Documentation

Generated by Doxygen 1.9.1

Chapter 1

README

Author

Srikrishnan Venkatasubramanian

1.0.1 Muti-threading with REALTIME OPERATING SYSTEMS (RTOS)

Before deep diving into the installation procedure and usage of the above source code understanding the below terminologies are very important.

1.0.1.0.1 General Purpose Operating system A general purpose computer that can be used to solve multiple problem required by the user. Solution to the problem is provided in the form of service and a problem can have multiple soultions in the form of services.

1.0.1.0.2 Embedded System A microcomputer part of a larger system that has been designed to solve a specific problem by providing one or two dedicated service.

1.0.1.0.3 Realtime Operating System (RTOS) As the name specifies it is an operating system that is used for time-intended applications that is required to process data and perform some actions without any delays.

1.0.1.0.4 Soft deadlines A deadline if not met would not lead to catastrophic consquences. A general purpose computer is a great example for such deadlines.

1.0.1.0.5 Hard deadlines A deadline if not met would lead to catastrophic consquences. An embedded system is a great example for such deadlines.

1.0.1.1 Description

Majority of the world is full of electronic devices that are used to perform a specific task ∗∗(embedded system)∗∗. If we take a car, it will have an embedded system with sensors that are used to monitor various parameters such as brake, vehicle speed, inner temperature, oil temperature etc. The processing and reporting of such data without any delays is higly essential when it comes to such systems. Thus usage of RTOS for this purpose is very important. This project has been developed to gather data from multiple sensors of an embedded system , process it and display them to the user in Realtime using muti-threaded processing. Since we dont have the actual sensor, the provided datset is to be used to emulate the process. Based on the given requirements the following have been achieved:-

- A thread to read data from the provided dataset every 1 second.
- 8 threads for each parameter updating the defined structure at their respective defined intervals.
- Use main thread to display the parameters to the console every 500ms.

Posix threads and timers have been used for this purpose and the program has been developed in C++.

1.0.1.2 Installation

1. The project has been implemented on QNX RTOS. In order to launch the project and test the application in RTOS create a QNX account, request for a license or choose the 30-day trial.
2. Download the QNX software center and install Momentix IDE followed by the same version QNX SDP on your system.
3. After the installtion, open momentix IDE and click on symbol i on the top right corner of the screen (should open the target perspective).
4. On the left hand pane right click and choose create new virutal machine.
5. Give a name and choose the software for the virtual machine to be created with (Vmware,Vbox,qemu).
6. Once the virtual machine has been spun the target should be succesfully connected through Momentix IDE.

1.0.1.3 Running the project

1. Extract the source files in your workspace.
2. Create a new project with File -> QNX project -> C/C++ executable -> Name, C++, x86_64 architecture and launch
3. Copy the source files on to the created project source files.
4. Clean the project folder.
5. Click on symbol i on the top right corner of the screen (should open the target perspective).
6. Click on the target and open the Target file system navigator.
7. Copy paste the dataset.csv file in the tmp directory ∗∗(Will not allow you to put files anywhere else)∗∗.
8. Build the project.
9. Run the project on the target.

For further documentation and information go through the realtime_documentation.pdf or the index.html in the HTML directory.

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

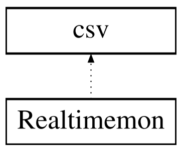
Chapter 5

Class Documentation

5.1 csv Class Reference

```
#include <csv.h>
```

Inheritance diagram for csv:



Protected Member Functions

- csv ()

    Default constructor stub that can be used based on requirement and design.
- virtual ∼csv ()

    Constructor.
- vector< string > read_csv_line (int line_number)

    Destructor.
- std::fstream & GotoLine (std::fstream &file, unsigned int num)

    Gets the particular line number to be read.

5.1.1 Detailed Description

Definition at line 21 of file csv.h.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 csv()

```
csv::csv ( )  [protected]
```

Default constructor stub that can be used based on requirement and design.

Definition at line 8 of file csv.cpp.

5.1.2.2 ∼csv()

```
csv::∼csv ( )  [protected], [virtual]
```

Constructor.

Default destructor stub that can be used based on requirement and design.

Definition at line 14 of file csv.cpp.

5.1.3 Member Function Documentation

5.1.3.1 GotoLine()

```
std::fstream & csv::GotoLine (
             std::fstream & file,
             unsigned int num )  [protected]
```

Gets the particular line number to be read.

The below function is used to read a particular line from a file.

It sets the file pointer for seekg to the line number.

Parameters

std::fstream&	file File handler provided as input from the read_csv_line(int line_number) function.
unsigned	int num Line number to be read from the file.

Returns

std::fstream File pointer.

Definition at line 25 of file csv.cpp.

5.1.3.2 read_csv_line()

```
vector< string > csv::read_csv_line (
            int line_number )  [protected]
```

Destructor.

Reads a particular line from a file, and tokenizes them into words using the delimitter ",".

The words are then stored into a vector which is returned by the function.

Parameters

int	line_number Get the line number to be read from a particular file.

Returns

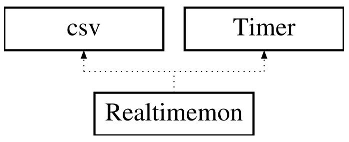
vector<string>

Definition at line 41 of file csv.cpp.

5.2 Realtimemon Class Reference

```
#include <Realtimemon.h>
```

Inheritance diagram for Realtimemon:



Public Member Functions

- Realtimemon ()

  Default constructor stub used to initialize the mutex.
- virtual ∼Realtimemon ()

  Default destructor stub used to destroy the mutex.
- int timer_create (int period)

  The defined function is used to create a timer for user defined time period.
- void wait_for_signal (void)

  The defined function is used to signal wait check that suspends the calling thread until there is a status change.
- void update_console (void)

  The defined function is used to display the data from the respective sensor variables every 500ms.
- void lock_variable (void)

  The defined function is used to lock a variable using mutex, thus preventing multiple threads trying to access a shared variable at the same time.Since elements of the base Timer class is protected its methods can only be accessed within the scope of the derived class.Below defined functions provides an efficient way to access base class outside anywhere maintaining at most safety and security and allows us to pass only one argument.
- void unlock_variable (void)

  The defined function is used to unlock a variable using mutex, thus releasing it to be accessed by other threads trying to update the variable.Since elements of the base Timer class is protected its methods can only be accessed within the scope of the derived class.Below defined functions provides an efficient way to access base class outside anywhere maintaining at most safety and security and allows us to pass only one argument.

Static Public Member Functions

- static void ∗ Store_data_in_vector (void ∗arg)

    The defined function is used along with a thread to read a line data every 1 second from the given dataset.csv file.
- static void ∗ Fuel_consumption (void ∗arg)

    The defined function is used to update the fuel_consumption variable in the structure parameter every 10 ms.
- static void ∗ Engine_speed (void ∗arg)

    The defined function is used to update the Engine_speed variable in the structure parameter every 500 ms.
- static void ∗ Engine_coolant_temperature (void ∗arg)

    The defined function is used to update the Enginer_coolant_temperature variable in the structure parameter every 2s.
- static void ∗ Current_gear (void ∗arg)

    The defined function is used to update the current_gear variable in the structure parameter every 100ms.
- static void ∗ Transmission_oil_temperature (void ∗arg)

    The defined function is used to update the Transmission_oil_temp variable in the structure parameter every 5s.
- static void ∗ vehicle_speed (void ∗arg)

    The defined function is used to update the vehicle_speed variable in the structure parameter every 100ms.
- static void ∗ Acceleration_speed_longitudinal (void ∗arg)

    The defined function is used to update the accelerarion variable in the structure parameter every 150ms.
- static void ∗ Indication_of_break_switch (void ∗arg)

    The defined function is used to update the brake_switch_indicator variable in the structure parameter every 100ms.

5.2.1 Detailed Description

Definition at line 17 of file Realtimemon.h.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 Realtimemon()

```
Realtimemon::Realtimemon ( )
```

Default constructor stub used to initialize the mutex.

Definition at line 8 of file Realtimemon.cpp.

5.2.2.2 ∼Realtimemon()

```
Realtimemon::∼Realtimemon ( )  [virtual]
```

Default destructor stub used to destroy the mutex.

Definition at line 14 of file Realtimemon.cpp.

5.2.3 Member Function Documentation

5.2.3.1 Acceleration_speed_longitudinal()

```
void ∗ Realtimemon::Acceleration_speed_longitudinal (
            void ∗ arg )  [static]
```

The defined function is used to update the accelerarion variable in the structure parameter every 150ms.

Parameters

void∗	arg General pointer passed as an argument to a function. This is used to pass the class object from the main function that allows it to access available variables and methods of the base and derived class .

Returns

void∗

Definition at line 210 of file Realtimemon.cpp.

5.2.3.2 Current_gear()

```
void * Realtimemon::Current_gear (
             void * arg )  [static]
```

The defined function is used to update the current_gear variable in the structure parameter every 100ms.

Parameters

void∗	General pointer passed as an argument to a function. This is used to pass the class object from the main function that allows it to access available variables and methods of the base and derived class .

Returns

void∗

Definition at line 132 of file Realtimemon.cpp.

5.2.3.3 Engine_coolant_temperature()

```
void * Realtimemon::Engine_coolant_temperature (
             void * arg )  [static]
```

The defined function is used to update the Enginer_coolant_temperature variable in the structure parameter every 2s.

Parameters

void∗	arg General pointer passed as an argument to a function. This is used to pass the class object from the main function that allows it to access available variables and methods of the base and derived class ..

Returns

void∗

Definition at line 106 of file Realtimemon.cpp.

5.2.3.4 Engine_speed()

```
void * Realtimemon::Engine_speed (
            void * arg )  [static]
```

The defined function is used to update the Engine_speed variable in the structure parameter every 500 ms.

Parameters

void∗	arg General pointer passed as an argument to a function. This is used to pass the class object from the main function that allows it to access available variables and methods of the base and derived class .

Returns

void∗

Definition at line 81 of file Realtimemon.cpp.

5.2.3.5 Fuel_consumption()

```
void * Realtimemon::Fuel_consumption (
            void * arg )  [static]
```

The defined function is used to update the fuel_consumption variable in the structure parameter every 10 ms.

Parameters

void∗	arg General pointer passed as an argument to a function. This is used to pass the class object from the main function that allows it to access available variables and methods of the base and derived class .

Returns

void∗

Definition at line 56 of file Realtimemon.cpp.

5.2.3.6 Indication_of_break_switch()

```
void * Realtimemon::Indication_of_break_switch (
            void * arg )  [static]
```

The defined function is used to update the brake_switch_indicator variable in the structure parameter every 100ms.

Parameters

void∗	arg General pointer passed as an argument to a function. This is used to pass the class object from the main function that allows it to access available variables and methods of the base and derived class .

Returns

void∗

Definition at line 236 of file Realtimemon.cpp.

5.2.3.7 lock_variable()

```
void Realtimemon::lock_variable (
            void  )
```

The defined function is used to lock a variable using mutex, thus preventing multiple threads trying to access a shared variable at the same time.Since elements of the base Timer class is protected its methods can only be accessed within the scope of the derived class.Below defined functions provides an efficient way to access base class outside anywhere maintaining at most safety and security and allows us to pass only one argument.

Parameters

void	

Returns

void

Lock variable

Definition at line 289 of file Realtimemon.cpp.

5.2.3.8 Store_data_in_vector()

```
void * Realtimemon::Store_data_in_vector (
            void * arg )  [static]
```

The defined function is used along with a thread to read a line data every 1 second from the given dataset.csv file.

(since data is usually collected every 1 second from an actual system this is done to emulate the system, where data is collected in a vector and the respective variables are updated according to the defined requirements.)

Parameters

void∗	arg General pointer passed as an argument to a function. This is used to pass the class object from the main function that allows it to access available variables and methods of the base and derived class .

Returns

void∗

Definition at line 27 of file Realtimemon.cpp.

5.2.3.9 timer_create()

```
int Realtimemon::timer_create (
            int period )
```

The defined function is used to create a timer for user defined time period.

Since elements of the base Timer class is protected its methods can only be accessed within the scope of the derived class. Below defined functions provides an efficient way to access base class outside anywhere maintaining at most safety and security and allows us to pass only one argument.

Parameters

int	period The time period for which the timer has to be created

Returns

int On successful timer creation yields 0 and error returns -1

Definition at line 263 of file Realtimemon.cpp.

5.2.3.10 Transmission_oil_temperature()

```
void * Realtimemon::Transmission_oil_temperature (
            void * arg )  [static]
```

The defined function is used to update the Transmission_oil_temp variable in the structure parameter every 5s.

Parameters

void∗	arg General pointer passed as an argument to a function. This is used to pass the class object from the main function that allows it to access available variables and methods of the base and derived class .

Returns

void∗

Definition at line 156 of file Realtimemon.cpp.

5.2.3.11 unlock_variable()

```
void Realtimemon::unlock_variable (
             void  )
```

The defined function is used to unlock a variable using mutex, thus releasing it to be accessed by other threads trying to update the variable.Since elements of the base Timer class is protected its methods can only be accessed within the scope of the derived class.Below defined functions provides an efficient way to access base class outside anywhere maintaining at most safety and security and allows us to pass only one argument.

Parameters

void	

Returns

void

Unlock Variable

Definition at line 303 of file Realtimemon.cpp.

5.2.3.12 update_console()

```
void Realtimemon::update_console (
             void  )
```

The defined function is used to display the data from the respective sensor variables every 500ms.

Parameters

void	

Returns

void

Definition at line 312 of file Realtimemon.cpp.

5.2.3.13 vehicle_speed()

```
void * Realtimemon::vehicle_speed (
             void * arg )  [static]
```

The defined function is used to update the vehicle_speed variable in the structure parameter every 100ms.

Parameters

void∗	arg General pointer passed as an argument to a function. This is used to pass the class object from the main function that allows it to access available variables and methods of the base and derived class .

Returns

void∗

Definition at line 184 of file Realtimemon.cpp.

5.2.3.14 wait_for_signal()

```
void Realtimemon::wait_for_signal (
            void  )
```

The defined function is used to signal wait check that suspends the calling thread until there is a status change.

Since elements of the base Timer class is protected its methods can only be accessed within the scope of the derived class. Below defined functions provides an efficient way to access base class outside anywhere maintaining at most safety and security and allows us to pass only one argument.

Parameters

void	

Returns

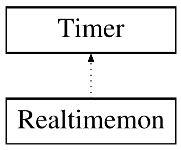
void

Definition at line 276 of file Realtimemon.cpp.

5.3 Timer Class Reference

```
#include <Timer.h>
```

Inheritance diagram for Timer:



Protected Member Functions

- Timer ()

    Default constructor stub that can be used based on requirement and design.
- virtual ∼Timer ()

    Constructor.
- void wait_next_activation (void)

    Destructor.
- int periodic_timer (uint64_t offset, int period)

    Signaling function.

5.3.1 Detailed Description

Definition at line 36 of file Timer.h.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 Timer()

```
Timer::Timer ( )  [protected]
```

Default constructor stub that can be used based on requirement and design.

Definition at line 12 of file Timer.cpp.

5.3.2.2 ∼Timer()

```
Timer::∼Timer ( )  [protected], [virtual]
```

Constructor.

Default Destructor stub that can be used based on requirement and design.

Definition at line 17 of file Timer.cpp.

5.3.3 Member Function Documentation

5.3.3.1 periodic_timer()

```
int Timer::periodic_timer (
              uint64_t offset,
              int period )  [protected]
```

Signaling function.

This function is used to calculate and set the required structure values for creating and setting a specified periodic timer.

Parameters

uint64←_t	constant in microsecond defined in the header. It is used for calculating and setting the requested time period in the struct itimerspec.
int	period Given as an input by the user. Defines the time interval for which the timer should be created

Returns

int returns -1 if error is setting the given time period.

Definition at line 40 of file Timer.cpp.

5.3.3.2 wait_next_activation()

```
void Timer::wait_next_activation (
            void  )  [protected]
```

Destructor.

The below function is used to suspend the calling process until a signal has been recieved.

Parameters

No	arguments

Returns

void.

Definition at line 25 of file Timer.cpp.

Chapter 6

File Documentation

6.1 csv.cpp File Reference

```
#include "csv.h"
```

6.1.1 Detailed Description

Author

Srikrishnan Venkatasubramanian.

6.2 csv.d File Reference

6.3 csv.h File Reference

```
#include <iostream>
#include <vector>
#include <fstream>
#include <sstream>
#include <string>
#include <limits>
```

Classes

- class csv

Macros

- #define CSV_H_

  Header gaurd to include certain files.
- #define FILENAME "/tmp/dataset.csv"

  Default path to the file on the target.

6.3.1 Detailed Description

Author

Srikrishnan Venkatasubramanian.

The Comma separated files (CSV) contains data of varying datatypes integer, string, double, float and depends on the provided dataset where the data collected may vary based on the given application. The csv class has been created to read a particular line from a given file and tokenize them according to requirement. The functions have been defined based on the requirements provided in the project description. Each line in the given dataset represents data from different sensors that have been collected every 1 second.

6.3.2 Macro Definition Documentation

6.3.2.1 CSV_H_

```
#define CSV_H_
```

Header gaurd to include certain files.

Definition at line 11 of file csv.h.

6.3.2.2 FILENAME

```
#define FILENAME "/tmp/dataset.csv"
```

Default path to the file on the target.

Definition at line 12 of file csv.h.

6.4 main.cpp File Reference

```
#include "Realtimemon.h"
```

Functions

- int main (void)
  The defined function acts the entry point that allows us to achieve the defined project requirements.

6.4.1 Function Documentation

6.4.1.1 main()

```
int main (
            void  )
```

The defined function acts the entry point that allows us to achieve the defined project requirements.

9 threads are created of which one thread is used to read data from csv, 8 are used to update the variables of interest using a shared structure, and main thread is used to display the variables to the console. Further, Realtimemon class object is created aand is used to access class methods and passed as an object to required threads.

Parameters

void	

Returns

int returns 0 on success and -1 on failure

Definition at line 17 of file main.cpp.

6.5 main.d File Reference

6.6 README.md File Reference

6.7 Realtimemon.cpp File Reference

```
#include "Realtimemon.h"
```

6.8 Realtimemon.d File Reference

6.9 Realtimemon.h File Reference

```
#include "csv.h"
#include "Timer.h"
#include <unistd.h>
#include <pthread.h>
```

Classes

- class Realtimemon

Macros

- #define SRC_REALTIMEMON_H_

  Header gaurd to include certain files.

6.9.1 Macro Definition Documentation

6.9.1.1 SRC_REALTIMEMON_H_

```
#define SRC_REALTIMEMON_H_
```

Header gaurd to include certain files.

Definition at line 10 of file Realtimemon.h.

6.10 Timer.cpp File Reference

```
#include "Timer.h"
```

Variables

- sigset_t sigst
  Signal.

6.10.1 Detailed Description

Author

Srikrishnan Venkatasubramanian.

6.10.2 Variable Documentation

6.10.2.1 sigst

```
sigset_t sigst
```

Signal.

Definition at line 9 of file Timer.cpp.

6.11 Timer.d File Reference

6.12 Timer.h File Reference

```
#include <sys/time.h>
#include <signal.h>
#include <time.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <stdio.h>
```

Classes

- class Timer

Macros

- #define SRC_TIMER_H_

  Header gaurd to include certain files.
- #define ONE_THOUSAND 1000
- #define ONE_MILLION 1000000
- #define OFFSET 1000000

  offset and period are in microsends.
- #define PERIOD10ms 10000

  Constant timer period macro for 10ms.
- #define PERIOD500ms 500000

  Constant timer period macro for 500ms.
- #define PERIOD2s 2000000

  Constant timer period macro for 2s.
- #define PERIOD100ms 100000

  Constant timer period macro for 100ms.
- #define PERIOD250ms 250000

  Constant timer period macro for 100ms.
- #define PERIOD5s 5000000

  Constant timer period macro for 5s.
- #define PERIOD150ms 150000

  Constant timer period macro for 150ms.
- #define PERIOD500ms 500000

  Constant timer period macro for 500ms.
- #define PERIOD1s 1000000

  Constant timer period macro for 1s.

6.12.1 Detailed Description

Author

Srikrishnan Venkatasubramanian.

Timers allows us to perform some tasks periodically at defined intervals or perform a task one-time from a defined time. Timers helps in easing the operations related to multi-threaded programming allowing us to wake up different threads at different times to perform respective tasks. The Timer class has been created mainly for handling the above defined activities. It has two functions where one is used to create a periodic timer and the other is used to emit a signal to indicate that the time. The type of signal used depends upon the defined requirements of the given application.

6.12.2 Macro Definition Documentation

6.12.2.1 OFFSET

```
#define OFFSET 1000000
```

offset and period are in microsends.

Definition at line 25 of file Timer.h.

6.12.2.2 ONE_MILLION

```
#define ONE_MILLION 1000000
```

Definition at line 23 of file Timer.h.

6.12.2.3 ONE_THOUSAND

```
#define ONE_THOUSAND 1000
```

Definition at line 22 of file Timer.h.

6.12.2.4 PERIOD100ms

```
#define PERIOD100ms 100000
```

Constant timer period macro for 100ms.

Definition at line 29 of file Timer.h.

6.12.2.5 PERIOD10ms

```
#define PERIOD10ms 10000
```

Constant timer period macro for 10ms.

Definition at line 26 of file Timer.h.

6.12.2.6 PERIOD150ms

```
#define PERIOD150ms 150000
```

Constant timer period macro for 150ms.

Definition at line 32 of file Timer.h.

6.12.2.7 PERIOD1s

```
#define PERIOD1s 1000000
```

Constant timer period macro for 1s.

Definition at line 34 of file Timer.h.

6.12.2.8 PERIOD250ms

```
#define PERIOD250ms 250000
```

Constant timer period macro for 100ms.

Definition at line 30 of file Timer.h.

6.12.2.9 PERIOD2s

```
#define PERIOD2s 2000000
```

Constant timer period macro for 2s.

Definition at line 28 of file Timer.h.

6.12.2.10 PERIOD500ms [1/2]

```
#define PERIOD500ms 500000
```

Constant timer period macro for 500ms.

Definition at line 33 of file Timer.h.

6.12.2.11 PERIOD500ms [2/2]

```
#define PERIOD500ms 500000
```

Constant timer period macro for 500ms.

Definition at line 33 of file Timer.h.

6.12.2.12 PERIOD5s

```
#define PERIOD5s 5000000
```

Constant timer period macro for 5s.

Definition at line 31 of file Timer.h.

6.12.2.13 SRC_TIMER_H_

```
#define SRC_TIMER_H_
```

Header gaurd to include certain files.

Definition at line 14 of file Timer.h.