**OS Lab Assignment- 1**

**Familiarize yourself with proc, vmstat, nproc, ps, top, grep commands using man and help**

1. **Collect the following basic information about your machine using proc. How many CPU cores does the machine have? How much memory, and what fraction of it is free? How many context switches has the system performed since bootup? How many processes has it forked since bootup?**

Use ‘nproc’ and ‘vmstat’.

Can also use:

cat /proc/cpuinfo :

cat /proc/meminfo

cat /proc/stat

**2. Every process consumes some resources (CPU, memory, disk or network bandwidth, and so on). When a process runs as fast as it can, one of these resources is fully utilized, limiting the maximum rate at which the process can make progress. Such a resource is called the bottleneck resource of a process. A process can be bottlenecked by different resources at different points of time, depending on the type of work it is doing.**

**Run each of the four programs (cpu, cpu-print, disk, and disk1) separately, and identify what the bottleneck resource for each is (without reading the code). For example, you may monitor the utilizations of various resources and see which ones are at the peak. Next, read through the code and justify how the bottleneck you identified is consistent with what the code does.**

**For each of the programs, you must write down three things: the bottleneck resource, the reasoning that went into identifying the bottleneck, (e.g., the commands you ran, and the outputs you got), and a justification of the bottleneck from reading the code.**

Commands used: top, lscpu, ifconfig, vmstat, ps –p PID, df -H

**3. Recall that every process runs in one of two modes at any time: user mode and kernel mode. It runs in user mode when it is executing instructions / code from the user. It executes in kernel mode when running code corresponding to system calls etc.**

**Compare (qualitatively) the programs cpu and cpu-print in terms of the amount of time each spends in the user mode and kernel mode, using information from the proc file system. For examples, which programs spend more time in kernel mode than in user mode, and vice versa? Read through their code and justify your observations.**

Instead create two programs, say helloworld.c (to print Hello world) and testadd.c (to add two numbers)

Compile these two using ‘gcc’ and create its object

Then, use ‘time’ command

**4. Recall that a running process can be interrupted for several reasons. When a process must stop running and give up the processor, it’s CPU state and registers are stored, and the state of another process is loaded. A process is said to have experienced a context switch when this happens. Context switches are of two types: voluntary and involuntary. A process can voluntarily decide to give up the CPU and wait for some event, e.g., disk I/O. A process can be made to give up its CPU forcibly, e.g., when it has run on a processor for too long, and must give a chance to other processes sharing the CPU. The former is called a voluntary context switch, and the latter is called an involuntary context switch.**

**Compare the programs cpu and disk in terms of the number of voluntary and involuntary context switches. Which program has mostly voluntary context switches, and which has mostly involuntary context switches? Read through their code and justify your observations.**

Use ps and then

Grep ctxt /proc/PID/status

**5. Open a bash shell. Find its pid. Write down the process tree starting from the first init process (pid = 1) to your bash shell, and describe how you obtained it. You may want to use the pstree command.**

* To open the bash shell

Sudo bash or **/bin/bash**, sudo bash will ask for password

* To get the process id

After entering bash mode :

1. $ps

It will display the many process id’s and one of them will be of bash

* Use ‘pstree’

**6. Consider the following commands that you can type in the bash shell: cd, ls, history, ps. Which of these are system programs that are simply executed by the bash shell, and which are implemented by the bash code itself?**

Use command ‘type’

**7. Run the following command in bash.**

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

**Find out the pid of the new process spawned to run this command. Go to the proc folder of this process, and describe where its I/O file descriptors 0, 1, 2 are pointing to. Can you describe how I/O redirection is being implemented by bash?**

1. Instead of cpu-print write your own c programeg, test.c, compile it using gcc –o test test.c and simply run it using ./test to check output
2. Now redirect the output of ./test to /tmp/tmp.txt

Using ./test > /tmp/tmp.text

1. Check whether the tmp.txt file exists in /tmp directory or not.

$ cd /tmp

…..tmp$ ls

Now you will see tmp.txt in /tmp directory

1. Type cd to come back .
2. Now type ./test > tmp/tmp.txt & echo $$

It will give two process id in which pick up last process id (ppid) for further processing

Ex: [1] 4502

4428<- it is parent process id use it.

Then, use cat command to check results

**8. Run the following command with cpu-print.**

**$./cpu-print | grep hello**

**Once again, identify which processes are spawned by bash, look at the file descriptor information in their proc folders, and use it to explain how pipes work in bash.**

Instead of cpu-print, create a new file and save the word ‘hello’ in it many times and use grep command in pipelining.