Create Simple Container

Anastasia Palashkina
Total Virtualization
Innopolis University
2024

Intro

The idea of the work is to create a container with isolated PID, MNT and NET using C language.

Features

- creating one container at a time
- upon completion of work with the container (exiting it), the disk is cleared, but its image is saved
- creates a 1GB loopback file using the dd command
- separated namespaces, network and PID namespace isolation
- to install sysbench or another additional tool you need to stop the container, manually replace the image with one with sysbench installed, then start the container.
- Description of function in the code:

  1. **ASSERT(int status, const char *msg)**:
     - This function is used to check the status of a given condition.
     - If the status is less than 0, it prints an error message using the perror function and exits the program with a failure status.
     - It returns the status value.
     - This function is used to handle errors in the code.

  2. **set_host_name(const char* hostname)**:
     - This function is used to set the hostname of the system.
     - It takes a string parameter `hostname` and uses the sethostname function to set the hostname of the system.
     - If the sethostname function fails, it calls the ASSERT function to handle the error.
     - This function does not return any value.
     - It is used to set the hostname of the system in the code.

  3. **setup_variables()**:
     - This function is used to set up environment variables for the current process.
     - It first clears all the environment variables using the clearenv function.
     - Then, it sets two environment variables using the setenv function: "TERM" with the value "xterm-256color" and "PATH" with the value "/bin/:/sbin:/usr/bin:/usr/sbin".
     - This function does not return any value.
     - It is used to set up environment variables in the code.

4. **run_system(const char* cmd)**:
   - This function is used to run a system command.
   - It takes a string parameter `cmd` which represents the command to be executed.
   - It uses the system function to execute the command.
   - If the return value of the system function is not 0, it calls the ASSERT function to handle the error.
   - This function does not return any value.
   - It is used to run system commands in the code.

5. **run_system_with_buffer(const char* cmd, size_t buffer_size)**:
   - This function is used to run a system command and capture the output in a buffer.
   - It takes two parameters: a string `cmd` representing the command to be executed, and a `buffer_size` representing the size of the buffer to store the output.
   - It uses the popen function to open a pipe and execute the command.
   - It then uses the fgets function to read the output of the command into the buffer.
   - If any of the function calls fail, it calls the ASSERT function to handle the error.
   - Finally, it closes the pipe using pclose and returns the buffer.
   - This function is used to run system commands and capture their output in the code.

6. **setup_filesystem()**:
   - This function is used to set up a filesystem for the code.
   - It first checks if a loopback file named "loopbackfile.img" exists using the access function.
   - If the file does not exist, it creates a 1GB loopback file using the dd command.
   - It then uses the losetup command to associate the loopback file with a loop device and captures the loop device name in a buffer.
   - If the loopback file did not exist, it downloads and installs Alpine Linux using the wget and tar commands.
   - Finally, it creates a temporary folder named "mnt", mounts the loopback device to the folder using the mount command, and returns the loop device name.
   - This function is used to set up the filesystem in the code.

7. **unsetup_filesystem(const char* loop_device)**:
   - This function is used to unmount the loopback device and remove the temporary folder created by the setup_filesystem function.
   - It takes a string parameter `loop_device` representing the loop device name.
   - It unmounts the loopback device using the umount command and removes the temporary folder using the rmdir command.
   - Finally, it releases the loop device using the losetup command.
   - This function does not return any value.
   - It is used to clean up the filesystem in the code.

8. **main_container(void* args)**:
   - This function is used as the entry point for the main container process.
   - It takes a void pointer `args` as a parameter, but it is not used in the function.
   - It defines an array `exec_args` containing the command to be executed ("/bin/sh") and a NULL pointer.
   - It then calls the execvp function to replace the current process with the specified command.
   - This function returns the exit status of the execvp function.

9. **child_fn(void* args)**:
   - This function is used as the entry point for the child process.
   - It takes a void pointer `args` as a parameter, but it is not used in the function.
   - It calls the set_host_name function to set the hostname of the container to "container".
   - It then calls the setup_variables function to set up environment variables.
   - It uses the chroot function to change the root directory to the "mnt" folder.
   - It uses the chdir function to change the current working directory to the root ("/").
   - It mounts the proc filesystem using the mount function.
   - It creates a stack for the main_container function using a char array.
   - It uses the clone function to create a new process with the main_container function as the entry point.
   - If the clone function fails, it prints an error message using perror and exits the program with a failure status.
   - It waits for the child process to exit using the wait function.
   - It unmounts the proc filesystem using the umount function.
   - This function returns the exit status of the child process.

10. **main()**:
    - This is the main function of the program.
    - It prints a welcome message.
    - It calls the setup_filesystem function to set up the filesystem and stores the loop device name in a variable `loop_device`.
    - It creates a stack for the child_fn function using a char array.
    - It uses the clone function to create a new process with the child_fn function as the entry point and several flags (CLONE_NEWPID, CLONE_NEWUTS, CLONE_NEWNET, CLONE_NEWNS) to create a new namespace for the child process.
    - If the clone function fails, it prints an error message using perror and exits the program with a failure status.
    - It waits for the child process to exit using the wait function.
    - It calls the unsetup_filesystem function to clean up the filesystem using the loop device name.
    - This function returns the exit status of the program.

Links

This project on Github : https://github.com/skylemn07/TV-container

Tests commands

Metric	Sysbench command	Why this command	What is interesting in sysbench output
CPU performance	sysbench --time=60 cpu --cpu-max-prime=64000 run	overloads all processor cores, which will help you see the difference between the total running time in different containers	Events per second, Latency avg
Scheduler performance	sysbench --num-threads=64 --test=threads --thread-yields=100 --thread-locks=2 run	show how well the system handles multiple threads accessing shared resources at the same time	Total time, Total number of events, Latency avg
Thread memory access	sysbench --threads=10 --time=60 memory --memory-oper=write run	write access to memory with page swapping is checked, we can track the impact on the data transfer rate in memory	Total time, Latency max
Memory access	sysbench --test=memory --memory-block-size=1M --memory-total-size=10G run	tests the memory performance by allocating a buffer of 1M size and a total memory size of 10G, and then runs the memory workload to measure the performance	Total time, Latency max
File I/O read/write	sysbench --test=fileio --file-total-size=512M --file-test-mode=rndrw --time=120 --max-time=300 --max-requests=0 run	performs big fileIO test for 5 min. to test fileio algorithm	File operations, Throughput

Table With Metrics

The average values of the indicators are indicated based on 10 repetitions of the test commands

	host machine	my container
CPU performance (Events per second)	125.49	105.36
CPU performance (Latency avg)	8.23	9.24
Scheduler performance (Total time, s)	10.0034	10.0114
Scheduler performance (Total number of events)	139325	66381
Scheduler performance (Latency avg)	4.59	9.65
Thread memory access (Total time)	7.2745	7.0892
Thread memory access (Latency max)	34.30	32.54
Memory access (Total time)	0.3976	0.6494
Memory access (Latency max)	0.17	1.70
File I/O (File operations, reads/s)	176.42	137.64
File I/O (File operations, writes/s)	118.45	95.12
File I/O (File operations, fsyncs/s)	378.35	288.45
File I/O (Throughput read, MiB/s)	2.77	1.97
File I/O (Throughput written, MiB/s)	1.85	1.24

Explanation Why Metrics Differ

The differences in metric results between the host machine and the container is attributed to various factors. Containers are designed to be lightweight and have a smaller footprint than virtual machines, meaning they utilize fewer resources. They also share the host's operating system and are isolated from the rest of the system, which impact their performance.

CPU test

The CPU performance difference is due to the overhead introduced by the container runtime, or the specific CPU allocation for the container.

File IO test

The differences in File I/O operations is attributed to the way containers handle storage and filesystems. Containers use storage drivers to manage the interactions between the container and the host filesystem, which can introduce additional overhead and affect performance.

Memory and Threads test

The memory access and thread memory access differences is be due to the way containers manage and allocate memory resources differently than a standard host machine.

Sources

1. [SysBenchExample] - “How to Benchmark Your System (CPU, File IO, MySQL) with Sysbench” https://www.howtoforge.com/how-to-benchmark-your-system-cpu-file-io-mysql-with-sysbench
2. [NSIT-USA] NSIT of the United States description of algorithm sha https://csrc.nist.gov/csrc/media/publications/fips/180/4/final/documents/fips180-4-draft-aug2014.pdf
3. [Kim17] D.Kim et. al.“Existing Deduplication Techniques”- 2017
4. [DockStorage] docker - about storage drivers https://docs.docker.com/storage/storagedriver/