Lab 6

Process Management

# Outcomes

* In this lab, you will learn how to manage processes on a Linux/Unix System using scripts, cronjobs/systemd timers, and resource limits.

Rubric (10 points total):

* Each step with points must be demonstrated to the TA.

# Procedures

Some ill-behaved processes, either intentional or not, could easily bring down a production system if they are not managed properly. For example, a careless programmer might repeatedly create processes, and within seconds, the system would become unresponsive and even crash eventually.

You will do all the work for this lab on your virtual machine, using both the root account, and user account you created for yourself during lab 1. For the purpose of this lab, you will be using a program to simulate various scenarios to stress test a system. The program can be downloaded from <http://www.cs.ucr.edu/~vhill/cs183/proc.c> (use wget to download the file to your virtual machine). Compile the program and name the executable 'proc' (note: this is a c program, so use gcc to compile it).

## Kill Runaway Processes Using a Script in Cron (5 points)

Run ./proc -h with your non-administrative user account on your VM to create a process that hogs the CPU. Use ps and top commands to identify the process. Write a script (Bash, Python or Perl - though Bash will be the fastest to code) that will be run as root in the system crontab or systemd timer every minute to catch and kill any runaway process (excluding the root processes) that consumes CPU time for more than 2 minutes. Show the script and crontab to the TA, and demo it.

## Limit the number of processes (5 points)

Run ./proc -n 10000 -s 100 with your user account to create 10 thousands processes, each process will sleep for 100 seconds. Then try to login from the host machine via ssh, ie. ssh yourUserName@yourvmip, and do ls -l after login. You should notice the slowness of your system. If not, increase the number of option -n for the proc program and try again.

Now, edit /etc/security/limits.conf as root and add a new line to set up a hard limit for the maximum number of processes to 20 for your user account. At the beginning of the file you should see a nice explanation of everything in the file and should hint you on what to add to the file. Remember to log yourself out and login again. Run the same command again and show what happens; take a screenshot of the output. Additionally, run ./proc -n 20 and see what happens. When you demo to the TA, explain why there are fewer processes (should be 18 in this case) allowed to be created by proc.

|  |
| --- |
| **Note**: most of the servers accessible to undergraduates at UCR have a limit on the number of processes that any individual user can have. This makes it so that no single student can use up too many resources and the server will be accessible to everyone. The counter to this is that if you try and create too many processes then you can hit the top limit and new ones (like ssh) cannot spawn. This most typically manifests itself when students are working with the fork() system call (as they do in CS 100) to spawn new processes and do not clean them up (making them zombies), leading to increasing numbers of processes and eventually hitting the limit and being unable to spawn any more. |

Notes & Tips: After you are done with this lab, make sure you change everything back to default.