# Implementing a top-interface for the SenseHat LED on a Raspberry Pi 3B

Group 24

Uppsala University

November 1, 2019

**Abstract**

*Monitoring computer systems is an important task for admins and users alike. A common tool to display cpu information on UNIX-systems is top. This work will present an example implementation of a similar interface for the SenseHat LED system, which is installed onto the Raspberry Pi 3. We will compare this setup with a web-based solution.*

## I. Introduction

First we will explain how an UNIX-system (here: Debian) can provide the necessary interfaces to access the relevant information required. Then we will talk about the way top displays this information and how we can utilize the given $8 \times 8$ grid of the LED. Finally we will compare this implementation with a (theoretical) web-based solution and discuss further improvements.

## II. Kernel Interface

Representing (kernel) data is not a new idea. The *proc* filesystem mentioned by Killian [Killian, 1984] was used to map address space to files so that processes could access their own image. This was also used to improve debugging techniques as this was a major problem before. Today's systems expand this idea and also introduce other subfilesystems [proc(5)]. We will focus on the *stat*-filesystem, which displays kernel/system statistics.

As everything under UNIX is a file, so is the *proc*-filesystem. To access the data we can use any available fileoperation. When the file */proc/stat* is read the kernel updates the file. This means whenever we want to read the new state, we will have to reopen the file.

## III. The top interface

Top presents the user with both an overview of the system's resources and a list of running processes with their associated values for cputime, memory usage et cetera. This is done dynamically so changes made to the system can be observed in real time. While this is helpful, the manpage describes it as a limited interactive interface for process manipulation.

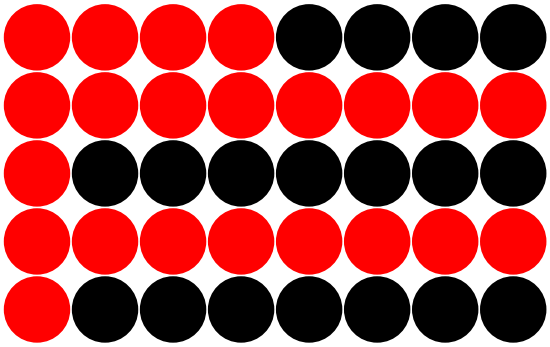
We also want to mention that there are many more implementations, **htop** to name one, which provide similar functionality with a slightly different user interface.

## IV. Output

### i. SenseHat LED

Our hardware consist of a Raspberry Pi 3B with a connected SenseHat LED $8 \times 8$ grid and a joystick. We decided to simplify the output to a general overview of the systems cpu state. We have 4 cores and can read all relevant information for each cpu and their sum from */proc/stat*. We decided to utilize one row of the LED for each batch of data, with the first representing the sum of all cpus, the second row the first

**Figure 1:** *example of total cpu usage of 50 %*



cpu and so on. Every light represents 12.5 % of cpu usage. An example can be seen in Figure 1. Here our total cpu usage (first row) is around 50% and 2 cores are at full capacity while the other cores are idle. Note that the first light on the idle cpu is still lid. The decision was made when we first tested our program, as no changes could be seen as there was no (noteful) load on any cpu. For the same reason we provide a template, which can be run in parallel to the top-program, to simulate an artificial work load.

### ii. Web-server

Now assuming we install a web server on the system which serves a page showing the current usage. For simplicity we will assume a static page, generated by some php code. While this setup is simple and easy, it is not real-time, as each static page represents a snapshot of the system. On the other hand, further information can be shown and the user interface is clean and common[1].

## V. Results

Our LED solution does work. It supplies the required information in an understandable and, maybe more important, common way. The web based solution does the same, while offering more possibilites in terms of space and design than the limited grid of the LED.

[1] by now one should assume that people know how the web works

Even though it is critical for such an interface to be in real-time, the web based approach can be used to display the history as graphs for better readability. One could also use a different approach and create a real-time display, altough this requires extensive knowledge in web development.

## VI. Discussion

The results of the project should be taken with a grain of salt. The final decision which approach is preferred should take into account when and where users may have to access this data. Is it even necessary to add another interface when one could just connect via ssh and run well programmed solutions like **htop**?

Our prototype shows that while being a simple solution, it also omits a lot of information which might be necessary for the users, e.g. the name of the process that has the highest cpu usage. Nevertheless, it could help to signal that something might be wrong with the system in such a figurative way, so investigation of a possible problem with suitable tools can begin earlier.

Next to this functional view there is obviously also the fun one can have by reading about and implementing these programs. It is also noteworthy that understanding basic concepts of the linux kernel (which the *proc*-filesystem is a part of) can help understanding our operating systems that we use daily. Even if our prototype or any work that resolves out of it are never used in a production system, the knowledge gathered are sure to be of use.

## References

[Killian, 1984] T. J. Killian (1984). Processes as Files *USENIX Association*, Summer Conference.

[proc(5)] Linux man pages