OPERATING SYSTEMS

ASSIGNMENT 1

ABDUL SAMI QASIM
(22I-1725)
CY-C

TO:
SIR BILAL HASSAN

QUESTION 1

Server Monitoring Script Report

This report provides an analysis of the server monitoring script provided. The script is designed to monitor disk usage, CPU usage, available memory, and perform log rotation on a server. It includes functions to check the usage thresholds, log messages to a file, and perform necessary actions when thresholds are exceeded.

Here's a breakdown of the key components and functionalities of the script:

1. **Thresholds**:
   - Disk Threshold: The `DISK_THRESHOLD` variable is set to 30, indicating that disk usage exceeding 30% will trigger an alert.
   - CPU Threshold: The `CPU_THRESHOLD` variable is set to 30, indicating that CPU usage exceeding 30% will trigger an alert.
   - Memory Threshold: The `MEMORY_THRESHOLD` variable is set to 10, indicating that available memory below 10MB will trigger an alert.

2. **Log File**:
   The LOG_FILE variable specifies the path to the log file where messages will be logged.
   The LOG_ROTATE_FILE variable specifies the log file to be rotated when it reaches the maximum size.
   The MAX_LOG_SIZE variable is set to 10M, representing a maximum log file size of 10 megabytes.

3. **Functions**:
   log_message(): This function logs messages with timestamps to the specified log file.
   check_disk_usage(): This function checks the disk usage by getting the percentage of disk space used on the root partition (`/`). If the usage exceeds the disk threshold, a log message is recorded, and an alert can be sent.

check_cpu_usage(): This function checks the CPU usage by parsing the output of the `top` command. If the usage exceeds the CPU threshold, a log message is recorded, and an alert can be sent.

check_memory_usage(): This function checks the memory usage by getting the available memory and total memory using the `free` command. If the available memory is below the memory threshold, a log message is recorded, and an alert can be sent.

perform_log_rotation(): This function performs log rotation by checking the size of the log file specified in `LOG_ROTATE_FILE`. If the size exceeds the maximum log size, the log file is rotated using `logrotate`, and a log message is recorded.

4. **Main Execution**:

The script starts by logging a message indicating the start of the server monitoring script.

A continuous loop is initiated to periodically check the disk usage, CPU usage, memory usage, and perform log rotation.

Inside the loop, the functions check_disk_usage(), check_cpu_usage(), check_memory_usage(), and perform_log_rotation() are called.

The loop sleeps for 30 seconds before repeating the monitoring process.

QUESTION 2

Compilation

To compile and run the program, follow these steps:

Copy the provided code into a file called instruction_simulator.c.

Open a terminal and navigate to the directory where instruction_simulator.c is located.

Compile the code using the following command:

```
gcc instruction_simulator.c -o instruction_simulator
```

Run the program by executing:

```
./instruction_simulator
```

Outputs

The program will execute two separate runs, each with a different sample program. Here's an explanation of the output you will see:

The size of the program being executed is displayed to indicate the number of op codes and operands in the program.
The program counter (PC) indicates the index of the currently executing instruction in the memory.
The opcode and operand of the current instruction are displayed.
Before executing the instruction, the state of the program is printed, including the values of the program counter, accumulator (ACC), and memory operand.
The instruction is then executed using the executeInstruction function.
After executing the instruction, the updated state of the program is printed, showing the new values of the program counter, accumulator, and memory operand.
The program counter is incremented by 2 to move to the next instruction.
The execution continues until a HALT opcode is encountered, at which point the program terminates.

Op Codes

Op codes are represented as constants or an enum in the code. Here's a description of the available op codes:

ADD (0): Performs addition. Adds the value of the operand to the accumulator.
SUB (1): Performs subtraction. Subtracts the value of the operand from the accumulator.
LOAD (2): Loads a value from memory. Sets the accumulator to the value stored in the memory at the operand index.
STORE (3): Stores a value in memory. Sets the value in the memory at the operand index to the value in the accumulator.
JMP (4): Jumps to a different instruction. Sets the program counter to the operand value.
HALT (5): Halts the program execution.

Memory Layout

The memory is represented as an array in the code. The MEMORY_SIZE constant defines the size of the memory. Each element in the memory array stores an op code or an operand value.

Sample Programs

The program includes two sample programs (program1 and program2) to demonstrate the CPU execution. Here's a brief description of each program:

program1: Performs addition, subtraction, loading, and halting. It showcases the basic functionality of the CPU.
program2: Loads a value from memory, performs addition, and halts. It demonstrates the use of memory operations.

QUESTION 3

Compilation

To compile the program, follow these steps:

Clone the repository to your local machine.
Compile the code using the following command:

gcc textwriter.c -o textwriter

Make the spellchecker.sh script executable by running the following command:

chmod +x spellchecker.sh

Ensure that you have the necessary dependencies installed: gcc, bash, and a C standard library.

Usage

Run the program by executing:

./spellchecker

You will be prompted to enter the text you want to check for spelling errors. Type in your text and press Enter.

The program will utilize the spell-checking script (spellchecker.sh) to analyze your input. It will display any misspelled words found in your text.

The original input text and the list of misspelled words will be displayed in the console.

Spell Checker Bash Script

The spell-checking script spellchecker.sh is an integral part of the program. It uses an algorithm to compare the input text against a list of known misspelled words. The script is automatically executed by the C program to perform the spell-checking process.

Example

applescript

Enter text for spell checking: This is a sampel text with misspelled words.

Original input:
This is a sampel text with misspelled words.

Misspelled words:
sampel

Known Issues

The current implementation only displays the first misspelled word found in the input text, even if there are multiple misspelled words in the misspelled_words.txt file. Unfortunately, this issue couldn't be resolved at the moment.

The program assumes that the misspelled_words.txt file exists in the same directory as the executable. Make sure to place it there, or modify the code to provide the correct path to the file.