Q1>Collect the following basic information about your machine using the /proc file system and answer the following questions:

a) How many CPU sockets, cores, and CPUs does the machine have?

Soln: We can check the number of CPU sockets by counting the unique physical ids listed in the ‘proc/cpuinfo’ file.



We can count the number of CPU cores by counting the unique core Ids listed in the ‘/proc/cpuinfo’ file.



To get the total number of CPUs we can count the lines in the ‘proc/cpuinfo’ file.



(b) What is the frequency of each CPU?

Soln:



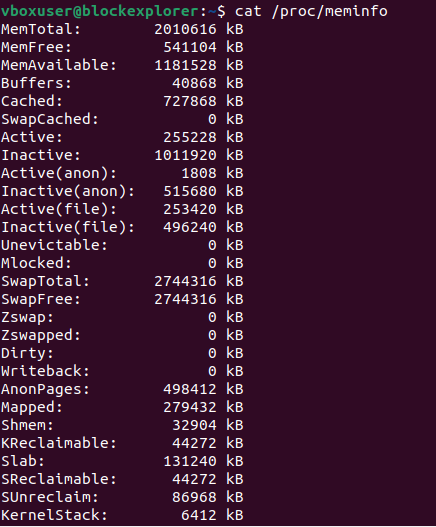
(c) How much memory does your machine have?

Soln:



(d) How much of it is free and available? What is the difference between them?

Soln:



(e) What is total number of user-level processes in the system?

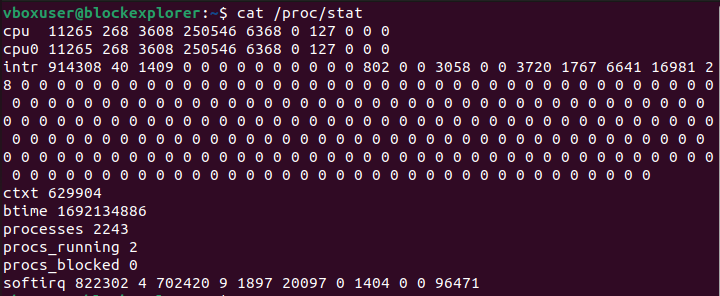
Soln:



This command lists all the processes (‘-e’) and extracts the usernames (‘-o users=’). It then sorts and remove the duplicates using ‘sort’ and ‘unique’ respectively, and finally counts the unique user account using ‘wc –l’.

(f) How many context switches has the system performed since bootup?

Soln: To find out the number of context switches that the system has performed since bootup we can examine the ‘/proc/stat’ file.



ctxt field denotes the number of context switches.

(g) What is the size of files in the /proc directory? Frame a question of investigation based on the file size observation.

Soln:

To find the size of entries in the ‘/proc’ directory as displayed by the typical listing command, one can use the ‘du’ (disk usage) command.

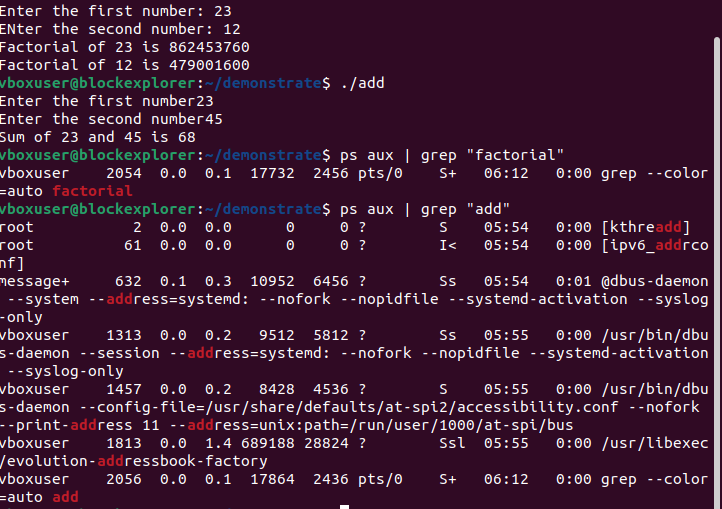
du –sh/proc

Q2> Create a directory and programs inside it. Run all the programs and compare the memory usage of these programs in terms of VMSize and VMRSS.

Soln: In order to solve this problem, the following steps must be followed:

Step 1: Navigate to the directory: Open a terminal and navigate to the directory containing the programs you want to run.

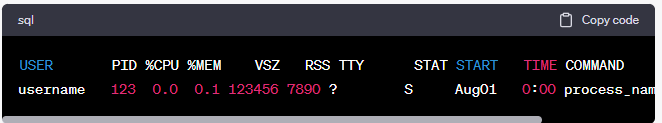
Step 2: Compile and run programs.



Step 3: Get the VSSize and VmRSS using the command:

ps aux | grep “program\_name”

The output of the above command is in the format:



Q3> Obtain the process id of the processes running in the system and display the count of child processes.

Soln:



Q4>Run strace along with the binary program of empty.c given in subdirectory strace.

What do you think the output of strace indicates in this case? How many different (system call)

functions do you see?

Next, use strace along with another binary program of hello.c (which is in the same directory).

Compare the two strace outputs,

(a) Which part of the strace output is common, and which part has to do with the specific

program?

(b) List all unique system call functions for each program and look up functionality of each.

Soln:

**strace** is a tool that allows you to trace system calls and signals made by a process. By using it with different programs, you can see how the programs interact with the operating system. Here's how to use **strace** with the **empty.c** and **hello.c** programs and analyze their outputs:

1. **Using strace with empty.c:**

Assuming you have compiled **empty.c** into an executable named **empty**, you can run **strace** as follows:

strace ./empty

The **empty** program does not have any specific functionality; it's likely just an empty program. When you run **strace** on it, you will see a series of system calls that the program makes. Since the program doesn't do anything, the system calls you see will be related to its startup and termination.

To count the number of different system call functions, you can run the following command:

strace ./empty 2>&1 | awk '{print $2}' | sort -u

This command will list all unique system call functions made by the program.

1. **Using strace with hello.c:**

Similar to the previous step, if you have compiled **hello.c** into an executable named **hello**, you can run **strace** on it:

strace ./hello

The **hello** program is likely a simple program that prints "Hello, World!" to the terminal. When you run **strace** on it, you will see additional system calls related to file I/O and standard output.

To count the number of different system call functions for this program:

strace ./hello 2>&1 | awk '{print $2}' | sort -u

This command will list all unique system call functions made by the program.

**Comparison of Outputs:**

(a) Common Part: The common part of the **strace** output between the two programs will likely include system calls related to program startup and termination, memory allocation, and dynamic linking. These parts are common to most programs regardless of their specific functionality.

Specific Part: The specific part of the **strace** output will show system calls that are unique to each program's functionality. For example, **hello.c** might have additional system calls related to standard output and file I/O.

(b) Unique System Call Functions:

* For **empty.c**, you might see basic system calls like **execve**, **arch\_prctl**, **brk**, **mmap**, **read**, **write**, **exit\_group**, etc.
* For **hello.c**, in addition to the basic system calls, you might see system calls like **open**, **close**, **fstat**, **writev**, and others related to file I/O and printing to the terminal.

You can look up the functionality of each system call using the **man** command in the terminal. For example:

man 2 execve

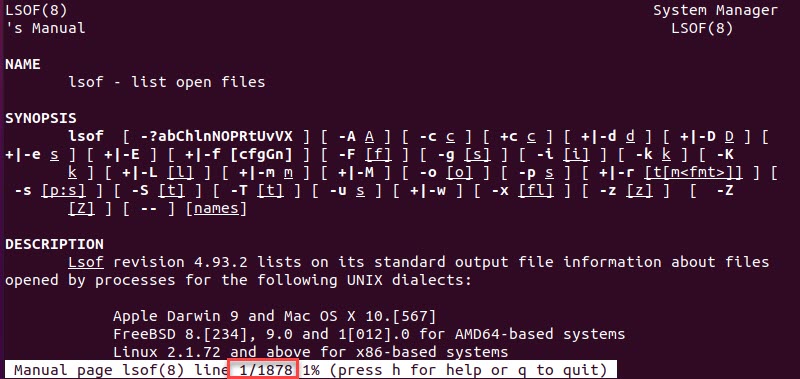
man 2 open

Replace the system call name with the one you want to learn more about. This will provide detailed information about the specific system call's purpose, parameters, and usage.

Q5> Run the executable openfiles in subdirectory files.

List the files which are opened by this program, and describe how you obtained the answer.

Soln:



To list the files that are opened by an executable named **openfiles** in the subdirectory **files**, you can use the **lsof** (List Open Files) command. This command allows you to see which files are currently opened by running processes. Here's how you can use it:

1. Open a terminal on your Linux system.
2. Navigate to the directory containing the **openfiles** executable in the **files** subdirectory. You can use the **cd** command to change the directory:

cd /path/to/files

1. Run the **openfiles** executable. Assuming it's a command-line program, you can simply type its name and press Enter:

./openfiles

1. While the **openfiles** program is running, open another terminal window (you can use a split terminal or a new tab, depending on your terminal emulator).
2. In the new terminal window, run the following **lsof** command to list the files opened by the **openfiles** program:

lsof -c openfiles

The **-c** flag is used to specify the process name (**openfiles** in this case) for which you want to list the open files.

1. The output of the **lsof** command will show you the list of files opened by the **openfiles** program. Each line will provide details about the file, such as the file descriptor, type, device, size, and file name.

Here's how you obtained the answer:

1. You navigated to the directory containing the **openfiles** executable using the **cd** command.
2. You ran the **openfiles** program.
3. You used the **lsof** command with the **-c** flag to specify the process name (**openfiles**) and list the open files for that process.

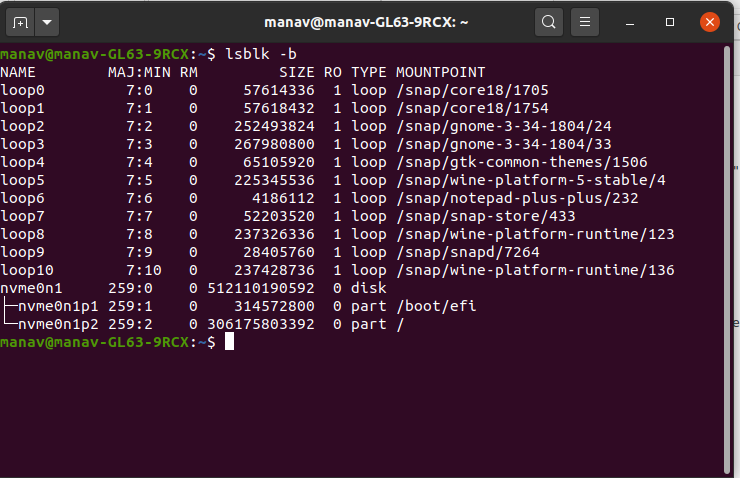
The **lsof** command provides a way to inspect which files a particular process has opened, which can be useful for troubleshooting, monitoring, and understanding the behavior of running programs.

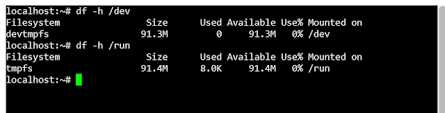
Q6> Find all the block devices on your system, their mount points and file systems present on them.

A mount point is a file system directory entry from where a disk can be accessed. A file system

described how data is organized on a disk. Describe how you obtained the answer.

Soln:







To find all the block devices on your system, their mount points, and the file systems present on them, you can use various commands such as **lsblk**, **df**, and **blkid**. Here's how you can obtain the information you're looking for:

1. **Using lsblk Command:** The **lsblk** command lists information about all available block devices, their mount points, and file systems.

lsblk

This command will display a tree-like structure of block devices along with their sizes, mount points, and file systems.

1. **Using df Command:** The **df** command displays information about file system disk space usage, including the device, mount point, and file system type.

df -h

The **-h** flag is used to display sizes in a human-readable format.

1. **Using blkid Command:** The **blkid** command displays attributes of block devices, including their file system type and UUID.

blkid

This command provides detailed information about each block device and its associated file system.

To obtain the answer, follow these steps:

1. Open a terminal on your Linux system.
2. Run the **lsblk** command to list block devices, mount points, and file systems.
3. Run the **df -h** command to check disk space usage and file systems.
4. Run the **blkid** command to get attributes of block devices and file systems.

By using these commands, you'll be able to gather information about the block devices on your system, their mount points, and the file systems present on them. The combination of these commands provides a comprehensive view of your system's storage setup.