CSCB09

Assignment 1: System Monitoring Tool

Zheyuan Wei

Date Updated: 2023-01-31

Table of Contents

Things to Note

- How did you solve the problem:
  - See Function Documentation for more details.
- An overview of the functions (including documentation):
  - See Function Documentation for more details.
- How to run (use) your program:
  - See Major Changes for more details. It is basically the same as the original assignment other than the changes listed below.

Major Changes

- Argument Handling
  - Now `--tdelay` can be set to any floating point number greater than 0.1, rather than only integers. The default value is 1.0. (For implementation details, see Function Documentation)
  - Now the program can handle arguments in any order.
    For example:
    `./xxx --system --user --graphics --sequential --samples=10 --tdelay=1`
    is equivalent to:
    `./xxx --sequential --graphics --user --system --tdelay=1 --samples=10 .`
  - Now the program can handle multiple arguments of the same type.
    For example:
    `./xxx --system --user --system --user --system --user`
    is equivalent to:
    `./xxx --system --user`
    Another example:
    `./xxx 10 1 --samples=5 --tdelay=2`
    is equivalent to:
    `./xxx --samples=5 --tdelay=2`
    and will finally be set to `samples:10 tdelay:1 .`
    > *There are mainly two reasons for this change:*
    >
    > - *This feature is more user-friendly along with the new **invalid argument handling** feature. It reduces the overall chance of error and makes the program more robust.*
    > - *This feature is more convenient for coding. (I used a `for` loop to handle the arguments, accepting the last (rightmost in CLA) argument of the same type and ignoring the rest.)*
  - Now the reduced argument keys are also supported. See full support list below:
    - `--system` is equivalent to `--sys`
    - `--user` is equivalent to `--u`
    - `--graphics` is equivalent to `--g`
    - `--sequential` is equivalent to `--seq`

    > *The main reason for this is that the program is more user-friendly and convenient to use, and also easier for me when debugging.*

- Invalid Argument Handling:
  - Improper argument usage is handled by the program by following appoaches:

        - If the argument is not recognized, the program will print out the usage, then ignore it and continue.
        - If the argument is recognized but the value cannot be recognized, the program will print out the usage message and set the value to the default value.
        - If the argument is recognized but the value is not valid (i.e., out of range), the program will print out the value range and set the value to the minimum (after thinking about it for a while, I decided to set no upper bound for the arguments. Only the lower bound is set).
            - If the argument is `--tdelay` and is less than 0.1, the value is set to 0.1.
            - If the argument is `--samples` and is less than 1, the value is set to 1.
- User Info Folding:
    - The lines of user info are limited to 5 (constant `user_count_max` in `userInfo()`). If there are more than 5 users, the program will fold the extra users into the last line like below:

```
### Session/Users ###
effendia        pts/1    (tmux(2335835).%0)
ponceca1        pts/2    (tmux(1883764).%5)
ponceca1        pts/3    (tmux(1883764).%0)
ponceca1        pts/4    (tmux(1883764).%1)
ponceca1        pts/5    (tmux(1883764).%2)
ponceca1        pts/6    (tmux(1883764).%3)
... Showing 5 of 90 users, use --all to show all users
```

> *The main reason for this is that the program was tested on the `mathlab` server. There are too many users on the server, and too much lines of user info will make the program hard to read whenever debugging or in actual use.*

> *The main reason for using `mathlab` is that the server does not require a UtorVPN connection, and the server is accessible from anywhere.*

    - Of course, the user can use the `--all` flag to show all users.
- Graphic Changes:
    - Minor Changes on CPU Info
        - Now the program can handle negative percentage increase, using `-` to indicate a decrease and `|` to indicate a increase.

> The main reason for this is that the assignment requirement is to show the positive percentage increase:
>
> - for CPU utilization:
>
> `|||| positive percentage increase`
>
> and yet in the demo video, the output shows exactly the CPU usage, rather than the

> percentage increase. So I decided to show the change rate of CPU usage rate, rather than following the self-contradictory requirement.

- Now the program will display the CPU usage in a more readable way. The program will display the CPU usage in a bar chart, within **one line**.

> *The main reason for this is that the program will encounter only `2` possibilities:*
>
> 1. With `--sequential` flag, the program will print out the CPU usage in a bar chart within one line, **each sample**. If you are writing the output to a file, you will see the CPU usage in each sample in a bar chart.
> 2. Without `--sequential` flag, the program will print out the CPU usage in a bar chart within one line, **each Tdelay**. If you are watching the output in real time, you will see the CPU usage in each time interval in a dynamic bar chart.
>
> *In either case, the bar chart is as readable as the original version, but the program is more efficient and the output is more compact.*
>
> *And... it's easier to code*

- Minor Changes on Memory Info
  - Addd a `|` before the number of change rate to make it more readable.
  - When the absolute value of change rate is less than `0.0100%` (which means the there will be `0` bars in the graphic), the program will print out a `@` instead of a `o` according to the real change rate.

> *This is to make the program more readable, and comforting to the user ("Okay the program is actually doing something, it's not broken" considering the change rate is likely to be small (less than `0.0100%`) on personal computers).*

```
### Memory Usage ### (Phys.Used/Tot -- Virtual Used/Tot)

6.93 GB /  7.77 GB ---  7.67 GB /  9.63 GB      |@| 0.0000%
6.93 GB /  7.77 GB ---  7.67 GB /  9.63 GB      |o| 0.0036%
6.93 GB /  7.77 GB ---  7.67 GB /  9.63 GB      |o| 0.0004%
6.93 GB /  7.77 GB ---  7.67 GB /  9.63 GB      |o| 0.0004%
6.93 GB /  7.77 GB ---  7.67 GB /  9.63 GB      |o| 0.0007%
6.93 GB /  7.77 GB ---  7.67 GB /  9.63 GB      |o| 0.0037%
6.93 GB /  7.77 GB ---  7.67 GB /  9.63 GB      |o| 0.0007%
6.93 GB /  7.77 GB ---  7.67 GB /  9.63 GB      |@| -0.0002%
6.93 GB /  7.77 GB ---  7.67 GB /  9.63 GB      |o| 0.0035%
6.93 GB /  7.77 GB ---  7.67 GB /  9.63 GB      |o| 0.0039%
```

Function Documentation

This documentation also includes the solution to the requirements of the assignment.

`main()`

What does it do:

- Handles the CLAs, and calls `showOutput()` with the appropriate arguments to print out the output according to the CLAs.
- If the CLAs are not valid, the program will print out the usage message and either exit or continue with the default values (see Major Changes for details).

How did I do it:

- CLA Handling
    1. Set up the default values for the CLAs.
    2. Set up the `char *` constants for the CLAs keys (i.e. `--system`, `--user`, etc.) and also the reduced keys (i.e. `-s`, `-u`, etc.).
    3. Special Case Judge: If `argc == 1`, no CLAs are provided, so the program will set all the CLAs to default and continue.
    4. Use a `for` loop to iterate through the CLAs.
    5. Match the CLAs with the keys/reduced keys:
        1. If there is a match, set the corresponding CLA to `true`.
        2. If there is no match, then it may be a positional argument (i.e. `10 1`).
            1. If it starts with a digit or `.`, then it is a positional argument.
                1. Used two boolean variables `samples_set` and `tdelay_set` to indicate whether the current argument is the first or second positional argument.
                2. Then use `atoi` and `atof` to convert the positional argument to `int` and `float` respectively.
            2. Else, it is an invalid CLA, so the program will print out the usage message and continue with the default values (see Major Changes for more details).

`showOutput()`

What does it do:

- It calls `CPUInfo()` and `MemoryInfo()` to get the information for the benefit of further processing.
- Call the functions below (in sequence) to print the information according to the CLAs.
    - `CPUInfo()`
    - `MemoryInfo()`
    - `userInfo()`
    - `sysInfo()`

- Usage: `showOutput(sys, user, graphic, seq, samples, tdelay, show_all);`

How did I do it:

- `seq` is a boolean variable indicating whether the output should be containing the ESCape codes (in the program, Linux ESCape codes are used to either refresh the screen or move the cursor). This function has two different part of code for `seq == true` and `seq == false`.
  - For `seq == true`, the program will print out the information sequentially without refreshing the screen. This is useful if the user wants to redirect the output to a file.
  - For `seq == false`, the program will print out the information using the ESCape codes to refresh the screen. This is useful if the user wants to watch the output in real time. `ESC[2J` is used to clear the screen, and `ESC[#B` is used to move the cursor down by `#` lines, when printing Memory Info.
- As mentioned in the Major Changes, the program is now able to handle `--tdelay` with support for floating point numbers. The program will use `usleep()` to wait for the specified time (in microseconds) before printing the sampling.

`CPUInfo()`

What does it do:

- Get the CPU info and usage and print it out.
- Usage:

```
// Get first CPU time
int last_CPU_time[3]={0,0,0};
int *last_CPU_time_ptr = last_CPU_time;
double last_CPU = 0;
CPUInfo(tdelay, last_CPU_time_ptr, graphic, &last_CPU);
```

How did I do it:

- Get the CPU usage from:
  - `sysconf(_SC_NPROCESSORS_ONLN)` in `unistd` library to get the number of processors.
  - the `/proc/stat` file to get the CPU usage.
- The calculation of CPU usage is :

```
(delta of( user + system )) / (delta of( user + system + idle )) * 100
```

> *Note that here the calculation has been simplified for simplicity.*

`MemoryInfo()`

What does it do:

- Get the Memory info and usage and print it out.
- Usage:

```
// Get first memory info
double last_Mem = 0;
MemoryInfo(graphic, tdelay, &last_Mem);
```

How did I do it:

- Use the `sysinfo()` function in `<sys/sysinfo.h>` to get the memory info, and use the `utsname` struct to store the info.
- The calculation of Memory usage is:

```
# Physical Memory Usage
(total - free) / total * 100

# Virtual Memory Usage
(total + swap - free) / (total + swap) * 100
```

`userInfo()`

What does it do:

- Get the User info and usage and print it out.
- Usage: `userInfo(show_all);` , where `show_all` is a boolean variable indicating whether the program should print out all the users or only the first `5` (depending on the value of `user_count_max` in `userInfo()`) users.

How did I do it:

- Use the `getutent()` function in `<utmpx.h>` to get the user info, and use the `utmp` struct to store the info.

`sysInfo()`

What does it do:

- Get the System info and usage and print it out.
- Usage: `sysInfo();`

How did I do it:

- Use the `utsname()` function in `<sys/utsname.h>` to get the system info.

Other functions

No other functions but the following macros are used in the program:

- `_MIN` and `_MAX` are used to get the minimum and maximum value of two numbers, as implemented in below:

```
#define _MIN(a, b) ((a) < (b) ? (a) : (b))
#define _MAX(a, b) ((a) > (b) ? (a) : (b))
```

Introduction

**This program is slightly different from the original assignment for both ease of programming*

The program should accept several command line arguments:

`--system`
to indicate that only the system usage should be generated

`--user`
to indicate that only the users usage should be generated

`--graphics` **(+2 bonus points)**
to include graphical output in the cases where a graphical outcome is possible as indicated below.

`--sequential`
to indicate that the information will be output sequentially without needing to "refresh" the screen (useful if you would like to redirect the output into a file)

`--samples=N`
if used the value N will indicate how many times the statistics are going to be collected and results will be average and reported based on the N number of repetitions.
If not value is indicated the default value will be 10.

`--tdelay=T`
to indicate how frequently to sample in seconds.
If not value is indicated the default value will be 1 sec.

The last two arguments can also be considered as positional arguments if not flag is indicated in the corresponding order: `samples tdelay`.

The reported "stats" should include:

- user usage
  - report how many users are connected in a given time
  - report how many sessions each user is connected to
- ystem usage
  - report how much utilization of the CPU is being done
  - report how much utilization of memory is being done (report used and free memory)
    - Total memory is the actual physical RAM memory of the computer.
    - Virtual memory accounts for the physical memory and swap space together -- swap is the amount of space (usually in disk or any other storage device) assigned by the OS to be used as memory in case of running out of physical space.
  - if the `--graphics` flag is used, generate a graphical representation showing the variation of memory used

Graphical representations
The following conventions were used while displaying the graphical outputs:

- or Memory utilization:

```
::::::@  total relative negative change
######*  total relative positive change

(OPTIONAL)
|o|    zero+
|@|    zero-
```

- for CPU utilization:

```
||||   positive percentage increase
----   negative percentage decrease
```