SAS® Software for the UNIX® Environment

John Carl Zeigler, SAS Institute Inc., Cary, NC
T. Randall Betancourt, SAS Institute Inc., Cary, NC
Chip Kelly, SAS Institute Inc., Cary, NC

ABSTRACT

The use of SAS® software as an application solution for the UNIX® environment is presented in three parts: an overview of UNIX operating systems, a discussion of the fundamentals of UNIX operating systems, and a tutorial on using SAS software in the UNIX environment.

INTRODUCTION TO UNIX OPERATING SYSTEMS

UNIX operating systems were the first whose language is not closely tied to the hardware environment in which it runs. This is the essence of a portable operating system and the primary reason for the increasing popularity of UNIX operating systems among workstation vendors. Workstation vendors can enter a portable operating system market without the costly overhead of developing a proprietary operating system.

The first implementation of the UNIX operating system was written in 1969 as part of a research project at AT&T Bell Labs by Ken Thompson and Dennis Ritchie. It was written in assembler language for a Digital Equipment Corporation (DEC) PDP-7. Frustration with an earlier effort at MIT to write a multi-user operating system called MULTICS led Thompson and Ritchie to this new effort. Shortly thereafter the UNIX operating system was moved to a PDP-11 and rewritten using B, a language developed by Thompson for this purpose. By the early 1970's the UNIX operating system had once again been rewritten in C, a language written by Ritchie. By the mid 1970s, AT&T's use of the UNIX operating systems had grown tremendously.

Growth in Usage

In addition to the rapid growth within AT&T, the use of UNIX operating systems spread rapidly among universities. This was a direct result of the unique technical relationship Bell Labs had established with universities during the initial stages of UNIX development. During this time, AT&T, under the terms of its 1956 Consent Decree, began to grapple with the issue of licensing software. Under the terms of the decree, the Bell System was required to provide telecommunications products to the various Bell Companies and to the nation, but nothing more. The transfer of software technology became a very complex legal issue within AT&T.

By 1974, AT&T was forced to establish nominal distribution fees to help offset the cost of licensing. By then, Bell Labs had produced several internal iterations of UNIX operating system. Versions 4 and 5 were distributed to external customers. Significantly, in 1974 AT&T received its first commercial request for a UNIX operating system. The debate within AT&T evolved into two camps: those who felt that the technology should be shared at a low cost to continue innovations and those who felt that AT&T should capitalize on the emerging commercial viability. The company chose to base the software price on market considerations and thus established its commercial interest in UNIX operating systems.

In 1976, Version 6 became widely available. In 1978, Version 7 was released. Version 7 is the last external distribution by the Bell Labs Research Group. (Version 8 was distributed internally.) Commercial distribution of UNIX operating systems began around 1977 with the release of the Programmer's Work Bench. In 1981, Bell Labs released UNIX System III, which combined some of the features of PWB/UNIX and Version 7. Momentum for AT&T's interest in UNIX operating systems increased as commercial products flourished. However, in January 1983, the United States Government announced its proposed Modified Final Judgement, which ultimately brought about the dissolution of the Bell System. AT&T's litigation with the U.S. government prevented System III from containing current technology. It was only after 1983, with AT&T's divestiture of the Bell System, that UNIX System V was announced. UNIX System V, complete with technical support and training, signaled AT&T's commitment to UNIX operating systems as viable commercial software.

UNIX System V features include better interprocess communication, messages, shared memory, and features taken from Berkeley Software Distribution (BSD) 4.3 (discussed in the next section), notably the vi editor. At this point, AT&T began a more aggressive campaign to market UNIX System V. In 1985, AT&T developed SVID, the external interface specification for UNIX System V.

Divergence

As stated earlier, schools and universities began to demonstrate an early interest in UNIX operating systems. They quickly began to be used in computer sciences departments nationwide. The simple structure of the UNIX operating systems provides an ideal environment for continued research and development of computing theory and modeling.

During the late 1970s, the University of California at Berkeley began working on version 32V. Version 32V was an early implementation for the VAX.™ When memory paging was added, it was renamed 3BSD. Earlier work at Berkeley produced extensions such as the vi text editor, the C shell, and the relational data base INGRESS. These efforts were conducted by Berkeley Computer System Research Group (CSRG), which produced UNIX releases 4.1, 4.2 and 4.3BSD. CSRG is funded by the Defense Advanced Research Projects (DARPA), part of the the Department of Defense. Release 4.1BSD was enormously popular in universities because of extensions such as the C shell, memory paging, vi text editor, job control, and networking. 4.2BSD allowed access to Ethernet and the Transmission Control Protocol, Interconnect Protocol (TCP/IP), a standardized communication protocol for multi-host environments. In 1986, 4.3BSD was released, featuring increased speed, overall reliability, and a reliable signal mechanism. Release 4.3BSD is the latest release of the UNIX operating systems from Berkeley.

Convergence

Within the past several years UNIX operating systems have become viable in the commercial world. Between the Berkeley and AT&T implementations there has been a divergence and thus a strong need for standards. Major software vendors need a standard interface to ensure application portability, thus saving development time and effort.

Forces Shaping the ANSI Standard

There are several forces shaping the standards efforts. These forces include the ANSI X3J11 C Programming Language committee, the IEEE P1003 Portable Operating Systems Interface (POSIX) committee, the System V Interface Definition, the Berkeley Software Distribution 4.3, and the X/Open Group. A brief description of each and the effect they have on the standards efforts follows.

ANSI X3J11 Programming C Standard Committee

The ANSI Programming Language C Standard Committee was formed to establish C language standards. The /usr/group standard was used as a framework for standardizing character handling, localization, I/O, string handling, date and time, and other general utilities. The formal approval for the ANSI Programming Language C Standard is expected by mid-1988.

IEEE P1003 Committee

Many members of /usr/group felt that not all standards issues had been addressed. Thus, in 1985 /usr/group Standards Committee merged with the newly formed IEEE P1003 Committee and adopted Draft 1 as the new IEEE standard called POSIX (Portable Operating System Interface for Computer Environments). POSIX is intended to define a language interface that can be implemented in a variety of environments. Such a tact takes into consideration not only System V, but other UNIX variants such as 4.3BSD, XENIX, and others. In order to represent the intent of /usr/group membership, /usr/group appointed a representative to the IEEE P1003 Committee. Also during this time, /usr/group formed the technical committee to address issues not part of the IEEE Standards proposal. Presently there are eight subcommittees of the technical committee. The following technical committees are working to provide proposed specifications to the IEEE:

Distributed File Systems	define a standard for transparent sharing of distributed files across POSIX conforming systems, including peer-to-peer protocols
Network Interface	define a Portable Network Interface to allow process communication in a media- and protocol-independent manner
Graphics	define the graphics primitive set as well as presentation technology such as windowing
Internationalization	define linguistic and national language requirements such as collation, encoding schemes, date and time, and expression and message handling
Security	define security features that conform to the Department of Defense Trusted Computer System Evaluation Criteria
Real-Time	define real-time standards for timing, synchronization, priority scheduling, and asynchronous event notification
Supercomputing	define supercomputing standards for batch processing, checkpoint and recovery, fast I/O, and Fortran development environments

UNIX System V Interface Definition Similar to /usr/group, SVID is an attempt to promote application portability. Unlike POSIX, SVID is a definition that is vendor-specific, notably to AT&T. The intent is to specify a particular user interface. Volume 1 of SVID specifies standards related to the base system and kernel extensions. Volume 2 defines utilities, software development support, systems administration, and terminal interface extensions.

Berkeley Software Distribution 4.3 While not actually a formalized standard, 4.3BSD has produced several technical innovations that have been incorporated into POSIX. As mentioned earlier, these innovations include the vi text editor, C shell, TCP/IP, reliable signals, and job control.

X/Open X/Open, a group of vendors promoting a standard, was formed in late 1984 by five major European computer manufacturers developing platforms similar to UNIX operating systems. This group was influenced by the work of both /usr/group and AT&T SVID. The charter of X/Open is more commercial because each manufacturer was interested in defining a set of standards that provided access to the growing number of UNIX software applications. Since its formation, X/Open has grown to include several U.S. computer manufacturers. Since several companies provide competing software products, X/Open has moved to an approach similar to the IEEE in order to become independent from vendors. X/Open is committed to the POSIX standard once it is complete.

FUNDAMENTALS OF UNIX OPERATING SYSTEMS

In order to discuss how the SAS System fits into the UNIX environment, the following is a brief outline of UNIX operating systems and what features particular to these systems were utilized in the development of the SAS System under UNIX operating systems.

Technical Background of UNIX Operating Systems

The Three Parts of UNIX Operating Systems The UNIX environment consists of three basic parts:

- file system
- kernel
- shell.

The *file system* is the disk data structure where all files and programs are stored; this is not discussed here except to note that a SAS data library is implemented as a set of files in a directory similar to Version 5 under VMS.

The *kernel* is the operating system proper. It manages all resources and allocates them to users as they are needed. The kernel performs services on behalf of programs through interfaces known as system calls. There are systems calls for opening files, getting the date, creating processes, and so on.

The *shell* is the command line interpreter. It finds and executes programs on behalf of the user by invoking the kernel, making the layer between the user and the kernel. There is nothing special about the shell in the sense that the kernel does not enforce which program the user takes as a shell. The standard shell or Bourne Shell is simply a program that knows how to start and control other programs.

I/O Redirection One of the most versatile concepts in UNIX operating systems is that of the software filter. A *filter* is a program that simply reads standard input, transforms the data somehow, and writes the result to standard output. In the UNIX environment, all programs have three files opened for them by the shell: the standard input, the standard output, and the standard error output. This frees filter programs from having to perform any special I/O operations to access files. They can simply perform the transformation. The shell has special syntax for attaching a file or device to the standard input or output of a program. By default, the standard I/O of any program is connected to your terminal. For example, the sort filter sorts its standard input and produces the sorted data on the standard output. If any errors occur, the error message is displayed on the standard error output. If you are typing in input to a command whose standard input is the terminal, you can signal the end of the data by typing CTRL-D (D):

```
$ sort
words
not
in
alphabetical
order
     ^D typed, not echoed
alphabetical
in
not
order
words
$
```

If the text you want to sort is stored in a file, then you can use I/O redirection to attach the standard input to that file. I/O redirection of the standard input is specified by the < character. In the example above, if the data are stored in the file unsorted.words, then you should type:

```
$ sort < unsorted.words
alphabetical
in
not
order
words
$
```

You can store the result in another file by redirecting the standard output with the > character:

```
$ sort < unsorted.words > sorted.words
```

The file sorted.words now contains the sorted list of words.

Files are known to programs as small integers called file descriptors. The file descriptor for the standard input is 0; for standard output, 1; for standard error, 2.

You can redirect the standard error output with the special shell syntax:

```
$ sort 2> error.messages
```

The file ERROR.MESSAGES contains any error messages produced by the SORT command.

In general, the Bourne Shell allows you to redirect I/O on any file discriptor with the syntax

```
d<file
```

or

```
d>file
```

where *d* is a digit representing the file descriptor you want to redirect.

All files opened on behalf of a program by the shell are called preopened files.

Pipes Another method of manipulating the standard I/O is to redirect the standard output of one command to the standard input of another. This is specified by separating the command with the vertical bar (|) character. For example, the who command lists on its standard output all the users who are logged in at the current time. To get a sorted list of these users, the user would give the following command:

```
$ who  |  sort
bach      tty2p3      Feb  4 09:00
bob       tty4p0      Feb  3 15:10
chopin    tty3p2      Feb  3 12:31
dan       tty2p0      Feb  4 19:53
tim       tty1p1      Feb  1 17:32
```

APPLICATION AREAS WHERE UNIX OPERATING SYSTEMS EXCEL

UNIX operating systems are designed to support an applications development environment. There are many application areas where the UNIX environment is particularly suitable:

- communications
- distributed computing
- decentralized computing
- departmental computing
- production prototyping.

Communications

UNIX operating systems have always been associated with computer communications. With the recent rise of local area networks, shared file systems, and remote login facilities, UNIX operating systems provide a cost-effective solution for applications requiring communications and data sharing over a wide geographical area.

Distributed Computing

The UNIX environment allows for data transfer and off-loading processes from main systems to satellite processors. For example, some SAS users use UNIX operating systems to offload the distribution of reports from mainframe computers where SAS software is used to process large data sets in production batch jobs.

Decentralized Computing

In the area of decentralized computing, UNIX operating systems allow data and reports to be communicated among several systems to coordinate the information processing effort.

Departmental Computing

Another advantage of the UNIX environment is its scalability. Because the UNIX operating systems are portable and run on a wide variety of architectures and machines, applications developed under the UNIX environment can be moved to larger machines as the need arises without suffering the tremendous software migration costs typically associated with such a venture. This is particularly important in departmental computing where changes in computing needs can occur more frequently than in the classic information center paradigm.

Production Prototyping

The UNIX operating system approach to computing is to provide a set of primitive tools, such as the filters, that can be combined to create more sophisticated tools and applications in a straightforward manner. This approach makes the UNIX environment particularly suitable for fast prototyping. *Fast prototyping* is a software engineering methodology in which design specifications are immediately rendered in software models that can be quickly developed to demonstrate or test the design. While these models may not have the performance or all the features required by the final design, fast prototyping allows design changes to be tried out in a small model before irrevocable decisions are made. Since UNIX operating systems are portable and scalable, prototypes do not need to be developed in production environments, freeing the production machine for important production applications. Because the ad hoc, improvisational nature of prototyping would have a detrimental affect on the production environment, this effort is best offloaded to smaller, more adaptable environments where changes to an information processing application can be tried out before affecting the production system.

HOW SAS SOFTWARE FITS INTO THE UNIX ENVIRONMENT

The SAS System complements UNIX operating systems and takes advantage of all the previously discussed features of the UNIX environment. Some special features of Release 6.03 of the SAS System under HP-UX can now be combined with UNIX, resulting in an excellent applications development environment.

PIPE Device in the FILENAME Statement

One feature added to the SAS System is the PIPE device, a new device type in the FILENAME statement. The PIPE device allows the user to associate a SAS fileref with the standard input or output of any arbitrary UNIX operating system command or program. The syntax of the pipe device is

FILENAME *fileref* **PIPE**"*any UNIX command*";

Input

If the fileref is used in an output context, then any data sent to the fileref are sent to the standard input of the command. The command is executed and reads all the data sent to the fileref in the SAS statement where the reference occurs. Any output produced by the command in either the standard output or the standard error output is copied into the SAS log (unless redirected). For example,

```
FILENAME LPRINT PIPE "lp -dc2700j2";
DATA A;
   FILE LPRINT;
   PUT 'Hello world!';
RUN;
```

causes any data sent to the fileref LPRINT to be sent to the UNIX line printer daemon for printing on the destination printer named c2700j2. The line printer daemon responds with a message giving the job number for the queued job that is sent to the log as

```
NOTE: Message received from the "lp -dc2700j2" command:
request id is c2700j2-234
```

Output

If a fileref defined with the PIPE device type is used in an input context, then the command is executed and any output it produces on the standard output is collected and sent to the SAS System through the fileref. The standard input of such a fileref is connected to the null file /dev/null (unless redirected). Any reads from the null file return the condition, and any writes are silently ignored.

For example, the ls command with the -s option prints the sizes (in disk blocks) and names of all files in the current directory. To read such a listing into a SAS data set, execute the following command

```
$ ls -s > listing
```

Then run a SAS DATA step program to convert the external file listing into a SAS data set. This has the disadvantage that the file listing appears in the listing itself, perhaps confusing the results. This can be avoided by using the PIPE device, as in

```
FILENAME LISTING PIPE "ls -s 2> /dev/null"
DATA INFO;
   INFILE LISTING;
   INPUT SIZE NAME $;
RUN;
```

Note that error messages are ignored by sending the standard error output to the null file.

Line Printer Daemon

In UNIX operating systems the task of managing the printer resources is given over to a task called the line printer daemon. A *daemon* is a process that manages system resources without the need for human intervention. Requests for printouts are sent to the line printer daemon via the LP command. The -DPRINTER option specifies that the output is to be sent to the destination printer.

The SAS System uses the PRINTER device type to specify that a fileref is to be associated with the line printer daemon. The statement

```
FILENAME PR PRINTER;
```

specifies that any data sent to the fileref PR are to be sent to the line printer daemon for queueing on the system default queue. A destination can be specified as

```
FILENAME PR PRINTER "c2700j2";
```

indicating that the data are to be sent to the line printer daemon for printing on the destination c2700j2. The LP options can be passed to the line printer daemon by adding them to the destination

```
FILENAME PR PRINTER "c2700j2  -oELT";
```

At the Institute, this specifies that the printer is to be set in landscape mode.

SASGACMD

The line printer daemon handles graphics hardcopy devices as well as printers. In order to allow graphics output to be sent to the line printer daemon, the SASGACMD access method has been added to SAS/GRAPH® software. In the GOPTIONS statement, SASGACMD can be used to specify that the graphics stream be sent to any arbitrary UNIX operating system command in a way similar to the PIPE device. The syntax is

GOPTIONS GACCESS="SASGACMD > *any UNIX command*";

This would most often be used to send the graphics stream to the line printer daemon, as in

```
GOPTIONS GACCESS="SASGACMD>lp -dhplj1 -ograph";
```

which specifies that the graphics stream be sent to a Hewlett-Packard LaserJet printer in graphics mode. This could be added to the DEVICES catalog to make it the permanent default. Particular printer destination names and options are site-specific.

Pre-opened Files

The SAS System makes use of the pre-opened input and output files by creating a fileref for any file already opened by the shell when the SAS System is invoked. The standard input, standard output, and standard error output are always opened by the shell and are given the special filerefs STDIN, STDOUT, and STDERR. Any other pre-opened file is given a fileref formed from its file descriptor. For example, invoking the SAS System with the command

```
$ sas < sales > data  3< other.data
```

causes four predefined filerefs to be available during the SAS session: STDIN is associated with the external file sales, STDOUT is associated with the external file data, STDERR is associated with the user's terminal, and the special fileref FILDES03 is associated with the external file other.data in input mode.

Shell Scripts

A new option has also been added to the SAS System, -STDIO, which allows the SAS System to use the standard I/O for source, listing, and log files. By invoking the system as

```
$ sas -stdin
```

the predefined filerefs, mentioned above are no longer available. Instead, the SAS System expects to receive SAS source statements from the standard input, produce the output on the standard output, and send the log to the standard error. This is designed to be used primarily in shell scripts or in controlling SAS output from another program.

For example, the shell script

```
#!/bin/sh

sas -stdin <<*EOF* > /dev/null 3> /dev/null

 libname perm '/usr/local/sasapps/perm';
 libname dayfix '/usr/local/sasapps/dayfix';
 libname util '/usr/local/sasapps/util';
 dm 'af c=util.toplevl.menu.program';
 endsas ;
*EOF*
```

invokes the SAS System and executes an AF application.

The <<*EOF* syntax is similar to redirection, except that the data are taken from the shell script itself up to the line containing *EOF*. In UNIX operating systems, this is called a *here document.*

This is how a canned menu-based SAS application might be set up so that users do not need to be aware of how to invoke the SAS System.

The SAS System can also be easily controlled from inside a C program using system calls of the standard UNIX operating system.

FUTURE ENHANCEMENTS

The future of the SAS System on UNIX operating systems includes supporting other machines in addition to the Hewlett-Packard 9000 systems. The SAS Display Manager System will also be expanded to take advantage of native windowing packages such as X windows.

Communications is another area of growth for the SAS System, which in the future will be able to communicate in a peer-to-peer fashion across a network. Enhanced SAS/SHARE® and similar products will be forthcoming.

REFERENCES

Lycklama, Heinz (1988), "Technical Activities Report," *CommUNIXations*, Vol. 7, No. 6, 2-9.

/usr/group (1987), *POSIX Explored*, Santa Clara, CA.

/usr/group (1987), *Your Guide to POSIX*, Santa Clara, CA.

Wilson, Otis (1985), "The Business Evolution of the UNIX System," *UNIX Review*, 47-51.

Zucker, Steven (1988), "Living Together," *UNIX Review*, Vol. 6, No.1, 44.

International Data Corporation (1987), "Personal Computer UNIX Market," *The UNIX Perspective*, Vol. 5, No. 10.