# Report LAB 1

130050032 Amit Malav
130050047 Lokit kumar paras.

1) No of CPU cores = 4 (using cat /proc/cpuinfo )
Total Memory = 8142244 kB ( using cat /proc/meminfo)
Memory Free = 1541904 kB
Fraction free = 0.1893
Context switches = 788395240 ( using cat/proc/stat as ctxt)
Processes forked = 407995

2)

cpu1

Bottleneck - cpu
Commands - top.
Output -Cpu usage is always almost 100%
Justification - It exhausts cpu resourses allocated to it(ie. cpu usage is always almost 100%). This is because the code is doing many calculations and
the context switches that happen are almost all Involuntary switches.

cpu1print

Bottleneck - cpu
Commands - top.
Output - Cpu usage gets maxed out for terminal process.
Justification - The code is printing text on the gnome terminal window. Even though the CPU can send such strings at high speed the printing on terminal takes time and hence becomes the bottle neck.

cpu2

Bottleneck - cpu
Commands - top.
Output -Cpu usage is always almost 100%
Justification - Here again the CPU usage is 100% as it uses gettimeoftheday() function which majorly works in the userspace and keeps the CPU busy. Get timeoftheday() uses Virtual Dynamically-linked Shared Object, a kernel-provided shared library that helps userspace perform a few kernel actions without the overhead of a system call, as well as automatically choosing the most efficient syscall mechanism.

disk

Bottleneck - disk
Commands - iostat -x 1
Output- disk usage becomes 100%
Justification - The program is reading files from a folder/disk. Since it takes time to open a file and read its contents, so continuously accessing the hard disk for a lot of files one after the other, makes disk usage as the bottle neck.

disk1

Bottleneck - cpu
Commands - top and iostat -x 1
Output- disk usage is almost 0% and CPU usage is 100%.
Justification - The program is reading the same file from a folder/disk again and again. So now the contents of the file are present

in the cache memory already, hence loading from disk is no longer the bottleneck, but reading of contents from cache utilises CPU and hence CPU becomes the bottleneck now.

3)

Command used - /proc/$pid/stat

```
perl -MPOSIX -l -0777 -ne '@f = /\(.*\)|\S+/gs;
printf "utime: %.2f\nstime: %.2f\n",
 map {$_/POSIX::sysconf( &POSIX::_SC_CLK_TCK )}@f[13..14]'
"/proc/$pid/stat"
```

Cpu1

User Time - 13.85

Kernel Time - 0.00

Observation- Spends more time in user mode and negligible time in kernel mode

Justification- The code is doing only computations,and no system calls are made, hence the process never switches to kernel mode voluntarily.

Cpu1print

User Time - 0.21

Kernel Time - 1.56

Observation- Spends more time in kernel mode and less time in user mode.

Justification- The code is making systemcall "printf" hence it constantly keeps switching to kernel mode voluntarily.

Cpu2

User Time - 12.38

Kernel Time - 0.01

Observation- This process also spends maximum time in user mode despite making system calls.

Justification- The reason for less kernel time despite gettimeoftheday() being a system call is because excultion gettimeofday occurs through Virtual Dynamically-linked Shared Object, a kernel-provided shared library that helps userspace perform a few kernel actions without the overhead of a system call, as well as automatically choosing the most efficient syscall mechanism.

4) Command used - cat /proc/[pid]/status

Cpu1

voluntary_ctxt_switches = 1

Involuntary cntxt switches = 6027

Disk

voluntary_ctxt_switches = 14493

Involuntary cntxt switches = 3648

Justification : The number of voluntary context switches in 'disk' program are more as it waits for disk I/O hence it needs to voluntarily leave the processor until it gets response from disk and I/O is possible. Whereas there is no such disk I/O involved in 'Cpu1' program, hence it hardly leaves CPU voluntarily. And since involuntary context switches are random and vary from time to time, no such observation can be made on it.