Script to check memory usage, cpu usage and to kill processes if they use more than the specified value.

Jilvin Jacob, Amal S Raj, Anish R

February 23 2017

https://github.com/jilvin/mem_cpu_check_kill

1 Introduction

What we are up to?

Our task was to write a shell script that checks on memory usage and cpu usage of various processes and to kill them if they use more than a specified value. Moreover we had to set it start at system startup.

2 Script Declaration

We are writing a shell script. So it is advised to add first line as:
#!/bin/sh

Script now:
#!/bin/sh

3 Obtaining information from system.

We need to obtain information on currenly working processes from the system. Here, we will be using the package 'ps'.
So we will be adding line as:
ps

Script now:
#!/bin/sh
ps

4 ps need flags

Simply using ps won't return what we actually need. We need pmem and pcpu values of all processes.
So we will be rewriting line as:
ps -e -o pmem,pcpu,pid,user,comm

Script now:
#!/bin/sh
ps -e -o pmem,pcpu,pid,user,comm

5 ps need "more" flags

When we use ps -e -o pmem,pcpu,pid,user,comm; We actually will have to skip the first return line before processing. To skip the titles, it is better to add an '=' after each format name.
So we will be rewriting line as:
ps -e -o pmem=,pcpu=,pid=,user=,comm=

Script now:
#!/bin/sh
ps -e -o pmem=,pcpu=,pid=,user=,comm=

6 Advantage of sorting

Here, we are to kill processes that use more resources than a specified value, so it is better to get a sorted return from ps.
So we will be rewriting line as:
ps -e -o pmem=,pcpu=,pid=,user=,comm= –sort=-pmem
The - symbol indicates that sorting must be in descending order with respect to pmem values.

Script now:
#!/bin/sh
ps -e -o pmem=,pcpu=,pid=,user=,comm= –sort=-pmem

7 Start reading values.

We haven't actually read the values into our script/process.So we will be adding a '—' operator and a while loop to read each line of return.
ps -e -o pmem=,pcpu=,pid=,user=,comm= –sort=-pmem —
while read size cpu pid user comm
do
done
Here size, cpu, pid, user, comm are variables of the script. ie values from ps will be saved to these variables.

Script now:
#!/bin/sh
ps -e -o pmem=,pcpu=,pid=,user=,comm= –sort=-pmem —
while read size cpu pid user comm
do
done

8 Declaring check values.

We have to declare the values to be checked for at the beginning of the script.So we will be adding the lines:
check_mem_val=10.0
check_cpu_val=10.0
We are using float values because the return from ps will be in percentage for pmem and pcpu.

Script now:
#!/bin/sh
check_mem_val=10.0
check_cpu_val=10.0
ps -e -o pmem=,pcpu=,pid=,user=,comm= –sort=-pmem —
while read size cpu pid user comm
do
done

9 Comparing values.

We have to compare the values(value obtained, value specified) in order to kill processes using more resources than specified. So we will be adding the lines within the while loop:
kill_mem=$(echo "$size¿,$check_mem_val" — bc)
kill_cpu=$(echo "$cpu¿,$check_cpu_val" — bc)
if ["$kill_mem" = "1"]
then
kill $pid # $size $user $comm
elif ["$kill_cpu" = "1"]
then
kill $pid # $size $user $comm
else
continue
fi
bc is a package that is used for comparisons of float values as shell isn't capable of doing that.
bc returns '1' if it the condition is satisfied.

The kill command is used for killing a process.

Script now:
#!/bin/sh
check_mem_val=10.0
check_cpu_val=10.0
ps -e -o pmem=,pcpu=,pid=,user=,comm= –sort=-pmem —
while read size cpu pid user comm
do
kill_mem=$(echo "$size¿,$check_mem_val — bc")
kill_cpu=$(echo "$cpu¿,$check_cpu_val" — bc)
if ["$kill_mem" = "1"]
then
kill $pid # $size $user $comm
elif ["$kill_cpu" = "1"]
then
kill $pid # $size $user $comm
else
continue
fi
done

10 Comparing user.

It is better to kill processes owned by the user not the root. So we will be adding an outer if structure.

Script now:
#!/bin/sh
check_mem_val=10.0
check_cpu_val=10.0
ps -e -o pmem=,pcpu=,pid=,user=,comm= –sort=-pmem —
while read size cpu pid user comm
do
if ["$user" = "golden-+"]
then
kill_mem=$(echo "$size¿$check_mem_val — bc")
kill_cpu=$(echo "$cpu¿$check_cpu_val" — bc)
if ["$kill_mem" = "1"]
then
kill $pid # $size $user $comm
elif ["$kill_cpu" = "1"]
then
kill $pid # $size $user $comm
else
continue
fi
fi
done

11 Infinite loop.

Also, we need to set an outer infinite loop so that the program is repeated all the time the system is up.

Script now:
#!/bin/sh
check_mem_val=10.0
check_cpu_val=10.0
while(true)
do
ps -e -o pmem=,pcpu=,pid=,user=,comm= –sort=-pmem —
while read size cpu pid user comm
do
if ["$user" = "golden-+"]
then
kill_mem=$(echo "$size¿$check_mem_val — bc")
kill_cpu=$(echo "$cpu¿$check_cpu_val" — bc)
if ["$kill_mem" = "1"]
then
kill $pid # $size $user $comm
elif ["$kill_cpu" = "1"]
then
kill $pid # $size $user $comm
else
continue
fi
fi
done
done

12 Control process usage.

We created a process to control cpu usage. We don't want it to use more of the cpu. So we will be sleeping/pausing the outer loop for 1s.
We will be using sleep command.

Script now:
#!/bin/sh
check_mem_val=10.0
check_cpu_val=10.0
while(true)
do
ps -e -o pmem=,pcpu=,pid=,user=,comm= –sort=-pmem —
while read size cpu pid user comm
do
if ["$user" = "golden-+"]
then
kill_mem=$(echo "$size¿$check_mem_val — bc")
kill_cpu=$(echo "$cpu¿$check_cpu_val" — bc)
if ["$kill_mem" = "1"]
then
kill $pid # $size $user $comm
elif ["$kill_cpu" = "1"]
then
kill $pid # $size $user $comm
else
continue
fi
fi
done
sleep 1
done

13 Adding the script to startup.(optional)

To make the script run on startup on Debian/Ubuntu do the following steps.

1) Copy the script to /etc/init.d directory

2) sudo update-rc.d mem_cpu_check_kill.sh defaults

To remove the script from running on startup, do:

sudo update-rc.d -f mem_cpu_check_kill.sh remove