The Automation of the Process of Computer Account Application and Processing

Edsel Adap

May 1994

1 Introduction

Our generation is entering the age of information. This is the time when communication is being revolutionized. Computers are playing a big role, in this revolution. As a result, they are being used more and more by more and more people. The number of computer users is steadily rising. In Andrews University alone, four years ago, there was approximately six hundred users on the Computer Science, Physics, and Engineering Departments' computer systems. Four years ago this computer network consisted of four unix workstations and about ten to fifteen terminals. Today that has increased to ten unix work stations and over 200 terminals in those three departments. Last year there was approximately one thousand user accounts. This year that figure has risen to over fifteen hundred. However, the number of of System Administrators through these four years did not increase. Four years ago there were two faculty and two students functioning as System Administrators. This year we have one faculty and three students functioning as System Administrators.

It is not surprising that the amount of work that the System Administrators do is increasing. It is the dream of most System Administrators to fully automate every repetitive and time consuming aspect of his or her job. Automating such time consuming and repetitive jobs will give the System Administrator more time to work on innovations and improvements for the system instead of continually fighting an uphill battle of maintenance and repairs to keep the system running.

One such time consuming and repetitive task is the Processing of Computer Account Request Forms. Every individual who wants to use the resources of a computer network must have a computer account on the network – in the case of this project the network or computer system under discussion is the Andrews University's Computer Science Department Unix computer network. This limited scope of discussion, however, does not limit the usefulness of this prject to this system. This project can be ported to work on the other Unix machines on campus run by the Computing Center and Academic Computing department.

In this paper I will discuss the problems involved in applying for computer accounts at Andrews University. I will also describe the process that the System Administrators (in the Computer Science Department of Andrews University) go through to process the account request forms. I will explain in fair detail the mechanism which I designed to eliminate the problems encountered with the traditional (pen and paper, filling out forms, getting signatures, etc...) method of applying for computer accounts and the processing of these applications.

Since this project is specific for the computing structure of Andrews University, I will use the term "system or network administrator" to refer to those of the Computer Science Department of Andrews University. The term "system" or "network" or "computer system" will correspond to the system or computer network run, administered and supported by the Computer Science department. In the future it may include the other systems on campus.

2 The Problem

Traditionally, when one applies for a computer account, one would acquire a computer account request form and fill in the necessary information. This information includes their name, ID number, address, telephone, major, school, department, whether they are staff, administration, Andrews student or a guest, and what they plan to use the system for. It is also necessary for them to acquire their advisor's signature. One may ask, why is all this information needed? Most of the information required is to aid the system administrators in deciding in which partition or directory their account will reside. For example, one who is a computer science major will be given an account where there is a relatively large amount of disk space. Engineering students will be given an account where their files reside in the Engineering department's file server. Physics students will have accounts that reside on the Physics department's computer. All other user accounts go into a smaller disk partition since they are not expected to collect large amounts of data in their use of the system. Although to the user the location of their files is irrelevant and transparent, to the system administrators it provides a method of organizing the file structure and allows a more efficient use of the resources. The advisor's signature is necessary in order to verify that the student is currently registered and is taking classes at Andrews University.

This traditional pen and paper system has worked well for several years. However it has become a chore for the system administrator to enter anywhere between one hundred to two hundred accounts in the first two weeks of each quarter as students register for computer classes. What makes this chore so tedious? To create an account the system administrator must assign each user a unique login name, a unique numerical user ID, a numerical group ID, a password, a home directory, and a shell. After all of these are assigned and assembled in a passwd entry, it must be entered into the Network Information Services (NIS) passwd database. The directories in which files necessary and used by this account must be created and the necessary files which sets up default parameters about the behavior of the account must be copied.

Assigning a unique login name could be made very easy by using the Andrews ID as the login name. However, many see the ID number as private information and that the number with the matching name should not be made readily available to the public. Since login names together with the real names are freely available for anyone to look at once the user is logged in, this method cannot be used. It was then decided that last names or some form of it will be used as login names. In case of duplicate last names, letters from the first name will be appended to it to form a unique login name. With over fifteen hundred accounts, it becomes difficult to make sure that the login name chosen is unique. In addition to that, there are other systems, such as Redwood (the administrative system) to which some of the system administrators of the computer science department do not have access. This makes it difficult for the system administrator to ensure that no login names will be the same on both systems. It is necessary that the accounts added to the computer science system does not have a duplicate login name on Redwood, since mail is delivered by a machine operated by the computer science system to Redwood. Having duplicate logins on the two systems would cause mail to be delivered incorrectly.

A unique numerical user id (uid) is necessary. This allows the computer to identify the person using the system. It is used to prevent users from accessing files owned by other users. Every file a user owns on the system is marked by this uid. To choose a uid, one simply has to take the greatest uid and add one to that number. However there may be some exceptions, which makes it necessary to look for duplicate uid's after choosing one. If duplicates are found then another uid must be chosen.

Choosing a group id (gid) is based on whether the applicant is a university faculty, staff, administrator, guest or student. If the applicant is a student then the applicant's major, department, or school is used to determine their gid from a database of gid's. This is not as difficult as finding unique login names and uid's since there are currently only one hundred and seventy-six groups known to the system. Nevertheless, it still becomes tedious when entering a large number of accounts at a time.

An initial password needs to be assigned to the account to prevent the casual intruder from obtaining and using an illegally acquired account. The password must be encrypted. There are several ways we can choose a password for a new account. We can choose a different password for each new account. The problem with using this method is that it becomes very difficult to concoct a unique password for every new account to be added. If we choose one password for all the new accounts, we are opening up the system to security problems. Once a person is able to break into a new account, that person has effectively broken into all the other new accounts. No matter which method we use, the system administrator must encrypt the password. It has been the practice of the CS system administrators to use "1st.pass" as the initial password for all new accounts. This is easy since we can simply encrypt the password once and use that encrypted form for all the new accounts. Unfortunately this poses a security risk. By discovering one encrypted form of the password, once a cracker breaks into one new account, he or she has effectively broken into all the new accounts. Alternatively we can encrypt "1st.pass" every time for each new account. Unix provides an encryption mechanism which may give several encrypted forms of one password. For example, one encryption of "1st.pass" is "WwRVZ7CPot4MY". Another form is "tnh9kgMgQ2p2." There are 4096 unique encrypted representations of every password. To provide a unique representation of a password the system administrator must run the encryption program as many times as there are accounts to be added. When adding a set of fifty accounts at one time, this process becomes very tedious, time consuming and error prone.

The process of deciding where a user's home directory will reside is similar to deciding the gid. The home directory is based on information about the major, department, school or status (whether they are a registered student or not, or whether they are a staff or faculty member) of the applicant. After determining the home directory for a user a shell is given to the user. This is usually "/bin/tcsh" for unix accounts, and "/bin/passwd" for PC accounts. A finished password file entry would look like this:

```
doej:tnh9kgMgQ2p2.:2766:114:John Doe, Example
Account:/homes/cis4/student/1994-2/doej:
/bin/tcsh
```

The fields are arranged as follows:

```
login:password:uid:gid:name:directory:shell
```

Once the password entry has been created, it must be entered into the NIS passwd database. After this, the home directory specified in the passwd entry must be created. Its ownership must be changed to match the uid or login name of the user. All the necessary files that sets up the default behavior of the account must then be copied. Among these files are ".login", ".cshrc", the .elm directory and the .elm/elmrc file. Once all of this is done, the information about the account regarding login name, group id, and the date processed must be written on the account request form to be kept in a file. This adds another task of filing forms. It may seem trivial but with large batches it becomes cumbersome and error prone.

Entering one of these entries once in a while is not difficult at all. It becomes very tedious when one has to enter them in large batches. Manually entering this raises the probability of errors. The login name may sometimes be chosen such as it is not unique. The encrypted password may be typed in incorrectly and since the encrypted form is irreversible (it cannot be easily decrypted) it becomes very frustrating for the user since they cannot log in. The uid assigned may have duplicates. Sometimes the system administrator forgets to create the home directory. A host of other typographical errors could complicate the matter.

Another problem of using paper in applying for accounts is that sometimes the hand writing on the form is unreadable. This either results in a very long delay in the creation of the account or in many cases the Account Request form is thrown away.

We have seen the problems involved in this method from the system administrator's point of view. Problems also exist from the applicant's point of view. There have been many cases when the applicant cannot find his or her advisor to obtain a signature. This causes delays since the administrators refuse to enter accounts without the advisor's signature. The account request forms sometimes get lost on its way to the system administrators. Some of the lab consultants on rare occasions misplace it or the system administrators themselves could sometimes lose it. This creates a sense of frustration not only on the System Administrators part but also on the user.

Another problem is how to notify the user when the account is created and what the password for the account is. Using the telephone is not a very attractive option especially when entering large batches of accounts. It is time consuming and not always successful. Using the postal system to notify the users of the creating of their account can become expensive on stamps which makes it an undesirable solution.

3 The Solution

A system must be devised which would automate the process described above. This system must acquire all the necessary information from the applicant, and perform a variety of verifications. This system must be able to verify if the applicant is a current student at Andrews University or a staff member. If the applicant is neither a current registered student nor a staff member then he or she must be denied a regular account. Alumni, non students and non staff members should apply for guest accounts using this system. This system must also be able to verify to some extent if the applicants are who they claim to be. It must be able to check the NIDA records and match the names with the ID number provided by the applicant. Of course the system can only check those who have entries in the NIDA records. Some guests may not have entries in NIDA. Consequently their identity must be checked by the System Administrators manually. This system also needs to check if the applicant already has an account. If an account already exists, the application must be denied. If the applicant has already applied for an account but the account has not yet been activated it must also be denied.

This system must also be able to make decisions with regards to the uid, gid, home directory and shell of the applicant and keep records of applications. It needs to be able to take the information about the applicant's major, school, department, or whether the individual is a guest or a student or staff, and choose a gid and a home directory based on that.

Flexibility is a very important consideration when creating packages like these. It must be able to accommodate changes in the system without the need to rebuild the programs. For example, when new majors are added, there must be a way to notify the program about this without having to rewrite part of it. When there is a change in the structure of the file systems, the program must be able to accommodate those changes when choosing home directories for users.

It must not lose any of the functionality of the pen and paper method. It must not take the control of who does and who does not get an account from the system administrator. It is necessary that the system administrator be the one who makes the final decision about who gets an account and who does not. It must be easy to use. After all, the whole idea behind the automating scheme is to make it easier and more convenient for the applicant to apply for an account and to reduce the work of the system administrator without taking control away from him. It must also keep accurate records of when an account request was filed and when it was processed.

I have designed the mechanism that will accomplish these requirements. In the following sections, I will be describing the process involved in creating this system. I will also describe in fair detail the internals of this mechanism.

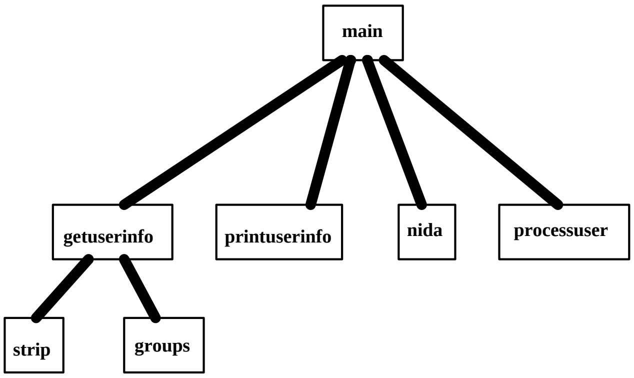

Figure 1: Top Level Structure

4 Development of the Mechanism

In the summer of 1993, during one of our Technical Support Staff meetings, the idea of a system that will make applying for accounts less cumbersome had come up. Mr. Dan Turk, Assistant Professor of Computer Information Science suggested that we should create a system that will allow students to apply for accounts on the computer. Shortly after that, I decided to look around in the internet to find public domain software packages that will allow us to do this. I searched various usenet newsgroups, particularly the comp.sys.sun.admin [3] and comp.unix.solaris [4] newsgroups, asked several mailing lists, particularly the Sun Managers' Mailing list [5], but was unable to find anything that did this and that would be easy to modify to work in our computing environment. Although I did not find any programs that do what I need, what I did find is source code for a password encryption algorithm. At that time I did not know that I would be using this for this project. I did not originally plan to include password selection in this project. That came as a later development.

I started writing code for this system in September 1993. The first module I wrote was the main program. All it did was call several functions: *getuserinfo*, *printuserinfo*, *nida* and *processuser*. Not all of those functions were operational at that time. It was not until March of 1994 that all the those functions actually did something. The basic structure of this program can be seen in figures 1, 2 and 3.

Getuserinfo is the function that collects all the information needed for creating the account. *Printuserinfo* is a function that prints out the information collected and gives the applicant a chance to verify the information entered and to re-enter the information if any mistakes are present. *Nida* is the function which does identity and status checks. The *processuser* function processes the application request.

The *getuserinfo* function collects all the information needed for the creation of an account. This function asks the applicant for their name, address, telephone number and AU ID number. The telephone number and address fields are only used for the purpose of keeping records. The *groups* function is called to present a menu of majors and departments, and their corresponding gid. The user is asked to enter a *gid* number. *Getuserinfo* also asks the user whether a PC only account is desired (no access to unix, but provides e-mail) or a UNIX account. It also asks whether the applicant is a staff, undergraduate or graduate student. This question is not asked if the user enters *guest* as their AU ID. All this information is stored in a structure called *userinfo*. *(Please see the listing of the file adduser.h at the source code section of this paper for the complete description of the structure).*

Getuserinfo asks the applicant what the intended use will be. This information is stored in the *use* member of the *userinfo* structure. The user is asked if the account will be used for word-processing, spreadsheets, databases, e-mail/gopher, remote login/telnet, programming, or other. Each one of these uses have a corresponding binary value. If the user replies with a 'yes' to any of these the corresponding bit is set to 1, otherwise it is set to 0. The resulting value is then *OR*ed with the current value of *userinfo.use*. Any combination of these will produce a unique 7-bit integer. The values assigned to *userinfo.use* is shown on Table 1. This information is used to decide if the applicant needs an account or not. If an account is needed, this information is used to decided what kind of account is needed – Unix, PC, or both. If the applicant does not indicate what the account will be used for, the value of *userinfo.use* will be 0 and the application procedure terminates. The applicant will not be given an account. The meanings of the other possible values of *userinfo.use* are discussed in the discussing of the *make_pwent* function.

USAGE\BIT#	6	5	4	3	2	1	0
Word Processing	0	0	0	0	0	0	1
Spreadsheets	0	0	0	0	0	1	0
Databases	0	0	0	0	1	0	0
E-Mail/Gopher	0	0	0	1	0	0	0
Remote Login	0	0	1	0	0	0	0
Programming	0	1	0	0	0	0	0
Other	1	0	0	0	0	0	0

Table 1: userinfo.use values

Once all the information about the user has been acquired, it is then necessary to check for the applicant's identity and NIDA status. All NIDA information is stored on Redwood, the Administrative Computer System. This system has restricted access and all the NIDA information is inaccessible to most people. How can the program check NIDA without having access to this information?

Aldy Hernandez, currently a Computer Science student here at Andrews University works for the Computing Center. He has written a program that runs on a TCP port on Redwood. This program accepts any AU ID number and returns the matching name, address, telephone number, and the status of the individual, whether they are a student, staff member or currently inactive or un registered. What I needed is a function that will communicate with the one on Redwood. I wrote the *nida* function to accomplish this. The *nida* function opens a TCP/IP socket connection to Redwood. The source code for this function is largely based on *Unix Network Programming* [8] by W. Richard Stevens and *Internetworking with TCP/IP, Volume III* [2] by Douglas L. Comer and David L. Stevens. This function sends the AU ID number of the applicant to Redwood and receives the student information such as name, address, telephone and status (student/staff/registered/un-registered) from Redwood. A lot of experimentation was necessary to determine the exact format of this output coming from the server program on Redwood. It was necessary to determine that format in order for the *nida* function to be able to extract the name and status fields. I found out that the output of the program on Redwood does not have constant field lengths and is not always terminated by the same character. This made it difficult for me to separate the fields. After many trials I have determined some field lengths that work well enough that the returned string of characters are split into letter groups that I can interpret. For example, the output of the program on Redwood my be the character string "IN-ACTIVE." Now due to the way the fields are divided, I may not have the whole word in the field after I have received it. Therefore I only scan for the words "IN-ACT" or "TIVE" and when these words are detected I can conclude that the applicant is not an active student. I found that field lengths of thirty-one for the first line, twenty-four for the second, thirty-one for the third, two for the fourth field, followed by three fields of length thirty-one, plus two terminating characters, then a field length of twenty-two, four fields of thirty one, followed by the two terminating characters. This is followed by three fields of thirty-five, followed by the 2 terminating characters, then a field of three junk characters, a field of twenty-seven characters, another field of three, followed by one of thirty-one characters seemed to give me a satisfactory field divisions which allows me to accurately parse the information. *(See the source code listing for nida.c at the end of this paper)*

The *nida* function extracts the first and last name fields and ignores the middle name or initial from the information provided by Redwood. It combines these two and compares them to the first name, last name pair provided by the applicant. If a match is found, the program continues otherwise an error message is printed and the program terminates. The status if the applicant, whether a student, staff member, registered or un-registered is returned as a numeric value. A value of 1 is returned if the person is a student, 2 if a staff member, 3 if inactive. If the person is registered for the current quarter then 10 is added to those numbers. Only values of 2, 10, 11, 12, and 13 are allowed to have an account. Others are rejected. Note that the number 13 does not have any meaning and so it should never occur.

The *nida* server program on Redwood is not guaranteed to be running at all times. Built into this function is a mechanism which detects when the server program is not running. Once this situation is detected the program notifies the applicant that the application cannot proceed and to try again later.

Once the user passes all the validity checks in the *nida* function, the *main* function calls the *processuser* function. This function checks the Network Information Services (NIS) password map to determine if the user already has an account. This involves reading every single entry in the password file and looking for a first name, last name match. If a match is found, the applicant is notified of the situation and the program terminates. If the no match is found, the queue of accounts to be processed is checked for a last name, first name match. This is necessary to prevent people from making multiple applications. Since the account application is not immediately processed, instead it is stored

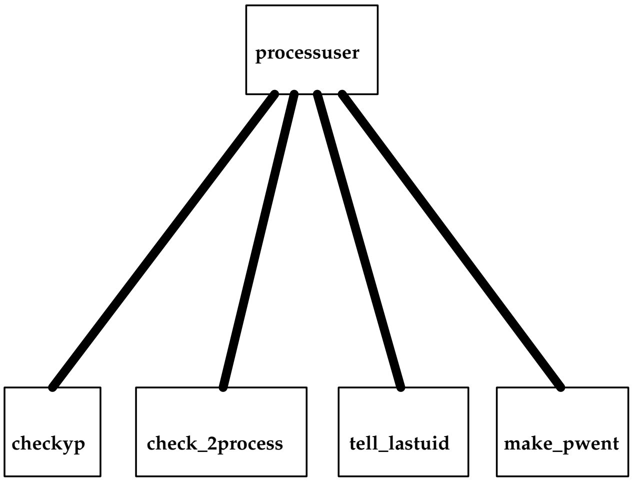

Figure 2: Processuser function structure

in a queue of accounts to be processed, it is necessary to check this queue to see if the person has already applied but has not yet been processed. While checking the NIS password map, it is scanned for the next available uid. This is done by taking the maximum uid below 9000 (9000 and above is reserved for the accounts added by the Computing Center), and adding 1 to it. This method alone of finding a uid, is not sufficient as it is guaranteed that multiple instances of this program running simultaneously will produce the same uid. This uid then must be checked against the queue of accounts waiting to be processed. If the uid is not found there, then it must be available, otherwise a new id is used. The program then needs to tell any other instances of this program that it has chosen a uid and that the other instances cannot use it. A file called ˜ *newuser/lastuid* is checked to see if the current uid has already been allocated by another instance of the program. If it is already used, a new one is calculated by adding one to the current uid, and this new number is written to the *lastuid* file, if the number currently in that file is not the same number, in order to prevent other instances from use this number.

We now have the name, uid and gid necessary for the password file entry. We still need a unique login name, a home directory and a shell before we can create the password entry. The function *make_pwent* is called by *processuser* to accomplish this task.

The login name is primarily determined by the applicant's last name. The last name is converted to all lowercase letters. The password files of both the CS systems and Redwood needs to be checked if this login has already been used. To check the CS systems' password file, we simply use the *getpwnam()* library function and give it the login name. If this function returns the value NULL this means that the login name does not yet exist and can be used. Checking the password file on Redwood is a little trickier. This is again due to that system's inaccessibility.

To accomplish this I used the Simple Mail Transport Protocol (SMTP). SMTP delivers mail by the login name. I can simply issue a *vrfy* (verify) command to the SMTP daemon and give it a login name. If the login name exists in the system, it returns the name of the user, otherwise it replies with a "user unknown." All that was necessary was to write another function similar to the *nida* function. What this function needs to do is to open a TCP/IP connection to Redwood's SMTP port. Then send the command *"vrfy login"* where login is the login name that needs to be checked. This function then needs to read the output from the SMTP port. This output is a one line output so it makes it relatively easy to parse. If the output had the word "unknown" in it, that means the login name is not used on Redwood and that it can be used for a login name. If an "unknown" was not returned, a name must have been returned. My function (called *redwood_mailcheck*) read this name, and compared it with the applicant's name. If they matched, then the user on Redwood is the same as the applicant and so we can use the same login name. If the names do not match, a new login name must be selected and the checks are repeated until a suitable login name is found.

If the first choice of a login name (the last name) is already used, it is modified by appending letters from the first name to the last name, starting from the first until all the letters of the first name are exhausted. Since the login name has a limit of 8 characters, it is sometimes necessary to delete characters from the last name and replace them with letters from the first name. The complete algorithm for choosing login name follows:

```
STEP 1:  login <-- last name; x=1;
STEP 2:  is login used on the CS systems or
         on Redwood?
         if 'NO' use the current login name;
         we are DONE.
STEP 3:  Have we exhausted all the characters
         in the first name?
         if 'YES' cannot choose a login name.
         print an error message and quit.
STEP 4:  is login name is less than 8
             characters?
         if 'YES' then copy the next letter
             of the first name to the end of
             the login name.  Go to STEP 2
STEP 5:  delete x characters from the login
             name and append x characters
             from the first name to the
             login name.
         x <-- x + 1; Go to STEP 2
```

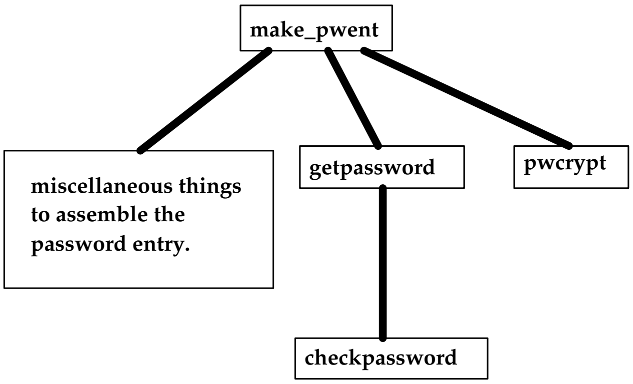

Figure 3: Makepwent function structure

Make_pwent decides what kind of account (PC only or UNIX) the applicant needs. This decision is based on the information given by the user about his desired use of the system. This information is stored in the variable *userinfo.use*. If the user does not plan to use any of the Unix specific uses such as remote login, programming, and the internet, then the account is marked as "PC Account NEEDED." Otherwise it is marked with "UNIX Account NEEDED" or "UNIX & PC Account NEEDED" depending on the specified use. This however does not override the applicants choice of an account. If the applicant specified that he or she wants a PC only account and the program decided the user may need UNIX access, the application is marked with "PC Account DESIRED" and "UNIX and PC NEEDED." When the user claims to need a UNIX account but the usage entered implies that only a PC account is needed then the account is marked as "UNIX Account DESIRED" and "PC Account Needed." The purpose of this is to give the System Administrator the final decision with regards to what kind of account the user should have. If the value of the variable *userinfo.use* is less than 16 *(please refer to Table 1)*, then only a PC account is needed. Values of *userinfo.use* divisible by 16 and less than 64 indicates that a UNIX account is needed. All other values of *userinfo.use* indicates that both UNIX and PC accounts are needed.

Another important decision this module does is to decide whether a staff account or a student account is needed. The difference between the two is the location and permanence of each. Staff accounts are placed in a partition where disk space usage is only limited by the physical size of the disk partition. Student accounts go into partitions where there is currently a 4 Megabyte soft limit and a 5 Megabyte hard limit on disk usage. Staff accounts are also relatively permanent. Student accounts get expired at the end of each quarter. In deciding whether a staff account or student account is needed, the status returned from the *nida* function must agree with what the user input in the *getuserinfo* function. The user must choose "staff/faculty" when asked for their status and this must agree with what NIDA reports. This is done to make the system immune to people who claim to be staff when in fact they are not.

Based on the two pieces of information (staff or student status, and the kind of account wanted – PC or UNIX) determined above, *make_pwent* assigns home directories by calling the *gethome* function. For Unix accounts the *gethome* function uses the *gid* and the *gr_ug* (graduate or undergraduate) members of the *userinfo* structure to decided where the home directory of the user should reside. Currently this is all hard coded into the program. Future plans include making this part of the program flexible by storing this information in a text file which the system administrator can edit. Allowing the flexibility of changing the home directory structure without editing and recompiling the program. Table 2 summarizes the group to home directory allocation table. The directories shown in table 2 are appended to "/homes/." The rest of the home directory path components are assembled by *make_pwent* based on the current year and quarter and the login name. For example, *gethome* may return "/homes/csu/" for a user who is in the csis group (group 105) and is an undergraduate student. If the login name is "adap" and the year is currently 1994, and it is the spring quarter (2^{nd} quarter of 1994), then *make_pwent* will then add "1994-2/adap" to the home directory field. The completed home directory field will be /homes/csu/1994-2/adap."

Once *make_pwent* obtains the home directory, it calls the *getpassword* function. This function turns off the printing of characters to the screen (for obvious security reasons). I used the *termio()* and *ioctl()* library function calls in order to accomplish this [8]. The user is asked to enter a password twice in order to verify the first password entered. Until the first and second passwords match the user will be repeatedly asked to enter a password. *Getpassword* calls the *checkpassword* [6] function found in a version of the npasswd [7] library. This is done to make sure the password is not an easily guessed password. I needed to modify this version of npasswd to tell it not to check the NIS passwd file, since the password entry I was checking is not yet in the NIS passwd file. To substitute for this change, I made *getpasswd* check for any substring of the applicant's password entry which matches the password entered. If a matching substring is found, the applicant will be asked for a new password. Once the password passes all these checks, it calls on the function *pwcrypt* [1] to encrypt the password.

Once the encrypted password is obtained and given to *make_pwent*, all we need is the shell. PC accounts simply get "/bin/passwd" as their shell to allow them

GID	Undergrad	Graduate	Staff
csis	csu	csg	sbaf
sgcsis	csu	csg	sbaf
fcis			sbaf
sba	sbag/student	sbag/student	sbaf
acef	sbag/student	sbag/student	sbaf
mgtmkt	sbag/student	sbag/student	sbaf
engr	engr/student	engr/student	engr/faculty
physics	phys1/student	phys1/student	phys1/faculty
bcisd	cis4/bcisd		cis4/bcisd
acad	sbgu/guest		
faculty			cis5/staff
others	cis4/student	cis4/student	

Table 2: GID to Home Directory Mapping

to change passwords while denying them UNIX access. UNIX accounts get "/bin/tcsh" as their shell. We now have all the necessary information to create the password entry: login name, password, uid, gid, real name, home directory, and shell. Assembling the password entry is trivially done by concatenating the fields together separated by colons. Once the password entry has been created, all this information is assembled into a message that will be sent to the System Administrator via electronic mail. This message includes all the data in the *userinfo* structure and in addition to that all the other pieces of information that the computer decided on. Here is a sample message sent to the System Administrator:

```
ID#:      61238
Name:     Edsel R. Adap
Address:  239 Meier Hall, Berrien
Springs, MI 49104-0900
Tel:      (616)471-8488
Group ID: 105 csis
Undergraduate
STUDENT account needed
UNIX & PC NEEDED
UNIX and PC ACCOUNT DESIRED
Login Name: adap
adap:Tt8/Iwiqzrw4s:2773:105:Edsel R. Adap:
/homes/csu/1994-2/adap:/bin/tcsh
```

This message, except for the password entry, is also written to the file, *~ newuser/newlog*, for the sake of keeping records. The password entry is stored in a file called 2process for later processing by the System Administrator. The recipients of this message can be easily specified by editing the file *~ newuser/who* and placing the internet addresses of the recipients in that file. This flexibility is useful when the task of creating accounts is given to another individual. It does not require that the program be modified in any way. Simply change one file, and the new individual in charge of processing accounts will receive the messages.

Notice that the account is not actually created. Only the password entry is created and saved in a file, and the System Administrators notified. This gives the System Administrators a chance to look at the accounts to be processed and make whatever changes they feel necessary. A string consisting of the id number, name and login name fields separated by colons is appended to the file *(~ newuser/cso_file)*. This file is used to add entries in the *CSO* electronic phone book. Another string consisting of the login name, first name and last name fields separated by comas, is written to a separate file *(~ newuser/novell)* which is sent to the novell server for processing PC accounts. The contents of this file is only sent to the Novell System Administrators after the UNIX part of the account has been entered into the NIS passwd file. This is done in order to maintain consistency between the Novell and Unix systems.

I have written two shell scripts that can be used to process these applications. One is *~ newuser/bin/createacct*. It is used as a pipe. When the System Administrator reads his mail using elm, he can simply pipe the message that contains the application to *createacct*. *Createacct* will then extract the password entry information and add it to the NIS passwd file. It then calls another script *(/local/etc/acctnew.auto)* which creates the necessary home directory, and copy the necessary files to the new directory.

The other shell script is *~ newuser/bin/process*. *Process* reads the *~ newuser/2process* file and adds all the applications in the *2process* file to the NIS passwd file. This script is useful for processing a batch of accounts at one time.

Both of these scripts must be run on the *YP* master server. When these scripts are run, the contents of the *2process* file which were processed are removed from the file and appended to the end of the *~ newuser/processed* file with the current date. This way a record of when accounts were created can be kept.

To implement this system, I created a dummy account on the system with a login name of *newuser*. This account does not have a password in order to allow any student or staff of Andrews University to apply for an account even from a remote place in the world. The main program of this system, I called *adduser*, is used as a shell to this account. When someone logs in as newuser the program is immediately executed and the user can apply for an account.

5 Observed Improvements

Since the implementation of this system, I have found more time to do other System Administration and Support duties. In the first 2 weeks of this quarter, approximately 100 accounts have been added. Before the the implementation of this system it took approximately three to five minutes to create one account. This could amount to several hours of work. With this system, I simply look at the mail I receive and if I approve of the account I simply pipe the message to ˜ *newuser/bin/createacct* and everything is done automatically. It only takes a few seconds of my time to process each account. If there is a batch of them to process, it would only take as long as it takes for me to type "˜ newuser/bin/process" in order to process all the accounts. Again that is only a few seconds. Since the implementation of the system, I spend only a minute or two to process accounts each night.

The process of application for an account has been made easier in the applicants point of view. They get results immediately. They no longer need to chase their advisors. They already know their password and they are assured that their accounts are most likely to be active the next day.

Both applicants and system administrators no longer have to worry about lost application forms. The system administrator does not have to worry about logging or record keeping since it is all done automatically by the computer. Unreadable hand writing is no longer a problem. This system is also friendly to the environment as it reduces the amount of paper we use.

This system is by no means completely finished. There are still some flexibility issues that I need to consider and implement, such as home directory configuration. I would also like to see a version of this program which uses an X-Window based interface. A help screen would also be useful in providing guidance to the applicant on how to apply for accounts. I am continuing the development of this software and will continue to do so in the following months.

REFERENCES

[1] Unknown Author. Unix password encryption algorithm. Available via usenet Newsgroups: Comp.security.unix, and Comp.archives.

[2] Douglas E. Comer and David L. Stevens. *Internetworking with TCP/IP, BSD Socket Version*, volume III. Prentice Hall, Englewood Cliffs, New Jersey 07632, 1993.

[3] Various Contributors. Comp.sys.sun.admin. Usenet News, 1993–1994. This is a collection of usenet news articles from the comp.sys.sun.admin newsgroup.

[4] Various Contributors. Comp.unix.solaris. Usenet News, 1993–1994. This is a collection of usenet news articles from the comp.unix.solaris newsgroup.

[5] Various Contributors. Sun managers mailing list. E–mail, Internet, Usenet News, 1993–1994. This is a collection of mail from System Administrators on the internet.

[6] Clyde Hoover. Checkpasswd. Available via anonymous ftp from ftp.cc.utexas.edu://pub/npasswd, March 1990.

[7] Clyde Hoover. Npasswd. Available via anonymous ftp from ftp.cc.utexas.edu://pub/npasswd, March 1990.

[8] W. Richard Stevens. *Unix Network Programming*. Prentice Hall, Englewood Cliffs, New Jersey 07632, 1990.