README

@mainpage mySystemStats

@author Huang Xinzi

> This program can help user to get basic system information, including system's memory usage, users' usage, CPU usage and OS information. You can display the information in the way you like, refreshing N times or print out sequentially, showing system or user usage only, or with graphics that can virtualize the system usage change. Also you are free to decide how many times the statistics will be displayed and interval between each time the information prints. Since the newest version implemented concurrency, the program is running more efficiently. In addition, when user hits Ctrl-C, the program will ask the user if it really wants to quit or not. And the program will ignore the Ctrl-Z signal completely.

How did I solve the problem?

1. Divide each functionality into modules:
    a. show program memory usage
    b. show system memory usage information (w/ graph)
    c. show CPU information (w/ graph)
    d. show users' information
    e. show operation system information
    f. set up signal handler when `ctrl-c` and `ctrl-z` are caught
2. Then design detailed function for each module:
    a. For example, when designing section (II), I first write a function `get_memory_info` to read memory usage information from Linux file "`/proc/meminfo`", and return the calculated memory use for comparing purpose. Then I write a helper function `show_memory_graph` to virtualize the memory usage difference between 2 iterations.
    b. Similar strategy is used when designing section (III) CPU usage.
3. Group the functions which have similar purpose into a single function.
    a. When displaying the system usage (system memory usage, core information, and CPU usage), instead of calling every function in main, I use a single function `show_sys_usage`.
    b. When setting up the signal handler, use one function for setting the signals caught in parent, and another function for setting in children.
4. Add helper functions.
    a. For example, when there is an error, we need a function to show the user error message. So I use a function `handle_error` to display error message and then terminate the program.
    b. I also have functions `move_up` and `move_down` to move the cursor up and down on the screen. This is useful when "refreshing" the screen. We can easily find the correct place for different information.
    c. Since every time we read the system usage information, we need to read the information concurrently. Then I use three helper functions `read_memory_info`, `read_cpu_info`, and `read_user_info` to fork a child and handle each child’s job. Then we can return to the sampling loop in parent so that parent can continue its work.
    d. In addition, I use a function `vertify_arg` to validate user's input argument.
5. Seperate the main driver program and the functions implementation.

An overview of the functions (including documentation)

1. Functions in `mySystemStats.c`

```
void move_up(int lines);
  /* Move the cursor up to find the correct place printing information.
     Takes a positive integer to indicate how many lines to move. */

void move_down(int lines);
  /* Move the cursor down to find the correct place printing information.
     Takes a positive to indicate how many lines to move. */

void ctrlc_handler(int sig);
/* A signal handling function to reset behaviour for SIGINT. When Ctrl-C
   is hitted, the program will ask the user whether it really wants to quit
   the program or not. If the user type 'y' or 'Y', terminate the program.
   Otherwise continue the execution. */

void ctrlz_handler(int sig);
/* A signal handling function to reset behaviour for SIGTSTP. When Ctrl-Z
   is hitted, the program will ignore the signal as the program should not
   be run in the background while running interactively. */

void set_signals_parent();
/* Reset the behaviours of the SIGINT and SIGTSTP signals in parent.
   We want to ignore the Ctrl-Z and ask the user whether it really wants
   to quit the program if it hits Ctrl-C. */

void set_signals_child();
/* Ignore the SIGINT and SIGTSTP signals in child, since we don't want
   the signals to interupt the job of child. */

void read_memory_info(int *fd, double prev_used, int graph);
  /* Fork a child to report memory utilization and write the information
     to the parent. After child has done its work, terminate the child.
     If in parent, return to the sampling loop.
     Takes an array of two integers that contains file descriptors of a
     pipe which connects the child and the parent.
     Takes the physical memory used from the previous iteration.
     Takes an integer flag to indicate if "--graphics" is been called. */

void read_cpu_info(int *fd, int tdelay);
  /* Fork a child to report CPU utilization and write the information
     to the parent. After child has done its work, terminate the child.
     If in parent, return to the sampling loop.
     When reading the cpu usage, sleep tdelay seconds between opening
     the Linux file "/proc/stat".
     Takes an array of two integers that contains file descriptors of a
     pipe which connects the child and the parent.
     Takes an integer tdelay to indicate the frequency of refreshing. */

void read_user_info(int *fd);
  /* Fork a child to report connected users and write the information
     to the parent. After child has done its work, terminate the child.
     If in parent, return to the sampling loop.
     Takes an array of two integers that contains file descriptors of a
     pipe which connects the child and the parent. */

void show_sys_usage(int sample, int tdelay, int graph_flag);
  /* Print system usage information and keep refreshing the information.
     If "--sequential" is called, display the information sequentially
     (i.e. w/o refreshing).
     If "--graphics" is called, virtualize the physical-use change.
     Takes an integer sample to indicate how many times it will print.
     Takes an integer tdelay to indicate the frequency of refreshing.
     Takes several integer flags to indicate the information desired. */

void vertify_arg(int argc, char *argv[], int *sample, int *tdelay,
          int *sys_flag, int *user_flag, int *sequential_flag,
          int *graph_flag, int *sample_flag, int *tdelay_flag);
  /* Validate the command line arguments user inputted.
     Use flags to indicate whether an argument is been called. */
```

2. Functions in `stats_functions.c`

```
void handle_error(char *message);
  /* Display error message and then terminate the program. */
```

```
void show_runtime_info();
  /* Show how much memory the program use during the compilation. */

void show_memory_graph(double curr_use, double previous_use);
  /* Using symbols representing the memory usage change.
     Takes two doubles representing current and previous memory use,
     and calculate the change based on the two inputs. */

void get_memory_info(long previous_use, int graph_flag);
  /* Display both physical and virtual memory usage and the total memory.
     If "--graphics" is called, virtualize the physical-use change. */

void calculate_cpu_use(int tdelay);
  /* Calculate CPU usage (in percentage) in real-time.
     Takes an positive integer to indicate how long will it refresh. */

void show_cpu_graph(double percent);
  /* Using "|" to represent the CPU usage change.
     Takes a double representing the current CPU usage. */

void show_cpu_info(double cpu_use);
  /* Prints the number of CPU cores and CPU usage percentage.
     Takes a double representing the current CPU usage. */

void show_session_user();
  /* Display user usage (username, terminal devices, IP address). */

void show_sys_info();
  /* Display basic system information (OS name, release information,
     architecture, OS version, etc.). */
```

How to run (use) my program?

1. Run with **make**:
   a. `make` or `make mySystemStats`: build the `mySystemStats` executable with warning flags.
   b. `make help`: display help message
   c. `make clean`: remove the `mySystemStats` executable and all object files
2. The program can take the following argument:

```
--system      Show the system usage only
--user        Show the users usage only
--graphics    Include a graphical output for system usage sections
--sequential  Output the system usage sequentially (without "refreshing")
--samples=N   Take a positive integer N and display the info N times
--tdelay=T    Take a positive integer T and display the info every T secs
```

3. Assumptions made:
   a. The display order is:

      Runtime Information, Memory Usage, Connected Users, CPU Usage, System information.

   b. The default value for "`--samples=N`" is 10, and the default value for "`--tdelay=T`" is 1.
   c. All arguments can be used together (even with themselves).
   d. Calling "`--samples=N`" or "`--tdelay=T`" multiple times with same input value will not result in error. But if the values are not consistent with each other, an error will occur.
   e. "`--samples=N`" and "`--tdelay=T`" can be considered as positional arguments (in order: samples tdelay) if they are not flagged. In this case no more than 2 integers can be taken as valid arguments.
   f. The program will intercept signals coming from `Ctrl-Z` and `Ctrl-C`. For the former, it will just ignore it as the program should not be run in the background while running interactively. For the latter, the program will ask the user whether it really wants to quit or not.