MyGKrellM

C++ Pool - Rush 3

Koalab koala@epitech.eu

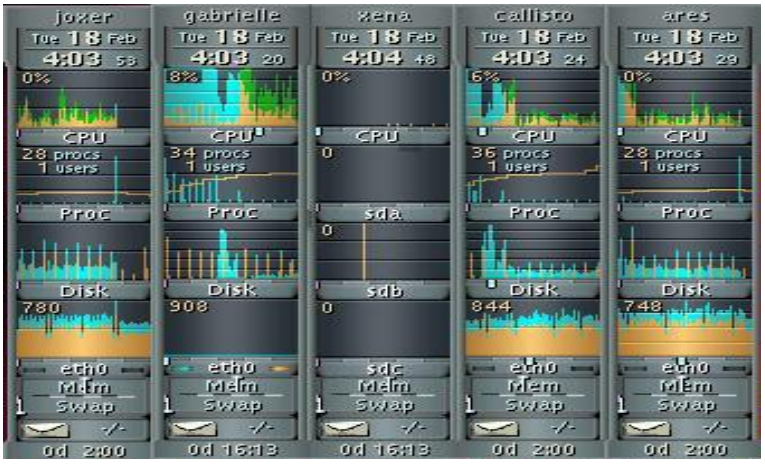
Abstract: *The purpose of this rush is to create a system monitoring tool based on a modular architecture.*

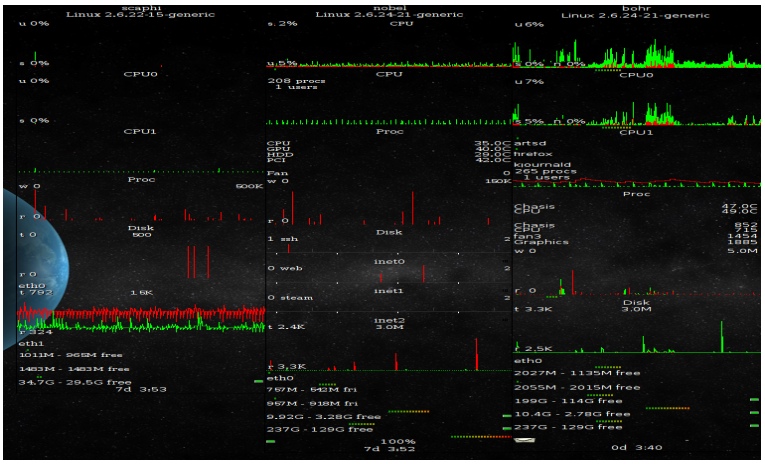
Contents

Chapter I

Explanation in pictures

A good illustration is always better than a bad explication, here is an idea of what we expect for this rush.





And a little link : http://en.wikipedia.org/wiki/GKrellM

Chapter II

Subject

II.1 Foreword

Let's be clear: this rush does not consist in a recode of `GKrellM`, but to do your own clone of it. This subject is the least restrictive of the three rushes, so enjoy yourself and feel free to add your own stuff. However, **dat ain't no party** and a certain number of mandatory steps have to be validated first.

There are three steps in this rush, plus a bonus step. Each step must be done entirely and perfectly before beginning the next one because the difficulty level is increasing. Grading will be incrementally too. Do not start the next step while the previous one is not perfectly finished and functional.

Your monitor can be seen as a module container. It is possible to activate and deactivate the available modules to adapt the information to our needs and even reorder them as we want. Useful.

`GKrellM` doesn't need privileged rights to run correctly. Neither does your monitor.

II.2 Display

Once properly configured and with a sexy skin, `GKrellM` is a very nice system monitor. But to reach this point, there's work to be done!

Your system monitor should be able to be launched in "text" mode or in "graphic" mode and keep the same functionalities. You must therefore think from the very first lines of your code to this constraint.

- In "text" mode, your monitor will display into your terminal. For this display you **MUST** use the `ncurse` library.
- In "graphic" mode, your monitor must display into a graphical window. You are completely free to choose the library of your choice for the graphical mode. Your only limit is your madness.

Whatever is the mode of the monitor, the visual quality and the ergonomy of the display will have a non negligible impact on your grade. For example, some data are more interesting when represented by a numeric value or by words, others by an histogram or a curve. Be imaginative. Any tasteless attempt "for fun" will be subject to an arbitrary malus. No question asked, **life's unfair**.

`GKrellM` has an awesome skin system. What about yours?

II.3 Design

The design of your monitor is entirely at your discretion. However, you must use at least the two following interfaces:

- `IMonitorModule` : This interface allows to describe the behavior of a module of your monitor.
- `IMonitorDisplay` : This interface allows to describe a display mode of your monitor.

The detail of these interfaces is your problem, but ask yourself why we took the time to impose their existence.

The minimal required work impose to be able to configure the modules used at the compilation (level -1). Be able to configure the modules used at the launch is quite better (level 0). Be able to add or remove modules while running would roxx (level 1).

II.4 Step 1

- Monitor core
- Module machine name and user name
- Module operating system and kernel version
- Module date and time

II.5 Step 2

- Module CPU (model, frequency, number of cores, activity, ...)
- Module RAM

II.6 Step 3

- Module of network load

II.7 Step 4 - Bonus

A rush during a pool is a bag filled with points for those playing **the game** to its end... Fourth and last step of this rush: create as many **USEFUL** additional modules as possible. The definition of "useful module" is at the arbitrary appreciation of the Koala at defense-time. No question asked, **life's unfair**. Be creative and intelligent. At the same time.

Special Bonus *: A module representing the favorite animal of the Koala Dan is automatically considered as an useful module. However, the representation of this animal must be animated to earn the points. If you don't know the favorite animal of Dan, manage by yourself to find it.*

Do not forget that the mandatory part must be perfectly terminated and functional to start counting the bonus points.

Chapter III

Instructions

You must turn in your work using the repository provided by Epitech.

Your repository will be read and gathered sunday morning at 10:00am. Only the files in your repository will be corrected during the defense. No exception will be made. No question asked, **life's unfair**.

If you have any question, we invite you to send an email to the authors of this subject.

- Use the `STL` as much as possible! And not only containers!
- The only and unique authorized functions of the `libc` are those which encapsulate the system calls.
- Any value passed by copy instead of by reference or by pointer must be justified, otherwise you'll lose points.
- Any value passed by pointer instead of by reference must be justified, otherwise you'll lose points.
- Any none `const` value passed as parameter must be justified, otherwise you'll lose points.
- Any member function or method which doesn't modify the current instance not being `const` must be justified, otherwise you'll lose points.
- There is no norm in `C++`. However, any code that we'll judge unreadable or too dirty may be punished. Be serious! This appreciation is at the arbitrary decision of the corrector. No question asked, **life's unfair**.
- Keep an eye on this subject regularly because it is likely to change until 4 hours before the turn-in time.

Good luck, and see you on sunday!

And for those who read until the end, some help, it has never hurt anyone:

- http://moun3im.free.fr/docmaster/articles/linux070.html
- http://tldp.org/LDP/Linux-Filesystem-Hierarchy/html/proc.html