Lab BSY MEM

Introduction & Prerequisites

This laboratory is to learn how to:

- Understand how Linux reports memory statistics and usage
- Understand the relationship between program memory requirements and Linux
- Understand practical ramifications of pages
- How to limit memory on a user basis
- How to limit memory by the use of a cgroup
- The relationship between virtual memory and a swap area
- How to initalise a swap area

The following resources and tools are required for this laboratory session:

- A ZHAW VPN session
- Any modern web browser
- Any modern SSH client application
- OpenStack Horizon dashboard: https://ned.cloudlab.zhaw.ch
- OpenStack account details
    - See Moodle
- Username to login with SSH into VMs in ned.cloudlab.zhaw.ch OpenStack cloud from your laptops
    - **`Ubuntu`**
- Ubuntu VM with at least 2 cores, preferably 4
- Installed C compiler, and tools (gcc/make)
    - `sudo apt update`
    - `sudo apt install build-essential`

Time

The entire session will require 90 minutes.

Assessment

No assessment foreseen

Task 1 – Setup & Basic Tasks

Setup your virtual machine with one or more cores

Check the number of CPUs and the number of online-cpus (using which command?)

Check the compiler installation (using which command?)

Install the cgroup-bin package

When this is installed edit `/etc/default/grub` and edit the relevant line to

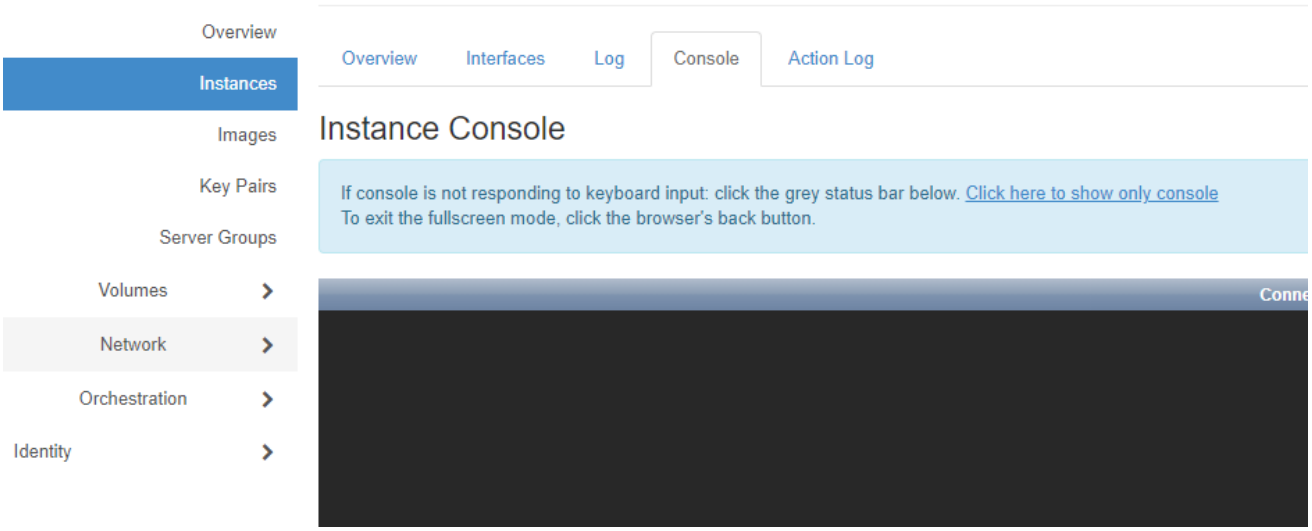
```
GRUB_CMDLINE_LINUX_DEFAULT="cgroup_enable=memory swapaccount=1"
```

Run

```
sudo update-grub
```

And reboot. This should install `cgexec`

Rebooting could take some time - you can check the VM console (https://ned.cloudlab.zhaw.ch): for the VM status.



Setup three to four terminal connections to your machine, it makes life easier

Subtask 1.1 – Basic memory under Linux

Download and unzip the file package `MEM_students_code.zip`.

Open the `MEM_lab.c` file and inspect the code. In the first section (**ToDo 1,**) call up a function to print out the PID of the running process - this will come in useful later (hint - use the getpid system call)

```
(void) printf( "----- the PID of this process is %i\n", getpid() );
```

Add an endless loop below this, exit, compile and run.

In a second terminal run top or htop - what memory parameters are useful to know? (Hint - use the documentation for htop to look for VIRT/RES/SHR)

In a third terminal using the command free (hint - man free) display the system memory parameters in kilobytes

Explain the parameters

Read the file `/proc/${pid}/status` specifically the memory related portions. What do the fields mean? (hint-man proc)
Note down the values or make a screenshot, we shall need this later
What is the difference between VmPin and VmLck?

Research the command `smem`, install if necessary. What information does smem give you about your system?

Note: `smem` is a useful tool for helping set up resource management in server systems

Go back to the code program. Remove the endless loop and insert code to read the page size (store it in a variable) and print it out. (**ToDo 2**) Build and run (hint - man getpagesize)

Verify this with the `getconf` command (hint - man getconf)

Now include code to reserve memory (hint - man malloc) the size of a number of pages (**ToDo 3**)

After each execution step of this code run the command (another terminal)

```
ps -o min_flt,maj_flt {pid}
```

Use the man page to understand the parameters. What do you notice?

Return to the code - for **ToDo 4** - use the `align_alloc` function to reserve a buffer of a number of pages size, aligned on a page boundary. Then use the function `mincore` to check whether the pages are in memory.

What do you see?

Linux uses lazy allocation - include an access to the buffer - **ToDo 5** - and run the code again. What do you see when you run the code and check the page faults reported by the ps command?

Subtask 1.2 – Limiting memory (1)

From reading the process status file we know the maximum amount of memory the process uses during startup. We can now attempt to limit this on a high level.

Research the `ulimit` command and use it to display the resource limitations. What precisely is limited?

Using the data from reading `/proc/${pid}/status` let us limit the available memory for the start phase of the test program to under the peak requirement. What happens?

How do you restore the unlimited memory access capability? What is your assessment of this method?

Subtask 1.3 – Limiting memory (2)

cgroups allows us to limit the resources used for individual processes. Here we will create a memory controller for our test process.

We define a cgroup memory controller. Use the man pages to understand the parameters

```
sudo cgcreate -a ubuntu:ubuntu -t ubuntu:ubuntu -g memory:myGroup
```

What memory controller files have been created? What can be used to set limitations of memory usage?

We check the peak memory usage of the process which should be, in section 5, high - around the 120M mark. We can now use this value to limit the process memory by writing an appropriate value into the group memory controller file

```
echo xxM > /sys/fs/cgroup/memory/myGroup/memory.limit_in_bytes
```

By running the process as follows

```
cgexec -g memory:myGroup process_name
```

The process will run under the condition set by the cgroups memory controller.

Check the results using the free command. Two things can happen - if it’s your lucky day the process will be killed. Why? The console output of your VM will give you a better hint. Explain it.

Subtask 1.4 – Setting up a swap area

Why should we bother with a swap area?

Because the principle of virtual memory depends on having excess secondary memory to enable a maximum number of processes to run “simultaneously.”

If using `free` it can be seen there is no swap area in secondary memory, then one needs to be setup. We do this in the following sequence

1.) Create a file that can be used for swapping
    a.) `sudo fallocate -l 1G /swapfile`
2.) Give this file root permissions only
    a.) `sudo chmod 600 /swapfile`
3.) Setup a Linux swap area in the file
    a.) `sudo mkswap /swapfile`
4.) Activate the swap file
    a.) `sudo swapon /swapfile`
5.) If this is to be permanent then
    a.) `sudo nano /etc/fstab`
    b.) And add: `/swapfile swap swap defaults 0 0`
6.) `sudo swapon --show` or
7.) `sudo free -h` will now show a swap area

Swappiness is a Linux kernel property that defines how often the system will use the swap space. Use the command to read the swappiness.

```
cat /proc/sys/vm/swappiness
```

What value do you get? The higher the value the more likely the kernel is to swap, the lower the value the more the kernel tries to avoid it. On production servers, a low swappiness is often preferred to decrease latencies and user available system memory.

Swappiness can be adjusted using:

```
sudo sysctl vm.swappiness=10
```

Read the manpage for swapoff

If your process was killed in Substep 1.3, setup the swap area and repeat the experiment. What happens now?

Cleanup

IMPORTANT: At the end of the lab session:

- **Delete** all -unused - OpenStack VMs, volumes, security group rules that were created by your team.

Additional Documentation

OpenStack Horizon documentation can be found on the following pages:

- User Guide: https://docs.openstack.org/horizon/latest/