Operating Systems Lab 1

Introduction

Submitted by:

Laiba Shahid 2017-EE-151

Suleman Saleem 2017-EE-166

Muhammad Usman Khan 2017-EE-187

Electrical Engineering Department

University of Engineering and Technology, Lahore

Exercise 1

In this question, we will understand the hardware configuration of your working machine using the /proc filesystem.

(a) Run command more /proc/cpuinfo and explain the following terms: processor and cores.

1. Processors

Processors also known as CPU is the brain of the computer. It ensures the functioning of all components in the computer. Processors consists of two subsystems:

- Arithmetic and Logic Unit (ALU)
- Control Unit (CU).

ALU handles all arithmetic and logical operations. **Control Unit (CU)** regulates and synchronizes the operations of the computer. Moreover, there are CPU registers to store fetched instructions and the results. The computer architecture helps to determine whether the CPU can process 32bit or 64bit instructions.

2. Cores

A core is **an execution unit** of a CPU. This unit is capable of reading and executing instructions. CPU or the processor can have a single core or multiple cores. When a system has more cores, it is called a multicore system. A CPU with two cores is called a dual-core processor. A CPU with four cores is called a quad-core processor.

If we run the following command **lscpu**

we get the following information regarding the processors and devices architecture

```
laibashahid@laibashahid:~$ lscpu
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   39 bits physical, 48 bits virtual
CPU(s):                          2
On-line CPU(s) list:             0,1
Thread(s) per core:              1
Core(s) per socket:              2
Socket(s):                       1
NUMA node(s):                    1
Vendor ID:                       GenuineIntel
CPU family:                      6
Model:                           142
Model name:                      Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz
Stepping:                        9
CPU MHz:                         2903.998
```

(b) How many cores does your machine have?

2 cores

```
Core(s) per socket:              2
Socket(s):                       1
```

(c) How many processors does your machine have?

2 processors

```
CPU(s):                          2
```

(d) What is the frequency of each processor?

```
CPU MHz:                         2903.998
```

(e) How much physical memory does your system have?

The total physical memory is 5141216 kb

```
laibashahid@laibashahid:~$ cat /proc/meminfo
MemTotal:        5141216 kB
MemFree:         3635148 kB
```

(f) How much of this memory is free?

The total free memory is 3635148 kb

(g) What is total number of number of forks since the boot in the system?

```
laibashahid@laibashahid:~$ vmstat -f
          2536 forks
```

h) How many context switches has the system performed since bootup?

```
laibashahid@laibashahid:~$ man proc | grep -n "context switches"
1807:                     of  voluntary  and  involuntary context switches (since Li
nux
2948:                          The number of context switches that  the  system  und
er-
```

OR

```
muk@muk-VirtualBox:~$ more /proc/stat
cpu  5115 408 3306 117712 17708 0 551 0 0 0
cpu0 2324 284 1588 60639 7644 0 230 0 0 0
cpu1 2790 124 1718 57072 10064 0 320 0 0 0
intr 246788 29 250 0 0 0 0 0 0 0 0 0 1094 0 0 870 0 0 4371 772 2162 27374 26
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ctxt 393982
btime 1615830704
processes 1692
procs_running 1
procs_blocked 0
softirq 227172 3 71004 144 976 27856 0 278 61488 0 65423
```

EXERCISE 2

In this question, we will understand how to monitor the status of a running process using the top

command. Compile the program cpu.c given to you and execute it in the bash or any other shell of your choice as follows.

$ gcc cpu.c -o cpu

$ ./cpu

This program runs in an infinite loop without terminating. Now open another terminal, run the top command and answer the following questions about the cpu process.

```
laibashahid@laibashahid:~/Downloads/intro-code$ top

top - 20:35:09 up  1:32,  1 user,  load average: 0.46, 0.25, 0.26
Tasks: 188 total,   2 running, 186 sleeping,   0 stopped,   0 zombie
%Cpu(s): 50.8 us,  0.0 sy,  0.0 ni, 48.7 id,  0.5 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   5020.7 total,   2429.3 free,   1216.4 used,   1375.0 buff/cache
MiB Swap:   2048.0 total,   2048.0 free,      0.0 used.   3536.5 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  11686 laibash+  20   0    2364    520    452 R  99.7   0.0   0:19.57 cpu
   1290 laibash+  20   0  847536  78296  48392 S   0.7   1.5   0:35.93 Xorg
   1470 laibash+  20   0 4207432 360032 124800 S   0.3   7.0   4:39.84 gnome-+
   2390 laibash+  20   0  825192  52068  39380 S   0.3   1.0   0:08.19 gnome-+
  10758 laibash+  20   0 3416628 309552 154420 S   0.3   6.0   2:02.29 firefox
  11035 laibash+  20   0 2988468 301984 158636 S   0.3   5.9   0:45.42 Web Co+
  11687 laibash+  20   0   20832   3764   3256 R   0.3   0.1   0:00.02 top
      1 root      20   0  168944  13016   8580 S   0.0   0.3   0:04.74 systemd
      2 root      20   0       0      0      0 S   0.0   0.0   0:00.01 kthrea+
      3 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_gp
      4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_pa+
      6 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworke+
      9 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 mm_per+
     10 root      20   0       0      0      0 S   0.0   0.0   0:00.28 ksofti+
     11 root      20   0       0      0      0 I   0.0   0.0   0:01.73 rcu_sc+
     12 root      rt   0       0      0      0 S   0.0   0.0   0:00.08 migrat+
     13 root     -51   0       0      0      0 S   0.0   0.0   0:00.00 idle_i+
```

- PID = 11686
- %CPU =99.7 and %MEM = 0
- The system is in running state.

EXERCISE 3

In this question, we will understand how the Linux shell (e.g., the bash shell) runs user commands by spawning new child processes to execute the various commands.

(a) Compile the program cpu-print.c given to you and execute it in the bash or any other shell of your choice as follows.

$ gcc cpu-print.c -o cpu-print
$ ./cpu-print

This program runs in an infinite loop printing output to the screen. Now, open another terminal and use the ps command with suitable options to find out the pid of the process spawned by the shell to run the cpu-print executable.

Using the command “ps -e”

```
1747 pts/0    00:00:05 cpu-print
```

Using the command “pidof cpu-print”

```
muk@muk-VirtualBox:~$ pidof cpu-print
1747
```

(b) Find the PID of the parent of the cpu-print process, i.e., the shell process. Next, find the PIDs of all the ancestors, going back at least 5 generations (or until you reach the init process).

Using the command “pstree -s -p <pid>”

```
muk@muk-VirtualBox:~$ pstree -s -p 1747
systemd(1)───systemd(920)───gnome-terminal-(1593)───bash(1603)───cpu-print(17+
```

(c) We will now understand how the shell performs output redirection. Run the following command.

/cpu-print > /tmp/tmp.txt &

Using the command given above, the foolwing result was obtained. In Linux shell, we can read from the file or write to file as well. In this command line, cpu-print’s output going to tmp.txt file.

```
muk@muk-VirtualBox:~/Documents/Lab1/intro-code$ ./cpu-print > /tmp/tmp.txt &
[1] 1844
muk@muk-VirtualBox:~/Documents/Lab1/intro-code$ ./cpu-print > /tmp/tmp.txt &
[2] 1845
muk@muk-VirtualBox:~/Documents/Lab1/intro-code$ ./cpu-print > /tmp/tmp.txt &
[3] 1846
```

(d) Run the following command “./cpu-print | grep hello &”. Use this information to explain how pipes are implemented by the shell.

```
muk@muk-VirtualBox:~/Documents/Lab1/intro-code$ ./cpu-print | grep hello &
[4] 1850
muk@muk-VirtualBox:~/Documents/Lab1/intro-code$ ./cpu-print | grep hello &
[5] 1852
muk@muk-VirtualBox:~/Documents/Lab1/intro-code$ ./cpu-print | grep hello &
[6] 1854
```

The output of one program as the input of another program without storing anything in the temporary file, pipe can be utilized. Pipes can run Multiple commands in a single command line. Grep command is used to search files matching words or patterns. Which is "Hello" in this case.

(e) Consider the following commands that you can type in the bash shell: cd, ls, history, ps. Which of these commands already exist as built-in executables in the Linux kernel that are then simply executed by the bash shell, and which are implemented by the bash code itself?

Built-in Commands: cd and history
Not Built-in Commands: ls, ps

```
muk@muk-VirtualBox:~/Documents/Lab1/intro-code$ type ls
ls is aliased to `ls --color=auto'
muk@muk-VirtualBox:~/Documents/Lab1/intro-code$ type cd
cd is a shell builtin
muk@muk-VirtualBox:~/Documents/Lab1/intro-code$ type history
history is a shell builtin
muk@muk-VirtualBox:~/Documents/Lab1/intro-code$ type ps
```

EXERCISE 4

> Compare the virtual and physical memory usage of both programs, and explain your observations. You can also inspect the code to understand your observations.

For Memory 1:

```
muk@muk-VirtualBox:~$ cd Documents
muk@muk-VirtualBox:~/Documents$ cd Lab1
muk@muk-VirtualBox:~/Documents/Lab1$ cd intro-code
muk@muk-VirtualBox:~/Documents/Lab1/intro-code$ gcc memory1.c -o memory1
muk@muk-VirtualBox:~/Documents/Lab1/intro-code$ ./memory1


Program : 'memory_1'
--------------------


PID : 1701
Size of int : 4

Press Enter Key to exit.
```

```
muk          1701  0.0  0.0   6284  4948 pts/0    S+   01:16   0:00 ./memory1
```

Virtual Mem: 6284 bytes

RSS: 4948 bytes

For Memory 2:

```
muk@muk-VirtualBox:~/Documents/Lab1/intro-code$ gcc memory2.c -o memory2
muk@muk-VirtualBox:~/Documents/Lab1/intro-code$ ./memory2


Program : 'memory_2'
--------------------


PID : 1834
Size of int : 4

Press Enter Key to exit.
```

```
muk          1834  0.0  0.0   6284  4928 pts/0    S+   01:17   0:00 ./memory2
```

Virtual Mem: 6284 bytes

RSS: 4928 bytes

Observation: It is observed that program requires more physical memory than allocated. As virtual memory exceeds in both cases.

Exercise 5

In this question, you will compile and run the programs disk.c and disk1.c given to you.

Firstly, mkdir was used to make a new folder ‘disk-files’, the foo.pdf was copied and placed into the ‘disk-files’. Then make-copies.sh was run to make 5000 copies. Finally, turn by turn disk.c and disk1.c were compiled and using iotop, the required info was obtained as required separately on another terminal.

```
muk@muk-VirtualBox:~$ cd Documents
muk@muk-VirtualBox:~/Documents$ cd Lab1
muk@muk-VirtualBox:~/Documents/Lab1$ ls
intro-code  intro-code.tgz  intro.pdf
muk@muk-VirtualBox:~/Documents/Lab1$ cd intro-code
muk@muk-VirtualBox:~/Documents/Lab1/intro-code$ ls
cpu    cpu-print    disk1.c  foo.pdf          memory1     me
mory2.c
cpu.c  cpu-print.c  disk.c   make-copies.sh  memory1.c
muk@muk-VirtualBox:~/Documents/Lab1/intro-code$ mkdir disk
-files
muk@muk-VirtualBox:~/Documents/Lab1/intro-code$ ls
cpu    cpu-print    disk1.c  disk-files  make-copies.sh  m
emory1.c
cpu.c  cpu-print.c  disk.c   foo.pdf     memory1         m
emory2.c
muk@muk-VirtualBox:~/Documents/Lab1/intro-code$ cp foo.pdf
 ./disk-files
muk@muk-VirtualBox:~/Documents/Lab1/intro-code$ rm foo.pdf
muk@muk-VirtualBox:~/Documents/Lab1/intro-code$ ls
cpu    cpu-print    disk1.c  disk-files      memory1     me
mory2.c
cpu.c  cpu-print.c  disk.c   make-copies.sh  memory1.c
```

This is for disk.c :

```
muk@muk-VirtualBox:~/Documents/Lab1/intro-code$ rm foo.pdf
muk@muk-VirtualBox:~/Documents/Lab1/intro-code$ ls
cpu    cpu-print    disk1.c  disk-files      memory1     me
mory2.c
cpu.c  cpu-print.c  disk.c   make-copies.sh  memory1.c
muk@muk-VirtualBox:~/Documents/Lab1/intro-code$ ./make-cop
ies.sh
muk@muk-VirtualBox:~/Documents/Lab1/intro-code$ gcc disk.c
 -o disk
muk@muk-VirtualBox:~/Documents/Lab1/intro-code$ ./disk
```

Using “sudo iotop” command,

```
Total DISK READ:      1712.98 K/s | Total DISK WRITE:        0.00 B/s
Current DISK READ:    1712.98 K/s | Current DISK WRITE:      0.00 B/s
   TID  PRIO  USER     DISK READ  DISK WRITE  SWAPIN     IO>    COMMAND
  8344 be/4 muk       1712.98 K/s    0.00 B/s  0.00 % 44.58 % ./disk
```

This is for disk1.c :

```
muk@muk-VirtualBox:~/Documents/Lab1/intro-code$ gcc disk1.
c -o disk1
muk@muk-VirtualBox:~/Documents/Lab1/intro-code$ ./disk1
```

Using “sudo iotop” command,

```
Total DISK READ:          0.00 B/s | Total DISK WRITE:         0.00 B/s
Current DISK READ:        0.00 B/s | Current DISK WRITE:       0.00 B/s
    TID  PRIO  USER     DISK READ  DISK WRITE  SWAPIN     IO>    COMMAND
      1 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % init splash
      2 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [kthreadd]
      3 be/0 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [rcu_gp]
      4 be/0 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [rcu_par_gp]
      6 be/0 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [kworker~kblockd]
      8 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [kworker~unbound]
      9 be/0 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [mm_percpu_wq]
     10 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [ksoftirqd/0]
     11 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [rcu_sched]
     12 rt/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [migration/0]
     13 rt/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [idle_inject/0]
     14 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [cpuhp/0]
     15 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [cpuhp/1]
     16 rt/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [idle_inject/1]
     17 rt/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [migration/1]
     18 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [ksoftirqd/1]
     20 be/0 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [kworker~kblockd]
     21 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [kdevtmpfs]
     22 be/0 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [netns]
     23 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [rcu_tasks_kthre]
     24 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [rcu_tasks_rude_]
     25 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [rcu_tasks_trace]
     26 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [kauditd]
     27 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [khungtaskd]
```