CM0605
Embedded Systems Engineering
Assignment Portfolio
Academic year 2017-18

Dr M J Brockway (Module Tutor), Dr David Kendall

September 2017

Issued Teaching week 6 in classes and on the web.

Due 8 January 2018, 2359; work to be submitted electronically using link provided on the e-learning portal ('Blackboard').

Marks and feedback available about three weeks therafter. Your tutors will provide verbal feedback on your progress with the portfolio in lab classes.

Academic Misconduct Except as stated in section 3, this is *individual* work and must be *your own*. If you refer to the work of others (academic papers, web resources, etc.) provide full references to your sources.

Submitting the portfolio There are three items in the portfolio: two pieces of work for Dr Michael Brockway, worth 25 marks each, and one piece for Dr David Kendall worth 50 marks. The complete portfolio of three items should be submitted together in a single PDF document on the due date. Within the portfolio each of the three pieces of work should be presented as a separate section headed with the title of the work (e.g. *Real-Time Scheduling*) and the lecturer who set it.

Managing your time Although the due date is in January, try to have all lab work completed by the end of the autumn term. There will be time in lab classes available for this, and you will be able to obtain advice in lab classes from your tutors.

1 Real-time Scheduling (M Brockway)

Learning Outcomes covered "Evaluate the scheduling and resource management requirements of real-time systems and make effective use of the associated algorithms"

1.1 Basic Scheduling Analysis (15 marks)

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).

1.2 Scheduling with Shared Resources (10 marks)

(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)

2 Reliability (M Brockway)

Learning Outcomes covered

1: Assess the issues relating to software engineering development of embedded systems and apply their knowledge in the creation of such systems.

2: Appraise and produce reliable, fault tolerant real-time software.

4: Assess the problems involved in developing distributed real-time systems and construct solutions to these problems

2.1 Report (12 marks)

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.

1. (6 marks) 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.
2. (3 marks) 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. (3 marks) Briefly suggest other harware risks, and how they might be mitigated.

2.2 An N-version Programming Investigation (13 marks)

Download and unzip `SensorSimulation.zip` .

This bundle includes a java class `SensorSim` which simulates a sensor measuring some analogue quantity like steam pressure, temperature or electrical voltage in some plant. It is a *noisy* sensor: the value it returns is a *nominal* value plus or minus a a random amount of *noise*. The noise is simulated using probability-theoretic techniques noise values are random numbers with a *normal* probability distribution with a configurable *standard deviation*. The larger this is, the noisier the simulated sensor.

To use the class, all you need to know about is the two configurable attributes, the nominal value and the amount of noise (the mean and standard deviation of

the normal population being simulated). These are passed into the constructor of `SensorSim` which then starts the simulation running in its own thread.

The nominal value can be adjusted by means of method `setNominal(double)` which the simulation is running, and the sensor reading (nominal val + noise) retrieved by method `double getRdg()`.

The class `Simulation` runs a simulated sensor with nominal value initially 100 and noise (standard deviation) 5, and at 500 millisecond intervals retrieves readings and displays them on a graph. Try it now:

`java Simulation`.

Notice that it also outputs data to the console in the format *reading (nominal): difference*. By redirecting output to a text file you can log this output for experimental purposes.

In addition, `Simulation` random-walks the nominal sensor value up and down.

A *faulty* sensor is one which sporadically gives an erroneous reading.
`FaultySensorSim` is a subclass of `SensorSim` which does this. Two extra attributes need to supplied to the constructor (besides the initial nominal value and the noise value) an integer specifying the fault frequency (actually the mean time between faults), and a double which is an erroneous value returned by `getRdg()` instead of the correct (though noisy) one. Fault occurrence is simulated by a Poisson process (like cars passing or rain drops landing) and the parameter is the mean interval between faults, in milliseconds.

Try using a `FaultySensorSim` instead of a `SensorSim` instance in `Simulation`. Open the source file `Simulation.java` and find the line in the `runSimulation` method which constructs a `SensorSim`.

```
sensor = new SensorSim(sensorNom, sensorErr);
```

Change this to -

```
sensor = new FaultySensorSim(sensorNom, sensorErr, 10000, 200);
```

Re-compile and run this to see the effect of a faulty sensor with a fault interval of 10000 milliseconds and a fault value of 200. Try other fault frequencies and fault values.

Introducing the Investigation Imagine a set-up 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.

Warm-up task 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). Redirect the output to a text file thus:

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

or if you want to see the output on the console as well as log it, try
`java Simulation | tee log.txt`.

Find out the maximum discrepancy of a sensor reading from its nominal value (ie maximum noise).

Also record the output of a couple of minutes' run of a Simulation application in which a `FaultySensorSim` is substituted, as explained above. How often are the faults occurring?

You need not submit your detailed logs but submit a brief report in each case 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.

Main task 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.

Make one of the simulated sensors is a `FaultySensorSim` and the remainder noisy but non-faulty `SensorSim` instances. Investigate the behaviour of this simulation with three (1+2) sensors: how high can the nominal value go before an emergency shut-down is triggered? Repeat with four (1 faulty + 3 non-faulty) sensors.

Can you devise a voter function that gives better protection against 'false alarm' shut-down?

One idea 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.

Investigate the ability of this voting approach to protect the system from false emergency shut-down due to a single faulty sensor and give a comparison with the simple averaging approach. Submit a report of your investigation including

1. (2 mks) a brief report of the warm-up task;
2. (2 mks) an abbrievated code listing of your modified `Simulation.java` employing an array of simulated sensors including a faulty one;
3. (4 mks) an abbrievated code listing of a further modified `Simulation.java` employing a voter function;
4. (2 mks) reports of results of each of these simulations over a few minutes' run with 3 (= 1+2) and with 4 (=1+3) sensors;
5. (3 mks) your conclusions.

You may abbreviate your code listings - I just need to know where your modifications fit into the orginal `Simulation.java`.

Your report of the investigation should clearly describe the procudure you followed and the settings you used and the results you obtained.

3 A Distributed Real-time System (D Kendall)

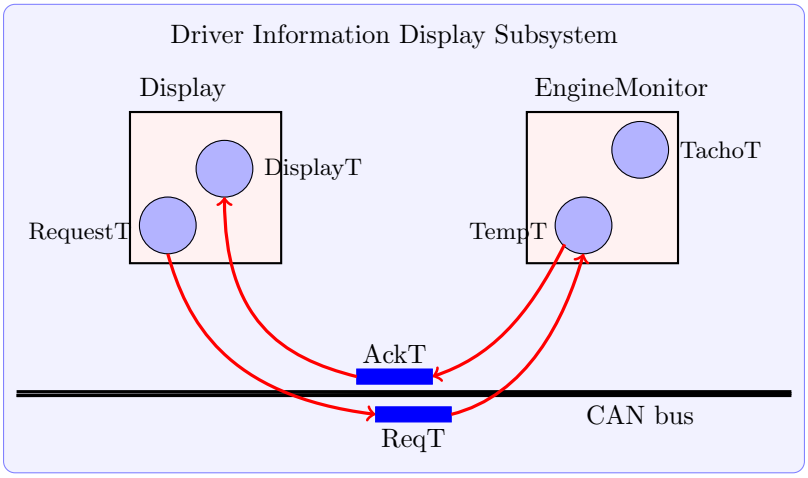
Learning Outcomes Assessed by This Work

- Assess the issues relating to software engineering development of embedded systems and apply their knowledge in the creation of such systems.
- Evaluate the scheduling and resource management requirements of real-time systems and make effective use of the associated algorithms.
- Assess the problems involved in developing distributed real-time systems and construct solutions to these problems

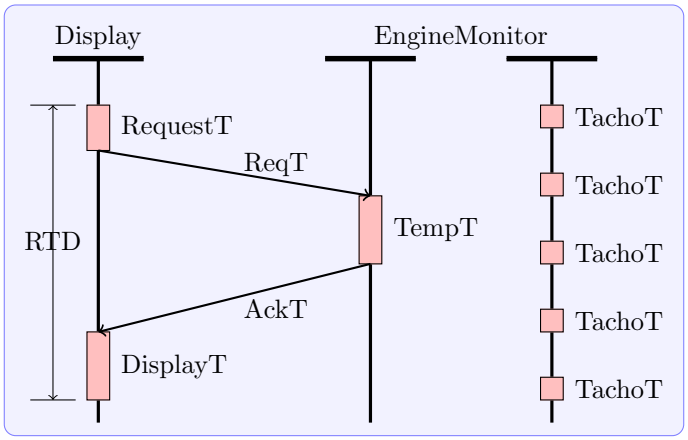
Academic Misconduct You may work on the **practical part** of this assessed exercise in groups of two or three. i.e., you can develop and test the small distributed system as a group and share your experimental results. However, the report you submit must be your own work - the description of your methods, any results obtained analytically, and the interpretation of the results must be your work alone - do not collaborate with your colleagues 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.

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 an event-driven system, implemented using a real-time operating system, uC/OS-II. The `Display` system is a time-triggered system. The task `Display.RequestT` periodically transmits a request message to the `EngineMonitor` processor. The `EngineMonitor.TempT` task is released by an interrupt handler that is triggered by a CAN controller on 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 runs at a higher priority than `TempT` and may be scheduled to run at the same time. The time-triggered task `Display.DisplayT` polls its CAN controller for the arrival of a message containing the temperature value and, when it receives a message, 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 Time-triggered task	500 ms
Display.DisplayT	Periodic Time-triggered task	20 ms
EngineMonitor.TempT	Periodic CAN interrupt posts OS_EVENT	Inherited
EngineMonitor.TachoT	Periodic uC/OS-II 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

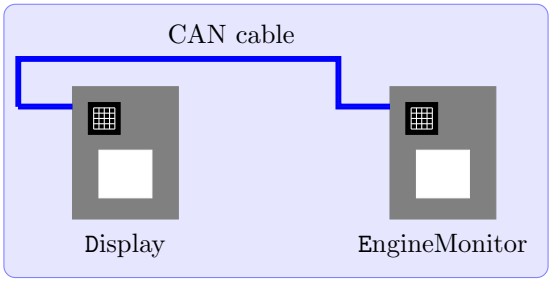
Your system is required to run on the ARM boards (LPC-2378STK) available in PB F1. You should begin your solution by downloading the file workspace.zip into a suitable directory. Unzip the file and open the workspace `workspace.eww`.

Projects are available for the two subsystems, as follows:

- Display
- EngineMonitor

The `Display` subsystem uses a time-triggered scheduler. The `EngineMonitor` subsystem uses uC/OS-II. The CAN driver is provided in `can.h` and `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. Write a short report (not more than 6 pages) that clearly addresses 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 accuracy 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 RTOS such as uC/OS-II affects the analysis of response times for this system and contrast this with the use of a time-triggered scheduler. 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)