Operating Systems
Memory Stress Tests in CGROUPS

Ameet Deshpande - (CS15B001)

October 31, 2017

1 What are CGROUPS

- CGROUPS are short for Control Groups which was introduced in Linux. Cgroups basically allow us to allocate resources such as CPU time, system memory, network bandwidth among user-defined groups of tasks. This seems very advantageous when we are expecting some kind of workload for the system.
- One potential downside is the fact that Control Groups are still user defined. If there was some way to automatically cluster process to enhance scheduling (like GAS) we could place those processes in the Control Groups. But nevertheless, Linux gives us the power to group processes and enforce certain constraints on the whole set of processes rather than single processes.
- CGROUPS also have a hierarchy like processes in Linux. Child CGROUPS can inherit attributes from the parent CGROUPS. One difference however is that in Linux Processes, all of them are childern of *init*. In other words, it is a single Hierarchy. In CGROUPS however. there are multiple hierarchies.

2 Applying Stress Tests

- The assignment mainly requires us to enforce certain memory conditions on the CGROUPS. We use the package called *stress* which is available for enforcing the memory constraints.
- *stress* spawns processes which stress the hardware and OS according to the parameters provided.
- The command *stress -m 3* spawns 3 processes which stress the hardware with $256MB$ each. We can further tune the amount of memory each process occupies.
- The command *stress -m 4 –vm-bytes 300M* basically stresses the hardware with 4 processes of $300MB$ each. These commands thus give us a lot of control over how much memory each process is using. At the same time, it allows us to get multiple processes under the same CGROUP.

3 Applying memory constraints to CGROUPS

- CGROUPS can be created and constraints can be applied to them. The following commands illustrate the same.

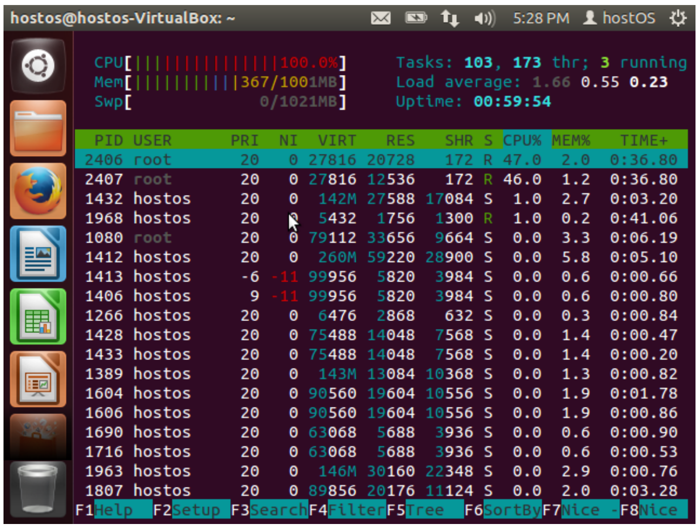
```
sudo cgcreate −g memory:osgroup
sudo cgset −r memory.limit_in_bytes=$((25∗1024∗1024))$ osgroup
sudo cgexec −g memory:osgroup stress −m 3
```

- The first command creates a control group. The second command sets the memory limit and the third spawns processes under the control group. *htop* is being used for monitoring the processes.

4 Experiments

4.1 Part 1

- *stress* basically provides processes which run for a very long time. We thus use them to test how the memory is doing. We set the CGROUPS memory limit as $100M$.
- We spawn 2 processes with $25M$ each, with a memory limit of $100M$. We expect the programs to run without a problem and the screenshot below suggest the same.

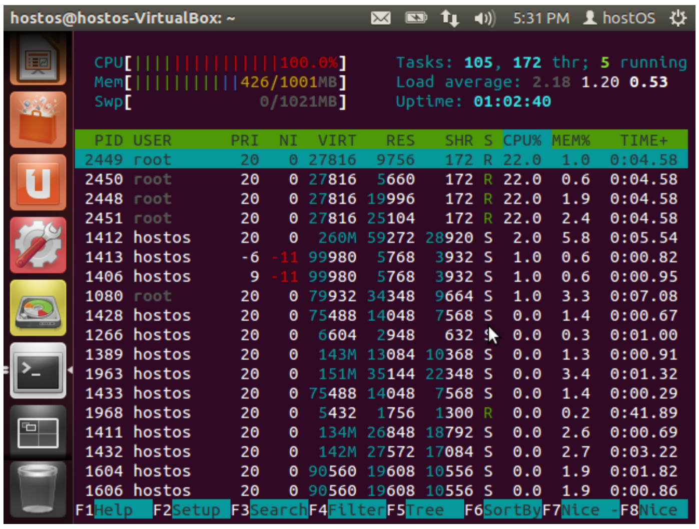


```
  CPU[||||||||||||||||100.0%]     Tasks: 103, 173 thr; 3 running
  Mem[|||||||||||367/1001MB]      Load average: 1.66 0.55 0.23
  Swp[              0/1021MB]     Uptime: 00:59:54

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+
 2406 root       20   0 27816 20728   172 R 47.0  2.0  0:36.80
 2407 root       20   0 27816 12536   172 R 46.0  1.2  0:36.80
 1432 hostos     20   0  142M 27588 17084 S  1.0  2.7  0:03.20
 1968 hostos     20   0  5432  1756  1300 R  1.0  0.2  0:41.06
 1080 root       20   0 79112 33656  9664 S  0.0  3.3  0:06.19
 1412 hostos     20   0  260M 59220 28900 S  0.0  5.8  0:05.10
 1413 hostos     -6 -11 99956  5820  3984 S  0.0  0.6  0:00.66
 1406 hostos      9 -11 99956  5820  3984 S  0.0  0.6  0:00.80
 1266 hostos     20   0  6476  2868   632 S  0.0  0.3  0:00.84
 1428 hostos     20   0 75488 14048  7568 S  0.0  1.4  0:00.47
 1433 hostos     20   0 75488 14048  7568 S  0.0  1.4  0:00.20
 1389 hostos     20   0  143M 13084 10368 S  0.0  1.3  0:00.82
 1604 hostos     20   0 90560 19604 10556 S  0.0  1.9  0:01.78
 1606 hostos     20   0 90560 19604 10556 S  0.0  1.9  0:00.86
 1690 hostos     20   0 63068  5688  3936 S  0.0  0.6  0:00.90
 1716 hostos     20   0 63068  5688  3936 S  0.0  0.6  0:00.53
 1963 hostos     20   0  146M 30160 22348 S  0.0  2.9  0:00.76
 1807 hostos     20   0 89856 20176 11124 S  0.0  2.0  0:03.28
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice
```

Figure 1

- Increasing the number of processes to 4, we can see that the memory utilization increased.

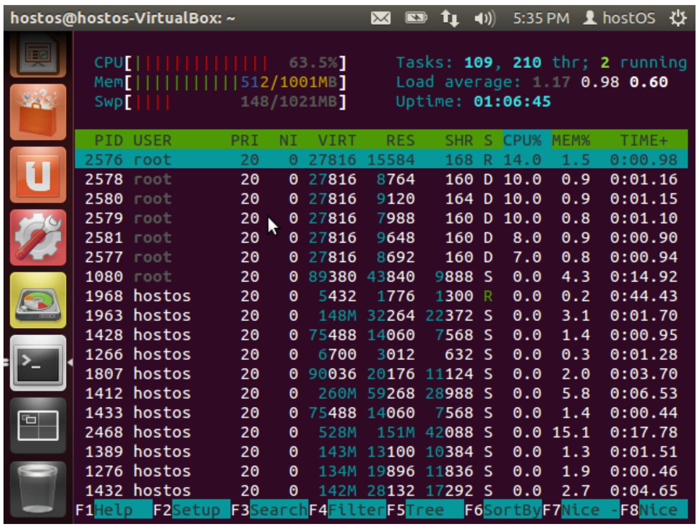

hostos@hostos-VirtualBox: ~   5:31 PM hostOS

```
  CPU[||||||||||||||||100.0%]     Tasks: 105, 172 thr; 5 running
  Mem[|||||||||||426/1001MB]      Load average: 2.18 1.20 0.53
  Swp[                0/1021MB]   Uptime: 01:02:40

  PID USER       PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+
 2449 root        20   0 27816  9756   172 R 22.0  1.0  0:04.58
 2450 root        20   0 27816  5660   172 R 22.0  0.6  0:04.58
 2448 root        20   0 27816 19996   172 R 22.0  1.9  0:04.58
 2451 root        20   0 27816 25104   172 R 22.0  2.4  0:04.58
 1412 hostos      20   0  260M 59272 28920 S  2.0  5.8  0:05.54
 1413 hostos      -6 -11 99980  5768  3932 S  1.0  0.6  0:00.82
 1406 hostos       9 -11 99980  5768  3932 S  1.0  0.6  0:00.95
 1080 root        20   0 79932 34348  9664 S  1.0  3.3  0:07.08
 1428 hostos      20   0 75488 14048  7568 S  0.0  1.4  0:00.67
 1266 hostos      20   0  6604  2948   632 S  0.0  0.3  0:01.00
 1389 hostos      20   0  143M 13084 10368 S  0.0  1.3  0:00.91
 1963 hostos      20   0  151M 35144 22348 S  0.0  3.4  0:01.32
 1433 hostos      20   0 75488 14048  7568 S  0.0  1.4  0:00.29
 1968 hostos      20   0  5432  1756  1300 R  0.0  0.2  0:41.89
 1411 hostos      20   0  134M 26848 18792 S  0.0  2.6  0:00.69
 1432 hostos      20   0  142M 27572 17084 S  0.0  2.7  0:03.22
 1604 hostos      20   0 90560 19608 10556 S  0.0  1.9  0:01.82
 1606 hostos      20   0 90560 19608 10556 S  0.0  1.9  0:00.86
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice
```

Figure 2

- Increasing the number of processes forces the swap space to be used as can be seen below. It can be seen that about 140M of SWAP is being used along with the main memory. We expect such programs to run slower given that large amount of disk I/O will be involved.


hostos@hostos-VirtualBox: ~   5:35 PM hostOS

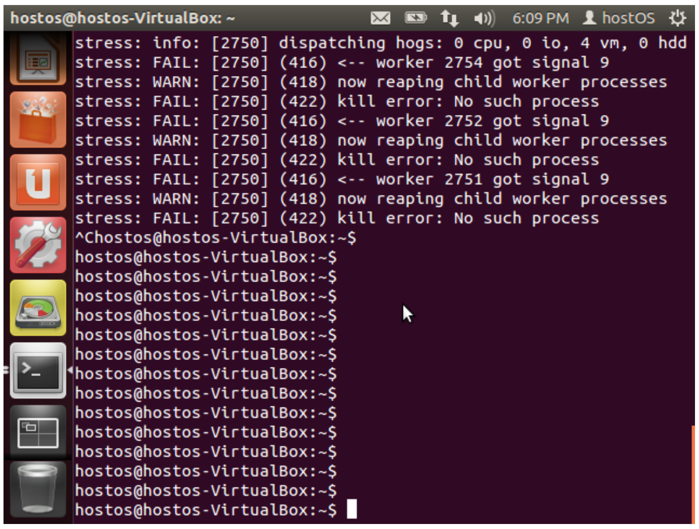
```
  CPU[||||||||||||||    63.5%]     Tasks: 109, 210 thr; 2 running
  Mem[||||||||||||512/1001MB]      Load average: 1.17 0.98 0.60
  Swp[||||          148/1021MB]    Uptime: 01:06:45

  PID USER       PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+
 2576 root        20   0 27816 15584   168 R 14.0  1.5  0:00.98
 2578 root        20   0 27816  8764   160 D 10.0  0.9  0:01.16
 2580 root        20   0 27816  9120   164 D 10.0  0.9  0:01.15
 2579 root        20   0 27816  7988   160 D 10.0  0.8  0:01.10
 2581 root        20   0 27816  9648   160 D  8.0  0.9  0:00.90
 2577 root        20   0 27816  8692   160 D  7.0  0.8  0:00.94
 1080 root        20   0 89380 43840  9888 S  0.0  4.3  0:14.92
 1968 hostos      20   0  5432  1776  1300 R  0.0  0.2  0:44.43
 1963 hostos      20   0  148M 32264 22372 S  0.0  3.1  0:01.70
 1428 hostos      20   0 75488 14060  7568 S  0.0  1.4  0:00.95
 1266 hostos      20   0  6700  3012   632 S  0.0  0.3  0:01.28
 1807 hostos      20   0 90036 20176 11124 S  0.0  2.0  0:03.70
 1412 hostos      20   0  260M 59268 28988 S  0.0  5.8  0:06.53
 1433 hostos      20   0 75488 14060  7568 S  0.0  1.4  0:00.44
 2468 hostos      20   0  528M  151M 42088 S  0.0 15.1  0:17.78
 1389 hostos      20   0  143M 13100 10384 S  0.0  1.3  0:01.51
 1276 hostos      20   0  134M 19896 11836 S  0.0  1.9  0:00.46
 1432 hostos      20   0  142M 28132 17292 S  0.0  2.7  0:04.65
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice
```

Figure 3

4.2 Part 2

- This section aims to check how the swappiness of the system effects the processes. We set the swappiness of the system to 0 to check how the processes respond. We expect the process to not even run when the amount of memory is not enough for it. The following is the result.



```
stress: info: [2750] dispatching hogs: 0 cpu, 0 io, 4 vm, 0 hdd
stress: FAIL: [2750] (416) <-- worker 2754 got signal 9
stress: WARN: [2750] (418) now reaping child worker processes
stress: FAIL: [2750] (422) kill error: No such process
stress: FAIL: [2750] (416) <-- worker 2752 got signal 9
stress: WARN: [2750] (418) now reaping child worker processes
stress: FAIL: [2750] (422) kill error: No such process
stress: FAIL: [2750] (416) <-- worker 2751 got signal 9
stress: WARN: [2750] (418) now reaping child worker processes
stress: FAIL: [2750] (422) kill error: No such process
^Chostos@hostos-VirtualBox:~$
hostos@hostos-VirtualBox:~$
hostos@hostos-VirtualBox:~$
hostos@hostos-VirtualBox:~$
hostos@hostos-VirtualBox:~$
hostos@hostos-VirtualBox:~$
hostos@hostos-VirtualBox:~$
hostos@hostos-VirtualBox:~$
hostos@hostos-VirtualBox:~$
hostos@hostos-VirtualBox:~$
hostos@hostos-VirtualBox:~$
hostos@hostos-VirtualBox:~$
hostos@hostos-VirtualBox:~$
hostos@hostos-VirtualBox:~$
hostos@hostos-VirtualBox:~$
```

Figure 4: No Swap

4.3 Performance

- To check how the performance of the program varies with the memory limit, a small Java program was written. The code is attached below.

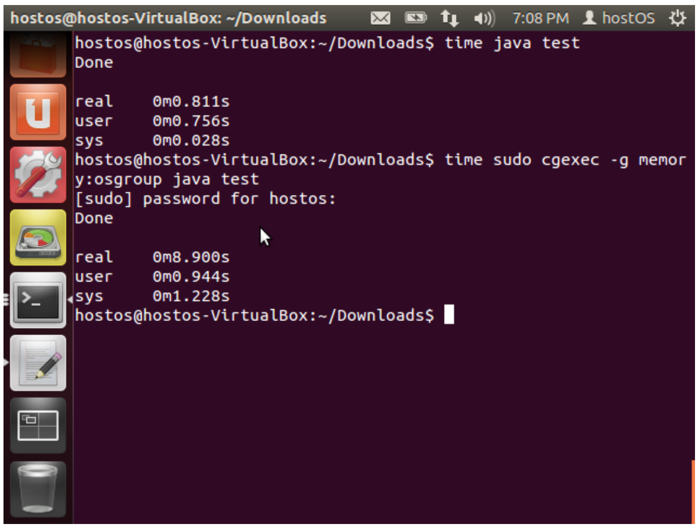
```
import java.util.*;

public class test {
    public static void main(String[] args){
        LinkedList<String> a = new LinkedList<String>();
        String temp = "oslab_projects";

        for(int i=0;i<1024*1024*8;++i){
            a.add(temp);
        }

        System.out.println("Done");
    }
}
```

The CGROUP size was limited to just $50MB$, while this program takes about $100MB$. The JAVA heap size was made around $1000MB$ so that garbage collector does not interfere in the performance measurement. The following screenshot measure the performance of the java programs with and without the CGROUP.



```
hostos@hostos-VirtualBox:~/Downloads$ time java test
Done

real    0m0.811s
user    0m0.756s
sys     0m0.028s
hostos@hostos-VirtualBox:~/Downloads$ time sudo cgexec -g memor
y:osgroup java test
[sudo] password for hostos:
Done

real    0m8.900s
user    0m0.944s
sys     0m1.228s
hostos@hostos-VirtualBox:~/Downloads$ 
```

Figure 5: No Swap

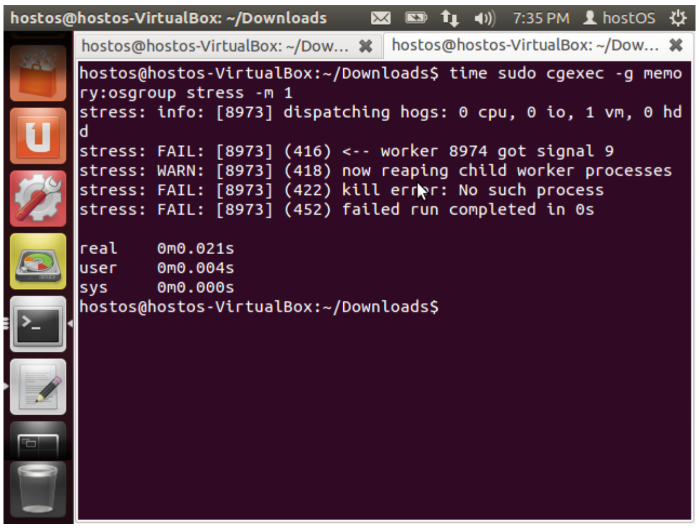
- As can be seen in the screenshot, when the heap size is kept unbounded, the program takes only 0.8 seconds to execute. When the Memory of the CGROUP is limited, the program takes 8.9 seconds to run. We can clearly see how usage of the SWAP affects the program time.

5 Soft and Hard Limits

- There are two ways of handling limits in CGROUPS. If the hard limit is selected, the total amount of physical memory used by the sum of all processes in the job will not be allowed to exceed the limit. If the processes try to allocate more memory, the allocation will succeed, and virtual memory will be allocated, but no additional physical memory will be allocated. The system will keep the amount of physical memory constant by swapping some page from that job out of memory. If the soft limit is selected, the job will be allowed to go over the limit if there is free memory available on the system. Only when there is contention between other processes for physical memory will the system force physical memory into swap and push the physical memory used towards the assigned limit. The following command helps us create CGROUPS with soft limits.

```
memory.soft_limit_in_bytes
```

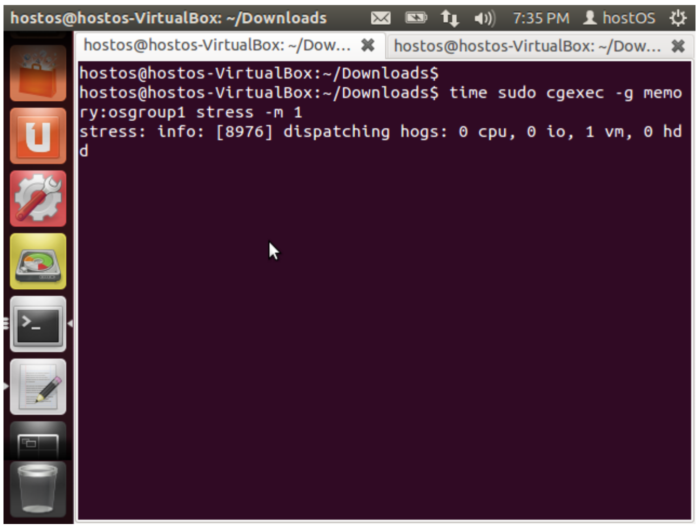
- We perform a very interesting experiment. We create 2 CGROUPS with the same limits and same tasks. We assign one of them Soft Limits and the other Hard Limits. We keep the memory usage of the processes such that a crash is expected. We see which of the processes crash first.



```
hostos@hostos-VirtualBox:~/Downloads$ time sudo cgexec -g memo
ry:osgroup stress -m 1
stress: info: [8973] dispatching hogs: 0 cpu, 0 io, 1 vm, 0 hd
d
stress: FAIL: [8973] (416) <-- worker 8974 got signal 9
stress: WARN: [8973] (418) now reaping child worker processes
stress: FAIL: [8973] (422) kill error: No such process
stress: FAIL: [8973] (452) failed run completed in 0s

real    0m0.021s
user    0m0.004s
sys     0m0.000s
hostos@hostos-VirtualBox:~/Downloads$
```

Figure 6: Fail



```
hostos@hostos-VirtualBox:~/Downloads$
hostos@hostos-VirtualBox:~/Downloads$ time sudo cgexec -g memo
ry:osgroup1 stress -m 1
stress: info: [8976] dispatching hogs: 0 cpu, 0 io, 1 vm, 0 hd
d
```

Figure 7: Success

- The results are as expected. Given that the first group has hard-limits, the processes get killed. But the second CGROUP has soft limits and hence it continues executing.