2017-EE-12

2017-EE-13

2017-EE-24

Muhammad Ahmad Sultan

Muhammad Mohsin

Rao Muhammad Harris Rafique

|  |  |  |
| --- | --- | --- |
|  | Operating Systems Lab - 1 |  |

**Q1) Use of proc:**

**a) Processor and cores in /proc/cpuinfo:**

**Processor:** A processor is a chip which is capable of executing instructions. There may be multiple cores on a single processor chip.

**Core:** Core is unit inside the processor which contains the ALU. Each core can run 1 process at a time.

**b) Cores**:

Each processor in 2017EE13 PC has 2 cores so /proc/cpuinfo file shows 2 here

**c) Processor**

The processor shows id number of processors in the chip. Since for the VM, 1 processor (out of a total of two) was allocated, it is showing only one processor with ID 0.

**d) CPU MHz:**

Shows the frequency of the CPU in MHz which is 2904 MHz or 2.70GHz as shown in model name.

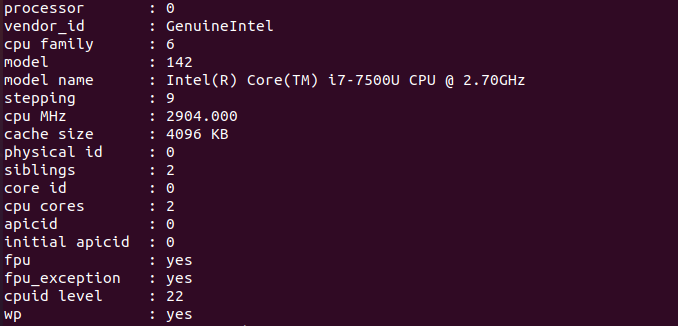


Fig: Output of /proc/cpuinfo

1. **Physical Memory:**

Using the cat proc/meminfo command the process can be given information about memory can be retrieved.

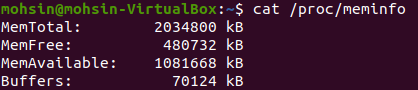


Fig:Output of /proc/meminfo

This shows that total physical memory (RAM) of the system. Its equal to 2034800 kB or 1.94 GB. For the VM, 2 GB RAM was allocated so it is near to that value. The difference is owing to the fact that the MemTotal gives the total RAM minus a few reserved bits and kernel binary code.

1. MemFree is the LowFree (accessible directly by kernel) + HighFree memory( the memory region after 860 MB of the physical memory) . This number is usually small as memory that is free is not used by anything. MemAvailable gives the RAM which is available for allocation of the new processes.
2. **Number of forks since boot:**

The /proc/stat file has processes info which gives the number of forks since startup of the system. The figure below gives the number of forks as 4363.

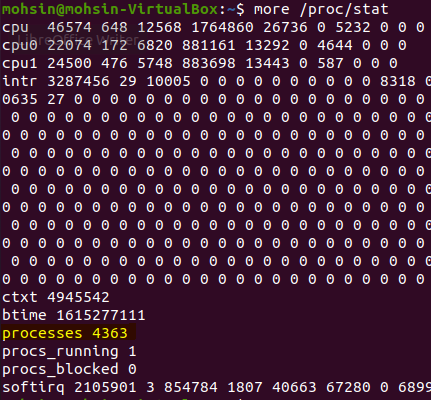


Fig: Output of /proc/stat

1. **Context Switches:**

**CTXT** field in /proc/stat gives the number of context switches and it is equal to 4945542 according to above figure. This is the number of times the process is changed on the cpu.

**Q2) Use of top command.**

After running the cpu.c program and then using the top command, the following output is received:

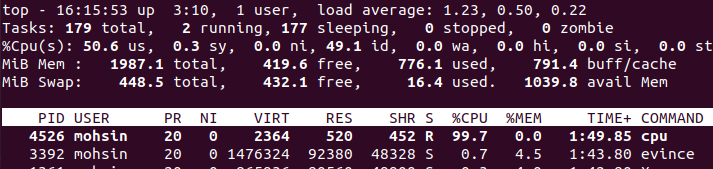


Fig: Output of top command

1. The cpu command has the pid of 4526
2. CPU utilization is 99.7 % and memory utilization is 0%.
3. State for the program is shown by **S** before the %CPU column and is R which means it is in running state.

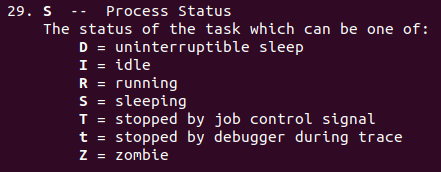


Fig: Information about S column in top command

**Q3)** **Bash Command Execution**

The program cpu-print.c is compiled and executed in one shell and in the other shell the ps command is executed with the pid of the cpu-print process.

1. Using ps -e command we find the PID of the process cpu-print.

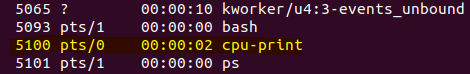


Fig: ps output using pid of cpu-print

1. The process tree of the process cpu-print is given by the command pstree -s -p [pid]. Where [pid] is the process id of cpu-print.



Fig: parents tree of cpu-print

1. I/O redirection: File descriptors and fork interact to make I/O redirection easy to implement. Fork copies the parent’s file descriptor table along with its memory, so that the child starts with exactly the same open files as the parent. The system call exec replaces the calling process’s memory but preserves its file table. This behavior allows the shell to implement I/O redirection by forking, reopening chosen file descriptors, and then exec-ing the new program. The new program can then change the file descriptors of the input, output or error as needed. Here the output file descriptor is changed from shell to /tmp/tmp.txt.

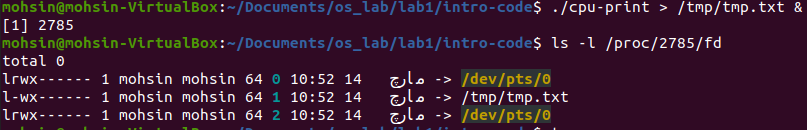


Fig: I/O redirection with output to file

1. A pipe is a small kernel buffer exposed to processes as a pair of file descriptors,one for reading and one for writing. Writing data to one end of the pipe makes thatdata available for reading from the other end of the pipe. Pipes provide a way for processes to communicate. The parent after forking creates the child. The output of the parent and input of child is the same pipe. So communication can take place between the process.

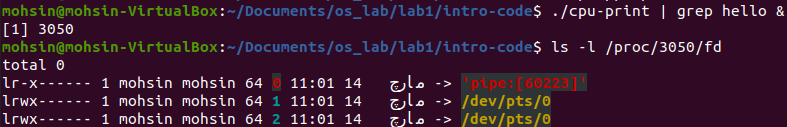


Fig: I/O redirection using pipes

e) The type command allows us to check whether a command is built-in or not.

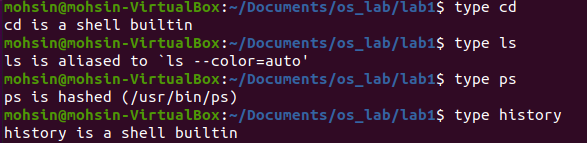


Fig: Type command output for various commands

**Q4) Virtual and Physical memory usage**

VSZ is the Virtual Memory Size. It includes all memory that the process can access, including memory that is swapped out, memory that is allocated, but not used, and memory that is from shared libraries, heaps and stacks.

RSS is the actual memory in the RAM (un swapped memory). Both vsz and rss give memory in Kilo Bytes.

As the code in both memory1.c and memory2.c initializes the integer array of 1,000,000 elements. So it occupies a space of 4 MB (size of (int) = 4 bytes). The remaining Vss memory also contains this memory plus any memory swapped out.

Fig: Ps memory output for memory1.c



Fig: Ps memory output for memory2.c

Memory2.c has more memory allocation as it has more instructions then memory1.c.

**Q5) Disk Utilization:**

As in Disk1.c same file is being read so it is stored in the cache and the I/O read become zero very quickly.

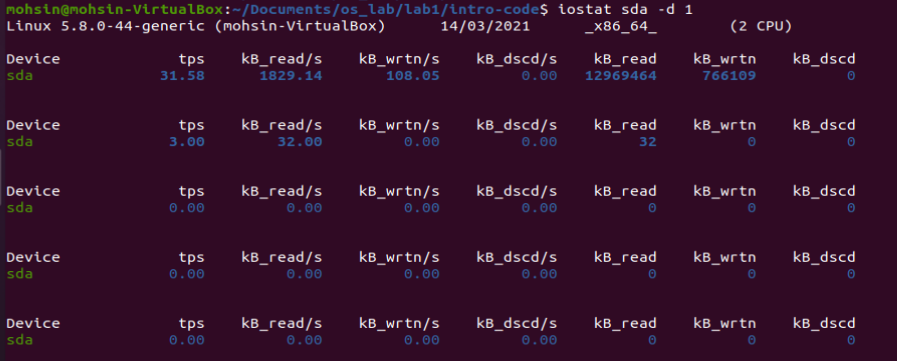
­

Fig: Output of iostat during disk1.c execution

In Disk.c, the file is picked randomly so it has more reads from the secondary storage but after a while, because after many thousands of files are read, its I/O also goes to zero as the files are cached.

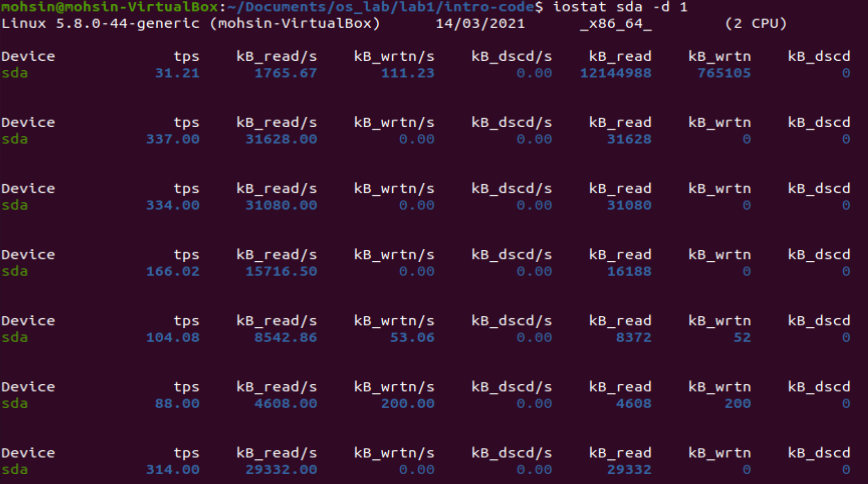


Fig: Output of iostat during disk.c execution

The following command was used to clear the cache

sync; echo 3 > /proc/sys/vm/drop\_caches;