IIT Bhubaneswar
School of Electrical Sciences
Operating Systems Lab (0-0-3)
Spring 2017
Lab Schedule: Thu (8:00-10:55)

Instructor: Debi Prosad Dogra (dpdogra@iitbbs.ac.in)
Teaching Assistant: Santosh K. K. (sk47@iitbbs.ac.in)

Laboratory 1 (**Submission Deadline: 15/1/2017 midnight**)

Points: **35**

Objectives: The goal of this lab is to give basic introduction to the OS, shell, and processes in general.

Practice During Laboratory Hours:

- Understanding the **proc** file system of linux. A command "**man proc**" will give you a lot of information. Understand the following files of a process: **cmdline, cpu, cwd,environ, exe, fd, maps, mem, root, stat, statm, status** through the man page of **proc**.
- See the contents of each of the files and understand their meaning.
- Learn how to use the following basic commands:
  -top tells you information about processes and their resource consumption.
  -ps tells you information about all processes in the system. **ps** has several useful command line arguments. For example, **ps -A** gives you information about all processes in the system.
  -**iostat** gives information about disk utilization. Understanding various useful command line arguments (**-c -x -m** etc.).
- Four simple C programs are provided to you: **cpu.c, cpu-print.c, disk.c, and disk1.c**. Create the cpu executable as follows for further use.
- The disk programs read a large number of files, and you must create and setup these files before you run the program. Create several such files (containing junk values, each of size 2MB) in a directory.
- Read briefly about **disk I/O and buffer caches**. When files are read from a hard disk, they are cached in memory. Cache is limited in size. Thus, when you read and write a large number of files in a short span, most files will have to be accessed from the disk, as the cache can only hold a small subset of these files. However, if you are reading only one small file multiple times, then that file will most likely be served from the cache after its first access.

Take Home Assignments for Submission:

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

ii. 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.c, cpu-print.c, disk.c, and disk1.c) 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.

iii. 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 program spends more time in kernel mode than in user mode, and vice versa? Read through their code and justify your observations.

iv. 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.

v. 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 can use the **pstree** command.

vi. 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.

vii. 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.

Submission Guidelines: One PDF file containing answers for each part of the assignment. Explain in your own words. Copying will affect your grade heavily. Electronic submission has to be sent to the TA and hard-copy (laboratory book) by **19/01/2017**.

Additional Reading: Read through **Chapter 0** in the **xv6** textbook, especially the description about how the **xv6** shell works.