LISA IV
October 17-19, 1990
Colo. Springs, Colorado

David A. Curry – SRI International
Samuel D. Kimery, Kent C. De La Croix, and Jeffrey R. Schwab – Purdue University

ACMAINT: An Account Creation and Maintenance System for Distributed UNIX Systems

ABSTRACT

ACMAINT is a network-based, centralized database system used to manage computer account creation and maintenance on the Purdue University Engineering Computer Network. ACMAINT allows the system administrator to perform account-related admintrative chores for any machine on the network from any attached system. Using ACMAINT, the system adminstrator can create new user accounts, add or delete accounts for existing users, change the global or per-account information associated with a user, place a message on a user's account(s), and enable or disable a user's account(s). Group information and mail aliases are managed in a similar fashion. ACMAINT utilizes a central database, stored on a single network machine, which contains a copy of all data under ACMAINT's control. The system administrator makes changes to the database via a network server running on the database machine, which in turn makes changes around the network via the use of another network server which runs on each machine. Programs which read, but do not write, the standard UNIX system databases such as the password file do not need to be modified to work with ACMAINT. Programs which write the standard databases must be modified or rewritten to converse with the ACMAINT database server. ACMAINT operates transparently to the user, uses minimal network and system resources, and can be used with binary-only UNIX systems.

Introduction

The Purdue University Engineering Computer Network[1] provides computing services to the seven Schools of Engineering using a network of approximately 25 UNIX timesharing machines and over 300 Sun workstations. Across the whole network, there are about 12,000 individual users and over 90,000 actual accounts. Because most of the computer accounts on these systems belong to students, there is a large amount of turnover each semester. In the School of Electrical Engineering alone, nearly 2,000 new accounts are created each fall, and between 1,000 and 2,000 accounts are deleted each summer.

Each School of Engineering has administrative control over the machines it owns. A system administrator in each school has the responsibility for creating, deleting, and updating computer accounts for the students, faculty, and staff in that school. The system administrators must coordinate their activities with each other in order to avoid creating problems. The large turnover in accounts presents several problems both in account creation (duplicate login names, etc.) and account deletion (deleting users with accounts on more than one department's machines, etc.).

Other problems surfaced as the network expanded to include more and more workstation systems. Changing information for a user with many accounts (e.g., an expiration date or classification) was a time-consuming process which required the system administrator to log in to each machine the user had an account on and change the information manually. This was particularly problematic at the start of each fall semester, when many users who were classified as ''freshman engineering'' became sophomores and selected a specific discipline, requiring a change in their classification.

The problems of managing unique user ids and login names, time-consuming account installation and maintenance processes, and the desire to have all user information centrally located convinced us that a system to manage computer account creation and maintenance on a network-wide basis was needed. The system selected had to meet several

[1]This work was done while David Curry was employed by the Purdue University Engineering Computer Network.

requirements, including:

- Centralized data storage
- Machine and vendor independence
- Flexibility in data to be stored
- Minimal changes to existing software
- Automated account installation
- Easy recovery from crashes
- Automated account deletion
- Simultaneous access for multiple users

In addition to these requirements, several other features were identified as ''nice to have'' but not required. These included the ability for the system administrator to change any piece of information from any host, the ability to change a piece of information (e.g., a user's password) on all hosts with a single command, the ability to generate ''batch'' updates to the database (e.g. from a Registrar's tape), and the ability to search the database for various combinations of information.

Previous Approaches to the Problem

Before deciding to implement our own account maintenance system, we examined other approaches to the problem of large network system administration. We had hoped that someone who had already faced our problems would have solved them to our satisfaction. Unfortunately, this was not the case. This section briefly describes the three systems we examined, and our reasons for not using them.

Yellow Pages

Yellow Pages (YP) [4] is a distributed network lookup service from Sun Microsystems. It is perhaps the most widely-known approach to system administration on a large network. YP maintains a set of files, called *maps*, which contain keys and associated values. YP maps can be replicated on several systems known as YP servers, each of which runs a server process for the maps. C library routines which query UNIX databases such as the password or hosts files are modified to query a YP server instead of reading the file directly. This enables the system administrator to make changes to only a few files (the ones on the YP master machines) and have the changes take effect on all the machines in the network. Programs require no modifications to use YP; when they are linked with the new C library they automatically gain this capability.

YP meets most of the requirements set forth in Section 1: it centralizes data storage, is flexible in what data may be stored, is machine and vendor independent, and requires no software changes. Unfortunately, there are no provisions for automated account installation or deletion, which we required. Also, because the YP maps are built from the standard UNIX databases, there are no provisions for allowing multiple users to modify the database simultaneously.

It may be possible to build utilities which provide these capabilities on top of YP, but we discovered several problems with YP which make it undesirable. First, using YP means significantly increasing the amount of network traffic generated by each machine on the network. Because database accesses are now turned into network accesses, even a simple **ls -l** command generates a large amount of traffic. A related problem involves the updating of YP maps on secondary (slave) servers: whenever a map is updated, the *entire map* is transferred over the network to each slave server. Thus, one user changing his password can result in several kilobytes of network traffic. Still another problem comes from the fact that although databases are maintained by YP, some of them must still reside on each machine as plain text files. These files must be updated on a regular basis in order to avoid becoming out of date with respect to the YP maps. Perhaps the most serious problem though, is that if the YP servers become unreachable for any reason (the server machine crashes, the network routing becomes confused, etc.), any program which uses YP will hang. This would not be a problem if only rarely used commands were affected, but a YP server failure affects even such simple commands as **ls**. Thus, the failure of a single machine could easily affect hundreds of others. For these reasons, Yellow Pages was rejected as a solution to our problem.

Athena Service Management System

The Service Management System (Moira) [2], developed by M.I.T.'s Project Athena, maintains a centralized database of user information, as well as other data, using Ingres from Relational Technology, Inc. A Moira process reads the database each night and generates password files, group files, and so on. These generated files are then transferred to the hosts they correspond to, and installed by another process after several consistency and validity checks have been made. Moira, as of February, 1988, handled 15,000 accounts and 1,000 machines.

Moira also meets most of the requirements listed in Section 1: it centralizes data storage, is flexible in the data that can be stored, provides automated account installation (but not complete account deletion), is machine and vendor independent, and allows the database to be updated simultaneously by multiple users. The system can recover from crashes, but this is done by loading ''plain text'' backup files (made nightly) back into the database (which brings it current with the last backup), and then by manually consulting log files and redoing any transactions which took place since the last backup.

The problems which we found with Moira were not as serious as those found with Yellow Pages, but were still serious enough to force us to reject Moira as a solution to our problem. One problem with Moira is that because it generates files on one machine and then propagates these files to their proper locations, machines can become out of date if they are down during the transfer. This is remedied by running the transfer several times during the day (checks are made to avoid repeated updates), but this is still not completely desirable. A related problem is that transferring complete files across the network generates a lot of excess network traffic, although much of this traffic is only generated late at night. Finally, the problem which most turned us against Moira was a financial one. Moira uses the Ingres database system from Relational Technology, Inc. This database system is very expensive, as are others such as Unify and Oracle, which could also be used. An experiment with ''University'' Ingres convinced us that for performance reasons, a commercial database product is required to implement Moira.

Asmodeus

Asmodeus [1], developed at Oregon State University, maintains a database of all user information, and provides access to that database through a series of *database daemons*. All changes to the database are made through these database daemons, as well as all database accesses. When a change is made to some part of the database, the database daemon sends commands to one or more *activity daemons*, which run on each network machine. These activity daemons then make changes in the password file, group file, or whatever. For example, when a user changes his password, a command is sent to a database daemon, which makes the change in the central database, and then sends a command to the activity daemon on the user's machine. The activity daemon then edits the regular UNIX password file, making the change complete. This approach has two primary advantages: changes take effect immediately rather than overnight, and programs which read the UNIX databases do not need to be modified since the use of the activity daemons preserves those files.

Unfortunately, the design of Asmodeus presents a few problems which, while not insurmountable, were in our view unacceptable. The most problematic part of Asmodeus is the use of multiple database daemons. Rather than having a single copy of the database, Asmodeus maintains multiple copies, with multiple database daemons. These daemons communicate among themselves in order to keep the multiple copies of the database current. In our view, the multiple daemons complicated matters by requiring database records to be locked before being changed (to prevent two different daemons from changing the same record), and so on. The algorithm for deciding when a database daemon was up or down was also deficient; a segmented network could easily result in two or more copies of the database being changed in different ways at the same time. Asmodeus uses a reliable datagram protocol, which we felt was more complex than necessary. By using a simple send-and-acknowledge scheme, the overhead of a reliable datagram protocol can be avoided. These problems in themselves would not have prevented us from opting to use Asmodeus rather than designing our own system. The database concurrency problems could be solved by running only a single database daemon, and the reliable datagram protocol could easily be removed.

Asmodeus is perhaps the closest match to what we wanted, and in fact much of ACMAINT's design is at least loosely based on it. If it hadn't been for the fact that, at the time we were looking for available systems, coding for Asmodeus had only just begun, ACMAINT would probably not exist today. Asmodeus fits all of the requirements outlined in Section 1, as well as making most of the ''nice to have'' features available or easily implementable.

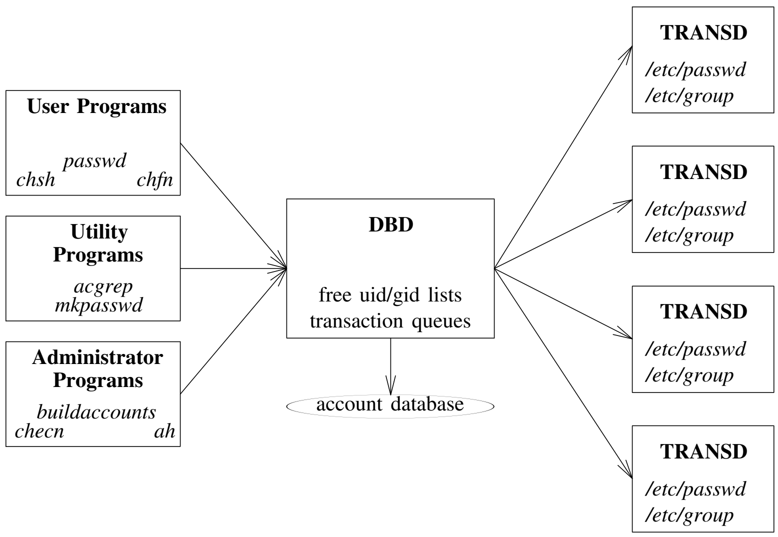


Figure 1 – Main Parts of the ACMAINT System

ACMAINT Design and Implementation

The design of ACMAINT satisfies both the primary requirements and the ''nice to have'' features of an account management system as described in Section 1. The storage of data about users, accounts, and groups is centralized on a single network host. A network server is provided to access the database, which allows changes to be made to the database from any host on the network. The use of other network servers, one on each network host, allows the installation and deletion of accounts, as well as the changing of account-related information, to be automated. ACMAINT's design divides these features into three main parts: the database daemon, the transaction daemon, and programs for users and system administrators. Figure 1 shows the relationship between these parts of the system.

The database daemon (DBD) is a network server which resides on the network host where the central database is stored. It is responsible for making all database retrievals, additions, deletions, and changes. The DBD accepts UDP datagrams containing database manipulation commands from other ACMAINT programs, acts on them, and returns the results in other datagrams. After performing any database updates required by a command it receives, the DBD constructs commands which are sent via UDP datagrams to the transaction daemons on the affected hosts, where changes to the appropriate UNIX database files are made in order to complete the command.

The transaction daemon (TRANSD) is a network server which resides on every network host under ACMAINT's control. It is responsible for installing and deleting accounts and groups, creating, moving, and deleting user's home directories, disabling or enabling accounts, and so on. The TRANSD accepts UDP datagrams containing commands sent from the DBD, and makes changes to the files residing on its host. For example, when a command to create a new account arrives, the TRANSD will add the appropriate line to */etc/passwd*, create the user's home directory, change its mode to the default mode for new user directories, and change the owner of the directory to the user. An indication of the success or failure of these operations is then returned to the DBD.

There is an important distinction to make between the DBD and TRANSD in the manner that they process data. The DBD understands only the *relationships* between the various pieces of data it handles. For example, it knows that when a new account is created on some host, ''something'' must be done on that host in order to actually install the new account, but is has no idea what that ''something'' may be. The TRANSD, on the other hand, has full understanding of the procedures involved in installing an account, changing a password, and so on. This distinction is important, because it allows ACMAINT to control hosts running any number of different UNIX variants. Because the DBD does not need to know exactly how to install an account, it does not need to know whether a host uses simply */etc/passwd*, the Berkeley *passwd.dir* and *passwd.pag* files, or even some ''shadow'' password file scheme. The TRANSD on each of these hosts will know the procedures required to install an account on that machine.

User-level account-related programs interface to the ACMAINT system via the DBD. Any program which modifies one of the account-related UNIX databases, such as **passwd**, **chfn**, **chsh**, etc., has been modified to instead send commands to the DBD. The DBD accepts these commands, modifies the database, and then sends commands to the TRANSD on the user's host. The TRANSD then modifies the file(s) which were previously modified directly by the program run by the user.

Similarly, new tools have been written for the system administrator that allow accounts to be created, deleted, and changed. The principal program used for all this is **ah** (Account Handler), which can function either in an interactive fashion or by reading ''batch'' files of commands. **ah** is described in more detail in a later section

Database Format

The ACMAINT database is implemented using a version of the Berkeley **ndbm** [5] library which has been modified to allow storage of larger records. The **ndbm** routines are not called directly; ACMAINT library routines which encode and decode database records to and from the format used in the database are used instead. This organization allows the easy substitution of some other database mechanism, such as the **gdbm** library from the GNU project or a commercial database management system.

There are five record types stored in the ACMAINT database. Three of these record types are ''primary records,'' that is, they pertain to data which will be reflected in the UNIX database files. The other two record types are ''secondary'' record types, and are created only as a side effect of creating primary records. Secondary records are used to allow information to be retrieved using other keys. In all primary records, the name of the system administrator who created the record and the time the record was created are maintained to provide rudimentary auditing. Primary records also contain an ''uninterpreted data'' field which is stored as a list of data separated by semicolons. This field is not interpreted by the DBD or TRANSD, but can be used by other ACMAINT programs which read the database and create permissions files and the like from information stored here.

The *user record* is a primary record which stores information that is unique to a user, but constant across all accounts the user may have. It is stored in the database using the user's login name as the key. There is one user record in the database for each person with an account on any of the systems managed by ACMAINT. Creation of a user record does not cause any UNIX database files to be modified. The user record contains:

- Numeric user id
- Login name
- Student id number
- Full name
- Office room number
- Office phone
- Home phone
- Mailbox address
- Group memberships
- Departmental affiliations
- Creation time
- Created by
- Uninterpreted data

The *account record* is a primary record which stores the information about a user which is not constant across all the user's accounts. It is stored in the database using the key ''*login@host*''. There is one account record for each account a user has. Creation of an account record causes a command to be sent to *host*'s TRANSD requesting that an account be created. The account record contains:

- Host name
- Login name
- Login group id
- Password (encrypted)
- Classification
- Courses
- Authorizing department
- Authorizing person
- Expiration date (mo/yr)
- Login shell
- Home directory
- Creation time
- Created by
- Uninterpreted data

The *group record* is a primary record which stores information about a group. It is stored in the database using the key ''*groupname*%group''; this allows group names to be the same as login names where necessary. Creation of a group record causes a command to be sent to all TRANSDs requesting that the group be created. The group record contains:

- Numeric group id
- Group name
- Password (encrypted)
- Authorizing person
- Number of members
- Creation time
- Created by
- Uninterpreted data

The *host record* is a secondary record which is used to maintain a list of all hosts a user has an account on. It is stored in the database using the key ''*login*%hostlist''. The contents of this record can be used to generate the keys required to retrieve all of a user's account records. This record is modified whenever an account record is created or deleted.

The *student id record* is a secondary record which is used to allow a user's login name to be determined given his student id number. It is stored in the database using the key ''*studentid*%sid''. This record is used primarily to allow detection of duplicate accounts for the same student, which ACMAINT will not permit (a local policy which was designed into the software).

The Database Daemon

When the DBD begins execution, it clears its controlling terminal, creates a UDP socket at a well-known port to receive packets, and opens the database. Next, the on-disk queue of unacknowledged packets (see below) is loaded so that the packets can be retransmitted. Finally, the DBD goes into an infinite loop of receiving and processing packets, and retransmitting packets which were not acknowledged.

When a packet is received, it is first examined to insure that it was sent by an authorized sender. This check presently consists only of making sure the packet came from a privileged port on the sending machine. Assuming the packet is legitimate, it is then processed. If the packet came from a host which the DBD thinks is ''down,'' the host is recorded as now being ''up.'' If the packet is an acknowledgement packet sent by a TRANSD, it is removed from the in-memory and on-disk queues of unacknowledged packets. If the packet contains a command, control passes to the function responsible for implementing that command.

A command is processed by retrieving the necessary data from the database and if necessary, modifying the data and storing it back in the database. A command packet is then constructed for each TRANSD affected by the command (if any), and queued for transmission to that TRANSD after any other pending traffic has been sent. The packet is also stored in a queue of unacknowledged packets for possible retransmission. Next, an acknowledgement packet (or error packet, if the command failed) is constructed and sent back to the sender of the command. The acknowledgement packet may contain data; for example, the packet containing the results of a ''fetch'' operation also serves as an acknowledgement of the command. The acknowledgement packet is stored in a queue of ''recent'' acknowledgements, in order to allow detection of duplicate command packets. If another command packet arrives which is a duplicate of this one (i.e., the client program retransmitted it because no acknowledgement was received), the acknowledgement packet is simply retransmitted, without performing the command a second time. This is important, since many commands such as deleting an account are not repeatable.[2]

As mentioned above, unacknowledged packets are stored in a queue for later retransmission. This queue is maintained in memory for immediate use by the DBD, and also on disk to preserve state across machine or DBD crashes. Each time through the main receive/retransmit loop, if there are packets to be retransmitted, the DBD retransmits one packet. When a packet is retransmitted, the host it is sent to is marked as ''down'' by the DBD. If more than 20 minutes have passed since the host was marked ''down,'' or if a new packet is received from the host, it is marked ''up'' again. This enables timely updates to hosts which have crashed without flooding the network with packets destined for hosts which are not responding. Packets are retransmitted in a simple round-robin fashion, although this algorithm can easily be changed if delays between retransmissions become too large.

The commands shown in Figure 2 may be sent to the DBD by an ACMAINT client program. Some commands, such as ''create_group'', allow some fields to be specified as ''*'', which instructs the DBD to select a unique value for that field. Any command which uses the syntax ''login@machine'' will allow the machine to be specified as ''*'', indicating that this command should be performed on all machines on which the user has an account.

[2]Commands are generally ''benignly'' non-repeatable: an accidental reissuance of a non-repeatable command will result in a bogus error message, but will not cause any permanent damage.

The Transaction Daemon

The TRANSD is started via **inetd** [6], in order to conserve resources on machines without a lot of memory (e.g. the Sun 3/50). When a packet is received, it is first checked for authenticity. This check currently consists of making sure the packet was sent from the host and port used by the DBD. The only type of packet a TRANSD expects to receive is a command packet. Other packet types (errors, acknowledgements, etc.) are logged and discarded. If the packet is a command packet, control is transferred to the function which implements that command.

A command is processed by modifying the appropriate UNIX database files (*/etc/passwd*, */etc/group*, etc.) and by creating or removing any necessary files and directories. An acknowledgement packet (or error packet, if the command failed) is then constructed and sent back to the DBD. The acknowledgement packet is stored in a queue of ''recent'' acknowledgements, in order to allow detection of duplicate command packets. If another command packet arrives which is a duplicate of this one (i.e., the DBD retransmitted it because no acknowledgement was received), the acknowledgement packet is simply retransmitted, without performing the command a second time. This is important, since the command may not be repeatable (e.g., changing a login name).

```
add_group gname login                          enable login[@machine]
change_acct login@machine field value          fetch_acct login@machine [field]
change_group gname field value                 fetch_group gname [field]
change_user login field value                  fetch_hosts login
delete_acct login@machine                      fetch_user login [field]
delete_group gname                             message login[@machine] message
delete_user login                              remove_group gname login
disable login[@machine] message                unmessage login[@machine]
create_acct login:machine:gid:passwd:classif:authdept:authorizer:\
            expdate:shell:homedir:courselist:uninterp
create_group gid or *:gname:passwd:authorizer:uninterp
create_user uid or *:login or *:sid:fullname:office:offphone:homephone:\
            mailbox:grouplist:affiliationlist:uninterp
```

Figure 2 – DBD Commands

```
add_group gname login                  delete_group gname
change_acct login field value          disable login message
change_group gname field value         enable login
create_acct password-file-line         message login message
create_group group-file-line           remove_group gname login
delete_acct login                      unmessage login
```

Figure 3 – DBD Commands to the TRANSD

The commands shown in Figure 3 may be sent to the TRANSD by the DBD. In general, these commands correspond to the DBD commands of the same name; only the arguments to the commands are different.

ACMAINT Client Programs

A large number of client programs have been written or planned in order to allow system administrators and ordinary users to access and change the information maintained by ACMAINT.

The Account Handler Program

The *account handler* program, **ah**, is the major ACMAINT interface for the system administrator. Using **ah**, the system administrator has access to all DBD commands. Commands can be executed interactively via prompts, or ''batch'' scripts can be created (either manually or by other programs) and used as input. The present interactive interface is line-oriented, although a screen- or window-oriented interface could easily be written if desired. The current version of **ah** is actually just an exceedingly clever shell script with a couple of ''helper'' programs for sending packets and verifying field contents.

One of the most useful features of **ah** is the *.ahrc* file. This file allows the system administrator to specify default values for ACMAINT database record fields so that they do not need to be specified for every command. For example, appropriate defaults can be set for the user's department affiliation, classification, and expiration date. Default values can be overridden simply by specifying a value for the field in question. The *.ahrc* file also allows the system administrator to define arbitrary variables which can then be substituted into **ah** commands using the shell's ''$'' syntax. For example, a variable can be defined which contains the names of all clients of a specifc file server, and then a single command can be used to add an account to all of those hosts.

Other System Administrator Programs

Other ACMAINT system administrator programs which have been written so far include a program which reads a password file and translates it into ACMAINT database records (used for placing new hosts under ACMAINT's control), a program which reads the database and generates a password file for any host in the database, and a similar program which reads the database and generates a group file.

Several other programs are under development or planned for the future. These include:

- **acgrep**, a program which allows the database to be searched using regular expressions, logical expressions, and relational expressions. For example, the database can be searched for all accounts on host ''harbor'' that expired on or before May, 1989 and print a list of login names:

```
acgrep -host harbor && \
        -expdate <= 0589 -print login
```

- a program to read a tape provided by the Registrar which contains the names, student id numbers, and graduation dates of all enrolled students and generates an **ah** batch script to create accounts for all new students and disable accounts for students who are no longer enrolled,
- a program to search the ACMAINT database for accounts which are about to expire and place messages on those accounts telling the user that the account will be disabled and deleted once it expires (this can be a shell script built on top of **acgrep**),
- a program which searches the database for accounts which have expired and generates both a shell script to place the files owned by those accounts onto tape and an **ah** batch script to delete the accounts (this can also be a shell script built on top of **acgrep**),
- **buildaccounts**, a front end to **ah** which has a screen-oriented interface and extensive error checking,
- **lahbel**, a program which reads the ACMAINT log files and generates ''sticky labels'' containing the login names and passwords of newly created accounts. These labels are then stuck to the forms which explain how to use the system, where to find documentation, etc. and given to the users.

The design of ACMAINT is extremely flexible and allows the development of all sorts of utility programs. As our experience with ACMAINT grows, other ideas for useful programs are sure to come up.

Modifications to User Programs

User-level programs which modify UNIX databases (**passwd**, **chfn**, and **chsh**) have been modified (or simply rewritten) to converse instead with the DBD. These modifications essentially involve replacing all the code involved in modifying the UNIX database file with about 25 to 50 lines of code calling various ACMAINT library functions to send and receive packets to and from the DBD. The wildcarding allowed by ACMAINT on database ''change'' commands has also allowed us to add a ''network wide'' option to these commands. By specifying the ''–n'' option to **passwd** or **chsh** (''–n'' is implicit for **chfn**), the user can request that the information be changed on all of his or her accounts.

Life With ACMAINT

ACMAINT was initially brought up for testing in April, 1989 and controlled approximately 200 hosts. As of August, 1989 it controlled all hosts in the Engineering Computer Network, a total of over 300 machines. It is regularly used by seven system administrators, as well as by users changing their passwords, etc. The DBD is presently running on a Sun 4/280 server with 32MB of memory and is providing very good response time. The database consumes about 22MB of disk space, which averages out to approximately 75KB per host maintained. The TRANSD is running on VAX-11/780s and CCI Power 6/32s under 4.3BSD, Gould NP-1s and PN9080s under UTX/32 2.0, and Sun 3/50s, 3/60s, 3/180s, 4/280s, and 4/390s under SunOS 4.0.3. At present, we are using ACMAINT Version 1.4.

Our system administrators are generally pleased with ACMAINT, although they are anxious to have the programs mentioned in the previous section. They report that the time required to install new accounts (the most time consuming task at the start of a semester) has been cut in half, the time required to move a user's directory from one file system to another has been reduced to almost nothing (simply use **ah** to change the home directory in the account record, and let the TRANSD move the directory), and the account deletion process has been greatly simplified since the whole procedure is now automated.

It is expected that as more auxiliary ACMAINT software is developed, the amount of time spent by the system administrator on account-related tasks will decrease even more. The number of mistakes made in installing accounts (forgetting to change the ownership of the home directory, typos in the password file, etc.) has already been greatly reduced due to the automation and error checking provided by ACMAINT.

Portability

ACMAINT was written with portability in mind, since all programs except the DBD must work on a wide variety of systems. At present, no attempt has been made to port it to any versions of UNIX not based on 4.2 or 4.3BSD. However, provided a Berkeley socket interface is available, porting should not prove difficult. Care has been taken to make ACMAINT as configurable as possible, so that it can work on as many different systems as possible without writing more code.

Porting the TRANSD to a new variant of UNIX involves modifying the code which implements each TRANSD command. This includes the proper procedures to update the password file, create an account, modify the group file, and so on. Most systems should be similar enough to the Berkeley or Sun variants that little modification will actually be necessary.

Making modifications to **passwd** and other programs is probably the most problematic part of porting ACMAINT. This is because source code is required to make the modifications, but most sites have binary-only systems. In the future, we hope to provide ''generic'' ACMAINT implementations of these commands which will be suitable for use on most systems.

Problems Encountered

Even after extensive testing before putting ACMAINT into use, we were unfortunate enough to run into several problems, mostly centered around synchronizing machines which had gotten out of date somehow.

Initially, getting error messages from a TRANSD back to the user (going through the DBD) was difficult, and many messages got lost. This has since been fixed by having the DBD keep more state information about commands it has sent to TRANSDs. In the next version of the system, error messages will be sent back to the user via electronic mail.

Another problem became noticeable once a large number of machines were placed under ACMAINT's control: because the DBD sent commands to the TRANSDs immediately, some ''network wide'' commands (e.g. changing the password for ''root'') would cause the DBD to become unresponsive for several minutes until all the packets were sent. This problem was solved by sending acknowledgements for these commands after the modification of the database but before all the TRANSDs had been contacted. (This is safe, since all TRANSDs must still acknowledge completion of the command to the DBD.)

Problems have also started to arise with the round-robin retransmission scheme used by the DBD. The algorithm works, but because there are so many machines under ACMAINT's control, delays between updates are starting to grow to unacceptable lengths. This will be fixed in the next version of the system by splitting the single retransmission queue into separate queues for each host. The whole back-off/retry scheme presently in use also needs a major overhaul, and will probably be fixed by going to some sort of per-host bounded exponential back-off scheme.

Future Directions

In the next version of ACMAINT (Version 2.0), we plan to make several changes and additions, including:

- Kerberos [3], from M.I.T.'s Project Athena, will be added for authentication between client programs and the DBD, and between the DBD and TRANSDs.
- Support in the TRANSD for the Berkeley ''shadow'' password file mechanism, which will be in the next version of Berkeley UNIX.
- The functions of the DBD will be split and provided by three cooperating processes:

DBD Talks to clients and manages the database. Writes a single work record to disk, and signals the BURSTD.

BURSTD Separates a work record written by the

DBD into individual packets and places them into multiple packet queue directories, and then signals the WIRED.

WIRED Handles communications from the DBD to the remote TRANSDs. Each remote machine has its own packet queue directory, and transmit speed is limited by the rate at which the remote machine can complete requests. A bounded exponential back-off scheme is used for retransmissions. This provides higher throughput and lower memory usage. Splitting the DBD functions into three processes allows the DBD to respond to user requests rapidly, leaving the time-consuming tasks of splitting work requests into packets for the TRANSDs and sending the packets to the remote machines to other processes.

Availability

When it is completed, we plan to make ACMAINT Version 2.0 available to the UNIX community at large via anonymous FTP and UUCP. It will also be posted to one of the USENET source groups. This will probably occur sometime in the fourth quarter of 1990.

We also have been carrying on discussions with Berkeley about including ACMAINT with the next release of Berkeley UNIX (following 4.3-Reno).

References

[1] M. E. Epstein, C. Vandetta, and J. Sechrest, ''Asmodeus: A Daemon Servant for the System Administrator,'' in *USENIX Conference Proceedings*, San Francisco, Summer, 1988, pp. 377-391.

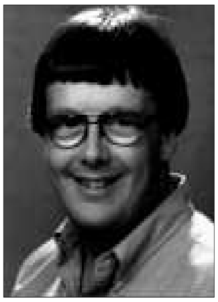
[2] M. A. Rosenstein, D. E. Geer, Jr., and P. J. Levine, ''The Athena Service Management System,'' in *USENIX Conference Proceedings*, Dallas, Winter, 1988, pp. 203-211.

[3] J. G. Steiner, C. Neuman, J. I. Schiller, ''Kerberos: An Authentication Service for Open Network Systems,'' in *USENIX Conference Proceedings*, Dallas, Winter, 1988, pp. 191-203.

[4] Sun Microsystems, *System and Network Administration*, Part No. 800-1733-10, May, 1988, pp. 349-387.

[5] *UNIX Programmer's Reference Manual, 4.3 Berkeley Software Distribution*, Computer Systems Research Group, Computer Science Division, Department of Electrical Engineering and Computer Science, University of California, Berkeley, April, 1986.

[6] *UNIX System Manager's Manual, 4.3 Berkeley Software Distribution*, Computer Systems Research Group, Computer Science Division, Department of Electrical Engineering and Computer Science, University of California, Berkeley, April, 1986.



Dave Curry discovered ''adventure'' on a PDP-11 running UNIX V6 in 1977 while still in high school. Since then he has worked for Purdue University, the Research Institute for Advanced Computer Science, and SRI International. He is the author of *Using C on the UNIX System* from O'Reilly & Associates, and recently wrote the popular white paper, *Improving the Security of Your UNIX System*. Dave can be reached at SRI International, 333 Ravenswood Avenue, Menlo Park, CA 94025 or electronically at davy@itstd.sri.com .

Sam Kimery has been involved with Unix since 1980. Prior to his employment at Purdue University, Sam was a systems programmer at Ford Aerospace, and also served time as a consultant. He has been employed as a programmer with the Engineering Computer Network at Purdue University since 1987. Sam can be reached at Purdue University, Electrical Engineering Building, West Lafayette, IN 47907, or electronically at kimery@ecn.purdue.edu.

Kent C. De La Croix recieved his Bachelor of Science degree in Computer Science from Purdue University in 1979. He has worked as a systems admistrator and computer programmer. He has been employed as a programmer with the Engineering Computer Network at Purdue University since 1984. Kent can be reached at Purdue University, Electrical Engineering Building, West Lafayette, IN 47907, or electronically at kcd@ecn.purdue.edu .

Jeff Schwab was exposed to UNIX V6 on a PDP-11 in 1975. He survived long enough to complete an M.S. in Computer Science at Purdue University. After stints at both IBM and Dec, Jeff is now back at Purdue as the Network Services Manager for the Engineering Computer Network. Jeff can be reached at Purdue University, Electrical Engineering Building, West Lafayette, IN 47907, or electronically at jrs@ecn.purdue.edu.