Final Report

A Report
Presented to
The Department of Electrical & Computer Engineering
Concordia University

In Partial Fulfillment
of the Requirements
Of COEN 320

By
Evan Lamenta 27240007 lamentae@hotmail.com
Jasen Ratnam 40094237 jasenratnam@hotmail.com
Karthikan Jeyabalan: 40032932 jeya.karthikan@gmail.com

Professor: Rodolfo Coutinho

Concordia University
December 7th, 2020

Table of Content

Detailed contribution of each member:

Evan Lamenta

Evan worked on writing the function responsible for reading the dataset into a data structure that could then be used by the rest of the program. He did the preprocessing of the dataset by manually removing unused variable columns.

Jasen Ratnam

Jasen participated in the design of the real-time system program to fetch and save data from sensors periodically with given periods and then send the data to display thread through shared memory space.

Karthikan Jeyabalan

Karthikan participated in the design of the real-time system. He also worked on the display real-time periodic thread which prints out the data to the console. He contributed on Introduction, analysis and implementation of the display data thread in the report.

Objective

The objective of the project is to implement a set of real-time tasks to monitor the vehicle data and display it to the driver. The project must be written in C++ and run on the QNX operating system. One data producer thread must be created to periodically read sensor data, and one data consumer thread must be created to periodically display information.

Introduction

This project consists of implementing a real-time monitoring and diagnosing a vehicle's health. This can be achieved by reading multiple sensors provided by the vehicle's engine control unit(ECU). The ECU is responsible for monitoring and logging all the sensor data of the vehicle. The data from the ECU can be retrieved using the ODB-2 port. The ODB-2 port is a standard port on most vehicles. For this project we will use a sample dataset from which will mock the data from the collected from the ECU. The given dataset was retrieved from a Kia Soul where all the data was sampled at a frequency of one second.

This project will display the variable of interest to the user at a predefined period. The monitoring system will display the fuel consumption, engine speed, engine coolant temperature, current gear, transmission oil temperature, vehicle speed, acceleration speed longitudinal and indication of break switch. Each variable will have its own periodicity as well.

The monitoring system is required to have a real-time periodic thread for each variable. This thread will be responsible for retrieving the appropriate data with respect to its periodicity.

The system will also have a real-time periodic thread to display all the information to the driver. This thread will have a predefined period of 0.5 seconds to update the driver display. In addition, each sensor thread will have to communicate with the display thread in order to display the correct data.

Analysis

In this project, we are using a dataset in the form of .csv, we will have a thread that would simulate the ODB-2 port. This thread will read the dataset every one second and update the variables of interests.

According to the requirements, we have 8 data producer threads with their respective periodicity (Table 1). These threads will periodically fetch their corresponding values which were updated using the .csv thread. These threads will wake up, execute and then sleep for their periodicity duration. These steps will take place recursively.

Variable	**Periodicity**
Full Consumption	10 ms
Engine Speed (RPM)	500 ms
Engine Coolant Temperature	2 s
Current Gear	100 ms
Transmission Oil Temperature	5 s
Vehicle Speed	100 ms
Acceleration Speed Longitudinal	150 ms
Indication of break switch	100 ms

Table 1: Variable of Interests with Periodicity

The monitoring system also has a real-time thread which is responsible for displaying the data to the driver. This thread, also known as data consumer thread, will refresh the display at 0.5 seconds using the values updated from the data producer threads. This thread will print all the values on to the display and then sleep for 0.5 seconds. These steps will also run recursively.

Design

To accomplish this project, our team planned to use C++ in the QNX IDE along with tools to facilitate working on this project remotely such as Facebook Messenger as a communication channel and for management tools, we will use Github and Google drive to manage our codes and files in an organized way.

The first step of the design is to preprocess the dataset. Only a subset of the variables in the data set will be used, so to save memory space the data set needs to be reduced to only the required variables.

The second step is to create reading threads, where each variable of interest will have its own thread that consists of reading a specific variable from the dataset and then updating a shared variable in memory. Each thread will have a period variable that will be used by the scheduler to schedule each thread as a task and place them in the appropriate queue.

A scheduling algorithm will be needed to schedule these threads, we will use the earliest deadline first scheduling algorithm for the system. Specifically non-preemptive EDF, this will simplify the implementation by avoiding the need for context-switching. One scheduler class will be created with functions for managing the queues of tasks to be performed. The scheduler will implement data structures for the ready, waiting and running queues.

The final step will be to create a thread to display the variables. The result of the system will be outputted using a command line interface to display the data collected from the OBD 2. This will result in the display that will run independently from the rest of the components. The display thread will get its data from the shared memory where each of the variable threads will update its own shared memory.

The system will use a shared memory where the data producer threads need to communicate with the data consumer thread to give it the periodic data gotten from the vehicle dataset. To facilitate this communication we will implement the concept of shared memory in our project. To use shared memory, we need to use semaphores and mutexes to protect it from mishandling and errors.

Implementation

The program was initially planned to be implemented using a signal timer and a scheduler. But while doing the implementation, we have realized that using the sleep() function and the default scheduler of QNX is much simpler than creating a scheduler and timer class from scratch.

To start the implementation, we preprocess the dataset given to keep only a subset of the variables in the data set that we will use in order to save memory space in the system. The dataset is preprocessed to keep only the Full Consumption, Engine Speed (RPM), Engine Coolant Temperature, Current Gear, Transmission Oil Temperature, Vehicle Speed, Acceleration Speed Longitudinal and the Indication of break switch. We initially had problems trying to read the dataset with the file placed on a windows path, we then realized that the dataset needs to be placed in the target system to be able to read. Hence, this preprocessed data set is then placed in the Target VM file system navigator in the QNX to be able to access it while running the code in the target system. The file is inaccessible if it is placed in a windows file outside of the virtual machine. We noted that the dataset consists of data gotten from sensors every second but we require to fetch data from sensors at smaller periods. This will be handled in the thread reading the dataset.

The code first runs the main thread where different threads for reading the file, fetching the wanted variables and displaying the variables are created and initialized, then the main thread waits for the threads to end before ending the program. Since the threads are infinitely periodic, they will never end, hence the main thread will never end and the program will never terminate.

Once the threads are created and initialized, the thread read_csv_thread() starts executing to read the dataset file line per line and places the variables of the current period from the sensors in a buffer array. Semaphore waiting and mutex locks are used to keep the variables safe from other threads. We had some difficulties getting the semaphores and mutex locks to work perfectly in our desired way, but we were able to make it work. Once the first line of the dataset is read and saved into the buffer, the thread then goes to sleep for 1 second. Going to sleep for 1 second ensures that the same variables are used for the second. This is needed because the dataset shows sensor variables gotten every second but we may need to fetch data from the sensors with periods less than a second. After 1 second, the thread wakes up and reads the second line and saves the variables from the sensors for the 2nd second. This process is repeated indefinitely.

There are 8 producer threads which read every data from sensors from the buffer array gotten in the reading thread. The first thread is the fuel_consumption() thread which produces the data from the fuel consumption sensor. This first producer thread waits until the semaphore and mutex locks are released from the file reader thread. This is needed to ensure that the program does not start fetching data before data is read or fetching the wrong data. Once the first producer gets the semaphore and is able to fetch data for the fuel consumption, the other 7 producer threads can execute consecutively without needing to check semaphores since we know that the data is already accurate from the first producer. Every producer thread has its own period that is followed infinitely. The threads wake up every period from the sleep function and fetch the current data from the sensor by taking their index from the buffer. This data is then saved for that period and it will be used until the next period even if the sensor now has another data. The first producer fetches for the fuel consumption data every 10ms, the second producer fetches for the engine speed in RPM every 500ms, the third producer fetches the engine coolant temperature every 2s, the fourth producer fetches the current gear every 100ms, the fifth producer fetches the transmission oil temperature every 5s, the sixth producer fetches the vehicle speed every 100ms, the seventh producer fetches the acceleration speed in longitudinal every 150ms and finally the 8th and last producer fetches the indicitation of the break switch every 100ms. These thread infinitely fetch data from the sensors every period. Since we are not using a OBDII port with continuously updating sensors but using a dataset with a fixed number of sensor data, the program will eventually reach the end of the file, where it will continue reading the same line over and over until the program is terminated by the user.

The program consists of one more thread, this is the consumer thread called display_variables(). This thread is independent of the other threads and displays the variables gotten from the sensors periodically. This thread will display the data fetched from the producer threads in the console periodically every period DispTime in microsecond. The program sets this period to a default of 0.5s, 500000 microsecond. This time can be changed to see the variable changes in a shorter or longer period. The thread displayed the data from sensors saved from the producers no matter what the actual data from the sensors are.

Lessons learned

The project required us to use the QNX neutrino operating system which allowed us to learn how to use this new operating system. The use of QNX made it much easier to implement real-time tasks. Normally, if the project was to be implemented on a windows operating system it would require the development of a scheduling algorithm and the creation of multiple queue data structures to keep track of the state of the threads that are running, waiting, or terminated. QNX provided the ability to schedule threads automatically which made the implementation of the project more efficient.

The design that was used for the project is coded in such a way that does not allow for easy extension of the system. If another thread is required for reading additional data from the dataset, the code for reading will have to be copied and added to the main file. The project could be made better by creating a class designed specifically for reading a generic amount of sensors that could be instantiated by a driver program. This would make it far easier to extend the system to read any number of sensors.

Conclusion

For every variable of interest the program was able to read the variable within the period outlined in the requirements. The program was able to display the information to the user with the specified period as well. The most important concepts used were how to create threads in QNX, how to put threads to sleep, and how to use mutex locks and semaphores to control access to variables of interest. Any future work that involves QNX or periodic real-time task scheduling can be attempted by the team now that experience has been gained during the development of this project.