Assignment 4 - CGroups and Namespaces

Shivansh Dhiman - 22m2120

Question 1 - CGroups

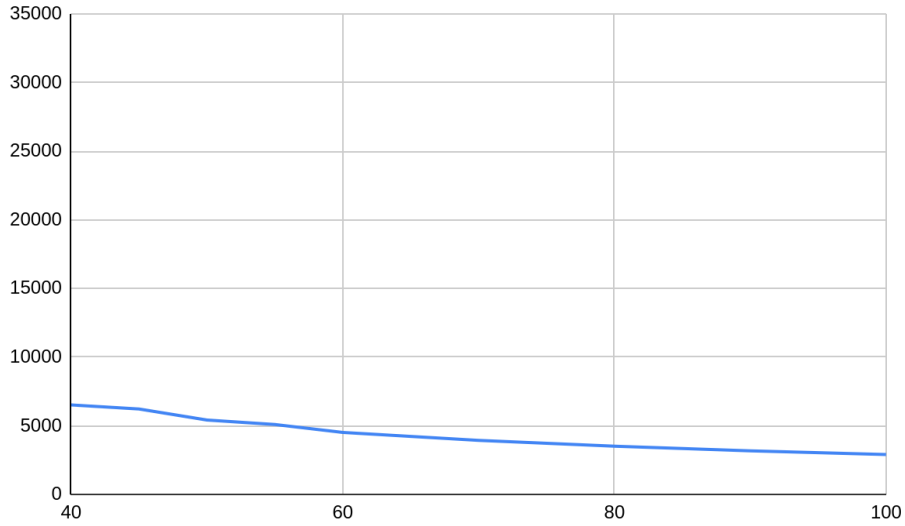
(a) **Question**: How can we limit the resources a process uses using cgroups?

Hypothesis: Cgroups use periods to check the resource usage of a process. It is defined in microseconds. During each period, the kernel checks if the process is taking more resources than specified. If so, it throttles the process until the specified quota for that resource is reached.

Data Representation: Here, I used three different periods and changed the quota for each of them.

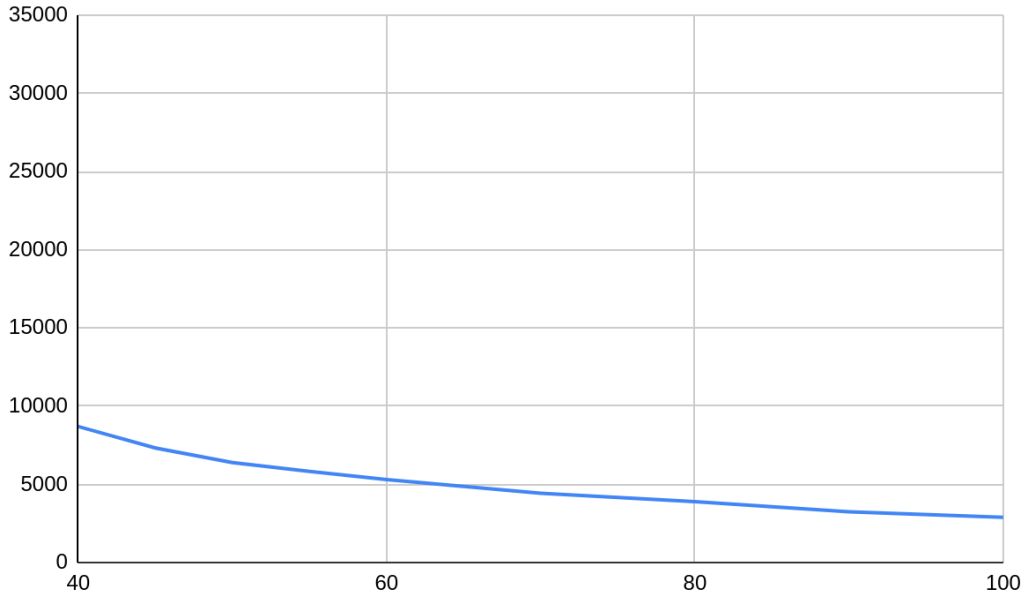
1. Period: 1000000 us = 1 second

quota	CPU percent	loop till 2B (in ms)	nr_periods	nr_throttled	cpuacct_usage
1000000	100	2897	3	0	2898
900000	90	3168	4	2	2971
800000	80	3505	4	3	3021
700000	70	3933	5	4	3024
600000	60	4508	5	4	2916
550000	55	5088	6	5	2919
500000	50	5418	6	5	2919
450000	45	6225	7	6	2926
400000	40	6516	7	6	2921



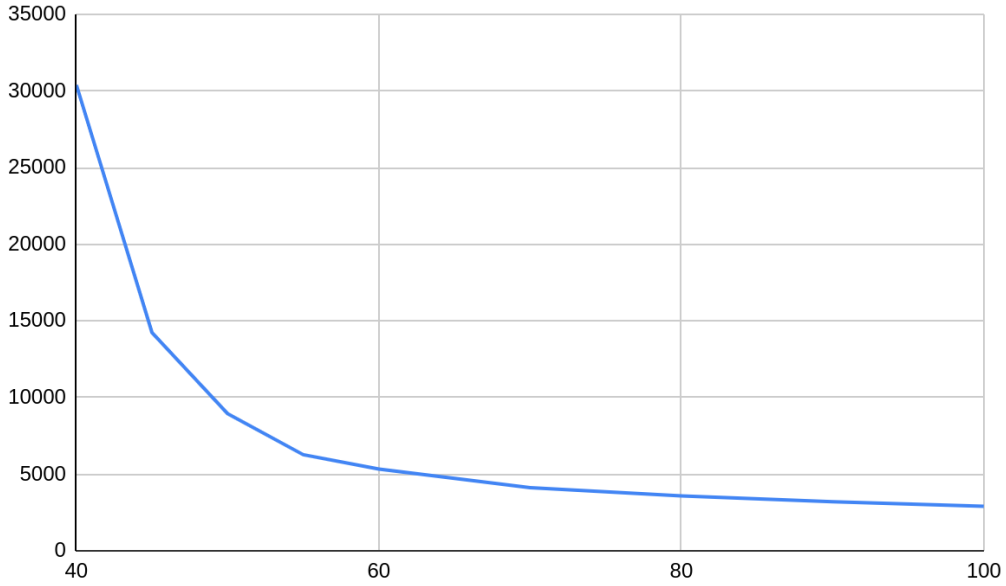
2. Period: 500000 us = 0.5 seconds

quota	CPU percent	loop till 2B (in ms)	nr_periods	nr_throttled	cpuacct_usage
500000	100	2894	7	0	2894
450000	90	3247	8	6	2956
400000	80	3887	9	7	3191
350000	70	4428	10	8	3230
300000	60	5309	12	10	3312
275000	55	5829	13	11	3359
250000	50	6394	14	12	3398
225000	45	7325	16	14	3480
200000	40	8695	19	17	3602



3. Period: 100000 us = 0.1 seconds

quota	CPU percent	loop till 2B (in ms)	nr_periods	nr_throttled	cpuacct_usage
100000	100	2896	31	0	2896
90000	90	3201	34	31	2893
80000	80	3587	39	36	2892
70000	70	4123	44	41	2894
60000	60	5334	57	54	3218
55000	55	6265	65	62	3481
50000	50	8948	92	89	3961
45000	45	14247	145	142	6443
40000	40	30397	306	304	12233



Observation: We can see that in each of the cases, as we decrease the quota, the CPU usage of that process decreases accordingly. The **graph is mostly linear** (especially in the first two values of the period)**,** up until a point where the overhead of throttling rises dramatically. It is important to note that, if there is a low enough quota and a large number of periods, the execution time drastically increases due to the overhead of checking and throttling the process. This time adds up and reflects in the “cpuacct_usage” statistic.

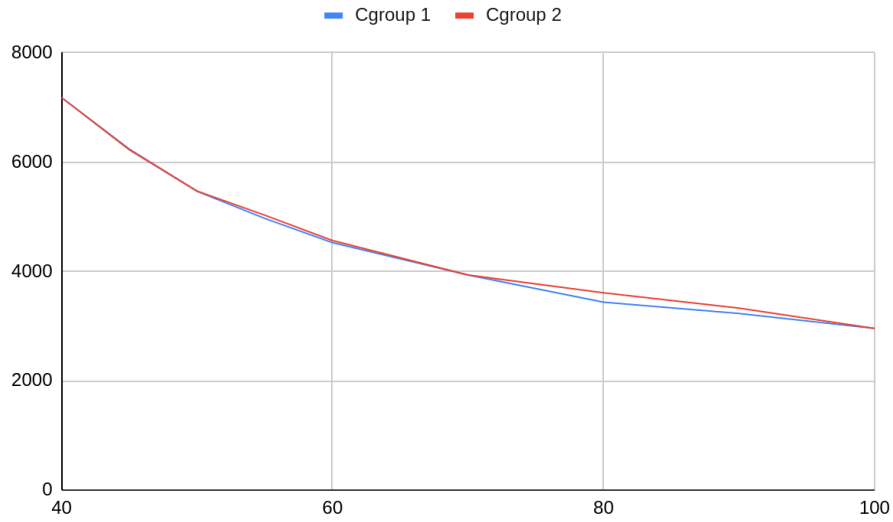
(b) **Question**: How can we limit the resources multiple processes use using multiple cgroups?

Hypothesis: Cgroups use periods to check the resource usage of a process. It is defined in microseconds. During each period, the kernel checks if the process is taking more resources than specified. If so, it throttles the process until the specified quota for that resource is reached.

Data Representation: Here, I used three different periods and changed the quota for each of them.

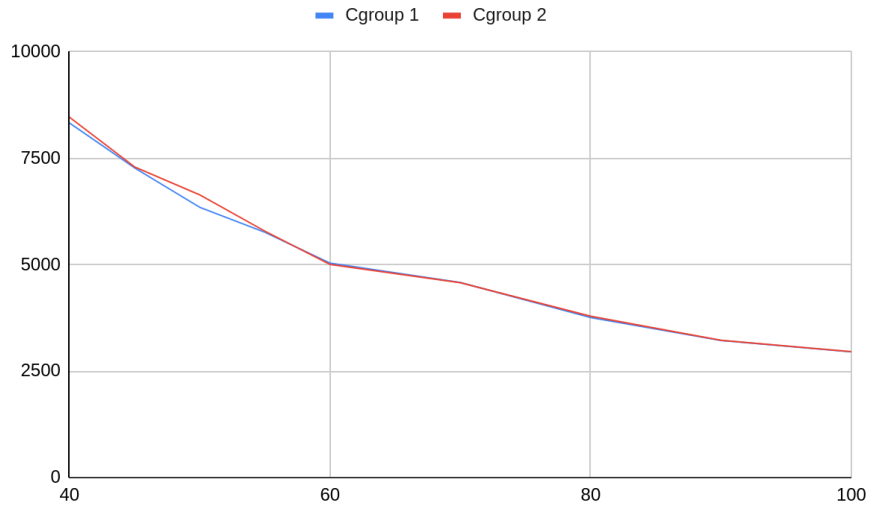
1. Period: 1000000us = 1s

quota	cpu percent	Cgroup 1	Cgroup 2
1000000	100	2959	2958
900000	90	3232	3330
800000	80	3438	3611
700000	70	3936	3934
600000	60	4526	4565
550000	55	4970	5030
500000	50	5466	5471
450000	45	6237	6227
400000	40	7181	7181



2. Period: 500000us = 0.5s

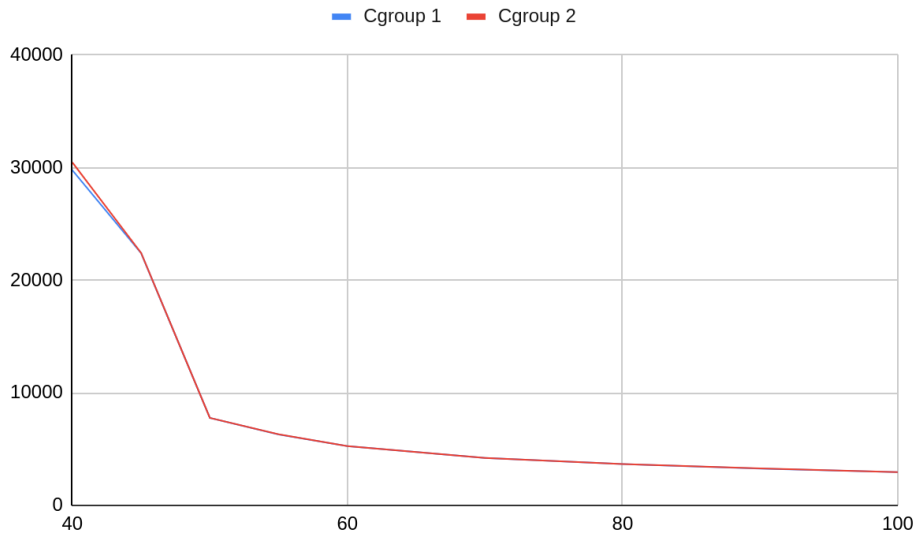
quota	cpu percent	Cgroup 1	Cgroup 2
500000	100	2951	2958
450000	90	3219	3223
400000	80	3760	3788
350000	70	4583	4579
300000	60	5036	5006
275000	55	5762	5790
250000	50	6349	6640
225000	45	7275	7296
200000	40	8330	8465



3. Period: 100000us = 0.1s

quota	cpu percent	Cgroup 1	Cgroup 2
100000	100	2959	2959
90000	90	3275	3278

80000	80	3673	3669
70000	70	4210	4209
60000	60	5270	5270
55000	55	6293	6308
50000	50	7769	7768
45000	45	22413	22433
40000	40	29781	30466



Observation: In the case of two cgroups, we can see that they behave identically to each other. We observe that in this way we can limit the resource usage of two processes using two cgroups. One notable difference from the earlier question is that the graph is not as linear. This is due to the fact that there is more throttling at lower quota values, hence leading to more overhead and longer execution time than expected.

Question 2 - Namespaces

(a) The first example in the provided link demonstrates creating a new process under a new namespace. This is done by passing special arguments to the ‘clone()’ system call. The program creates a new UTS Namespace for the new process. This namespace allows the processes in it to have their separate hostname and NIS domain name. In this example, the newly spawned process has a new hostname called “bizarro”, whereas the original hostname was “antero”.
While recreating the program, the original hostname was “abcd”, but the newly created namespace had the hostname “wxyz”. This demonstrates that the process is running in a UTS Namespace.

```
root@abcd:/home/sd/Desktop/cg# gcc demo_uts_namespaces.c
root@abcd:/home/sd/Desktop/cg# uname -n
abcd
root@abcd:/home/sd/Desktop/cg# ./a.out wxyz
PID of child created by clone() is 28511
uts.nodename in child:  wxyz
uts.nodename in parent: abcd
```

In the second example, we add a process to an existing namespace. This is done by “setns()” system call with the file descriptor of that namespace as an argument. Here an existing Mountpoint namespace was created.

```
root@abcd:/home/sd/Desktop/cg# gcc ns_exec.c -o ns_exec
root@abcd:/home/sd/Desktop/cg# hostname
abcd
root@abcd:/home/sd/Desktop/cg# ./ns_exec /proc/10171/ns/uts bash
root@wxyz:/home/sd/Desktop/cg# hostname
wxyz
root@wxyz:/home/sd/Desktop/cg# exit
exit
root@abcd:/home/sd/Desktop/cg#
```

(b) To create a new namespace using the ‘unshare’ command, we use the following:**sudo unshare -pf --mount-proc bash**

```
sd@abcd:~/Desktop/cg$ sudo unshare -pf --mount-proc bash
root@abcd:~/Desktop/cg# ps aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.2  0.0  22444  4624 pts/1    S    15:46   0:00 bash
root           9  0.0  0.0  37372  3300 pts/1    R+   15:46   0:00 ps aux
root@abcd:~/Desktop/cg# exit
exit
sd@abcd:~/Desktop/cg$ ps aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.0 225676  9604 ?        Ss   Apr09   0:10 /lib/systemd/systemd --system --deserialize 36
root           2  0.0  0.0      0     0 ?        S    Apr09   0:00 [kthreadd]
root           3  0.0  0.0      0     0 ?        I<   Apr09   0:00 [rcu_gp]
root           4  0.0  0.0      0     0 ?        I<   Apr09   0:00 [rcu_par_gp]
root           6  0.0  0.0      0     0 ?        I<   Apr09   0:00 [kworker/0:0H-kb]
root           9  0.0  0.0      0     0 ?        I<   Apr09   0:00 [mm_percpu_wq]
root          10  0.0  0.0      0     0 ?        S    Apr09   0:00 [ksoftirqd/0]
root          11  0.0  0.0      0     0 ?        I    Apr09   0:30 [rcu_sched]
```

To add a process to an existing namespace, we use the following command:

sudo nsenter -t <PID> -a

```
sd@abcd:~/Desktop/cg$ sudo nsenter -t 6981 -a
root@abcd:/# ps aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.0  22440  4900 pts/1    S+   16:55   0:00 bash
root          23  0.6  0.0  22432  4716 pts/2    S    16:58   0:00 -bash
root          33  0.0  0.0  37372  3428 pts/2    R+   16:58   0:00 ps aux
root@abcd:/#
```

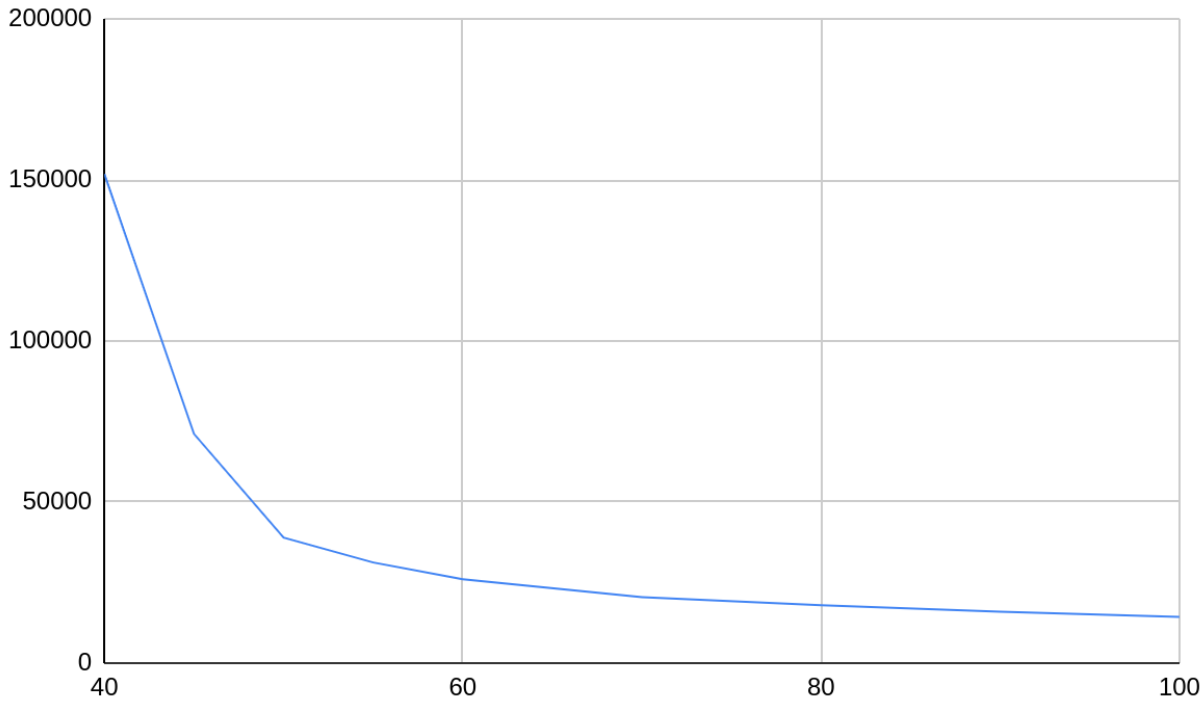
(c) The new program is stored in “new_demo_uts_namespaces.c”. The following screenshot shows the usage.

```
root@abcd:/home/sd/Desktop/cg# gcc new_demo_uts_namespaces.c -o new_demo_uts_namespaces
root@abcd:/home/sd/Desktop/cg# ./new_demo_uts_namespaces 8872
PID of child created by clone() is 9007
pidfd:/proc/8872/ns/uts
uts.nodename in child:  wxyz
uts.nodename in parent: abcd
child has terminated
root@abcd:/home/sd/Desktop/cg#
```

Question 3 - CGroups + Namespaces

(i) These are the observed execution times of processes in a namespace and a cgroup. Quotas were changed according to the period. Files involved in this experiment are “b.cpp”, “fivepid.c” and “b.sh”.

quota	cpu percent	Namespace+cgroup (in ms)
100000	100	14302
90000	90	15976
80000	80	17928
70000	70	20479
60000	60	25992
55000	55	31226
50000	50	38983
45000	45	71164
40000	40	151869



Observation: Here also observe the effect of lowering quota in cgroups inside a namespace. The effect is similar to that without a namespace. Each of the 5 processes in the namespace is throttled to limit the CPU usage according to the quota specified. Towards lower values of quota, we observe that the execution time increases exponentially because there is a significant overhead of throttling and context switches.