ITEC340

Dr Ahmad Faour

Performance Monitoring Tool

By:

Georgios Abboud, Jason Abdalah, Alexandre younes

Version: 1.0

3/5/2024

Project definition:

Implementation of a script that monitors system performance metrics such as CPU usage, memory usage, disk I/O, and network I/O in real-1me and logs the data for analysis.

Installation and setup:

For a Unix user to run the script you've provided, the requirements largely depend on the specific commands and utilities the script uses. here's a list of tools and commands that the script use:

1. Bash Shell: The script is a Bash script, indicated by the shebang #!/bin/bash. Bash is usually pre-installed on most Unix/Linux systems.

2. Core Utilities (touch, chmod, dirname, etc.): These are basic commands used in the script and are part of the GNU core utilities. They are standard on almost all Unix/Linux distributions and should be pre-installed.

3. awk: Used for text processing in the script. It is part of the GNU core utilities and should be pre-installed on most Unix/Linux systems.

4. grep: This command is used for pattern searching in files and output. It is also part of the GNU core utilities and should be pre-installed.

5. ip and ifconfig: For network interface and traffic information. ifconfig is part of the net-tools package, which might not be installed by default on newer systems as it's being replaced by the ip command from the iproute2 package.

6. sensors: For reading CPU temperature. This requires the lm-sensors package.

7. ps: For process information. It is part of the procps package, which is usually pre-installed.

8. df and lsblk: For disk usage and block device information. These are typically pre-installed.

9. top: For CPU usage. It is part of the procps package, which is usually pre-installed.

10. ss or netstat: For network connection statistics. ss is part of the iproute2 package, and netstat is part of net-tools. One or both might be pre-installed, but newer systems tend to favor ss.

11. lscpu: For CPU architecture information. It is part of the util-linux package, which should be pre-installed.

If some of these tools are not installed...

If the necessary tools (ps, df, lsblk, top, ss/netstat, lscpu) are not installed on your Unix/Linux system, you can install them using your distribution's package manager.

Tools necessary to install:

For Debian, Ubuntu, and derivatives:

These distributions use the `apt` package manager. The `procps`, `net-tools`, `iproute2`, and `util-linux` packages can be installed with the following commands:

```
sudo apt install procps net-tools iproute2 util-linux
```

For Fedora, CentOS (version 8 and earlier), Red Hat:

Fedora uses `dnf`, and CentOS/RHEL 8 and earlier versions use `yum` as their package manager. The commands are similar for both managers:

```
 # Use `yum` instead of `dnf` for CentOS/RHEL 8 and earlier
sudo dnf install procps-ng net-tools iproute util-linux
```

`lm-sensors for CPU temperature monitoring`, they can be installed using the system's package manager. For systems using apt (like Debian, Ubuntu, and derivatives), the installation commands would be:

```
sudo apt install lm-sensors net-tools iproute2
```

For Fedora, CentOS (version 8 and earlier), Red Hat:

Use `dnf` to install the packages:

```
# Use `yum` instead of `dnf` for CentOS/RHEL 8 and earlier
sudo dnf install lm_sensors net-tools iproute
```

After installing, you might need to run `sudo sensors-detect` to set up `lm-sensors`.

To run the script:

Make the Script Executable

Before you can run the script, you need to make it executable. This is done by setting the execute permission on the file. Open a terminal and navigate to the directory where main.sh is saved. Run the following command:

```
chmod +x main.sh
```

This command changes the mode of the file to add (+) execute (x) permissions.

Run the Script

With the script now executable, you can run it directly from the terminal. the script, use the following command:

```
/path/main.sh
```

Script download:

You can download the script from:

https://github.com/alex6t5/PerfMonShell

or

https://drive.google.com/drive/folders/10We9rzwnUes21HJoop1IxZ4HcKPAPxSo?usp=sharing

or

https://sgub-my.sharepoint.com/:f:/g/personal/202200442_sgub_edu_lb/Eqzm4qQuHOtDoeRy9GO_R_IBrWQWtkX6EyJ8NphN2Jq5aw?e=MqsD4z

How does the script work:

This Script is designed to continuously monitor and display system information, including CPU, memory, network, and disk usage, with an interactive mode for additional actions. Here's a detailed overview of its functionality and user interaction:

Core Functionality

- **Continuous Monitoring:** Automatically updates and displays system information in real-time, refreshing every 10 seconds.
- **Interactive Pause:** Allows the user to pause the automatic refresh by pressing 'p', enabling access to a menu for additional actions.

User Interaction and Options

- **Automatic Refresh:** If no input is received within 10 seconds, the script refreshes the displayed system information.
- **Pause and Menu:** Pressing 'p' pauses the refresh and presents a menu with options to log detailed information or quit.
- **Logging Options:** The user can choose to log detailed network, memory, disk, or CPU information to respective log files.
- **Resuming Monitoring:** After an action is selected or logging is completed, the user can resume the automatic refresh.
- **Quitting:** The user has the option to exit the monitoring script.

Behind the Scenes

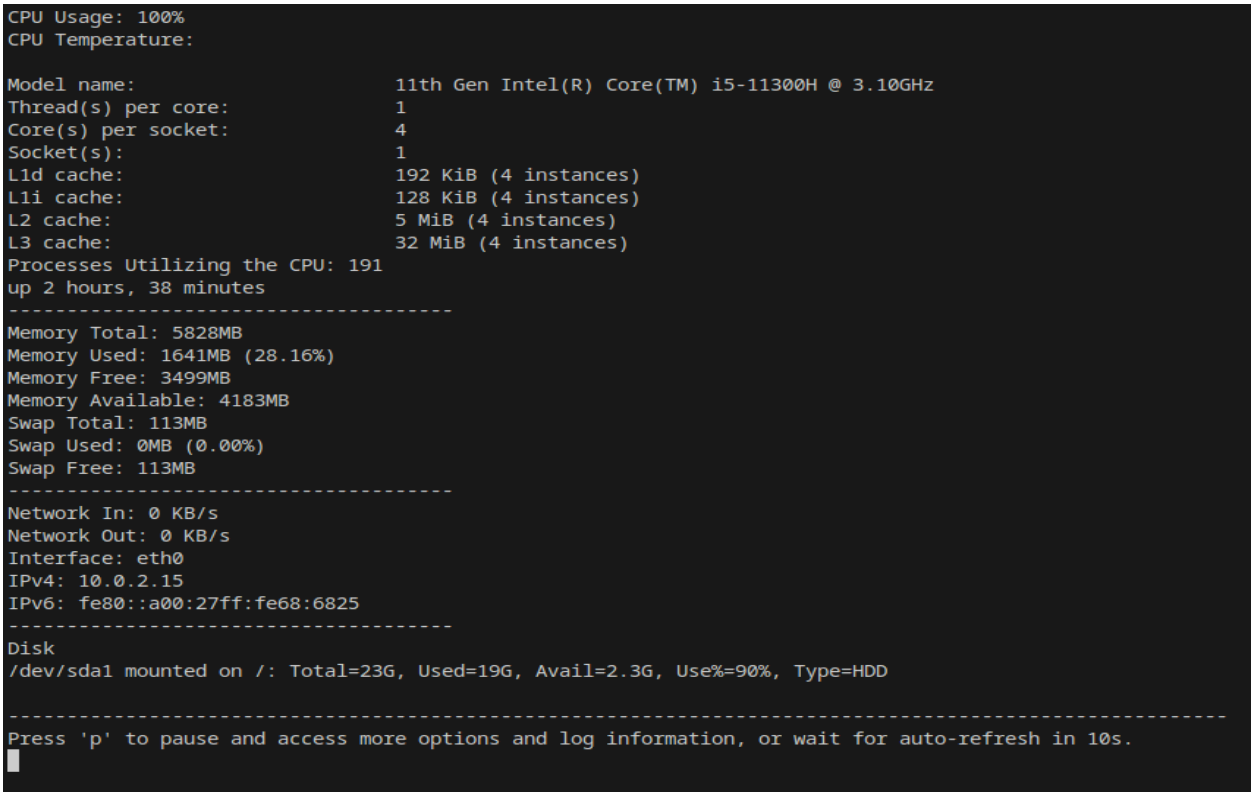
System Information Display: Utilizes functions like display_cpu_info, display_memory_usage, get_network_traffic, get_network_info, and get_disk_usage to gather and display relevant system metrics.

Logging to Files: Specific functions (log_network_info_to_file, log_memory_usage_to_file, log_disk_usage_to_file, log_cpu_info_to_file) are called based on user selection to log detailed information into predefined log files.

User Input Handling: Uses the read command with a timeout for the initial pause prompt and without a timeout for menu selection, capturing user choices and controlling the script's flow based on those choices.

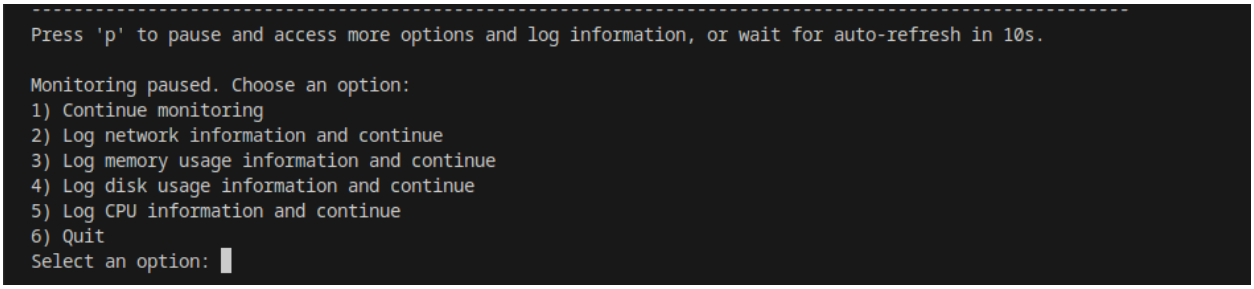
How to use:

The script will start monitoring cpu usage example image below:



1. The script will continuously monitor and display system information, and will refresh every 10 seconds

2. If the user presses “p” it will pause the continues monitoring and the user will get additional options.



3. The user has the option to log specific information by selecting one of the available choices, or to continue with real-time monitoring, or exit.

Script explanation:

Logs variable

```
#!/bin/bash

# Define log file paths
CPU_LOG_FILE="$(dirname "$0")/cpu_logs.log"
NETWORK_LOG_FILE="$(dirname "$0")/network_logs.log"
DISK_LOG_FILE="$(dirname "$0")/disk_logs.log"
MEMORY_LOG_FILE="$(dirname "$0")/memory_logs.log"
```

Sets variables for the paths to four log files, named according to the category of system information they will store. The log files are located in the same directory as the script.

```
# Ensure the log files exist
for LOG_FILE in "$CPU_LOG_FILE" "$NETWORK_LOG_FILE" "$DISK_LOG_FILE"
"$MEMORY_LOG_FILE"; do
    if [ ! -e "$LOG_FILE" ]; then
        touch "$LOG_FILE"
        chmod 0640 "$LOG_FILE"
    fi
done
```

Iterates over each log file path, creating the file with touch if it does not exist, and setting permissions to read-write for the owner and read-only for the group with chmod 0640, ensuring no access for others.

Get_network_info function

```
# Function to get network information
get_network_info() {
    # Detect Wi-Fi and Ethernet interfaces
    wifi_interface=$(iw dev | awk '$1=="Interface"{print $2}')
    ethernet_interface=$(ip link | awk -F': ' '$0 !~ "lo|vir|wl|^[^0-
9]"{print $2;getline}')
    # Wi-Fi SSID, if available
```

```
    if [ -n "$wifi_interface" ]; then
        ssid=$(iwgetid -r)
        echo "Wi-Fi Interface: $wifi_interface"
        echo "SSID: $ssid"
    fi
    # IP addresses for all interfaces
    for interface in $wifi_interface $ethernet_interface; do
        if [ -n "$interface" ]; then
            ipv4=$(ip -4 addr show "$interface" | grep -oP
'(?<=inet\s)\d+(\.\d+){3}')
            ipv6=$(ip -6 addr show "$interface" | grep -oP
'(?<=inet6\s)[\da-f:]+')
            echo "Interface: $interface"
            echo "IPv4: $ipv4"
            echo "IPv6: $ipv6"
            echo "--------------------------------------"
        fi
    done
}
```

This function collects and displays network information, including Wi-Fi and Ethernet interface details, SSID for Wi-Fi, and IP addresses for all detected interfaces.

Detecting Network Interfaces

Wi-Fi Interface Detection: Uses iw dev to list Wi-Fi devices and extracts the interface name(s) using awk.

Ethernet Interface Detection: Utilizes ip link to list network interfaces and filters out non-Ethernet ones (like loopback lo, virtual vir, and Wi-Fi wl) using awk.

Wi-Fi SSID

Checks if a Wi-Fi interface is detected. If so, it retrieves the current SSID using iwgetid -r and displays the Wi-Fi interface name along with the SSID.

IP Addresses for Interfaces

Iterates over the detected Wi-Fi and Ethernet interfaces.

For each interface, it retrieves the IPv4 and IPv6 addresses using ip addr show and grep with a regular expression to match the IP addresses.

Displays the interface name along with its IPv4 and IPv6 addresses, if available.

Get_network_traffic function

```
# Function to get the number of bytes received and transmitted by the
interface
get_network_traffic() {
    # Dynamically determine the active network interface if not set
    if [ -z "$INTERFACE" ]; then
        INTERFACE=$(ip route | grep '^default' | awk '{print $5}' | head -
n 1)
    fi

    # Ensure INTERFACE is not empty
    if [ -z "$INTERFACE" ]; then
        echo "No active network interface found."
        return 1
    fi

    # Paths to the network statistics
    rx_bytes_path="/sys/class/net/$INTERFACE/statistics/rx_bytes"
    tx_bytes_path="/sys/class/net/$INTERFACE/statistics/tx_bytes"

    # Check if the statistics files exist
    if [ ! -f "$rx_bytes_path" ] || [ ! -f "$tx_bytes_path" ]; then
        echo "Network statistics not available for interface $INTERFACE."
        return 1
    fi

    # Read the initial number of bytes received and transmitted
    rx_bytes_before=$(cat "$rx_bytes_path")
    tx_bytes_before=$(cat "$tx_bytes_path")
    sleep 1 # Wait for a second to measure the traffic

    # Read the number of bytes received and transmitted after 1 second
    rx_bytes_after=$(cat "$rx_bytes_path")
    tx_bytes_after=$(cat "$tx_bytes_path")

    # Calculate the difference and convert to KB
    rx_diff_kb=$(((rx_bytes_after - rx_bytes_before) / 1024))
    tx_diff_kb=$(((tx_bytes_after - tx_bytes_before) / 1024))

    echo "Network In: $rx_diff_kb KB/s"
    echo "Network Out: $tx_diff_kb KB/s"
}
```

The get_network_traffic function in the script performs real-time network traffic monitoring for the primary network interface on a Unix/Linux system. Here's a more detailed breakdown of its operations:

1. Dynamic Interface Selection: Initially, the function checks if an environment variable $INTERFACE is set to specify the network interface to monitor. If not set, it automatically identifies the primary network interface used for the default route to the internet. This is achieved by parsing the output of ip route, which lists network routes, and extracting the interface name associated with the default route.

2. Validation and Setup: The function then verifies that an interface has been successfully identified. If no interface is found or specified, it exits with an error message indicating that no active network interface could be found. For the identified interface, it constructs file paths to access the system's network statistics specifically for received (rx_bytes) and transmitted (tx_bytes) bytes. These statistics are maintained by the kernel and available in the /sys/class/net/<interface>/statistics/ directory for each network interface.

3. Initial Traffic Measurement: Before measuring network traffic, the function reads the initial values of received and transmitted bytes from the respective files. These values represent the total bytes received and transmitted by the interface since it was activated.

4. Interval Waiting: The script then pauses for one second using sleep 1. This delay allows network activity to occur, which can be measured by comparing the byte counts before and after the pause.

5. Final Traffic Measurement: After the pause, the function reads the new values of received and transmitted bytes. These values are now compared with the initial readings to calculate the difference, representing the bytes received and transmitted during the one-second interval.

6. Calculation and Output: The differences in bytes are then converted to kilobytes (KB) by dividing by 1024. The resulting values represent the inbound (received) and outbound (transmitted) network traffic rates in KB/s. These rates are printed to the console.

Get_disk_usage Function

```
# Function to get disk usage with capacity and type
get_disk_usage() {
    echo "Disk"
    df -h | grep '^/dev/' | while IFS= read -r line; do
        device=$(echo "$line" | awk '{print $1}')
        mount_point=$(echo "$line" | awk '{print $6}')
        size=$(echo "$line" | awk '{print $2}')
        used=$(echo "$line" | awk '{print $3}')
        avail=$(echo "$line" | awk '{print $4}')
        use_perc=$(echo "$line" | awk '{print $5}')
        # Determine if SSD or HDD
        type=$(lsblk -no ROTA "$device" | awk '{if ($1=="1") print "HDD";
else print "SSD"}')
        echo "$device mounted on $mount_point: Total=$size, Used=$used,
Avail=$avail, Use%=$use_perc, Type=$type"

    done
}
```

The get_disk_usage function in the script provides a detailed report on the disk usage of mounted devices, including their capacity, used space, available space, usage percentage, and whether they are SSDs or HDDs.

1. Disk Usage Overview: The function starts by invoking df -h, which lists all mounted filesystems along with their sizes in a human-readable format. It filters this list to include only devices that are typically physical disk partitions (those entries starting with /dev/).

2. Parsing Disk Information: For each line corresponding to a mounted device, the script extracts and assigns the device name, mount point, total size, used space, available space, and usage percentage using awk.

3. Determining Disk Type (SSD or HDD): For each device, the script then determines whether it is an SSD or HDD. This is achieved by using lsblk -no ROTA "$device", which queries the rotational characteristic of the device. A value of 1 indicates a rotational device (HDD), while a value of 0 indicates a non-rotational device (SSD). This information is processed with awk to output "HDD" or "SSD" accordingly.

4. Output: Finally, for each device, the script prints a summary line that includes the device name, mount point, total size, used space, available space, usage percentage, and the determined type (SSD or HDD).

Display_memory_usage function

```
display_memory_usage() {
    total_mem=$(free -m | awk '/Mem:/ {print $2}')
    used_mem=$(free -m | awk '/Mem:/ {print $3}')
    free_mem=$(free -m | awk '/Mem:/ {print $4}')
    available_mem=$(free -m | awk '/Mem:/ {print $7}')
    used_mem_perc=$(awk "BEGIN {printf \"%.2f\",
($used_mem/$total_mem)*100}")

    # Swap space details
    total_swap=$(free -m | awk '/Swap:/ {print $2}')
    used_swap=$(free -m | awk '/Swap:/ {print $3}')
    free_swap=$(free -m | awk '/Swap:/ {print $4}')
    used_swap_perc=$(awk "BEGIN {printf \"%.2f\",
($used_swap/$total_swap)*100}")

    echo "Memory Total: ${total_mem}MB"
    echo "Memory Used: ${used_mem}MB (${used_mem_perc}%)"
    echo "Memory Free: ${free_mem}MB"
    echo "Memory Available: ${available_mem}MB"
    echo "Swap Total: ${total_swap}MB"
    echo "Swap Used: ${used_swap}MB (${used_swap_perc}%)"
    echo "Swap Free: ${free_swap}MB"
    echo "--------------------------------------"

}
```

The display_memory_usage function provides a detailed report on the system's memory (RAM) and swap space usage, including total, used, free, and available memory, as well as the percentage of memory used. It also details swap space usage in a similar manner.

1. Memory Usage Calculation: The function uses free -m to fetch memory and swap usage data in megabytes. It parses this output with awk to extract total, used, and free memory, as well as the total, used, and free swap space. The available memory (memory available for starting new applications without swapping) is also extracted.

2. Percentage Calculations: It calculates the percentage of used memory and swap space with respect to their total capacities. These calculations are formatted to two decimal places for readability.

3. Output: The function then prints a summary of these statistics to the console.

Display_cpu_info function

```
display_cpu_info() {
    cpu_usage=$(top -bn1 | grep "Cpu(s)" | awk '{print 100 - $8"%"}')
    cpu_temp=$(sensors | awk '/^Core 0:/ {print $3}')
    cpu_info=$(lscpu | grep -E 'Model name|Socket\(s\)|Core\(s\) per
socket|Thread\(s\) per core|L1d cache|L1i cache|L2 cache|L3 cache')
    process_count=$(ps -e --no-headers | wc -l)
    uptime_info=$(uptime -p)

    echo "CPU Usage: $cpu_usage"
    echo "CPU Temperature: $cpu_temp"
    echo
    echo "$cpu_info"
    echo "Processes Utilizing the CPU: $process_count"
    echo "$uptime_info"
    echo "--------------------------------------"
}
```

The display_cpu_info function gathers and displays comprehensive information about the CPU's current state, including usage, temperature, specifications, and system process count.

1. CPU Usage: Utilizes top -bn1 to fetch the current CPU usage snapshot, extracting the idle time with awk and subtracting it from 100% to get the total CPU usage percentage.

2. CPU Temperature: Retrieves the temperature of the first CPU core using sensors, a tool for reading hardware sensor data, and awk to parse the output for the temperature of "Core 0".

3. CPU Specifications: Gathers detailed CPU information using lscpu, which lists important CPU architecture information like model name, number of sockets, cores per socket, threads per core, and cache sizes. It filters this information to include only the most relevant details using grep.

4. Process Count: Counts the total number of processes currently running on the system using ps -e --no-headers | wc -l, which lists all processes and pipes the output to wc -l to count the lines.

5. System Uptime: Fetches the system's uptime in a human-readable format using uptime -p, showing how long the system has been running since the last reboot.

6. Output: Prints the gathered CPU usage, temperature, detailed specifications, total process count, and system uptime to the console.

log_network_info_to_file function

```
# Enhanced log_network_info_to_file function with additional network
details
log_network_info_to_file() {
    echo "Logging network information to $NETWORK_LOG_FILE... Press any
button to continue"
    {
        echo "--------------------------------------"
        echo "Network Information Log: $(date '+%Y-%m-%d %H:%M:%S')"
        echo "--------------------------------------"

        # Existing network info and traffic
        get_network_info
        get_network_traffic

        # Network interface statistics
        echo -e "\nNetwork Interface Statistics:"
        ifconfig | awk '/flags/{print $1} /RX packets/{print "Received: "
$5} /TX packets/{print "Transmitted: " $5 "\n"}'

        # Connection statistics (requires netstat or ss)
        echo -e "\nCurrent Network Connections:"
        ss -tuln

        echo "--------------------------------------"
    } >>"$NETWORK_LOG_FILE"
}
```

The log_network_info_to_file function captures and logs detailed network information to a specified file. It starts by notifying the user about the logging process, then records a timestamped entry that includes network configurations, real-time traffic data, interface statistics, and active network connections. This information is obtained through calls to **get_network_info**, **get_network_traffic**, and system commands like ifconfig and ss. The logged data provides a comprehensive snapshot of the network's current state.

log_memory_usage_to_file function

```
# Enhanced log_memory_usage_to_file function with additional memory
details
log_memory_usage_to_file() {
    echo "Logging enhanced memory usage information to $MEMORY_LOG_FILE...
Press any button to continue"
    {
        echo "Enhanced Memory Usage Information Log: $(date '+%Y-%m-%d
%H:%M:%S')"
        display_memory_usage
        # Display buffers and cache
        free -m | awk '/Mem:/ {printf "Buffers/Cache: Used: %sMB, Free:
%sMB\n", $6, $7}'
        # Display top memory-consuming processes
        echo "Top memory-consuming processes:"
        ps aux --sort=-%mem | head -n 11
        echo "--------------------------------------"
    } >>"$MEMORY_LOG_FILE"
}
```

The log_memory_usage_to_file function logs detailed memory usage information to a specified file. It prompts the user before proceeding and includes a timestamp for when the log is made. The function utilizes **display_memory_usage** to show general memory statistics, then adds details on buffers and cache usage directly from the free command. Additionally, **it lists the top memory-consuming processes using ps**, sorted by memory usage. This comprehensive memory usage snapshot is appended to the memory log file.

log_cpu_info_to_file function

```
log_cpu_info_to_file() {
    echo "Logging CPU information to $CPU_LOG_FILE... Press any button to
continue"
    {
        echo "--------------------------------------"
        echo "CPU Information Log: $(date '+%Y-%m-%d %H:%M:%S')"
        echo "--------------------------------------"
```

```
        # CPU usage and temperature
        cpu_usage=$(top -bn1 | grep "Cpu(s)" | awk '{print 100 - $8"%"}')
        cpu_temp=$(sensors | awk '/^Core 0:/ {print $3}')
        echo "CPU Temperature: $cpu_temp"

        # Detailed CPU information
        cpu_info=$(lscpu | grep -E 'Model name|Socket\(s\)|Core\(s\) per
socket|Thread\(s\) per core|L1d cache|L1i cache|L2 cache|L3 cache')
        echo "$cpu_info"

        # Top 10 processes consuming the most CPU
        echo "Top 10 CPU-consuming processes:"
        ps -eo %cpu,pid,user,command --sort=-%cpu | head -n 11
        echo

        process_count=$(ps -e --no-headers | wc -l)
        uptime_info=$(uptime -p)

        echo "Processes Utilizing the CPU: $process_count"
        echo "$uptime_info"

        echo "--------------------------------------"
    } >>"$CPU_LOG_FILE"
}
```

The log_cpu_info_to_file function is designed to log a comprehensive set of CPU-related information into a designated file, $CPU_LOG_FILE. It starts by prompting the user to proceed with the logging process. Here's a breakdown of its operations with a bit more detail for clarity:

1. CPU Usage and Temperature:

CPU Usage: Retrieves the current CPU usage percentage by parsing the output of the top command. It specifically looks for the line indicating CPU idle time and calculates the usage percentage from it.

CPU Temperature: Gathers the temperature of the first CPU core using the sensors command, which reads hardware sensor data for temperature monitoring.

2. Detailed CPU Information: Executes the lscpu command to collect extensive details about the CPU architecture, including model name, number of sockets, cores per socket, threads per core, and cache sizes.

Top CPU-consuming Processes: Lists the top 10 processes by CPU usage to identify which processes are consuming the most CPU resources at the time of logging. This is achieved by sorting the output of the ps command by CPU usage percentage.

3. Additional System Information: Also lists: Process Count, System Uptime

4. Logging: All the collected information—from CPU usage and temperature to detailed CPU specifications and top CPU-consuming processes—is then appended to the specified CPU log file. This ensures a historical record of CPU performance and system activity.

update_monitoring function

```
update_monitoring() {
    local choice
    local pause=0 # Flag to control pausing

    while true; do
        clear

        display_cpu_info
        display_memory_usage
        get_network_traffic
        get_network_info
        get_disk_usage

        if [[ $pause -eq 0 ]]; then
            echo
            echo "------------------------------------------------------------
-----------------------------------------------"
            echo "Press 'p' to pause and access more options and log
information, or wait for auto-refresh in 10s."

            read -t 10 -n 1 -s -r choice
            echo
            if [[ $choice == "p" || $choice == "P" ]]; then
                pause=1
            fi
        fi

        if [[ $pause -eq 1 ]]; then
            echo "Monitoring paused. Choose an option:"
            echo "1) Continue monitoring"
```

```
            echo "2) Log network information and continue"
            echo "3) Log memory usage information and continue"
            echo "4) Log disk usage information and continue"
            echo "5) Log CPU information and continue"
            echo "6) Quit"
            read -n 1 -s -r -p "Select an option: " choice
            echo

            case $choice in
            1)
                pause=0
                ;;
            2)
                log_network_info_to_file
                read -n 1 -s -r # Pause for user to acknowledge
                pause=0
                ;;
            3)
                log_memory_usage_to_file
                read -n 1 -s -r # Pause for user to acknowledge
                pause=0
                ;;
            4)
                log_disk_usage_to_file
                read -n 1 -s -r # Pause for user to acknowledge
                pause=0
                ;;
            5)
                log_cpu_info_to_file
                read -n 1 -s -r # Pause for user to acknowledge
                pause=0
                ;;
            6)
                echo "Quitting..."
                exit 0
                ;;
            *)
                echo "Invalid choice, please select a valid option."
                ;;
            esac
        fi
    done
}
# Start the monitoring loop
update_monitoring
```

The update_monitoring function in the script is a continuous loop designed for **real-time system monitoring**, offering both automatic refresh and interactive options for logging detailed system information.

1. Continuous Monitoring Loop: The function enters an infinite loop, continuously updating and displaying system information until explicitly exited by the user.

2. Display System Information: On each iteration, it clears the screen and displays current CPU information, memory usage, network traffic, network configuration details, and disk usage by calling respective functions for each category.

3. Interactive Pause Mechanism:

The script waits for user input for 10 seconds, during which the user can press 'p' to pause the automatic refresh. If 'p' is pressed, the script enters a paused state, allowing the user to interact with additional options.

If no input is received within 10 seconds, the script automatically refreshes the displayed information.

4. Paused State Options: When paused, the user is presented with options to:

Continue monitoring without logging (resumes the loop).

Log network, memory, disk, or CPU information to their respective log files and then resume monitoring.

Quit the monitoring script entirely.

5. User Input Handling: The script uses read to capture user input, adjusting its behavior based on the selected option. For logging actions, it calls specific functions to append detailed information to log files, then waits for an additional key press before resuming monitoring.

6. Loop Control and Exit: The loop continues indefinitely until the user chooses to quit. The pause flag is toggled based on user actions, controlling whether the script is in an active monitoring state or a paused, interactive state.