OS Assignment 02 Solutions

1. 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 are in your machine?

```
# Number of CPU sockets

grep -m 1 'physical id' /proc/cpuinfo | wc -l

# Number of cores per socket

grep 'cpu cores' /proc/cpuinfo | uniq

# Total number of logical CPUs

grep -c '^processor' /proc/cpuinfo
```

(b) What is the CPU model name and stepping of each processor in the system?

```
# CPU model name

grep 'model name' /proc/cpuinfo | uniq

# CPU stepping

grep 'stepping' /proc/cpuinfo | uniq
```

(c) What is the frequency of each CPU?

```
grep 'cpu MHz' /proc/cpuinfo
```

(d) What is the total amount of memory in your machine?

```
grep 'MemTotal' /proc/meminfo
```

(e) How much memory is currently free, and how much is available? What distinguishes free memory from available memory?

```
grep -E 'MemFree|MemAvailable' /proc/meminfo

# Explanation:

# Free memory: This is the amount of memory that is not being used at all (it is completely unallocated).
```

```
# Available memory: This is the amount of memory that is available for allocation to processes. It includes free memory and reclaimable memory (memory that can be reclaimed from caches).
```

(f) How much swap memory is configured on your machine? How much of it is currently used?

```
grep -E 'SwapTotal|SwapFree' /proc/meminfo
```

(g) What version of the kernel is currently running on your machine?

```
cat /proc/version
```

(h) How many user-level processes are currently running in the system?

```
ps -e --no-headers | wc -l
```

(i) What is the total number of context switches that have occurred since the system was booted?

```
grep 'ctxt' /proc/stat
```

(j) How long has your system been running since the last boot? What is the average load on the system?

```
# Uptime and load averages

uptime

# Alternatively for just uptime

cat /proc/uptime
```

(k) What is the size of the files located in the /proc directory?

```
ls -l /proc | awk '{print $5, $9}'

# or

du -h --max-depth=1 /proc
```

(l) What are the top 5 processes consuming the most memory on your system? Provide their PID, user, and memory usage.

```
ps aux --sort=-%mem | head -n 6
```

2. Collect the read-and-write statistics for disks and disk usage for mounted file systems:

(a) What are the read-and-write statistics for the disks on your machine?

Here is the command to display this information:

```
cat /proc /diskstats
```

This file provides detailed information about I/O operations for each disk device. Each line in the output corresponds to a single device and contains the following fields:

1. Major number
2. Minor number
3. Device name
4. Reads completed successfully
5. Reads merged
6. Sectors read
7. Time spent reading (ms)
8. Writes completed
9. Writes merged
10. Sectors written
11. Time spent writing (ms)
12. I/Os currently in progress
13. Time spent doing I/Os (ms)
14. Weighted time spent doing I/Os (ms)

You can also format the output for better readability. Here's a sample script:

```
awk '{print "Device: " $3 "\nReads Completed: " $4 "\nSectors Read: " $6 "\nTime Spent Reading (ms): " $7 "\nWrites Completed: " $8 "\nSectors Written: " $10 "\nTime Spent Writing (ms): " $11 "\n"}' /proc/diskstats
```

(b) Display the disk usage for each mounted filesystem and explain the output.

You can use the `df` command to display the disk usage for each mounted filesystem:

```
df -h
```

The `df` command shows the amount of disk space used and available on Linux file systems. The `-h` option makes the output human-readable. Here's an example of what the output might look like:

```
Filesystem      Size  Used Avail Use% Mounted on
udev            7.8G     0  7.8G   0% /dev
tmpfs           1.6G  2.0M  1.6G   1% /run
/dev/sda1       118G   25G   88G  22% /
tmpfs           7.8G   47M  7.8G   1% /dev/shm
tmpfs           5.0M  4.0K  5.0M   1% /run/lock
tmpfs           7.8G     0  7.8G   0% /sys/fs/cgroup
/dev/sda2       200G  150G   50G  75% /home
```

Explanation of the Output:

- **Filesystem**: The name of each filesystem.
- **Size**: Total size of the filesystem.
- **Used**: Amount of space used on the filesystem.
- **Avail**: Amount of space available on the filesystem.
- **Use%**: Percentage of the filesystem's total space that is used.
- **Mounted on**: The directory where the filesystem is mounted.

Here is an example of how you might document your findings:

(a) Read-and-Write Statistics for Disks

- **Output**

```
Device: sda
Reads Completed: 1203456
Sectors Read: 23456789
Time Spent Reading (ms): 456789
Writes Completed: 1234567
```

```
Sectors Written: 34567890

Time Spent Writing (ms): 567890
```

- **Explanation**
  - **Device**: The device name, e.g., sda.
  - **Reads Completed**: The total number of read operations completed.
  - **Sectors Read**: The total number of sectors read.
  - **Time Spent Reading**: The total time spent on reading operations in milliseconds.
  - **Writes Completed**: The total number of write operations completed.
  - **Sectors Written**: The total number of sectors written.
  - **Time Spent Writing**: The total time spent on writing operations in milliseconds.

(b) Disk Usage for Mounted Filesystems

- **Output**

```
Filesystem      Size  Used Avail Use% Mounted on

udev            7.8G     0  7.8G   0% /dev

tmpfs           1.6G  2.0M  1.6G   1% /run

/dev/sda1       118G   25G   88G  22% /

tmpfs           7.8G   47M  7.8G   1% /dev/shm

tmpfs           5.0M  4.0K  5.0M   1% /run/lock

tmpfs           7.8G     0  7.8G   0% /sys/fs/cgroup

/dev/sda2       200G  150G   50G  75% /home
```

- **Explanation**
  - **/dev/sda1**:
    - **Size**: 118G
    - **Used**: 25G
    - **Available**: 88G
    - **Usage**: 22%

        - **Mounted on**: Root directory (/)
    - **/dev/sda2**:
        - **Size**: 200G
        - **Used**: 150G
        - **Available**: 50G
        - **Usage**: 75%
        - **Mounted on**: Home directory (/home)

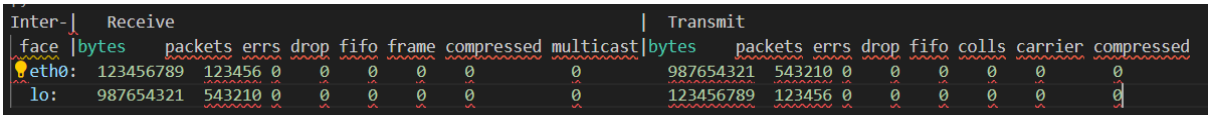
3. Collect the packet statistics for network interface and current network:

(a) What are your machine's RX and TX packet statistics for each network interface?

You can use the /proc/net/dev file to get the RX (receive) and TX (transmit) packet statistics for each network interface. Here's the command:

```
cat / proc / net / dev
```

This file provides detailed statistics for each network interface. The output typically looks like this:



To format this for easier reading:

```
awk 'NR>2 {print "Interface: " $1 "\nRX Bytes: " $2 "\nRX Packets: " $3
"\nTX Bytes: " $10 "\nTX Packets: " $11 "\n"}' /proc/net/dev
```

(b) What is the current network configuration of your machine, including IP addresses and interface statuses?

You can use the ip command to display the current network configuration:

```
ip addr show
```

This command provides detailed information about all network interfaces, including their IP addresses, status (UP or DOWN), and other relevant information.

Example of Collecting and Explaining Network Statistics and Configuration

Here is an example of how you might document your findings:

(a) RX and TX Packet Statistics for Network Interfaces

- **Output**:

```
Interface: eth0

RX Bytes: 123456789

RX Packets: 123456

TX Bytes: 987654321

TX Packets: 543210



Interface: lo

RX Bytes: 987654321

RX Packets: 543210

TX Bytes: 123456789

TX Packets: 123456
```

Explanation:

- **eth0**:
    - **RX Bytes**: Total number of bytes received.
    - **RX Packets**: Total number of packets received.
    - **TX Bytes**: Total number of bytes transmitted.
    - **TX Packets**: Total number of packets transmitted.
- **lo (Loopback Interface)**:
    - Similar statistics for the loopback interface.

(b) Current Network Configuration

- **Output:**

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
group default qlen 1000
```

```
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

    inet 127.0.0.1/8 scope host lo

       valid_lft forever preferred_lft forever

    inet6 ::1/128 scope host

       valid_lft forever preferred_lft forever

2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
state UP group default qlen 1000

    link/ether 00:11:22:33:44:55 brd ff:ff:ff:ff:ff:ff

    inet 192.168.1.100/24 brd 192.168.1.255 scope global dynamic eth0

       valid_lft 123456sec preferred_lft 123456sec

    inet6 fe80::211:22ff:fe33:4455/64 scope link

       valid_lft forever preferred_lft forever
```

Explanation:

- **lo (Loopback Interface)**:
  - **Status**: UP and RUNNING.
  - **IP Address**: 127.0.0.1 (IPv4), ::1 (IPv6).
- **eth0**:
  - **Status**: UP and RUNNING.
  - **MAC Address**: 00:11:22:33:44:55.
  - **IPv4 Address**: 192.168.1.100/24.
  - **IPv6 Address**: fe80::211:22ff:fe33:4455/64.

4. Collect the information regarding file descriptors as mentioned below:

(a) Identify the number of open file descriptors for a specific process

You need the process ID (PID) of the process you are interested in. Once you have the PID, you can use the following command to count the number of open file descriptors:

```
# Replace <PID> with the actual process ID
```

```
ls /proc/<PID>/fd | wc -l
```

This command lists all the file descriptors for the specified process and counts them using wc -l.

(b) List all the open file descriptors and their corresponding files for a specific process

Again, using the PID of the process, you can list all the open file descriptors and their corresponding files with the following command:

```
# Replace <PID> with the actual process ID

ls -l /proc/<PID>/fd
```

This command provides a detailed listing of the open file descriptors, showing symbolic links to the actual files they correspond to.

Example of Collecting and Explaining File Descriptor Information

Here is an example of how you might document your findings:

(a) Number of Open File Descriptors for a Specific Process

- **Command:**

```
ls / proc / 1234 / fd | wc - l
```

- **Output**:

```
28
```

- **Explanation**:
    - The process with PID 1234 has 28 open file descriptors.

(b) List of All Open File Descriptors and Their Corresponding Files for a Specific Process

- **Command**:

```
ls - l / proc / 1234 / fd
```

- **Output**:

```
total 0
lrwx------ 1 user user 64 Jul 30 10:00 0 -> /dev/pts/0
lrwx------ 1 user user 64 Jul 30 10:00 1 -> /dev/pts/0
lrwx------ 1 user user 64 Jul 30 10:00 2 -> /dev/pts/0
lr-x------ 1 user user 64 Jul 30 10:00 3 -> /proc/1234/fd
lrwx------ 1 user user 64 Jul 30 10:00 4 -> socket:[12345]
lrwx------ 1 user user 64 Jul 30 10:00 5 -> /home/user/somefile.txt
...
```

- **Explanation**:
  - **0, 1, 2**: Standard input, output, and error are connected to /dev/pts/0.
  - **3**: A file descriptor for the /proc/1234/fd directory itself.
  - **4**: A socket connection.
  - **5**: An open file descriptor for /home/user/somefile.txt.
  - The list continues for all open file descriptors of the process.

5. Collect the information for the following:

(a) List all the currently mounted filesystems, mount points, and filesystem types.

You can use the mount command or view the contents of the `/proc/mounts` file to get this information.

```
mount
```

or,

```
cat / proc / mounts
```

These commands display information about all currently mounted filesystems, including their mount points and filesystem types.

(b) Find the block devices on your system and their sizes?

You can use the `lsblk` command to list all block devices along with their sizes.

Alternatively, you can use the `fdisk` command with the `-l` option to get a detailed layout of your disks.

```
sudo fdisk -l
```

Example of Collecting and Explaining Filesystem and Block Device Information

Here is an example of how you might document your findings:

(a) List of Currently Mounted Filesystems, Mount Points, and Filesystem Types

- **Command**:

```
mount
```

- **Output**:

```
/dev/sda1 on / type ext4 (rw,relatime,data=ordered)

tmpfs on /run type tmpfs (rw,nosuid,nodev,mode=755)

tmpfs on /dev/shm type tmpfs (rw,nosuid,nodev)

/dev/sda2 on /home type ext4 (rw,relatime,data=ordered)

...
```

- **Explanation**:
    - **/dev/sda1**: Mounted on / with filesystem type ext4.
    - **tmpfs**: Mounted on /run and /dev/shm with filesystem type tmpfs.
    - **/dev/sda2**: Mounted on /home with filesystem type ext4.

(b) Block Devices and Their Sizes

- **Command**:

```
lsblk
```

- **Output**:

```
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT

sda      8:0    0 232.9G  0 disk
```

```
├─sda1   8:1    0 118.5G  0 part /
├─sda2   8:2    0  98.9G  0 part /home
└─sda3   8:3    0  15.5G  0 part [SWAP]
sdb      8:16   0 465.8G  0 disk
├─sdb1   8:17   0 465.8G  0 part /mnt/external
```

- **Explanation**:
  - **sda**: A disk with a total size of 232.9G.
    - **sda1**: A partition with a size of 118.5G, mounted on /.
    - **sda2**: A partition with a size of 98.9G, mounted on /home.
    - **sda3**: A partition with a size of 15.5G, used as swap space.
  - **sdb**: An additional disk with a total size of 465.8G.
  - **sdb1**: A partition with a size of 465.8G, mounted on /mnt/external.

6. Identify a running process and find all its child processes. After this, send a SIGSTOP signal to a process to stop it and then a SIGCONT signal to resume it. Document the steps and the results.

Following is the commands to identify a running process and its child processes, and to send signals:

Identify a Running Process and Find All Its Child Processes

1. **Identify a Running Process**: First, find the PID of the process you are interested in. You can use the `ps` command to list processes.

```
ps aux
```

   Identify the PID of the process you want to investigate.

2. **Find Child Processes**: Use the pstree or ps --ppid command to find the child processes of the identified PID.

```
# Replace <PID> with the actual process ID

pstree -p <PID>
```

Or,

```
# Replace <PID> with the actual process ID
ps --ppid <PID>
```

Send a SIGSTOP Signal to a Process to Stop It

3. **Send SIGSTOP**: Use the kill command with the -SIGSTOP signal to stop the process.

```
# Replace <PID> with the actual process ID

kill -SIGSTOP <PID>
```

Send a SIGCONT Signal to Resume the Process

4. **Send SIGCONT**: Use the kill command with the -SIGCONT signal to resume the process.

```
# Replace <PID> with the actual process ID

kill -SIGCONT <PID>
```

Example of Collecting and Documenting the Steps and Results

Step-by-Step Documentation

1. **Identify a Running Process**:
    - **Command**:

```
ps aux
```

    - **Output** (example):

```
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME
COMMAND

user       1234  0.0  0.1  15920  1036 pts/0    Ss   10:00   0:00 bash

user       2345  0.0  0.2  46272  3456 pts/0    R+   10:05   0:00 ps aux
```

    - **Explanation**: We will use the process with PID 1234 (bash) for this example.
2. Find Child Processes:
    - **Command**:

```
pstree -p 1234
```

- **Output:**

```
bash(1234)─┬─pstree(2345)
           └─other_child_process(3456)
```

- **Explanation**: The bash process (PID 1234) has two child processes: pstree(PID 2345) and other_child_process (PID 3456).

3. **Send SIGSTOP to Stop the Process**:
    - **Command**:
    - **Explanation:** The bash process (PID 1234) is now stopped. To confirm, you can check the process status.

```
ps -p 1234 -o stat
```

- Expected output should include T, indicating the process is stopped.

4. **Send SIGCONT to Resume the Process**:
    - **Command**

```
kill -SIGSTOP 1234
```

- **Explanation:** The bash process (PID 1234) is now resumed. To confirm, you can check the process status again.

```
ps -p 1234 -o stat
```

Expected output should show the status without T, indicating the process is running.

7. Run ‘strace‘ on a system utility (e.g., ls). What are the first 10 system calls made by this utility? Also, run ‘strace‘ on two different utilities (e.g., cat and grep). Compare their system call traces and identify the unique system calls made by each utility.

Following is the command and steps to run strace and analyze system calls:

Step 1: Run `strace` on a System Utility (e.g., `ls`)

1. **Run `strace` on `ls`**:

```
strace -o ls_trace.txt ls
```

This command runs `strace` on the `ls` command and outputs the trace to `ls_trace.txt`.

2. **Extract the First 10 System Calls**:

```
head -n 10 ls_trace.txt
```

This command displays the first 10 lines of the `ls_trace.txt` file, which correspond to the first 10 system calls.

Step 2: Run `strace` on Two Different Utilities (e.g., `cat` and `grep`)

1. **Run `strace` on `cat`**:

```
strace -o cat_trace.txt cat /dev/null
```

This command runs `strace` on the `grep` command with an empty string and `/dev/null` as the input and outputs the trace to `grep_trace.txt`.

2. **Run `strace` on `grep`**:

```
strace -o grep_trace.txt grep "" /dev/null
```

This command runs `strace` on the `grep` command with an empty string and `/dev/null` as the input and outputs the trace to `grep_trace.txt`.

Step 3: Compare System Call Traces and Identify Unique System Calls

1. **Extract System Calls from `cat_trace.txt`**:

```
awk '{print $1}' cat_trace.txt | sort | uniq > cat_syscalls.txt
```

This command extracts the system call names from `cat_trace.txt`, sorts them, and saves the unique names to `cat_syscalls.txt`.

2. **Extract System Calls from `grep_trace.txt`**:

```
awk '{print $1}' grep_trace.txt | sort | uniq > grep_syscalls.txt
```

This command extracts the system call names from `grep_trace.txt`, sorts them, and saves the unique names to `grep_syscalls.txt`.

3. **Identify Unique System Calls for `cat`**:

```
comm -23 cat_syscalls.txt grep_syscalls.txt > unique_cat_syscalls.txt
```

This command finds system calls that are unique to `cat` and saves them to `unique_cat_syscalls.txt`.

4. **Identify Unique System Calls for `grep`**:

```
comm -13 cat_syscalls.txt grep_syscalls.txt > unique_grep_syscalls.txt
```

This command finds system calls that are unique to `grep` and saves them to `unique_grep_syscalls.txt`.

Example of Documenting the Steps and Results

Step 1: First 10 System Calls Made by `ls`

- **Command**:

```
strace -o ls_trace.txt ls

head -n 10 ls_trace.txt
```

- **Output** (example):

```
execve("/bin/ls", ["ls"], 0x7ffec9e7f9d8 /* 63 vars */) = 0

brk(NULL)                                = 0x562a4b6ab000

arch_prctl(0x3001 /* ARCH_??? */, 0x7ffd69d106d0) = -1 EINVAL (Invalid
argument)

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,
0) = 0x7f715d92e000

access("/etc/ld.so.preload", R_OK)       = -1 ENOENT (No such file or
directory)
```

```
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=139579, ...},
AT_EMPTY_PATH) = 0

mmap(NULL, 139579, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f715d90d000

close(3)                                = 0

openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC)
= 3
```

Step 2: System Call Traces for `cat` and `grep`

1. **System Calls for `cat`**:
    - **Command**:

```
strace -o cat_trace.txt cat /dev/null

awk '{print $1}' cat_trace.txt | sort | uniq > cat_syscalls.txt
```

    - **Output** (example from `cat_syscalls.txt`):

```
brk

close

execve

fstat

mmap

openat

read
```

2. **System Calls for `grep`**:
    - **Command**:

```
strace -o grep_trace.txt grep "" /dev/null

awk '{print $1}' grep_trace.txt | sort | uniq > grep_syscalls.txt
```

    - **Output** (example from `grep_syscalls.txt`):

```
brk

close

execve

fstat

mmap

openat

read

write
```

Step 3: Unique System Calls

1. **Unique System Calls for `cat`**:
    - **Command**:

```
comm -23 cat_syscalls.txt grep_syscalls.txt > unique_cat_syscalls.txt
```

    - **Output** (example from `unique_cat_syscalls.txt`):

```
(empty if no unique system calls)
```

2. **Unique System Calls for `grep`**:
    - **Command**:

```
comm -13 cat_syscalls.txt grep_syscalls.txt > unique_grep_syscalls.txt
```

    - **Output** (example from `unique_grep_syscalls.txt`):

```
write
```