Lab 5 - Linux Memory Managment

1 – Setup & Basic Tasks

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

- nproc
- lscpu
- cat /proc/cpuinfo

Check the compiler installation (using which command?)

- gcc --version

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

Tip: use `tmux`

Look up for a cheat sheet, but here are a few useful commands:

Combination	Description
tmux	Create a tmux session
CTRL+b into c	Create a terminal
CTRL+b into %	Split current pane vertically
CTRL+b into "	Split current pane horizontally
CTRL+b into <nr>	Switch to existing terminal <nr>
CTRL+d into <nr>	Close terminal <nr>

1.1 – Basic memory under Linux

Todo 1 - In the first section 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). Add an

endless loop below this, exit, compile and run.

See MEM_lab.c - Todo 1

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)

```
$ htop
```

```
1  [||||||||||||||||||||||||||||||||||||100.0%]   Tasks: 37, 30 thr; 2 running
2  [||                                    1.3%]   Load average: 0.39 0.10 0.03
Mem[|||||||||                      153M/3.84G]   Uptime: 03:24:57
Swp[                                   0K/0K]

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 3172 ubuntu     20   0  3984   836   756 R 100.  0.0  0:20.57 ./out.t
 3173 ubuntu     20   0  8336  3800  2948 R  0.0  0.1  0:00.07 htop
 1227 ubuntu     20   0  8336  4012  3136 S  0.0  0.1  0:49.30 htop
 1063 ubuntu     20   0  8408  4740  3100 S  0.0  0.1  0:07.52 tmux
```

Tip: Press `M` to order htop by memory usage.

Column	Description
M_SIZE (VIRT)	The size of the virtual memory of the process.
M_RESIDENT (RES)	The resident set size (text + data + stack) of the process (i.e. the size of the process's used physical memory).
M_SHARE (SHR)	The size of the process's shared pages.

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

```
$ free --kilo
```

```
              total        used        free      shared  buff/cache   available
Mem:        4127260      162123     3318194         991      646942     3718672
Swap:             0           0           0
```

Tip: Use e.g. -k to display kibibytes (1024 or 2^10 bytes) instead of --kilo for kilobytes (1000 or 10^3 bytes).

Read the file /proc/${pid}/status specifically the memory related portions. What do the fields mean? (hint-man proc)

Attribute	Description
Mems_allowed	Mask of memory nodes allowed to this process.
RssAnon	Size of resident anonymous memory.
RssShmem	Size of resident shared memory.
VmLck	Locked memory size.
VmPeak	Peak virtual memory size.
VmPin	These are pages that can't be moved because something needs to directly access physical memory.
VmSize	Virtual memory size.
VmSwap	Swapped-out virtual memory size.

VmLck

The amount of memory that has been "pinned" or kept in physical memory and is not allowed to be swapped out to disk. This is often used for critical system processes or processes that require low-latency access to memory.

VmPin

The amount of memory that has been locked or "locked-in" physical memory, meaning it cannot be paged out or swapped out to disk. This is typically used for processes that need to guarantee access to a certain amount of memory, such as databases or real-time applications.

What is the difference between VmPin and VmLck?

Attribute	Description
VmLck	Locked memory size.
VmPin	These are pages that can't be moved because something needs to directly access physical memory.

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

smem is used to report memory usage with shared memory divided proportionally.

```
 PID User     Command                         Swap      USS      PSS      RSS
3172 ubuntu   ./out.t                            0      208      260     1872
3148 ubuntu   tmux                               0      596     1063     3440
3173 ubuntu   htop                               0      852     1166     3800
1227 ubuntu   htop                               0      880     1210     4016
3174 ubuntu   -bash                              0     1720     1933     5104
1085 ubuntu   -bash                              0     1720     1937     5100
3159 ubuntu   -bash                              0     1720     1937     5076
1105 ubuntu   -bash                              0     1724     1947     5092
3149 ubuntu   -bash                              0     1764     1990     5308
1095 ubuntu   -bash                              0     1776     2009     5212
3110 ubuntu   -bash                              0     1800     2029     5324
1064 ubuntu   -bash                              0     1848     2084     5456
1063 ubuntu   tmux                               0     1864     2324     4744
 901 ubuntu   /lib/systemd/systemd --user        0     1532     2838     9472
3185 ubuntu   /usr/bin/python3 /usr/bin/s        0     6244     8179    12620
```

Column	Description
USS	Unique Set Size for unshared memory
PSS	Proportional Set Size = shared + unshared memory
RSS	Resident Set Size: Held in RAM, will not swap to disk

Todo 2 - Remove the endless loop and insert code to read the page size (store it in a variable) and print it out.

See MEM_lab.c - Todo 2 and output at the end of Task 1.1.

Verify this with the getconf command

```
$ getconf PAGE_SIZE
4096
```

Todo 3 - Include code to reserve memory (hint - man malloc) the size of a number of pages.

See MEM_lab.c - Todo 3 and output at the end of Task 1.1.

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

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

See PS output at the end of Task 1.1.

Attribute	Description
min_flt	Number of minor page faults
max_flt	Number of major page faults

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.

The pages are not resident in the memory before accessing them.

Here we write random characters into the memory as displayed with the samples.

See MEM_lab.c - Todo 4 and output at the end of Task 1.1.

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

The count for page faults reset when starting the program anew as a new process starts and different memory is allocated.

See MEM_lab.c - Todo 5 (included in Part 4) and output at the end of Task 1.1.

OUTPUT: Terminal of complete program

```
Hello MEM Lab
------------ Part 1:  Simple program check memory of process
----- the PID of this process is 4700

Press enter to continue

------------ Part 2:  whats the page size
Page size: 4096
Press return to continue

------------ Part 3:  generate some memory area
Allocated 16384 bytes of memory at address 0x55c54984fac0

Press return to continue

------------ Part 4: Are these pages in memory?
Allocated 16384 bytes of memory at address 0x55c549854000
mincore:
Before buffer access:
Page 0: not in memory
Page 1: not in memory
Page 2: not in memory
Page 3: not in memory
Memory samples:
After buffer access:
Page 0: in memory
Page 1: in memory
Page 2: in memory
Page 3: in memory
Memory samples: abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijkl
Press return to continue

------------ Part 5: Lets limit the available memory
Check the number of page faults
Press return to continue

Carry out the instructions in the lab guide to limit the available memory and repeat
Bye MEM Lab
```

OUTPUT: PS of complete program

```
ubuntu@ubuntu:~/08/P05$ ps -o minflt,maj_flt 4700
MINFLT  MAJFL
    89      0
ubuntu@ubuntu:~/08/P05$ ps -o minflt,maj_flt 4700
MINFLT  MAJFL
    89      0
ubuntu@ubuntu:~/08/P05$ ps -o minflt,maj_flt 4700
MINFLT  MAJFL
    90      0
ubuntu@ubuntu:~/08/P05$ ps -o minflt,maj_flt 4700
MINFLT  MAJFL
    96      0
ubuntu@ubuntu:~/08/P05$ ps -o minflt,maj_flt 4700
MINFLT  MAJFL
 37637      0
```

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?

The following command `ulimit -a` fetches all limits for a user.

```
elia@elia-ThinkPad-X1-Nano-Gen-1:~/文档/Ausbildung/Schule/Studium/ZHAW/6. Semester/BSY/Lab/P05$ ulimit -a
real-time non-blocking time  (microseconds, -R) unlimited
core file size              (blocks, -c) 0
data seg size               (kbytes, -d) unlimited
scheduling priority                 (-e) 0
file size                   (blocks, -f) unlimited
pending signals                     (-i) 62372
max locked memory           (kbytes, -l) 2012356
max memory size             (kbytes, -m) unlimited
open files                          (-n) 1024
pipe size                (512 bytes, -p) 8
POSIX message queues         (bytes, -q) 819200
real-time priority                  (-r) 0
stack size                  (kbytes, -s) 8192
cpu time                   (seconds, -t) unlimited
max user processes                  (-u) 62372
virtual memory              (kbytes, -v) unlimited
file locks                          (-x) unlimited
```

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?

For example. If we limit a process that requires 90 MB (92'160 kB) and we limit it to 50 MB (51'200 kb) using the command `ulimit -m 51200`, the program will receive an error when trying to allocate more than 50 MB and therefore will crash.

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

- `ulimit -m unlimited`
- It might be helpful in the first time to assign unlimited memory access. But if a process has flaws, it might be dangerous.

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`

Parameter	Description
sudo	Run command with admin privileges
cgcreate	Create a cgroup
-a ubuntu:ubuntu	Define user & group to access the cgroup as admins. User: Ubuntu, Group: Ubuntu
-t ubuntu:ubuntu	Define user & group that the cgroup will apply to. User: Ubuntu, Group: Ubuntu
-t memory:myGroup	Define subsystem and cgroup name. Subsystem: Memory, Name: myGroup

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

- The following files have been created
  - memory.limit_in_bytes
  - memory.usage_in_bytes
  - memory.max_usage_in_bytes
  - memory.failcnt
- What can be used to set limitations of memory usage?

- For example we could use:
    - `sudo cgset -r memory.limit_in_bytes=500000 myGroup`

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 2M > /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.

- The process will be killed because it has exceeded the memory limit set by myGroup. OOM comes into action and kills the process.
- Use the command `dmesg` to display the system log:

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.

```
legacy_va_layout             oom_kill_allocating_task
elia@elia-ThinkPad-X1-Nano-Gen-1:/proc/sys/vm$ cat swappiness
60
elia@elia-ThinkPad-X1-Nano-Gen-1:/proc/sys/vm$ 
```

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?

The swaps swapped a lot due to swapping the swappiness configuration of the memory for more swappiness. Swap :)

```
               total        used        free      shared  buff/cache   available
Mem:         4030528      196008     2416532         972     1417988     3589212
Swap:        1048572       76800      971772
```