ABSTRACT
To optimize system performance and maximize productivity, UNIX performance and resource usage should be monitored. The monitoring is often done manually by sporadic checking using commands such as 'top', 'ps'. A utility has been developed using SAS that monitors and identifies extremely resource-consuming processes, and sends e-mail to notify the owner of the processes. The utility is automatically invoked by another SAS macro in Windows NT. With the regular monitoring in appropriate frequency, performance issues can be detected early, ensuring system efficiency and productivity. This paper presents a number of techniques in interacting SAS with UNIX while going through the major steps of monitoring and reporting of the utility. The techniques include reading from different UNIX command outputs, processing data from UNIX command outputs, taking advantage of information available from UNIX, and sending data dependent e-mail using UNIX mail facility through SAS data step.

Key Words: UNIX, monitoring, performance, top, filename, pipe, email

INTRODUCTION
In the statistical programming environment, there are scenarios where highly resource-consuming processes may occur. For instances, conducting statistical analysis, such as Fisher’s Exact Test, and sorting, merging, and manipulating large datasets are resource consuming. Infinite loops may occur when developing macro programs. It may also happen that an illegally terminated process or a UNIX session is abandoned yet keeps running and consuming resources. These processes need to be detected and cleaned in order to free resources for more productive uses. There are a number of system commands such as 'ps', 'psr', 'top', and 'w' available in UNIX for monitoring performance (Thacher, 2000). However, one needs to type the command manually, read the command output, and identify and report any abnormal findings. This monitoring process can be time-consuming and boring. Automating all the checking, reading, identifying and reporting would be ideal.

Most SAS programmers have enough knowledge about UNIX to work in the UNIX environment, but may not be strong enough to develop a UNIX performance monitoring utility using the UNIX script language, C or Perl. Does a SAS programmer need to learn another language before writing such a utility? The answer is no. Such a utility can be easily handled by SAS with a few common UNIX commands. The expected functionality of such a utility is data reading, data processing, and reporting or messaging, and SAS is well-known for its data reading and manipulation ability. SAS also provides ways to interact with UNIX. The FILENAME statement with the PIPE in SAS allows a programmer to take advantage of UNIX commands to obtain operating system information from UNIX command output. Examples of using the FILENAME statement with the PIPE can be found in SUG papers (LeBouton and Rice, 2000; Mao, 2001). However, UNIX command output is sometimes not presented in row and column format so reading the output can be quite challenging and special techniques may be needed in order to utilize the information from the output.

In this paper, two utility macros used to monitor the UNIX performance are introduced: %batchtop (scheduler) and %qtopps (monitor). The challenge and techniques of reading data from the UNIX command outputs, taking advantage of information available from UNIX, and using the mail/messaging functionality of SAS are discussed. The utility macros, along with all code presented in the paper, are developed with SAS V6.12 and tested with both SAS V6.12 and SAS V8 in the UNIX environment for %qtopps and in Windows NT for %batchtop.

UTILITY DESIGN
In order to execute monitoring regularly and automatically, the authors designed a SAS macro in Windows NT to invoke the monitoring part in UNIX in a desired frequency. In other words, the utility has two parts to do the work: scheduler and monitor. The overall design of the utility can be represented by the flow chart below.

---

Figure 1. Overall design of the UNIX performance monitor.
IMPLEMENTATION OF SCHEDULER

One way to schedule a SAS job in UNIX is to use UNIX crontab (access privilege may be required). Scheduling can also be achieved using a pure SAS approach. The SAS approach is a macro (%batchtop) in Windows NT that iterates repeatedly. In each macro iteration, a connection between NT and UNIX is established through SAS/CONNECT and the monitor %qtopps for UNIX performance monitoring is submitted remotely. The %batchtop program then goes into sleep (using the SLEEP function) for a period of time, and then another iteration starts.

IMPLEMENTATION OF MONITOR

The monitor involves several key steps such as reading UNIX command output, data processing, identifying 'problem' process, finding user's email address and sending email notice. Details and tips of each of those steps are discussed.

1. READ FROM COMMAND 'w'

Several system commands/tools are available for UNIX performance monitoring; this paper uses commands 'w' and 'top'. Processes with a long idling time are screened using the output of the 'w' command and processes consuming high system resources will be caught with the 'top' command.

We start with the output of command 'w' since it is easier and output of command 'top' will be discussed in the following section. Command 'w' with the h option ('w –h') presents a list of users and their processes in row and column format (Figure 2). The information provided in the order of columns is user ID, terminal type, login time and idling time, JCPU, PCPU, and process name as the last column. With combination of list input and column input, this output data can be easily handled by 'filename pipe'. Length of idling time is in cumulative hours; the idling time in number of days, hours, and minutes is calculated.

```sas
/* Reading in the output of command 'w'; 
filename wout pipe "w -h";
data woutput(drop=terminal logintm idle jcpu pcpu);
length userid terminal logintm idle $8;
infile wout truncover;
input userid $ terminal $ logintm $ @23 idle 30-36 jcpu $ 37-44
cpu $ 45-49 what $140.;
run;
```

2. READ FROM UNIX COMMAND 'top'

Command 'top' provides a list of the processes using the most CPU, CPU load average and process running time. The information is updated every few seconds. To get a snap shot of such information, one can use the command 'top –d1'. Unlike the 'w' command discussed in the previous section, the 'top –d1' command produces an output not completely in row and column format (Figure 3). What is even more complicated is that the output of 'top –d1' cannot even be read as shown in Figure 3 when reading through 'filename pipe' because there are lines with length of over 200 characters. This can be clearly shown if the output is directed to a file and the file is opened with the vi editor (Figure 4). Figure 4 shows that lines 1, 2 and 3 have over 200 characters. These lines cannot be handled by SAS V6.12, since the maximum length of character variable is 200. For SAS Version 8, variable length is not a problem; however the information is still not available when the whole line is read into a single variable since information such as the user ID, process ID and CPU usage are all stacked together. Facing such a challenge, our strategy is to read in the data first, and then break down a single variable to multiple variables to get useful information.

By taking a close look at Figure 4, a recurring pattern ‘^[B’ can be seen. With the help of an editor capable of displaying hex codes, such as UltraEdit, the ‘^[B’ has hex code ‘1B5B42’x. Thus comes the solution, i.e. to use the ‘1B5B42’ as the delimiter to read in the output of command 'top –d1'. The relevant code is on the next page.

Figure 2. Output of UNIX command 'w –h'.
After reading, the 'top' command output is now stored in SAS dataset, shown in Figure 5. The printout (Figure 5) is more like the screen output of command 'top' (Figure 3), but all information is contained in a single variable 'PRSLINE'. To make use of the data, the data need to be processed, i.e. 'PRSLINE' need to be broken into usable components.

Figure 3. Output of UNIX command 'top -d1' to the screen.

Figure 4. Output of UNIX command 'top -d1' generated from a remote submission from NT and directed to a file.
3. PROCESSING DATA FROM COMMAND ‘top’

In Figure 5, the first three lines contain information such as server name, day of week, date and time when ‘top’ command was executed, CPU load average and total number of processes, etc. Each line is processed individually by applying SAS functions such as INDEX, SCAN and SUBSTR. Retrieved information is stored in macro variables.

Starting line 16 in Figure 5, the data is in row and column format, though the current data has one whole line in one single variable. At this point, there are two ways to break down the variable PRSLINE to get process ID, user name, process running time and CPU percentage etc. One way is, again, to use SAS function(s). An alternative way is to write the data to a text file and read from text file back to a SAS dataset. When reading back using list input, columns such as process ID, user name, process running time and CPU percentage go into separate variables, and the information becomes useful. In order to tell whether a job started in the night or day, job starting date and time are derived by subtracting length of job running time from current date and time.

4. IDENTIFYING ‘PROBLEM’ PROCESS

Now all the information such as user ID, process ID, process status, length of process running time and CPU usage are available. With this information, the criteria of ‘problem’ processes can be applied, so resource-consuming processes can be identified. The following is an example of criteria for identifying a ‘problem’ process.

- Any job/process running for more than 2 days.
- Any job/process taking over 80% CPU for over 1 hour during a weekday.
- A night job consuming over 80% CPU, but not finished at 9:00 a.m. on the next weekday.
- Multiple sessions running by the same user with total 80% CPU consumption for over 1 hour on a weekday.

Data from ‘w’ command and from ‘top’ command are put together before applying the criteria. The implementation of the criteria can be something like this.

```sas
%*implementing company policy regarding CPU usage;
%*data toplist;
retain cnt 0;
%* to iniitizel macro var tot to zero;
call symput('tot', trim(left(cnt)));
set toplist;
/*extremely long process/job */
if (rday >=2)
/*day job*/
or (8*3600<=sttm<=16*3600 and (pctcpu > 80 or sumcpu >80) and runtime >=1*60*60)
/*night job */
or ((sttm>16*3600 or sttm<8*3600) and (pctcpu > 80 or sumcpu >80) and curtime > 9*3600);
%*count the total number of problem processes;
cnt +1;
call symput('user'||left(cnt),trim(userid));
call symput('tot', trim(left(cnt)));
runt;
```

The problem process is numbered and corresponding user’s ID is stored in macro variable ‘usern’.

5. FINDING USER’S EMAIL

After the identification of a ‘problem’ process, the next step should be reporting of the process. Shall we send the mail to the UNIX mail facility? It seems that few companies are using the
UNIX mail facility for email communication but rather use mail servers such as Lotus Notes, Outlook, etc. To report 'problem' processes to those mail servers, however, the user's email address is not yet available to SAS. Email addresses can be obtained in two ways. One way is to create a SAS dataset containing all UNIX user ID and their email address. However, updating the dataset is required once a new UNIX user is added. Maintenance on such a dataset is always a difficult task. In many companies, a user's email ID is a combination of the first and last name with a dot in between, and this information can be obtained from UNIX through the UNIX user ID. So the 2nd way to get a user's email address is to take advantage of the UNIX user account information (such as user's first name and last name) through the UNIX shell command `finger`. The utility presented in this paper uses this 2nd method. Again, `filename pipe` is used to read the output of UNIX command `finger userid`, where userid is obtained from the previous `top` command output. Using `filename pipe` to read the output of `finger` is much easier than reading the output of the `top` command. A typical output from `finger` is shown in Figure 6. By locating the 'login name' line, the user's login ID, first name, and last name can be easily obtained. The user's email address is concatenated by first name, a dot, and last name plus the email server name. The email address is stored in a macro variable, which is resolved in subsequent data step.

```
/users/qk60333/finger g100238
Login name: g100238  (messages off) In real life: Sam Mao
  Digs: K  
Directory: /users/qk600238  Shell: /usr/bin/bash
Or since Feb 21 10:55:59 on ttyp8 from 10.20.185.44x0.0
```

Figure 6. Output of UNIX command `finger`.

6. EMAIL MESSAGING

Finally, the email address is available together with all the process related information such as UNIX users ID, process ID, process status, percentage of CPU usage, process start time and length of process running time. This information is stored in a SAS dataset with one observation for each 'problem' process. Server names and CPU load average, etc., are available via macro variable. The detected 'problem' process is ready to be reported to the owner of the process and to the person who is responsible for monitoring the UNIX system performance.

The SAS system in UNIX sends all emails by using the UNIX mail server through two SAS-provided external shell scripts. By defining a filename with EMAIL device type, i.e. 'filename email', mail can be sent from a SAS data step. When monitoring UNIX performance, it is possible that one user may have more than one 'problem' process. In that case, one message for one 'problem' process will result in one user receiving multiple messages. Ideally, all 'problem' processes of one owner should be sent to the owner with a single e-mail. By applying conditional logic in the data step, one email is sent per user containing all observations ('problem' processes) the user has. The email suggests the process owner take appropriate actions regarding the 'problem' process, including killing the process if it is a run-away process. The UNIX command for killing the 'problem' process is provided in the email. An example of such command is `kill processID`, where processID is the ID of the 'problem' process. So typing the command exactly as it is provided in the email will help the user to kill the process easily. The following is the code. Figure 7 is a sample email message generated by the utility in production environment.

```
/* sending email to notify the owner of the CPU consuming process. Also a copy of email will be sent
to Application User Responsible (AUR) */
/*define file name, send a copy of mail to AUR*/
filename reports email 'sam.mao@quintiles.com';

data _null_;
  file reports;
  set toplist end=eof;
  *cleaning up and starting a new email*;
  if eof then put '!EM_ABORT!';
  else put '!EM_NEWMSG!';
  run;

** printing the email header **;
if first.userid then do;
  *Specifying receiver of the email*;
  put '!EM_TO!' emailad;
  * Copying email to AUR*;
  put '!EM_CC!' sam.mao@quintiles.com;
  *specifying email subject*;
  put '!EM_SUBJECT!' 'server usage remainder';
  put 'Dear ' fstname ': ';
  put ''; 
  put 'You have a process/job running in UNIX server ' &srvname ' . The process is consuming about '
  put pctcpu '% CPU';
  put xxxxxxx '/';
  put ' ';
  put '@1 User ID' @14 'Process ID' @70 'CPU_ID' @78 'CPU usage (%)' @94 'Time (min)';
end;

**Reporting the details of the problem **;
**process(es) **;
  put @1 userid @14 prsid @70 cpuid @78 pctcpu @94 rmins;
  put ' ';

** printing the email ending part **;
if last.userid then do;
  put 'To optimize overall system performance, it would be highly appreciated...'
  put ' 1. xxxx';
  put ' 2. xxxx';
  put ' 3. xxxx';
  put ' '; 
  put 'Sam';
  put ' ';
  *Sending the email*;
  put '!EM_SEND!';
end;

* Stopping sending email if no more data; 
if eof then put '!EM_ABORT!';
*cleaning up and starting a new email*;
else put '!EM_NEWMSG!';
run;
```
DISCUSSION

Two versions of this utility are developed. One is the remote version since its scheduling part is in NT side and monitoring part is remotely submitted. This is the version the paper discussion is based on. The other version is the crontab version, since its scheduler is the UNIX crontab. It is worth mentioning that the monitoring part of the crontab version is slightly different from that of the remote version. For example, the output of the `top` command when submitted directly from UNIX is a single long line (with length of over 1000) instead of multiple lines as shown in Figure 4. The same technique can be used to read such output, i.e. directing the output to file, finding out the recurring part, using the recurring part as delimiter.

CONCLUSION

`filename pipe` and `filename email` together with other techniques play a central role in developing this utility. By automating the process of monitoring and messaging with this utility, the task of monitoring is greatly simplified. This is especially true in terms of the automated messaging between the person who monitors the system and the owner of the 'problem' process. With proper frequency of monitoring, performance issue can be detected promptly and appropriate measures can be taken to maintain an efficient production environment.

REFERENCES


ACKNOWLEDGEMENTS:

The authors would like to thank Dave Avila for setting up UNIX crontab and John Tan for helping resolve issues related to SAS and UNIX interaction. Appreciation is also extended to John Morrill and Lori Griffin for reviewing the manuscript.

SAS® is a registered trademark or trademark of SAS Institute Inc. in the USA and other countries. © Indicates USA registration.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Sam Mao
Quintiles, Inc.
P.O. Box 9708
Kansas City, MO 64137-9708
Phone: (816)767-6000
e-mail: sam.mao@quintiles.com

Jay Zhou
Quintiles, Inc.
P.O. Box 9708
Kansas City, MO 64137-9708
Phone: (816)767-6000
e-mail: jay.zhou@quintiles.com

Figure 7. An example of the email reporting to the owner of the ‘problem’ process.