Lab 1 - CS 333

-- 140050002, 140050007

Exercise 1

Machine has 4 cpu cores. Which can be seen in /proc/cpuinfo.

Machine's memory information can be seen in /proc/meminfo
MemTotal: 8146892 kB
MemFree: 5039740 kB
61% of total memory is free.

System has performed 6801130 context switches since bootup which was seen from /proc/stat

7469 processes are being forked since bootup which can be seen in /proc/stat

Exercise 2

cpu.c
Bottleneck resource : CPU
Reasoning : %CPU utilized is 100% . (can be seen while running "top" command)
Justification : cpu.c file only does multiplication in while true loop. Which uses ALU of CPU.

cpu-print.c
Bottleneck resource : Display Rate (Monitor)
Reasoning : Writing(Displaying) to the terminal continuously.
Justification :
Neither CPU nor I/O nor memory reaches significant high value. But from the code we can observe that its purpose is to print the time every time we are in the while true loop. This result will change depending on the speed at which the display can print to the terminal.

disk.c
Bottleneck resource : I/O device - disk
Reasoning : Reading rate from disk reaches around 300 MB/s which is sufficiently high.
(can be seen under rkB/s while running "iostat -x 1").
Justification :
disk.c randomly reads files in while true loop. Each file is 2 Mb large and there are 10000 files. Since cache size is very less than 2*10000 = 20Gb, and file reading is random, so next file can not be predicted. Thus most of the times cache miss occurs. Which results in bottleneck.

disk1.c
There are no bottleneck resources.

Reasoning : Neither CPU nor I/O nor memory reaches significant high value.
(use “top” see %CPU, %MEM and use “iostat - x 1” see rkB/s,wkB/s)
Justification :
disk1.c reads only one 2 Mb file which can be stored in cache, RAM. So there is no need to access disk or any other I/O device. There is no computation which can lead to exhaustive use of all resources of CPU.

Exercise 3

cpu.c spends more time in user mode than in kernel mode because program does multiplication which is executed in user space. (does not need kernel privileges)

cpu-print.c spends more time in kernel mode than in user mode because program asks for current time of system for which process must go in kernel mode.

The amount of time each process spends in user mode and kernel mode can be seen in /proc/pid/stat file where pid is id of process.
14th and 15th space separated value in file corresponds to time process spends in user mode and kernel mode respectively in number of clock ticks.

pid can be obtained by “top” command on terminal. Or by “ps ax | grep ./nameOfExecutableFile” (without quotes) eg. “ps ax | grep ./cpu”

Exercise 4

disk.c has mostly voluntary context switches and cpu.c has mostly involuntary context switches.

Number of context switches can be seen in /proc/pid/status file.
Where pid is process id which can be obtained by “ps ax | grep ./nameOfExecutableFile” (without quotes) eg. “ps ax | grep ./cpu”

	Voluntary context switches	Involuntary context switches
disk.c	254301	4961
cpu.c	1	11016

cpu.c runs infinite loop for multiplication which does not have any system call. Thus there will not be any

Exercise 5

pid of bash shell is 4937. Which is obtained by running “ps ax | grep terminal”

Process tree is obtained by “pstree --highlight-pid 4937“

init ----> lightdm ----> lightdm ----> init ----> gnome-terminal

Exercise 6

ls and ps are executed by bash shell. Their executables can be found in /bin folder.
cd and history are shell builtin which are implemented by bash code itself.

This can be seen using “type -a commandName” eg. “type -a cd”

Exercise 7

Pid of new process is 9036.

I/O file descriptor 0 and 2 points to /dev/pts/0

It has following type : link to character device (inode/chardevice)

I/O file descriptor 1 points to /tmp/tmp.txt

Directory /proc/pid/fd contains one entry for each file which the process has open, named by its file descriptor, and which is a symbolic link to the actual file. Following are file descriptor and links to which they point.

0 is standard input,

1 standard output,

2 standard error

Thus in our case,
standard output is redirected to /tmp/tmp.txt thus I/O file descriptor 1 points to /tmp/tmp.txt

A non existent file-system structure ends up being a ‘chardevice’ type structure(Usually assigned to external character devices connected to the system). Since neither input nor any error exists, file descriptor 0 and 2 link to a chardevice.

Exercise 8

Pid of new process for executable is 12043 and grep is 12044

For 12043
I/O file descriptor 0 and 2 points to /dev/pts/0
        It has following type : link to character device (inode/chardevice)
I/O file descriptor 1 points to pipe:[398895]
        And type : link to pipe (inode/fifo)

For 12044
I/O file descriptor 1 and 2 points to /dev/pts/0
        It has following type : link to character device (inode/chardevice)
I/O file descriptor 0 points to pipe:[398895]
        And type : link to pipe (inode/fifo)

Output of executable process is pointing to the same pipe as the input of grep process is pointing to. Which means all output of executable is redirected as input of grep process using pipe. This is how pipe works in bash.