
                g_listenForPacket = 0;
                g_terminateFlag = 1;
            }
        }
    }


    IPXCloseSocket( g_dbServerSocket );
    puts ("\nExiting ...");
}


//_____________________________________________________________________________

void InitialiseDBServerDetails (void)
{
  FILE *fp;
  BYTE networkAddress[10];
  int ccode;

    clrscr();
    if (IPXInitialize()){
        puts ("IPX is not installed !!");
        exit (1);
    }


    // Get the net address of this PC (to be used as the DB server's name)

    IPXGetInternetworkAddress (networkAddress);


    // Set the name of the Database server

    MakeServerName (g_dbServerName, networkAddress);

    // Set the type and socket number of the database server

    g_dbServerType = IntSwap ((int)DB_SERVER_TYPE);
    g_dbServerSocket = (WORD) IntSwap ((int) DB_SERVER_SOCKET);


    // Open Database server socket
```

```
    ccode = IPXOpenSocket ( (BYTE *) &g_dbServerSocket, (BYTE) SHORT_LIVED );
    if ( ccode ){
        printf("    IPXOpenSocket error = %i\n", ccode);
        puts ("    NOTE");
        puts ("    ----");
        puts ("         254 = socket table full");
        puts ("         255 = socket already open\n");
        exit(1);
    }
}


//_____________________________________________________________________

void LoadDatabaseFiles (void)
{
  int ccode = 0;
  size_t size;
  FILE *passFilePtr;


    // Login to File server if necessary

    if (g_fileServerName[0] != '\0') {
        ccode = LogFS();
        if (ccode != 0) {
            puts ("Could not load any database files");
            if (g_firstLoadingFlag == 0) exit (1);
            puts ("Database has not been changed... Restarting old one.");
            g_successfulLoadFlag = 0;
            return;
        }
        else puts ("Successful Login.");
    }


    // Open PASSWORD.DBF file and load into memory

    if ((passFilePtr = fopen(Name("PASSWORD.DBF"),"rt")) == NULL){
        perror("cannot open PASSWORD.DBF");
        ccode = 1;
    }
    else {
        fread ( (void *) &g_pass, sizeof(struct pass),
                NUM_OF_ACCESS_GROUPS, passFilePtr);
        fclose (passFilePtr);
        printf ("Now copying %s to RAM\n", Name("PASSWORD.DBF"));
    }


    // Load other files into memory and set max. menu & app IDs

    ccode += CopyFileToRAM (Name("MENU.DBF"),
                            (void far *)&g_menu,
                            &size);
    g_maxMenuID = size / sizeof(struct menu);
    if (g_maxMenuID > 0) g_maxMenuID--;  //because menuID starts from zero

    ccode += CopyFileToRAM (Name("AP_BATCH.NDX"),
                            (void far *)&g_apBatchNdx,
                            &size);
    ccode += CopyFileToRAM (Name("AP_BATCH.DBF"),
                            (void far *)&g_apBatch,
                            &size);
    ccode += CopyFileToRAM (Name("AP_ACCES.DBF"),
                            (void far *)&g_apAccess,
                            &size);
    g_maxAppID = size / sizeof(struct apAccess);


    // Are all files loaded?

    if (ccode != 0){
        puts ("Could not load all the database files");
        if (g_firstLoadingFlag == 0) {
            puts ("\nExiting...");
            exit (1);
        }
```

```
        g_successfulLoadFlag = 1;
    }
    else puts ("All the database files have been loaded.");

    g_successfulLoadFlag = 0;
    g_firstLoadingFlag = 1;
    if (g_fileServerName[0] != '\0') {
        Logout();
        printf ("Logout from file server %s.\n", g_fileServerName);
    }

}


//_____________________________________________________________________________

void BeginAdvert (void)
{
    if (g_loadDBFlag == 0)  *(WORD *)&g_advertisingSocket[0]=0x00;
    AdvertiseService ( g_dbServerType, g_dbServerName, g_advertisingSocket );
    puts ("The Database Server is now advertising its services ...");
}


//_____________________________________________________________________________

void EndAdvert (void)
{
  int ccode;

    puts ("Cancelling Database Server's advert ...");
    ccode = ShutdownSAP();
    if (ccode)  puts ("Had problems stopping advertising.");
    else puts ("Database Server's advert has been cancelled.");
    IPXCloseSocket( *(WORD *)&g_advertisingSocket );
}


//_____________________________________________________________________________

void Set_EcbSendWs (void)
{
    memset (&g_ecbSendWs, 0, sizeof(ECB));
    g_ecbSendWs.ESRAddress = NULL;
    g_ecbSendWs.socketNumber = g_dbServerSocket;
    g_ecbSendWs.fragmentCount = 2;
    g_ecbSendWs.fragmentDescriptor[0].address = &g_ipxheaderSendWs;
    g_ecbSendWs.fragmentDescriptor[0].size = sizeof(IPXHeader);
    g_ecbSendWs.fragmentDescriptor[1].address = &g_sendWs;
    g_ecbSendWs.fragmentDescriptor[1].size = sizeof(g_sendWs);
}


//_____________________________________________________________________________

void Set_EcbReceiveWs (void)
{
  BYTE index;
    for (index = 0; index < MAX_QUEUE; index++){
        memset (&g_ecbReceiveWs[index], 0, sizeof(ECB));
        g_ecbReceiveWs[index].ESRAddress = Esr_ReceiveWs;
        g_ecbReceiveWs[index].socketNumber = *(WORD *)&g_advertisingSocket;
        g_ecbReceiveWs[index].fragmentCount = 2;
        g_ecbReceiveWs[index].fragmentDescriptor[0].address =
              &g_ipxheaderReceiveWs[index];
        g_ecbReceiveWs[index].fragmentDescriptor[0].size =
              sizeof(IPXHeader);
        g_ecbReceiveWs[index].fragmentDescriptor[1].address =
              &g_receiveWs[index];
        g_ecbReceiveWs[index].fragmentDescriptor[1].size =
              sizeof(g_receiveWs[0]);
    }
}
```

```
//_____________________________________________________________________

void SendToWs (BYTE ptr)
{

    // Set the destination address

    memcpy((void *)&g_ipxheaderSendWs.destination,
           (void *)&g_ipxheaderReceiveWs[ptr].source,
           12);


    // Set the first bridge address

    memcpy((void *)&g_ecbSendWs.immediateAddress,
           (void *)&g_ecbReceiveWs[ptr].immediateAddress,
           6);


    // Do the actual sending

    IPXSendPacket (&g_ecbSendWs);
}



//_____________________________________________________________________

void far Esr_ReceiveWs (void)
{
    _AX = _ES;
    _DS = _AX;


    // Received something so increment queue count and request queue position

    g_queueCount++;
    g_requestQueuePtr = (g_requestQueuePtr + 1)%MAX_QUEUE;


    // If queue full or listening to be disabled, stop receiving and leave

    if (g_queueCount == MAX_QUEUE)      return;

    if (g_listenForPacket == 0)      return;


    // listen for next request

    IPXListenForPacket (&g_ecbReceiveWs[ g_requestQueuePtr ]);
}



//_____________________________________________________________________

int ServiceTheRequest(void)
{
    #ifdef VMDEBUG
        printf ("\nretry = %u ",g_receiveWs[ g_serviceQueuePtr ].retry);
        printf ("serviceQ = %i ",g_serviceQueuePtr);
        printf ("requestQ = %i ",g_requestQueuePtr);
        printf ("Qcount = %u   ",g_queueCount);
        prt (&g_ipxheaderReceiveWs[g_serviceQueuePtr].source, 12);
        printf ("\nrequestType = %x",
                g_receiveWs[g_serviceQueuePtr].requestType );
    #endif

    g_sendWs.requestNumber = g_receiveWs[ g_serviceQueuePtr ].requestNumber;

    switch ( g_receiveWs[ g_serviceQueuePtr ].requestType ) {

        case MENU_RECORD:

            GetMenu();
            break;

        case APPLICATION_RECORD:

            GetApplication();
```

```
            break;

        case PASSWORD_RECORD:

            GetPassword();
            g_sendWs.responseType = 0;
            break;

        case LOAD_DATABASE:

            memset ( g_fileServerName, 0, SERVER_NAME_LENGTH);
            strcpy(g_fileServerName,g_receiveWs[g_serviceQueuePtr].data);
            strupr(g_fileServerName);

            memset (g_password, 0, 9);
            strcpy(g_password,
                   &g_receiveWs[g_serviceQueuePtr].data[SERVER_NAME_LENGTH]);
            strupr(g_password);

            g_loadDBFlag = 1;
            g_listenForPacket = 0;
            g_ptr = g_serviceQueuePtr;
            puts("\nClosing advertising socket");
            IPXCloseSocket( *(WORD *)&g_advertisingSocket );
            return 0;

        default:  // could not process request

            g_sendWs.responseType = 0xFF;

    }

    return 1;

}


//_____________________________________________________________________________

void GetMenu (void)
{
  WORD menuID;

    menuID = *(WORD *) &g_receiveWs[ g_serviceQueuePtr ].data[0];
    #ifdef VMDEBUG
        printf ("menuID = %x\n", menuID);
    #endif

    if (menuID <= g_maxMenuID){
        memcpy(g_sendWs.data, &g_menu[menuID], sizeof (struct menu));
        g_sendWs.responseType = 0;

    }
    else g_sendWs.responseType = 1;

}


//_____________________________________________________________________________

void GetApplication (void)
{
  WORD appID;
  int num;
  int pos;

    appID = *(WORD *)&g_receiveWs[ g_serviceQueuePtr ].data[0];
    #ifdef VMDEBUG
        printf ("appID = %x ", appID);
    #endif


    if (appID == 0) {
        g_sendWs.responseType = 1;
        return;
    }
```

```
    //The first appID is 0x0001 but is at position zero in the database array

    appID--;  //for reason given in previous comment

    if (appID > g_maxAppID){
        g_sendWs.responseType = 2;
        return;
    }

    num = g_apBatchNdx[appID].numRecs;
    pos = g_apBatchNdx[appID].recPos;
    memcpy (g_sendWs.data, &num, sizeof(int));
    g_sendWs.data[2] = g_apAccess[appID].access;
    memcpy (&g_sendWs.data[3], &g_apBatch[pos], num*sizeof(struct apBatch));
    #ifdef VMDEBUG
        printf ("batchfile is %s\n", g_apBatch[pos].batchFile);
    #endif

}


//_____________________________________________________________________________

void GetPassword (void)
{
  int rec, size, num;
  int passStep = RESPONSE_SIZE/sizeof(struct pass);

    rec = *(int *)&g_receiveWs[ g_serviceQueuePtr ].data[0];
    num = (NUM_OF_ACCESS_GROUPS - rec);

    if (num < passStep)
        size = sizeof (struct pass) * num;
    else
        size = sizeof (struct pass) * passStep;

    memcpy (g_sendWs.data, &g_pass[rec], size);

}


//_____________________________________________________________________________

void DisplayUsage (void)
{
    puts   ("-----------------------------------------------------");
    puts   ("              Database Server Installation");
    puts   ("-----------------------------------------------------");

    puts ("\nsyntax: DB [<Server Name>]");

    puts ("\n Server Name = server where database files are located");

    puts ("\n   If the server name is not specified,");
    puts   ("   the program will attempt to load database");
    puts   ("   files from the current directory");
    puts   ("-----------------------------------------------------");
    exit (1);
}


//_____________________________________________________________________________

int CopyFileToRAM (char *diskFile, void far **tBuf, size_t *size)
{
  FILE *diskFPtr;


    // Open file on disk (hard disk, floppy or server disk) for reading

    if ((diskFPtr = fopen(diskFile,"rb")) == NULL){
        perror ("cannot open diskFile for reading");
        printf ("file = %s\n", diskFile);
        return (1);
    }
```

```
    // determine size of disk file

    fseek (diskFPtr, 0L, SEEK_END);
    *size = (size_t) ftell (diskFPtr);
    fseek (diskFPtr, 0L, SEEK_SET);


    // create space for file transfer buffer in memory

    if ((tBuf[0] = farmalloc (*size)) == NULL){
        puts ("Not enough memory to allocate file transfer buffer");
        printf ("for file %s\n", diskFile);
        return (1);
    }


    // load disk file into memory

    printf ("Now copying %s to RAM.\n", diskFile);
    fread ((void far *)tBuf[0], *size, 1, diskFPtr);


    // return

    fclose (diskFPtr);
    return (0);
}



//_____________________________________________________________________________
/*                          Name
                            ------

    This routine determines whether a file should be obtained from a local
    drive or from the file server's hard disk. If from the server, the drive
    letter is prepended to the file name
*/

char *Name (char *filename)
{
    if (g_fileServerName[0] == '\0') strcpy (g_diskFile, filename);
    else {
        strcpy (g_diskFile, "S:");
        strcat (g_diskFile, filename);
    }

    return g_diskFile;
}



//_____________________________________________________________________________
/*                          LogFS
                            -------

   This routine logs into the file server specified by g_fileServerName
   and maps drive S: to the database directory
*/

int LogFS (void)
{
  WORD connectionID, oldConnectionID;
  int ccode;
  char path[17];
  char buf[10] = {"SYS:VM_DB"};
  char drive[2] = {"S"};


    printf ("Attempting Login to file server %s...\n",g_fileServerName);
    oldConnectionID = GetPreferredConnectionID ();

    ccode = AttachToFileServer (g_fileServerName, &connectionID);
    if ( (ccode!= 0) && (ccode != 0xF8) ) {
        Logout();
        g_login = 1;
        printf ("Error %d attaching to file server \n", ccode);
        return 1;
    }
```

```
    SetPreferredConnectionID (connectionID);

    ccode = LoginToFileServer ("VM_DATABASE", OT_USER, g_password);
    if (ccode!= 0) {
        Logout();
        g_login = 1;
        printf ("Error %d logging into file server \n", ccode);
        return 1;
    }


    // Map drive S: to the directory

    strcpy (path, g_fileServerName);
    strcat (path, "/");
    strcat (path, buf);

    ccode = MapDriveUsingString ("ADD", drive, path);
    if (ccode!= 0) {
        Logout();
        g_login = 1;
        printf ("Error %d mapping drive S: to SYS:VM_DB \n", ccode);
        return 1;
    }

    g_login = 0;
    return 0;
}


//_____________________________________________________________________________

int CheckPass (void)
{
  static char a[9] = "";
  static int i =0;
  static time_t start, now;

    if (kbhit()){

        if (i == 0) {
            time (&start);
            memset (a, 0, 9);
            puts ("You have 5 seconds to type in password to exit");
        }

        a[i] = getch();

        i++;

        if (a[i-1] == 27) {
            i = 0;
            puts ("ESC key hit - Resetting");
        }

        if (a[i-1] == '\r') {
            a[i-1] = '\0';
            strupr (a);
            if (strcmp (a, g_exitPassword) == 0) return 0;
            else  {
                i = 0;
                puts ("Incorrect password");
            }
        }

        if (i > 8) {
            i = 0;
            puts ("Incorrect password");
        }
    }

    if (i != 0) {
        time (&now);
        if ((now - start) > 5) {
            i = 0;
            puts ("5 secs is over - Resetting");
        }
    }
```

```
    return 1;
}


//_____________________________________________________________________________
/*                          MakeServerName
                            --------------

    This routine makes a server name out of the 6 byte network node address.
    Each byte of the address is converted to two characters in the name.
    The value of each nibble in the byte is taken as the offset from the
    character 'A'.
*/

void MakeServerName (char *sName, char *add)
{
  int i, j;

    for (i = 4, j = 0; i < 10; i++, j += 2){
        sName[j] = ((add[i] & 0xf0) / 16) + 65;
        sName[j+1] = (add[i] & 0x0f) + 65;
    }

    sName[12] = '\0';
}


//_____________________________________________________________________________

void InitialiseCommunication (void)
{
    g_requestQueuePtr = 0;
    g_serviceQueuePtr = 0;
    g_queueCount = 0;
    Set_EcbSendWs ();
    Set_EcbReceiveWs ();
    IPXListenForPacket (&g_ecbReceiveWs[0]);
}
```

APPENDIX C4

L O A D D B . C

LOADDB.C is the source file for the program that remotely loads a new copy of the Virtual Machine (VM) database into the memory of the PC that acts as the VM database server.

C4.1 Description of the Code

The code that composes the LOADDB.C program is described under this section. The description starts with an explanation of the main() function and then some important functions called by main(). The significant parts of the code is presented in pseudocode format.

C4.1.1 Function — main()

The main() function receives an argument from the command line specifying the name of the server where the VM database files to be loaded are located. The usage and the explanation of the arguments is given in the function DisplayUsage(). The function main() does not return any values at all.

The pseudocode that describes the operation of main() is presented below.

```
void main (int argc, char *argv[])
{
    Set up request buffer for the load database service
    Locate all the VM database servers that have to be updated
    for (every VM database server) {
        Send the request and wait for the response
```

```
        Print message corresponding to the response on the screen
    }
}
```

C4.1.2 Function — Initialise()

This function does not receive any parameters or return any values.The pseudocode that describes the operation of Initialise() is presented below.

```
void Initialise (void)
{
   initialise IPX communication

   set the socket to communicate with the VM database servers

   Locate all the VM database servers that have to be updated

   Set the number of requests to the VM database servers to zero
}
```

Each time the program requests for information the count of the number of requests is incremented. In this way, it is possible to determine whether a response from a server is for the correct request.

C4.1.3 Function — Esr_ReceiveDb()

This function is an interrupt service routine that is called whenever the PC running the LOADDB program receives something from a VM database server. Its sole function is to set a flag if the received response from the server is to the correct request.

Esr_ReceiveDb() does not receive any parameters or return any values.

The pseudocode that describes the operation of Esr_ReceiveDb() is presented below.

```
void far Esr_ReceiveDb (void)
{
    Setup the DS register
    /* because the following conditions are true
       when this routine is called:
            • A far pointer to the ECB is in the
              ES:SI register pair
            • The DS register does not point to the
              program's data area
    */

    if (response matches the request)
        set flag that indicates successful receipt of information
        from the VM database server

    else
        re-enable listening for a response from the VM database server
}
```

C4.1.4 Function — DatabaseRequest()

This function manages the sending of requests (and the receiving of the response) to a VM database server. The global variable g_dbTablePosition points to an entry in the table of known VM database servers (the table is called g_dbTable) that specifies the server to which the request is to be made.

DatabaseRequest() does not receive any parameters. The returned values are:

- no VM database server responded or
- the error type from the server.

The pseudocode that describes the operation of DatabaseRequest is presented below.

```
BYTE DatabaseRequest (void)
{
    increment request number and reset flag that indicates received from server

    Open the socket to communicate with the VM database servers

    Set the send and receive Event Control Blocks

    Start listening for a response from the VM database server

    Send update command to VM database server and wait for a reply.
    /* If a response is not received within the specified time-out value
       the update command is resent upto MAX_RETRY times.                */

    if (the VM database server has responded)
        close communication socket and return error value from server

    else
        close communication socket and return 'unsuccessful' value
}
```

C4.1.5 Function — LocateDatabaseServer()

This function does not receive any parameters, but returns the number of VM database servers that have been located (up to a maximum specified by MAX_DB_TABLE_POSITIONS). The VM database servers are located by searching the bindery of the preferred file server for all bindery objects of the type 'VM database server'. Every time a VM database server is operational, it advertises its presence on the network. This advert is noted by all file servers and bridges in the network which then keeps a tab of the address and the name of the advertising server.

The pseudocode that describes the operation of LocateDatabaseServer() is presented below.

```
BYTE LocateDatabaseServer (void)
{
    Set the VM database server object type for the bindery scan

    Scan the file server's bindery for all known VM database servers
    // (max. number of servers = MAX_DB_TABLE_POSITIONS)

    Fill up the table of VM database servers

    Return the number of located servers
}
```

C4.1.6 Functions — Set_EcbSendDb() and Set_EcbReceiveDb()

These function prepares the Event Control Block for the send to and receive from VM database server operations.

C4.2 Program Listing

```
/*
```

```
                                File name: LOADDB.C
```

```
Compiler Model COMPACT
C++ options turned to "CPP extension only"


Project File should include the following files:
    loaddb.c
    lnit.lib
```

```
Software that communicates with VM database servers to update the VM database.

By S.J. Isaac

Last revised: 13 March 1993
*/


#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <string.h>
#include <time.h>
#include <nit.h>
#include <nxt.h>

#include "c:\h\uvm.h"


#define MAX_RETRY 3
#define MAX_DB_TABLE_POSITIONS 4
#define POP_ATTRIBUTE 0x5b


//_____________________________________________________________________________
//                        Global variable declaration

struct              // Information buffer to send to the database
    {
    BYTE requestType;
    WORD requestNumber;
    BYTE retry;
    char data[REQUEST_SIZE];
    }g_sendDb;

struct              // Information buffer to receive from the database
    {
    BYTE responseType;
    WORD requestNumber;
    char data[RESPONSE_SIZE];
    }g_receiveDb;

struct              // Table of Database servers that have been located
    {
    char name[48];
    BYTE address[12];
    }g_dbTable[MAX_DB_TABLE_POSITIONS];

struct
    {
    char name[SERVER_NAME_LENGTH];
    char password[9];
    }g_fileServer;


ECB  g_ecbReceiveDb;
ECB  g_ecbSendDb;
IPXHeader g_ipxheaderSendDb, g_ipxheaderReceiveDb;
```

```
BYTE g_return_from_db_flag = 0;
WORD g_dbServerType;
BYTE g_numberOfServers = 0;
int  g_dbTablePosition = 0;
int  g_dbSocket;
WORD g_primaryConnectionID;


//_____________________________________________________________________________
//                        Declaration of routines

void Initialise (void);
void far Esr_ReceiveDb (void);
BYTE DatabaseRequest (void);
BYTE LocateDatabaseServer (void);
void Set_EcbSendDb (void);
void Set_EcbReceiveDb (void);
void DisplayUsage(void);


//_____________________________________________________________________________

void main (int argc, char *argv[])
{

  int  ccode;
  BYTE c;


    // Check for correct usage

    if (argc != 2) DisplayUsage();


    // Set up non-changing parameters in database request buffer

    memset ((void *)&g_fileServer, 0, sizeof (g_fileServer));
    strncpy (g_fileServer.name, argv[1], SERVER_NAME_LENGTH);
    strcpy (g_fileServer.password, getpass("password for file server: "));
    memcpy (g_sendDb.data, &g_fileServer, sizeof (g_fileServer));
    g_sendDb.requestType = LOAD_DATABASE;

    Initialise ();

    for (g_dbTablePosition = 0; g_dbTablePosition < g_numberOfServers;
         g_dbTablePosition++) {

        c = DatabaseRequest ();

        switch ((int) c){

            case 0:

                puts ("Database successfully UPDATED.");
                break;

            case 1:

                puts ("Update FAILED... Database Server TERMINATED!!");
                break;

            case 2:

                puts ("You supplied the wrong server name and/or password");
                break;

            case 0xFF:
                puts ("Database server could not process request");
                break;

            default: puts ("This server is probably DEAD!");
        }
    }
    exit (0);
}
```

```
//_____________________________________________________________________

void Initialise (void)
{
  BYTE ccode;


    // initialize IPX

    ccode = IPXInitialize();
    if (ccode != 0){
        puts ("IPX is not installed !!");
        exit (1);
    }


    g_dbSocket = IntSwap ( (int) DB_SERVER_SOCKET );


    // obtain the address of the database

    ccode = LocateDatabaseServer();
    if (ccode == 0){
        puts ("Unable to find even a single database server");
        exit (1);
    }


    // initialise database request number

    g_sendDb.requestNumber = 0;
}



//_____________________________________________________________________

void far Esr_ReceiveDb (void)
{
    _AX = _ES;
    _DS = _AX;
    if (g_sendDb.requestNumber == g_receiveDb.requestNumber)
        g_return_from_db_flag = 1;
    else IPXListenForPacket( &g_ecbReceiveDb );
}



//_____________________________________________________________________

BYTE DatabaseRequest (void)
{
  int       ccode;
  time_t    start, now;
  double    time_out = 1.0;
  int       c;


    // set other variables

    g_sendDb.requestNumber++;
    g_return_from_db_flag = 0;


    // open the workstation socket to communicate with the database

    ccode = IPXOpenSocket ( (BYTE *) &g_dbSocket, (BYTE) SHORT_LIVED );
    if ( ccode ) printf("IPXOpenSocket error = %i\n", ccode);


    // Set receive and send ECBs

    Set_EcbReceiveDb ();
    Set_EcbSendDb ();


    // Start listening for a response from the database

    IPXListenForPacket( &g_ecbReceiveDb );
```

```
    // send and wait for database to reply

    printf ("Sending 'Update Command' to database server %s\n",
            g_dbTable[g_dbTablePosition].name);

    for (g_sendDb.retry=0; g_sendDb.retry<MAX_RETRY; g_sendDb.retry++){

        IPXSendPacket (&g_ecbSendDb);

        for (start = time(NULL), now = start;
             time_out > difftime(now, start);
             now = time(NULL) )  {

            if (g_return_from_db_flag){   // Has the database responded?

                g_return_from_db_flag = 0;
                IPXCloseSocket (g_dbSocket);
                puts ("RESPONDED !!");
                return (g_receiveDb.responseType);
            }

            IPXRelinquishControl();
        }

        printf ("Timeout %i :    ", g_sendDb.retry + 1);
    }

    IPXCloseSocket (g_dbSocket);
    puts ("NO RESPONSE !!");
    return (0xCC);
}


//_____________________________________________________________________________

BYTE LocateDatabaseServer (void)
{
  int  oCode, pCode;
  long objectID = -1;
  char objectName[48];
  WORD objectType;
  char objectHasProperties;
  char objectFlag;
  char objectSecurity;
  BYTE propertyValue[128];
  BYTE moreSegments;
  BYTE propertyFlags;

    g_dbServerType =  (WORD) IntSwap ((int)DB_SERVER_TYPE);
    for (objectID = -1, oCode = 0, g_numberOfServers = 0;
        ( !oCode && (g_numberOfServers < MAX_DB_TABLE_POSITIONS) ); ){
        oCode = ScanBinderyObject ("*",
                                   g_dbServerType,
                                   &objectID,
                                   objectName,
                                   &objectType,
                                   &objectHasProperties,
                                   &objectFlag,
                                   &objectSecurity);
        if (oCode == 0){

            // a database server has been located

            pCode = ReadPropertyValue (objectName,
                                       g_dbServerType,
                                       "NET_ADDRESS",
                                       1,
                                       propertyValue,
                                       &moreSegments,
                                       &propertyFlags);
            if (pCode == 0){

                // The NET_ADDRESS has been obtained

                strcpy (g_dbTable[g_numberOfServers].name, objectName);
                memcpy (g_dbTable[g_numberOfServers].address,
                        propertyValue, 12);
                g_numberOfServers++;
```

```
            }
        }
    }

    return g_numberOfServers;
}


//_____________________________________________________________________

void Set_EcbSendDb (void)
{
  int ccode, transportTime;


    // fill the ipxheader for g_sendDb

    memcpy((void *)&g_ipxheaderSendDb.destination,
           g_dbTable[g_dbTablePosition].address, 12);
    g_ipxheaderSendDb.packetType = 4;


    // fill_ecbSendDb ();

    memset (&g_ecbSendDb, 0, sizeof(ECB));
    g_ecbSendDb.ESRAddress = NULL;
    g_ecbSendDb.socketNumber = g_dbSocket;
    g_ecbSendDb.fragmentCount = 2;
    g_ecbSendDb.fragmentDescriptor[0].address = &g_ipxheaderSendDb;
    g_ecbSendDb.fragmentDescriptor[0].size = sizeof(IPXHeader);
    g_ecbSendDb.fragmentDescriptor[1].address = &g_sendDb;
    g_ecbSendDb.fragmentDescriptor[1].size = sizeof(g_sendDb);

    ccode = IPXGetLocalTarget (g_dbTable[g_dbTablePosition].address,
                               g_ecbSendDb.immediateAddress,
                               &transportTime);

    if (ccode){
        puts ("IPXGetLocalTarget error in routine DatabaseRequest");
        exit (1);
    }
}


//_____________________________________________________________________

void Set_EcbReceiveDb (void)
{
    memset (&g_ecbReceiveDb, 0, sizeof(ECB));
    g_ecbReceiveDb.ESRAddress = Esr_ReceiveDb;
    g_ecbReceiveDb.socketNumber = g_dbSocket;
    g_ecbReceiveDb.fragmentCount = 2;
    g_ecbReceiveDb.fragmentDescriptor[0].address = &g_ipxheaderReceiveDb;
    g_ecbReceiveDb.fragmentDescriptor[0].size = sizeof(IPXHeader);
    g_ecbReceiveDb.fragmentDescriptor[1].address = &g_receiveDb;
    g_ecbReceiveDb.fragmentDescriptor[1].size = sizeof(g_receiveDb);
}


//_____________________________________________________________________

void DisplayUsage (void)
{
    puts ("\nUsage: LOADDB <server>");
    puts ("  server = file server where database files are found\n");
    exit (1);
}
```

APPENDIX C5

VM_CREAT.C

VM_CREAT.C is the source file for the program that creates special users on a file server for the operation of the Virtual Machine (VM) system. Only the supervisor on a file server will be able to use VM_CREAT.EXE (which can be found in the SYS:PUBLIC/BIN directory).

Syntax: VM_CREAT [<password file>]

If the password file name is not supplied, VM_CREAT will request the relevant information from a VM database server in which case there must be at least one VM database server operational in the network. It is best to get the information from the VM database server. If the operation has been successful, no error information will be received at all. After running the program some new groups and some new users have been created on the file server. The groups are VM groups and the details can be obtained from the VM database administrator.

C5.1 Description of Code

C5.1.1 Function — main()

The function main does not return any values. The pseudocode is presented below.

```
void main (int argc, char *argv[])
{
    Initialise();
    Get information about VM groups
    Create the special VM users
}
```

C5.1.2 Function — Initialise()

This function does not return any values or receive any parameters. Initialise() checks whether the user who has invoked the program VM_CREAT is the supervisor and prepares communication with the network.

C5.1.3 Function — Fill_VM_User()

This function does not return any values or receive any parameters. Fill_VM_User() obtains all information about the VM groups from either the VM database server or a file (the file name was supplied on the command line when starting VM_CREAT). The information is placed in a global array of structures called g_vm_user.

C5.1.4 Function — CreateVM()

This function does not return any values or receive any parameters. CreateVM() creates the VM groups and also some special users corresponding to the VM groups in the bindery of the file server. The pseudocode is presented below

```
void CreateVM (void)
{
    for (all the VM groups){
        create group with the VM group name on the file server
        create property GROUP MEMBERS for the group
        attempt to create user with the name of the VM group
        if (successfully created user){
            create property GROUPS_I'M_IN for the user
            create property SECURITY_EQUALS for the user
            create property ACCOUNT_BALANCE for the user
            create property VM_OLDPASS for the user
            old = NULL;  // set the old password for this object as NULL
        }
        if (the user already exists){
            obtain old password for the user
            encrypt the old password
            set old = the encrypted old password
        }

        // change the password for the user

        encypt the new password
        if (the new password is not the same as the old password)  {
            change the password in the bindery for the user
            keep a copy of the new password in the property VM_OLDPASS
        }
    }
}
```

C5.1.5 Other Functions

The following functions are described in Appendix C4 (LOADDB.C):

Esr_ReceiveDb(), DatabaseRequest() and LocateDatabaseServer().

C5.2 Program Listing

```
/*

                                File name: VM_CREAT.C


Compiler Model LARGE
C++ options turned to "CPP extension only"


Project File should include the following files:
     vm_creat.c
     lnit.lib
     t2lowl.lib
     t2lcscap.lib


 Software that coordinates with the database software to create
 special users for the virtual machine environment.

 By S.J. Isaac

 Last revised: 26 January 1992
*/

#include <stdio.h>
#include <mem.h>
#include <time.h>
#include <string.h>
#include <stdlib.h>
#include <conio.h>
#include <nit.h>
#include <nxt.h>

#include "cscape.h"
#include "ostdlib.h"
#include "teddecl.h"
#include "msys.h"
#include "sled.h"
#include "popdecl.h"


#include "c:\h\uvm.h"



#define MAX_RETRY 3
#define MAX_DB_TABLE_POSITIONS 4
#define MAX_AP_SERVERS 7
#define POP_ATTRIBUTE 0x5b


//_____________________________________________________________________________
//                        Global variable declaration

struct              // Information buffer to send to the database
    {
    BYTE requestType;
    WORD requestNumber;
    BYTE retry;
    char data[REQUEST_SIZE];
    }g_sendDb;

struct              // Information buffer to receive from the database
    {
    BYTE responseType;
    WORD requestNumber;
    char data[RESPONSE_SIZE];
    }g_receiveDb;

struct              // Table of Database servers that have been located
    {
    char name[48];
```

```
    BYTE address[12];
    }g_dbTable[MAX_DB_TABLE_POSITIONS];

struct
    {
    char groupName[VM_GROUP_NAME_LENGTH];
    char userName[VM_USER_NAME_LENGTH];
    char password[VM_PASSWORD_LENGTH];
    }g_vm_user[NUM_OF_ACCESS_GROUPS];


ECB  g_ecbReceiveDb;
BYTE g_return_from_db_flag = 0;
WORD g_dbServerType;
BYTE g_numberOfServers = 0;
int  g_dbTablePosition = 0;
int  g_dbSocket;
WORD g_primaryConnectionID;
int  g_local = 0;
FILE *g_fp;


//_____________________________________________________________________________
//                        Declaration of routines

void Initialise (void);
void far Esr_ReceiveDb (void);
BYTE DatabaseRequest (void);
BYTE LocateDatabaseServer (void);
BYTE GetPasswordRec (int rec);
void DisplayUsage(void);
void Fill_VM_User(void);
void CreateVM(void);
void Scramble(char *plainText, char *c);
void CleanUp (void);



//_____________________________________________________________________________

void main (int argc, char *argv[])
{
  char buf[256];

    // initialize the display
    disp_Init(def_ModeText, FNULL);
    hard_InitMouse();
    sedwin_ClassInit();

    atexit(disp_Close);

    if (argc > 2) DisplayUsage();
    if (argc == 2){
        if (strcmp(argv[1],"/?") == 0) DisplayUsage();
        if ((g_fp = fopen(argv[1],"rt")) == NULL){
            sprintf(buf,"Cannot open %s ",argv[1]);
            pop_Prompt(buf, -1,-1,-1,-1, POP_ATTRIBUTE,bd_1 );
            exit(1);
        }
        g_local = 1;
    }

    Initialise();
    Fill_VM_User();
    CreateVM();

    disp_Close();

}


//_____________________________________________________________________________

void Initialise (void)
{
  BYTE ccode;
  WORD objType;
```

```
  char objName[48];
  long objID;
  BYTE loginTime[7];
  WORD oldConnectionID;


    // initialize IPX
    ccode = IPXInitialize();
    if (ccode != 0){
        pop_Prompt("IPX is not installed !!",
                   -1,-1,-1,-1, POP_ATTRIBUTE,bd_1 );
        exit(1);
    }

    // set the preferred fileServer to the primary fileServer
    g_primaryConnectionID = GetPrimaryConnectionID();
    if (g_primaryConnectionID != 0){
        oldConnectionID = GetPreferredConnectionID ();
        SetPreferredConnectionID (g_primaryConnectionID);
    }
    else {
        pop_Prompt("Cannot find the primary fileServer\nPlease login again",
                   -1,-1,-1,-1, POP_ATTRIBUTE,bd_1);
        exit(1);
    }



    // Check whether the user is SUPERVISOR
    ccode = GetConnectionInformation (GetConnectionNumber(),
                                      objName,
                                      &objType,
                                      &objID,
                                      loginTime);

    if (stricmp(objName, "SUPERVISOR") != 0){
        pop_Prompt ("You have to be the supervisor to run this program",
                    -1,-1,-1,-1, POP_ATTRIBUTE, bd_1);
        exit(1);
    }


    if (g_local == 0){

        g_dbSocket = IntSwap ( (int) DB_SERVER_SOCKET );

        // obtain the address of the database
        ccode = LocateDatabaseServer();
        if (ccode == 0){
            pop_Prompt ("Unable to find a database",
                        -1,-1,-1,-1, POP_ATTRIBUTE,bd_1);
            exit (1);
        }

        // initialise database request number
        g_sendDb.requestNumber = 0;

    }
}


//_____________________________________________________________________

void far Esr_ReceiveDb (void)
{
    _AX = _ES;
    _DS = _AX;
    if (g_sendDb.requestNumber == g_receiveDb.requestNumber)
        g_return_from_db_flag = 1;
    else IPXListenForPacket( &g_ecbReceiveDb );
}



//_____________________________________________________________________

BYTE DatabaseRequest (void)
{
  ECB       ecbSendDb;
```

```
IPXHeader ipxheaderSendDb, ipxheaderReceiveDb;
int       ccode, transportTime;
time_t    start, now;
double    time_out = 0.5;
int       c;
char      buf[256];


          // Initialisations



          // set other variables

  g_sendDb.requestNumber++;
  g_return_from_db_flag = 0;

          // open the workstation socket to communicate with the database

  ccode = IPXOpenSocket ( (BYTE *) &g_dbSocket, (BYTE) SHORT_LIVED );
  if ( ccode ){
      sprintf(buf,"IPXOpenSocket error = %i", ccode);
      pop_Prompt(buf,-1,-1,-1,-1, POP_ATTRIBUTE,bd_1);
  }

          // fill_g_ecbReceiveDb ()

  memset (&g_ecbReceiveDb, 0, sizeof(ECB));
  g_ecbReceiveDb.ESRAddress = Esr_ReceiveDb;
  g_ecbReceiveDb.socketNumber = g_dbSocket;
  g_ecbReceiveDb.fragmentCount = 2;
  g_ecbReceiveDb.fragmentDescriptor[0].address = &ipxheaderReceiveDb;
  g_ecbReceiveDb.fragmentDescriptor[0].size = sizeof(IPXHeader);
  g_ecbReceiveDb.fragmentDescriptor[1].address = &g_receiveDb;
  g_ecbReceiveDb.fragmentDescriptor[1].size = sizeof(g_receiveDb);


  c = g_dbTablePosition;
  do{
          // fill the ipxheader for g_sendDb

      memcpy((void *)&ipxheaderSendDb.destination,
             g_dbTable[g_dbTablePosition].address, 12);
      ipxheaderSendDb.packetType = 4;

          // fill_ecbSendDb ();

      memset (&ecbSendDb, 0, sizeof(ECB));
      ecbSendDb.ESRAddress = NULL;
      ecbSendDb.socketNumber = g_dbSocket;
      ecbSendDb.fragmentCount = 2;
      ecbSendDb.fragmentDescriptor[0].address = &ipxheaderSendDb;
      ecbSendDb.fragmentDescriptor[0].size = sizeof(IPXHeader);
      ecbSendDb.fragmentDescriptor[1].address = &g_sendDb;
      ecbSendDb.fragmentDescriptor[1].size = sizeof(g_sendDb);

      ccode = IPXGetLocalTarget (g_dbTable[g_dbTablePosition].address,
                                 ecbSendDb.immediateAddress,
                                 &transportTime);
      if (ccode){
          pop_Prompt("IPXGetLocalTarget error in routine DatabaseRequest",
                     -1,-1,-1,-1, POP_ATTRIBUTE,bd_1);
          exit(1);
      }

          // Start listening for a response from the database

      IPXListenForPacket( &g_ecbReceiveDb );

          // send and wait for database to reply

      for (g_sendDb.retry=0; g_sendDb.retry<MAX_RETRY; g_sendDb.retry++){
          IPXSendPacket (&ecbSendDb);
          for (start = time(NULL), now = start;
              time_out > difftime(now, start);
              now = time(NULL)){
              if (g_return_from_db_flag){   // Has the database responded?
                  IPXCloseSocket (g_dbSocket);
                  pop_Message(NULL, -1,-1,-1,-1, POP_ATTRIBUTE,bd_1);
                  return (g_receiveDb.responseType);
```

```
                }
                IPXRelinquishControl();
            }
            sprintf (buf, "Time out on database server %s",
                    g_dbTable[g_dbTablePosition].name);
            pop_Message (buf, -1,-1,-1,-1, POP_ATTRIBUTE,bd_1);
        }

        g_dbTablePosition++;
    } while (c != (g_dbTablePosition %= g_numberOfServers));
    pop_Message(NULL, -1,-1,-1,-1, POP_ATTRIBUTE,bd_1);
            // re-obtain the address of the database

    ccode = LocateDatabaseServer();
    if (ccode == 0){
        pop_Prompt ("Unable to find a database",
                    -1,-1,-1,-1, POP_ATTRIBUTE,bd_1);
        exit (1);
    }

    IPXCloseSocket (g_dbSocket);      // hangs station so not closed
    return (0xff);
}


//_____________________________________________________________________________

BYTE LocateDatabaseServer (void)
{
  int  oCode, pCode;
  long objectID = -1;
  char objectName[48];
  WORD objectType;
  char objectHasProperties;
  char objectFlag;
  char objectSecurity;
  BYTE propertyValue[128];
  BYTE moreSegments;
  BYTE propertyFlags;

    g_dbServerType =  (WORD) IntSwap ((int)DB_SERVER_TYPE);
    for (objectID = -1, oCode = 0, g_numberOfServers = 0;
        ( !oCode && (g_numberOfServers < MAX_DB_TABLE_POSITIONS) ); ){
        oCode = ScanBinderyObject ("*",
                                   g_dbServerType,
                                   &objectID,
                                   objectName,
                                   &objectType,
                                   &objectHasProperties,
                                   &objectFlag,
                                   &objectSecurity);
        if (oCode == 0){  // a database server has been located
            pCode = ReadPropertyValue (objectName,
                                       g_dbServerType,
                                       "NET_ADDRESS",
                                       1,
                                       propertyValue,
                                       &moreSegments,
                                       &propertyFlags);
            if (pCode == 0){  // The NET_ADDRESS has been obtained
                strcpy (g_dbTable[g_numberOfServers].name, objectName);
                memcpy (g_dbTable[g_numberOfServers].address,
                        propertyValue, 12);
                g_numberOfServers++;
            }
        }
    }

    return g_numberOfServers;
}


//_____________________________________________________________________________

BYTE GetPasswordRec (int rec)
{
    memcpy(g_sendDb.data, &rec, sizeof (int));
    g_sendDb.requestType = PASSWORD_RECORD;
```

```
    return DatabaseRequest();
}


//_____________________________________________________________________________

void DisplayUsage(void)
{
  char buf[256];

    strcpy (buf,"usage: VM_CREAT [fileName]\n");
    strcat (buf,"       fileName = Path & name of file containing GROUPS\n");
    strcat (buf,"       fileName is optional and if not specified\n");
    strcat (buf,"            the program will try to communicate with \n");
    strcat (buf,"            a database server on the network.\n\n");
    strcat (buf,"Note:  VM_CREAT /? displays usage.\n\n");

    pop_Prompt(buf, -1,-1,-1,-1, POP_ATTRIBUTE,bd_1 );
    exit(1);
}


//_____________________________________________________________________________

void Fill_VM_User(void)
{
  BYTE ccode;
  int  i, j, step, size;

    size = sizeof(g_vm_user[0]);
    if (g_local == 0){
        step = RESPONSE_SIZE / size;
        for(i = 0; i < NUM_OF_ACCESS_GROUPS; i += step){
            ccode = GetPasswordRec (i);
            if (ccode != 0){
                pop_Prompt("Unable to get needed records from database",
                           -1,-1,-1,-1, POP_ATTRIBUTE,bd_1);
                exit(1);
            }
            for (j=0; (j < step) && ((j+i) < NUM_OF_ACCESS_GROUPS); j++){
                memcpy(&g_vm_user[j+i], &g_receiveDb.data[j*size], size);
            }
        }
    }
    else{
        fread((void *) &g_vm_user[0], size, NUM_OF_ACCESS_GROUPS, g_fp);
    }
}


//_____________________________________________________________________________

void CreateVM (void)
{
  int i;
  int ccode, oCcode;
  char buf[128], newPass[128], oldPass[128];
  char buf2[50], *old;
  int segmentNumber;
  BYTE moreSegments, propertyFlags;


    for (i=0; i<NUM_OF_ACCESS_GROUPS; i++){
        if (g_vm_user[i].groupName[0] != '\0'){

            // Create VM group
            ccode = CreateBinderyObject (g_vm_user[i].groupName,
                                         OT_USER_GROUP,
                                         BF_STATIC,
                                         0x31);
            if ( (ccode != 0x00) && (ccode !=0xEE) ){
                sprintf(buf,"Didn't create group\n%s\nerror = %x",
                        g_vm_user[i].groupName, ccode);
                pop_Prompt (buf,-1,-1,-1,-1,POP_ATTRIBUTE,bd_1);
            }
            else{
                // Creat property GROUP_MEMBERS for VM group
```

```
    ccode = CreateProperty (g_vm_user[i].groupName,
                            OT_USER_GROUP,
                            "GROUP_MEMBERS",
                            BF_STATIC | BF_SET,
                            0x31);
    if ( (ccode != 0x00) && (ccode !=0xED) ){
        strcpy (buf, "Didn't create property GROUP_MEMBERS for");
        sprintf (buf2, " group\n%s\nerror = %X",
                 g_vm_user[i].groupName, ccode);
        strcat (buf, buf2);
        pop_Prompt (buf,-1,-1,-1,-1,POP_ATTRIBUTE,bd_1);
    }
}

// Create VM user
oCcode = CreateBinderyObject (g_vm_user[i].userName,
                              OT_USER,
                              BF_STATIC,
                              0x31);
if ( (oCcode != 0x00) && (oCcode !=0xEE) ){
    sprintf(buf,"Didn't create user\n%s",g_vm_user[i].userName);
    pop_Prompt (buf,-1,-1,-1,-1,POP_ATTRIBUTE,bd_1);
}
if (oCcode == 0x00){
    // Creat property GROUPS_I'M_IN for VM user
    ccode = CreateProperty (g_vm_user[i].userName,
                            OT_USER,
                            "GROUPS_I'M_IN",
                            BF_STATIC | BF_SET,
                            0x31);
    if ( (ccode != 0x00) && (ccode !=0xED) ){
        strcpy (buf, "Didn't create property GROUPS_I'M_IN for");
        sprintf    (buf2, " user\n%s\nerror = %X",
                 g_vm_user[i].userName, ccode);
        strcat (buf, buf2);
        pop_Prompt (buf,-1,-1,-1,-1,POP_ATTRIBUTE,bd_1);
    }

    // Creat property SECURITY_EQUALS for VM user
    ccode = CreateProperty (g_vm_user[i].userName,
                            OT_USER,
                            "SECURITY_EQUALS",
                            BF_STATIC | BF_SET,
                            0x32);
    if ( (ccode != 0x00) && (ccode !=0xED) ){
        strcpy(buf,"Didn't create property SECURITY_EQUALS for");
        sprintf    (buf2, " user\n%s\nerror = %X",
                 g_vm_user[i].userName, ccode);
        strcat (buf, buf2);
        pop_Prompt (buf,-1,-1,-1,-1,POP_ATTRIBUTE,bd_1);
    }


    // Creat property ACCOUNT_BALANCE for VM user
    ccode = CreateProperty (g_vm_user[i].userName,
                            OT_USER,
                            "ACCOUNT_BALANCE",
                            BF_STATIC | BF_ITEM,
                            0x32);
    if ( (ccode != 0x00) && (ccode !=0xED) ){
        strcpy(buf,"Didn't create property ACCOUNT_BALANCE for");
        sprintf    (buf2, " user\n%s\nerror = %X",
                 g_vm_user[i].userName, ccode);
        strcat (buf, buf2);
        pop_Prompt (buf,-1,-1,-1,-1,POP_ATTRIBUTE,bd_1);
    }

    // Creat property VM_OLDPASS for VM user
    ccode = CreateProperty (g_vm_user[i].userName,
                            OT_USER,
                            "VM_OLDPASS",
                            BF_STATIC | BF_ITEM,
                            0x33);
    if ( (ccode != 0x00) && (ccode !=0xED) ){
        strcpy (buf, "Didn't create property VM_OLDPASS for");
        sprintf    (buf2, " user\n%s\nerror = %X",
                 g_vm_user[i].userName, ccode);
        strcat (buf, buf2);
        pop_Prompt (buf,-1,-1,-1,-1,POP_ATTRIBUTE,bd_1);
    }
```

```
                // Set the old password for this object to NULL
                old = NULL;

            }


            if (oCcode == 0xEE){
                // Obtain old password since the object already exists
                segmentNumber = 1;
                ccode = ReadPropertyValue (g_vm_user[i].userName,
                                           OT_USER,
                                           "VM_OLDPASS",
                                           segmentNumber,
                                           buf,
                                           &moreSegments,
                                           &propertyFlags);
                if (ccode != 0x00){
                    sprintf (buf, "Unable to obtain old password for\n%s",
                             g_vm_user[i].userName);
                    pop_Prompt (buf,-1,-1,-1,-1,POP_ATTRIBUTE,bd_1);
                }
                else{
                    Scramble (buf, oldPass);
                    old = oldPass;
                }
            }


            // Change Password for the user
            Scramble (g_vm_user[i].password, newPass);
            if (strcmp(old, newPass) != 0){
                ccode = ChangeBinderyObjectPassword(g_vm_user[i].userName,
                                                    OT_USER,
                                                    old,
                                                    newPass);
                if (ccode != 0x00){
                    sprintf (buf, "Unable to change password for\n%s",
                             g_vm_user[i].userName);
                    pop_Prompt (buf,-1,-1,-1,-1,POP_ATTRIBUTE,bd_1);
                }
                else{
                    // Keep a copy of the new password to enable changes
                    // to be made later
                    segmentNumber = 1;
                    moreSegments = 0;
                    ccode = WritePropertyValue(g_vm_user[i].userName,
                                               OT_USER,
                                               "VM_OLDPASS",
                                               segmentNumber,
                                               g_vm_user[i].password,
                                               moreSegments);
                    if (ccode != 0x00){
                        sprintf(buf,
                                "Unable to keep copy of new password for\n%s",
                                g_vm_user[i].userName);
                        pop_Prompt (buf,-1,-1,-1,-1,POP_ATTRIBUTE,bd_1);
                    }

                }
            }
        }
    }
}


//_____________________________________________________________________________

void Scramble(char *plainText, char *c)
{
  size_t pLength;
  char p[130];
  char k[130];
  int i;

    strcpy(p, "zc");
    strcpy(k, "kg");
```

```
    strncat(p, plainText, 127);
    pLength = strlen(p);

    for (i = 2; i < pLength; i++){
        k[i] = abs((((p[i]*p[i-1])/(p[i-2]+1)) + (k[i-1]+k[i-2])) % 26) + 65;
    }
    for (;i <127;i++){
        k[i] = abs((k[i-1] * k[i-2] / (k[i-3]+4)) % 26) + 65;
    }
    k[127] = '\0';
    strcpy (c, &k[2]);
}
```

APPENDIX C6

M P . C

MP.C is the source file for the program that maps a drive letter to a directory on a file server. This program differs from Novell's MAP utility in the following ways:

1. MAP can be used to map both search drives and normal drives where as MP can only be used to map search drives.
2. Both MAP and MP sets the MS-DOS PATH environmental variable when setting a search drive. MP however, sets all the PATH variables in all the MS-DOS command shells while MAP only sets the PATH variable in the main parent's environment.
3. MAP can create many search drive mappings to the same directory on the file server. MP only creates one search mapping and then merely increases the drive Attachment Count (dAC)for the mapped drive letter for every subsequent mapping that is required for that directory.

C6.1 Description of Code

The significant parts of the code is presented in pseudocode format.

C6.1.1 Function — main()

The main() function receives two arguments from the command line specifying the operation and the full path specification of the directory to be mapped. The usage is

given in the function DisplayUsage(). The function main() does not return any values at all.

The pseudocode that describes the operation of main() is presented below.

```
void main (int argc, char *argv[])
{
    Open SAC file

    if (required search mapping already exists){

        if (delete operation) {
            decrement dAC

            if (dAC = 0){
                delete search mapping and remove drive letter from all
                the PATHs up to the main parent
            }
            exit(0);
        }

        if (insert operation){
            increment dAC

            exit(0);
        }
    }

    if (insert operation){
        create search mapping and insert drive letter in all
        the PATHs up to the main parent

        exit(0);
    }
}
```

C6.1.1 Function — Get_dAC()

This is a general purpose function that will either set the dAC value in the SAC file or else obtain the current dAC value from the SAC file. The Get_dAC() function receives the following three parameters:

var	(Input or Output dependent on operation) The variable that stores the drive attachment count. After a read operation, var contains the current number of attachments for the specified drive letter. During a write operation, var contains the number of attachments that should be recorded in the appropriate position (corresponding to the drive letter) in the SAC file.
drive	(Input) the drive letter for which the dAC should be set
operation	(Input) read from SAC file= 0; any other value means write to SAC file.

No Pseudocode is presented for this function.

C6.2 Program Listing

```
/*
```

File Name: MP.C

```
Compiler model COMPACT

Project file should include the following files:
    mp.c
    lib.vm
    cnit.lib
```

```
    This routine is to be used in place of Novell's MAP program in batch
    files that will set up search mappings necessary to run applications.
    During the insert operation, no new mapping is set up if a mapping
    already exists to the required path.
    During the delete operation, a mapping to the required path will only
    be deleted if such a mapping exists.

    It also sets the PATH environment variable upto a maximum of 20 levels
    deep.


    by S.J. Isaac

    Last Revised: 25 April 1993
*/



#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <nit.h>

#include "h:\c\r\h\uvm.h"


//_____________________________________________________________________________
//                  Global Variable Declarations

FILE *g_sac;


//_____________________________________________________________________________
//                  Function Declarations

void DisplayUsage(void);
void Get_dAC (char *var, char drive, int operation);


//_____________________________________________________________________________
//                  Main

void main (int argc, char *argv[])
{
  BYTE driveNumber;
  char driveLetter;
  char directoryPath[255];
  char mapPath[255];
  char server[48];
  char path[255];
  char *drive;
  char delDrive[4];
  char operation;
  char dAC;
  int  ccode;
  int  i;
  int  serverLength;
```

```
int  mapPathLength;
WORD connectionID;

  //print help message if necessary

  if (argc != 3) DisplayUsage();

  strupr(argv[1]);
  if (strcmp (argv[1], "I") == 0) ccode =1;
  if (strcmp (argv[1], "D") == 0) ccode =1;
  if (ccode != 1) DisplayUsage();

  operation = argv[1][0];
  strcpy (mapPath, strupr(argv[2]));
  mapPathLength = strlen(mapPath);


  // Open SAC file

  if ((g_sac = OpenSAC("r+b")) == NULL) dAC = -1;


  // Change all '\' to '/' in the mapPath

  for (i = 0; i < mapPathLength; i++){
      if (mapPath[i] == '\\') mapPath[i] = '/';
  }


  // Get server name from the path to be mapped

  serverLength = strcspn (mapPath, "/");
  if (serverLength == mapPathLength) {
      printf("\nfull path must include server name and volume name");
      DisplayUsage();
  }
  strncpy (server, mapPath, serverLength);
  server[serverLength] = '\0';


  // Get the connection ID of this server

  ccode = GetConnectionID (server, &connectionID);
  if (ccode != 0) {
      printf("\nUnable to get server %s", server);
      DisplayUsage();
  }


  // Does the required search mapping exist

  for (driveNumber = 0; driveNumber < 26; driveNumber++){
      driveLetter = (char)driveNumber + 65;
      if (ccode = IsSearchDrive( driveLetter )){
          if ((ccode = GetFullPath(driveNumber, directoryPath))==0x00){
              if (strcmp(directoryPath, mapPath) == 0){

                  // If delete search mapping

                  if (operation == 'D'){

                      // Decrement dAC and exit if unnecessary to delete

                      if (g_sac != NULL){
                          Get_dAC (&dAC, driveLetter, 0);
                          dAC--;
                          Get_dAC (&dAC, driveLetter, 1);
                          fclose (g_sac);
                          if (dAC > 0) exit (0);
                      }

                      // Deleting search mapping unavoidable

                      ccode = MapDrive (connectionID,
                                        0xff,
                                        mapPath,
                                        DRIVE_DELETE,
                                        0,
                                        &driveLetter);
```

```
                    if (ccode == 0x00) {

                        // Delete drive letter from the DOS PATH & exit

                        strcpy (path, getenv ("PATH"));
                        delDrive[0] = driveLetter;
                        strcpy (delDrive+1, ":.");
                        drive = strstr (path, delDrive);
                        if (*(drive + 3) == '\0') *(drive - 1) = '\0';
                        else {
                            i = strlen (drive + 4);
                            memmove (drive, drive + 4, i + 1);
                        }
                        SetParentEnv ("PATH=", path);
                        exit (0);
                    }
                    else {
                        printf ("\nerror %d during MAP DEL\n", ccode);
                        exit(1);
                    }

                }

                // If insert search mapping just increment dAC & exit

                if (operation == 'I'){
                    if (g_sac != NULL) {
                        Get_dAC (&dAC, driveLetter, 0);
                        dAC++;
                        Get_dAC (&dAC, driveLetter, 1);
                    }

                    // no need to insert because mapping already exists

                    exit(0);
                }
            }
        }
    }
}


if (operation == 'I'){

    // required mapping does not exist, so create one

    driveLetter = '\0';
    ccode = MapDrive (connectionID,
                      0xff,
                      mapPath,
                      DRIVE_INSERT,
                      15,
                      &driveLetter);
    if (ccode ==0x00) {
        if (g_sac != NULL) {
            Get_dAC (&dAC, driveLetter, 0);
            dAC++;
            Get_dAC (&dAC, driveLetter, 1);
        }


        // There is no further need for mapPath, so it is just used as
        // a temporary buffer from now on

        strcpy (mapPath, getenv("PATH"));


        // check whether drive letter already exists in the path

        delDrive[0] = driveLetter;
        strcpy (delDrive+1, ":.");
        drive = strstr (mapPath, delDrive);


        // if drive letter does not exist then append to path

        if (drive == NULL) {
            sprintf(path,"%s;%c:.",mapPath, driveLetter);
            SetParentEnv ("PATH=", path);
        }
```

```
            exit(0);
        }
        else {
            printf ("\nerror %d during MAP INS\n", ccode);
            exit(1);
        }
    }

}


//_____________________________________________________________________________
//                    DisplayUsage

void DisplayUsage(void)
{
    printf("\n\n     Intelligent search drive mapping operation");
    printf("\n     ------------------------------------------\n");
    printf("\nsyntax:   mp <I|D> <path>\n");
    printf("\n     I = insert");
    printf("\n     D = delete");
    printf("\n  path = full path specification eg. eg/sys:public/bin");
    exit(1);
}


//_____________________________________________________________________________
//                    Get_dAC

void Get_dAC (char *var, char drive, int operation)
{
  char index;
  long offset;

    index = drive - 'A';
    offset = SERVER_NAME_LENGTH + USER_NAME_LENGTH + 7 + index;
    fseek (g_sac, offset, SEEK_SET);
    if (operation == 0)
        fread ((void *)var, 1, 1, g_sac);
    else fwrite ((void *)var, 1, 1, g_sac);
}
```

APPENDIX C7

V M S E T . C

VMSET.C is the source file for the program that is to be used in place of the MS-DOS SET command where an environmental variable is to be set to a directory on a file server. In a multiple file server environment, it is unwise to use an explicit drive letter specification in the SET command because the drive letter might refer to a file server's hard disk other than the intended one. Using the VMSET command, a full directory specification that includes the server name and volume name (to remove ambiguity) can be given. At run time, the server name and volume name specification is translated into an appropriate drive letter.

For example

Suppose that the INCLUDE environmental variable has to be set to allow a C compiler to locate the include files. Also, suppose that the include files are located in the directory eg/sys:ap/c/include. If a batch file is written by a file server administrator to allow users to run the C compiler, the administrator could include a line in the batch file (suppose that when the batch file is being written the drive letter g: has been mapped to a directory on eg/sys:):

```
SET INCLUDE=g:\ap\c\include
```

Unfortunately, if a user attaches to server 'eg' after logging in to another server (say 're'), the drive letter g: might be mapped to a directory on the other server.

So the compiler will search for the include files on the wrong server with the above SET INCLUDE statement.

If the line in the batch file is:

VMSET INCLUDE=eg/sys:ap/c/include

even if the user were to login from the server 're' and then attach to the server 'eg', when the batch file is run, eg/sys: is translated into a drive letter (say j:) and the line above becomes similar to:

SET INCLUDE=j:\ap\c\include

(note that all the slashes have been translated into backslashes as well).

VMSET not only sets the current environment, but also all environments up to the original parent's environment.

C7.1 Description of Code

The code that composes the VMSET.C program is described under this section. The description starts with an explanation of the main() function and then some important functions called by main(). The significant parts of the code is presented in pseudocode format.

C7.1.1 Function — main()

The main() function receives an argument from the command line specifying the name of the environmental variable to be set and the full path specification. The usage is given in the function DisplayUsage().

```
void main (int argc, char *argv[])
{
    Change all '/' to '\' in the path
    Separate the tokens in the argument
    if (the environmental variable does not have to be reset){
        if (the environmental variable VMPROG exists){
            if (drive letter in VMPROG specifies the correct server){
                use drive letter in VMPROG to set the environment
                exit(0);
```

```
        }

        Determine which drive the required server is on and use it to set
        the environment
    }
    else reset the environmental variable
}
```

C7.1.2 Function — Token()

This function takes three parameters and is used to separate the environmental variable's name, server name, volume name and the path from the argument string supplied to VMSET. No pseudocode is presented for this function.

C7.1.3 Function — CheckDrive()

This function is used to determine whether a specific drive letter points to the correct server and volume. If the drive letter does point to the correct server and volume, then the environmental variable is set. No pseudocode is presented for this function.

C7.1.4 Function — WriteInEnv()

This function either resets the environmental variable or else modifies it according to the option supplied as a parameter to WriteInEnv(). If option = 0, the environment is set. Any other value of option resets the environmental variable.

```
int WriteInEnv (int option, BYTE driveNumb)
{
  char nVar[256];

    if (option == 0){
        store in nVar the path to be set (including drive letter)

    }
    else nVar[0] = '\0';

    Set the environmental variable to nVar

    return (0);

}
```

C7.2 Program Listing

```
/*
```

File name: VMSET.C

```
Compiler Model: COMPACT
C++ options turned to "CPP extension only"

Project File should include the following files:
    vmset.c
    lib.vm
    lnit.lib
```

```
    This program tries to set environmental variables to a
    path on a specified server. It is used in place of the DOS SET
    command. This program converts the <server>/<volume> specification
    into a drive letter.


    by S. J. Isaac
    date: 25 April 1993
*/


#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <nit.h>


#include "h:\c\r\h\uvm.h"


//_____________________________________________________________________________
//                    Global Variable Declaration

char g_envar[40];
char g_volume[40];
char g_pathName[128];
WORD g_conID;



//_____________________________________________________________________________
//                    Declaration of Routines

void DisplayUsage (void);
void Token (char *in, char *out, char *separator);
int  CheckDrive (BYTE driveNumber);
int  WriteInEnv (int option, BYTE driveNumb);




//_____________________________________________________________________________
//                    Main

void main (int argc, char *argv[])
{
  char server[40];
  char line[128];
  char buf[255];
  char drive[4];
  char *ptr;
  int  ccode, i;
  BYTE driveNum;
  size_t len;


    puts("");

    if (argc != 2) DisplayUsage();
```

```
strcpy (buf, argv[1]);
strupr (buf);
len = strlen (buf);


// Change all '/' to '\' in buf

for (i = 0; i < len; i++){
    if (buf[i] == '/') buf[i] = '\\';
}


// Separate the tokens in buf

Token (buf, g_envar, "=");


// Check whether environment variable has to be reset

ptr = strtok (NULL, "\\");
if (ptr) {


    // Do not reset

    strcpy (server, ptr);
}
else{


    // Reset

    ccode = WriteInEnv (1, 5);
    exit (ccode);
}


// Continue separating the tokens in buf

Token (NULL, g_volume, ":");

ptr = strtok (NULL, "/");
if (ptr)  strcpy (g_pathName, ptr);
else      g_pathName[0] = '\0';


// Get the connectionID of the required server

ccode = GetConnectionID (server, &g_conID);
switch (ccode){
    case 0x00: break;
    case 0xF8: puts("Already attached to file server"); break;
    case 0xFC: puts("Unknown file server");          exit (1);
    case 0xFE: puts("Server bindery locked");        exit (1);
    case 0xFF: puts("No response from file server"); exit (1);
    default:   puts("Error in VM_SET.EXE");          exit (1);
}


// Check whether environmental variable VMPROG exists

if ((ptr = getenv("VMPROG")) != NULL){


    // VMPROG exists, determine whether drive letter specifies the
    // correct server for the SET command.

    strcpy (line, ptr);
    ptr = strchr(line, ':');
    strncpy (drive, ptr-1, 2);
    drive[2] = '\0';
    strupr (drive);
    driveNum = drive[0] - 'A';
    ccode = CheckDrive (driveNum);
    if (ccode < 2)      exit (ccode);
}
```

```
    // Determine which drive the required server is on

    for (i = 5; i < 26; i++){
        ccode = CheckDrive (i);
        if (ccode < 2)  exit (ccode);
    }

}


//_____________________________________________________________________________
//                    DisplayUsage

void DisplayUsage (void)
{
    puts("-----------------------------------------------------------------");
    puts("This program will set an environmental variable to a path on a");
    puts("specific server\n");

    puts("                              * * * * * * *");
    puts("syntax:  VMSET <envar>=<pathname>\n");

    puts("   envar = environmental variable to set");
    puts("pathname = the path including the server & volume");

    puts("\n                              * * * * * * *");
    puts("Example:\n");

    puts("          VMSET include=eg/sys:ap/msc/include\n");
    puts("          Note: There are no spaces in <envar>=<pathname>");
    puts("                There is an '=' sign between envar & pathname");
    puts("-----------------------------------------------------------------");
    exit (1);
}


//_____________________________________________________________________________
//                    Token

void Token (char *in, char *out, char *separator)
{
  char *ptr;

    ptr = strtok (in, separator);
    if (ptr)  strcpy (out, ptr);
    else  DisplayUsage();
}


//_____________________________________________________________________________
//                    CheckDrive

int CheckDrive (BYTE driveNumber)
{
  int  ccode;
  BYTE directoryHandle;
  WORD connectionID;
  char buf[128];
  char *ptr;


    ccode = GetDriveInformation (driveNumber,
                                 &connectionID,
                                 &directoryHandle);

    if (directoryHandle != 0x00){


        // Valid drive letter, so check whether
        // this is the server required.

        if (g_conID == connectionID){


            // Correct server, so check whether correct volume

            ccode = GetDirectoryPath (directoryHandle, buf);
```

```
            ptr = strtok (buf, ":");
            if (ptr) {
                if (strcmp (ptr, g_volume) == 0){


                    // Correct volume, so write into file
                    ccode = WriteInEnv (0, driveNumber);
                    return (ccode);
                }
            }
        }
    }

    return (2);
}


//_____________________________________________________________________________
//                  WriteInEnv

int WriteInEnv (int option, BYTE driveNumb)
{
  char var[256], nVar[256];


    sprintf (var, "%s=", g_envar);
    if (option == 0){
        sprintf (nVar, "%c:\\%s", driveNumb + 'A', g_pathName);
    }
    else nVar[0] = '\0';

    SetParentEnv (var, nVar);

    return (0);

}
```

APPENDIX C8

REGVM.C

REGVM.C is the source file for the program that registers an application in the Server Attachment Count (SAC) file. The registration process simply indicates the name of the file server and the number of applications on that file server that the user is currently using. If there are no applications being used from a particular file server, the connection to that file server can be released.

C8.1 Description of Code

The significant parts of the code is presented in pseudocode format.

C8.1.1 Function — main()

The main() function receives two arguments from the command line specifying the operation and the name of the file server to be registered. The usage is given in the function DisplayUsage(). The function main() does not return any values at all but does set the MS-DOS errorlevel to one if an error occurred during the registration process.

The pseudocode that describes the operation of main() is presented below.

```
void main (int argc, char *argv[])
{
    Open SAC file

    // obtain information about the primary server
    if (server name to be registered is the same as primary server name){
        close SAC file
```

```
        exit(0);
    }


    // modify server attachment count

    if (server name to be registered is found in the SAC file){

        increment or decrement attachment count in SAC file
        (dependent on operation)

        close SAC file

        exit(0);
    }

    close SAC file

    exit(1);  // an error occurred
}
```

C8.2 Program Listing

```
/*
                                File Name: REGVM.C

Compiler Model COMPACT

Project file should include the following files:
    regvm.c
    lib.vm
    cnit.lib


    This program increments the server attachment count in the server
    attachment count file if the 'i' option is used and decrements the
    same if the 'd' option is used.
    The exit code is set to 1 if it is unable to perform this operation.


    by S.J. Isaac
    Last Revised: 26 April 1993
*/



#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "h:\c\r\h\uvm.h"


void DisplayUsage(void);


void main (int argc, char *argv[] )
{
  char operation;
  char serverName[48];
  size_t offset;
  FILE *sac;
  SAC_struct primary;

  struct serverAttachment
        {
        char name[SERVER_NAME_LENGTH];
        unsigned char count;
        }serverAttachment;
```

```
    // display help if necessary

    puts ("");

    if (argc != 3)      DisplayUsage();

    strupr(argv[1]);
    if (strcmp(argv[1], "I") == 0)      operation = 'I';
    if (strcmp(argv[1], "D") == 0)      operation = 'D';
    if ((operation != 'I') && (operation != 'D'))      DisplayUsage();


    // convert arguments to uppercase

    strupr(argv[2]);
    strcpy (serverName, argv[2]);


    // Open SAC file

    if ((sac=OpenSAC("r+b")) == NULL) exit (1);
    fseek (sac, 0, SEEK_SET);


    // obtain information about the primary server

    offset = sizeof (SAC_struct);
    fread ((void *)&primary, offset, 1, sac);
    if ( strcmp(primary.primaryServer, serverName) == 0){
        fclose (sac);
        exit(0);   // Should always be attached to the primary server
    }


    // modify server attachment count

    fread ((void *)&serverAttachment,
           sizeof(struct serverAttachment),
           1,
           sac);

    while (!feof(sac)) {
        if ( strcmp(serverAttachment.name, serverName) == 0) {
            if (operation == 'I')
                if (serverAttachment.count < 0xFF)
                    serverAttachment.count++;
            if (operation == 'D')
                if (serverAttachment.count > 0x00)
                    serverAttachment.count--;
            fseek(sac, (long)offset, SEEK_SET);
            fwrite ((void *)&serverAttachment,
                    sizeof(struct serverAttachment),
                    1,
                    sac);
            fclose (sac);
            exit(0);
        }
        fread ((void *)&serverAttachment,
               sizeof(struct serverAttachment),
               1,
               sac);
        offset += sizeof(struct serverAttachment);
    }

    fclose (sac);
    exit(1);  // an error occurred
}


void DisplayUsage(void)
{
    puts ("-------------------------------------------------------");
    puts ("syntax: REGVM <I|D> <serverName>");
    puts ("         I = increment server attachment count");
    puts ("         D = decrement server attachment count");
    puts ("serverName = name of server where this batch file exists");
    puts ("-------------------------------------------------------");
    puts ("\n");
    exit(1);
}
```

APPENDIX C9

GETPARAM.C

GETPARAM.C is the source file for the program that obtains command line parameters for applications and stores them in the environmental variable VMPARAM. Using this program it is possible to place applications that require command line parameters in the Virtual Machine (VM) menu program. Since the user's choice from the menu runs a batch file that starts the application, the name of the application (say APPLIC.EXE) in the batch file should have %VMPARAM% after it, e.g. APPLIC %VMPARAM%. (GETPARAM.EXE should immediately precede the line that starts the application.) When the batch file is run, %VMPARAM% is replaced by the actual command line parameters obtained with GETPARAM.

C9.1 Code Description

This program is very simple and is easy to follow from the actual code itself, so no further description of the code is necessary.

C9.2 Program Listing

```
/*
```

File Name: GETPARAM.C

```
Compiler model COMPACT
C++ options turned to "CPP extension only"


Project file should include the following files:
    getparam.c
    cnit.lib
    lib.vm
```

```
    This is a program to get the necessary parameters for another
    program that follows this in a batch file. The parameters that
    have been obtained can be accessed from the batch file with
        %VMPARAM%
    VMPARAM is set as an environmental variable.


    By S.J. Isaac
    Date: 25 April 1993
*/


#include <stdio.h>

#include "h:\c\r\h\uvm.h"


//_____________________________________________________________________
//                    Declaration of Routines

void DisplayUsage (void);



//_____________________________________________________________________
//                      Main

void main (void)
{
  char param[256];

    DisplayUsage();

    gets (param);

    SetParentEnv ("VMPARAM=", param);
}



//_____________________________________________________________________
//                    DisplayUsage

void DisplayUsage (void)
{
    puts ("");
    puts ("_____________________________________________________________");
    puts ("");
    puts ("   Please enter parameters (if any) after the prompt");
    puts ("");
    puts ("              PARAMETERS>");
    puts ("");
    puts ("   Hit the ENTER key when finished.");
    puts ("_____________________________________________________________");
    puts ("");
    printf ("PARAMETERS> ");
}
```

APPENDIX C10

V M _ M E N U . C

VM_MENU.C is the source file for the program that presents the Virtual Machine (VM) menu to the user. To help describe the program please consider Figure C10.1 below.

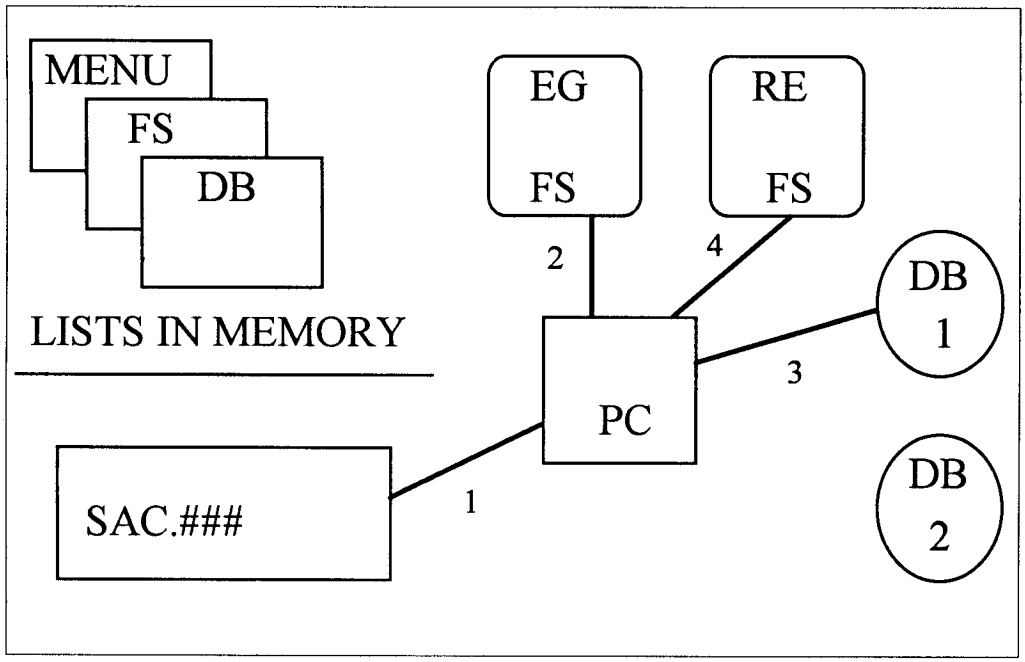

Figure C10.1: VM_MENU Program

The VM_MENU program is located on all the file servers and becomes available to a user upon successful login. This program is usually started from within a batch file and is executed in the user's PC. The steps taken by this program to present a menu of

applications to the user are described below, and correspond to the numbered links in Figure C10.1.

1. As soon as the program is invoked, it checks the Server Attachment Count (SAC) file created by the login utility to see whether it is the correct SAC file for the session (this is done by cross referencing the information contained in the SAC file about the user with the information in the file server's bindery designated as the primary file server in the SAC file). If the SAC file is verified, then a list of attached file servers from the SAC file is placed in the memory of the user's PC (shown in Figure C10.1 as a partially hidden rectangle with the letters 'FS' in the portion called 'LISTS IN MEMORY").

2. From the information in the SAC file VM_MENU checks the bindery of the primary (or home) file server for all known VM database servers and places this information in a list in memory. In Figure C10.1, the primary file server is called 'EG' and the list of VM database servers is shown as a rectangle with the letters 'DB'. Two file servers called 'EG' and 'RE' are shown in the figure. Also obtained from the home file server is the user's VM group.

3. The first VM database server (from the list maintained in memory) is set as the preferred server. If this server fails, then communication will be established with the next VM database server on the list and so on until all the servers on the list have been exhausted. (In Figure C10.1, two VM database servers called 'DB 1' and 'DB 2' are shown.) To the preferred VM database server a request for the main menu is made, and this is presented to the user. Each time the user selects another menu from the presented menu the ID of the requested menu is passed on to the database server and the new menu from the database server is presented to the user. In this way a list of selected menus (up to 20 levels) deep is created in the PC's memory. The user can go back up the levels of the menus by pressing the 'ESC' key on the keyboard. Backtracks do not require communication with the database server. However, any selection made at that level is still fetched from the

database server, even if it is an option that has previously been selected at that level. In this way, the menus are managed until an application is selected from a menu.

4. When an application has been selected, the database server provides information about all file servers that have the application and the access categories that can use the application. If the user has the correct access rights, the VM_MENU program then decides which file server to attach to, based on the following considerations in order of priority:

   - Is the application present on the home server? (It is always preferred to start the application on the home server because the user could have more rights here than anywhere else.)
   - Is the application on any other file servers that the PC has already attached to? Connect to the first one that has been found.
   - Is the application on any active file server? Connect to the first one that has been found.

   Figure C10.1 shows a connection being made to a file server called 'RE'.

The Environmental Variable — VMPROG

When VM_MENU ends it is unloaded from the PC memory. The information to start the program must accordingly be stored in a safe place. The following is therefore done:

- A search mapping is made to the directory on the file server containing the batch file that will start the application.
- The name of the batch file (and the drive on which it is located) is stored in the environmental variable VMPROG. This environmental variable is set in all the environmental blocks up to the parent environment.

Example: Suppose a batch file called 'APP.BAT' found on a file server called 'RE' and stored in the directory 'RE/SYS:PUBLIC/BIN' will start the application that the user

has selected from the menu. Firstly a search mapping is made to this directory if it does not already exist, e.g.' W: = RE/SYS:PUBLIC/BIN'. Then the environmental variable VMPROG is set to 'W:APP.BAT'.

Ending

If the program ends due to an error, the errorlevel is set to 1. If the program ends because the user chooses to quit the menu system, the errorlevel is set to 3. If an application has been selected, the errorlevel is set to 0.

On normal ending, the program detaches from servers to which a connection need no longer be maintained, and current information is placed in the SAC file. The display is also cleaned up and files are closed.

C10.1 Description of Code

C10.1.1 Function — main()

The function main does not return any values or receive any parameters. The pseudocode is presented below.

```
void main (void)
{
    initialise the display and VMcommunication

    get the main menu from the VM database server

    keep chaining through the menus until the user selects an application
    or otherwise quits the menu.

    if (application has been selected) set the environmetal variable VMPROG

    clean up display and exit
}
```

C10.1.2 Function — Initialise()

The function Initialise() does not return any values or receive any parameters. The pseudocode is presented below.

```
void Initialise (void)
{
    initialize the display and IPX communication

    open server attachment count file or exit if it cannot be found

    obtain information about the primary server and other attached servers

    obtain the address of the database
```

```
    Get access rights from the primary server // = user's VM group
    initialise database request number
}
```

C10.1.3 Functions That Can Be Found In Appendix C4 (LoadDB.C)

The following functions are described in Appendix C4:

1. Esr_ReceiveDb()
2. DatabaseRequest()
3. LocateDatabaseServer()

C10.1.4 Function — GetIDRec()

This is a utility function used by the functions GetMenu() and GetApplication(). Its function is to fill in the request buffer with the ID of the requested object and the request type (i.e. MENU_RECORD or APPLICATION_RECORD).

C10.1.5 Function — GetMenu()

This function's purpose is to get a menu (specified by a menuID) from the VM database server. Menu IDs start from zero (the main menu). The function returns the completion code of the operation (zero represents success).

C10.1.6 Function — GetApplication()

This function's purpose is to get some details of an application from the VM database server. GetApplication() receives one parameter; the application ID that is required. Application IDs start from one. The function returns the following values:

0	successful
0xBB	could not find any active file servers with the requested application
0xF0	unable to get information from the VM database server
0xFF	the user does not have the necessary rights to access the application

The pseudocode describing this function is presented below.

```
BYTE GetApplication (WORD ID)
{
    Get information from VM database server
```

```
    if (unsuccessful) return 0xF0;

    if (user is not allowed to access application) return 0xFF;

    if (application is found on primary server){
        set primary server as the preferred server
        return 0x00;
    }

    if (application is located on any other attached server) {
        set first attached server to be found as the preferred server
        return 0x00;
    }

    if (application is found on any active unattached server){
        attach to the first server that is located
        login to that file server with the VM group name and password
        that the user belongs to.
        if (successful login){
            map a search drive to sys:public\bin on the file server
            record application information in relevant buffers
            return 0x00;
        }
    }

    return 0xBB; // could not find any active file servers
}
```

C10.1.7 Function — ManageMenus()

This function manages the presentation of menus to the user. The function ManageMenus() does not receive any parameters. The function returns -1 if the user desires to quit the main menu, otherwise it returns the number of the selected application (where the first application seen on the menu will return 0 and the next 1 and so on).

The pseudocode is presented below.

```
void ManageMenus (void)
{
    do{
        present the menu corresponding to the value of g_currentMenu
        /* g_currentMenu is an index into the menu structure */

        if (user hits the ESC key){
            g_currentMenu--; /* so that the previous menu will be presented */
        }
        else{
            if (user selects a menu){
                Get the menu from the VM database server
                g_currentMenu++;
            }
            if (user selects an application){
                return the number of the selected application
            }
        }
    }while (user does'nt exit the main menu and hasn't selected an application)

    return -1; //exit the main menu
}
```

C10.1.8 Function — SetEnvVar_VMPROG()

When VM_MENU terminates on the selection of an application, the location of the application has to be stored in a safe place. For this reason, an environmental variable called VMPROG is set. If the batch file that starts the application is to be found on drive X: and is called NEWAPP.BAT, then an operation similar to the MS-DOS operation SET VMPROG=X:NEWAPP.BAT is carried out. One difference with the MS-DOS SET operation is that VMPROG is set in all the environments, no matter how many levels it is nested.

C10.1.9 Function — GetAccessRights()

This function does not receive or return any values. It is used to determine the VM group that the user belongs to. The pseudocode is presented below.

```
void GetAccessRights (void)
{
    fill information on all the access groups from the VM database server

    get the object ID for all the VM groups on the home file server

    Get all the groups that the user belongs to on the home file server

    Compare the object IDs of the groups that the user belongs to with the
    object IDs of the VM groups on the home file server until a match is found.

    if (a match has been found){
        set the user's VM group as the matched group
        /* note: every user belongs to only one VM group */
    }
    else the user does not have VM access
}
```

C10.1.10 Function — IntelligentMap()

This function does not receive any parameters. It is used to create a search mapping to a directory on a file server if such a mapping does not already exist. If this operation is successful, the drive letter corresponding to the desired mapping is returned. If an error occurs, the error message is displayed on the screen and the program VM_MENU terminates.

C10.1.11 Function — CleanUp()

This function does not receive any parameters or return any values. It is used to update the Server Attachment Count (SAC) file with current information. It also disconnects from unnecessary file servers.

C10.1.12 Function — MenuScr()

This function uses C-Scape Interface Management routines and is used to display the menus with keyboard and mouse support.

C10.2 Program Listing

```
/*
                                File name: VM_MENU.C

Compiler Model LARGE
C++ options turned to "CPP extension only"

Project File should include the following files:
    vm_menu.c
    lnit.lib
    lib.vm
    t2lowl.lib
    2lcscap.lib

Workstation software that communicates with the database software to
present the virtual machine environment to the user.

By S.J. Isaac

Last revised: 18 May 1993

*/

#include <stdio.h>
#include <mem.h>
#include <string.h>
#include <stdlib.h>
#include <conio.h>
#include <io.h>
#include <dos.h>
#include <nit.h>
#include <nxt.h>

#include "cscape.h"
#include "ostdlib.h"
#include "teddecl.h"
#include "msys.h"
#include "sled.h"
#include "popdecl.h"

#include "h:\c\r\h\uvm.h"


#define MAX_RETRY 3
#define MAX_DB_TABLE_POSITIONS 4
#define MAX_AP_SERVERS 7
#define POP_ATTRIBUTE 0x5b


//_____________________________________________________________________________
//                        Global variable declaration

struct              // Information buffer to send to the database
    {
    BYTE requestType;
    WORD requestNumber;
    BYTE retry;
    char data[REQUEST_SIZE];
    }g_sendDb;

struct              // Information buffer to receive from the database
    {
    BYTE responseType;
    WORD requestNumber;
    char data[RESPONSE_SIZE];
    }g_receiveDb;

struct              // Table of Database servers that have been located
    {
    char name[48];
```

```
    BYTE address[12];
    }g_dbTable[MAX_DB_TABLE_POSITIONS];

struct g_whoAmI    // Particulars about the user running the shell
    {
    char server[SERVER_NAME_LENGTH];
    char userName[USER_NAME_LENGTH];
    BYTE loginTime[7];
    BYTE access;
    char vmName[VM_USER_NAME_LENGTH];
    char vmPassword[VM_PASSWORD_LENGTH];
    }g_whoAmI;

struct g_menuChoice
    {
    BYTE type;
    char name[MENU_NAME_LENGTH];
    WORD ID;
    };

struct             // Menus can be upto 20 levels deep
    {
    char name[MENU_NAME_LENGTH];
    struct g_menuChoice choice[MENU_ITEMS];
    }g_menu[20];

struct
    {
    char name[SERVER_NAME_LENGTH];
    BYTE attachments;
    }g_fileServer[MAX_AP_SERVERS];

struct g_application
    {
    char server[SERVER_NAME_LENGTH];
    WORD appID;
    char batchFile[13];
    }g_application;

ECB  g_ecbReceiveDb;
BYTE g_return_from_db_flag = 0;
WORD g_dbServerType;
BYTE g_numberOfServers = 0;
int  g_dbTablePosition = 0;
int  g_dbSocket;
int  g_numOfAppServers = 0;
int  g_currentMenu = 0;
WORD g_primaryConnectionID;
char g_sacFile[50];
size_t g_whoAmISACSize = SERVER_NAME_LENGTH + USER_NAME_LENGTH + 7;


//_____________________________________________________________________________
//                        Declaration of routines

void Initialise (void);
void far Esr_ReceiveDb (void);
BYTE DatabaseRequest (void);
BYTE LocateDatabaseServer (void);
BYTE GetUserRec (char *server, char *userName);
BYTE GetIDRec (WORD ID, BYTE request);
BYTE GetMenu (WORD ID, int menuNumber);
BYTE GetApplication (WORD ID);
BYTE GetPasswordRec (int rec);
int  ManageMenus (void);
int  SetEnvVar_VMPROG (void);
void GetAccessRights (void);
char IntelligentMap (void);
void CleanUp (void);
int  MenuScr(int menuNumber);
void Scramble(char *plainText, char *c);



//_____________________________________________________________________________

void main (void)
{
```

```
    int  ccode;
    char buf[256];
    BYTE c;

      atexit (disp_Close);

      Initialise ();
      if (GetMenu(0,0) != 0x00) {
          pop_Prompt("Unable to get the main menu from the database",
                     -1,-1,-1,-1, POP_ATTRIBUTE,bd_1);
          exit(1);
      }
      g_currentMenu=0;
      do{
          ccode = ManageMenus();
          if (ccode != -1){

              // an application has been selected

              c = GetApplication(g_menu[g_currentMenu].choice[ccode].ID);
          }
          else {

              //shell has to terminate

              sprintf(buf,"Goodbye %s", g_whoAmI.userName);
              pop_Prompt(buf, -1,-1,-1,-1, POP_ATTRIBUTE,bd_1);
              CleanUp();
              exit(3);
          }
      }while(c != 0x00);
      ccode = SetEnvVar_VMPROG();

      CleanUp();
      exit (0);
}


//_____________________________________________________________________________

void Initialise (void)
{
   BYTE ccode;
   WORD objType;
   char objName[48];
   long objID;
   BYTE loginTime[7];
   WORD connectionNumber;
   FILE *sac;
   char sacExtension[4];
   char sacName[13];
   char serverName[48];
   WORD oldConnectionID;
   int  c;
   char buf[256];
   char *tempEnvVar;


     // initialize the display

     disp_Init(def_ModeText, FNULL);
     hard_InitMouse();
     sedwin_ClassInit();


     // initialize IPX

     ccode = IPXInitialize();
     if (ccode != 0){
         pop_Prompt("IPX is not installed !!",
                    -1,-1,-1,-1, POP_ATTRIBUTE,bd_1 );
         exit(1);
     }


     // set the preferred fileServer to the primary fileServer

     g_primaryConnectionID = GetPrimaryConnectionID();
     if (g_primaryConnectionID != 0){
```

```
    oldConnectionID = GetPreferredConnectionID ();
    SetPreferredConnectionID (g_primaryConnectionID);
}
else {
    pop_Prompt("Cannot find the primary fileServer\nPlease login again",
               -1,-1,-1,-1, POP_ATTRIBUTE,bd_1);
    exit(1);
}


// determine the server attachment count file name

connectionNumber = GetConnectionNumber();
itoa (connectionNumber, sacExtension, 10);
strcpy (sacName, "SAC.");
strcat (sacName, sacExtension);


// open server attachment count file or exit if it cannot be found

if ((tempEnvVar=getenv ("TEMP")) != NULL) {
    strcpy (g_sacFile, tempEnvVar);
    if (tempEnvVar[strlen(tempEnvVar)-1] != '\\')
        strcat (g_sacFile, "\\");
    strcat (g_sacFile, sacName);
    if ((sac=fopen(g_sacFile,"r+b")) == NULL) {
        sprintf(buf,"Unable to open sacFile on %s", tempEnvVar);
        pop_Prompt(buf, -1,-1,-1,-1, POP_ATTRIBUTE,bd_1);
        exit(1);
    }
}


// obtain information about the primary server

ccode = GetConnectionInformation (connectionNumber,
                                  objName,
                                  &objType,
                                  &objID,
                                  loginTime);
fread ((void *)&g_whoAmI, g_whoAmISACSize, 1, sac);
if (stricmp(objName, g_whoAmI.userName) != 0) ccode = 2;
if (memcmp(loginTime, g_whoAmI.loginTime, 7) != 0) ccode = 2;
GetFileServerName (g_primaryConnectionID, serverName);
if (stricmp (serverName, g_whoAmI.server) != 0) ccode = 2;
if (ccode == 2){
    pop_Prompt("Incorrect information in sacFile\nPlease login again",
               -1,-1,-1,-1, POP_ATTRIBUTE,bd_1);
    exit(1);
}


// Fill fileServer structure

fseek (sac, sizeof (SAC_struct), SEEK_SET);
memset(&g_fileServer[0], 0, MAX_AP_SERVERS * sizeof(g_fileServer[0]));
c = 0;
do {
    fread ((void *)&g_fileServer[c], sizeof (g_fileServer[0]), 1, sac);
    c++;
}while (!feof(sac) && c < MAX_AP_SERVERS);
if (feof(sac)) g_numOfAppServers = c - 1;
else g_numOfAppServers = c;
fclose(sac);

g_dbSocket = IntSwap ( (int) DB_SERVER_SOCKET );


// obtain the address of the database

ccode = LocateDatabaseServer();
if (ccode == 0){
    pop_Prompt ("Unable to find a database",
                -1,-1,-1,-1, POP_ATTRIBUTE,bd_1);
    exit (1);
}


// Get access rights from the primary server
```

```
    GetAccessRights ();


    // initialise database request number

    g_sendDb.requestNumber = 0;
}


//_____________________________________________________________________

void far Esr_ReceiveDb (void)
{
    _AX = _ES;
    _DS = _AX;
    if (g_sendDb.requestNumber == g_receiveDb.requestNumber)
        g_return_from_db_flag = 1;
    else IPXListenForPacket( &g_ecbReceiveDb );
}


//_____________________________________________________________________

BYTE DatabaseRequest (void)
{
  ECB         ecbSendDb;
  IPXHeader   ipxheaderSendDb, ipxheaderReceiveDb;
  int         ccode, transportTime;
  struct time start, now;
  float       time_out = 0.1;
  int         c;
  char        buf[256];


            // Initialisations



            // set other variables

    g_sendDb.requestNumber++;
    g_return_from_db_flag = 0;


            // open the workstation socket to communicate with the database

    ccode = IPXOpenSocket ( (BYTE *) &g_dbSocket, (BYTE) SHORT_LIVED );
    if ( ccode ){
        sprintf(buf,"IPXOpenSocket error = %i", ccode);
        pop_Prompt(buf,-1,-1,-1,-1, POP_ATTRIBUTE,bd_1);
    }


            // fill_g_ecbReceiveDb ()

    memset (&g_ecbReceiveDb, 0, sizeof(ECB));
    g_ecbReceiveDb.ESRAddress = Esr_ReceiveDb;
    g_ecbReceiveDb.socketNumber = g_dbSocket;
    g_ecbReceiveDb.fragmentCount = 2;
    g_ecbReceiveDb.fragmentDescriptor[0].address = &ipxheaderReceiveDb;
    g_ecbReceiveDb.fragmentDescriptor[0].size = sizeof(IPXHeader);
    g_ecbReceiveDb.fragmentDescriptor[1].address = &g_receiveDb;
    g_ecbReceiveDb.fragmentDescriptor[1].size = sizeof(g_receiveDb);


    c = g_dbTablePosition;
    do{
            // fill the ipxheader for g_sendDb

        memcpy((void *)&ipxheaderSendDb.destination,
                g_dbTable[g_dbTablePosition].address, 12);
        ipxheaderSendDb.packetType = 4;


            // fill_ecbSendDb ();

        memset (&ecbSendDb, 0, sizeof(ECB));
        ecbSendDb.ESRAddress = NULL;
```

```
        ecbSendDb.socketNumber = g_dbSocket;
        ecbSendDb.fragmentCount = 2;
        ecbSendDb.fragmentDescriptor[0].address = &ipxheaderSendDb;
        ecbSendDb.fragmentDescriptor[0].size = sizeof(IPXHeader);
        ecbSendDb.fragmentDescriptor[1].address = &g_sendDb;
        ecbSendDb.fragmentDescriptor[1].size = sizeof(g_sendDb);

        ccode = IPXGetLocalTarget (g_dbTable[g_dbTablePosition].address,
                                   ecbSendDb.immediateAddress,
                                   &transportTime);
        if (ccode){
            pop_Prompt("IPXGetLocalTarget error in routine DatabaseRequest",
                       -1,-1,-1,-1, POP_ATTRIBUTE,bd_1);
            exit(1);
        }


            // Start listening for a response from the database

        IPXListenForPacket( &g_ecbReceiveDb );


            // send and wait for database to reply

        for (g_sendDb.retry=0; g_sendDb.retry<MAX_RETRY; g_sendDb.retry++){
            IPXSendPacket (&ecbSendDb);
            for (gettime(&start), gettime(&now);
                 time_out > DiffFloatTime(&now, &start);
                 gettime(&now) )  {

                if (g_return_from_db_flag){   // Has the database responded?

                    IPXCloseSocket (g_dbSocket);
                    return (g_receiveDb.responseType);
                }
                IPXRelinquishControl();
            }
        }

        g_dbTablePosition++;
    } while (c != (g_dbTablePosition %= g_numberOfServers));


            // re-obtain the address of the database

    ccode = LocateDatabaseServer();
    if (ccode == 0){
        pop_Prompt ("Unable to find a database",
                    -1,-1,-1,-1, POP_ATTRIBUTE,bd_1);
        exit (1);
    }

    IPXCloseSocket (g_dbSocket);
    return (0xff);
}


//_____________________________________________________________________________

BYTE LocateDatabaseServer (void)
{
  int  oCode, pCode;
  long objectID = -1;
  char objectName[48];
  WORD objectType;
  char objectHasProperties;
  char objectFlag;
  char objectSecurity;
  BYTE propertyValue[128];
  BYTE moreSegments;
  BYTE propertyFlags;

    g_dbServerType =  (WORD) IntSwap ((int)DB_SERVER_TYPE);
    for (objectID = -1, oCode = 0, g_numberOfServers = 0;
        ( !oCode && (g_numberOfServers < MAX_DB_TABLE_POSITIONS) ); ){
        oCode = ScanBinderyObject ("*",
                                   g_dbServerType,
                                   &objectID,
                                   objectName,
```

```
                                    &objectType,
                                    &objectHasProperties,
                                    &objectFlag,
                                    &objectSecurity);
        if (oCode == 0){

            // a database server has been located

            pCode = ReadPropertyValue (objectName,
                                       g_dbServerType,
                                       "NET_ADDRESS",
                                       1,
                                       propertyValue,
                                       &moreSegments,
                                       &propertyFlags);
            if (pCode == 0){

                // The NET_ADDRESS has been obtained

                strcpy (g_dbTable[g_numberOfServers].name, objectName);
                memcpy (g_dbTable[g_numberOfServers].address,
                        propertyValue, 12);
                g_numberOfServers++;
            }
        }
    }

    return g_numberOfServers;
}


//_____________________________________________________________________

BYTE GetUserRec (char *server, char *userName)
{
    memcpy(g_sendDb.data, server, SERVER_NAME_LENGTH);
    memcpy(&g_sendDb.data[SERVER_NAME_LENGTH], userName, USER_NAME_LENGTH);
    g_sendDb.requestType = USER_RECORD;
    return DatabaseRequest();
}


//_____________________________________________________________________

BYTE GetIDRec (WORD ID, BYTE request)
{
    memcpy(g_sendDb.data, &ID, 2);
    g_sendDb.requestType = request;
    return DatabaseRequest();
}


//_____________________________________________________________________

BYTE GetMenu (WORD ID, int menuNumber)
{
  BYTE ccode;

    ccode = GetIDRec (ID, MENU_RECORD);
    if (ccode == 0)  // successful
        memcpy (&g_menu[menuNumber], g_receiveDb.data, sizeof(g_menu[0]));
    return ccode;
}


//_____________________________________________________________________

BYTE GetApplication (WORD ID)
{
  BYTE ccode;
  WORD oldConnectionID, connectionID;
  int  num, i, j, k;
  char driveLetter = '\0';
  char loginPass[128];

  struct g_application *apServer[MAX_AP_SERVERS];
```

```
ccode = GetIDRec (ID, APPLICATION_RECORD);
if (ccode != 0) return 0xf0;  // unsuccessful

num = *(int *)&g_receiveDb.data[0]; //no. of servers having this app

for (i = 0; i < MAX_AP_SERVERS; i++) {
    j = 3+i*sizeof(struct g_application);
    apServer[i] = (struct g_application *)&g_receiveDb.data[j];
}


// check whether correct access rights

if ((g_whoAmI.access & g_receiveDb.data[2]) == 0) return 0xff; // no rights


// check whether primary server appears on the list

for (i = 0; i < num; i++) {
    if (strcmp(g_whoAmI.server, apServer[i]->server) == 0){
        memcpy(&g_application, apServer[i], sizeof (struct g_application));
        return 0x00; // successful
    }
}


// check whether other attached servers appear on list

for (k = 0; k < num; k++) {
    for (j = 0; j < MAX_AP_SERVERS; j++) {
        if (g_fileServer[j].name[0] != '\0') {
            if (strcmp(g_fileServer[j].name, apServer[k]->server) == 0){
                memcpy(&g_application, apServer[k],
                       sizeof (struct g_application));
                return 0x00; // successful
            }
        }
    }
}


// check for first server on the list which is active

for (i = 0; i < num; i++) {
    ccode = AttachToFileServer (apServer[i]->server, &connectionID);
    if (ccode == 0){
        oldConnectionID = GetPreferredConnectionID ();
        SetPreferredConnectionID (connectionID);

        Scramble (g_whoAmI.vmPassword, loginPass);
        ccode = LoginToFileServer (g_whoAmI.vmName, OT_USER,
                                   loginPass);
        if (ccode == 0) {

            // map search drive to sys:public\bin

            driveLetter = '\0';
            ccode = MapDrive (connectionID,
                              0xff,
                              "SYS:PUBLIC/BIN",
                              DRIVE_INSERT,
                              16,
                              &driveLetter);

            if (ccode == 0) {

                // fill application information

                memcpy(&g_application, apServer[i],
                       sizeof (struct g_application));


                // create entry into fileServer attachment buffer

                if (g_numOfAppServers < MAX_AP_SERVERS){
                    strcpy(g_fileServer[g_numOfAppServers].name,
                           g_application.server);
                    g_fileServer[g_numOfAppServers].attachments = 0;
                    g_numOfAppServers++;
```

```
                    }

                    return 0x00;  // successful
                }
            }
        }
    }
    return 0xBB; // could not find any active servers
}


//_____________________________________________________________________________

BYTE GetPasswordRec (int rec)
{
    memcpy(g_sendDb.data, &rec, sizeof (int));
    g_sendDb.requestType = PASSWORD_RECORD;
    return DatabaseRequest();
}


//_____________________________________________________________________________

int ManageMenus(void)
{
  int  select, ap=0;
  BYTE ccode;

    do{
        select = MenuScr(g_currentMenu);
        if (select == 0)  g_currentMenu--;
        else{
            select--;
            if (g_menu[g_currentMenu].choice[select].type == MENU){
                if ((g_currentMenu+1)<20){
                    ccode = GetMenu(g_menu[g_currentMenu].choice[select].ID,
                                    g_currentMenu+1);
                    if (ccode == 0)  g_currentMenu++;
                }
            }
            else
                if (g_menu[g_currentMenu].choice[select].type == APPLICATION)
                    ap=1;
        }
    }while ((g_currentMenu!=-1) && (ap!=1));
    if (ap==1) return select;
    return -1;  //exit main menu
}


//_____________________________________________________________________________

int SetEnvVar_VMPROG(void)
{
  char batchFileString[20];
  char driveLetter;


    //make intelligent mapping to relevant server containing the batch file

    driveLetter = IntelligentMap();


    //Compose string containing driveletter and batchfile name

    memset (batchFileString, 0, sizeof (batchFileString) );
    batchFileString[0] = driveLetter;
    batchFileString[1] = '\0';
    strcat (batchFileString, ":");
    strcat (batchFileString, g_application.batchFile);


    //Environmental variable VMPROG is now set

    SetParentEnv ("VMPROG=", batchFileString);

    return 0;   // successful
```

```
}


//_____________________________________________________________________________

void GetAccessRights (void)
{

  long objID[NUM_OF_ACCESS_GROUPS], checkID;
  int  ccode;
  int  i, j, step, size;
  BYTE propertyValue[128];
  BYTE moreSegments;
  BYTE propertyFlags;
  struct pass
    {
    char group[VM_GROUP_NAME_LENGTH];
    char user[VM_USER_NAME_LENGTH];
    char password[VM_PASSWORD_LENGTH];
    }pass[NUM_OF_ACCESS_GROUPS];


    // fill buffer with information on access groups from the database

    size = sizeof(pass[0]);
    step = RESPONSE_SIZE / size;
    for(i = 0; i < NUM_OF_ACCESS_GROUPS; i += step){
        ccode = GetPasswordRec (i);
        if (ccode != 0){
            pop_Prompt("Unable to get needed records from database",
                       -1,-1,-1,-1, POP_ATTRIBUTE,bd_1);
            exit(1);
        }
        for (j=0; (j < step) && ((j+i) < NUM_OF_ACCESS_GROUPS); j++){
            memcpy(&pass[j+i], &g_receiveDb.data[j*size], size);
        }
    }


    // Get bindery object id for the VM groups on primary server

    for(i = 0; i < NUM_OF_ACCESS_GROUPS; i++){
        if ( pass[i].group[0] != '\0'){
            ccode = GetBinderyObjectID (pass[i].group, OT_USER_GROUP,
                                        &objID[i]);
            if (ccode != 0) {
                pop_Prompt("Get ObjectID error in routine GetAccessRights",
                           -1,-1,-1,-1, POP_ATTRIBUTE,bd_1);
                exit(1);
            }
        }
        else objID[i] = 0L;
    }


    // which of the VM groups does this user belong to?

    ccode = ReadPropertyValue (g_whoAmI.userName,
                               OT_USER,
                               "GROUPS_I'M_IN",
                               1,
                               propertyValue,
                               &moreSegments,
                               &propertyFlags);
    if (ccode != 0) {
        pop_Prompt("ReadPropertyValue error in routine GetAccessRights",
                   -1,-1,-1,-1, POP_ATTRIBUTE,bd_1);
        exit(1);
    }

    j=0;
    do {
        checkID = LongSwap ( *((long *)&propertyValue[j]) );
        for (i = 0; i < NUM_OF_ACCESS_GROUPS; i++) {
            if (i == 0) g_whoAmI.access = 1;
            else g_whoAmI.access *= 2;
            if (checkID == objID[i]){
                strcpy(g_whoAmI.vmName, pass[i].user);
                strcpy(g_whoAmI.vmPassword, pass[i].password);
```

```
                if (i == (NUM_OF_ACCESS_GROUPS - 1)) g_whoAmI.access = 0xff;
                i=256;
                j=256;
            }
        }
        if (checkID == 0){
            pop_Prompt ("You do not have virtual machine access!!",
                        -1,-1,-1,-1, POP_ATTRIBUTE,bd_1);
            exit(1);
        }
        j += 4;
    }while(j < 128);
}


//_____________________________________________________________________

char IntelligentMap (void)
{
  BYTE driveNumber;
  char driveLetter;
  char directoryPath[255];
  char mapPath[50];
  int  ccode;
  WORD connectionID;

    strcpy (mapPath, g_application.server);
    strcat (mapPath, "/SYS:PUBLIC/BIN");


    // Get the connection ID of this server

    ccode = GetConnectionID (g_application.server, &connectionID);

    for (driveNumber = 6; driveNumber < 26; driveNumber++){

        // Does the required search mapping exist

        driveLetter = (char)driveNumber + 65;
        if ((ccode = IsSearchDrive( driveLetter )) == 1){
            if ((ccode = GetFullPath(driveNumber, directoryPath))==0x00){
                if (strcmp(directoryPath, mapPath) == 0){

                    // no need to insert because mapping already exists

                    return driveLetter;
                }
            }
        }
    }


    // required mapping does not exist, so create one

    driveLetter = '\0';
    ccode = MapDrive (connectionID,
                      0xff,
                      mapPath,
                      DRIVE_INSERT,
                      16,
                      &driveLetter);
    switch (ccode){
        case 0x00: return driveLetter;
        case 0x98: printf("\nVolume does not exist");       break;
        case 0x9c: printf("\nInvalid Path");                break;
        case 0xb0: printf("\nSearch Drive Vector is Full"); break;
        case 0xb1: printf("\nDrive does not exist");        break;
        case 0xb5: printf("\nNo Drives available");         break;
        default  : printf("\nError 0x%x in map operation", ccode);
    }
    pop_Prompt ("MapDrive error in routine IntelligentMap",
                -1,-1,-1,-1, POP_ATTRIBUTE,bd_1);
    exit(1);
}


//_____________________________________________________________________
```

```
void CleanUp (void)
{
  int c;
  int handle;
  long size;
  WORD connectionID;
  FILE *sac;

    SetPreferredConnectionID (g_primaryConnectionID);


    // Open SAC file

    if ((sac=fopen(g_sacFile,"r+b")) == NULL){
        pop_Prompt("Unable to open sacFile for writing in routine CleanUp",
                   -1,-1,-1,-1, POP_ATTRIBUTE,bd_1);
        exit(1);
    }


    // Update SAC file

    fwrite ((void *)&g_whoAmI, g_whoAmISACSize, 1, sac);


    // Set sac to start position of server names

    fseek (sac, sizeof (SAC_struct), SEEK_SET);

    for (c = 0; c < g_numOfAppServers; c++){
        if (g_fileServer[c].attachments == 0){

            // disconnect from unnecessary file servers

            if (strcmp(g_fileServer[c].name, g_application.server) != 0){
                GetConnectionID (g_fileServer[c].name, &connectionID);
                DetachFromFileServer (connectionID);


                // insert NULL as fileServer name

                g_fileServer[c].name[0] = '\0';
            }
        }
        if (g_fileServer[c].name[0] != '\0'){
            fwrite((void *)&g_fileServer[c],sizeof(g_fileServer[0]),1,sac);
        }
    }
    handle = fileno (sac);
    size = tell (handle);
    chsize (handle, size);
    fclose (sac);
}


//_____________________________________________________________________________

int MenuScr(int menuNumber)
{
  menu_type menu;
  sed_type  sed;
  int       ret;
  int       count;
  int       c[MENU_ITEMS];
  int       i;

    menu = menu_Open();

    for (count = 0, i = 0; count < MENU_ITEMS; count++){

        //Check for valid menu entry

        if (g_menu[menuNumber].choice[count].ID != 0){
            c[i] = count;
            menu_Printf(menu, "@p[%d,5]@f[%2d. %s]", i, NULL, &menu_funcs,
                        i + 1, g_menu[menuNumber].choice[count].name);
            i++;
        }
    }
```

```
    menu_Flush(menu);

    sed = sed_Open(menu);
    sed_SetColors(sed, 0x1f, 0x1f, 0x74);

    sed_SetBorder(sed, bd_mouse);
    sed_SetBorderColor(sed, 0x13);
    sed_SetBorderTitle(sed, g_menu[menuNumber].name);
    sed_SetPosition(sed, 5, 23);
    if (i == 0){  // in case the menu is empty
        sed_Close(sed);
        return 0;
    }
    if (i < 15) sed_SetHeight(sed, i);
    else sed_SetHeight(sed, 15);
    sed_SetWidth(sed, 30);
    sed_SetMouse(sed, sedmou_Track);

    sed_Repaint(sed);
    ret = sed_Go(sed);

    sed_Close(sed);
    if (ret == 0) return(ret);
    return ((c[ret - 1]) + 1);
}


//_____________________________________________________________________________

void Scramble(char *plainText, char *c)
{
  size_t pLength;
  char p[130];
  char k[130];
  int i;

    strcpy(p, "zc");
    strcpy(k, "kg");

    strncat(p, plainText, 127);
    pLength = strlen(p);

    for (i = 2; i < pLength; i++){
        k[i] = abs((((p[i]*p[i-1])/(p[i-2]+1)) + (k[i-1]+k[i-2])) % 26) + 65;
    }
    for (;i <127;i++){
        k[i] = abs((k[i-1] * k[i-2] / (k[i-3]+4)) % 26) + 65;
    }
    k[127] = '\0';
    strcpy (c, &k[2]);
}
```

APPENDIX C11

L G . C

A special login program to be used in place of NetWare's login program has been developed. This program calls the login utility supplied by Novell. To help describe the program please consider Figure C11.1 below.

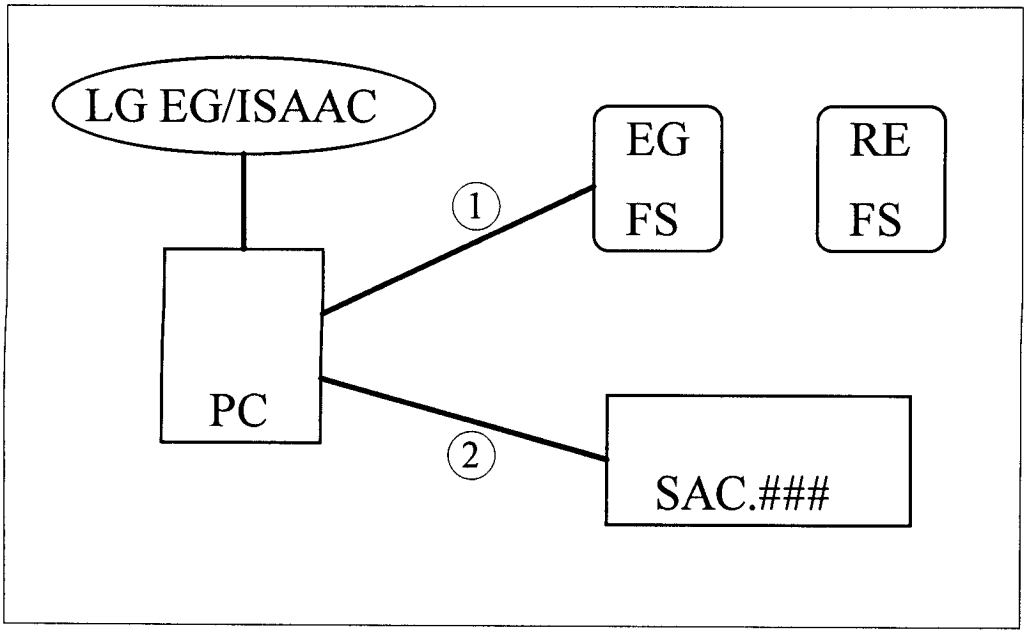

Figure C11.1: Login Program

To login to the VM, a user types:

```
LG <login name>
```

where, the login name is really composed of two names in the format:

```
<file server name>/<user's name>
```

From the example shown in Figure C11.1, the login name is EG/ISAAC (file server name is EG and the user's name is ISAAC). The LG program will try to log the user into a file server called EG which is shown as step one in the figure. (The figure shows two file servers called EG and RE respectively. The file server called EG will try to authenticate the user.) Upon successful login, a Server Attachment Count (SAC) file is created. This is the second step. The SAC file contains information as to the number of unterminated applications on each file server that the user has selected.

The SAC file is created in the directory that the 'TEMP' environmental variable is set to. If the TEMP environmental variable does not exist, the SAC file is created (in order of preference) on the 'C:\TEMP' directory or in the 'H:\' directory. The TEMP environmental variable is set to whichever directory the SAC file was created on.

The name of the SAC file is composed of the letters SAC, a full stop (.), and an extension — which is the connection number to the home file server. If a user has logged into the same file server twice, he will have two SAC files corresponding to the two file server connection numbers of the two PCs he has logged in from. If a file server is running NetWare v 3.10 or 3.11, then a maximum of 250 connection numbers exist. So, if the connection number of a PC to a user's home file server is '7', then the name of the SAC file for that user would be 'SAC.7'.

Each time an application found on a particular file server is started, the attachment count to that file server in the SAC file is incremented by one. When the application ends, the attachment count is decremented by one. In this way, it is possible to determine when a connection to a file server has to be terminated.

The SAC file is created by the login program because it is easy to determine the home file server at this stage. Afterwards, a program could change the home file server, and then it would be difficult to locate information about the user. When the login program ends, the SAC file only contains the home file server's name, the user name, the user's

login time, the drive attachment count array, and the segment address of the parent's Program Segment Prefix (PSP).

The first three items in the SAC file are for the purpose of establishing whether the SAC file is the correct SAC file for the session. The home file server provides information on who is logged into that file server on that connection. The user's name and the user's login time from the file server are compared with the information in the SAC file before any of the other information is used.

The parent's PSP is stored so that all the environmental blocks from the parent onwards can be located later. It is possible to have many copies of the MS-DOS command processor 'COMMAND.COM' in memory, and each copy would have an environmental block associated with it. Every program that is run under a particular copy of the 'COMMAND.COM' would normally inherit the MS-DOS environment at that level. In this way, it is possible (and sometimes happens) that the user will be operating several levels away from the original parent's environment. Some programs need to set an environmental variable to a value in all the environmental blocks from the current environment up to the original parent's environment. An example of this is the 'MP.EXE' program (discussed above under Structure of Application Batch Files). The normal method in MS-DOS PCs, to chain backwards from the current PSP up to the original PSP, does not work. The NetWare redirector shell (NETx.COM) seems to change the field in the current PSP that points to its parent. Consequently, it is only possible to go back one level using the standard method. To solve this problem, another method using the original parent's PSP and the MS-DOS Memory Control Blocks (MCBs) has been devised. This method needs the location of the original parent's PSP to work and it seems best to place this in the SAC file.

The structure of the SAC file is given in Table C11.1.

NAME	SIZE	TYPE
Home File Server's Name	SERVER_NAME_LENGTH	character string
User's Name	USER_NAME_LENGTH	character string
login time	7 bytes	Array of BYTES
drive attachment count	26 bytes	Array of BYTES
Original Parent PSP	2 bytes	WORD
file server one	SERVER_NAME_LENGTH	character string
Number of Attachments to file server one	1 byte	BYTE
file server two	SERVER_NAME_LENGTH	character string
Number of Attachments to file server two	1 byte	BYTE
file server three	SERVER_NAME_LENGTH	character string
Number of Attachments to file server three	1 byte	BYTE
.		
.		
.		

Table C11.1: Structure of the SAC File

For the program LG to work, the original login program supplied by Novell should be renamed to IN180393.EXE. The weird rename is done to prevent the average user from running this program

C11.1 Program Listing

```
/*

                                File Name: LG.C
Compiler model COMPACT

Project file should include the following files:
    lg.c
    lib.vm
    cnit.lib


    This is a program to log a user into a file server.

    •   It is used instead of the login utility supplied by Novell.
    •   It calls the login utility supplied by Novell.
    •   This program must be found in whichever directory LOGIN is found

    •   In addition to logging in a user to a fileserver, it also creates
        a Server Attachment Count (SAC) file on (in order of preferrence):

        1    %TEMP%\sac.xxx    (TEMP is an environment variable)
        2    c:\temp\sac.xxx
        3    h:\sac.xxx

            (h: is map rooted to the user's home directory by the
            login script and xxx is the connection number of the
            workstation to the file server)


    •   The structure of the SAC file is:

        ------------------------------------------------------

        1    primary server name           SERVER_NAME_LENGTH
        2    user name                     USER_NAME_LENGTH
        3    login time                    7  bytes
        4    drive attachment buffer       26 bytes
        5    parent PSP segment address    2  bytes

                    --- ******* ---
        6    server1                       SERVER_NAME_LENGTH
        7    Number of attachments on 1    1  byte
        8    server2                       SERVER_NAME_LENGTH
        9    Number of attachments on 2    1  byte
        10   server3                       SERVER_NAME_LENGTH
        11   Number of attachments on 3    1  byte

        ------------------------------------------------------


    by S.J. Isaac
    Last Revised: 26 April 1993
*/


#include <stdio.h>
#include <conio.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include <dir.h>
#include <dos.h>
#include <nit.h>

#include "h:\c\r\h\uvm.h"

#define LOGIN  "IN180393"    // Novell's Login utility
```

```
char g_sacFile[50] = "";


//_____________________________________________________________________
//                    Declaration of routines

void DisplayUsage (void);
void CreateSacFile (void);
void Lgo (void);


//_____________________________________________________________________
//                    Main

int main (int argc, char *argv[] )
{
  int ccode;
  char *c, buf[256];

    puts ("");

    if (argc < 2)  DisplayUsage();

    // logout from all file servers

    Lgo();


    // attempt to login using the login utility supplied by Novell

    ccode = spawnvp(P_WAIT, LOGIN, argv);
    if (ccode != 0) {
        strcpy (buf, argv[0]);
        strcpy (buf+2, LOGIN);
        ccode = spawnv(P_WAIT, buf, argv);
        if (ccode != 0) {
            puts ("Unable to LOGIN to file server!");
            exit(1);
        }
    }

    // successful login, so create a SAC file

    CreateSacFile();

    return 0x00;
}


//_____________________________________________________________________
//                    DisplayUsage

void DisplayUsage (void)
{
    puts ("-----------------------------------------");
    puts ("Usage:       LG <user name>");
    puts ("  example    LG eg/james");
    puts ("-----------------------------------------");
    exit(1);
}


//_____________________________________________________________________
//                    CreateSacFile

void CreateSacFile (void)
{
  FILE *sac;
  int  ccode;
  WORD objType;
  char objName[48];
  char fileServerName[48];
  long objID;
  BYTE loginTime[7];
```

```
char sacName[8];
char sacExtension[4];
int  connectionNumber;
char driveBuffer[26];
char envVar[12];
char *tempEnvVar;
PSP_struct far *parentPSPPtr;
unsigned int far *parentEnvPtr;


  // make a far pointer to the psp of LG

  parentPSPPtr = (PSP_struct far *) MK_FP (_psp, 0x00);


  // make a far pointer to the PSP of LG's parent

  parentPSPPtr = MK_FP (parentPSPPtr->pspseg, 0x00);


  // determine the SAC file name

  connectionNumber = GetConnectionNumber();
  itoa (connectionNumber, sacExtension, 10);
  strcpy (sacName, "SAC.");
  strcat (sacName, sacExtension);


  // open SAC file using TEMP environment variable

  ccode = 0;
  if ((tempEnvVar=getenv ("TEMP")) != NULL) {
      strcpy (g_sacFile, tempEnvVar);
      if (tempEnvVar[strlen(tempEnvVar)-1] != '\\')
          strcat (g_sacFile, "\\");
      strcat (g_sacFile, sacName);
      if ((sac=fopen(g_sacFile,"wb")) == NULL) ccode = 1;
  }else ccode = 1;


  // if unable to open SAC file on TEMP environment variable

  if (ccode != 0){
      strcpy (g_sacFile, "c:\\temp\\");
      strcat (g_sacFile, sacName);
      if ((sac=fopen(g_sacFile,"wb")) == NULL){
          strcpy (g_sacFile, "h:\\");
          strcat (g_sacFile, sacName);
          if ((sac=fopen(g_sacFile,"wb")) == NULL){
              puts("unable to create server attachment count file");
              puts("on either c:\\temp or h:\\");
              Lgo ();
              exit(1);
          }else strcpy (envVar, SAC_PATH_ON_H_DRIVE);
      }else strcpy (envVar, SAC_PATH_ON_C_DRIVE);


      // check whether environment variable needs backslash at end

      if (strcmp(envVar, SAC_PATH_ON_C_DRIVE) == 0)
          envVar[strlen(envVar)-1] = '\0';


      // make a far pointer to the environment of LG's parent

      parentEnvPtr = MK_FP (parentPSPPtr->envseg, 0x00);


      // Set TEMP environment variable

      SetLevelEnv (parentEnvPtr, "TEMP=", envVar);
  }


  // Obtain information to store in SAC file

  GetFileServerName(0, fileServerName);

  ccode = GetConnectionInformation (connectionNumber,
                                    objName,
```

```
                                        &objType,
                                        &objID,
                                        loginTime);
    if (ccode != 0){
        printf("Error %i in GetConnectionInformation()\n", ccode);
        exit(1);
    }


    // Store in file

    fwrite ((void *) fileServerName, SERVER_NAME_LENGTH, 1, sac);
    fwrite ((void *) objName, USER_NAME_LENGTH, 1, sac);
    fwrite ((void *) loginTime, 7, 1, sac);

    memset ((void *)driveBuffer, 0, 26);
    fwrite ((void *)driveBuffer, 26, 1, sac);

    fwrite ((void *)&parentPSPPtr->pspseg, sizeof (unsigned int), 1, sac);

    fclose (sac);
}



//_____________________________________________________________________________
//                      Lgo

void Lgo (void)
{
    if (g_sacFile[0] != '\0') {
        if (remove(g_sacFile) != 0){
            perror("could not delete server attachment count file");
        }
    }
    Logout();
}
```

APPENDIX C12

L G O . C

The program LGO deletes the Server Attachment Count (SAC) file created for the current session and then logs out the user from all attached file servers to terminate the current session. LGO uses Novell's logout utility. For LGO to work, the logout utility supplied by Novell should be renamed to OT180393.EXE. The rename operation will prevent ordinary users from logging out using Novell's logout program (otherwise SAC files will not be deleted) as well as enable LGO to find Novell's logout program.

C12.1 Program Listing

```
/*
```

File Name: LGO.C

```
Compiler model COMPACT

Project file should include the following files:
    lgo.c
    cnit.lib
```

```
    This utility logs a user out of the VM system
    .    It is used in place of Novell's LOGOUT utility
    .    It deletes the server attachment count file before logging out


    by S.J. Isaac

    Last Revised: 20 April 1993
*/



#include <stdio.h>
#include <stdlib.h>
#include <process.h>
#include <string.h>
#include <nit.h>


#define LOGOUT  "OT180393"    // Novell's Logout utility


char g_sacFile[50];


int GetSacName (void);


//---------------------------------------------------------------------------
//                  Main

void main (int argc, char *argv[])
{
  int ccode;
  char *c, buf[256];

    puts ("");

    ccode = GetSacName();
    if (ccode == 0){
        if (remove(g_sacFile) != 0){
            perror("could not delete server attachment count file");
        }
    }


    // Actual logout

    ccode = spawnvp(P_WAIT, LOGOUT, argv);
    if (ccode != 0) {
        strcpy (buf, argv[0]);
        c = strrchr (buf, '\\');
        strcpy (c+1, LOGOUT);
        ccode = spawnv(P_WAIT, buf, argv);
        if (ccode != 0) {
            puts ("Unable to LOGOUT from file server");
            exit(1);
        }
    }
}
```

```
//---------------------------------------------------------------------------
//                    GetSacName

int GetSacName (void)
{
  FILE *sac;
  int  ccode;
  char sacName[8];
  char sacExtension[4];
  int  connectionNumber;
  char *tempEnvVar;

    SetPreferredConnectionID (GetPrimaryConnectionID());


    // determine the server attachment count file name

    connectionNumber = GetConnectionNumber();
    itoa (connectionNumber, sacExtension, 10);
    strcpy (sacName, "SAC.");
    strcat (sacName, sacExtension);


    SetPreferredConnectionID (GetDefaultConnectionID());


    // open server attachment count file or return if it cannot be found

    if ((tempEnvVar=getenv ("TEMP")) != NULL) {
        strcpy (g_sacFile, tempEnvVar);
        if (tempEnvVar[strlen(tempEnvVar)-1] != '\\')
            strcat (g_sacFile, "\\");
        strcat (g_sacFile, sacName);
        if ((sac=fopen(g_sacFile,"wb")) == NULL) {
            puts ("unable to locate server attachment count file");
            printf("on %s \n", tempEnvVar);
            return 1; //failed
        }
    }

    fclose (sac);

    return 0; //success
}
```