

CS 314: Operating Systems Laboratory

Lab 1

Nampally Pranav
190010026

Answer 1

1.a

The term "processor" tells us the processor number in the system(index of processor). My laptop has 0 to 5 processors. you can see the ans1.txt[output of given command] and ans1(i).txt[output of lscpu] for reference.

The term "cores" tells us the number of cores present in a processor of a given index.

1.b

My computer has 6 cores.

1.c

My computer has 6 processors.

1.d

The frequencies of each processor are as follows:

Processor No.	Frequency(in MHz)
processor 0	2591.998
processor 1	2591.998
processor 2	2591.998
processor 3	2591.998
processor 4	2591.998
processor 5	2591.998

Table 1: Frequencies of each processor

1.e

The physical memory on my computer is: 8144948 KB. Found by running the command: more /proc/meminfo .

1.f

Free memory on my computer is: 570884 kB. Found by running the command: `more /proc/meminfo`

1.g

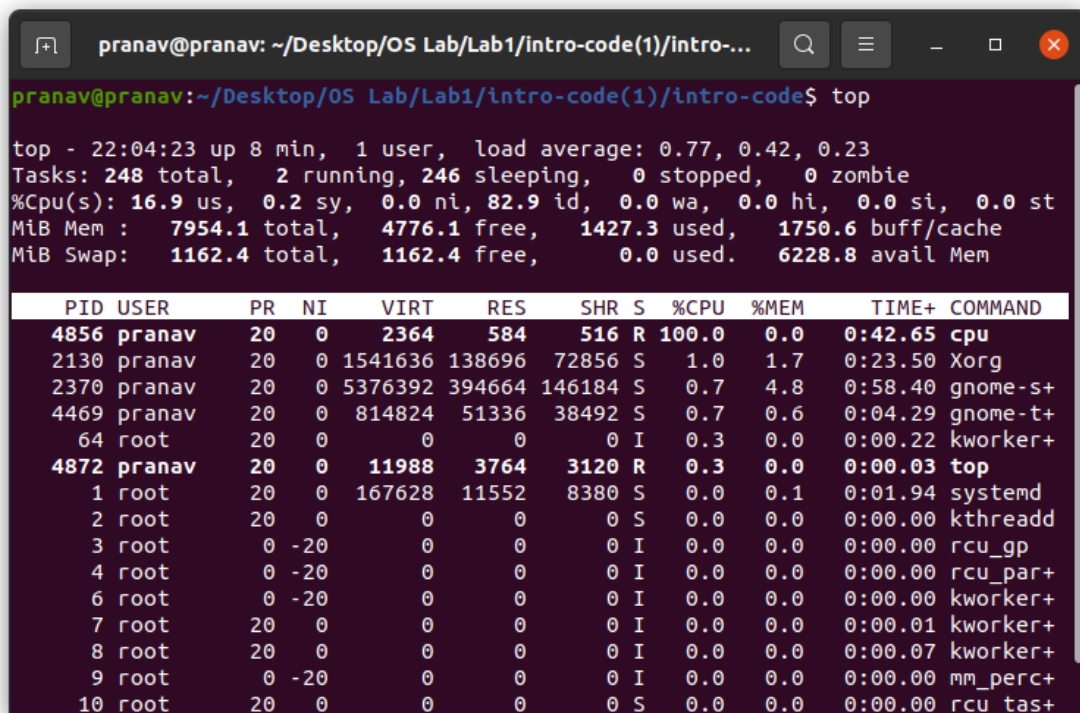
Total no. of forks since boot: 54468. Found by running the command: `more /proc/stat` and checking the value beside processes.

1.h

Total no. of context switches since boot: 23708093. Found by running the command: `more /proc/stat` and checking the value beside ctxt.

Answer 2

I ran the `cpu.c` program in parallel when I have run the `top` command in another terminal. The below Figure 2 shows the output received.



```
pranav@pranav: ~/Desktop/OS Lab/Lab1/intro-code(1)/intro-code$ top

top - 22:04:23 up 8 min, 1 user, load average: 0.77, 0.42, 0.23
Tasks: 248 total, 2 running, 246 sleeping, 0 stopped, 0 zombie
%Cpu(s): 16.9 us, 0.2 sy, 0.0 ni, 82.9 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 7954.1 total, 4776.1 free, 1427.3 used, 1750.6 buff/cache
MiB Swap: 1162.4 total, 1162.4 free, 0.0 used, 6228.8 avail Mem

  PID USER      PR  NI   VIRT   RES   SHR  S  %CPU  %MEM    TIME+  COMMAND
 4856 pranav    20   0   2364    584    516  R 100.0   0.0   0:42.65  cpu
 2130 pranav    20   0 1541636 138696 72856  S   1.0   1.7   0:23.50  Xorg
 2370 pranav    20   0 5376392 394664 146184  S   0.7   4.8   0:58.40  gnome-s+
 4469 pranav    20   0 814824  51336 38492  S   0.7   0.6   0:04.29  gnome-t+
   64 root       20   0     0     0     0  I   0.3   0.0   0:00.22  kworker+
 4872 pranav    20   0  11988   3764   3120  R   0.3   0.0   0:00.03  top
    1 root       20   0 167628  11552  8380  S   0.0   0.1   0:01.94  systemd
    2 root       20   0     0     0     0  S   0.0   0.0   0:00.00  kthreadd
    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_par+
    6 root        0 -20     0     0     0  I   0.0   0.0   0:00.00  kworker+
    7 root       20   0     0     0     0  I   0.0   0.0   0:00.01  kworker+
    8 root       20   0     0     0     0  I   0.0   0.0   0:00.07  kworker+
    9 root        0 -20     0     0     0  I   0.0   0.0   0:00.00  mm_perc+
   10 root       20   0     0     0     0  S   0.0   0.0   0:00.00  rcu_tas+
```

Figure 1: Answer 2

2.a

The PID of the process running `cpu` command is: 4856.

2.b

CPU consumed: 100%

Memory consumed: 0%

2.c

The command is in running state as shown by "R+" under Status STAT when running command: `ps -aux`. Also it show "R" in above Figure 2.

Answer 3

Files for reference: `ans3.txt` shows output of "`ps axjf`".

3.a

The PID of `cpu-print` command is: 55635. This is found by running the command: "`ps -e`" (or) "`ps -ely`".

3.b

To get PID of parent processes we run the command: "`ps axjf`".

The past 5 generation parent PIDs are: (`./cpu-print`)55635 => 54551 => 54544 => 2042 => 1(INIT process)

Parent PID	PID	Command
54551	55635	<code>./cpu-print</code>
54544	54551	<code>bash</code>
2042	54544	<code>/usr/libexec/gnome-terminal-server</code>
1	2042	<code>/lib/systemd/systemd -user</code>
-	1	INIT Command

Table 2: Parent PIDs

3.c

Process ID: 56457 for the process created when we run the given command. The file descriptor values can be found by checking properties for the files in `/proc/56457/fd`.

File descriptors	Target
2(Standard Error)	<code>/dev/pts/0</code>
1(Standard Output)	<code>/tmp/tmp.txt</code>
0(Standard Input)	<code>/dev/pts/0</code>

This means that the standard input for this process is the terminal (`/dev/pts/0`) and the output for this process is given at `/tmp/tmp.txt`. So, the shell handles file redirection by changing the file descriptors (present inside the `fd` dir. which is inside `process_id` dir. which is in `proc` dir.).

3.d

Newly spawned Process IDs are : 57356, 57357 for the processes created when we run the given command. The file descriptor values can found by checking properties for the files in `/proc/57356/fd` and `/proc/57357/fd`.

For PID: 57356

File descriptors	Target
0(Standard Input)	<code>/dev/pts/0</code>
1(Standard Output)	pipe:[669293]
2(Standard Error)	<code>/dev/pts/0</code>

For PID: 57357

File descriptors	Target
0(Standard Input)	pipe:[669293]
1(Standard Output)	<code>/dev/pts/0</code>
2(Standard Error)	<code>/dev/pts/0</code>

We see that the output of the first process(57356) is given at pipe:[669293] which then goes as input to the second process which is 57357. Thus, the pipelining works by interlinking the targets of the file descriptors of the processes as done above.

3.e

By running the "which" command for each of the given 4 commands we can say the following:

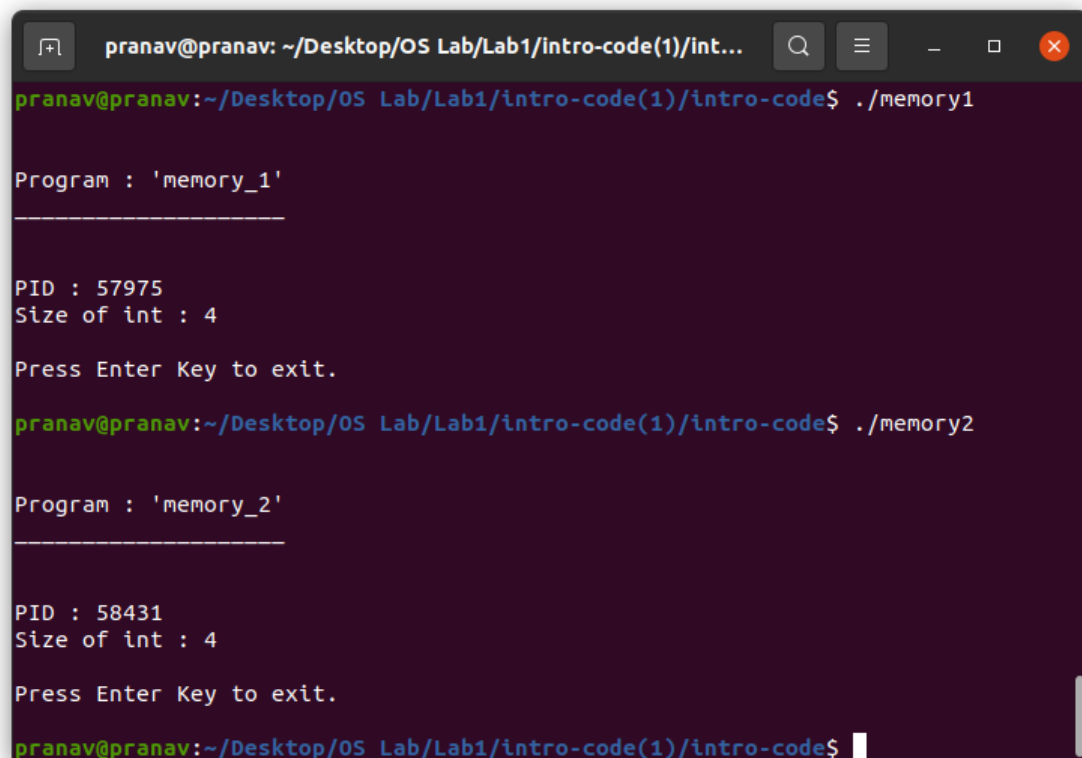
The commands : `ls` and `ps` are executed by the bash shell.(built-in executables exist for these)

The commands: `history` and `cd` are implemented by the bash code itself

The command "which" for `ls` returns: `/usr/bin/ls`, which for `ps` returns: `/usr/bin/ps`. Whereas the which command doesn't returns anything for `history`, `cd` commands. Thus showing that built-in executables exist for `ls` and `ps`.

Answer 4

Files for reference: `ans4aa.txt` has "ps -aux" output for `memory1.c` and `ans4bb.txt` has "ps -aux" output for `memory2.c`.

A terminal window with a dark background and light-colored text. The window title is 'pranav@pranav: ~/Desktop/OS Lab/Lab1/intro-code(1)/int...'. The prompt is 'pranav@pranav:~/Desktop/OS Lab/Lab1/intro-code(1)/intro-code\$'. The user enters './memory1'. The output shows 'Program : 'memory_1'', a separator line, 'PID : 57975', 'Size of int : 4', and 'Press Enter Key to exit.'. The user then enters './memory2'. The output shows 'Program : 'memory_2'', a separator line, 'PID : 58431', 'Size of int : 4', and 'Press Enter Key to exit.'. The prompt is now 'pranav@pranav:~/Desktop/OS Lab/Lab1/intro-code(1)/intro-code\$' with a cursor.

```
pranav@pranav:~/Desktop/OS Lab/Lab1/intro-code(1)/intro-code$ ./memory1

Program : 'memory_1'
-----

PID : 57975
Size of int : 4

Press Enter Key to exit.

pranav@pranav:~/Desktop/OS Lab/Lab1/intro-code(1)/intro-code$ ./memory2

Program : 'memory_2'
-----

PID : 58431
Size of int : 4

Press Enter Key to exit.

pranav@pranav:~/Desktop/OS Lab/Lab1/intro-code(1)/intro-code$
```

Figure 2: Answer 4

The command used is: "ps -aux". And the output I received is given below by running each of the programs.

```
memory1.c:
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
pranav    57975  0.0  0.0   6280  4860 pts/0    S+   19:06   0:00 ./memory1
```

```
memory2.c:
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
pranav    58431  0.0  0.0   6280  4940 pts/0    S+   19:21   0:00 ./memory2
```

The VSS term shows us the Virtual Memory size used by the process. And the RSS shows us the Resident Memory used by the process.

Here we see that memory1.c and memory2.c occupy the same amount of Virtual memory, whereas the Resident Memory(Physical memory) used is greater for memory2.c than for memory1.c.

This is because the memory.c program just declares an array and just prints some meta to the terminal (so only some physical memory for array declaration is taken). Whereas, the memory2.c program also does the work of initializing the values in the array and also changes inside the array, so basically it does an extra work of adding data to array (as compared to memory1.c) this causes it to use more memory in physical RAM as compared to memory1.c.

Answer 5

The command used to run the bash script is: `bash make-copies.sh`

As for the main question, I ran the `iostat` and `top` in terminals besides the one where the `disk.c` and `disk1.c` were being run.

We find that for `disk.c` utilizes 99.7% of the CPU, whereas the `disk1.c` utilizes just 62.8% of CPU. We also see that for `disk.c` the `kB_read` goes on increasing in 10000s of kB this is because the `disk.c` selects the files at random and reads them. Whereas the `disk1.c` has increment in `kB_read` value in just 1000kB and after sometime it stagnates, this is because the `disk1.c` just accesses a single file to read, once it finds it the file goes into buffer thus showing lower CPU utilization and `kB_read` values.