ECS 408 : Assignment 1

Sourav Yadav 17285

January 19, 2021

Exercises

1.
(a) : Processor represents each logical CPU with a unique processor number. A logical CPU can be a hyperthreaded sibling, a shared core in dual/quad core, or a separate physical CPU.

If number of cores equal number of processors that means hyperthreading is disabled.

Core is a independent physical unit that implements a processor. Virtual cores are called threads. Eg. A PC with 4 cores can use hyper-threading to provide eight threads.

(b) : Number of Cores = 4

(c) : Number of Processors = 8

(d) : Frequency of Each Processor = 2.80Ghz

(e) : Physical Memory = 8066652 kB

(f) : Memory free = 1958060 kB

(g) : Total number of forks since bootup = 18850 forks

(h) : Context switches since bootup = 1496333

2.
(a) PID of process = 4233

(b) CPU = Nearly 100% Mem = Nearly 0%

(c) State = Running (R)

3.
(a)

(b) PID of main cpu-print process = 4722
PID of Parent (bash) = 2207
PID of ancestor (guake) = 2015
PID of ancestor (gnome-session-b) = 1783
PID of ancestor (systemd) = 1516
PID of ancestor (systemd) = 1

(c) After running ./cpu-print > /tmp/tmp.txt & which is equivalent to running ./cpu-print 1> /tmp.txt &, we see that the file descriptor 1(stout) now point to /tmp/tmp.txt, sending all output to /tmp/tmp.txt.

(d) In linux, pipe(2) means allocating a file descriptor such that it is reading end of one and writing end of the other process. Like in our case, running ./cpu-print — grep hello & creates two different processes with file descriptor 2(stdout) of ./cpu-print and file descriptor 1(stdin) of grep hello pointing to the same file.

(e) Simple checking /bin/"command" will return commands which have executables.
ps and ls are built-in executables in the Linux kernel.
cd and history executed by bash shell.

4.
Memory1 : Virtual - 6268K Real - 5200K

Memory2 : Virtual - 6268K Real - 5172K

We see that both program consumes nearly the same memory and it was expected, both used nearly same amount of space for the array. Memory2.c has a greater time complexity but the space required is the same.

5.
Before running disk, Read - 595.05kBps Write - 2046.65kBps,
After runnning disk, Read - 942.2kBps Write - 2034.56KBps,

Before running disk1, Read - 1017.3kBps Write - 2021.82kBps,
After running disk1, Read - 1015.24kBps Write - 2015kBps.

We see a sharp increase in read speed after running disk.c due to reopening of random files persistently. However, the same observation is not noticed in disk1.c, because it opens the same file repeatedly which is stored in disk buffer cache after the first run and hence not needing to be read again from the disk.