

COMPUTER SECURITY IN THE UNIX OPERATING SYSTEM
AND THE INGRES DATA BASE MANAGEMENT SYSTEM

by

LORI LYNN SABRACK

B.A. and B.S., Miami University, 1980

A MASTERS THESIS

submitted in partial fulfillment of the

requirements for the degree

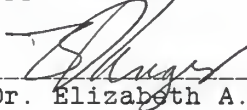
MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1987

Approved by



Dr. Elizabeth A. Unger

LD
2668
.T4
CmSC
1987
522
C. 2

ALL207 309634

CONTENTS

Chapter 1 - Computer Security.....	1
1.1 Introduction.....	1
1.2 Computer Security.....	1
1.2.1 Background.....	1
1.2.2 Types.....	2
1.2.3 Issues.....	3
1.3 UNIX.....	4
1.3.1 Introduction.....	4
1.3.2 File System Structure.....	4
1.3.3 Security Weaknesses and Strengths.....	5
1.3.4 System Administrator.....	7
1.4 INGRES.....	7
1.4.1 Introduction.....	7
1.4.2 INGRES Files.....	8
1.4.3 Security Weaknesses and Strengths.....	9
1.4.4 INGRES Implementation.....	10
1.5 Commercial Security Methods Available.....	11
1.5.1 Introduction.....	11
1.5.2 Netlock.....	11
1.5.3 Memory Cards.....	11
1.6 Current Security Problem.....	12
1.7 Remaining Chapters.....	12
Chapter 2 - Requirements.....	13
2.1 Introduction.....	13
2.2 General Requirements.....	13
2.3 Specific Requirements.....	14
2.3.1 Front End Processor to INGRES Related UNIX Commands.....	15
2.3.2 Front End Processor to INGRES Commands.....	15
Chapter 3 - Detailed Design.....	16
3.1 Introduction.....	16
3.2 Interface to the User.....	16
3.3 UNIX Commands.....	17
3.4 Error and Interrupt Handling.....	27
3.5 Control Flow.....	28
3.6 Commands Excluded.....	31
Chapter 4 - Implementation.....	32
4.1 Introduction.....	32
4.2 Variables.....	32
4.3 Routines.....	33
4.4 C Programs.....	41
Chapter 5 - Conclusion.....	42

5.1	Summary.....	42
5.2	Problems.....	42
5.3	Further Extensions.....	43

LIST OF FIGURES

Figure 3-1.	Menu of INGRES Related UNIX Commands.....	17
Figure 3-2.	Options for creatdb Command.....	19
Figure 3-3.	Options for destroydb Command.....	19
Figure 3-4.	Options for helptr Command.....	20
Figure 3-5.	Options for ingres Command.....	22
Figure 3-6.	List of User's Relations.....	23
Figure 3-7.	Options for printr Commands.....	24
Figure 3-8.	Options for purge Command.....	25
Figure 3-9.	Options for restore Command.....	26
Figure 3-10.	Options for sysmod Command.....	26
Figure 3-11.	Relations for sysmod Command.....	27
Figure 3-12.	High Level Design of INGRES related UNIX Commands.....	29
Figure 3-13.	Expanded High Level Design of INGRES related UNIX Commands.....	30
Figure 4-1.	Control Flow for INGRES Data Base Commands.....	35
Figure 4-2.	Control Flow for INGRES Relation Commands.....	36
Figure 4-3.	User Interface and File Access.....	37

Chapter 1 - Computer Security

1.1 Introduction

Computer Security has become an increasingly important subject in today's society where the use of computers has been incorporated either directly or indirectly in all phases of everyday life. This paper addresses current computer security breaches as well as methods used to enhance computer security. The INGRES (INTERactive Graphics and Retrieval System) Data Base Management System (DBMS) operating under the UNIX* operating system is the target for security enhancements discussed in detail in this paper and the one implemented at Kansas State University. In preparation for such an implementation, INGRES, UNIX and their relationship are discussed in terms of current security measures, strengths and weaknesses and an example of current usage. Finally, suggestions are made for future possible enhancements to help eliminate security problems in the INGRES environment.

1.2 Computer Security

1.2.1 *Background* The technological advances that have taken place in the area of computers have forced computer security to expand. In the early systems, users had access to all physical resources and what protection that did exist on the machines was done at the highest level although most computation was done at a much lower level. As the early systems matured, there was a need for users to share resources as machines were able to support many more users and a variety of functions such as information processing and computational services. In addition to resource allocation becoming a major necessity, it was becoming equally evident that steps toward computer security and protection were very much a concern. [Dep79]

Not only is the mere increase of users a major contributor to the growing concern of computer security, but the

* UNIX is a trademark of Bell Laboratories

capability of the users is alarming. The Nihilist Order, a group of teen-age computer hackers in California, has succeeded in breaking into computer systems and successfully produced lists of credit cards and telephone calling card numbers. The impetus behind their actions is more a question of conquering the system than a question of greed. The status gained among their peers is the reward. Although computer crime consulting firms exist to search out such illegal acts, and are often successful at preventing hackers from accessing corporate systems, it does not end there. The members of the firm are then frequently harassed as the hackers access their personal numbers in an attempt to get instant revenge.

The actual amount of both users and criminals indicates that there is a problem. It is reported by the Communications Fraud Control Association that hackers are responsible for \$500,000 worth of phone fraud per year. [Col86] In addition, for the small business sector, estimated at a total of 16 million installations by 1986, \$300 million is lost in fraud every year. [Kat83,Nat85] The need applies in the private businesses but is more urgently a problem in government applications as they traditionally lag behind. At least one-half of the more than 17,000 computers in the Department of Defense (DOD) need stricter access controls. In the Fall of 1985, President Reagan signed a directive that established an organization that is responsible for government wide computer security policies. The Computer Security Center, was formed to serve the DOD in 1981 and now serves on a national level, develops standards and demonstrates methods to best handle various security problems. [Pet85] It is clear that the need exists for tighter controls and that, although issues are being considered, much work still needs to be done.

1.2.2 *Types* The concept of computer security engulfs many areas of risk that can be classified into the 6 categories described here. Natural disasters such as unexpected power surges, or complete power outages, can threaten secure data, but precautions can be used to minimize losses. Complete sets of back-up files stored on tapes or disks should be made regularly. In addition, devices to prevent loss of data on magnetic media during power failures should be incorporated along with emergency generators to operate air conditioning units. The second category, labeled as terrorism, would include those cases where the actual destruction of data is involved and surveillance equipment is one means by which this problem can be attacked. The

next category, theft by computer, is likely to be the most popular security risk. This includes such things as stealing copyrighted software which is considerably easier as networks become more popular. [Kat83] Also, companies such as Central Point Software, Inc., sell lock-busting schemes to computer owners who are then able to copy programs such as LOTUS 1.2.3, used in business applications. Copying of software is illegal unless the people who buy the software are producing back-up copies for their own use. It is estimated, however, that there are as many as nine illegally copied versions of software in use for each legitimate copy. [Gra86] Theft would also include the illegal retrieving or modifying of data bases, such as employee records or bank account data. Because of such intelligent illegal data manipulations, creative and sophisticated security systems are emerging. Another category of security risk is weakness in software design. The duties of the software designer are numerous and often include understanding the previously manual operations, understanding the capabilities of the computer, designing a system to automate those manual efforts most efficiently, writing and testing computer programs to accomplish these tasks, documenting the system, training customers to properly use it and supporting the system. This long list of duties allows many opportunities for a weakness to be incorporated and security can be threatened. The fifth category includes honest errors which will likely always occur but which may be minimized through better verification and testing procedures. [Kat83] The final category relates directly to the hardware rather than the software. Physical security, including locks on the doors to the computer room, alarm systems, guards and fire prevention, must be a serious consideration as access to the actual hardware by unauthorized personnel may be a risk to a secure system.

1.2.3 *Issues* As has been discussed, the advent of technological advances in the use of computers has created a greater need for security and protection mechanisms. Several issues arise, however, when discussing the implementation of these measures. As the computer's capabilities become more versatile, more first-time users than ever are accessing computers and their peripherals. An extension of that idea indicates that more word processing is currently being done on computers which indicates that a large amount of classified company information is being processed without proper security controls. One of the biggest obstacles in computer security is that the user does not practice good security techniques. For example, the misuse of passwords is prevalent in today's systems where

users maintain easy to determine passwords, share them with other users or seldom change them. The log on procedure must include the use of a password, but it must be safe and simple without being too rigid, or ways to avoid it will be developed. There are two basic ways in which to handle password distribution. The first relies on the user changing his own password which often leads to the use of a common word, name or address which is often easily decipherable. This may or may not include password aging by the system where the password expires and must be changed by the user after, say, 30 days. The second method maintains that the system is responsible for distributing new, random passwords. Concerns here are that this may not be a secure and rapid means and that passwords generated may be difficult to remember. In addition, there is little to no instruction for new or, for that matter, existing users how to best handle security problems and there is a reluctance in naming computer security officers to handle such tasks. Another time-consuming job to better ensure computer security, although it is often bothersome, is the making of back-up disks. It is essential, however, to maintain these disks as theft, human errors or even natural disasters can occur. [Hig83] Looking more into the internals, the need also exists for security modules to perform in the same dynamic way as application programs. They must be able to add, delete and modify users to secure files and provide administrative control over password changes. There must also be an effective, prompt way in which to report security violations as they occur. An audit trail of all recent transactions is often a means of helping conduct a search for wrongdoing.

1.3 UNIX

1.3.1 *Introduction* The UNIX operating system developed in the late 1960's by Bell Laboratories, although not originally developed with security in mind, is a relatively secure system. It allows the system administrator some flexibility in making the system secure or not. It is, then, the administrator's duty to find the balance between an environment that restricts the users as much as necessary and one in which users share data.

1.3.2 *File_System_Structure* The UNIX file system structure provides the hierarchically-organized directories and files. Normally, the file system divides each disk drive into 1024-byte blocks (although, it may vary between 512 and 8192) numbered 0 to the number of possible blocks on that disk. Block zero is the boot block and is not used by the

file system. Block one is the super block which contains the size of the disk, and the sizes of the two remaining sections of the disk. The next section is the i-list which is of variable length and contains i-nodes. An i-node is a 64-byte table containing information about a file such as it's size, owner and permissions, whether it is an ordinary file, directory or a special file. In addition, the i-node also contains the disk address list which is a list of 13 block numbers, the first 10 of which are the first 10 blocks of the file. The eleventh block number gives the number of a block that contains up to 256 more block numbers and similarly for the twelfth and thirteenth block numbers. Although this would allow files of enormous size, UNIX places a more practical limit on the maximum size. All devices, as well as files, directories, disks and memory, are thought of as files. When actions are requested by programs on devices files, UNIX translates it into actions on the actual devices. This results in the devices being treated independently as files which enhances security because all I/O for the device passes through channels and users cannot access the devices directly. [Woo85]

1.3.3 *Security Weaknesses and Strengths* UNIX was developed, as were most systems, before the recent urge to include security measures, so it admittedly has some weaknesses in its design. Probably the biggest flaw UNIX has is in crash protection or in the handling of conditions on the machine that temporarily cripple the system's operations. The culprit here is the lack of checking for the allocation of resources and exceeding the limits. This can result in disaster for the system if done by a mischievous user or even malfunctioning programs and there is no easy solution. It is, however, relatively easy to determine the cause of the disaster, or identify the culprit and act on it. In the case of unauthorized users accessing data, the degree of security is more adequate. Eleven bits of protection information along with a user identification number and a user group number (UID and GID) are associated with each UNIX file. Nine bits of information specify permission to read, to write and to execute the file to the owner, the owner's group and to all other users. In addition, the GID and UID bits allow developers to write programs which will be executed by users and will maintain files accessible to users only by that program. The idea of permission bits for a directory has slightly different meaning than for files. If a user has permission to execute a directory, this actually means permission to search the directory for a given file in order to access files in that directory. Write permission of a directory is translated to

mean that creation and deletion of files may take place. [Rit83] File protection information can be modified with the three commands chown, chmod and chgrp which determine who may read, write or execute the file. The rm command serves to delete files and, although it will ask for confirmation on a file for which the requester does not have write permission, the file can be removed, regardless of the mode or owner. [Woo85]

UNIX maintains a single user, called the "super-user", who has the ability to read any file and write any non-directory. It can also change protection modes, owner UID and GID bits and may execute privileged system calls. Since so much power is given to one user, it is clear that this is a flaw in the security system.

Another means exists of attaining super-user status. After a disk pack or tape has been mounted as a file system, the system will accept what it contains. Therefore, by mounting a device, an authorized user may be able to corrupt a system. Disallowing the mounting capabilities appears to be the only solution. [Rit83]

There is an extension to the file protection modes that enhances security. The umask command enables users to allow their files to be as accessible by others as they wish. All files subsequently created are given permissions based on the user's default creation mask and can be specified to be as lenient or strict as desired. This gives an individual the ability to impose a restriction on the users of its files.

UNIX also maintains a crypt command that affords greater protection using an encryption mechanism. A key is associated with the scrambling of the file to ensure privacy and both the command and the key are again used to decrypt the file. In addition, the file may be packed before encrypting which will serve both to add more security and save space by using file compression.

In order to maintain different divisions of users, groups are used to logically bind users together. Passwords can also be attached to groups to restrict those users not in the group from changing to it. The newgrp command will

allow this change to the restricted group only if the password is input correctly.

UNIX performs probably better than most systems on the issue of password security. Each user is associated with a password that must be entered (but is not written to the user's screen) at each login and is stored in encrypted form. [Woo85]

1.3.4 *System Administrator* The System Administrator is responsible for taking care of a system by keeping unauthorized people off the system, keeping users from accessing each others sensitive information, preventing integrity loss and denying service to excessive resource requests. UNIX offers assistance in many of these areas. Auditing programs can be found in UNIX that locate inconsistencies and security violations. They check such things as device files, system files writable by anyone, logins without passwords, and those logins that have not been recently used.

Users can also be required to periodically change their passwords. A password aging technique mandates a maximum number of weeks that the password is valid and a minimum number of weeks that must transpire before the password may again be changed. In addition, the users should be made aware that the passwords chosen should not be simple, easy to decipher words like first name, birth date or address. Users should also be made aware of any tools used to increase security, and better practices to avoid corruption.

1.4 INGRES

1.4.1 *Introduction* The INGRES DBMS is a relational data base developed at the University of California at Berkeley to be run under the UNIX operating system. Primarily programmed in C, INGRES allows users to access their data which is represented by a collection of tables in the data base. Users are able to interact with even very large data bases through the high-level QUERy Language called QUEL. It is a powerful calculus based language that allows actions to be performed on a data base based on arithmetic functions, set valued functions or aggregate functions. [Hel75] The output is in table form and examples of particular functions include create, append, delete, and replace relations, permit the use of your relation for other users and define integrity constraints on relations. In addition, INGRES

controls concurrency so that many users can access a data base simultaneously if the data base security and protection controls allow access. To load or unload a data base of its data, utilities are provided, and system resources are also monitored to provide better control and performance. INGRES maintains an integrated data dictionary that contains all system information, such as the tables defined and the column names associated with those tables.

The underlying problem with designing relational data bases is that users must decide what data must be represented in their data bases and, once accomplished, may result in several possible representations which can lead to inconsistencies in the data base. This, however, merely requires that the design of the data base be studied and chosen efficiently. [Row82]

1.4.2 *INGRES_files* When installing INGRES, the user must first define an INGRES "super-user", called *ingres*, on the machine who will ultimately own all of the INGRES data bases and all of the running software. After logging in as that user, the INGRES tapes are copied in and can be run from the parent directory of the user *ingres* (denoted by *.../*) chosen by the system administrator. There is a collection of programs in *.../bin* which are executed by the INGRES programs, a library used to compile user programs and a concurrency device to be installed in the kernel. The directories necessary for proper execution of INGRES (located in *.../*) are:

<i>bin</i>	binary files constituting INGRES
<i>files</i>	files used by INGRES
<i>data/base</i>	data bases created by users
<i>demo</i>	demonstration package

The files under *.../data/base* are files created at the time each data base is created. They are of four types. An administration file contains the initialization information and the user identification of the data base administrator (DBA). The system relations files have predefined names and are owned by the DBA. There are 6 of these system relation files or catalogs for every data base on the system. They contain information relating to each tuple in the data base, individual domains of every relation and secondary indices in the data base. Also there are two catalogs, protection and integrity, that store the respective constraints for each relation. The third type of file created for each data base stores protection predicates specified by the DBA to

authorize accessibility to data bases. There are also other files created that are not shared by the DBA and serve as temporary storage files. Additional, but unnecessary, directories for minimally using INGRES that are given to the user on the INGRES tape are:

doc	documentation
lib	object file libraries
source	INGRES system source code

A "users" file is maintained by INGRES and contains information regarding the name, identification number, status and permission rights of each user. An INGRES user must be entered into this file which can be thought of as similar to the UNIX /etc/passwd file. [All81]

1.4.3 Security Weaknesses and Strengths A definite need for data base protection exists because it is estimated that 85% of all computer activity involves data handling. Much of this handled data is sensitive and unauthorized data access could be extremely serious. The longer the wait to solve the security problem, the less likely it will be that cost effective solutions will be available.

One approach to the problem uses kernel architecture whereby all functions pertinent to security are included in the kernel while all functions that are not important are excluded. It is not yet clear that a kernel design for secure data management will minimize the amount of code involved in protection and still allow all the necessary functions of data independence, good performance, flexible reorganization and sophisticated query language to continue. The major reason for instituting a kernel architecture was the difficulty in program verification for large pieces of code. The design was applied to INGRES because of its availability in both the particular development and university environments, its structured programming, and its ease in the retrofitting procedure since it was a relational model. The conclusions found from this implementation were that the retrofit was done rather easily, however, other functions of a DBMS affecting security, such as back-up recovery, and a general security policy were not included. This method met with arguments from one of the key persons responsible for developing INGRES, Michael Stonebraker, as to its feasibility. [Dow79]

The standard INGRES DBMS makes use of the UNIX file protection method in its handling of data bases as files. When INGRES is initialized on a system and the user ingres is created, any data base creations form files listed in .../data/bases. These files list ingres as the owner and the actual creator of the data base (the user who called "ingres" and the "creatdb" UNIX commands) is listed in these files as the DBA for all future references. This is done through the use of the UNIX sticky bit, whereby the user's effective ID is set to ingres, so any files created at this point will belong to ingres and, in this case, with an "owner read, owner write, no other access" permission. Therefore, these files cannot be modified or even perused by anyone other than the DBA, ingres and the system's super-user. In this way, it is difficult to access an INGRES data base other than to execute "ingres". [Sto76]

By the use of some QUEL commands, however, a DBA can add permissions for users to retrieve, replace, delete or append relations. The use of these commands, then, automatically creates the possibility of security problems. The more users given access to a data base, the greater the possibility that the data base can be corrupted by an unauthorized user. A command to be used by the DBA is also available to clean up temporary system relations, remove extraneous files and either report on or destroy expired relations. This helps keep a more secure data base as unnecessary information is discarded. [Sie85]

1.4.4 *INGRES_Implementation* The Comprehensive Epilepsy Program (CEP) which began in 1980 in California was able to handle all aspects of data management and analysis. Flexibility and generality in the operations of the facilities were important since CEP is composed of several groups ranging from social science research to biomedical research. The needs of the facility in the area of data handling included:

- a. data input must be fast and precise; a variety of forms must be acceptable and a large volume of data was to be expected.
- b. different forms containing information about the same patient must be organized easily and properly into the data base.

- c. transferring of data to and from the DBMS to analysis programs must be relatively easy to do.

The INGRES DBMS was chosen to be used and it was found to be reasonably flexible, its input and output could be sent directly to other UNIX programs via the "pipe" capability, however, the users found the command language QUEL to be somewhat difficult to use and not very "user-friendly". The data base security never seemed to become a problem in this environment, but rather it was the communication of the employees at the data handling center of the CEP with the other areas that was the problem. [Gre85]

1.5 Commercial Security Methods Available

1.5.1 *Introduction* The current solutions to security problems are more commonly including a combination of software and hardware. An additional peripheral is included that must communicate with the software in order to better protect the system.

1.5.2 *Netlock* One example of a new hardware peripheral is the Netlock System by Datakey, Inc. which is a security device allowing customer control over remote access to host computers. The Netlock authenticates the remote users' access and is able to determine who is trying to access the system and from where. The Netlock system consists of an intelligent box that accepts a physical key from the user. The key is encrypted with information determining what access that user has on the system. The key's contents are read, and, based on those readings, access into the system is granted or denied. This system may be particularly helpful in a network environment, although it can also be used to "lock" users from particular features or, in this case, data bases. The software must be modified to validate the key's information and determine if accessibility should be granted. This may not be an extremely feasible solution to the data base security problem, but it may be a viable product for other security problems. [Dat85]

1.5.3 *Memory_Cards* Similar to the Netlock implementation, memory cards are used to verify a user's capability to access a particular entity. That entity can be a bit of information, a file or an entire machine. The current generation of cards, typically taking the shape of a credit card, contains both a microprocessor and memory to provide encryption and adequate storage of access information. Again this may not be a feasible solution to this security concern, but it will likely have many applications. [Fis85]

1.6 Current Security Problem

Given the above information, the problem exists whereby an unauthorized user may be able to access highly sensitive data in a particular data base. If a user is able to create relations in a data base and disable all permissions, the data is thought to be secure. If, however, other unauthorized users are able to illegally and rather easily acquire passwords which allow them access to such data bases, added security measures are necessary. Although INGRES maintains certain security measures, additional steps must be taken. The implementation discussed in this paper enhances INGRES security by placing another security measure on the data bases.

1.7 Remaining Chapters

The remainder of this paper deals with the implementation of additional security measures to the INGRES DBMS at Kansas State University. Chapter 2 supplies both the general and specific requirements of the implementation. Following in Chapter 3 is the Design and Chapter 4 is the actual code constituting the implementation. Chapter 5 then deals with future possible work in computer security and a conclusion of the paper.

Chapter 2 - Requirements

2.1 Introduction

The current security measures of the INGRES DBMS, although adequate for most data bases, can be enhanced by an inquiring front end program. The need exists, especially at an institution like Kansas State University, for those data bases that have highly privileged information such as grades and personal records.

2.2 General Requirements

Although permission can be granted or denied on a relation basis in INGRES, the opportunity exists whereby an unauthorized user may be able to either log into the system as another user, or modify the file permissions of the relations and view the data. This type of illegal retrieval of the data in INGRES data bases must be eliminated. Since INGRES uses the UNIX file protection system, read and write permission is granted only to "ingres" who is the owner of all data base files. To legally manipulate or peruse the data base, the user must be the Data Base Administrator (DBA) or an authorized user given permission by the DBA. This means of protection, however, does not discourage those users who are determined to gain the information. Via programs written to access user's passwords, ill-gotten logins can be used. An example of such a program, called 'su' (which stands for super-user), taken from [Woo85] is the following shell routine that can reside in a directory where the su (normally found in /bin) command may be run:

```
stty -echo
echo "Password: \c"
read X
echo ""
stty echo
echo $1 $X | mail outside!creep &
sleep 1
echo Sorry
rm su
```

This program is called by a user entering su expecting to become the super-user of the system. This user is then prompted for a password, however, echo to the terminal has been turned off, so it does not appear on the screen. After

the password has been entered by the user, it is mailed to 'creep'; the program sleeps for 1 second and responds 'Sorry' to the user. The user just assumes that they have mistyped the password and try 'su' again. This 'su' program has since removed itself, however, and the user's next call to 'su' is to the correct system program. With programs such as the above 'su' command in existence, it is necessary to take all precautions with password safety, however, additional measures must be incorporated to eliminate data theft and corruption as much as possible.

2.3 Specific Requirements

The INGRES DBMS data manipulations are based on two kinds of commands, UNIX and QUERy Language (QUEL). The following is a list of the different commands and a brief description of what they do:

UNIX Commands

copydb - create batch files to copy out a data base and restore it.
creatdb - create a data base.
destroydb - destroy an existing data base.
equel - embedded QUEL interface to C (???).
help - get information about a data base.
ingres - INGRES relational data base management system.
printr - print relations.
purge - destroy all expired and temporary relations.
restore - recover from an INGRES or UNIX crash.
sysmod - modify system relations to predetermined storage structures.
usersetup - setup users file.

QUEL Commands

append - append tuples to a relation.
copy - copy data into/from a relation from/into a UNIX file.
create - create a new relation.
define - define a subschema.
delete - delete tuples from a relation.
destroy - destroy existing relation(s).
help - get information about how to use INGRES or about relations in the data base.
index - create a secondary index on an existing relation.
integrity - define integrity constraints.
modify - convert the storage structure of a relation.
permit - add permissions to a relation.

print - print relations.
range - declare a variable to range over a relation.
replace - replace values of domains in a relation.
retrieve - retrieve tuples from a relation.
save - save a relation until a specified date.
view - define a virtual relation.

These command interface directly with the data bases according to permissions stored in administration files kept on each data base.

2.3.1 *Front_End_Processor_to_INGRES_Related_UNIX_Commands*

The UNIX commands that manipulate INGRES data bases must further protect the data by validating authorized users via data base keys. This is mainly needed in the 'printr' command where permissions are necessary to access the data. All permission data should then be done on the relation level thus continuing to allow every INGRES user the ability to create data bases, purge their old relations, modify their contents or structures and destroy them. These INGRES related UNIX commands, however, must be run by a command processor that will be in a menu format. Based on the command chosen, the additional security measure should also occur.

2.3.2 *Front_End_Processor_to_INGRES_Commands*

The INGRES commands listed above are responsible for modifying or deleting existing relations and should include additional protection mechanisms to ensure that only legitimate, authorized persons are accessing the data bases. This front end processor will be incorporated in the UNIX front end processor within the INGRES command call. It must also be accessible by executing another command that would handle only INGRES QUEL commands. A mere validation of the user identification number is insufficient as password stealing is common. Any of the above commands for which permission is necessary must then first be validated for that user's current knowledge of the relation's key. If the accurate information cannot be given, access to that relation must be denied. When creating a relation, however, the user should be given the option to associate it with a key or not. If the protection of the data is critical, a key must be given; otherwise, the omission of the key may allow unauthorized users to gain access to the data. In this way, all operations such as append, destroy, delete, permit, print, retrieve, and performing operations on relations must be validated as to the user's current knowledge of the key to that relation before modification.

Chapter 3 - Detailed Design

3.1 Introduction

The INGRES DBMS includes a few UNIX commands that deal, for the most part, with the data bases as a whole and are more frequently used by either the DBA or the INGRES super-user, `ingres`. In addition, there are INGRES commands, initiated by first calling the UNIX command "`ingres`", that operate on specific data base relations and tuples within those relations. The design discussed in this chapter is for an implementation of an inquiring front end through which the INGRES UNIX commands and the INGRES QUEL commands must be executed. In the following sections each of the INGRES related UNIX commands will be described and their design will be addressed. In addition, the control flow of the implementation and some other considerations are discussed.

3.2 Interface to the User

The INGRES inquiring front end is a menu driven interface between the user and all INGRES related commands. By executing the command "`newingres`", the user is supplied with a list of options as to which command to execute. The format of that menu is as appears in Figure 3-1.

Below are the INGRES related UNIX commands available:

- 1) CHANGEKEY - change a relation's key.
- 2) COPYDB - create batch files to copy out a data base and restore it.
- 3) CREATDB - create a data base.
- 4) DESTROYDB - destroy an existing data base.
- 5) HELPR - get information about a data base.
- 6) INGRES - INGRES relational data base management system.
- 7) LISTREL - list relations a user has access to in a certain data base.
- 8) PRINTR - print relations.
- 9) PURGE - destroy expired and temporary relations.
- 10) RESTORE - recover from an INGRES or UNIX crash.
- 11) SYSMOD - modify system relations to predetermined storage structures.
- 12) EXIT - exit from this user's session.

Please Enter COMMAND NAME or NUMBER or '?' for Help:

Figure 3-1. Menu of INGRES Related UNIX Commands

The user can then access any of the commands by entering the correct command number and will then be prompted for the information needed for that command. The user may continue to enter command numbers and execute INGRES commands until all data base work is accomplished. This may include, for example, creating one data base, executing QUEL commands on yet another data base, and destroying yet a third data base. In this way, the user has flexibility to request one command and, when complete, continue with another. Exiting the front end processor is a valid command and must be entered when the INGRES work session is to be terminated. Below are descriptions and designs of all of the other valid commands that may be executed by the INGRES front end.

3.3 UNIX Commands

The INGRES related UNIX commands generally revolve around the maintenance of the databases, and, therefore, in the case of each of the commands described below, the user is first prompted for the data base name on which the user wishes to work. Following is a listing of each of the INGRES related UNIX commands accompanied by a brief description and explanation of design.

3.3.1 An added command, "changekey", is executable by INGRES users via the front end processor. "Changekey" allows the DBA to change the key of an encrypted relation or apply a key to a previously unencrypted relation. The user is prompted for the relation name and the old key, if applicable. The new key is requested and, after having been input, is requested again to verify the appropriate key. The relation is then encrypted using the new key, and the decrypted file is removed.

3.3.2 The "copydb" command will create batch files, copy.in and copy.out, for a particular data base, which will create a copy of a data base or restore it after destruction. If this command is chosen, the front end processor will prompt the user for the full path name of the directory where the two files are to be created. In addition, the user will be asked if this is to be done on specific relations or on all relations owned by the user. If any of the relations specified have been previously encrypted using the INGRES "create" command or the UNIX "changekey" command, the user will now be prompted for the key to that specific relation. The front end processor will then decrypt the relation using the supplied key and will continue with the next encrypted relation, if applicable. If the user supplies an incorrect key, they will be alerted by a standard error message that the particular relation could not be decrypted because the key was incorrect and that if access to that relation is still desired, the user must begin again. The user will also be prompted as to whether or not the -u option is to be specified when calling "copydb". This option allows the user to run "copydb" with a different user identification, and, although it allows successful creation of the copy files, it does not imply that the user can necessarily access the specified relations. The UNIX command "copydb" is then executed by the front end processor and following successful completion, the previously decrypted files will be encrypted to restore the relations to their original state.

3.3.3 The "creatdb" command allows an INGRES user to create a new data base or modify the status of an existing data base. In the first case, the person executing the command becomes the DBA of the newly created data base; in the second case, the user must be the DBA. Several options are also offered with this command to specify such things as concurrency control schemes or query modification. After choosing "creatdb" via the front end processor menu discussed above, another submenu is displayed to the user to specify the desired options as shown in Figure 3-2.

creatdb options

- 1) -uname = specify a different DBA called 'name'.
- 2) -e = modify options for an existing database.
- 3) -m = specifies that the UNIX directory in which the data base is to reside already exists.
- 4) +/-c = turns on (+) or off (-) the concurrency control scheme.
- 5) +/-q = turns on (+) or off (-) query modification.
- 6) EXIT = exit from this option session.

Please enter the number of the option desired:

Figure 3-2. Options for creatdb Command

The user is then able to specify options and the front end processor will formulate a command and execute it.

3.3.4 The "destroydb" command allows either the DBA or the INGRES super-user, ingres, to remove all references and all related files of an existing data base. After the user has input the data base name according to the front end processor's request, a menu is supplied, as shown in Figure 3-3, to allow the user to specify that the UNIX directory storing all of the data base files should not be removed. Following the option selection, the "destroydb" command will be executed and the data base removed.

destroydb options

- 1) -s = INGRES superuser must use this to execute destroydb.
- 2) -m = specifies that the UNIX directory in which the data base resides is not to be removed.
- 3) EXIT = exit from this option session.

Please enter the number of the option desired:

Figure 3-3. Options for destroydb Command

3.3.5 To give information about a specified relation(s) in a particular data base, the "helpr" command is used. After the user has input the data base name, the front end processor will prompt the user for relation names for which help is requested. In addition, prior to completing the "helpr" command, the user may specify an option based on the menu in Figure 3-4.

helpr options

- 1) -uname = specify a different DBA called 'name'.
- 2) +/-w = wait/do not wait for the data base.
- 3) EXIT = exit from this option session.

Please enter the number of the option desired:

Figure 3-4. Options for helpr Command

3.3.6 If the command "ingres" is requested, the user will be prompted for the data base name and the relations on which work is to be done. The user will no longer have direct access to the "ingres" command which will enforce an added layer of security. Instead, the only access to INGRES commands will be through the front end processor, "newingres", and specifying the command "ingres" from the menu or by executing compiled EQUDEL programs to be discussed below. The validation checks done by this front end processor will monitor and prevent the unauthorized use of privileged data bases. After the relation names on which the user requests work to be done have been gathered, a list is formed to determine the following:

- a. that the user has been given permission by the DBA to view the specified data base relations. This information is stored in an INGRES administration relation kept in the data base's directory.
- b. if any of the requested relations are encrypted. When a relation is created, the user will have the opportunity, prior to exiting the command processor, of having that relation encrypted to provide further security control. A

list of those encrypted relations within the data base will be maintained by the command processor in a file called ".crypt" located in each data base directory. This file will only be accessible by this command.

An error message will be output if a user requests a relation that does not exist. However, if the relation does exist, the user has access permission, and it is encrypted, the user will then be prompted for the key for decrypting the data base relation. The encrypted relation is stored in the data base directory in a file by the same name as the relation name which is how unencrypted data base relations are currently maintained by INGRES. The encrypted relation will then be decrypted into a temporary file and copied into the relation name file in the data base directory in order that all INGRES commands will operate successfully. After the relations have been decrypted, the front end processor provides an option menu to the user in the form of Figure 3-5.

ingres options

- 1) +/-U = enable/disable direct update of the system relations and secondary indices.
- 2) -uname = specify a different DBA called 'name'.
- 3) -cN = set the minimum field width for printing character domains to N.
- 4) -ilN = set integer output field width to N.
- 5) -flxM.N = set floating point output field width to M characters with N decimal places.
- 6) -vX = set the column separator for retrieves to the terminal and print commands to be X.
- 7) -rM = set modify mode on the retrieve command to M.
- 8) -nM = set modify mode on the index command to M.
- 9) +/-a = set/clear the autoclear option in the terminal mode.
- 10) +/-b = set/reset batch update.
- 11) +/-d = print/do not print the dayfile.
- 12) +/-s = print/do not print any of the monitor messages, including prompts.
- 13) +/-w = wait/do not wait for the database.
- 14) EXIT = exit from this option session.

Please enter the number of the option desired:

Figure 3-5. Options for ingres Command

The "ingres" command is then executed by the front end processor passing the data base name and any other appropriate parameters. When the "ingres" command has completed (the user has entered "\q" or "<cntl d>" signifying termination of INGRES and QUEL data base manipulations), the command processor will finish any duties necessary. If relations had been decrypted prior to calling "ingres", they will now be encrypted using the same key. The decrypted file will be unlinked, so that access to the legible file is impossible. If, however, a decrypted file of a previously known relation does not exist, it likely means that a "delete" (to delete a relation) command was executed. In this case, the front end processor will verify in the "admin" (administration) relation, located in each data base directory, that the relation does not exist. Also a check is made against all of the relations in the data base versus those that existed prior to the "ingres" call to

determine if any new data bases were created. If "create" was executed to create a new relation, the command processor will ask the user if this new relation should be encrypted and, if so, what is the key. After the user has input the key, they are again prompted for it to ensure that typing mistakes were not made. The relation will then be encrypted and the original file storing the relation will be unlinked.

3.3.7 The "listrel" command enables an INGRES user to determine to which relations in a given data base access has been given. In addition, it will be indicated on the output whether or not the listed relation has been encrypted by the DBA determining that any user must also know the key in order to execute any command on that relation. An example of output from this command is shown in Figure 3-6.

```
      User Name:  sabrack
      Data Base Name:  employee

Relation Name      Relation Name
managers           first_shift
second_shift      third_shift
overtime*         payroll*
family_info*      previous_exp
dept_to_mgrs      equipment
```

* indicates that the relation is encrypted.

Figure 3-6. List of User's Relations

3.3.8 The "printr" command prints specified relations out of the particular data base. Therefore, after the data base name has been retrieved, the front end processor requests the relation(s) which are to be printed. In addition, the user will be prompted for the keys if the relation has been encrypted and the relation will be decrypted based on these keys. Flags are also accepted with this command and are shown to the user as in Figure 3-7. The command is then executed and the user is returned to the front end command menu.

printr options

- 1) -uname = specify a different DBA called 'name'.
- 2) -cN = set the minimum field width for printing
- 3) -ilN = set integer output field width
to N.
- 4) -flxM.N = set floating point output field
width to M characters with N
decimal places.
- 5) -vX = set the column separator for retrieves to
the terminal and print commands to be X.
- 6) +/-w = wait/do not wait for the database.
- 7) EXIT = exit from this option session.

Please enter the number of the command desired:

Figure 3-7. Options for printr Commands

3.3.9 "Purge" allows the DBA or the INGRES super-user to purge all expired and temporary relations. In the case of this command, a user may or may not specify a data base name. If one is not given, all of the data bases for which the user is the DBA or, if the user is ingres and the -s option is chosen, all data bases will be purged. Therefore, the front end processor will request that a particular data base name or 'all' be specified. The user will then be shown the flag menu as listed in Figure 3-8 and the appropriate command will be executed. NOTE: If the -f flag is used, the .crypt files, which store the names of the encrypted relations in the data base, may be removed.

purge options

- 1) -p = expired user relations are deleted.
- 2) -f = causes unrecognizable files to be deleted.
- 3) -a = causes messages to be printed about the pending operation and execute it only if the response is a 'y'.
- 4) -s = INGRES superuser must use this to execute purge.
- 5) +/-w = wait/do not wait for the database.
- 6) EXIT = exit from this option session.

Please enter the number of the option desired:

Figure 3-8. Options for purge Command

3.3.10 The "restore" command enables either the DBA for a data base or the INGRES super-user to recover a data base after an INGRES or UNIX crash. Similar to the "purge" command above, either a particular data base is specified or all data bases for which the user is the DBA or, if the -s option is specified in the menu below by ingres, all data bases will be restored. Once again the options appear to the user as in Figure 3-9 and, upon completion of the user requests, the command is executed to restore the requested data base(s). NOTE: If the restore command executes with no errors, the purge command is executed. If the -f flag is used, the .crypt files which store the names of the encrypted relations in the data base may be removed.

restore options

- 1) -p = if restore completes with no errors, purge is called and expired user relations are deleted.
- 2) -f = if restore completes with no errors, purge is called and unrecognizable files will be deleted.
- 3) -a = causes messages to be printed about the pending operation and execute it only if the response is a 'y'.
- 4) -s = INGRES superuser must use this to execute restore.
- 5) +/-w = wait/do not wait for the database.
- 6) EXIT = exit from this option session.

Please enter the number of the option desired:

Figure 3-9. Options for restore Command

3.3.11 In order for a DBA to modify its data base's system relations or for the super-user, ingres, to modify any data base's system relations, the command "sysmod" must be executed. Figures 3-10 and 3-11 specify the screen output to the user to specify options and which system relations to modify to gain maximum access performance when running INGRES.

sysmod options

- 1) -s = INGRES superuser must use this to execute sysmod.
- 2) +/-w = wait/do not wait for the database.
- 3) EXIT = exit from this option session.

Please enter the number of the option desired:

Figure 3-10. Options for sysmod Command

sysmod relations

- 1) relation
- 2) attribute
- 3) indexes
- 4) tree
- 5) protect
- 6) integrities
- 7) EXIT = exit from this option session.

Please enter the number of the relation desired:

Figure 3-11. Relations for sysmod Command

3.4 Error and Interrupt Handling

An error handling routine will be called in the case of system errors or non-recoverable errors by the user. This will normally terminate the "newingres" command. Sometimes, however, it may only cause the current command to be terminated and further execution of the INGRES commands may continue. Interrupts such as a hang up of the terminal, a delete of the command or a power fail may cause momentary or complete disruption of the command. If the command is terminated, however, an interrupt handling routine will be called first to clean up all residue of previous calls. This would include the encrypting of any previously decrypted relations and the removal of those decrypted relation's files. In this way, an interrupted INGRES session is not susceptible to data theft or corruption by unauthorized users. Specific error messages and their meanings include the following:

- a. **ERROR: Wrong Key Given!**
The key that was input was incorrect. If access to the relation is still desired, it will be necessary to begin the command again. This may be output during any of the commands "copydb", "ingres", or "printr".
- b. **ERROR: Invalid Data Base Name!**
A data base name exceeding 14 characters was entered. The user will then be prompted for a valid data base

name.

- c. ERROR: Data Base Does Not Exist!
The data base requested by the user does not exist.
- d. ERROR: Relation Does Not Exist!
The relation requested by the user via the "ingres" command does not exist in this data base.
- e. ERROR: System Call Failed!
A system call such as a create, open, read or write of a file has failed. This may occur if too many files were open at a time or if there was a system overload and the command could not be executed.
- f. ERROR: Key will not be added/changed!
The user failed to accurately input the new key for a relation, given two chances to do so. Therefore, at this time, the key will not be added or modified for this relation.

3.5 Control Flow

The general control flow in this design travels from the main command processor of the INGRES front end to the particular command chosen and then back to the main command processor as shown in Figure 3-12. A more specific look at the detailed design results in flow which appears much like Figure 3-13. Although the menu acting as the command processor is still shown as both the start and the end of a particular cycle of a command, the functions between are more detailed. The data base name must first be entered and, depending on the command, relations must then be gathered, keys must be requested, and relations are decrypted. Finally, in all cases, except listing of a users relations, a particular command menu is generated to allow the user to choose options specific to that command and then the execution of that command by the front end processor follows.

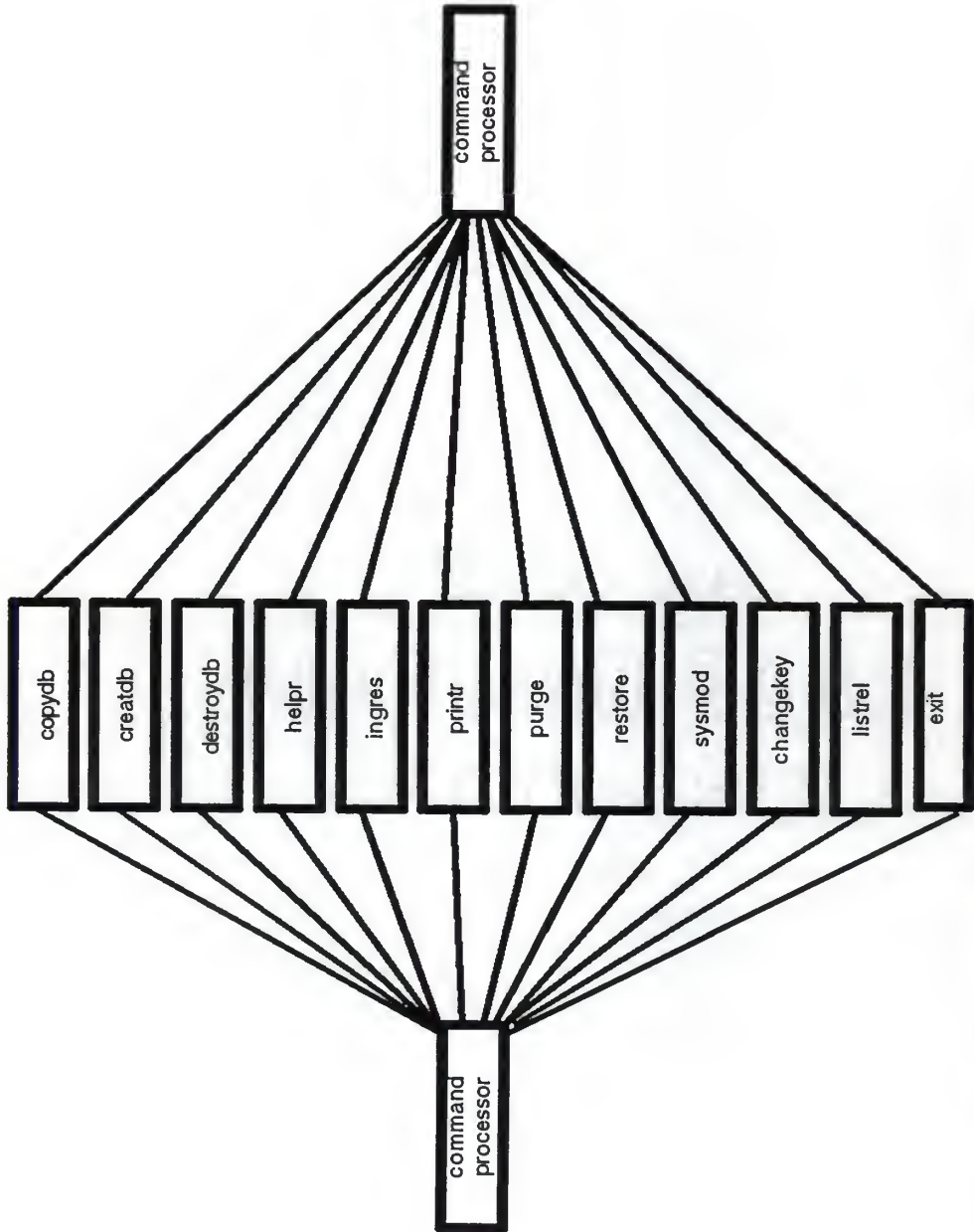


Figure 3 – 12 High Level Design of INGRES related UNIX Commands

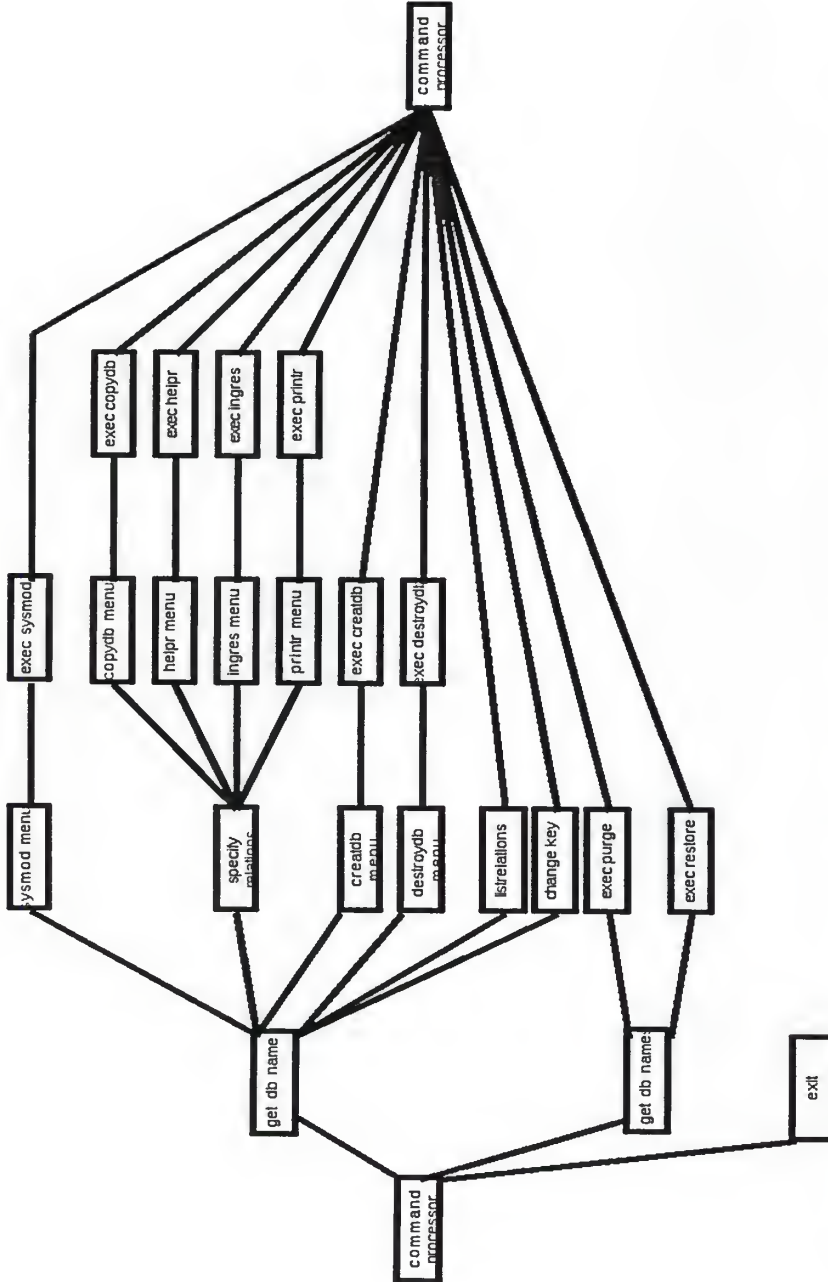


Figure 3 - 13 Expanded High Level Design of INGRES related UNIX Commands

3.6 Commands Excluded

Some considerations exist that prohibit the inclusion of two INGRES related UNIX commands in this front end processor.

3.6.1 The UNIX command "usersetup" reads and reformats the /etc/passwd file to create a new file which becomes the INGRES users file kept in /usr/ingres/files/users. This command is executed only once to initially create the users file, therefore, it was felt that it's inclusion in this command processor would be unnecessary. It is hoped that at some time the "usersetup" command will be capable of also updating or modifying the users file with new or changed users as listed in the password file, but until that time, it will be omitted from this processor.

3.6.2 The "equel" command provides a user with a way of interfacing the C programming language with INGRES. It is comprised of the EQUQL pre-compiler and the EQUQL runtime library. Because it's product is actually executable C code with Embedded QUEL statements, it cannot feasibly be included in this INGRES front end processor. The compiled code may be executed outside the realms of this INGRES front end, and, therefore, the decryption of the used relations is impossible prior to the call to the EQUQL program. The encryption mechanism utilized in this implementation does, however, guarantee that use of protected and encrypted relations will not be successfully included in the EQUQL programs as access to the relations cannot properly be accomplished.

Chapter 4 - Implementation

4.1 Introduction

The INGRES front end processor deals with the various UNIX commands that manipulate the data bases and their relations. This chapter deals with the implementation of that processor and describes the routines and global variables involved. The actual C code for the implementation is included in Appendix A.

4.2 Variables

Several global variables are needed to implement the INGRES front end processor as many routines are involved in either manipulating them or reading them and processing other data based on their values. The following is a list of those variables and a brief explanation of their use.

1. cryptfile - A character array that holds the full path name of the ".crypt" file in the data base's directory.
2. relname - A character array that holds the full path name of the relation that is currently being worked on.
3. dbname - A character array holding the full path name of the current data base being worked on.
4. database - The name of the data base being worked on. This is used to construct file paths for the ".crypt" file and the temporary file created to store the encrypted or decrypted data base prior to moving it to its destination file.
5. files - An array of character pointers pointing to the list of relations that are encrypted and that the user has requested work on.

6. keys - An array of character pointers pointing to the list of encryption keys that the user has requested work on. These are the actual user keys that have been encrypted and are stored in the ".crypt" file in their encrypted form.
7. errno - An integer used to determine the reason for an error in system calls.
8. newkey - A pointer to a character string that holds the new key for the relation currently being worked on.
9. cfd - The file descriptor of the ".crypt" file in the current working data base.
10. placenum - An integer used as a place holder in the ".crypt" file to determine which relation within the file is currently being worked on.
11. crkey - A structure composed of a character array that is long enough to store the key resulting from the UNIX "makekey" function. The output of the call is placed in a file that is then read based on this structure.
12. cr - A structure composed of two arrays of character pointers. One array points to the names of the relations that have been encrypted in the data base and the other array points to the encrypted keys of each of those relations. This is the structure used in building and reading the ".crypt" file in each data base directory.

4.3 Routines

The commands discussed in Chapter 3 require several routines to handle things such as prompting the user for the data base name and handling system errors or interrupts. Figures 4-1, 4-2, and 4-3 exhibit the flow of control within this implementation. The INGRES related UNIX commands that will

entail only prompting the user for information and building a command line are highlighted in Figure 4-1. Most of these commands deal with the data base as a whole. The circle named "actual operations on data bases and relations" is expanded in Figure 4-2. It displays the interaction between the routine necessary to handle those INGRES commands that work with the relations of a data base. The bubble titled "routines for INGRES data base commands" includes those highlighted in the previous Figure. Figure 4-3 displays the user interface and file access necessary in this implementation. The prompts deal mainly with relation names and keys and the files are used to manipulate those keys in encrypting and decrypting those relations. Below are brief explanations that further define all of the routines that appear in these Figures.

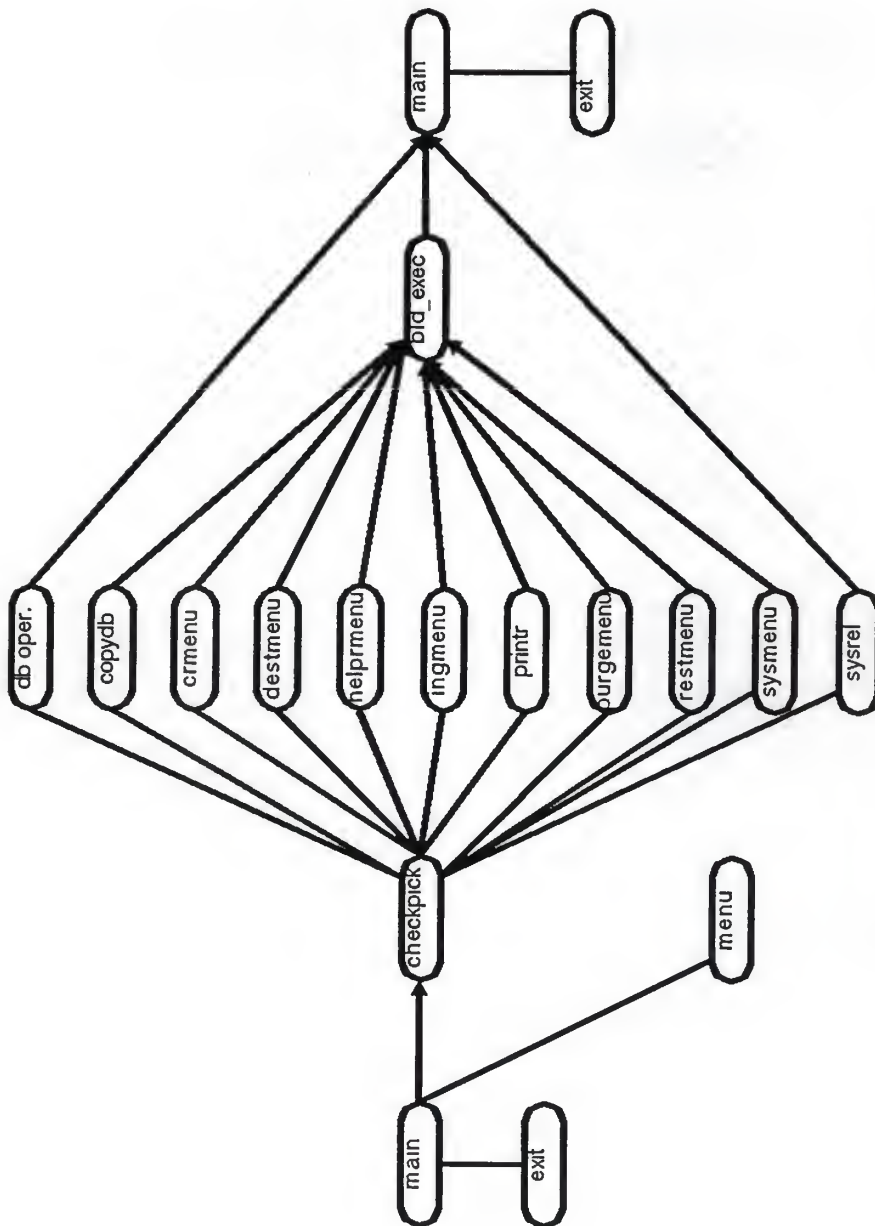


Figure 4 - 1 Control Flow for Data Base Commands

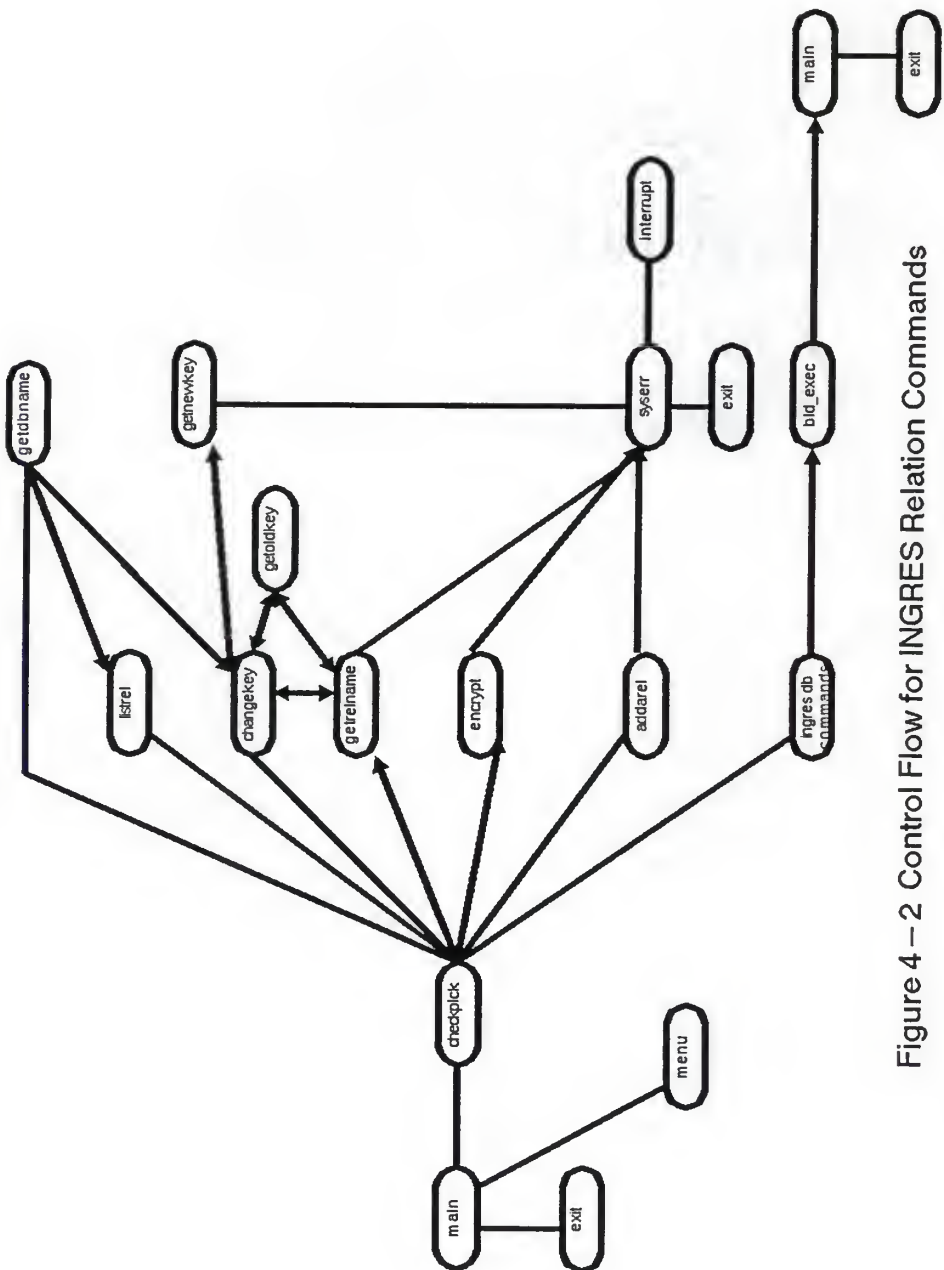


Figure 4 - 2 Control Flow for INGRES Relation Commands

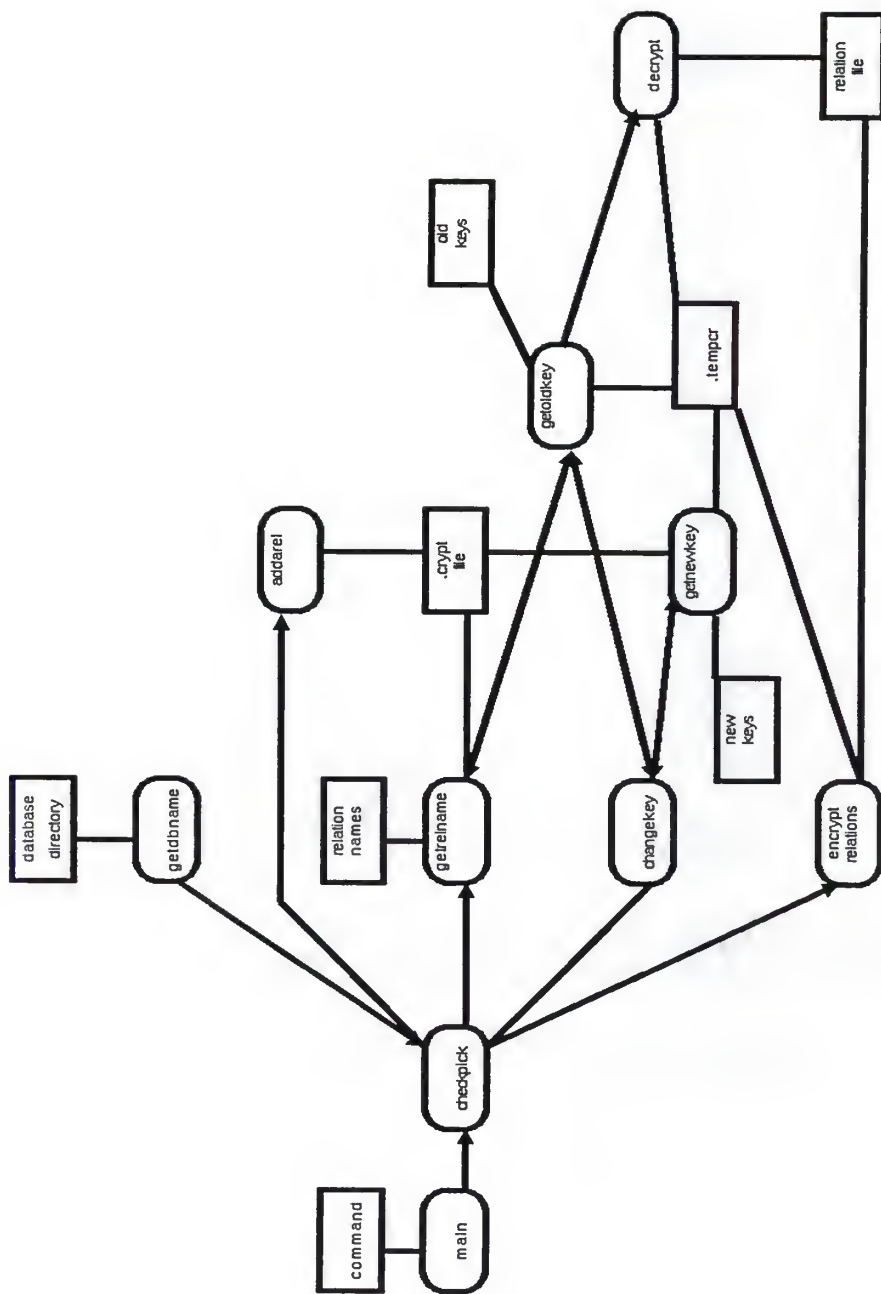


Figure 4 - 3 User Interface and File Access

4.3.1 The checking for the user's command request is accomplished in the main() routine. The input can either be the number associated with the command in the help menu, or it can be the name of the command itself. This allows flexibility in parsing user input.

4.3.2 The menu() routine prints the help menu to the user with a listing of valid commands and a brief description of the function of the command. It completes by asking the user to input the command name or number selected.

4.3.3 After the valid command has been input, the checkpick() routine will do the high level processing of the command. The major responsibility of this routine is to call other routines to accomplish the set up necessary to execute any of the commands.

4.3.4 The getdbname() routine is responsible for prompting the user for a data base name on which to work. The syntax for a valid data base name is checked by ensuring only names of 14 or fewer characters are input. In addition, a check is made to validate that it is indeed an existing data base. Otherwise, an error message, "ERROR: Data Base Does Not Exist!" will be output and the user is asked to input another data base name. Input of three consecutive invalid data base names will result in prompting for another command.

4.3.5 The user must be prompted for one or more relations in the majority of the commands. Therefore, the getrelname() routine will accept as a parameter an "s" or "m" for a single relation or multiple relations. Getrelname() prompts the user for a relation name and, based on another parameter ("e" for an existing relation or "n" for a new relation), it will do one of the following:

1. verify that the relation exists; scan the ".crypt" file (lists of all of the encrypted relations in that data base) in the data base's directory and determine if the given relation appears there. If it is listed, return to the calling routine that this relation is encrypted.
2. verify that the relation does not exist; place this relation name on a "newrelation" list to be used after the "ingres" call to verify all new

relations and ask the user if a key should be applied.

3. return an error if the relation does not exist or if a system call failed.
4. return to the calling program since the user has completed the list of relations to be worked on.

4.3.6 The `getoldkey()` routine is called when an encrypted relation is to be decrypted and the user must first be prompted for the appropriate key. Echo will be turned off on the user's terminal in order that the response to the prompting for the key cannot be viewed. This key will then be used to create an encryption key to be stored in the ".crypt" file in the data base's directory with the relation's name. This implementation is similar to the way in which passwords are maintained in the UNIX password file. The user inputs a password (or key); it is then encrypted using the "makekey" UNIX function (which generates encryption keys) and is stored in its encrypted form. The output of the "makekey" command is redirected to a file and then read and stored as that relation's key. When the user's response to "What is your old key?" is received, it is sent to the makekey command; an encryption key is generated and is compared to that encryption key stored in the ".crypt" file. Therefore, the keys themselves are not stored in any file in raw form, but rather their encrypted form is saved and used for comparisons. In this way proper control of the keys is maintained. This routine will output an error that the wrong key was given if it is incorrect and the user will be prompted for another command on which to work.

4.3.7 The user is asked, within the `getnewkey()` routine, to enter the new key for encrypting a relation. It is called when a new relation is created or when a DBA decides to add or change the encryption key for an existing relation. Echo will be turned off on the user's terminal in order that the response to the prompting for the key cannot be viewed. The user will be asked to input the key twice to ensure that typing mistakes do not occur. If the key is not entered identically twice in a row, the user is given another chance to input it twice. After the second try, this routine will respond that, at this time, the key will not be added or changed. Within this routine the `encrypt_relations()` routine is called to encrypt the given relation with the newly input key. After the encryption takes place, the new

key is input to the UNIX "makekey" command and the result is stored in the ".crypt" file to be used for comparison purposes the next time that relation is accessed.

4.3.8 Given the key retrieved in the above routine, the encrypt_relations() routine will actually encrypt the given relation. It will use the "crypt" UNIX command to encrypt the relation and place the newly encrypted file in a temporary file (called .tempcr in the data base directory) and then move it back into it's original file in the data base directory which is named the same as the relation name.

4.3.9 The decrypt() routine does the opposite of the above routine. It is called by getoldkey() once it is determined from the user the appropriate key for this relation. It then will decrypt the relation stored in the data base directory under the relation name and place it in a temporary file. It will then move that temporary file back into the file which is the same as the relation name.

4.3.10 The build_exec() routine will formulate the command that is to be executed and will then do so. Upon successful completion, it will return to the main menu for another command to be worked on.

4.3.11 A routine called syserr() is responsible for producing a message when a system call has failed. It will print out a number associated with a particular place in the code for debugging purposes.

4.3.12 The interrupt_handler() will handle the cases when an interrupt is received and clean up work is necessary. The relations that have been decrypted must be encrypted before the program terminates and this routine is responsible for handling this.

4.3.13 The user must be able to view the list of relations for which he has access permission and the listrel() routine performs this function. In addition, it will flag those relations that are encrypted in the output.

4.3.14 The addarel() routine will add a relation to the encrypted relation file (".crypt") in the data base's directory. This is necessary either when a DBA decides to add a key to an existing relation via the "changekey" command or if a new relation is created via the "ingres" command.

4.3.15 Each of the Figures noted in Chapter 3 is generated by routines to accept the response of the user based on the desired option or command. Certain help commands are also available to give the user more information about the command.

4.4 C Programs

The above description of implementation was programmed using the C Programming Language and the UNIX Operating System. The actual modules comprising the implementation can be found in Appendix A.

1983.

[Row82] Rowe, L.A., "Ingres Relational Database Management System", Mini/Micro 82 Conference Record, Published by IEEE, September, 1982.

[Sie85] Siegal, P., Woodfill, J., Ranstrom, J., Meyer, M., and Allman, E., "Ingres Version 7 Reference Manual", Nov. 18, 1985.

[Sto76] Stonebraker, M., Wong, E. and Kreps, P., "The Design and Implementation of Ingres", Trans. Database Systems, Vol. 2, No. 3, Sept. 1976.

[Woo85] Wood, Patrick H. and Kochan, Stephen G., *UNIX System Security*, 1985.

Chapter 5 - Conclusion

5.1 Summary

The subject of computer security, in light of today's wide use of computers, and the vast amount of data stored on them, has become critical. This paper has taken a brief look at the UNIX operating system environment and certain security strengths and weaknesses it possesses. Items such as password aging were suggested as ways to enhance security on an existing system. The INGRES DBMS was also discussed and, based on the interaction between the two, an inquiring INGRES front end was designed and implemented. It enables users to encrypt data base relations that may contain very sensitive data. This may have particular importance at an institution like Kansas State University for student data, personnel records and grades. By utilizing this menu driven system, an INGRES user can access INGRES commands and data bases and strengthen the security that exists. A user is asked to input a key in order to access a particular relation of a data base if it was encrypted on creation. This is a decision of the DBA based on the value of the data that the relation contains. The keys are maintained on the system similar to the way in which the passwords for the users are. After the user inputs a key, the key is encrypted and stored in its encrypted form. The next time access to that relation is necessary, the key is again prompted for, encrypted and compared to the stored value. In addition to added security for the data base information, the INGRES front end processor creates a more user friendly environment. Menus are printed that display parameters sent to the various routines and give a brief explanation of their use. This allows new users to feel more comfortable with using the INGRES DBMS.

5.2 Problems

During the design and development of the implementation of the inquiring INGRES front end, some problems arose. In order to serve as a front end with no modification to INGRES code, it was necessary to demand relation names prior to the actual execution of the INGRES commands. If work is to be done on a series of protected and encrypted relations, this could be time consuming and an irritant to the user. Likely the best implementation of a more secure INGRES DBMS would have been to include such implementation within the existing

INGRES code. It was decided in this implementation, however, not to do so. Also, extra overhead results from system calls necessary to encrypt and decrypt both the relations and keys. This will not be a significant problem here at Kansas State University, however, based on the only moderate use of INGRES. The inclusion of the EQUQL command in the list of possible commands executed by the front end processor was prohibited due to the availability of executing EQUQL programs outside of this front end environment. QUQL statements are embedded within a C program and are compiled and executable by the user at any time. This implementation, however, does prohibit users from accessing encrypted relations via EQUQL programs as the relations are not readable.

5.3 Further Extensions

Based on the problem discovered with the EQUQL programming environment, it would be beneficial to amend this inquiring front end and include support for EQUQL programs. This may be possible by forcing EQUQL programs to include certain routines which would be responsible for handling all encryption/decryption algorithms. Without the user interface, however, this problem becomes more significant. Another possible enhancement to the INGRES DBMS security system is the possibility of hashing or encrypting entire relations and/or data bases and storing them under different names so that files could not maliciously be tampered with. This may impede the less serious threats, although total penetration of security is always possible.

BIBLIOGRAPHY

- [All81] Allman, Eric, "How to Set up Ingres", 1981.
- [Col86] Columbus Dispatch, "Pulling the Plug on Hackers", April 6, 1986.
- [Dat85] "Datakey-Net-Lock Access Security System", Datakey, Inc., January, 1985.
- [Dep79] Department of Defense, "The DoD Computer Security Initiative Program", Proceedings of the Seminar on the DoD Computer Security Initiative Program, July, 1979.
- [Dow79] Downs, Deborah and Popek, Gerald J., "Data Base Management Systems Security and Ingres", International Conference on Very Large Data Bases, Published by IEEE, October, 1979.
- [Fis85] Fisher, K. W., "Memory Card Conference - Battelle Columbus Laboratories - April 10 and 11, 1985", April, 1985.
- [Gra86] Gray, Patricia Bellew, "A Software-Lock Breaker Becomes A Hero to Some, a Villain to Others", Wall Street Journal, February 7, 1986.
- [Gre85] Greenberg, David A. and Woods, Stephen C., "Establishing a Medical Research Computer Facility", Computer Biology Medicine, 1985.
- [Hel75] Held, G.H., "Ingres -- A Relational Data Base System", American Federation of Information Processing Societies Conference Proceedings, May, 1975.
- [Hig83] Highland, Harold Joseph, "Impact of Microcomputers on Total Computer Security", Computer Security, June, 1983.
- [Kat83] Katzin, Emanuel, "Problem of Security", Technical Communications, Forth Quarter, 1983.
- [Nat85] Nation's Business, "Heading Off Crime Losses", page 12, February, 1985.
- [Pet85] Peterson, I., "Federal Computer Security Concerns", Science News, p. 230, October 12, 1985.
- [Rit83] Ritchie, Dennis M., "On the Security of UNIX", UNIX Programmer's Manual, Section 2, AT&T Bell Laboratories,

Appendix A

Application Code

```

#include <stdio.h>
#include <sys/types.h>
#include <sys/signal.h>
#include <sys/file.h>
#include "stat.h"
#include <errno.h>
#include "defines.h"

#define quote "\""

char *cmd[] = /* List of Available Valid Commands*/
{
    "ch", "co", "cr", "de", "he", "in",
    "li", "pr", "pu", "re", "sy", "ex", 0,
};

char cryptfile[PATHLENGTH]; /* Holds the full poth name of the */
/* .crypt file. */
char relname[PATHLENGTH]; /* Holds the full path name of the */
/* relation name. */
char filename[PATHLENGTH]; /* Holds the poth nome of the */
/* temporary file for encrypting. */
char namebuf[PATHLENGTH]; /* Holds the full path of the */
/* relation name. */
char name[PATHLENGTH]; /* Holds the name of the relation. */
char finalname[MAXREL][PATHLENGTH]; /* Holds the finol name of the */
/* relation. */
char junkls[PATHLENGTH]; /* Holds the ls command of the */
/* relation name. */
char lsit[PATHLENGTH]; /* Stores the ls commond of the */
/* relation. */
char dbname[PATHLENGTH]; /* Holds the full poth name of the */
/* data base being worked on. */
char database[DBLENGTH]; /* Holds the doto base name. */
char files[MAXREL][DBLENGTH]; /* Holds the relations thot the */
/* user wants to work on that are */
/* encrypted. */
char makekeys[MAXREL][KEYLENGTH]; /* Holds the result of 'makekey' */
/* being executed on the keys of */
/* the encrypted relations that */
/* the user wants to work on. */
char userkeys[MAXREL][KEYLENGTH]; /* Holds the user inputted keys */
/* for the relations that the user */
/* wants to work on. */
int place[MAXREL]; /* Holds the placenum of the */
/* associated files and keys. */
int totalenc; /* The totol number of encrypted */
/* relations that the user is */
/* working on. */
int size; /* The size of the relation name. */
int i, namefd;
char relations[MAXREL][DBLENGTH];
char rellist[PATHLENGTH]; /* Holds the list of relations on */
/* which the helpr command will work. */
char workrel[DBLENGTH]; /* Working relation name. */
char pathname[PATHLENGTH]; /* Holds the path name of the source */
/* or destination of the copydb call. */
char newkey[CRYPTKEYLEN]; /* Holds the relation's new key. */
char exingres[COMMANDLENGTH]; /* Holds the ingres command call. */
char excreatdb[COMMANDLENGTH]; /* Holds the creatdb command coll. */
char exdest[COMMANDLENGTH]; /* Holds the destroydb command call. */
char exhelp[COMMANDLENGTH]; /* Holds the help command call. */
char excopy[COMMANDLENGTH]; /* Holds the copydb command call. */
char exprintr[COMMANDLENGTH]; /* Holds the printr command call. */
char expurge[COMMANDLENGTH]; /* Holds the purge command call. */
char exrestore[COMMANDLENGTH]; /* Holds the restore command call. */
char exsysmod[COMMANDLENGTH]; /* Holds the sysmod command call. */
char option[COMMANDLENGTH]; /* Holds the options for the command */

```

```

char option1[COMMANDELENGTH]; /* Holds the options for the command */
/* calls. */
char cbuf[100]; /* Character buffer for gets routine.*/
int placenum; /* the place number for the relation */
int errno; /* Global variable to store the error */
/* number generated by the system */
/* calls. */

struct cr_relkey; /* Structure for storing used */
/* relations and their keys. */

main()
{
    int register differ, a, i;

    setsig(); /* Set up the signal handling */

    /* Determine the choice made by the user and complete some */
    /* preliminary set-up work if necessary. */

    while (TRUE)
    {
        for (i=0; i<MAXREL; i++)
        {
            makekeys[i][0] = '\0';
            place[i] = 0;
            files[i][0] = '\0';
            finalname[i][0] = '\0';
            userkeys[i][0] = '\0';
            relations[i][0] = '\0';
        }
        totalenc = 0;

        printf("\nPlease Enter COMMAND NAME or NUMBER or '?' for Help: ");
        gets(cbuf); /* Get the user's response */

        /* If the user's response is a number, convert it and process it */
        if ((cbuf[0] >= '1') && (cbuf[0] <= '9'))
        {
            a = atoi(cbuf);
            checkpick(a);
            cbuf[0] = '\0';
            continue;
        }

        /* If the user's response is an exit, get out of here */
        if ((cbuf[0] == 'e') && (cbuf[1] == 'x'))
        {
            printf("\nFinished with this INGRES/UNIX Session -");
            printf("\nGoodbye!\n\n");
            exit(0);
        }

        /* If the user's response is 'questionable', show the menu. */
        if ((cbuf[0] == '?') || (cbuf[0] == '\n'))
        {
            menu();
            cbuf[0] = '\0';
            continue;
        }

        /* Determine which command was requested and then call */
        /* checkpick to process it. */
        for (a = 0; a < NCMD5; a++)
        {
            differ = strncmp(cbuf,cmd[a],2);
            if (differ == 0)

```

```

    }
    o++;
    checkpick(o);
    cbuf[0] = '\0';
    break;
}
}
if (differ != 0)
{
    printf("\nThis is not a legitimate response.\n");
    printf("Please Try Again!\n");
}
}
}

/*****
 *
 *   CHECKPICK
 *
 *****/
*
* Function: This routine will determine what command was chosen
* and perform the appropriate action. Some commands
* require that a relation that has been encrypted be
* decrypted prior to manipulation or viewing of that
* relation. This routine will call another routine
* to do that prompting for the appropriate relation
* name and key.
*
* Input: None.
*
* Output: Calls to other subroutines.
*
* Returns: SUCCESS or FAILURE.
*/

checkpick(pick)
int pick;
{
    int i;
    int ret;

    switch (pick)
    {
    case 1: /* CHANGEKEY */
        printf("\nCHANGEKEY ---\n");
        /* call routine to modify the data base key */
        if (chongekey() == FAILURE)
        {
            printf("\nChongekey Foiled!\n");
            return (FAILURE);
        }
        break;

    case 2: /* COPYDB */
        printf("\nCOPYDB ---\n");
        if (getdbname('o') == FAILURE)
            return (FAILURE);

        /* call routine to get the relation names to work on */
        /* This routine also will decrypt the files */
        ret = getrelname('m','b','o');
        switch (ret)
        {
            case MATCH: /* If a MATCH or SUCCESS, then */
            case SUCCESS: /* run copydb to copy the data */

```

```

        copydbcall();          /* base's relations.          */
    case FAILURE:
        totalenc--;           /* Subtract one from the total */
                             /* number encrypted because the */
                             /* last one failed.             */
        for(l=0; l<totalenc; l++)
        {
            placenum = place[l];
            strncpy (newkey, userkeys[placenum], DBLENGTH);
            sprintf(reiname,"%s%s%s",dbname,"/",files[placenum]);
            sprintf(filename,"%s%s%s",dbname,"/",files[placenum]);
            sprintf(junk1s,"%s%s%s",dbname,"/",".lshald");

            encrypt_relations();
        }
        if (ret == FAILURE)
            return (FAILURE);
        break;
    case NOMATCH:
    default:
        copydbcall();
        break;
}
break;

case 3:          /* CREATDB */
    printf("\nCREATDB —\n");
    if (getdbname('n') == FAILURE)
        return (FAILURE);

    /* fork and exec /usr/ingres/bin/creatdb */
    if (crccall() == FAILURE)
        return (FAILURE);
    break;

case 4:          /* DESTROYDB */
    printf("\nDESTROYDB —\n");
    if (getdbname('a') == FAILURE)
        return (FAILURE);

    /* fork and exec /usr/ingres/bin/destroydb */
    if (destcall() == FAILURE)
        return (FAILURE);
    break;

case 5:          /* HELPR */
    printf("\nHELPR —\n");
    if (getdbname('a') == FAILURE)
        return (FAILURE);

    /* call routine to get the relation names to work on */
    /* This routine also will decrypt the files          */
    if (getreiname('m','b','l') == FAILURE)
        return (FAILURE);

    /* fork and exec helpr */
    if (helprcall() == FAILURE)
        return (FAILURE);
    break;

case 6:          /* INGRES */
    printf("\nINGRES —\n");

```

```

    if (getdbname('a') == FAILURE)
        return (FAILURE);

    /* call routine to get the relation names to work on */
    /* This routine also will decrypt the files */
    if (getrelname('m','b','a') == FAILURE)
        return (FAILURE);

    /* fork and exec ingres */
    if (ingrescall() == FAILURE)
        return (FAILURE);

    for(l=0; l<totalenc; l++)
    {
        placenum = place[l];
        strncpy (newkey, userkeys[placenum], DBLENGTH);
        sprintf(relname,"%s%s%s",dbname,"/",files[placenum]);
        sprintf(filename,"%s%s%s",dbname,"/",files[placenum]);
        sprintf(junkls,"%s%s%s",dbname,"/",".lshold");

        /* call routine to encrypt those relations that had */
        /* been decrypted before the ingres call. */
        encrypt_relations();
    }

    break;

case 7:          /* LISTRELATIONS */
    printf("\nLISTRELATIONS --\n");

    /* call routine to list the relations to which */
    /* he has access to in the specified data base. */
    listrel();
    break;

case 8:          /* PRINTR */
    printf("\nPRINTR --\n");
    /* Call routine to get the data base name on which */
    /* to work. */
    if (getdbname('a') == FAILURE)
        return (FAILURE);

    /* call routine to get the relation names to be printed */
    /* This routine also will decrypt the files */
    if (getrelname('m','b','a') == FAILURE)
        return (FAILURE);
    /* fork and exec printr */
    if (printrcall() == FAILURE)
        return (FAILURE);
    break;

case 9:          /* PURGE */
    printf("\nPURGE --\n");
    /* Call routine to get the data base name on which */
    /* to work. */
    if (getdbname('a') == FAILURE)
        return (FAILURE);

    /* call routine to get the relation names to be purged */
    /* This routine also will decrypt the files */
    if (getrelname('m','b','a') == FAILURE)
        return (FAILURE);
    /* fork and exec purge */
    if (purgecall() == FAILURE)
        return (FAILURE);
    break;

```

```

case 10: /* RESTORE */
    printf("\nRESTORE —\n");
    /* Call routine to get the data base name on which */
    /* to work. */
    if (getdbname('o') == FAILURE)
        return (FAILURE);

    /* fork and exec restore */
    if (restcall() == FAILURE)
        return (FAILURE);
    break;

case 11: /* SYSMOD */
    printf("\nSYSMOD —\n");
    /* Call routine to get the data base name on which */
    /* to work. */
    if (getdbname('o') == FAILURE)
        return (FAILURE);
    /* fork and exec sysmod */
    if (syscall() == FAILURE)
        return (FAILURE);
    break;

case 12: /* EXIT */
    printf("\nFinished with this INGRES/UNIX Session —");
    printf("\nGoodbye!\n\n");
    exit(0);
    break;

default: /* ERROR */
    printf("\nThis is not a legitimate response.\n");
    printf("Please Try Again!\n");
    break;
}
pick = 0;
return(SUCCESS);
}

/*****
 *
 * GETRELNAME
 *
 *****/
 *
 * Function: This routine will prompt the user for one or more
 * relation names based on a parameter stating 's' =
 * single or 'm' = multiple. It will then verify that
 * the relation exists, see if it is encrypted, and decrypt
 * it if necessary. If the relation does not exist, it
 * means that it is to be created and should be put on a
 * "newrelation" list.
 *
 * Input: num = s (single) or m (multiple).
 * state = e (existing) or n (new) or b (both).
 * what = a (array of relations) or l (list in a character
 * array.
 *
 * Output: Calls to other subroutines.
 *
 * Returns: Global variables files[], makekeys[], userkeys[] and places[].
```



```

*          SUCCESS or FAILURE.
*
*/

getrelnome(num,state,what)
char num;
char state;
char what;
{
    struct stat *buf, buffer;
    int m, numreq, fd, cfd, number, i, j, encryptnum;
    lang nbytes;

    sprintf(rellist,"%s", " ");

    nbytes = 65;
    numreq = 0;
    encryptnum = 0;
    placenum = 0;
    buf = (&buffer);

    /* The first thing that must be done is to find out how
    /* many relations we are looking for--> 's' = one and 'm' =
    /* multiple.
    /*
    if (num == 's')
    {
        number = 1;
        printf("\nPlease enter the name of the RELATION on which you");
        printf(" would like\n to work: ");
    }
    else
    {
        number = MAXREL;
        printf("\nPlease enter the RELATIONS on which you would like ");
        printf("to work\n (listed one at a time).\n To complete list, ");
        printf("simply enter o 'q'.\n");
    }
    for (i=0; i<number; i++)
    {
        /* Now form a list of all the relations that the user
        /* wants to work on. The user can create, modify or print
        /* relations, so they must at this time give all relations
        /* on which they want to work during this session.
        /*
        /* The maximum number of relations that can be worked on
        /* at one time is MAXREL (?).
        /*
        scanf("%s", relations[numreq]);
        getch(); /* Dummy Getch to clean out the buffer */

        if (*relations[numreq] == 'q')
            break;

        sprintf (relname,"%s%s%s",dbname,"/",relations[numreq]);
        sprintf (filename,"%s%s%s",dbname,"/",relations[numreq]);
        sprintf(junk1s,"%s%s%s",dbname,"/",".lshld");

        sprintf(lsit,"%s %s %s %s", "ls", filename, ".", ">", junk1s);

        if (system(lsit) == FAILURE)
        {
            syserr(34);
            return(FAILURE);
        }
        if ((namefd = (open (junk1s, O_RDWR, 660))) == FAILURE)
        {
            syserr(35);
            return(FAILURE);
        }
        size = strlen(dbname) + 15; /* The 15 is for the '/' and the
        /* relation name.
        /*

```

```

if (read(namefd,namebuf,size) <= 0)
{
    close(namefd);
    return(NOTEXIST);
}
close(namefd);

for (i=0; i<14; i++)
    name[i] = namebuf[i + strlen(dbname) + 1];
strcat(finalname[numreq],quote);
strcat(finalname[numreq],namebuf);
strcat(finalname[numreq],quote);

sprintf (rellist, "%s %s",rellist,relations[numreq]);

if (stat(namebuf, buf) != 0)
{
    if (num == 's')
        return (NOTEXIST);
    else
    {
        printf("ERROR: Relation %s Does Not Exist!\n",
            relations[numreq]);
        continue;
    }
}

sprintf (workrel,"%s",relations[numreq]);

numreq++;      /* Bump this only if it exists */
}              /* End of the number for */

if (what == 'l')
return (SUCCESS);

if (stat(cryptfile, buf) == 0) /* Does .crypt exist? */
{
    /* If so, open it */
    if ((cfd = open(cryptfile,O_RDWR,660)) == FAILURE)
    {
        syserr(1);
        return(FAILURE);
    }
    if (lseek(cfd,0L,0) == FAILURE)
    {
        syserr(2);
        close(cfd);
        return (FAILURE);
    }
}
placenum = 0;
while (nbytes > 0)
{
    if ((nbytes = read(cfd,&relkey,sizeof relkey)) > 0)
    {
        /* Put the names of all of the encrypted files */
        /* stored in .crypt into "files" and all of the */
        /* encrypted keys into "makekeys". */

        strncpy (files[placenum], relkey.relation, DBLENGTH);
        strncpy (makekeys[placenum], relkey.key, KEYLENGTH);

        placenum++;
    }
    else if (nbytes == -1)
    {
        syserr(3);
        return(FAILURE);
    }
}

```

```

}
close(cfd);

j = placenum;

for(m=0; m<numreq; m++)
{
    for(placenum=0; placenum<j; placenum++)
    {
        if(strcmp(relations[m],files[placenum]) == 0)
        {
            strncpy (workrel,files[placenum],DBLENGTH);
            place[encryptnum++] = placenum;
            totalenc++;
            if (num == 's') /* must be from changekey */
                return (MATCH);
            else /* must be from an ingres command */
            {
                printf("\nRelation %s:",files[placenum]);
                if (getoldkey() == FAILURE)
                {
                    printf("Cannot allow access to %s\n",
                        files[placenum]);
                    return (FAILURE);
                }
                continue;
            }
        }
        /* The end of the strcmp for matched relation names */
    }
    /* The end of the placenum for */
}
/* The end of the numreq FOR for string comparing */

if (num == 's')
    return (NOMATCH);
else
    return (SUCCESS);

}
/* The end of the stat check of .crypt */
else /* stat return != 0 which means that there are */
    /* no encrypted relations in this data base. */
{
    if(numreq == 0)
    {
        if (stat(finalname[i], buf) != 0)
            return (NOTEXIST);
        else
            return (NOTFOUND);
    }
    for(i=0; i < numreq; i++)
    {
        if (stat(finalname[i], buf) != 0)
            printf("ERROR: Relation %s Does Not Exist!\n",
                relations[numreq]);
    }
    return (NOTFOUND);
}
}

```

```

/.....
*
*   GETNEWKEY   *
*
*.....

```

```

*
* Function: This routine will prompt the user for the new key
*           for a relation when either creating a new relation
*           via the ingres command or changing the key of a
*           relation via the changekey command.
*
* Input: None.
*
* Output: Calls to other subroutines.
*
* Returns: Global variable newkey.
*          SUCCESS or FAILURE.
*
*/

getnewkey()
{
char *verifykey[CRYPTKEYLEN]; /* user's second input of the key */
char *makekey[100]; /* string to hold makekey command */
char testkey[CRYPTKEYLEN]; /* string to hold output of */
                          /* makekey command. */

struct stat *newbuf, nbuffer;
struct crkey enkey;
struct cr jnker;
int ret, whence, kfd, fd, tries, match;
long offset;

tries = 0;
match = FALSE;
newbuf = (&nbuffer);

while ((match == FALSE) && (tries < 2))
{
if (system("stty -echo") == FAILURE)
{
syserr(36);
return(FAILURE);
}

printf("\nWhat is your new key? ");

gets(newkey);
printf("\nPlease reenter your new key: ");
gets(verifykey);

if (system("stty echo") == FAILURE)
{
syserr(37);
return(FAILURE);
}

printf("\n\tProcessing...\n");

if (strcmp(newkey,verifykey) != 0)
{
tries++;
if (tries == 2)
printf("Mismatch! Unable to Apply New Key!\n\n");
else
printf("Mismatch! Please Try Again!!\n\n");
continue;
}

match = TRUE;
strncpy (userkeys[placenum], newkey, KEYLENGTH);
}
/* The user was given two tries to input a new key twice */
/* and was unsuccessful, therefore, terminate this */
/* command by returning FAILURE. */
if (tries == 2)

```

```

    return(FAILURE);
}
if (strlen(newkey) == 0)
{
    /* Put the key in the .crypt file */
    if ((ret = stat(cryptfile, newbuf)) != 0)
    {
        if ((kfd = creat(cryptfile,477)) < 0)
        {
            syserr(4);
            return(FAILURE);
        }
        close(kfd);
    }

    if ((kfd = open(cryptfile,O_RDWR,660)) == FAILURE)
    {
        syserr(6);
        return(FAILURE);
    }
    crypt.relation[0] = '\0';
    crypt.key[0] = '\0';
    crypt.busybit = 0;

    if (lseek(kfd,(long)(placenum * sizeof crypt),0) == FAILURE)
    {
        unlockbox();
        close(kfd);
        syserr(7);
        return(FAILURE);
    }

    if(write(kfd,&crypt,sizeof crypt) < 0)
        return (FAILURE);
    close(kfd);

    printf("\tCompleted -- NO key will be used!\n");

    return(SUCCESS);
} /* End of strlen(newkey) == 0 if statement */

/* Encrypt the key for ultimate storage in the .crypt file. */
sprintf(makey,"%s%s%s | %s > %s","echo ",newkey,"|s","/usr/lib/makekey",KEY);

if (system(makey) == FAILURE)
{
    recover();
    syserr(8);
    return (FAILURE);
}

if ((fd = open(KEY,O_RDWR,660)) < 0)
{
    recover();
    syserr(37);
    return(FAILURE);
}

if (lseek(fd,0L,0) == -1)
{
    recover();
    syserr(38);
    return(FAILURE);
}

if (read(fd,&enkey,sizeof enkey) < 0)
{
    recover();
}

```

```
    unlackbox();
    close(fd);
    syserr(9);
    return(FAILURE);
}
close(fd);

sprintf(testkey, "%c%c%c%c%c%c%c%c%c%c", enkey.key[2], enkey.key[3],
enkey.key[4], enkey.key[5], enkey.key[6], enkey.key[7], enkey.key[8],
enkey.key[9], enkey.key[10], enkey.key[11], enkey.key[12]);

/* Put the key in the .crypt file */
if ((ret = stat(cryptfile, newbuf)) != 0)
{
    if ((kfd = creat(cryptfile, 477)) < 0)
    {
        syserr(10);
        return(FAILURE);
    }
    close(kfd);
}

if ((kfd = open(cryptfile, O_RDWR, 660)) == FAILURE)
{
    syserr(11);
    return(FAILURE);
}

strncpy (crypt.relation, workrel, DBLENGTH);
strncpy (crypt.key, testkey, strlen(testkey));

if (lseek(kfd, (long)(placenum * sizeof crypt), 0) == FAILURE)
{
    unlackbox();
    close(kfd);
    syserr(12);
    return(FAILURE);
}

if(write(kfd, &crypt, sizeof crypt) < 0)
    return (FAILURE);
close(kfd);

if (encrypt_relations() == FAILURE)
    return(FAILURE);
else
    return (SUCCESS);
}

/*****
 *
 * ENCRYPT_RELATIONS*
 *
 * *****
 *
 * Function: This routine will encrypt the relations again after
 *           the user has finished the desired manipulations to
 *           them.
 *
 * Input: None.
 *
 * Output: Calls to other subroutines.
 *
 * Returns: SUCCESS or FAILURE.
 *
 */
```

```

encrypt_relations()
{
char cryptit[PATHLENGTH+PATHLENGTH]; /* Stores the crypt command */
char movit[PATHLENGTH+PATHLENGTH+PATHLENGTH]; /* Stores the mv command */
char tempfile[PATHLENGTH]; /* Stores the name of the temporary file */

int index, efd, readret;
struct cr ecrypt;
struct stat *ebuf, ebuffer;

    ebuf = (&ebuffer);

    sprintf(tempfile,"%s.%s",relname,"tempcr");

    if (lockbox() == FAILURE)
        return (FAILURE);

    /* If so, open it and read it. */
    if ((efd = open(cryptfile,O_RDWR,660)) == FAILURE)
    {
        unlockbox();
        close(efd);
        return(FAILURE);
    }
    if (lseek(efd,(long)(placenum * sizeof ecrypt),0) == -1)
    {
        unlockbox();
        close(efd);
        syserr(15);
        return(FAILURE);
    }
    if ((readret = read(efd,&ecrypt,sizeof ecrypt)) < 0)
    {
        unlockbox();
        close(efd);
        syserr(16);
        return(FAILURE);
    }

    index = place[placenum];

    sprintf(cryptit,"%s %s < %s > %s","/usr/bin/crypt",newkey,
        finalname[index], tempfile);

    if (system(cryptit) == FAILURE)
    {
        recover();
        unlockbox();
        close(efd);
        syserr(17);
        return(FAILURE);
    }

    /* Change it to be = FREE */
    ecrypt.busybit = FREE;

    if (lseek(efd,(long)(plocenum * sizeof ecrypt),0) == FAILURE)
    {
        unlockbox();
        close(efd);
        syserr(18);
        return(FAILURE);
    }
    if (write(efd,&ecrypt,sizeof ecrypt) <= 0)
    {
        close(efd);
        unlockbox();
        syserr(19);
        return(FAILURE);
    }
}

```

```

    }
    close(efd);
    unlockbox();

    sprintf(movit, "%s %s %s", "/bin/mv", tempfile, finolnome[index]);

    if (system(movit) == FAILURE)
    {
        recover();
        syserr(20);
        return(FAILURE);
    }
    return (SUCCESS);
}

```

```

/*****
 *
 *   CHANGEKEY   *
 *
 *****/

```

```

* Function: This routine will enable the user to change the
*           key for a relation given that the current correct
*           key is first input.
*
* Input: None.
*
* Output: Calls to other subroutines.
*
* Returns: SUCCESS or FAILURE.
*
*/

```

```

chongekey()
{
    printf("\nWith this command you may change the key of a relation");
    printf(" in a database\nfor which you are the DBA.\n");
    if (getdbname('o') == FAILURE)
        return(FAILURE);

    switch (getrelname('s', 'e', 'o'))
    {
        case NOTFOUND:
        case NOMATCH:
            printf("\nThat relation is not encrypted. Therefore, you");
            printf(" must want to encrypt\nthis relation.\n");
            if (getnewkey() == FAILURE)
                return(FAILURE);
            break;

        case NOTEXIST:
            printf("That relation does not exist!\n");
            return(FAILURE);

        case MATCH:
            if (getoldkey() == FAILURE)
                return(FAILURE);
            if (getnewkey() == FAILURE)
                return(FAILURE);
            break;

        default:
            break;
    }

    return(SUCCESS);
}

```


}

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

```
*  
* Function: This routine will output an error that a system  
*           call failed. It will print out the number associated  
*           with a particular place in the code for debugging  
*           purposes.  
*  
* Input: Error number.  
*  
* Output: Calls to other subroutines.  
*  
* Returns: SUCCESS or FAILURE.  
*  
*/
```

```
syserr(errarnum)  
int errarnum;  
{  
    printf("SYSTEM ERROR number %d has occurred!\n", errarnum);  
    printf("Ensure that all of the encrypted relations on which\n");  
    printf("\tyou were working are still encrypted!\n");  
    /* First, ensure that echo is turned back on! */  
    system("stty echo");  
}
```

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

```
*  
* Function: This routine will enable the user to find out what  
*           relations he has access to in a given data base.  
*  
* Input: None.  
*  
* Output: Printout of user's relations.  
*           Calls to other subroutines.  
*  
* Returns: SUCCESS or FAILURE.  
*  
*/
```

```
listrel()  
{  
    printf("\nThis command will list those relations for which you\n");  
    printf("have permissions in a specified data base\n");  
    if (getdbname('a') == FAILURE)  
        return (FAILURE);  
  
    /**** Check relations in this database and verify permissions ***/  
    /**** before listing relations allowed to this user. *****/  
}
```

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

```

*          *
*   GETDBNAME   *
*          *
*.....*
*
* Function:  This routine is used to prompt the user for a data
*            base name on which to work.  In addition, a check
*            is made to validate that it is indeed an existing
*            data base.
*
* Input:  None.
*
* Output:  A global variable storing the data base name.
*
* Returns:  FAILURE if data base does not exist after three
*           tries.
*
*/

getdbname(state)
char state;
{
    struct stat *dbuf, dbbuffer;
    int gotdb, tries;
    dbbuf = (&dbbuffer);
    tries = 0;
    gotdb = FALSE;

    while ((gotdb == FALSE) && (tries < 3))
    {

        /* The first thing that must be done is to find out which */
        /* data base the user wants to work on. */
        printf("\nPlease enter the name of the DATA BASE on which");
        printf(" you would like\nto wark:  ");
        scanf("%s", database);
        getchar();

        if (strlen(database) > 14)
        {
            printf("\nERROR:  Invalid Data Base Name!\n\n");
            continue;
        }

        sprintf (dbname,"%s%s",DBPATH,database);

        if (state == 'n')
            return(SUCCESS);

        if (stat(dbname, dbbuf) != 0)
        {
            printf("\nERROR:  Data Base Does Not Exist!\n\n");
            tries++;
            continue;
        }
        gotdb = TRUE;
    }

    /* Next check to see if there are any encrypted relations */
    /* in the data base that the user is to be working on. */
    /* The file /usr/ingres/data/bases/<database_name>/crypt */
    /* keeps a listing of those encrypted relations. */
    /* If there are no encrypted relations, then we know that */
    /* no encryption is necessary and we can return from this */
    /* this routine to checkpick. */
    sprintf (cryptfile,"%s%s%s",DBPATH,database,"/crypt");

    /* The user was given three tries to input a valid data */
    /* base name and was unsuccessful, therefore, terminate */

```

```

    /* this command by returning FAILURE. */
    if (tries == 3)
        return(FAILURE);
    else
        return (SUCCESS);
}

/*****
 *
 *   GETOLDKEY
 *
 *****/
*
* Function: This routine will get the old key from the user and
*           then it will call decrypt() to decrypt the given relation
*           with the given key.
*
* Input: None.
*
* Output: A global variable storing the data base name.
*
* Returns: FAILURE if data base does not exist after three
*          tries.
*/

getoldkey()
{
    char *makit[100]; /* Stores the makekey command */
    char *testkey[CRYPTKEYLEN];
    char oldkey[KEYLENGTH]; /* Holds the relation's old key */
    int i, fd;
    struct crkey junkit;

    if (system("stty -echo") == FAILURE)
    {
        syserr(39);
        return(FAILURE);
    }
    printf("\nThat relation is encrypted.\nWhat is the key? ");
    gets(oldkey);
    if (system("stty echo") == FAILURE)
    {
        syserr(40);
        return(FAILURE);
    }
    printf("\nProcessing...\n");

    strncpy (userkeys[plocenum], oldkey, KEYLENGTH);
    /* Encrypt the user's key and store it in .crypt eventually */
    sprintf(makit,"%s%s | %s > %s","echo ",oldkey,"ls","/usr/lib/makekey",KEY);

    if (system(makit) == FAILURE)
    {
        syserr(21);
        return(FAILURE);
    }

    if ((fd = open(KEY,O_RDONLY,660)) < 0)
    {
        syserr(22);
        return(FAILURE);
    }

    if (lseek(fd,0L,0) == FAILURE)

```

```

    {
        syserr(23);
        return(FAILURE);
    }

    if (read(fd,&junkit,sizeof junkit) < 0)
    {
        syserr(40);
        return (FAILURE);
    }

    sprintf(testkey,"%c%c%c%c%c%c%c%c%c%c",junkit.key[2],junkit.key[3],
        junkit.key[4],junkit.key[5],junkit.key[6],junkit.key[7],junkit.key[8],
        junkit.key[9],junkit.key[10],junkit.key[11],junkit.key[12]);

    /* Verify that the existing key ond the user's inputted key */
    /* ore identical. */
    if(strcmp(testkey,makekeys[placenum]) != 0)
    {
        printf("ERROR: Wrong Key Given\n");
        return(FAILURE);
    }

    sprintf(relname,"%s%s%s",dbnome,"/",files[placenum]);

    if (decrypt(oldkey) == FAILURE)
        return (FAILURE);

    return (SUCCESS);
}

```

```

/*****
 *          *
 *   DECRYPT   *
 *          *
 *****/

```

```

* Function: This routine will decrypt the given relation with
*           the global key.
*
* Input: None.
*
* Output: A decrypted relation.
*
* Returns: SUCCESS or FAILURE.
*
*/

```

```

decrypt(oldkey)
    chor oldkey[CRYPTKEYLEN];
    {
        /* Need to include lockbox and unlockbox around the crypt */
        /* command as I did in encrypt_relations. */
        chor decryptit[PATHLENGTH+PATHLENGTH]; /* Stores the decrypt command */
        chor movit[PATHLENGTH+PATHLENGTH]; /* Stores the mv command */
        chor tempfile[PATHLENGTH]; /* Stores the name of the */
        /* temporary file */
        int index, dfd;
        struct cr decrypt;

        sprintf(tempfile,"%s.%s",relnome,"tempcr");
        if (lockbox() == FAILURE)
            return(FAILURE);

        /* Open it ond read it. */
        if ((dfd = open(cryptfile,O_RDWR,660)) == FAILURE)
        {
            unlockbox();

```

```

close(dfd);
syserr(24);
return(FAILURE);
}
if (lseek(dfd,(long)(plocenum * sizeof decrypt),0) == FAILURE)
{
unlockbox();
close(dfd);
syserr(25);
return(FAILURE);
}
if (read(dfd,&decrypt,sizeof decrypt) <= 0)
{
unlockbox();
close(dfd);
syserr(26);
return(FAILURE);
}
if (decrypt.busybit == BUSY)
{
printf("\nThis Relation is Currently Being Used!\n");
printf("Please Try Again Later.\n\n");
unlockbox();
close(dfd);
return(FAILURE);
}
else /* Must be == FREE */
decrypt.busybit = BUSY;

index = ploc[plocenum];

sprintf(decryptit,"%s %s < %s > %s","/usr/bin/crypt",oldkey,
        finolnome[index],tempfile);

if (system(decryptit) == FAILURE)
{
unlockbox();
close(dfd);
syserr(28);
return(FAILURE);
}
if (lseek(dfd,(long)(plocenum * sizeof decrypt),0) == FAILURE)
{
unlockbox();
close(dfd);
syserr(29);
return(FAILURE);
}
if (write(dfd,&decrypt,sizeof decrypt) <= 0)
{
close(dfd);
unlockbox();
syserr(30);
return(FAILURE);
}
close(dfd);

unlockbox();

sprintf(movit,"%s %s %s","/bin/mv",tempfile,finolnome[index]);

if (system(movit) == FAILURE)
{
syserr(31);
return(FAILURE);
}
}

```

```

}

```

```
/*.....
 *
 *   SETSIG   *
 *
 *.....
 *
 * Function:  This routine will set up the signal handling.
 *
 * Input:    None.
 *
 * Output:   None.
 *
 * Returns:  None.
 */
setSIG()
{
#ifdef REAL_THING /*ZZZZZZZZZZZ REMOVE THIS IFDEF */
    signal(SIGTERM, SIG_IGN);
    signal(SIGHUP, SIG_IGN);
    signal(SIGQUIT, SIG_IGN);
    signal(SIGINT, SIG_IGN);
#endif
}

/*.....
 *
 *   RECOVER  *
 *
 *.....
 *
 * Function:  This routine will recaver from a grave error in the
 *           middle af processing.
 *
 * Input:    Nane.
 *
 * Output:   Nane.
 *
 * Returns:  Nane.
 */
recover()
{
    int kfd;
    struct stat *newbuf, nbuffer;

    newbuf = (&nbuffer);

    /* Put the key in the .crypt file */
    if (stat(cryptfile, newbuf) != 0)
        return (SUCCESS);

    if ((kfd = open(cryptfile,O_RDWR,660)) == FAILURE)
    {
        syserr(32);
        return(FAILURE);
    }
    crypt.relation[0] = '\0';
    crypt.key[0] = '\0';
    crypt.busybit = 0;

    if (lseek(kfd,(lang)(placenum * sizeof crypt),0) == FAILURE)

```

```
    {
      unlockbox();
      close(kfd);
      syserr(33);
    }

    if(write(kfd,&crypt,sizeof crypt) < 0)
      return (FAILURE);
    close(kfd);

    printf("\tDue to System ERROR — NO key will be used!\n");

    return(SUCCESS);
}
```

```

#define TRUE          1
#define FALSE        0
#define MATCH        2 /* Two compared strings are equal */
#define NOMATCH      3 /* Two compared strings are not */
                        /* equal. */

#define FAILURE      -1
#define SUCCESS     0
#define DONE        1
#define CONTINUE    2
#define NOTFOUND    -2 /* Relation name was not listed */
                        /* in the .crypt file. */
#define NOTEXIST    -3 /* The relation does not exist. */
#define KEY         "/usr/ingres/data/base"
#define DBPATH      "/usr/ingres/data/base"

#define MAXOPTS     14 /* Maximum number of options that */
                        /* are allowed in one call. */
#define MAXREL      10 /* Maximum number of relations to be */
                        /* worked on at a time. This may be */
                        /* changed. */
#define KEYLENGTH   15 /* Allowable length of a key. */
#define DBLENGTH    15 /* Maximum length of a data base name */
                        /* or a relation name. */
#define CRYPTKEYLEN 14 /* The length of an encrypted key */
#define PATHLENGTH  50 /* Maximum length of the path for */
                        /* the data base files. */
#define NCMDS       12 /* Number of available UNIX INGRES */
                        /* Commands. */
#define BUSY        1 /* This indicates in the "busybit" */
                        /* integer that someone has got this */
                        /* relation and is working on it. */
#define FREE        0 /* This indicates in the "busybit" */
                        /* integer that no one is working */
                        /* on this relation. */

#define LEFTARROW   "<"
#define RTARROW    ">"
#define COMMANDLENGTH 65 /* The Maximum length of one of the */
                        /* executed commands. */
#define OPTIONLENGTH 30 /* The Maximum length of the combined */
                        /* options to a command. */

struct crkey /* Structure for the .tempcr files */
{
    char key[CRYPTKEYLEN];
};

struct cr /* Structure for the .crypt files */
{
    char relation[DBLENGTH];
    char key[CRYPTKEYLEN];
    int busybit;
};

struct cr crypt;

```



```
atoi(s)
char s[];
{
    int i, n;

    n = 0;
    for (i=0; s[i] >= '0' && s[i] <= '9'; ++i)
        n = 10 * n + s[i] - '0';
    return(n);
}
```

```

#include "defines.h"

extern char optian[COMMANDLENGTH];
extern char excopy[COMMANDLENGTH];
extern char datobase[DBLENGTH];
extern char rellist[PATHLENGTH];
extern char pothnome[PATHLENGTH];

/*****
 *
 *   COPYDBCALL
 *
 *****/
/*
 * Function: This routine is used to formulote the cammand copydb.
 *
 * Input: None.
 *
 * Output: A screen of options.
 *
 * Returns: SUCCESS or FAILURE.
 */

copydbcoll()
{
    char aptbuf[100][3];
    int j, i, o, differ;

    sprintf(option,"%s", " ");
#ifdef DEBUG
    printf("excopy = %s\n",excopy);
    printf("aption = %s\n",aption);
    printf("copydbcoll database = %s\n",dotabase);
#endif

    copydbmenu();

    for (i=0; ;i++)
    {
        sconf("%s", optbuf[i]);

        /* If the user's response is 'questionable', show the menu again */
        if ((*optbuf[i] == '?') || (*optbuf[i] == '\n'))
        {
            copydbmenu();
            continue;
        }

        /* If the user's response is an exit, get out of here */
        if ((*optbuf[i] == '2') || (*optbuf[i] == 'q'))
        {
            getch();
#ifdef DEBUG
            printf("inside the ex part and i = %d!\n",i);
#endif
            for (j=0; j<=i; j++)
            {
                if (copydbcheck(aptbuf[j]) == DONE)
                {
#ifdef DEBUG1
                    printf("Got to just after copydbcheck and returning\n");
#endif
                    break;
                }
            }
            sprintf(pathname,"%s", " ");
            printf("\nWhat is the full path name of the directory");
            printf(" where you wish\nto copy the files? ");
        }
    }
}

```

```
        gets(pothnome);
        sprintf(excopy, "%s%s %s %s %s", "copydb", option, dotobase, pothnome, rellist);
#ifdef DEBUG
        printf("excopy before exec = %s\n", excopy);
#endif
        printf("\n%s\n", excopy);

        if (system(excopy) == FAILURE)
        {
            syserr(101);
        }
        return(SUCCESS);
    }
    /* If the user's response is o number, convert it and process it */
    if ((*optbuf[i] == '1') || (*optbuf[i] == '2'))
    {
#ifdef DEBUG
        printf("1-9 optbuf [%d] = %c and o = %d\n", i, optbuf[i][0], o);
#endif
        continue;
    }
    printf("\nOption %s is not o legitimote option.\n", optbuf[i]);
}
}
```

```
#include "defines.h"
extern char option[COMMANDLENGTH];

/*****
 *
 *   COPYDBCHECK
 *
 *****/
* Function: This routine is used to check and prompt the user for
*           more information based on which options the user has
*           requested for the copydb command.
*
* Input: None.
*
* Output: A list of options.
*
* Returns: The options to be executed.
*/

copydbcheck(pick)
char pick[3];
{
    char buf[5];
    int o, m;

    o = atoi(pick);

#ifdef DEBUG
    printf("o = %d and pick = %s\n", o, pick);
#endif
    switch(o)
    {
        case 1:
            printf("What is the user's login nome? ");
            gets(buf);
            sprintf(option, "%s %s%s", option, "-u", buf);
#ifdef DEBUG
            printf("in case 1\n");
            printf("option = %s\n", option);
#endif
            break;

        case 2:
#ifdef DEBUG
            printf("in case 14\n");
#endif
            return(DONE);
            break;

        default:
            break;
    }
    for (m=0; m<5; m++)
        buf[m] = 0;

    return (CONTINUE);
}
```



```

#include "defines.h"

extern char option[COMMANDLENGTH];
extern char excreatdb[COMMANDLENGTH];
extern char database[DBLENGTH];

/*****
 *
 *      CRCALL      *
 *
 *****/
* Function: This routine is used to formulate the command creatdb.
*
* Input: None.
*
* Output: A screen of options.
*
* Returns: The options to be executed.
*/

crCALL()
{
    char optbuf[100][3];
    int j, i, a, differ;

    sprintf(option,"%s"," ");
#ifdef DEBUG2
    printf("excreatdb = %s\n",excreatdb);
    printf("option = %s\n",option);
    printf("crCALL database = %s\n",database);
#endif

    crmenu();

    for (i=0; ;i++)
    {
        scanf("%s", optbuf[i]);
        getchar();

        /* If the user's response is 'questionable', show the menu again */
        if ((*optbuf[i] == '?') || (*optbuf[i] == '\n'))
        {
            crmenu();
            continue;
        }

        /* If the user's response is an exit, get out of here */
        if ((*optbuf[i] == '6') || (*optbuf[i] == 'q'))
        {
#ifdef DEBUG
            printf("inside the crex part and i = %d!\n",i);
#endif
            for (j=0; j<=i; j++)
            {
                if (crcheck(optbuf[j]) == DONE)
                {
#ifdef DEBUG
                    printf("Got to just after crcheck and returning\n");
#endif
                    break;
                }
            }
            sprintf(excreatdb,"%s%s %s","creatdb",option,database);
#ifdef DEBUG
            printf("excreatdb before exec = %s\n",excreatdb);
#endif
            printf("\n%s\n",excreatdb);
        }
    }
}

```

```
        if (system(excreatdb) == FAILURE)
            syserr(101);
        return(SUCCESS);
    }
    /* If the user's response is a number, convert it and process it */
    if ((*optbuf[i] >= '1') && (*optbuf[i] <= '6'))
    {
#ifdef DEBUG
        printf("1-9 optbuf [%d] = %c and o = %d\n", i, optbuf[i][0], o);
#endif
        continue;
    }
    printf("\nOption %s is not a legitimate option.\n", optbuf[i]);
}
}
```

```
#include "defines.h"
extern char option[COMMANDELENGTH];

/*****
 *
 *      CRCHECK      *
 *
 *****/
*
* Function: This routine is used to check and prompt the user for
*           more information based on which options the user has
*           requested for the creatdb command.
*
* Input: None.
*
* Output: A list of options.
*
* Returns: The options to be executed.
*
*/

crcheck(pick)
char pick[3];
{
    char buf[5];
    int a;

    a = atoi(pick);

#ifdef DEBUG2
    printf("o = %d and pick = %s\n",a,pick);
#endif
    switch(o)
    {
        case 1:
            printf("Whot is the user's login name? ");
            gets(buf);
            sprintf(option,"%s %s%s",option,"-u",buf);
#ifdef DEBUG2
            printf("in cose 1\n");
            printf("option = %s\n",option);
#endif
            break;

        case 2:
            sprintf(option,"%s %s",option,"-e");
#ifdef DEBUG2
            printf("in case 2\n");
            printf("option = %s\n",option);
#endif
            break;

        case 3:
            sprintf(option,"%s %s",option,"-m");
#ifdef DEBUG2
            printf("in case 3\n");
            printf("option = %s\n",option);
#endif
            break;

        case 4:
            while(TRUE)
            {
                printf("Turn On (+) or Off (-) concurrency control? ");
                gets(buf);
                if ((buf[0] == '+') || (buf[0] == '-'))
                    break;
            }
    }
}
```



```
    }
    sprintf(option,"%s %s%s",option,buf,"c");
#ifdef DEBUG2
    printf("option = %s\n",option);
    printf("in case 4\n");
#endif
    break;

    case 5:
    while(TRUE)
    {
        printf("Turn On (+) or Off (-) Query Modification? ");
        gets(buf);
        if ((buf[0] == '+') || (buf[0] == '-'))
            break;
    }
    sprintf(option,"%s %s%s",option,buf,"q");
#ifdef DEBUG2
    printf("option = %s\n",option);
    printf("in case 5\n");
#endif
    break;

    case 6:
#ifdef DEBUG2
    printf("in case 6\n");
#endif
    return(DONE);
    break;

    default:
    break;
}
return (CONTINUE);
}
```

```
/*.....
 *
 *      CRMENU      *
 *      .....
 *
 * Function: This routine is used to print the options of the UNIX
 *           command creatdb to the user.
 *
 * Input: None.
 *
 * Output: A screen of options.
 *
 * Returns: The options to be executed.
 *
 */
crmenu()
{

    char aptbuf[100][3];
    int j, i, o, differ;

    printf("\n\n");
    printf("\t\t\tcreatdb options\n");
    printf("\t\t\t-----\n\n");
    printf("    1) -uname = specify a different DBA called");
    printf(" 'name'.\n");
    printf("    2) -e = modify options for an existing database.\n");
    printf("    3) -m = specifies that the UNIX directory in");
    printf(" which the dotbase\n\t\t\tis to reside already exists.\n");
    printf("    4) +/-c = turns on (+) or off (-) the concurrency");
    printf(" control scheme.\n");
    printf("    5) +/-q = turns on (+) or off (-) query");
    printf(" modification.\n");
    printf("    6) EXIT = exit from this option session.\n");
    printf("\n\nPlease enter the number of the option desired: ");

}
```

```

#include "defines.h"

extern char option[COMMANDENGTH];
extern char exdest[COMMANDENGTH];
extern char dotobase[DBLENGTH];

/*****
 *
 *   DESTCALL
 *
 *****/
*
* Function: This routine is used to formulote the command
*           destroydb.
*
* Input: None.
*
* Output: A screen of options.
*
* Returns: The options to be executed.
*
*/

destcoll()
{
    chor optbuf[100][3];
    int j, i, o, differ;

    sprintf(option,"%s"," ");
#ifdef DEBUG2
    printf("exdest = %s\n",exdest);
    printf("option = %s\n",option);
    printf("dest database = %s\n",dotobase);
#endif

    destmenu();
    for (i=0; ;i++)
    {
        scanf("%s", optbuf[i]);

        /* If the user's response is 'questionable', show the menu again */
        if ((*optbuf[i] == '?') || (*optbuf[i] == '\n'))
        {
            destmenu();
            continue;
        }

        /* If the user's response is an exit, get out of here */
        if ((*optbuf[i] == '3') || (*optbuf[i] == 'q'))
        {
            break;
        }

#ifdef DEBUG
        printf("inside the destex port ond i = %d!\n",i);
#endif
        for (j=0; j<=i; j++)
        {
            if (destcheck(optbuf[j]) == DONE)
            {
#ifdef DEBUG1
                printf("Got to just after ingcheck and returning\n");
#endif
                break;
            }
        }

#ifdef DEBUG
        printf("exdest before exec = %s\n",exdest);
#endif
        printf("\n%s\n",exdest);
    }
}

```

```
        if (system(exdest) == FAILURE)
        {
            syserr(101);
        }
        return(SUCCESS);
    }
    /* If the user's response is a number, convert it and process it */
    if ((*optbuf[i] >= '1') && (*optbuf[i] <= '3'))
    {
#ifdef DEBUG
        printf("1-9 optbuf [%d] = %c and o = %d\n", i, optbuf[i][0], o);
#endif
        continue;
    }
    printf("\nOption %s is not a legitimate option.\n", optbuf[i]);
}
}
```

```
#include "defines.h"
extern chor option[COMMANLENGTH];

/*****
 *
 *   DESTCHECK
 *
 *****/
/*
 * Function: This routine is used to check and prompt the user for
 *           more information based on which options the user has
 *           requested for the destroydb command.
 *
 * Input: None.
 *
 * Output: A list of options.
 *
 * Returns: The options to be executed.
 */

destcheck(pick)
chor pick[3];
{
    chor buf[5];
    int o;

    o = otoi(pick);

#ifdef DEBUG
    printf("o = %d and pick = %s\n",o,pick);
#endif
    switch(o)
    {
        case 1:
            sprintf(option,"%s %s",option,"-s");
#ifdef DEBUG
            printf("in cose 1\n");
            printf("option = %s\n",option);
#endif
            break;

        case 2:
            sprintf(option,"%s %s",option,"-m");
#ifdef DEBUG
            printf("option = %s\n",option);
            printf("in cose 2\n");
#endif
            break;

        case 3:
#ifdef DEBUG
            printf("in cose 3\n");
#endif
            getch();
            return(DONE);
            break;

        default:
            break;
    }
    return (CONTINUE);
}
```

```
/*.....
 *
 *      DESTMENU      *
 *      *            *
 *.....
 *
 * Function: This routine is used to print the options of the UNIX
 *           command destroydb to the user.
 *
 * Input: None.
 *
 * Output: A screen of options.
 *
 * Returns: The options to be executed.
 *
 */

destmenu()
{
    printf("\n\n");
    printf("\t\t\tdestroydb options\n");
    printf("\t\t\t-----\n\n");
    printf("    1) -s = INGRES superuser must use this to execute");
    printf(" destroydb.\n");
    printf("    2) -m = specifies that the UNIX directory in");
    printf(" which the dot base\n\t\tresides is not to be removed.\n");
    printf("    3) EXIT = exit from this option session.\n");
    printf("\n\nPlease enter the number of the option desired: ");
}
}
```

```

#include "defines.h"

extern char optian[COMMANLENGTH];
extern char exhelpr[COMMANLENGTH];
extern char database[DBLENGTH];
extern char rellist[PATHLENGTH];

/*****
 *
 *   HELPCALL
 *
 *****/
*
* Function: This routine is used to formulate the command help.
*
* Input: None.
*
* Output: A screen of options.
*
* Returns: SUCCESS or FAILURE.
*/

helprcall()
{
    char aptbuf[100][3];
    int j, i, a, differ;

    sprintf(optian,"%s", " ");
#ifdef DEBUG2
    printf("exhelpr = %s\n",exhelpr);
    printf("optian = %s\n",optian);
    printf("helprcall database = %s\n",database);
#endif

    helpmenu();

    for (i=0; ;i++)
    {
        scanf("%s", aptbuf[i]);

        /* If the user's response is 'questionable', show the menu again */
        if ((*aptbuf[i] == '?') || (*aptbuf[i] == '\n'))
        {
            helpmenu();
            continue;
        }

        /* If the user's response is an exit, get out of here */
        if ((*aptbuf[i] == '3') || (*aptbuf[i] == 'q'))
        {
            getchar();
#ifdef DEBUG2
            printf("inside the helprex part and i = %d\n",i);
#endif
            for (j=0; j<=i; j++)
            {
                if (helpcheck(aptbuf[j]) == DONE)
                {
#ifdef DEBUG2
                    printf("Got to just after helpcheck and returning\n");
#endif
                    break;
                }
            }

            sprintf(exhelpr,"%s%s %s %s","helpr",optian,database,rellist);
#ifdef DEBUG
            printf("exhelpr before exec = %s\n",exhelpr);
#endif
        }
    }
}

```

```
#endif
    printf("\n%s\n",exhelpr);
    if (system(exhelpr) == FAILURE)
    {
        syserr(101);
    }
    return(SUCCESS);
}
/* If the user's response is a number, convert it and process it */
if ((*optbuf[i] >= '1') && (*optbuf[i] <= '3'))
{
#ifdef DEBUG2
    printf("1-9 optbuf [%d] = %c and o = %d\n", i,optbuf[i][0], o);
#endif
    continue;
}
printf("\nOption %s is not a legitimate option.\n",optbuf[i]);
}
}
```



```
#include "defines.h"
```

```
extern option[COMMANDELENGTH];
```

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

```
*
*   HELPRCHECK   *
*
*.....
```

```
* Function: This routine is used to check and prompt the user for
*           more information based on which options the user has
*           requested for the helpr command.
```

```
* Input: None.
```

```
* Output: A list of options.
```

```
* Returns: The options to be executed.
```

```
*/
```

```
helprcheck(pick)
```

```
  chor pick[3];
```

```
{
```

```
  chor buf[5];
```

```
  int o, m;
```

```
  a = otoi(pick);
```

```
#ifdef DEBUG
```

```
  printf("a = %d and pick = %s\n",a,pick);
```

```
#endif
```

```
  switch(o)
```

```
  {
```

```
    cose 1:
```

```
      printf("What is the user's login name? ");
```

```
      gets(buf);
```

```
      sprintf(option,"%s %s%s",option,"-u",buf);
```

```
#ifdef DEBUG
```

```
  printf("in case 1\n");
```

```
  printf("option = %s\n",option);
```

```
#endif
```

```
    break;
```

```
    cose 2:
```

```
      while(TRUE)
```

```
      {
```

```
        printf("Wait (+) or Do Not Wait (-) for the Doto Base? ");
```

```
        gets(buf);
```

```
        if ((buf[0] == '+') || (buf[0] == '-'))
```

```
          break;
```

```
      }
```

```
      sprintf(option,"%s %s%s",option,buf,"w");
```

```
#ifdef DEBUG
```

```
  printf("option = %s\n",option);
```

```
  printf("in case 2\n");
```

```
#endif
```

```
    break;
```

```
    cose 3:
```

```
#ifdef DEBUG
```

```
  printf("in case 3\n");
```

```
#endif
```

```
    return(DONE);
```

```
    break;
```

```
  default:
```

```
        break;
    }
    for (m=0; m<5; m++)
        buf[m] = 0;
#ifdef DEBUG
    printf("•buf = %s\n",buf);
#endif
    return (CONTINUE);
}
```



```

#include "defines.h"

extern char option[COMMANDENGTH];
extern chor exingres[COMMANDENGTH];
extern char database[DLENGTH];

/*****
 *
 *   INGRESCALL
 *
 *****/
/*
 * Function: This routine is used to formulote the commond ingres.
 *
 * Input: Nane.
 *
 * Output: A screen af options.
 *
 * Returns: SUCCESS or FAILURE.
 */

ingrescoll()
{
    char optbuf[100][3];
    int j, i, o, differ;

    sprintf(option,"%s"," ");
#ifdef DEBUG
    printf("exingres = %s\n",exingres);
    printf("option = %s\n",option);
    printf("ingrescall database = %s\n",database);
#endif

    ingmenu();

    for (i=0; ;i++)
    {
        scanf("%s", optbuf[i]);

        /* If the user's response is 'questionable', show the menu ogain */
        if ((optbuf[i][0] == '?') || (optbuf[i][0] == '\n'))
        {
            ingmenu();
            continue;
        }

        /* If the user's response is on exit, get out of here */
        if (((optbuf[i][0] == 'i') && (optbuf[i][1] == '4'))
            || (*optbuf[i] == 'q'))
        {
            getchcor();
#ifdef DEBUG
            printf("inside the ex port ond i = %d!\n",i);
#endif
            for (j=0; j<=i; j++)
            {
                if (ingcheck(optbuf[j]) == DONE)
                {
#ifdef DEBUG
                    printf("Got to just offer ingcheck ond returning\n");
#endif
                    break;
                }
            }
            sprintf(exingres,"%s%s %s","ingres",option,database);
#ifdef DEBUG
            printf("exingres before exec = %s\n",exingres);
#endif
        }
    }
}

```

```
        printf("\n%s\n",exingres);
        if (system(exingres) == FAILURE)
        {
            syserr(101);
        }
        return(SUCCESS);
    }
    /* If the user's response is a number, convert it and process it */
    if ((*optbuf[i] >= '1') && (*optbuf[i] <= '9'))
    {
#ifdef DEBUG
        printf("1-9 optbuf [%d] = %c and a = %d\n", i,optbuf[i][0], a);
#endif
        continue;
    }
    printf("\nOption %s is not a legitimate option.\n",optbuf[i]);
}
}
```

```

#include "defines.h"
extern char option[COMMANDELENGTH];

/.....
*
*   INGCHECK   *
*
*.....
*
* Function: This routine is used to check and prompt the user for
*           more information based on which options the user has
*           requested for the ingres command.
*
* Input: None.
*
* Output: A list of options.
*
* Returns: The options to be executed.
*
*/

ingcheck(pick)
char pick[3];
{
    char buf[5];
    int a, m;

    a = atoi(pick);

#ifdef DEBUG
    printf("a = %d and pick = %s\n",a,pick);
#endif
    switch(a)
    {
        case 1:
            while(TRUE)
            {
                printf("Enable (+) or Disable (-) Direct Update? ");
                gets(buf);
                if ((buf[0] == '+') || (buf[0] == '-'))
                    break;
            }
            sprintf(option,"%s %s%s",option,buf,"U");
#ifdef DEBUG
            printf("in case 1\n");
            printf("option = %s\n",option);
#endif
            break;

        case 2:
            printf("What is the user's login name? ");
            gets(buf);
            sprintf(option,"%s %s%s",option,"-u",buf);
#ifdef DEBUG
            printf("in case 2\n");
            printf("option = %s\n",option);
#endif
            break;

        case 3:
            printf("What is the minimum field width? ");
            gets(buf);
            sprintf(option,"%s %s%s",option,"-c",buf);
#ifdef DEBUG
            printf("in case 3\n");
            printf("option = %s\n",option);
#endif
    }
}

```

```

#endif
    break;

    case 4:
        printf("What is the integer output field width? ");
        gets(buf);
        sprintf(option,"%s %s%s",option,"-i",buf);
#ifdef DEBUG
        printf("option = %s\n",option);
        printf("in case 4\n");
#endif
    break;

    case 5:
        printf("Set the floating point output field width to M");
        printf(" characters with N decimal places.\n");
        printf("What is the l x M.N parameter? ");
        gets(buf);
        sprintf(option,"%s %s%s",option,"-f",buf);
#ifdef DEBUG
        printf("option = %s\n",option);
        printf("in case 5\n");
#endif
    break;

    case 6:
        printf("What is the column separator? ");
        gets(buf);
        sprintf(option,"%s %s%s",option,"-v",buf);
#ifdef DEBUG
        printf("option = %s\n",option);
        printf("in case 6\n");
#endif
    break;

    case 7:
        printf("What is the modify mode on the retrieve command? ");
        gets(buf);
        sprintf(option,"%s %s%s",option,"-r",buf);
#ifdef DEBUG
        printf("option = %s\n",option);
        printf("in case 7\n");
#endif
    break;

    case 8:
        printf("What is the modify mode on the index command? ");
        gets(buf);
        sprintf(option,"%s %s%s",option,"-n",buf);
#ifdef DEBUG
        printf("option = %s\n",option);
        printf("in case 8\n");
#endif
    break;

    case 9:
        while(TRUE)
        {
            printf("Enable (+) or Disable (-) Autoclear Option? ");
            gets(buf);
            if ((buf[0] == '+') || (buf[0] == '-'))
                break;
        }
        sprintf(option,"%s %s%s",option,buf,"a");
#ifdef DEBUG
        printf("option = %s\n",option);
        printf("in case 9\n");
#endif
    break;

```

```

    case 10:
        while(TRUE)
        {
            printf("Enable (+) or Disable (-) Batch Update? ");
            gets(buf);
            if ((buf[0] == '+') || (buf[0] == '-'))
                break;
        }
        sprintf(option,"%s %s%s",option,buf,"b");
#ifdef DEBUG
        printf("option = %s\n",option);
        printf("in case 10\n");
#endif
        break;

    case 11:
        while(TRUE)
        {
            printf("Print (+) or Do Not Print (-) the Dayfile? ");
            gets(buf);
            if ((buf[0] == '+') || (buf[0] == '-'))
                break;
        }
        sprintf(option,"%s %s%s",option,buf,"d");
#ifdef DEBUG
        printf("option = %s\n",option);
        printf("in case 11\n");
#endif
        break;

    case 12:
        while(TRUE)
        {
            printf("Print (+) or Do Not Print (-) Monitor Messages? ");
            gets(buf);
            if ((buf[0] == '+') || (buf[0] == '-'))
                break;
        }
        sprintf(option,"%s %s%s",option,buf,"s");
#ifdef DEBUG
        printf("option = %s\n",option);
        printf("in case 12\n");
#endif
        break;

    case 13:
        while(TRUE)
        {
            printf("Wait (+) or Do Not Wait (-) for the Data Base? ");
            gets(buf);
            if ((buf[0] == '+') || (buf[0] == '-'))
                break;
        }
        sprintf(option,"%s %s%s",option,buf,"w");
#ifdef DEBUG
        printf("option = %s\n",option);
        printf("in case 13\n");
#endif
        break;

    case 14:
#ifdef DEBUG
        printf("in case 14\n");
#endif
        return(DONE);
        break;

    default:

```



```
        break;
    }
    for (m=0; m<5; m++)
        buf[m] = 0;
#ifdef DEBUG
    printf("buf = %s\n",buf);
#endif
    return (CONTINUE);
}
```



```
/*.....
 *          LOCKBOX          *
 *.....
 *
 * Function: This routine will handle the concurrency problem of
 *           two users wanting to access the same encrypted relation
 *           at the same time. Before reading, checking and
 *           modifying a .crypt file's busy bit, run "lockbox".
 *           After the change has taken place run "unlockbox".
 *
 * Input:  None.
 *
 * Output: None.
 *
 * Returns: An OK to open the .crypt file.
 */
#include <stdio.h>

lockbox()
{
    FILE *fptr;
    int i, j;

    for (i=0; i<50; i++)
    {

        if ((j=access("/usr/ingres/lock.box",0)) == -1)
        {

            if ((fptr = fopen("/usr/ingres/lock.box", "o+")) != NULL)
            {
                fclose(fptr);
                return( 0 );
            }
            else
                fprintf(stderr,"Can't create /usr/ingres/lock.box file\n");
        }
        sleep(1);
    }
    fprintf(stderr,"INGRES Front End times out waiting on lockbox\n");
    return( -1 );
}
```

```
/******  
*  
*      MENU      *  
*  
*****  
*  
* Function:  This routine is used to print the options of the INGRES  
*           related UNIX cammonds to the User in a menu type format.  
*  
* Input:    Nane.  
*  
* Output:   A screen of options.  
*  
* Returns:  The cammond number to be executed = pick.  
*  
*/  
  
menu()  
{  
  
    printf("\n\n\n");  
    printf("    Below are the INGRES related UNIX cammonds available:\n\n");  
    printf("    1) CHANGEKEY - change a relation's key.\n");  
    printf("    2) COPYDB - create batch files to copy out a");  
    printf(" data base and restare it.\n");  
    printf("    3) CREATDB - create a data base.\n");  
    printf("    4) DESTROYDB - destroy on existing data base.\n");  
    printf("    5) HELPR - get infarmation about a data base.\n");  
    printf("    6) INGRES - INGRES relational data base");  
    printf(" management system.\n");  
    printf("    7) LISTREL - list relations a user has");  
    printf(" access to in a certain data base.\n");  
    printf("    8) PRINTR - print relations.\n");  
    printf("    9) PURGE - destroy all expired and temporary");  
    printf(" relations.\n");  
    printf("   10) RESTORE - recover from an INGRES or UNIX crash.\n");  
    printf("   11) SYSMOD - madify system relations to predetermined");  
    printf(" storage structures.\n");  
    printf("   12) EXIT - exit from this user's session.\n");  
  
}
```

```

#include "defines.h"

extern char option[COMMANDELENGTH];
extern char exprintr[COMMANDELENGTH];
extern char database[DBLENGTH];
extern char rellist[PATHLENGTH];

/*****
 *
 *   PRINTRCALL
 *
 *****/
/*
 * Function: This routine is used to formulate the command
 *           printr.
 *
 * Input: None.
 *
 * Output: A screen of options.
 *
 * Returns: The options to be executed.
 */
printrcall()
{
    char optbuf[100][3];
    int j, i, a, differ;

    sprintf(option,"%s", " ");
#ifdef DEBUG
    printf("exprintr = %s\n",exprintr);
    printf("option = %s\n",option);
    printf("printr database = %s\n",database);
#endif

    printmenu();

    for (i=0; ;i++)
    {
        scanf("%s", optbuf[i]);
        getchar();

        /* If the user's response is 'questionable', show the menu again */
        if ((optbuf[i] == '?') || (optbuf[i] == '\n'))
        {
            printmenu();
            continue;
        }

        /* If the user's response is an exit, get out of here */
        if ((optbuf[i] == '7') || (optbuf[i] == 'q'))
        {
            break;
        }

#ifdef DEBUG
        printf("inside the printrex part and i = %d\n",i);
#endif
        for (j=0; j<=i; j++)
        {
            if (printrcheck(optbuf[j]) == DONE)
                break;
        }
        sprintf(exprintr,"%s%s %s %s", "printr",option,database,rellist);
#ifdef DEBUG
        printf("exprintr before exec = %s\n",exprintr);
#endif
        printf("\n%s\n",exprintr);

        if (system(exprintr) == FAILURE)

```

```
        syserr(101);
        return(SUCCESS);
    }
    /* If the user's response is a number, convert it and process it */
    if ((*optbuf[i] >= '1') && (*optbuf[i] <= '7'))
    {
#ifdef DEBUG
        printf("1-9 optbuf [%d] = %c and a = %d\n", i, optbuf[i][0], a);
#endif
        continue;
    }
    printf("\nOption %s is not a legitimate option.\n", optbuf[i]);
}
}
```

```
#include "defines.h"

extern char option[COMMANLENGTH];

/*****
 *
 *   PRINTRCHECK
 *
 *****/
*
* Function: This routine is used to check and prompt the user for
* more information based on which options the user has
* requested for the printr command.
*
* Input: None.
*
* Output: A list of options.
*
* Returns: The options to be executed.
*/

printrcheck(pick)
char pick[3];
{
    char buf[5];
    int a, m;

    o = atoi(pick);

#ifdef DEBUG
    printf("o = %d and pick = %s\n",a,pick);
#endif
    switch(o)
    {
        case 1:
            printf("What is the user's login name? ");
            gets(buf);
            sprintf(option,"%s %s%s",option,"-u",buf);
#ifdef DEBUG
            printf("in case 1\n");
            printf("option = %s\n",option);
#endif
            break;

        case 2:
            printf("What is the minimum field width? ");
            gets(buf);
            sprintf(option,"%s %s%s",option,"-c",buf);
#ifdef DEBUG
            printf("in case 2\n");
            printf("option = %s\n",option);
#endif
            break;

        case 3:
            printf("What is the integer output field width? ");
            gets(buf);
            sprintf(option,"%s %s%s",option,"-i",buf);
#ifdef DEBUG
            printf("option = %s\n",option);
            printf("in case 3\n");
#endif
            break;

        case 4:
            printf("Set the floating point output field width to M");
    }
}
```

```
        printf(" chocharacters with N decimol places.\n");
        printf("Whot is the lxm.N porometer? ");
        gets(buf);
        sprintf(option,"%s %s%s",option,"-f",buf);
#ifdef DEBUG
        printf("option = %s\n",option);
        printf("in cose 4\n");
#endif
        break;

        cose 5:
        printf("Whot is the column seporotor? ");
        gets(buf);
        sprintf(option,"%s %s%s",option,"-v",buf);
#ifdef DEBUG
        printf("option = %s\n",option);
        printf("in cose 5\n");
#endif
        break;

        cose 6:
        while(TRUE)
        {
            printf("Wait (+) or Do Not Wait (-) for the Data Bose? ");
            gets(buf);
            if ((buf[0] == '+') || (buf[0] == '-'))
                break;
        }
        sprintf(option,"%s %s%s",option,buf,"w");
#ifdef DEBUG
        printf("option = %s\n",option);
        printf("in cose 6\n");
#endif
        break;

        cose 7:
#ifdef DEBUG
        printf("in cose 7\n");
#endif
        return(DONE);
        break;

        default:
            break;
    }
    for (m=0; m<5; m++)
        buf[m] = 0;
#ifdef DEBUG
    printf("buf = %s\n",buf);
#endif
    return (CONTINUE);
}
```



```

#include "defines.h"

extern char option[COMMANLENGTH];
extern char expurge[COMMANLENGTH];
extern char database[DLENGTH];

/*****
 *
 *   PURGECALL
 *
 *****/
*
* Function: This routine is used to formulate the command
*           purge.
*
* Input: None.
*
* Output: A screen of options.
*
* Returns: The options to be executed.
*/

purgcall()
{
    char optbuf[100][3];
    int j, i, a, differ;

    sprintf(option,"%s", " ");
#ifdef DEBUG
    printf("expurge = %s\n",expurge);
    printf("option = %s\n",option);
    printf("purge database = %s\n",database);
#endif

    purgmenu();

    for (i=0; i++)
    {
        scanf("%s", optbuf[i]);
        getchar();

        /* If the user's response is 'questionable', show the menu again */
        if ((*optbuf[i] == '?') || (*optbuf[i] == '\n'))
        {
            purgmenu();
            continue;
        }

        /* If the user's response is an exit, get out of here */
        if ((*optbuf[i] == '6') || (*optbuf[i] == 'q'))
        {
#ifdef DEBUG
            printf("inside the purgex port and i = %d!\n",i);
#endif
            for (j=0; j<=i; j++)
            {
                if (purgecheck(optbuf[j]) == DONE)
                {
#ifdef DEBUG1
                    printf("Got to just after purgecheck ond returning\n");
#endif
                    break;
                }
            }
            sprintf(expurge,"%s%s %s","purge",option,database);
#ifdef DEBUG
            printf("expurge before exec = %s\n",expurge);
#endif
        }
    }
}

```

```
#endif
    printf("\n%s\n",expurge);
    if (system(expurge) == FAILURE)
    {
        syserr(101);
    }
    return(SUCCESS);
}
/* If the user's response is a number, convert it and process it */
if ((*optbuf[i] >= '1') && (*optbuf[i] <= '6'))
{
#ifdef DEBUG
    printf("1-9 optbuf [%d] = %c and a = %d\n", i,optbuf[i][0], a);
#endif
    continue;
}
printf("\nOption %s is not a legitimate option.\n",optbuf[i]);
}
}
```

```
#include "defines.h"
extern char option[COMMANLENGTH];

/*****
 *
 *   PURGECHECK   *
 *
 *****/
* Function: This routine is used to check and prompt the user for
* more information based on which options the user has
* requested for the purge command.
*
* Input: None.
*
* Output: A list of options.
*
* Returns: The options to be executed.
*/

purgecheck(pick)
char pick[3];
{
    char buf[5];
    int a, m;

    a = atoi(pick);

#ifdef DEBUG
    printf("a = %d and pick = %s\n",a,pick);
#endif
    switch(a)
    {
        case 1:
            sprintf(option,"%s %s",option,"-f");
#ifdef DEBUG
            printf("in case 1\n");
            printf("option = %s\n",option);
#endif
            break;

        case 2:
            sprintf(option,"%s %s",option,"-p");
#ifdef DEBUG
            printf("in case 2\n");
            printf("option = %s\n",option);
#endif
            break;

        case 3:
            sprintf(option,"%s %s",option,"-a");
#ifdef DEBUG
            printf("in case 3\n");
            printf("option = %s\n",option);
#endif
            break;

        case 4:
            sprintf(option,"%s %s",option,"-s");
#ifdef DEBUG
            printf("option = %s\n",option);
            printf("in case 4\n");
#endif
            break;
    }
}
```

```
case 5:
    while(TRUE)
    {
        printf("Wait (+) or Do Not Wait (-) for the Dato Base? ");
        gets(buf);
        if ((buf[0] == '+') || (buf[0] == '-'))
            break;
    }
    sprh~tf(option, "%s %s%s", option, buf, "w");
#ifdef DEBUG
    printf("option = %s\n", option);
    printf("in case 5\n");
#endif
    break;

case 6:
#ifdef DEBUG
    printf("in case 6\n");
#endif
    return(DONE);
    break;

default:
    break;
}
for (m=0; m<5; m++)
    buf[m] = 0;
return (CONTINUE);
}
```

```
/*.....
 *
 *      PURGEMENU      *
 *.....
 *
 * Function:  This routine is used to print the options of the UNIX
 *            command purge to the user.
 *
 * Input:    None.
 *
 * Output:   A screen of options.
 *
 * Returns:  The options to be executed.
 */

purgemenu()
{

    printf("\n\n\n");
    printf("\t\t\tpurge options\n");
    printf("\t\t\t-----\n");
    printf("    1)  -p = expired user relations are deleted.\n");
    printf("    2)  -f = causes unrecognizable files to be");
    printf(" deleted\n");
    printf("    3)  -o = causes messages to be printed about");
    printf(" the pending operation\n\t\t\tand execute it only");
    printf(" if the response is o 'y'.\n");
    printf("    4)  -s = INGRES superuser must use this to execute");
    printf(" purge.\n");
    printf("    5)  +/-w = wait/do not wait for the database.\n");
    printf("    6)  EXIT = exit from this option session.\n");
    printf("\nPlease enter the number of the option desired: ");

}
}
```

```

#include "defines.h"

extern char option[COMMANLENGTH];
extern char exrestore[COMMANLENGTH];
extern char database[DBLENGTH];

/*****
 *          RESTCALL          *
 *****/
*
* Function: This routine is used to formulate the command
*          restore.
*
* Input: None.
*
* Output: A screen of options.
*
* Returns: The options to be executed.
*/

restcall()
{
    char optbuf[100][3];
    int j, i, o, differ;

    sprintf(option,"%s"," ");
#ifdef DEBUG
    printf("exrestore = %s\n",exrestore);
    printf("option = %s\n",option);
    printf("restore database = %s\n",database);
#endif

    restmenu();

    for (i=0; ;i++)
    {
        scanf("%s", optbuf[i]);
        getch();

        /* If the user's response is 'questionable', show the menu again */
        if ((optbuf[i] == '?') || (optbuf[i] == '\n'))
        {
            restmenu();
            continue;
        }

        /* If the user's response is an exit, get out of here */
        if ((optbuf[i] == '6') || (optbuf[i] == 'q'))
        {
            break;
        }

#ifdef DEBUG
        printf("inside the restorex port and i = %d!\n",i);
#endif

        for (j=0; j<=i; j++)
        {
            if (restcheck(optbuf[j]) == DONE)
            {
#ifdef DEBUG1
                printf("Got to just after restcheck and returning\n");
            }
            break;
        }

        sprintf(exrestore,"%s%s %s","restore",option,database);
#ifdef DEBUG
        printf("exrestore before exec = %s\n",exrestore);
            }
    }
}

```

```
#endif
    printf("\n%s\n", exrestore);
    if (system(exrestore) == FAILURE)
    {
        syserr(101);
    }
    return(SUCCESS);
}
/* If the user's response is a number, convert it and process it */
if ((*optbuf[i] >= '1') && (*optbuf[i] <= '6'))
}
#ifdef DEBUG
    printf("1-9 aptbuf [%d] = %c and a = %d\n", i, aptbuf[i][0], a);
#endif
    continue;
}
    printf("\nOption %s is not a legitimate option.\n", aptbuf[i]);
}
}
```



```
#include "defines.h"
extern char option[COMMANLENGTH];

/*****
 *
 *   RESTCHECK
 *
 *****/
* Function: This routine is used to check and prompt the user for
* more information based on which options the user has
* requested for the restore command.
*
* Input: None.
*
* Output: A list of options.
*
* Returns: The options to be executed.
*/

restcheck(pick)
char pick[3];
{
    char buf[5];
    int o, m;

    o = otoi(pick);

#ifdef DEBUG
    printf("o = %d and pick = %s\n",o,pick);
#endif
    switch(a)
    {
        case 1:
            sprintf(option,"%s %s",option,"-p");
#ifdef DEBUG
            printf("in case 1\n");
            printf("option = %s\n",option);
#endif
            break;

        case 2:
            sprintf(option,"%s %s",option,"-f");
#ifdef DEBUG
            printf("in case 2\n");
            printf("option = %s\n",option);
#endif
            break;

        case 3:
            sprintf(option,"%s %s",option,"-a");
#ifdef DEBUG
            printf("in case 3\n");
            printf("option = %s\n",option);
#endif
            break;

        case 4:
            sprintf(option,"%s %s",option,"-s");
#ifdef DEBUG
            printf("option = %s\n",option);
            printf("in case 4\n");
#endif
            break;
    }
}
```

```
    cose 5:
    while(TRUE)
    {
        printf("Wait (+) or Do Not Wait (-) for the Doto Bose? ");
        gets(buf);
        if ((buf[0] == '+') || (buf[0] == '-'))
            break;
    }
    sprintf(option,"%s %s%s",option,buf,"w");
#ifdef DEBUG
    printf("option = %s\n",option);
    printf("in cose 5\n");
#endif
    break;

    cose 6:
#ifdef DEBUG
    printf("in cose 6\n");
#endif
    return(DONE);
    break;

    default:
        break;
}
for (m=0; m<5; m++)
    buf[m] = 0;

return (CONTINUE);
}
```

```
/*.....
 *
 *      RESTMENU      *
 *      *            *
 *.....
 *
 * Function:  This routine is used to print the options of the UNIX
 *            command restore to the user.
 *
 * Input:    None.
 *
 * Output:   A screen of options.
 *
 * Returns:  The options to be executed.
 */
restmenu()
{
    printf("\n\n");
    printf("\t\t\trestore options\n");
    printf("\t\t\t-----\n");
    printf("    1)  -p = if restore completes with no errors.");
    printf(" purge is called\n\t\t\tand expired user relations are deleted.\n");
    printf("    2)  -f = if restore completes with no errors.");
    printf(" purge is called and\n\t\t\tunrecognizable files will be");
    printf(" deleted.\n");
    printf("    3)  -a = causes messages to be printed about");
    printf(" the pending operation\n\t\t\tand execute it only");
    printf(" if the response is a 'y'.\n");
    printf("    4)  -s = INGRES superuser must use this to execute");
    printf(" restore.\n");
    printf("    5)  +/ -w = wait/do not wait for the database.\n");
    printf("    6)  EXIT = exit from this option session.\n");
    printf("\nPlease enter the number of the option desired: ");
}
}
```

```

#include "defines.h"
#include <sys/signol.h>

extern chor option[COMMANDLENGTH];
extern chor option1[COMMANDLENGTH];
extern char exsysmod[COMMANDLENGTH];
extern chor database[DBLENGTH];

/*****
 *
 *      SYSCALL      *
 *
 *****/
*
* Function: This routine is used to formulate the command
*           sysmod.
*
* Input: None.
*
* Output: A screen of options.
*
* Returns: The options to be executed.
*
*/

syscall()
{
    chor optbuf[100][3];
    int j, i, a, differ;
    int done = 0;
    j = -1;

    sprintf(option, "%s", " ");
    sprintf(option1, "%s", " ");
#ifdef DEBUG2
    printf("exsysmod = %s\n", exsysmod);
    printf("option = %s\n", option);
    printf("option1 = %s\n", option1);
    printf("sysmod dotobase = %s\n", database);
#endif

    sysmenu();

    for (i=0; done != DONE ;i++)
    {
        scanf("%s", aptbuf[i]);
        getchar();

        /* If the user's response is 'questionable', show the menu again */
        if ((*aptbuf[i] == '?') || (*aptbuf[i] == '\n'))
        {
            sysmenu();
            continue;
        }

        /* If the user's response is an exit, get out of here */
        if ((*aptbuf[i] == '3') || (*aptbuf[i] == 'q'))
        {
            for (j=0; j<=i; j++)
            {
                if (syscheck(optbuf[j]) == DONE)
                {
                    done = DONE;
                    break;
                }
            }
        }
    }

    /* If the user's response is o number, convert it and process it */

```

```

        if ((*optbuf[i] >= '1') && (*optbuf[i] <= '3'))
            continue;
        printf("\nOption %s is not a legitimate option.\n",optbuf[i]);
    }

    sysrel();
    done = 0;

    for (i=0; done != DONE ;i++)
    {
        scanf("%s", optbuf[i]);

/* If the user's response is 'questionable', show the menu again */
        if ((*optbuf[i] == '?') || (*optbuf[i] == '\n'))
        {
            sysrel();
            continue;
        }
/* If the user's response is an exit, get out of here */
        if ((*optbuf[i] == '7') || (*optbuf[i] == 'q'))
        {
            for (j=0; j<=i; j++)
            {
                if (sysrelcheck(optbuf[j]) == DONE)
                {
                    done = DONE;
                    break;
                }
            }
/* If the user's response is a number, convert it and process it */
            if ((*optbuf[i] >= '1') && (*optbuf[i] <= '7'))
                continue;
            printf("\nOption %s is not a legitimate option.\n",optbuf[i]);
        }
    }

    sprintf(exsysmod,"%s%s %s %s","sysmod",option,database,option1);
#ifdef DEBUG2
    printf("exsysmod before exec = %s\n",exsysmod);
#endif
    printf("\n%s\n",exsysmod);
    getchar();

    if (system(exsysmod) == FAILURE)
        syserr(101);
    return(SUCCESS);
}

```

```
#include "defines.h"

extern char option[COMMANDELENGTH];

/*****
 *
 *   SYSCHECK
 *
 *****/
* Function: This routine is used to check and prompt the user for
*           more information based on which options the user has
*           requested for the sysmod command.
*
* Input: None.
*
* Output: A list of options.
*
* Returns: The options to be executed.
*/

syscheck(pick)
char pick[3];
{
    char buf[5];
    int a, m;

    a = atoi(pick);

    switch(a)
    {
        case 1:
            sprintf(option,"%s %s",option,"-s");
            break;

        case 2:
            while(TRUE)
            {
                printf("Wait (+) or Do Not Wait (-) for the Data Base? ");
                gets(buf);
                if ((buf[0] == '+') || (buf[0] == '-'))
                    break;
            }
            sprintf(option,"%s %s%s",option,buf,"w");
            break;

        case 3:
            return(DONE);
            break;

        default:
            break;
    }
    for (m=0; m<5; m++)
        buf[m] = 0;

    return (CONTINUE);
}
```



```
#include "defines.h"

extern char option1[COMMANLENGTH];

/*****
 *
 *   SYSRELCHECK   *
 *
 *****/
*
* Function: This routine is used to check and prompt the user for
*           the relations to be modified using the sysmod command.
*
* Input: None.
*
* Output: A list of options.
*
* Returns: The options to be executed.
*
*/

sysrelcheck(pick)
char pick[3];
{
    int a, m;

    a = atoi(pick);

    switch(a)
    {
        case 1:
            sprintf(option1,"%s %s",option1,"relation");
            break;

        case 2:
            sprintf(option1,"%s %s",option1,"attribute");
            break;

        case 3:
            sprintf(option1,"%s %s",option1,"indexes");
            break;

        case 4:
            sprintf(option1,"%s %s",option1,"tree");
            break;

        case 5:
            sprintf(option1,"%s %s",option1,"protect");
            break;

        case 6:
            sprintf(option1,"%s %s",option1,"integrities");
            break;

        case 7:
            return(DONE);
            break;

        default:
            break;
    }

    return (CONTINUE);
}
```

```
/*.....  
*  
*   UNLOCKBOX   *  
*  
*.....  
*  
* Function: This routine will handle the concurrency problem of  
*           two users wanting to access the some encrypted relation  
*           of the same time. After reading, checking and  
*           modifying o .crypt file's busy bit, run "unlockbox".  
*           This follows the use of "lockbox" before doing the  
*           reading, checking and writing.  
*  
* Input: None.  
*  
* Output: None.  
*  
* Returns: None.  
*  
*/  
  
unlockbox()  
{  
    unlink("/usr/ingres/lock.box");  
    return( 0 );  
}
```

COMPUTER SECURITY IN THE UNIX OPERATING SYSTEM
AND THE INGRES DATA BASE MANAGEMENT SYSTEM

by

LORI LYNN SABRACK

B.A. and B.S., Miami University, 1980

AN ABSTRACT OF A MASTER'S THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1987

CI 051
18-1

Abstract

Computer security has become an increasingly important subject in today's society where the use of computers has been incorporated either directly or indirectly in all phases of everyday life. This paper will address current computer security breaches as well as methods used to enhance computer security. The INGRES (Interactive Graphics and Retrieval System) Data Base Management System (DBMS) operating under the UNIX* operating system will be the target for security enhancements to be discussed in detail in this paper and one to be implemented at Kansas State University. The design for the implementation using INGRES and UNIX discusses in detail the relevant design features of INGRES and UNIX and the relationship that exists between them. The focus of the design is current security measures, strengths and weaknesses of the combined systems and included is an example of current usage.

* UNIX is a trademark of Bell Laboratories