

# Question\_1 Report

## **Script Overview:**

The provided shell script, named `server_monitor.sh`, automates the management and monitoring of server resources.

Given Tasks:-

### **1. Disk Usage Monitoring:**

Monitors disk usage and generates an alert if it exceeds a specified threshold (default: 30%).

### **2. CPU Usage Monitoring:**

Monitors CPU usage and generates an alert if it exceeds a specified threshold (default: 30%).

### **3. Memory Usage Monitoring:**

Monitors memory (RAM) usage and generates an alert if available memory falls below a specified threshold (default: 10% free).

### **4. Log Rotation:**

Implements log rotation for a specified log file to prevent it from growing beyond a specified size (default: 10MB).

## **Script Structure:**

### **Log Rotation Function:**

- This function check the log file exceeds the limit.

- Create new file if rotation is required and change the current file of log to **.old**

### Main portion of script:

- Creating log file.
- Functions calls for Disk, Memory, and CPU.
- Maintaining the information of script start and end time

### How to use script:

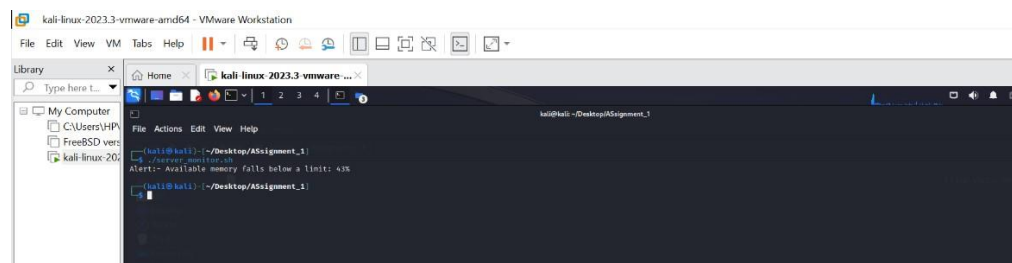
- Use Linux server to execute the script.
- Check the log file to monitor resources, any alerts script start end time and errors.

### Output:

This script is not showing directly output to the file except alerts. To review logs you must use following command:

```
(kali@kali)~(/Desktop/Assignment_1)
$ cat /var/log/server_monitor.log
Fri Oct 13 06:55:49 AM EDT 2023: Server monitoring tasks completed.
Fri Oct 13 06:56:14 AM EDT 2023: Server monitoring tasks completed.
Fri Oct 13 08:14:08 AM EDT 2023 - Error occurred:
Fri Oct 13 08:14:18 AM EDT 2023 - Error occurred:
Fri Oct 13 08:17:26 AM EDT 2023 - Error occurred:
Fri Oct 13 08:17:36 AM EDT 2023 - Error occurred:
Fri Oct 13 08:17:47 AM EDT 2023 - Error occurred:
Fri Oct 13 08:17:57 AM EDT 2023 - Error occurred:
Fri Oct 13 08:20:00 AM EDT 2023 - Server monitoring started
Fri Oct 13 08:20:00 AM EDT 2023 - Server monitoring ended
Fri Oct 13 08:26:11 AM EDT 2023 - Server monitoring started
Fri Oct 13 08:26:11 AM EDT 2023 - Memory usage threshold exceeded: %
Fri Oct 13 08:26:11 AM EDT 2023 - Server monitoring ended
2023-10-13 08:27:47 - Server monitoring script started.
2023-10-13 08:27:48 - Log file not found: /path/to/your/log/file.log
2023-10-13 08:27:48 - Server monitoring script completed.
2023-10-13 08:29:28 - Server monitoring script started.
2023-10-13 08:29:28 - Log file not found: /path/to/your/log/file.log
2023-10-13 08:29:28 - Server monitoring script completed.
2023-10-13 08:29:28 - Server monitoring script completed.
2023-10-13 08:29:37 - Server monitoring script started.
2023-10-13 08:29:37 - Log file not found: /path/to/your/log/file.log
2023-10-13 08:29:37 - Server monitoring script completed.
2023-10-13 08:29:37 - Server monitoring script completed.
2023-10-13 08:30:19 - Server monitoring script started.
2023-10-13 08:30:19 - Log file not found: /path/to/your/log/file.log
2023-10-13 08:30:19 - Server monitoring script completed.
```

- Also execute your file to check there is any alert



- If your file is not executing then try this:

```
kali@kali: ~/Desktop/OS/A1
File Actions Edit View Help
kali@kali:~/Desktop/OS/A1$ sudo apt install bc
[sudo] password for kali:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
bc is already the newest version (1.07.1-3+b1).
The following packages were automatically installed and are no longer required:
gcc-12-base libarmadillo11 libcodecs2-1.1 libcurl3-nss libgcc-12-dev libgumbo1 libgupnp-igd-1.0-4
libjim0.81 libnfs13 libobjc-12-dev libstdc++-12-dev libtclap2 nss-plugin-pem python3-jdcal
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 1 not upgraded.
kali@kali:~/Desktop/OS/A1$ sudo chown kali /var/log
kali@kali:~/Desktop/OS/A1$ sudo chmod u+w /var/log
kali@kali:~/Desktop/OS/A1$
```

## Conclusion:

The server\_monitor.sh script provides basic server resource monitoring and log rotation functionality.

# Question\_2 Report

## Compilation and Running Commands:

1. gcc instruction\_simulator.c -o t2
2. ./t2

## Output :

```
kali@kali: ~/Desktop/OS/A1
File Actions Edit View Help
(kali@kali)~[~/Desktop/OS/A1]
$ gcc instruction_simulator.c -o t2
(kali@kali)~[~/Desktop/OS/A1]
$ ./t2
Before execution of instruction no 1:
PC=0 MAR=0 MBR=393226 IR=393226 ACC=10
Memory: [0]=393226 [1]=17170442 [2]=67108864 [3]=0 [4]=0 [5]=0 [6]=42 [7]=0 [8]=0 [9]=0 [10]=30 [11]=0 [12]=0 [13]=0 [14]=0 [15]=0 [16]=0 [17]=0 [18]=0 [19]=0

After execution of instruction no 1:
PC=0 MAR=0 MBR=393226 IR=393226 ACC=52
Memory: [0]=393226 [1]=17170442 [2]=67108864 [3]=0 [4]=0 [5]=0 [6]=42 [7]=0 [8]=0 [9]=0 [10]=30 [11]=0 [12]=0 [13]=0 [14]=0 [15]=0 [16]=0 [17]=0 [18]=0 [19]=0

Before execution of instruction no 2:
PC=1 MAR=1 MBR=17170442 IR=17170442 ACC=52
Memory: [0]=393226 [1]=17170442 [2]=67108864 [3]=0 [4]=0 [5]=0 [6]=42 [7]=0 [8]=0 [9]=0 [10]=30 [11]=0 [12]=0 [13]=0 [14]=0 [15]=0 [16]=0 [17]=0 [18]=0 [19]=0

After execution of instruction no 2:
PC=1 MAR=1 MBR=17170442 IR=17170442 ACC=10
Memory: [0]=393226 [1]=17170442 [2]=67108864 [3]=0 [4]=0 [5]=0 [6]=42 [7]=0 [8]=0 [9]=0 [10]=30 [11]=0 [12]=0 [13]=0 [14]=0 [15]=0 [16]=0 [17]=0 [18]=0 [19]=0

Before execution of instruction no 3:
PC=2 MAR=2 MBR=67108864 IR=67108864 ACC=10
Memory: [0]=393226 [1]=17170442 [2]=67108864 [3]=0 [4]=0 [5]=0 [6]=42 [7]=0 [8]=0 [9]=0 [10]=30 [11]=0 [12]=0 [13]=0 [14]=0 [15]=0 [16]=0 [17]=0 [18]=0 [19]=0
(kali@kali)~[~/Desktop/OS/A1]
$
```

## Explanation of code:

- An enum named opcode and defines operations like ADD, SUB, LOAD, STORE and HALT.
- Variables of registers and memory. Memory is an array of size 20 to store memory location of data and instruction. Variables to cater the number of instruction is also made.
- Random instructions are loaded in memory to avoid null output.

- For Fetch ,decode and execute cycle a while loop is used that runs till the end of instructions.
- First fetch the next instruction and load it in PC register as memory address.
- Print the memory variable and all registers before the execution of instruction.
- Gets opcode and operands which are actual the data from IR.
- Checks if the instructions opcode is HALT the loop breaks.
- Next a switch is used that checks the opcode of instruction and execute according to it.
- Print the memory and registers after execution of instruction.
- Increment PC and the number of instruction.
- When the instructions finish the loop exits and return 0 is executed.

**Conclusion:**

The above code explains the fundamental principles of instruction execution in computer system and how memory and registers change throughout the process.