COEN6341 REALTIME PROJECT

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 and open them in Momentix IDE.
2. Build the project.
3. click on symbol i on the top right corner of the screen (should open the target perspective).
4. Click on the target and open the Target file system navigator.
5. Copy paste the dataset.csv file in the tmp directory ∗∗(Will not allow you to put files anywhere else)∗∗.
6. Run the project on the target.

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Class Documentation

4.1 csv Class Reference

Public Member Functions

- csv ()

  Default constructor stub that has been auto-generated and 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.

4.1.1 Constructor & Destructor Documentation

4.1.1.1 csv()

```
csv::csv ( )
```

Default constructor stub that has been auto-generated and can be used based on requirement and design.

4.1.1.2 ∼csv()

```
csv::∼csv ( )  [virtual]
```

Constructor.

Default destructor stub that has been auto-generated and can be used based on requirement and design.

4.1.2 Member Function Documentation

4.1.2.1 GotoLine()

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

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.

4.1.2.2 read_csv_line()

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

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>

4.2 parameters Struct Reference

Structure that holds variables of different datatypes, representing different sensor variables to be updated and displayed on the console.

Public Attributes

- double fuel_consumption
- int Engine_speed
- int Enginer_coolant_temperature
- int current_gear
- int vehicle_speed
- double acceleration
- int Brake_switch_indicator
- int Transmission_oil_temp

4.2.1 Detailed Description

Structure that holds variables of different datatypes, representing different sensor variables to be updated and displayed on the console.

4.2.2 Member Data Documentation

4.2.2.1 acceleration

```
double parameters::acceleration
```

4.2.2.2 Brake_switch_indicator

```
int parameters::Brake_switch_indicator
```

4.2.2.3 current_gear

```
int parameters::current_gear
```

4.2.2.4 Engine_speed

```
int parameters::Engine_speed
```

4.2.2.5 Enginer_coolant_temperature

```
int parameters::Enginer_coolant_temperature
```

4.2.2.6 fuel_consumption

```
double parameters::fuel_consumption
```

4.2.2.7 Transmission_oil_temp

```
int parameters::Transmission_oil_temp
```

4.2.2.8 vehicle_speed

```
int parameters::vehicle_speed
```

4.3 Timer Class Reference

Public Member Functions

- Timer ()

  Default constructor stub that has been auto-generated and 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.

4.3.1 Constructor & Destructor Documentation

4.3.1.1 Timer()

```
Timer::Timer ( )
```

Default constructor stub that has been auto-generated and can be used based on requirement and design.

4.3.1.2 ∼Timer()

```
Timer::∼Timer ( )  [virtual]
```

Constructor.

Default Destructor stub that has been auto-generated and can be used based on requirement and design.

4.3.2 Member Function Documentation

4.3.2.1 periodic_timer()

```
int Timer::periodic_timer (
             uint64_t offset,
             int period )
```

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.

4.3.2.2 wait_next_activation()

```
void Timer::wait_next_activation (
             void  )
```

Destructor.

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

Parameters

No	arguments

Returns

void.

Chapter 5

File Documentation

5.1 COEN6341.cpp File Reference

Classes

- struct parameters

  Structure that holds variables of different datatypes, representing different sensor variables to be updated and displayed on the console.

Macros

- #define COEN6341_H

  Header gaurd to include certain files.

Functions

- 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.
- void ∗ Fuel_consumption (void ∗arg)

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

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

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

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

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

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

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

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

  The defined function is used to create threads, initiate mutex and destroy them on completion.

Variables

- vector< string > tokenized_data

  Global vector to store the data read every 1 second.
- pthread_mutex_t mutex_realtime

  mutex definition
- struct parameters line_data

  Global declaration of structure object line_data.

5.1.1 Detailed Description

Author

Srikrishnan Venkatasubramanian.

The given source files acts as the main interface between the two modules csv and timer defined as classes. Based on the requirements defined in the project description, threads have been created to read data occasionally to emulate the behavior of data arriving from sensor every 1 second. Additionally separate threads have been created for updating the sensor data to their respective sensor variables defined in a structure. The main thread is used to output the values on the console.

5.1.2 Macro Definition Documentation

5.1.2.1 COEN6341_H

```
#define COEN6341_H
```

Header gaurd to include certain files.

5.1.3 Function Documentation

5.1.3.1 Acceleration_speed_longitudinal()

```
void* Acceleration_speed_longitudinal (
             void * arg )
```

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. If the argument is to be used it needs to be type casted appropriately before usage.

Returns

void∗

5.1.3.2 Current_gear()

```
void* Current_gear (
            void * arg )
```

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

Parameters

void∗	arg General pointer passed as an argument to a function. If the argument is to be used it needs to be type casted appropriately before usage.

Returns

void∗

5.1.3.3 Engine_coolant_temperature()

```
void* Engine_coolant_temperature (
            void * arg )
```

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. If the argument is to be used it needs to be type casted appropriately before usage.

Returns

void∗

5.1.3.4 Engine_speed()

```
void* Engine_speed (
            void * arg )
```

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. If the argument is to be used it needs to be type casted appropriately before usage.

Returns

void∗

5.1.3.5 Fuel_consumption()

```
void* Fuel_consumption (
            void * arg )
```

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. If the argument is to be used it needs to be type casted appropriately before usage.

Returns

void∗

5.1.3.6 Indication_of_break_switch()

```
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.

Parameters

void∗	arg General pointer passed as an argument to a function. If the argument is to be used it needs to be type casted appropriately before usage.

Returns

void∗

5.1.3.7 main()

```
int main (
            void  )
```

The defined function is used to create threads, initiate mutex and destroy them on completion.

Parameters

No	arguments

Returns

void

5.1.3.8 Store_data_in_vector()

```
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.

(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. If the argument is to be used it needs to be type casted appropriately before usage.

Returns

void*

5.1.3.9 Transmission_oil_temperature()

```
void* Transmission_oil_temperature (
            void * arg )
```

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. If the argument is to be used it needs to be type casted appropriately before usage.

Returns

void∗

5.1.3.10 update_console()

```
void update_console ( )
```

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

Parameters

No	arguments

Returns

void

5.1.3.11 vehicle_speed()

```
void* vehicle_speed (
            void * arg )
```

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. If the argument is to be used it needs to be type casted appropriately before usage.

Returns

void∗

5.1.4 Variable Documentation

5.1.4.1 line_data

```
struct parameters line_data
```

Global declaration of structure object line_data.

5.1.4.2 mutex_realtime

```
pthread_mutex_t mutex_realtime
```

mutex definition

5.1.4.3 tokenized_data

```
vector<string> tokenized_data
```

Global vector to store the data read every 1 second.

5.2 COEN6341.d File Reference

5.3 csv.cpp File Reference

5.3.1 Detailed Description

Author

Srikrishnan Venkatasubramanian.

5.4 csv.d File Reference

5.5 csv.h File Reference

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.

5.5.1 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.

5.5.2 Macro Definition Documentation

5.5.2.1 CSV_H_

```
#define CSV_H_
```

Header gaurd to include certain files.

5.5.2.2 FILENAME

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

Default path to the file on the target.

5.6 README.md File Reference

5.7 Timer.cpp File Reference

Variables

- sigset_t sigst

  Signal.

5.7.1 Detailed Description

Author

Srikrishnan Venkatasubramanian.

5.7.2 Variable Documentation

5.7.2.1 sigst

```
sigset_t sigst
```

Signal.

5.8 Timer.d File Reference

5.9 Timer.h File Reference

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 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.

5.9.1 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.

5.9.2 Macro Definition Documentation

5.9.2.1 OFFSET

```
#define OFFSET 1000000
```

offset and period are in microsends.

5.9.2.2 ONE_MILLION

```
#define ONE_MILLION 1000000
```

5.9.2.3 ONE_THOUSAND

```
#define ONE_THOUSAND 1000
```

5.9.2.4 PERIOD100ms

```
#define PERIOD100ms 100000
```

Constant timer period macro for 100ms.

5.9.2.5 PERIOD10ms

```
#define PERIOD10ms 10000
```

Constant timer period macro for 10ms.

5.9.2.6 PERIOD150ms

```
#define PERIOD150ms 150000
```

Constant timer period macro for 150ms.

5.9.2.7 PERIOD1s

```
#define PERIOD1s 1000000
```

Constant timer period macro for 1s.

5.9.2.8 PERIOD2s

```
#define PERIOD2s 2000000
```

Constant timer period macro for 2s.

5.9.2.9 PERIOD500ms [1/2]

```
#define PERIOD500ms 500000
```

Constant timer period macro for 500ms.

5.9.2.10 PERIOD500ms [2/2]

```
#define PERIOD500ms 500000
```

Constant timer period macro for 500ms.

5.9.2.11 PERIOD5s

```
#define PERIOD5s 5000000
```

Constant timer period macro for 5s.

5.9.2.12 SRC_TIMER_H_

```
#define SRC_TIMER_H_
```

Header gaurd to include certain files.