Developing utility monitor in C++ for linux system

Jurijs Zuravlovs, Dmytro Taras

05-02-2023

Contents

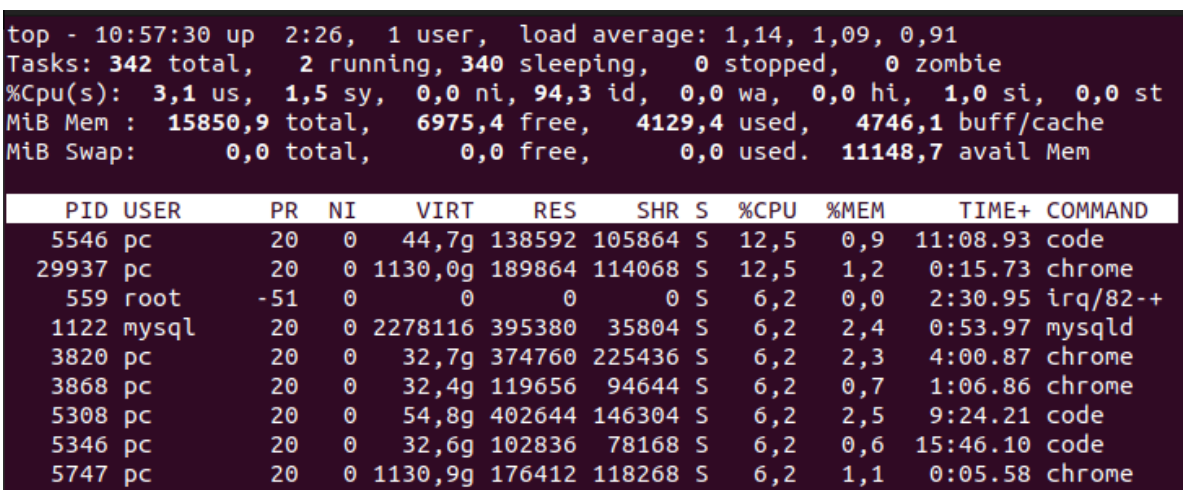
Introduction

The main goal of this project was to develop an linux utility program for monitoring the system. The key concepts is that the system should work from terminal and display the CPU utilization, RAM usage and so on.

Some common Linux utilities include “ls” for listing files, “grep” for searching through text, and “apt” for installing software packages. Overall, Linux utilities are an essential part of the Linux operating system and are widely used by system administrators and users.

Top command

Top is a command line utility for Linux and Unix-like operating systems that displays information about the processes running on the system. It is similar to the Unix command ps, but provides a more dynamic real-time view of a running system. By writing `top` in terminal it can display system summary information as well as a list of processes or threads currently being managed by the Linux kernel. It can also display a summary of the resources used by each process.



```
top - 10:57:30 up  2:26,  1 user,  load average: 1,14, 1,09, 0,91
Tasks: 342 total,   2 running, 340 sleeping,   0 stopped,   0 zombie
%Cpu(s):  3,1 us,  1,5 sy,  0,0 ni, 94,3 id,  0,0 wa,  0,0 hi,  1,0 si,  0,0 st
MiB Mem :  15850,9 total,   6975,4 free,   4129,4 used,   4746,1 buff/cache
MiB Swap:      0,0 total,      0,0 free,      0,0 used.  11148,7 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   5546 pc        20   0   44,7g 138592 105864 S  12,5   0,9  11:08.93 code
  29937 pc        20   0 1130,0g 189864 114068 S  12,5   1,2   0:15.73 chrome
    559 root     -51   0       0      0      0 S   6,2   0,0   2:30.95 irq/82-+
   1122 mysql     20   0 2278116 395380  35804 S   6,2   2,4   0:53.97 mysqld
   3820 pc        20   0   32,7g 374760 225436 S   6,2   2,3   4:00.87 chrome
   3868 pc        20   0   32,4g 119656  94644 S   6,2   0,7   1:06.86 chrome
   5308 pc        20   0   54,8g 402644 146304 S   6,2   2,5   9:24.21 code
   5346 pc        20   0   32,6g 102836  78168 S   6,2   0,6  15:46.10 code
   5747 pc        20   0 1130,9g 176412 118268 S   6,2   1,1   0:05.58 chrome
```

Figure 1: top

Top monitoring

Top monitor just works like top command. It displays the system summary information and list of processes currently being managed by the Linux kernel. It also displays a summary of the resources used by each process and runtime since the system was booted. Monitor it is nice way to overview the system and see what is going on.

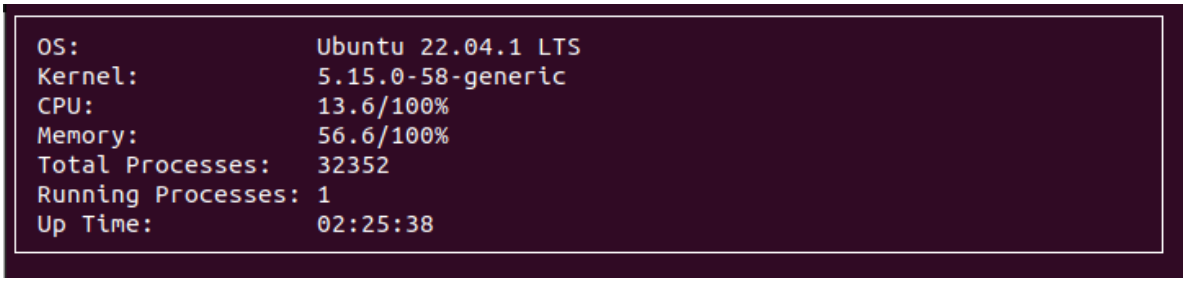


Figure 2: monitoring top

System information

System information for the process manager is derived from the following system files:

1. Kernel information - /proc/version
2. Operating system - /etc/os-release
3. Memory utilization - /proc/meminfo
4. Total processes - /proc/meminfo
5. Running processes - /proc/meminfo
6. Up time - /proc/uptime
7. CPU usage - /proc/stat

The following code snippet shows the paths to these files.

- "/proc/" is the root directory for the virtual file system in Linux that provides information about system processes, hardware, and configuration.
- "/stat" is a file under the "/proc" directory that provides information about the current status of the system, including CPU utilization and the number of processes.
- "/uptime" is a file under the "/proc" directory that gives the length of time that the system has been running.
- "/meminfo" is a file under the "/proc" directory that provides information about the memory usage of the system.
- "/version" is a file under the "/proc" directory that provides information about the Linux kernel version and build information.
- "/etc/os-release" is a file that provides information about the distribution and version of the operating system.

The implementation is shown below.

Implementation

These are constants in a C++ program that define file paths for accessing system information on a Linux operating system.

```
...
// Paths to system files
//virtual file sys
 const std::string procDirectory{"/proc/"};
 const std::string statFilename{"/stat"};
 const std::string uptimeFilename{"/uptime"};
 const std::string meminfoFilename{"/meminfo"};
 const std::string versionFilename{"/version"};
 const std::string oSPath{"/etc/os-release"};
 ...
```

In the context of the given file paths, a parser would read the contents of the files, such as "/proc/stat" or "/etc/os-release", and extract information such as system uptime, memory usage, and version information. This information can then be used by other parts of the program or further processed as needed.

Using the Ncurses library

Ncurses is a library for creating text-based user interfaces (TUI) in a terminal emulator. It allows you to create windows, move the cursor, and control text and color attributes.

To start working with ncurses, you need to do the following:

1. Install ncurses on your system, if it is not already installed.
2. Include the "ncurses.h" header in your C or C++ program.
3. Initialize the ncurses library by calling "initscr()".
4. Create windows and draw text using the ncurses functions.
5. Refresh the screen to display the changes by calling "refresh()".
6. Clean up the ncurses environment by calling "endwin()" when you are done.

```
#include <ncurses.h>

int main() {
  initscr(); // Initialize ncurses
  printw("Hello, World!"); // Print text to the screen
  refresh(); // Refresh the screen to display the changes
  getch(); // Wait for user input
  endwin(); // Clean up ncurses
   return 0;
}
```

ncurses is a powerful library for creating TUIs in a terminal emulator and it offers many features for advanced terminal-based user interfaces, including:

- Keyboard input: ncurses provides functions for reading input from the keyboard, such as "getch()".
- Color support: ncurses supports the use of color in your TUI, including defining color pairs and setting the foreground and background colors for text.
- Window management: ncurses provides functions for creating, moving, resizing, and updating windows. It also supports scrolling within windows.
- Mouse support: ncurses provides functions for reading mouse events and using the mouse in your TUI.
- Attributes: ncurses provides functions for controlling the appearance of text on the screen, including setting bold, underline, and reverse video.
- Terminal detection: ncurses automatically detects the terminal type and adjusts its behavior to match the capabilities of the terminal.
- Portability: ncurses is highly portable and can be used on a wide range of platforms, including Linux, Unix, and Windows.

It's important to note that ncurses only provides TUI functionality and does not support graphical user interfaces (GUIs). When working with ncurses, it's also important to understand the limitations of the terminal emulator and terminal capabilities.

NCurses implementation

The code is a C++ implementation of a system monitor display using the NCurses library. The NCursesDisplay class has two methods, "ProgressBar" and "DisplaySystem".

The "ProgressBar" method takes a float value representing the percent of utilization and returns a formatted string with the utilization and "/100%".

The "DisplaySystem" method takes a reference to a System object and a pointer to an NCurses WINDOW object and displays the information about the system on the window. The method uses the NCurses library functions "mvwaddstr" and "waddstr" to write strings to the window and "wrefresh" to refresh the window.

```
...
void NCursesDisplay::DisplaySystem(System& system, WINDOW* window) {
  int row{0};
  ...
  // display system information of the system
  mvwaddstr(window, ++row, 2,("Total Processes:   " +
  std::to_string(system.TotalProcesses())).c_str());
  mvwaddstr(window, ++row, 2,("Running Processes: " +
  std::to_string(system.RunningProcesses())).c_str());
  mvwaddstr(window, ++row, 2,("Up Time:           " +
  Format::ElapsedTime(system.UpTime())).c_str());
  // refresh the window
  wrefresh(window);
}
...
```

The "Display" method initializes NCurses, creates a window, and displays the information about the system in an infinite loop, updating the information every second. The method uses the NCurses library functions "initscr", "noecho", "cbreak", "newwin", "box", and "endwin" to initialize, create, and terminate the NCurses environment.

```
...
void NCursesDisplay::Display(System& system) {
  // initizalize ncurses -screen, no echo, no delay, no cursor
  initscr();      // start ncurses
  noecho();       // do not print input values
  cbreak();       // terminate ncurses on ctrl + c

  int x_max{getmaxx(stdscr)};// get max x value of screen
  WINDOW* system_window = newwin(9, x_max - 1, 0, 0);

  while (1) {
    box(system_window, 0, 0);
    DisplaySystem(system, system_window);
    wrefresh(system_window);
    // refresh screen to match memory
    refresh();
    std::this_thread::sleep_for(std::chrono::seconds(1));
  }
  endwin();
}
...
```

Parses implementation

```
...
// Reads and returns the system uptime
long LinuxParser::UpTime() {
  long uptime = 0.0;
  std::string temp = "0.0";
  std::string line;
  std::ifstream stream(procDirectory + uptimeFilename);
  if (stream.is_open()) {
    std::getline(stream, line);
    std::istringstream linestream(line);
    linestream >> temp;
  }
  uptime = std::atoi(temp.c_str());
  return uptime;
```

```
}
...
```

This function is called `LinuxParser::UpTime()` it retrieves the system uptime in seconds. It does the following:

- Defines a variable `uptime` and sets it to 0.0.
- Opens the file `/proc/uptime` using an input file stream `stream`.
- If the file is open, it reads the first `line` of the file and stores it in the `line` variable.
- It converts the string `line` into a stream of data using `istringstream`.
- Reads the first item in the stream, which is the uptime value, and stores it in the `temp` variable.
- Converts the string stored in temp into an integer and stores it in the `uptime` variable.
- Returns the `uptime` value.

Output

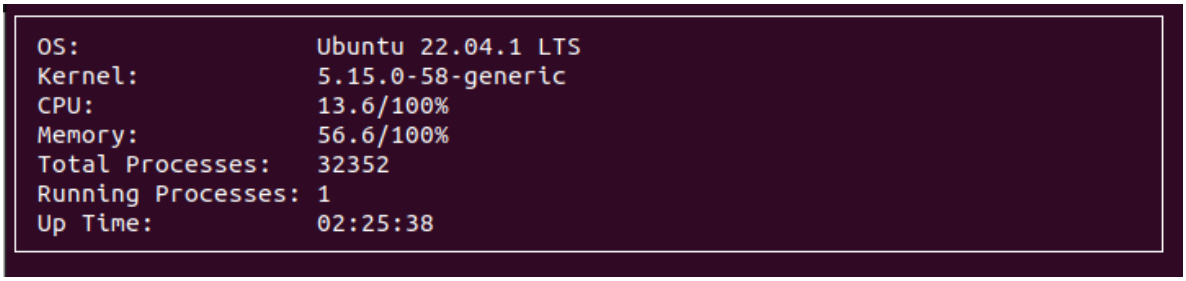


Figure 3: Output of the program

Overall, it's a pleasant experience with Ncurses, a simple way to implement an interface for any terminal instance. However, there are plenty more options to implement a more fancy interface and add more information, colors, and features.

References:

- How to start with Ncurses.
- How to make Ncurses text.
- Nice youtube tutorial.
- top(1) - Linux manual page.
- Github repo.