1. The command netstat -nat lists all current TCP connections. How can I count the number of connections in each state?

```
Desktop $ netstat -nat | awk '/^tcp/ {print $6}' | sort | uniq -c

  17 ESTABLISHED

   8 LISTEN
```

2. How can we read the stdin, stdout, and stderr streams of a running process using the cat command?

short answer:

```
cat /proc/<pid>/fd/<0|1|2>
```

`0` = stdin `1` = stdout, `2` = stderr

tl;dr; As of 2020, you cannot do that (or anything similar) if `/proc/<pid>/fd/<fd>` is a socket.

The stdin, stdout, stderr of a process may be any kind of file, not necessarily pipes, regular files, etc. They can also be sockets.

On Linux, the `/proc/<pid>/fd/<fd>` are a special kind of symbolic links which allow you to open **from the scratch** the actual file a file descriptor refers to, and do it even if the file has been removed, or it didn't ever have any presence in any file system at all (e.g. a file created with `memfd_create(2)`).

But sockets are a notable exception, and they cannot be opened that way (nor is it obvious at all how that could be implemented: would an `open()` on `/proc/<pid>/fd/<fd>` create another connection to the server if that fd is a connected socket? what if the socket is explicitly bound to a local port?).

Recent versions of Linux kernels have introduced a new system call, `pidfd_getfd(2)`, which allows you to "steal" a file descriptor from another process, in the same way you were able to pass it via Unix sockets, but without the collaboration of the victim process. But that hasn't yet made its way in most Linux distros.

https://unix.stackexchange.com/a/613450

3. What is an inode and what is its purpose?

An inode is a filesystem structure storing file metadata (e.g., size, permissions) and data block pointers, used to manage files without storing names.

4. When a new file is created using the touch command, what happens step-by-step in the filesystem, especially in terms of the inode?

The filesystem allocates an inode, sets metadata (e.g., creation time), links it to a directory entry with the filename, and updates the filesystem's inode table.

5. What does the stat command show about a file's inode, and how should we interpret that information?

```
arash@ubuntu24:~/Desktop$ stat text.txt
  File: text.txt
  Size: 492         Blocks: 8          IO Block: 4096   regular file
Device: 8,2 Inode: 656712      Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/   arash)   Gid: ( 1000/   arash)
Access: 2025-05-16 18:38:00.673900673 +0000
Modify: 2025-05-16 18:38:08.265579770 +0000
Change: 2025-05-16 18:38:08.265579770 +0000
 Birth: 2025-05-16 18:30:59.004625775 +0000
```

```
File: text.txt
Size: 492 Blocks: 8 IO Block: 4096   regular file
```

- **File:** Name of the file.
- **Size:** File size in bytes (here: `492` bytes).
- **Blocks:** Number of 512-byte blocks allocated on disk. `8` blocks = 4096 bytes (which matches the block size).
- **IO Block:** Filesystem block size used for I/O. Usually 4096 bytes on modern systems.
- **regular file:** Type of file (could also be directory, symlink, etc.).

Storage and Ownership

`Device: 8,2 Inode: 656712 Links: 1`

- **Device:** Identifier of the device storing the file.
- **Inode:** Unique number identifying the file's metadata.
- **Links:** Number of hard links pointing to this inode (here: only one).

Permissions and Ownership

`Access: (0664/-rw-rw-r--) Uid: ( 1000/ arash) Gid: ( 1000/ arash)`

- **Access (0664):** File permissions in octal and symbolic format.
- `rw-rw-r--` means:
  - Owner (`arash`): read/write
  - Group (`arash`): read/write
  - Others: read
- **UID/GID:** File owner and group.

Timestamps

```
Access: 2025-05-16 18:38:00.673900673 +0000
Modify: 2025-05-16 18:38:08.265579770 +0000
Change: 2025-05-16 18:38:08.265579770 +0000
Birth: 2025-05-16 18:30:59.004625775 +0000
```

- **Access:** Last time the file was *read*.
- **Modify:** Last time *file content* was changed.
- **Change:** Last time *metadata* (permissions, ownership, etc.) was changed.
- **Birth:** File creation time (only shown on some filesystems like ext4 with crtime support, btrfs, etc.).

6. When using the mv command, what changes occur at the `inode` level? Under what conditions does the `inode` stay the same or change?

- `mv` at the same directory:
  - Nothing changed

```
arash@ubuntu24:~/Desktop$ mv text.txt text1.txt
arash@ubuntu24:~/Desktop$ stat text1.txt
  File: text1.txt
  Size: 492         Blocks: 8          IO Block: 4096   regular file
Device: 8,2 Inode: 656712      Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/   arash)   Gid: ( 1000/   arash)
Access: 2025-05-16 18:38:00.673900673 +0000
Modify: 2025-05-16 18:38:08.265579770 +0000
```

```
Change: 2025-05-16 18:40:50.592352092 +0000
 Birth: 2025-05-16 18:30:59.004625775 +0000
```

- `mv` to another directory:
  - updated the path entry to `changed-path/text1.txt`

```
arash@ubuntu24:~/Desktop$ mv text1.txt changed-path/text1.txt
arash@ubuntu24:~/Desktop$ stat changed-path/text1.txt
  File: changed-path/text1.txt
  Size: 492         Blocks: 8          IO Block: 4096   regular file
Device: 8,2 Inode: 656712      Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/   arash)   Gid: ( 1000/   arash)
Access: 2025-05-16 18:38:00.673900673 +0000
Modify: 2025-05-16 18:38:08.265579770 +0000
Change: 2025-05-16 18:42:27.712056481 +0000
 Birth: 2025-05-16 18:30:59.004625775 +0000
```

7. When you run ls, how does it retrieve the names of all files and directories in the current directory?

`ls` uses `opendir()` and `readdir()` to step through all the files in the directory. If it needs more information about one of them it calls `stat()`.

https://stackoverflow.com/a/204211
https://medium.com/better-programming/how-does-ls-work-14fdc2b85308

8. When you run the top command, it shows process states such as running, sleeping, zombie, and stopped. Can you explain what each state means and when it occurs?

- `running` shows how many processes are handling requests, executing normally, and have CPU access.
- `sleeping` indicates processes awaiting resources, which is a normal state.
- `stopped` reports processes exiting and releasing resources; these send a termination message to the parent process.
- `zombie` refers to a process waiting for its parent process to release it; it may become orphaned if the parent exits first.
  https://www.redhat.com/en/blog/interpret-top-output

9. If your server shows a high load average and a high number of context switches, what could this indicate?

Context switching is the process of switching the CPU from one process, task or thread to another. In a multitasking operating system, such as Linux, the CPU has to switch between

multiple processes or threads in order to keep the system running smoothly. This is necessary because each CPU core without hyperthreading can only execute one process or thread at a time. If there are many processes or threads running simultaneously, and very few CPU cores available to handle them, the system is forced to make more context switches to balance the CPU resources among them.

Context switching is an essential function of any multitasking operating system, but it also comes at a cost. The whole process is computationally intensive, and the more context switches that occur, the slower the system becomes. This is because each context switch involves saving the current state of the CPU, loading the state of the new process or thread, and then resuming execution of the new process or thread. This takes time and consumes CPU resources, which can slow down the system.

The impact of context switching on system performance can be significant, especially in systems with many processes or threads running simultaneously.

https://www.netdata.cloud/blog/understanding-context-switching-and-its-impact-on-system-performance/

10. Try running `atop`, `htop`, and `mon`. Where do these tools retrieve their data from?

- **/proc filesystem** (e.g., `/proc/stat`, `/proc/meminfo`, `/proc/[pid]`)
- **/sys filesystem** (for hardware stats)
- Kernel interfaces for real-time data (e.g., CPU, memory, disk I/O)