KF6010
Distributed Real-Time Systems
Course Work 2018-19

Module tutor: Alun Moon

1 Assessment submission and feedback

Date of hand out to students:
11 February 2019

Mechanism to be used to disseminate to students:
eLP

Date and Time of Submission by Student:
23.59 on 16 May 2019

Mechanism for Submission of Work by Student:
You should submit a single report in pdf format via eLP using the link as follows:

- Report: `Assessment->Report`.

The answers in your report must be labelled clearly with the section number and question number to which they relate.

Date by which Work, Feedback and Marks will be returned to Students:
6 June 2019

Mechanism(s) for return of assignment work, feedback and marks to students:
email with meeting by appointment on request.

2 Real-time Scheduling

2.1 Basic Scheduling Analysis

Consider the following set of fixed-priority periodic tasks:

Process	CPU	Pd	Deadline
A	42	330	60
B	72	720	660
C	6	240	240
D	132	360	300
E	18	48	48
F	18	4800	120

(a) What is the total *CPU utilisation* of the set? Show how you computed it.
(1 mark)

(b) Is the set schedulable using the *rate-monotonic* priority assignment? Give the reasoning by which you decide this (utilisation bound theorem? Completion (response) time computation and theorem?)
(2 marks)

(c) Suppose the programming of process E is streamlined so that the CPU time required by it reduces from 18 to 14. Show that a rate-monotonic schedule is now feasible and compute the utilisation and the task response times. Show the details of the response-time estimate calculation and explain the principles and assumptions on which it is based. Are the deadlines met?
(6 marks)

(d) If the set is scheduled using *deadline-monotonic* fixed-priority assignment, what are the response times? Are the deadlines met?
(2 marks)

(e) Discuss (in around 500 words) the limitations of using utilisation bounds as a simple test of schedulability, and discuss how the method can be made less pessimistic. Devise and sketch the time-line of an example which fails the basic test but passes an improved test.
(4 marks)

2.2 Scheduling with Shared Resources

(a) Discuss the kinds of *resources* a set of tasks might have to share mutually exclusively – and give an example which exhibits *priority inversion*. How does simple *priority inheritence* address this, and what are its limitations? Discuss, with a suitable example, and suggest a possible alternative to simple priority inheritence.
(6 marks)

(b) A set of 5 tasks a, b, c, d, e share two mutually exclusive resources S, T. The tasks have priorities, release time offsets and execution patterns as follows. E denotes a time tick during which the task is executing with neither resource locked; `S`, `T`, `(S+T)` denote ticks during which the task is executing with S, T, or both resources locked, respectively.

Process	priority	release time	execution pattern
a	5(hi)	14	`EESSEE`
b	4	10	`EETTEE`
c	3	8	`EEEE`
d	2	4	`EESSSS(S+T)(S+T)(S+T)SEE`
e	1	0	`EETTTTTTTTEE`

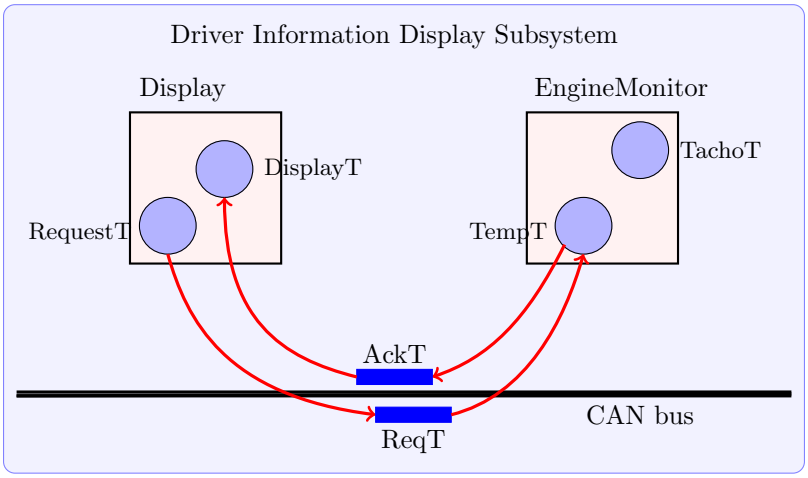
Draw a time line show how these processes run in a schedule which uses the priority inheritence protocol. Show when processes are indirectly as well as directly blocked and provide comments explaining the blocks and dynamic priority changes.

(4 marks)

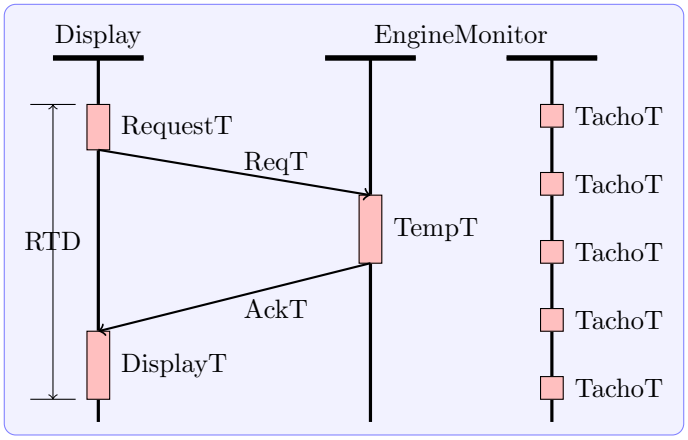
3 A Distributed Real-time System

3.1 Introduction

This problem concerns the development and analysis of a distributed real-time system which uses `CAN` for inter-processor communications. You are required to develop and consider the analysis of a simple distributed system programmed in C. The system is a **Driver Information Display Subsystem**, a component of an automobile communication and control system. Software on one processor, `Display`, requests the current value of the engine temperature from the `EngineMonitor`, running on another processor. The `Display` system displays the current engine temperature value returned by the `EngineMonitor` system. The temperature task `TempT` of the `EngineMonitor` runs concurrently with a task `TachoT` that periodically measures the speed of rotation of the engine. The software task structure of the system is depicted in the following diagram:



The `EngineMonitor` system is a time-triggered system. The `Display` system is an event-triggered system implemented using the concurrency mechanisms of `mbed OS-5`. The task `Display.RequestT` periodically transmits a request message to the `EngineMonitor` processor. The `EngineMonitor.TempT` task periodically polls the CAN controller for the arrival of a request message. When a request message is received, the `TempT` task reads the local temperature and sends a reply message containing the temperature value. Another periodic task, `TachoT`, runs on the `EngineMonitor` processor. It may be scheduled to run before `TempT` in the same frame. The event-triggered task `Display.DisplayT` is released by the interrupt handler for the CAN controller, which runs on the arrival of a message containing the temperature value. When it receives a message, it displays the value. Thus, tasks on the two processors engage in a simple message exchange, as indicated by the following message sequence diagram:



Task/Message	Arrival Event	Period
Display.RequestT	Periodic mbed OS-5 task	500 ms
Display.DisplayT	Periodic mbed OS-5 task	Inherited
EngineMonitor.TempT	Periodic time-triggered task	Inherited
EngineMonitor.TachoT	Periodic time-triggered task	20 ms
CAN.ReqT	Periodic Completion of RequestT	Inherited
CAN.AckT	Periodic Completion of TempT	Inherited

The computation times of tasks remains for you to measure and the transfer times of messages should be calculated.

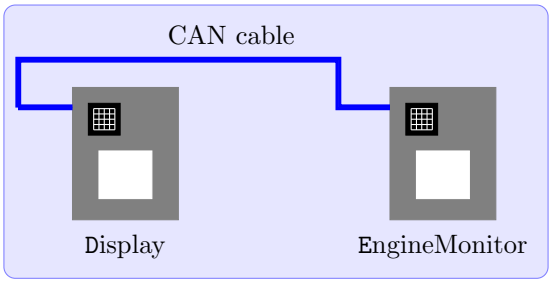
3.2 Procedure

Your system is required to run on the FRDM-K4F boards You should begin your solution by importing the git repositories for the `Display` and `EngineMonitor` subsystems as follows:

- `mbed import https://github.com/davidkendall/Display`
- `mbed import https://github.com/davidkendall/EngineMonitor`

The `Display` subsystem uses `mbed OS-5`. The `EngineMonitor` subsystem uses a time-triggered scheduler. The CAN driver is provided in each case in `src/can.h` and `src/can.c`.

Develop your software, following the guidance provided in the commented project files. Instrument your software to gather measurements for later analysis.



3.3 Deliverables and Assessment

There are 50 marks allocated to this part of the assessment. Your report should include answers that clearly address the requirements below:

1. Write a short commentary on your software for the Driver Information Display Subsystem. Focus on the aspects that you consider interesting and/or challenging. Critically evaluate your solution. (You may attach selected extracts from your code in an appendix to your report.)

   (20 marks)

2. Measure the computation times of the tasks: `TempT` and `TachoT`. Describe the measurement procedure(s) used and explain the likely quality of the results. Using your measurements, calculate the worst-case time between the arrival of a `ReqT` message and the release of an `AckT` message on the `EngineMonitor` processor. Discuss how use of a time-triggered scheduler affects the analysis of response times for this system and contrast this with the use of a RTOS such as mbed OS-5. Critically evaluate the advantages and disadvantages of each technique.

   (15 marks)

3. Calculate the response time of the message `AckT`. Explain clearly the method adopted to perform the calculation. You may assume for the purpose of this calculation that there is no jitter in the release time of any messages

   (7 marks)

4. Explain what you understand by *predictability* and discuss its role in a sound engineering approach to the development of embedded control systems, such as the one in this scenario.

   (8 marks)

Conducting the practical work You will need to work on the **practical part** of this assessed exercise with a partner, i.e., you should develop and test the small distributed system with a partner and share your experimental results. However, the report you submit must be your own work - the description of your methods, any results obtained analytically, and the interpretation of the results must be your work alone - do not collaborate with your partner over the interpretation of the data or the writing of your report. If you refer to the work of others (academic papers, web resources, etc.) provide full references to your sources.

4 Reliability

1. A car production line contains a number of conveyors and robotic tools controlled by networked processors which communicate data gathered by various sensors. This system requires certain level of fault tolerance, as a failure of any part of it will result in the production line clogging, loss of productiom and possible injury to personnel and damage to the cars.

   (a) Explain the different levels of *fault tolerance* of real-time systems and appraise their applicability to this system. Which do you consider the most appropriate? Give reasons.

   (6 marks)

   (b) The control system consists of a number of components communicating via a network. Give a critical appraisal of the risks posed by the network and suggest an architecture that minimises the risks.

   (3 marks)

   (c) Briefly suggest other harware risks, and how they might be mitigated.

   (3 marks)

2. Download and unzip SensorSimulation.zip. Read the file `README.md`, which includes instructions for use of the simulator. Imagine a scenario in which a sensor monitors some condition of some plant - steam pressure or temperature, say, or current level in the windings of a generator or motor - and the plant must be shut down immediately if the reading becomes dangerously high.

   The sensor may be "noisy" like the simulated one, and the "noise" level may well be around 2.5 to 5% of the full-scale reading of the sensor. In the simulation, imagine the full-scale reading is 200: so a noisy sensor with noise = 5 is realistic.

   Suppose that we have to do an emergency shut-down if the value reaches 150. A noisy sensor might exceptionally trigger a shut-down when the nominal value is only 140 or even less, because "noise" boosts the reading to 150. We may decide we have to live with that. Alternatively we can create several sensors and take as our reading their *average*. Theory predicts that the "noise" would cancel out, and the noise in the average of n sensor readings is only $(1/\sqrt{n})^{th}$ of the noise in a single reading. Of course, is is probably uneconomic or impractical to have a very large number of sensors.

   There is another potential problem, however. Our sensor might be faulty, and give us a high reading from time to time and this would trigger a shut-down. The shut-downs are required for safety reasons, but are expensive and we really need to avoid false-alarms. We might again employ several sensors, of which one might be faulty. Averaging the readings again will give some protection from a faulty high reading but the average could

still be much higher than it should be. A better approach might be to implement a *voter* which checks all the readings are within tolerance of each other and rejects "out-liers", returning the average of the remainder.

(a) Record the output of a couple of minutes' run of the basic `Simulation` application as provided (ie, using a `SensorSim` with nominal value initially = 100 and noise = 5). You can redirect the output to a text file, as follows:

```
$ java Simulation > log.txt
```

Repeat this exercise for a couple of minutes' run of a Simulation application in which a `FaultySensorSim` is substituted.

In each case, state the number of outputs, the maximum absolute discrepancy of a sensor reading from its nominal value, and the mean and standard deviation of the sensor readings.

(3 marks)

(b) Make a copy of `Simulation.java` and change the code in the `runSimulation()` method so that an array of SensorSim objects are employed rather than a single one. They should all be given the same initial nominal value and noise value and the random-walk should adjust them all in the same way. The reading should be an average of their readings. Rather than hard-coding the size of this array, make it a parameter passed in through the constructor. Create a system with 1 faulty sensor and 2 non-faulty sensors. Run the simulator for a couple of minutes. Capture and present the results in your report, describing the procedure and parameters used to obtain them. Repeat the simulation with 1 faulty sensor and 3 non-faulty sensors and present these results in your report also. Include an abbreviated code listing of your modified `Simulation.java` file as a clearly titled appendix to your report.

(3 marks)

(c) Investigate the ability of an approach based on a voter function to reduce the number of false emergency shut-downs that may occur with the simple averaging approach considered above. For example, one approach is to reject any reading that deviates too far (more than twice the noise level) from the average of the other readings, taking this as 'the' average reading. Create simulations based on this idea for 1 faulty + 2 non-faulty sensors and 1 faulty + 3 non-faulty sensors. Present your procedure, parameters, and results for each case. Critically evaluate and compare both approaches. Include an abbreviated code listing of your modified `Simulation.java` file as a clearly titled appendix to your report.

(7 marks)

5 Further information

Learning Outcomes assessed in this assessment: This assignment assesses all module learning outcomes, as indicated below:

1. Evaluate the scheduling and resource management requirements of real-time systems and how to make effective use of the associated algorithms
2. Assess the problems involved in developing distributed real-time systems and their solutions
3. Assess the issues relating to software engineering development of real-time distributed systems and apply your knowledge in the creation of such systems.
4. Appraise and produce reliable, fault tolerant real-time software.
5. Research a distributed systems problem and reflect upon the technical solution, considering security issues and professional standards

Referencing Style: Harvard

Expected size of the submission: Your report should be about 9 to 10 A4 pages in length, excluding appendices (assuming 10pt and normal margins). There is no fixed penalty for exceeding this limit but unnecessary verbosity, irrelevance and 'padding' make it difficult for the marker to identify relevant material and may lead to some loss of marks.

Assignment weighting: 100%

Academic Integrity Statement: You must adhere to the university regulations on academic conduct. Formal inquiry proceedings will be instigated if there is any suspicion of plagiarism or any other form of misconduct in your work. Refer to the University's Assessment Regulations for Northumbria Awards if you are unclear as to the meaning of these terms. The latest copy is available on the University website.