Mkuser - or how we keep the usernames straight

Gretchen Phillips - SUNY@Buffalo
Ken Smith - SUNY@Buffalo

June 1989

1. Introduction

The State University of New York at Buffalo has an ever growing Unix based computing environment. The primary instructional Unix machines now include a VAX 11/785 and a Sperry 7000/40 both running Berkeley 4.3, as well as an Encore Multimax running Umax 4.2 Release 3.3. Additionally, Sun workstations now proliferate on the campus. Administration of any number of Unix machines becomes a headache when username administration comes into the picture. Additionally, with NFS filesystems, users must have consistent uids along with usernames (where consistency is for the convenience of the user).

The total number of unique usernames in the timesharing environment is over 1200 and workstation accounts number about 500. When Unix timesharing was the primary location for users, and system administration had only one point, management of usernames and uids was relatively straightforward. Initially, a simple c-shell script was sufficient for adding users (when we had only one machine). This grew to a C program that could take a batch file for processing instructional accounts and would check existence of duplicate usernames and uids on the other timesharing hosts. This system fell apart when workstations began to proliferate on the campus. There was more than one point for system administration; individual departments or schools hired system administrators, individual workstation owners could add and delete users, and although cooperation was desired, no tool was available to maintain usernames over the entire network in a consistent fashion. To this end, we developed our newest incantation of *mkuser*. It has a daemon uid and username server and database manipulator, as well as a client program for manipulating appropriate system and user files on hosts throughout the network.

2. Philosophy of *mkuser*

The philosophy of usernames, in general, isn't a complicated thing. People need a method of access to resources and some place to store their work. Managing this can be as simple as one username where everyone shares, cooperates and never makes a mistake, or as complicated as one (or more) username for each user. We settled on one username per user. Later, we additionally settled on one uid per user.

The original script we used for making users was simplistic at best and dangerous at worst. It simply prompted for necessary passwd fields, did some minimal checks and appended to /etc/passwd. The growth of our environment and expansion from one where

users were concentrated on a single machine to users spread over four machines (or more), meant that we had to come up with a more sophisticated method for maintaining accounts. Additionally, at the beginning of a semester it could be the case that three hundred new accounts needed to be created. Needless to say, this was an ugly task.

We developed a C program that would prompt for necessary information in an interactive fashion or read a batch file for massive account additions. One essential feature that we added was to have it check for username duplication on the other timesharing hosts. This was possible because all system administration was handled by a limited number of trusted people and the hosts were equivalenced. The reasons for this feature were two fold. It would be a bad thing if two different people ended up with the same username. It would be inconvenient if a single person ended up with two different usernames.

With the additional constraint of a consistent unified uid base and the proliferation of Sun workstations and system administrators on campus, we decided that we must develop some type of uid/username server. With this tool any person who generated a username could be confident that neither the username nor the uid conflicted with another on the network (either within a sub-domain or between domains). It is especially critical within a domain for the purposes of mail delivery when a domain spans several machines. The incantation became *mkuserd* (the server) and *mkuser* (the client).

3. Overview

The *mkuser* and *mkuserd* programs are basically straightforward. The client, *mkuser* collects information from the system administrator. This information can come from interactive or batch input. It then makes requests to the daemon, *mkuserd*, based on this information. *mkuserd* in turn responds to transaction requests and sends appropriate information back to the client. *mkuser* then uses this information in creating new accounts. Additionally, *mkuserd* stores information about the user in a centralized database.

4. *mkuserd*

The *mkuserd* program runs on one of our timesharing machines. It is the core of this network implementation. The functions of *mkuserd* include:

- determining if the user has an existing account
- determining if a username is in use
- providing unique uids
- collecting password files from client machines and updating data base
- maintaining useful information about the user

mkuserd can be described as an transaction processor. It takes transaction requests from *mkuser* clients, processes them and returns some information. If a client supplies a user

identifier, then if that unique identifier has a username associated with it, *mkuserd* will return the username and uid. If no username is associated with that unique identifier, *mkuserd* returns a null string indicating no username exists. If a client supplies a username, then *mkuserd* will determine if that username is in use. If it is, then *mkuserd* returns the full name of the user associated with it. Finally, *mkuserd* will return an unused uid when requested.

Access to this information is provided through hash tables based on username and unique identifier. This allows efficient access to the data through the two primary key areas. Unused uids are stored in a array and supplied to client programs as uid transaction requests are made.

4.1. *mkuserd* database

mkuserd keeps information on all accounts that are created and in fact all accounts in the client password files. In particular it keeps records that include:

- username
- user-id
- group-id
- unique identifier for each user
- user's full name
- department information
- expiration date
- machine name(s) where user has account

Whenever a *mkuser* client connects to *mkuserd*, *mkuserd* requests a copy of the current **/etc/passwd** file from the client host. It uses this information to update any account and uid information that may have been inserted or deleted "by hand" and reminds the *mkuser* client that additions should be done only by *mkuser*. With current password files, *mkuserd* keeps its internal database up to date. It checks that uids still have the same value that exist in the database. If inconsistencies are found, then it sends an informational message back to the client. The local system administrator is responsible for making the appropriate changes based on the messages sent by *mkuserd*.

4.2. *mkuserd* security

Security of the data is provided by assuring that the connection between server and client is run on a privileged port. This prevents random users from connecting to the port and obtaining any potentially sensitive data. Additionally, the transmission of the unique identifier is uni-directional. The client sends this to the server but the server never sends it back. This prevents data from being transmitted should the port be compromised.

4.3. *mkuserd* problems

mkuserd has some faults. It is a memory pig. Since *mkuserd* runs on a single host, if the host is inaccessible, then *mkuser* cannot be used to generate accounts. It is, however, possible to move the database and *mkuserd* to another host and then reconfigure client configuration files to know the new location of the server without recompiling the client

or server programs.

5. *mkuser* clients

The *mkuser* client program takes input describing users and connects to *mkuserd*. *mkuser* makes transaction requests from *mkuserd* and uses this information to create local accounts. It creates password file entries and home directories based on the username, uid and gid provided by *mkuserd*, as well as information stored in a local configuration file. The input to *mkuser* can be either interactive or batch. Typically, batch files are created from student registration data where student identifier, full name, course registration and university standing are recorded. These files are processed by *mkuser* and unique identifiers are passed to *mkuserd*. If that identifier is present in the database, the username is returned to *mkuser*. If no account exists, *mkuser* generates a username and then asks *mkuserd* to verify the uniqueness of it. Upon verification of uniqueness of the username, *mkuser* requests a uid from *mkuserd* for the username.

5.1. username generation

The generation of usernames on other SUNY@Buffalo machines (VMS and CMS) is based on the unique identifier. This results in usernames of the type v117fpl4 and c999il1t. We hoped to be more generous in our generation of usernames and settled on an algorithm of selecting the first unused username generated based on a combination of the first name, middle initial and last name of the user. This can be overridden with a runtime flag in the interactive mode that will prevent the *mkuser* client from generating a username and will instead prompt for one. This is especially handy when *mkuser* cannot generate a unique username using this algorithm and the system administrator must generate the name using one of those human brain based algorithms.

5.2. *mkuser* configuration

mkuser consults a local configuration file. Local system parameters, established by the system administrator, are stored in the *mkuser* configuration file. The configuration file is stored in /usr/local/adm/mkuser/mkuserd.conf. Configuration parameters stored here include:

- location of the server
- username prefix
- home directory location
- quota limits

These variables include the location of home directory and quota values. These are based on groups established by the system administrator. This permits *mkuser* parameters to be set by the local administrator. For example, one administrator may have all home directories in a flat directory, say /users, while another may choose to segregate into /users/faculty and /users/student. Default quota values can be set for different groups or invocations of *mkuser*. Local administrators can supply configuration parameters to the

username generation algorithm so that usernames can be generated with a departmental prefix. The prefix was not part of the original username generation scheme but was a compromise for those departments who wish to have some distinction among groups of users based on username.

Parameters are read at runtime so they may be changed as the system administrator deems necessary. This is handy when partitions get full and new users need to have home directories in different partitions.

5.3. *mkuser* expiration

In addition to adding users to a system, *mkuser* can be used to expire users. A runtime flag specifies that it is a expiration run, rather than addition and usernames are expired from the password file. Expiration dates are stored in the data base. An expiration run will deactivate all accounts that have passed their expiration date. This feature is critical in our environment where there are well defined boundaries on account activity. Expiration dates are only checked on an expiration run so accounts are only removed at the request of the administrator.

6. Conclusion

This system does not completely free system administrators of account administration. System administrators still have the responsibility of correcting any problems in their /etc/passwd files. They must still obtain registration lists to feed to the *mkuser* client. They must still run *mkuser* for adding accounts. It does relieve them (or some campus wide administrator) from having to maintain a list of available uids. It does relieve them of having to check to see if a user has an existing account. It does relieve them (for the most part) of having to use some algorithm in their brain for generating usernames. Finally, it does offer some method for expiring accounts based on expiration dates. In the overall view, *mkuser* and *mkuserd* can make the administration of a distributed environment, by a group of distributed administrators, a less frustrating task.