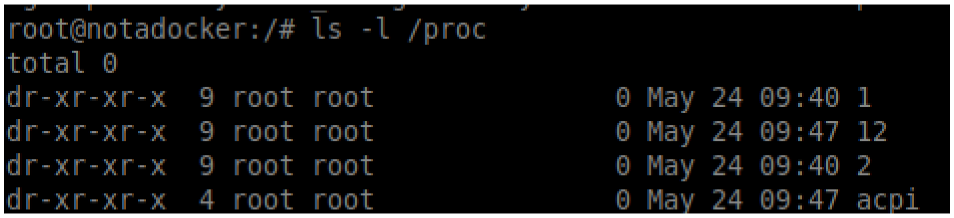
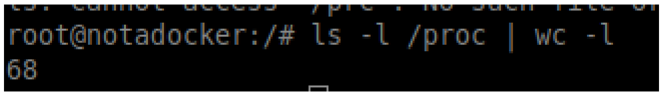
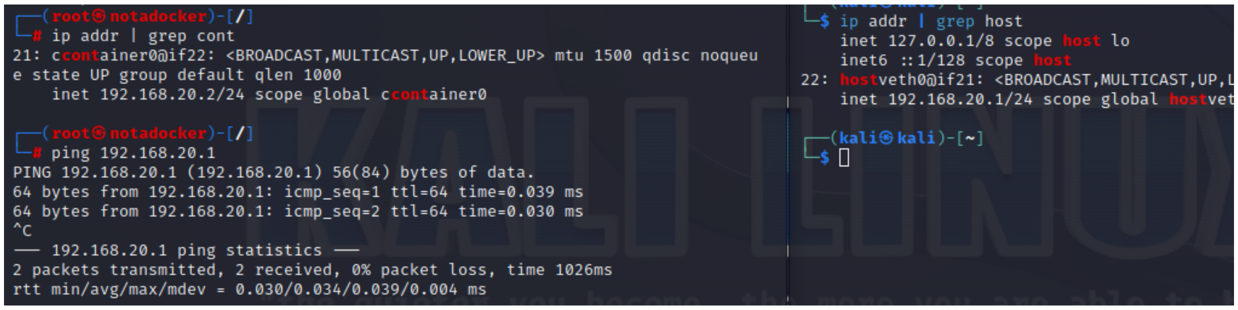
Intro

The idea of the work is to create minimalistic container to study techniques used in containerization engines and sandboxes. The parent process spawns child using `clone` function with namespaces isolation flags as parameter. Child process performs basic environment initialization: mounts of special filesystems like `proc` , `sys`, environmental variables, network interfaces configuration. Then, child process spawns shell with standard `fork-exec` sequence.

Features

- Rootfs is saved from custom Docker image. It is based on the `debian:bullseye-slim` and contains `sysbench` utility for performance benchmarking, `ping` utility for network connectivity testing
- Host environment preparation is done via `configure.sh` bash script which performs extracting, setup, mount of rootfs and configuration of network interfaces to allow containers . This script is also responsible for cleaning artifacts left from the previous run
- PID isolation
  - As we can see on the screenshot, the number of processes within the container is much less than on average Linux distro. Moreover, there are only 3 numbered process entries in the `proc fs`
  - 

    ```
    root@notadocker:/# ls -l /proc
    total 0
    dr-xr-xr-x  9 root root               0 May 24 09:40 1
    dr-xr-xr-x  9 root root               0 May 24 09:47 12
    dr-xr-xr-x  9 root root               0 May 24 09:40 2
    dr-xr-xr-x  4 root root               0 May 24 09:47 acpi
    ```
  - 

    ```
    root@notadocker:/# ls -l /proc | wc -l
    68
    ```
- Network namespace isolation & connection with host namespace
  - On the screenshot we can see possibility to ping host IP address from inside the container
  - 

    ```
    ┌──(root㉿notadocker)-[/]
    └─# ip addr | grep cont
    21: ccontainer0@if22: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueu
    e state UP group default qlen 1000
        inet 192.168.20.2/24 scope global ccontainer0

    ┌──(root㉿notadocker)-[/]
    └─# ping 192.168.20.1
    PING 192.168.20.1 (192.168.20.1) 56(84) bytes of data.
    64 bytes from 192.168.20.1: icmp_seq=1 ttl=64 time=0.039 ms
    64 bytes from 192.168.20.1: icmp_seq=2 ttl=64 time=0.030 ms
    ^C
    ── 192.168.20.1 ping statistics ──
    2 packets transmitted, 2 received, 0% packet loss, time 1026ms
    rtt min/avg/max/mdev = 0.030/0.034/0.039/0.004 ms
    ```

    ```
    └─$ ip addr | grep host
        inet 127.0.0.1/8 scope host lo
        inet6 ::1/128 scope host
    22: hostveth0@if21: <BROADCAST,MULTICAST,UP,L
        inet 192.168.20.1/24 scope global hostvet

    ┌──(kali㉿kali)-[~]
    └─$
    ```

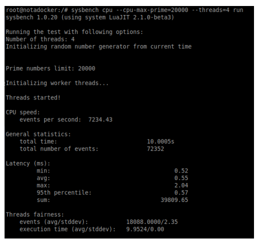
    - Additionally, access to WAN might be granted to the container using NAT with the following command: `sudo iptables -t nat -A POSTROUTING -s 192.168.20.0/24 -o <outbound_interface> -j MASQUERADE` where `<outbound_interface>` is the interface on host from which traffic is going to the default gateway

Run the project

```
./configure.sh
cmake -S . -B build
cd build && make && cd -
sudo build/ccontainer
```

Performance comparison

- Note that before performing I/O tests, utility has to be created with `sysbench fileio --file-total-size=512M --file-num=1 prepare`
    - P.S. few files might be created instead of just one. It will affect the test and results might be slightly different

Metric	Commands	CContainer	Docker	Host
CPU - 4 threads	`sysbench cpu --cpu-max-prime=20000 --threads=4 run` With multiple threads we can check whether (and how) multi threading affects performance in the container.	10.0005s 	10.0006s	10.0003s
CPU - 1 thread	`sysbench cpu --cpu-max-prime=20000 --threads=1 run` Calculating max-prime number is CPU-bound task. Performance might be measured relative to the execution time.	10.0003s	10.0004s	10.0001s
File IO Write	`sysbench fileio --file-total-size=512M --file-num=1 --file-test-mode=rndwr --file-rw-ratio=0 --time=30 run` This command performs benchmarks I/O with random writes. It outputs a number of various metrics such as number of writes performed. (non	written, MiB/s: 143.95	written, MiB/s: 190.65	written, MiB/s: 195.76

```
root@notadocker:/# sysbench cpu --cpu-max-prime=20000 --threads=4 run
sysbench 1.0.20 (using system LuaJIT 2.1.0-beta3)

Running the test with following options:
Number of threads: 4
Initializing random number generator from current time


Prime numbers limit: 20000

Initializing worker threads...

Threads started!

CPU speed:
    events per second:  7234.43

General statistics:
    total time:                          10.0005s
    total number of events:              72352

Latency (ms):
         min:                                    0.52
         avg:                                    0.55
         max:                                    2.04
         95th percentile:                        0.57
         sum:                                39809.65

Threads fairness:
    events (avg/stddev):           18088.0000/2.35
    execution time (avg/stddev):   9.9524/0.00
```

Metric	Commands	CContainer	Docker	Host
	mixed with reads) However, we are interested in the throughput (MiB/S).			
File IO Read	`sysbench fileio --file-total-size=512M --file-num=1 --file-test-mode=rndrd --file-rw-ratio=0 --time=30 run` This command performs benchmarks I/O with random reads (non mixed with writes). This is artificial test which would approximately show us reads performance. Although, in real applications reads are often mixed with writes	read, MiB/s: 3326.45	read, MiB/s: 3123.45	read, MiB/s: 3254.85
Memory	`sysbench memory --memory-block-size=4K --memory-total-size=2G run` Performs default memory write test with block size aligned to the page size. The command outputs the number of seconds needed to finish the task. It's important because it might show the ability of container to handle memory intensive tasks.	1.8006s	1.8962s	1.8020s

- For some reason, writes were slower in my container. I have went through possible causes variants, but unfortunately didn't find meaningful justification yet. The next step would be execution of container on different host (to ensure that the problem is not hardware / OS specific) and then trying lower level monitoring tools from the bcc collection.

Sources

- https://linuxhint.com/use-sysbench-for-linux-performance-testing/
- https://github.com/89luca89/basic_c_container