RCLOCK
The documentation
A terminal digital clock for POSIX

Author
Rômulo Peres de Moraes

Index

Chapter 1. CASE Tools

Good software is never built without help, and the Rclock isn't an exception. Here is a list of softwares that will help the development of this project:

1.1. Visual Studio Code

The Visual Studio Code is a extensible cross-platform code editor that will help the process of coding and debugging the software. Any other code editor is acceptable for work with the Rclock project, however, the code editor has to follow configurations that allow a better interoperability among the editors. The configuration is the following:

- The selected code editor (or even the VS code) must have the Tab Size defined to 4 (spaces)

1.2. GCC compiler

The Gnu Compiler Collection (GCC) is the tool that will be used to generate the final executable, the version of the compiler that will be used on the project is the **13.2.1**

1.3. Meson

Meson is a build system for a wide set of languages, currently supporting C, C++, D, Fortran, Java, Rust. This build system will be used instead of the Make because of its friendly way of use

Chapter 2. Software definition

Here are all the definitions of the software that will be built, this section consists of a set of questions and its answers.

2.1. What is the data that will be manipulated be the software

The main set of data that the Rclock will work on is date and time. Even that the OS is able to give us the current date and time, the software will also accept a date and a time from the user.

2.2. What is the features and the performance required by the software?

2.2.1. Features

This software has a initial set of required features, divided into three groups. The **User** group, that has features which enable the user to modify the program. The **Clock** group, that is a set of features for the clock itself. The **Project** group, that is a set of technical features for the project.

2.2.1.1. User group

Here is the set of required features for the user group

1. The system shall offer a way for the user to pass configurations flags through the command-line arguments
2. The system shall make the users choose the zoom of their preference

2.2.1.2. Clock group

Every single clock feature, for example, clock customization and time features are listed here

1. The system shall show the current date beyond the clock itself
2. The system shall give the possibility of hide the current date
3. The system shall have the feature of change the clock color
4. The system shall have the feature of change the date color
5. The system shall have the feature of change the color of each clock digit
6. The system shall have the feature of change the color of the colons that split the digits pairs
7. The system shall offer the customization of the date format using a string
8. The system shall offer a way of set a custom time for the clock, for hours, minutes, and seconds
9. The system shall use the current time for those segments (hours, minutes and seconds) that weren't set by a custom time flag
10. The system shall offer a way to set a custom time using the format: xx:xx:xx
11. The system shall offer a way of set a custom date for day, month and year
12. The system shall use the current date for those segments (day, month and year) that weren't set by a custom time flag

2.2.1.3. Project group

1. The system shall implement each clock digit and colon on a separate Ncurses window
2. The system project shall have a well structured file system

2.2.2. Performance

The performance of an application is a crucial point while planning and building the application, however, as the Rclock works only with Hours, minutes and seconds, the performance must be 1 frame per second. For each second lapsed, and if the clock is also

showing the seconds, the clock will be updated, however, if the clock isn't showing the seconds, the frame rate will keep the same, but with the utility of update the clock's position whenever the terminal be resized

2.3. Required interfaces

Create interfaces in a console is not usual, by default, we would have to handle a couple of events that the window could generate. Using the Ncurses library, the situation gets better when we talk about design an interface and update only specific pieces inside the console buffer.

2.3.1. Clock digits

The clock digits, as the main part of the program, will be designed using different background colors. With the great support of colors that the Ncurses offers to the developers, and of course, the support of colors that the terminal that is hosting the application should have, the design and customization will be great. The clock's look and feel suggests something really digital, the edges of the numbers are squared, but still a beautiful design.

2.3.2. Clock date

As the date is usually shown as a string, on this project it won't be different, the Rclock date will be placed at the bottom of the clock digits, its presence won't change the clock position, that must be always at the center of the terminal

2.4. Required validation criteria

The validation criteria will be useful for detect any unexpected behavior that the Rclock program may generate, and its validation criteria are based on user input, that changes the normal behavior of the software somehow. Here is all possible user inputs and what they do:

2.4.1 Hiding the date

This flag hide the clock date completely. The clock itself remains unchanged.

Full command: --hide-date
Abbreviation: -h

2.4.2. Changing clock color

This flag changes the color of the entire clock. This flag doesn't affect the date color

Full command: --clock-color <color-name>
Abbreviation -c <color-name>

2.4.3. Changing date color

This flag changes the color of the date displayed bellow the clock

Full command: --date-color <color-name>
Abbreviation: -d <color-name>

2.4.4. Changing the color of specific clock digit

This flag changes the color of a specific clock digit. To specify the correct digit to be customized, a roman number, between I and VI must be part of the flag

Command: --color-VI <color-name>

2.4.5. Changing the color of the colons

This flag changes the color of the colons that divide the clock segments

Full command: --colon-color <color-name>
Abbreviation: -o <color-name>

2.4.6. Changing the date format

The flag of change date format needs a strftime function format string

Full command: --date-format <string-in-strftime-function-format>
Abbreviation: -f <string-in-strftime-function-format>

2.4.7. Set a custom time for the clock

The following commands set a custom time to the clock, they change the hours, minutes, seconds and the full time respectively

Full command: --custom-hour <0-23>
Abbreviation: -H <0-23>

Full command: --custom-minute <0-59>
Abbreviation: -M <0-59>

Full command: --custom-second <0-59>
Abbreviation: -S <0-59>

Full command: --custom-time <xx:xx:xx format>
Abbreviation: -T <xx:xx:xx format>

2.4.8. Set a custom date

The following commands set a custom date to the program, they change the day, month, year and a full date respectively

Full command: --custom-day <integer>
Abbreviation: -D <month-day>

Full command: --custom-month <integer>
Abbreviation: -O <integer>

Full command: --custom-year <integer>
Abbreviation: -Y <integer>

Full command: --custom-date <DD/MM/YYYY>
Abbreviation: -D <DD/MM/YYYY>

2.4.9. Hide the seconds of the clock

This command hide the seconds of the clock, the clock itself must still aligned to the center

Full command: --hide-seconds
Abbreviation: -i

Chapter 3. Software construction

Here is all the planning to build the software, its logic, data structures and how will be translated to a programming language in the future

3.1 Data structures

Here is the collection of all data structures that will be necessary to handle crucial data to the application

3.1.1. Digits matrix

Design the clock digits in the buffer may be a challenging part of the logic, even because there's a set of possible numbers between 0 and 9 and each clock digit may be any of them. The digits matrix will be used as an iterable map, that will make it possible to draw the numbers on the terminal.

3.1.1.1. The format of the matrix

The matrix will have three dimensions, the first dimension will be use to define each digit, the index 0 will hold the digit 0, the index 1 will hold the digit 1 and so on… The two remaining dimensions will be used for hold the format of each digit.

3.1.1.2. Matrix building constants

The matrix will use two macros for better understanding of what's part of a number and what's just a empty place in the final buffer. The two macros are de following:

- COLOR - Is a 'pixel' of a number, defined by the number 1
- INVIS – Is a empty place, defined by the number 0

3.1.2. Windows array

As said previously in the project's requirements, each clock digit will be hold by a different Ncurses window, to make the process more easier, all windows will be hold by a array, making the process of redraw iterable. The colon windows will be stored in the same array, even because they will also be modified during the program run.

3.1.3. Colors array

The built-in set of colors that the Rclock supports will be placed inside a array of colors and fetched its existence and its code code when necessary. It is a fact that an array isn't a smart approach usually, by the reason of its O(n) nature, however, the built-in set of colors isn't large enough to make a significant difference while running the program.

3.2. Software architecture

This section of the documentation will explain how the project is structured, and show how the modules of the software will work together to achieve the final software described previoulsy

3.2.1 The project filesystem

The tree of the project is divided into directories, each one holding files of the category described by the directory name

- **bin/ -** stores the final binary after compiling
- **build/ -** directory used by the Meson tool for build the software
- **docs/ -** directory used for hold the documentations in general, for development and for the final user
- **include/ -** stores all the header files of the program
- **libs/ -** directory used for store the third-party libraries
- **src/ -** directory used for hold all the source code files
- **tests/ -** all tests are done here

3.2.2. Modules

For a better understanding and organization of this project, the source code will be split by modules, each one executing a task that together makes the program.

3.2.2.1. shapes.c

The shapes.c is a module that stores the clock digits and the colon, everytime that a module of the program needs a digit or a colon shape, the module will need ask for it.

3.2.2.2. datetime.c

The datetime.c is a module that will handle everything that involves dates and time, format, generate and read dates are examples of what this module does.

3.2.2.3. screen-manager.c

The screen-manager.c is a module that will take care of build and prepare the windows and place placeholders to be filled with the contents, that are the digits, colons and the date.

3.2.2.4. designer.c

The designer.c is a module that will draw the contents on the places given by the screen-manager.c module. The digits, colons and the date are printed here.

3.2.2.5. colors.c

The colors.c is a module that will handle everything that involves colors, load built-in colors and get colors for each element on screen is done here.

3.2.2.6. arguments.c

The arguments.c is a module that will handle the user input through the command-line arguments, its job is to parse them and generate a struct that offers in different fields what each part of the software will use.

3.2.3. How the procedures should be implemented

Here is a briefing of the most important procedures that each module will have for the correct operations of the Rclock program

3.2.3.1. datetime module

The datetime module has a couple of procedures that will be useful to parse and generate dates/time, a great part of the requirements listed previously are done here. The procedures of this module are the following:

3.2.3.1.1. generateDateAndTime()

Once called, this function return a struct filled with the current date and time, a great part of the remaining program will work on this data, updating its content for customization or even for the normal Rclock operation.

3.2.3.1.2. setNewTime()

Once called, this procedure will update the time generated by the generateDateAndTime() function using the value provided by the user. Those features of define a custom time for the clock are done here.

3.2.3.1.3. setNewDate( )

Once called, this procedure will update the date generated by the generateDateAndTime( ) function using the value provided by the user. Those features of define a custom date for the application are done here.

3.2.3.1.4. verifyForDateAndTimeErrors( )

Once called, this procedure will check for possible range errors, for date and time. Errors like set the hours to 26 or month to 15 are captured here.

3.2.3.1.5. parseDate( )

Once called, this function will parse the date using the format DD/MM/YYYY and return its contents and a possible parse error flag

3.2.3.1.6. parseTime( )

Once called, this function will parse the time using the format xx:xx:xx and return its contents and a possible parse error flag

3.2.3.1.7. generateDateString( )

Once called, this function will return the date that will be shown below the clock, if a custom format be provided by the user, this function will use it instead of the default format.

3.2.3.1.8. incrementClockSecond( )

Once called, this procedure will increment the date/time structure in one second and normalize the result to be possible increment minutes, hours, and days automatically. The normalize only must be done when the seconds reached its range, changing the minutes, hours and so on...

3.2.3.2. screen-manager module

The screen-manager module will be responsible for generate windows and define placeholders that will be filled with the Rclock contents. These are the main procedures that will be used

3.2.3.2.1. generateWindows( )

Once called, this procedure will generate and populate an array with Ncurses windows, that will hold the digits and colons in the future, the date window will be stored in an individual variable

3.2.3.2.2. setPlaceholders( )

Once called, this procedure will generate the placeholders for each digit, colon and date windows. The generated placeholders will be stored inside the module, since the screen-manager is the only component that can change these positions, there’s no sense on store this data outside the module. The windows positions that hold the digits and colons will be

stored using an array, the date window position will be stored in a struct. There's the possibility of the user want to hide the seconds of the clock, for that reason, this procedure must be notified to ignore the last three windows, that would be used for the last colon and the seconds.

3.2.3.2.3. detectWindowResizes( )

Once called, this function will check for window resizes. Every window resize must be noticed for rearrange all the components placed on screen

3.2.3.2.4. moveWindowsToPlaceholders( )

Once called, this procedure will move all components to their placeholders defined by the setPlaceholders( ).

3.2.3.3. colors module

The colors module will load all available colors and set the selected colors for each component of the program based on the user input, every acess to colors to draw something on screen must be requested to this module

3.2.3.3.1 loadBuiltinColors( )

Once called, this procedure will fill an array inside the module with all available colors and colors ID.

3.2.3.3.2. setComponentsColors( )

Once called, this procedure will save the colors given by the user through command-line arguments after check if the same exists. If no values were given, a default color will be used

3.2.3.3.3. getComponentColor( )

Once called, this function will verify the given argument that identifies a component and return the color ID that matches the identifier

3.2.3.4. shapes module

The shapes module will be a proxy between the digits/colon shapes and the rest of the program, any acess to these shapes must be asked to this module

3.2.3.4.1. getClockDigit( )

Once called, this function will return a pointer to the clock shape specified by the given argument

3.2.3.4.2. getClockColon( )

Once called, this function will return a pointer to the colon shape

3.2.3.5. designer module

The designer module goal is draw the Rclock components on their windows. Select windows, select numbers and use colors is done here

3.2.3.5.1. normalizeSegment( )

Once called, this function will fill an array of two positions with the two numbers of a clock segment (hours, minutes or seconds). This function is useful because numbers lower than 10 are represented only by one digit.

3.2.3.5.2. drawSegment( )

Once called, this procedure will draw both digits of the specified clock segment (hours, minutes or seconds)

3.2.3.5.3. drawDate( )

Once called, this procedure will draw the date at the bottom of the clock if not disabled by the user

3.2.3.5.4. arguments module

This module will get and parse all the command-line arguments that user may pass, returning a struct with values ready to be used for each module

3.2.4. How the project should be translated to a programming language

After listing the main parts of the software, the remaining work is just join everything to make the final software. The main structure of the program will be a loop that sleeps for 1 second, for each second, a set of functions will run to keep the clock updated and the screen responsible

3.2.5. How the tests should be performed

Test a software is a very important step on software development, and a software like the Rclock that has a high level of customization must be tested at the same level. All the tests will be performed inside the **tests/** directory, the first tests will be performed using unit tests, when a great part of the modules be ready for use, integration tests will be performed to make sure that the software will work as expected.