**OS Lab Assignments**

**Assignment 1**

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?**

Ans

1. cat /proc/cpuinfo :
2. cat /proc/meminfo
3. cat /proc/PID/status (where PID will be noted using ‘top’ command)
4. vmstat -f

**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.**

Ans

* compile the program using ‘gcc cpu.c’ (Use ‘cpu.c’ program send via Email)
* check the output using ‘./a.out’
* To know the usage of CPU, memory and shell of particular process---

ps –p PID –o %cpu,%mem, cmd (where PID will be noted using ‘top’ command)

**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.**

**Ans.**

* compile the program using ‘gcc swap.c’ (Use ‘swap.c’ program send via Email)
* run the program and check the output using ‘./a.out’
* 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.**

Ans:

* compile the program using ‘gcc cpu.c’ (Use ‘cpu.c’ program send via Email)
* run the program and check the output using ‘./a.out’
* Open another terminal and run:

cat /proc/PID/status (where PID will be noted using ‘top’ command)

**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 admin password)

* **To get the process id**

Enter in bash mode:

* + sudo bash (sudo bash will ask for admin password)
  + ps

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

* pstree –p : To display the process starting with init (), along with process id

**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?**

Ans

Read about internal and external commands.

And type following command to know whether a command is internal (simple executed by shell) or external (by kernel).

type command name

**EXAMPLE:** type cd : It is a shell built in (Internal command executed by bash shell)

**EXAMPLE:** type cat : cat is /bin/cat (external command showing its path)

**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 program eg, cpu.c, compile it using ‘gcc cpu.c’ and simply run it using ‘./a.out’ to check output
2. Now redirect the output of ./a.out to /tmp/tmp.txt using ./cpu > /tmp/tmp.text
3. Now type ./cpu >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.

1. $ ls –l /proc/4428/fd (where 4428 is the PID)

It will show u three file descriptor 0 (standard input), 1(standard output), 2 (standard error). These would be pointing to their device.

**TROUBLESHOOTING:**

1. Instead of cpu-print write your own c program eg, cpu.c, compile it using ‘gcc cpu.c’ and simply run it using ‘./a.out’ to check output
2. Now redirect the output of ./a.out to /tmp/tmp.txt using ./cpu > /tmp/tmp.text
3. Check whether the tmp.txt file exists in /tmp directory or not.

$ cd /tmp

…..tmp$ ls

Now you will see tmp.txt in /tmp dir

1. Type $ cd to come back.
2. Now type ./cpu >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.

1. $ ls –l /proc/4428/fd (where 4428 is the PID)

It will show u three file descriptor 0 (standard input), 1(standard output), 2 (standard error). These would be pointing to their device.

**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.**

**Ans.**

1. Instead of cpu-print write your own c program eg, swap.c, compile it using ‘gcc swap.c’ and simply run it using ‘./a.out’ to check output
2. Now type ./swap | grep hello

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.

1. $ ls –l /proc/4428/fd (where 4428 is the PID)

It will show u three file descriptor 0 (standard input), 1(standard output), 2 (standard error). These would be pointing to their device.