CS 314: Operating Systems Laboratory
Lab 1

Nampally Pranav
190010026

Answer 1

1.a

The term "processor" tells the us the processor number in the sysytem(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.

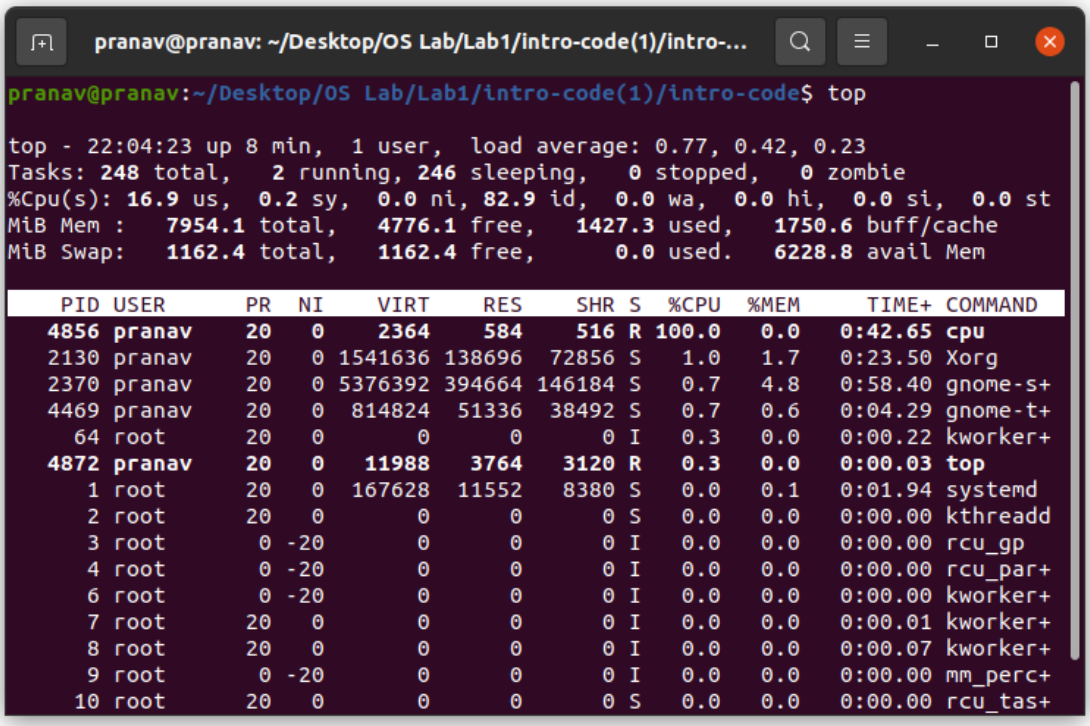

Figure 1: Answer 2

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 fornd 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 => 2024 => 1(INIT process)

Parent PID	PID	Command
54551	55635	./cpu-print
54544	54551	bash
2042	54544	/usr/libexec/gnome-terminal-server
1	2042	/lib/systemd/systemd –user
-	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 found by checking properties for the files in /proc/56457/fd.

```
|       File descriptors          |    Target        |
|=================================================|
|      2(Standard Error)          |  /dev/pts/0      |
|      1(Standard Output)         |  /tmp/tmp.txt    |
|      0(Standard Input)          |  /dev/pts/0      |
```

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)          |  /dev/pts/0      |
|      1(Standard Output)         |  pipe:[669293]   |
|      2(Standard Error)          |  /dev/pts/0      |

For PID: 57357
|       File descriptors          |    Target        |
|=================================================|
|      0(Standard Input)          |  pipe:[669293]   |
|      1(Standard Output)         |  /dev/pts/0      |
|      2(Standard Error)          |  /dev/pts/0      |
```

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.

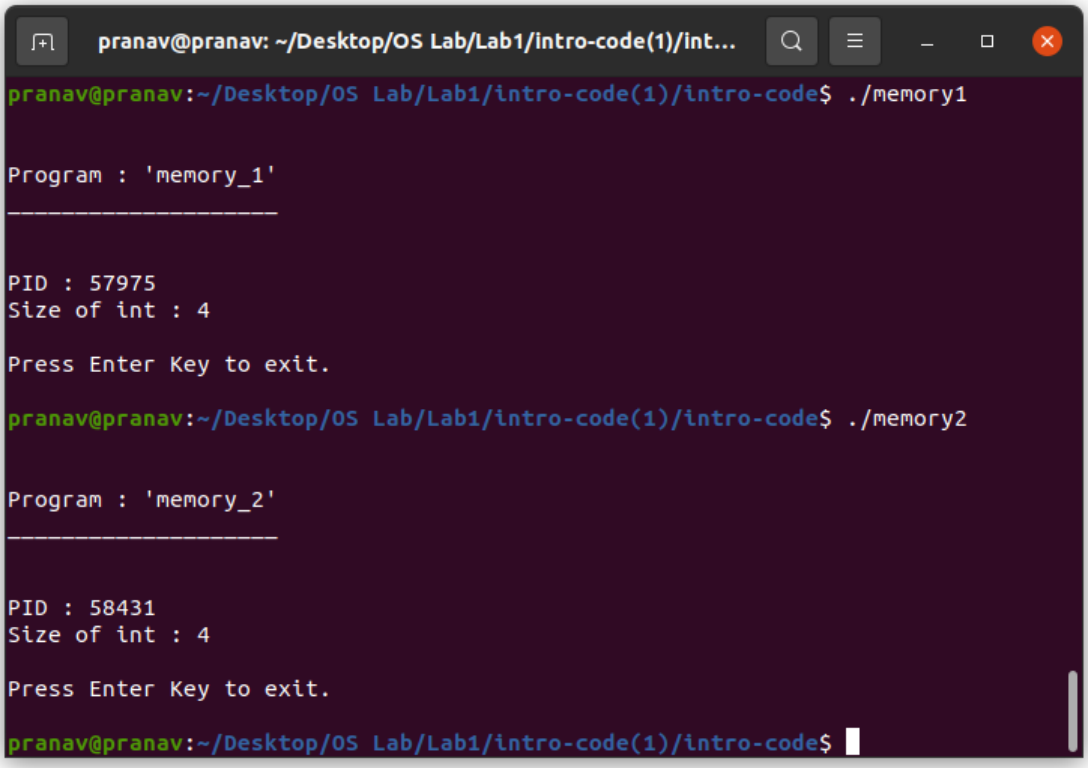

Figure 2: Answer 4

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 kBs 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 1000kBs 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.