System Monitoring Tool

Overview

This project is a C program designed to report key system utilization metrics, including CPU usage, memory consumption, and the number of cores, based on user-defined parameters such as the number of samples, delay intervals, and flags passed during execution. The program provides real-time graphical representations of resource utilization, allowing users to visualize how these metrics change over time. Designed for Linux-based systems, the tool offers customization options through command-line arguments,enabling users to specify sample size, time delay, and the specific system metrics they wish to monitor.

Problem Solving Approach

To solve this problem, I took a modular approach by breaking it down into distinct tasks, including CPU usage calculation, memory utilization, and core information display. I implemented helper functions to handle system queries, such as retrieving total memory, fetching CPU statistics, and determining the number of cores, ensuring the main logic remained clean and focused. By researching the correct formulas for CPU and memory metrics, I ensured real-time accuracy in calculations. The display functions were designed to update dynamically using ANSI escape sequences, improving user experience by providing a continuously updating view of system performance. Additionally, I structured the argument parser to handle both positional and flagged inputs efficiently, including validation checks to prevent errors. This structured approach made the code efficient, readable, and adaptable for future enhancements.

Assumptions:

- The function assumes a standard terminal size (such as those used in lab machines) to ensure proper positioning of escape code-based output.
- The plotting logic depends on ANSI escape codes for cursor movement.
- Positional argument 0 for samples will print invalid samples. (Assumes samples > 0).
- N = 0 specified for--samples=N flag will print `Error: Invalid value for --samples` and program ends execution.
- Assumes that the first positional argument provided is the samples followed by tdelay.
- If positional arguments are specified then the flags of samples and tdelay shouldnot be specified.

Features:

- Monitors memory usage and displays a graph representation.
- Displays the current CPU load as a percentage.
- Shows the number of CPU cores and their clock speed.
- Customizable sample size and time delay for monitoring.
- Visualizes the data over a specified number of samples.

Table of Contents

Functions Overview

CPu Functions

`Function: get_cpu_usage`

Description:

Reads CPU data from `/proc/stat` and calculates total and idle CPU times.

Parameters:

- `long int* total_time` : Stores the total CPU time.

- `long int* idle_time`: Stores the idle CPU time.

Example:

```
 long int total_time, idle_time;
get_cpu_usage(&total_time, &idle_time);
```

`Function: parse_cpu_usage`

Description:

Parses CPU usage data to calculate total CPU time and idle CPU time from a string line passed on by get_cpu_usage.

Parameters:

- `line` (char*): CPU data as a string.
- `total_time` (long int*): Stores the total CPU time (sum of all times).
- `idle_time` (long int*): Stores the idle CPU time (value at index 3).

Returns:

No return value. Updates `total_time` and `idle_time` via pointers.

Example:

```
 long int total_time, idle_time;
parse_cpu_usage("cpu 123 456 789 1011", &total_time, &idle_time);
printf("Total: %ld, Idle: %ld\n", total_time, idle_time);
```

`Function: calculate_cpu_usage`

Description:

Calculates the total_cpu_usage by taking the total difference between two cpu times and idle times.

Parameters:

- `total_diff` (long int): *stores the total difference between two cpu times.
- `idle_diff` (long int): *stores the difference between the tow idle times of the cpu.

Returns:

Returns the CPU usage as a percentage using the formula:

```
cpu_usage = ((total_diff - idle_diff) / total_diff) * 100;
```

```
    - total cpu utilization time: Ti = useri + nicei + sysi + idlei + IOwaiti + irqi +
    softirqi
    - idle time: Ii = idlei
      where i represents a sampling time; hence the CPU utilization at time i is given
      by Ui = Ti - Ii.
    - Then, the CPU utilization will be given by (U2 - U1)/(T2 - T1) * 100
```

`Function: print_graph_structure_cpu`

Description:

Prints the y and x-axis structure for CPU based on the number of samples provided.

Parameters:

- `samples` (int): *The length of x-axis depends upon number of samples to be displayed, *calls the helper function* `print_horizontal_axis`.

Returns:

- No return value, Only prints.

`Function: update_graph_cpu`

Description:

Plots each sample of the calculated CPU usage in the terminal using escape codes (as specified in the assignment documentation). Determines the row and column position relative to the terminal size for accurate visualization.

Parameters:

- `cpu_usage` (float): Determines the height (row) for plotting CPU usage, where each ' `|` ' represents a 10% scale.
- `sample_index` (int): Represents the x-axis position, divided by the number of samples.
- `offset` (int): Measures the height from the top of the terminal to adjust the plotted position accordingly.

Scale: CPU Usage to Y-Position Mapping

CPU Usage (%)	Plotted Height (Bars)
`0 ≤ cpu_usage < 10`	`0`
`10 ≤ cpu_usage < 20`	`1`
`20 ≤ cpu_usage < 30`	`2`
`30 ≤ cpu_usage < 40`	`3`
`...`	`...`
`80 ≤ cpu_usage < 90`	`8`
`90 ≤ cpu_usage ≤ 100`	`9`

Memory Functions

Function: get_total_mem

Description:

Reads the total system memory from `/proc/meminfo` and returns it in gigabytes (GB).

Parameters:

- **None**

Returns:

- `(float)` : The total memory in **GB**.

Implementation Details:

- Opens `/proc/meminfo` in **read mode**.
- Reads the first value, which represents the total memory in kilobytes.
- Converts the value from **KB to GB** using the formula:

Function: get_free_mem

Description

Retrieves dynamic memory statistics, including **free memory**, by reading `/proc/meminfo` .

Parameters

- `long *mem_free` → Stores the free memory.

Implementation Details

- Opens `/proc/meminfo` for reading.
- Iterates through the file, searching for **"MemFree"**,
- Stores the extracted value in the corresponding variable.
- Closes the file after reading.

For Memory Utilization:

Memory Utilization (GB) = total_memory - mem_free` This formula calculates the actual memory usage of the system.

- `total_memory` : Total available RAM.
- `mem_free` : Unused/free memory.

**By subtracting free memory from total memory, we determine how much RAM is actively used **by applications and system processes. This approach is mostly used in system monitoring **to analyze resource utilization and optimize performance.

`Function: calculate_mem_info`

Description

Calculates the **memory usage** by subtracting the free memory from the **total memory**.

Parameters

- `float mem_total` → The total system memory in **GB**.
- `float *mem_usage` → Pointer to store the calculated memory usage in **GB**.

Implementation Details

- Calls `get_free_mem()` to retrieve **free memory**
- Computes memory usage as:

`Function: print_graph_structure_memory`

Description:

Prints the y and x-axis structure for Memory based on the number of samples provided.

Parameters:

- `samples` (int): *The length of x-axis depends upon number of samples to be displayed, *calls the helper function* `print_horizontal_axis`.

Returns:

- No return value, Only prints.

`Function: plot_memory_usage`

Description:

Plots memory usage as a vertical bar graph using **escape codes** in the terminal.
Each `#` represents a portion of total memory usage, with a **12-bar scale**.

Parameters:

- `float total_memory` → The total system memory in **GB**.
- `float mem_usage` → The used memory in **GB**.
- `int sample_index` → The position on the **x-axis** to plot the sample.

Implementation Details:

- The memory usage is mapped to a **12-bar scale**:
- Calculates the height using formula:
- ** mem_usage / total_memory` * 100 **

Cores Functions

`Function: get_cores`

Description

Retrieves the number of available CPU cores on the system using the `sysconf` function provided by the `<unistd.h>` header.

Parameters

- **None**

Returns

- `int` → The number of CPU cores available on the system.

Implementation Details

- Uses `sysconf(_SC_NPROCESSORS_ONLN)` to fetch the number of online processors.
- If `sysconf` fails (`returns -1`), the function prints an error message using `perror` and exits the program.

Example Usage

```
int cores = get_cores();
printf("Number of CPU cores: %d\n", cores);
```

`Function: get_max_freq`

Description:

Retrieves the maximum CPU frequency from `/sys/devices/system/cpu/cpu0/cpufreq/cpuinfo_max_freq` and returns it in GHz.

Returns:

- `(float)` : Maximum CPU frequency in GHz.

Implementation:

- Reads frequency in kHz and converts it to GHz.
- Exits on file read or close failure.

`Function: cores_diagram`

Description:

Generates a visual representation of CPU cores in a grid format,displaying cores as boxes. The function arranges cores in rows of up to four cores per row.

Parameters:

- `num_cores` (`int`): The total number of CPU cores to display.

Notes:

- Each core is represented as a small boxed unit (`+ -- +`).
- The function dynamically adjusts rows based on the number of cores.
- Cores are grouped in rows of four when possible, with proper spacing.

Additional Functions

Function: is_digit

Description:

Checks whether a given character is a numeric digit (0-9).

Parameters:

- `c (char)` : The character to check.

Returns:

- `(int)` : Returns 1 if the character is a digit ('0' to '9'), otherwise returns 0.

Implementation:

- Compares the character with '0' and '9' to determine if it falls within the numeric range.
- Returns 1 for numeric characters and 0 for all others.

Function: is_valid_integer

Description:

Checks whether a given string represents a valid positive integer.

Parameters:

- `str (char *)` : The string to check.

Returns:

- `(int)` : Returns 1 if the string is a valid positive integer, otherwise returns 0.

Implementation:

- Returns 0 immediately if the string represents a negative number (i.e., starts with '-').
- Uses `strtol` to attempt conversion to an integer.
- If there are extra non-numeric characters remaining after conversion, the function returns 0.
- Otherwise, it returns 1.

Description:

`Function: display_info`

Description:

Displays real-time CPU and memory usage graphs based on the provided flag settings.

Parameters:

- `f` (`flag_val *`): A pointer to a structure containing user-defined settings for graph display.

Implementation:

- Dynamically updates CPU and memory usage.
- Displays memory usage in GB.
- Plots real-time graphs for CPU and memory performance.
- Utilizes ANSI escape sequences to maintain a structured display.
- Includes a delay to control update frequency.

`Function: display_cores_info`

Description:

Displays information about the number of CPU cores and their maximum frequency, along with a visual core diagram.

Implementation:

- Retrieves the number of CPU cores using `get_cores()` .
- Fetches the maximum CPU frequency using `get_max_freq()` .
- Displays the core count and maximum frequency in GHz.
- Calls `cores_diagram()` to generate a visual representation of CPU cores.

`parse_arguments`

Description:

Parses command-line arguments to configure monitoring options for CPU, memory, and cores.

Parameters:

- `argc` (`int`): Number of arguments.
- `argv` (`char **`): Argument values.
- `f` (`flag_val *`): Structure to store parsed flags.

Implementation:

- Supports **positional arguments** (`samples`, `tdelay`).
- Supports **flags** (`--memory`, `--cpu`, `--cores`, `--samples=N`, `--tdelay=N`).
- **Error handling** for invalid values and duplicate flags.
- **Defaults to all monitoring features** if no flags are provided.

How to Run (use) your program

Prerequisites:

- **gcc (GNU Compiler Collection)**: Make sure `gcc` is installed on your Linux system.
- **Linux OS**: This program is intended to be compiled and run on a Linux-based system.

If compiled using the following command on linux system (specified with

-std=c99 or equivalent):

- *gcc -std=c99 -o tooba_cscb09_a1 tooba_cscb09_a1.c** Results in a Warning!

```
warning: implicit declaration of function ‘usleep’; did you mean ‘sleep’? [-Wimplicit-function-declaration]
  282 |         usleep(f->tdelay);
      |         ^~~~~~
      |         sleep
```

** Without specifying -std=c99 compiles without displaying any erorrs and warnings and continues with ** proper execution.

Example Usage:

- *./tooba_cscb09_a1 45 600000 --cores --memory --cpu

Example Output:

```
Nbr of samples: 45 -- every 600000 microSecs (0.600 secs)

v Memory  3.84 GB
 15 GB  |
        |
        |
        |
        |
        |
        |
        |
        |###############################################
        |
        |
        |
 0 GB   ----------------------------------------------

v CPU  0.00 %
  100%  |
        |
        |
        |
        |
        |
        |
        |
        |:::::::::::::::::::::::::::::::::::::::::::::::
   0%   ----------------------------------------------

v Number of Cores: 20 @ 4.80 GHz
+ -- +   + -- +   + -- +   + -- +
|    |   |    |   |    |   |    |
+ -- +   + -- +   + -- +   + -- +

+ -- +   + -- +   + -- +   + -- +
|    |   |    |   |    |   |    |
+ -- +   + -- +   + -- +   + -- +

+ -- +   + -- +   + -- +   + -- +
|    |   |    |   |    |   |    |
+ -- +   + -- +   + -- +   + -- +

+ -- +   + -- +   + -- +   + -- +
|    |   |    |   |    |   |    |
+ -- +   + -- +   + -- +   + -- +

+ -- +   + -- +   + -- +   + -- +
|    |   |    |   |    |   |    |
+ -- +   + -- +   + -- +   + -- +
```