Effective seismic processing under UNIX

Mark C. Lane

ABSTRACT

The UNIX operating system is becoming more prevalent as a seismic processing environment. Tips for increasing productivity under a UNIX workstation system are presented.

INTRODUCTION

Workstations —small, powerful computers—are becoming more and more popular in the geophysical community. Many of these computers run the UNIX operating system. Although UNIX has enjoyed increasing popularity over at least the last decade, it is a new environment for many geophysicists. UNIX differs in many ways from the mainframe operating systems which many of us are accustomed to using. This paper describes some features of UNIX which will help new users to understand how the system operates and consequently to increase their productivity.

UNIX is famous for its **man** pages. These are good places to look for information about new commands or to check for special features of common commands. Typing **man <command_name>** will give a concise description of the command <command_name>. The command **man man** will give additional information about the man command itself.

System Configurations

Although UNIX is appearing on many new mainframes and supercomputers, this paper will deal with techniques which are more applicable to workstations and minicomputers, particularly in a networked environment. This is most likely to be the type of UNIX system geophysicists will be using now and in the near future. There are many ways to configure workstations. Although they can be used in isolation, it is usual to have several interconnected workstations with a connection to a mainframe, supercomputer or minicomputer. This connectivity is a relatively new concept for many users. Those accustomed to a single mainframe handling all tasks may find it beneficial to understand networked systems. This paper discusses single user, multiple users of a single computer, and multiple users on multiple connected computers. This mirrors a mental model progressing from a local (mine) to a global (ours) viewpoint of the system.

SINGLE USER SYSTEMS

A single user view of a system has one computer totally dedicated to one user. In this case the user has no concerns about contending with anyone else for resources; the user alone pays the price of his inefficiencies. A user may run several jobs

simultaneously, each competing for system resources. By providing the system with information about the importance of these jobs, the user can make more efficient use of the system. A good goal is the maintenance of rapid interactive response while still using the computer fully when the interactive job doesn't need the resources.

As an illustration, consider a user who starts a seismic processing job, then wants to edit a parameter file ('job deck' to us old-timers) to set up for another run. The editor will compete on equal footing with the processing job unless the system puts different constraints on these jobs. A good solution is to put the processing job in the **background** and to reduce its priority (be **nice**). This will make the editing much faster and still let the background job make full use of the system during pauses in editing (even between keystrokes). A quick way to do this uses the **nice** command and **output redirection**. For example, suppose an application is called 'myap' and it accepts the argument 'myarg'. The user could execute the command **nice +15 myap myarg > outfile.list &**. This will start the application with reduced priority (higher nice value). The nice value of 15 is reasonable for background processing jobs, although the system will accept a range of values. **man nice** should show the range for a given system. The '>' is the redirection operator; it will send the output which would normally appear on the screen into the file 'outfile.list'. The '&' will put the job into the background, immediately returning control of the computer to the user, who can then start editing.

For more complex commands or command sequences the **batch** command is appropriate. For example, suppose the user has a file called 'myjobs' which contains a list of commands, one per line. Each command may be preceded by **nice +15** to reduce its priority. When the command **batch myjobs** is executed, several things happen. The batch job is queued, then run in the background when system load permits. The commands in the file are then executed sequentially in the order they occur in the file 'myjobs'. This method is useful when jobs need to operate sequentially, passing data from one to the next. An additional feature is that command outputs which would usually be sent to the screen will be mailed to the user upon completion of all the jobs. **man mail** will give an overview of the mail system (often called **e-mail**, for electronic **mail**). The command **atq** will show the queue's contents, including job numbers. **atrm <job_number>** will cancel a job while it is still in the queue.

On the subject of canceling jobs it should be noted that, on occasion, a user may want to kill a job which is running. Generally this involves two steps: identifying the offending job and killing it. For jobs which were started and put into background using the & command, the **jobs** command will provide information, including job numbers. **kill %<nn>** ,where <nn> is one of these numbers, will kill the associated job. Note, however, that the 'jobs' command will not work if a user logs out and then in again. In that case **ps ugx** will show all the users jobs and give a number (**PID**) to identify the job. **kill -9 <PID>** will kill the job identified by the number <PID>.

The user who forgets to nice a job can **renice** it once it is running by using **renice +15 <PID>**, where <PID> is obtained as above, from the 'ps' command. Note that programs can be written to automatically 'nice' themselves. 'ps' can reveal the 'nice' values of running jobs.

MULTIPLE USERS ON A SINGLE SYSTEM

When many users share a system, the procedures discussed above apply. In addition, though, users can seriously affect each other's jobs. One user running a large job without reducing its priority can seriously decrease the performance of other users' editors or interactive processing. Because UNIX is a relatively rules-free system, user cooperation and tolerance is required to maximize system performance. A few jobs running at normal priority can effectively stop jobs which have a 'nice' of 15. A good guideline is to put long jobs in the background at reduced priority using 'batch' and to let interactive jobs run at normal priority.

As more users start jobs on a system, the available memory per user decreases. UNIX offers **virtual memory**, in which the system uses disk storage to simulate additional memory. This allows the system to continue operating, but at a considerable slowdown. Memory access times are generally quoted in nanoseconds while disk access times are quoted in milliseconds; worst case slowdowns on the order of 10 000 to 1 000 000 are possible. In these extreme cases the system is said to be **thrashing**. Working on a thrashing computer feels like typing on an electric typewriter without plugging it in! Because of this, it is often better to run large jobs sequentially rather than simultaneously. It is also possible for the system to run out of **swap space**, the portion of disk allotted to virtual memory, in which case jobs fail immediately. An **out of memory** error message is indicative of this situation. On the Geology/Geophysics workstation cluster at the U of C, workstations can generally run only one commercial seismic processing job at a time.

MULTIPLE USERS ON MULTIPLE CONNECTED SYSTEMS

With many users and many connected computers, the situation can get quite complex. Not only do users have to worry about other users' jobs on the system, but their own jobs on other systems. In this situation the main concern is to reduce the transfer of data over the network connecting the computers. Although these nets may be considered fast for many computing applications, seismic data can easily swamp a net. Typical local nets are of the broadcast type. This means that every computer receives all messages sent by every other computer. Thus, only one computer can transmit at a time, effectively blocking every other computer's communications. The best way to reduce this load is to process data on a computer which has a disk directly connected to that computer. This eliminates transferring data on the net, both on input and on output.

A worst case scene might involve a user on workstation A, running an interactive filtering program on computer B, which is accessing data on a disk connected to workstation A and displaying results back on A. Data must move from A to B then back to A in order to be displayed on A. Then the data would move from A to B for filtering, then back to A for storage. Even with only two computers and disks this method is highly inefficient. If the user logged onto workstation A and ran the program on A, then no network traffic would be involved. Although this situation may seem a bit contrived, it does occur. With windowing user interfaces, it is easy to lose track of one's location on the net.

An ideal situation would involve several users, each doing interactive processing on their own workstations, accessing data on a disk on that workstation,

with a low priority background job running on each workstation. Here each computer would be fully utilized, yet interactive processing would get preferential treatment. Although this may not be possible in practice, it represents a goal to strive for.

CONCLUSION

In a system of networked workstations running the UNIX operating system, it is important for users to be aware of how the system treats their jobs. This can prevent inadvertent and/or unnecessary contention for resources. A few simple functions can improve the system performance, giving priority to interactive jobs, yet running background jobs efficiently.