Exam
Concurrent and Real-Time Programming

2011–10–22, 08.00–13.00

You are allowed to use the Java quick reference and a calculator.
Also dictionaries for English (and the native language for each student) is allowed.

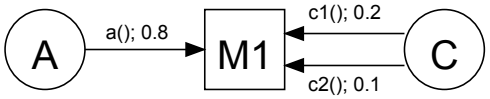
To pass the exam (grade 3), you have to solve most of the first 5 problems (which are more of a theoretical type), and you have to develop an acceptable solution to problem 6 (programming). At least a partial solution to problem 7 is required for the highest grade (5).

---

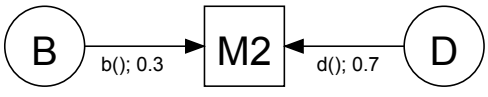
1. In a multi-threaded system there can be different types/levels of preemption. Tell with one sentence each what the following means, and for each one give one reason to use it and one reason not to use it.
   a) Non-preemptive scheduling
   b) Preemption-point
   c) Interrupt-driven preemption (natively between machine instructions according to compiler and hardware, as we normally assume)

   (3p)

2. A realtime system consists of four threads named A, B, C and D that communicate through two monitors M1 and M2. The graph below shows the monitor methods called from each thread, annotated with the worst case execution time per method. The table shows the period (T) and worst case execution time (C) for each thread. Calculate the worst case response times for the four threads. The system is scheduled using RMS and utilizes the basic inheritance protocol. All times are given in milliseconds.



	T	C
A	2	1
B	5	1
C	15	2
D	17	2



Recall:

$$R_i = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

(3p)

3. In Java (and most other modern languages such as C#) we have automatic memory management using a Garbage Collector (GC).

   a) Give two reasons why that is a good idea. *(1p)*

   b) In a real-time system with strict priorities and fixed-priority preemptive scheduling, what is the difficulty with having a GC in combination with hard real-time requirements? How is that solved in real-time systems/Java (concerning what thread performs the GC work and when). *(2p)*

4. A real-time system with strict priorities is to be scheduled with fixed-priority scheduling, with priorities assigned according to the deadline-monotonic principle. It is ensured (by detailed knowledge about the application, the run-time system and its hardware) that blocking times do not cause more than 4ms delay per thread. It is desirable that a set of (the highest priority) threads for feedback control has priorities assigned in the order A, B, C, and D. The table below gives partial information about the system:

Thread	C (ms)	T (ms)	D (ms)
A	2	9	6
B	2		
C	2		
D			

   a) Considering threads A, B and C only. What are the shortest possible deadlines and periods for B and C?

   (2p)

   b) Thread D should run with a lower priority than A-C, and it should run 10 times less frequent than thread A. What is the longest possible WCET (C) of thread D?

   (2p)

   Recall:

$$R_i = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

5. In a new robot project a mobile robot is assisting a traditional robot arm (non-moving and clamped to the floor). For safety reasons, the clamped robot is equipped with a safety switch connected to a laser-curtain (a wall of laser beams) surrounding the robot, henceforth referred to as the fence. Any object passing the fence puts the robot arm in standby mode, allowing both humans and mobile robots to move safely within the robot work area.

   The reset action to put the robot arm back to work is to push a button on the robot controller, which happens to be placed behind the fence. The reset button was originally installed for human safety purposes, but now it is also pushed by the mobile robot.

   During the first tests the mobile robot was programmed to move into the fenced area to reset the robot arm whenever it was put in standby mode and then move out of the area. However, what the programmer forgot was that the mobile robot (while passing the fence on its way out) put the robot arm back in standby mode. Since the robot arm need some preparation time before resuming work from standby, the situation ended up in the mobile robot moving in and out of the fenced area all the time while the robot arm is doing no useful work at all (being in standby or preparing to resume all the time). Hence there will never be any work carried out despite both robots doing things all the time. (The problem was later solved by moving the controller outside the robot's fenced workspace.)

a) In code, the following happens after ordering mob.rob to start the system (with '*' denoting passage of the fence):

```
mob.rob     safety    clamp.rob
=======     ======    ========
do {
approach
     *-----> detect ----> standby
pushOn ------------------> on
depart                     prep
     *-----> detect ----> standby
} while                  (!on)
```

What is the situation called when multiple activities go on (not being blocked) but without making progress?

(1p)

b) Now two clamped one-armed robots are installed side by side such that they can assist each other in certain operations on workpieces where two arms are needed. They also share tools (attached to their wrist by means of a tool exchanger).

From a work scheduling point of view (within so called manufacturing execution systems) both workpieces (W), robots (R) and tools (T) are resources that must be locked. However, if we consider the situation that robot R1 only works on workpiece W1 and robot R2 only works on workpiece W2, we can ignore the workpieces from a potential deadlock point of view.

Each robot arm is driven by a process, here called P1 and P2. When one robot arm wants assistance from the other in a dual-arm operation, the other robot is considered a resource and as such must be locked before use.

Consider the following programs:

```
P1                                  P2
------                              ------
takeR1 // me working                takeR2 // me working
  takeT1                              takeT2
   takeT2                             useT2
   useW1T1T2                          giveT2
   giveT2                             takeT1
   takeR2                             useT1
   useW1T1R2 // assisted              giveT1
   giveR2                           giveR2 // me done
  giveT1                            helpR1 // assisting
giveR1 // me done
```

Draw the resource allocation graph of the system, and determine if there is a risk for deadlock. In case of deadlock, give suggestion for how to change the processes to resolve the situation. Analyse and motivate why the performance of the two robots is good or bad in your final program. If it is bad, can you improve the performance by modifying the locking scheme in the processes further? Assume all useXX methods are independent of each other.

(4p)

6. Timed Monitor

Background

In the course we have relied on scheduling analysis to get an idea of worst case response times. Typically threads interact via shared resources (often protected by monitors). The response times then depend on the blocking times which in turn depend on the worst case execution times of the monitor methods. However, correctly estimating the WCET can be difficult, and it is therefore desirable to also supervise the running system to check that estimated WCETs and blocking times are not overrun. This problem is about supervising execution times for monitor methods, and taking actions when blocking times and WCETs (used in the static analysis of the system) are exceeded.

Introduction

To keep track of execution time on a per-thread basis, a method:

```
public static native long executionTimeMicros()
```

have been implemented in the class `se.lth.cs.realtime.RTSystem`. It returns the execution time for the current thread (starting at zero when the thread is started). Time is given in microseconds.

For reading the system time the standard `System.currentTimeMillis()` method call is available. Time is given in milliseconds. Time resolution is assumed to be 1 ms.

The WCETs and blocking times (to be compared to the execution time and system time) could be given using static code annotation or by other means. Here we assume it to be hardcoded in the code of each method.

Thread priorities can be read by the `int Thread.getPriority()` method call. Furthermore, the `static Thread Thread.currentThread()` method returns the currently executing thread as a Thread.

Problem

Consider a system using a single monitor M with three methods a, b and c, all presumably without arguments and return values (since that would not have any influence on our topic). For simplicity we assume all threads are of type Thread and that they are scheduled by a priority-based real-time scheduler (strict priorities, basic priority inheritance, etc.). The estimated WCETs for methods a, b and c are 2, 4 and 12 ms respectively. There is no indirect blocking in the system.

Only the c method uses `wait()`, and does so after half the WCET. Also, only the highest priority thread is to call method a; that is, it should be the same thread all the time and that thread should have the maximum priority that can be assigned. Method b contains an algorithm that should fulfill its WCET but the convergence of the iterations therein is not fully tested.

Develop the monitor such that the following specification is met:

1. Register the first caller of method a and thereafter any other calling thread should be thrown a RTError("Call rejected").
2. Always check that the caller of a is running with `MAX_PRIORITY`, and that the callers of b and c have `MAX_PRIORITY-1` or lower.
3. If the call of a returns later than its WCET or worst case response time, a RTError("Method a too late") should be thrown. It should be checked that the method is completed within its WCET, and also within the anticipated system time. Hint: what is the blocking time of a?
4. Methods b and c, in case of an exceeded WCET, should throw an error such as RTError("Method b too late");
5. The execution time spent in c on waiting (evaluating the condition and doing the call; the actual wait does not increase the execution counter) should be recorded separately. In case method c exceeds its WCET, the thrown error should include the waiting-delay in its message (".."+toString(delay)),
6. In case method b (which contains an iteration and is less tested) takes too long time, method a should report b:s execution time as possible blocking time in it's error message. Since b is a bit uncertain in the estimated WCET, the b execution time could prove useful in debugging a overruns.

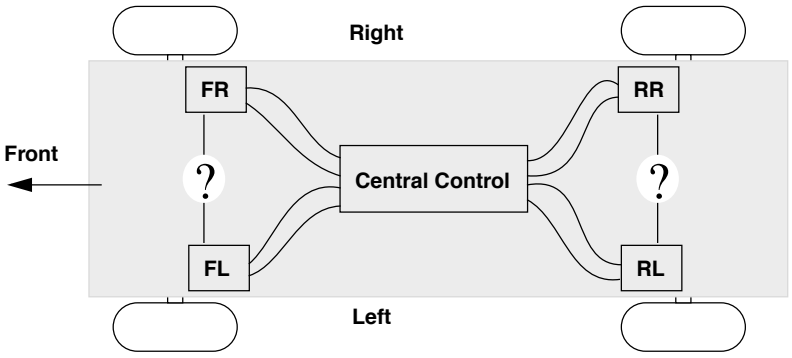
7. A max-priority supervisory thread wants to be notified when an overrun has occurred. Create a monitor method called `supervisor` that blocks until an overrun has been detected and then returns the error message as a string.
8. The supervisory thread also wants to be notified as soon as possible when b has caused an overrun, even if b has not yet returned. The expected worst case response time for b is 14 ms. Hint: use an extra monitor with a helper thread.

For grade 3 (a pass, not aiming at a higher grade) requirement 8 can be omitted.

(12p)

7. Brake-by-wire

Background: To reduce cost, there is an aim to use so called X-by-wire in modern vehicles, where X can be steer, fly, brake, etc. Here we study braking of vehicles such as cars or trucks. Consider a four-wheel truck with a central control computer and distributed processors for brake control, one per wheel. The available software for the central computer uses RTEventBuffer:s to send control orders to the wheels and to receive responses from them. Since it is crucial that the vehicle does not get unsafe in the case that one network cable is broken, we want to use redundant communication (double networking connections). Assume there is such redundant networking, accomplished by two network cables/wires between the central control and each of the brake control nodes (FR, FL, RR, and RL), as shown in this figure:



The lines/connections between the brake controllers for wheels on the same axis can be used in case both connections to the central computer is lost, for equal braking on both wheels.

Problem: Design the distributed brake-control systems such that:

- The central control software (run by the main thread of that computer) sends brake commands or setpoints to each of the brake controllers. You decide when and how that is done (e.g., periodically).
- Each brake controller performs a periodic control task. During that there could be hardware problems detected, which then should be reported to the central control.
- If any one cable between the central control and a brake control is broken, the vehicle should perform as if there where no fault, except that a warning is shown to the driver by (centrally) calling Panel.brakeWarning();
- Each brake controller needs to detect within 40ms if one or both of its connections to the central is broken. You may assume that all communication is predictable and data/objects is transferred within 2ms. Further assume that all networking methods are potentially blocking (in case of a read and no data is available).
- The central control should be kept aware of any broken cables. If messages can be received from a communication socket, it can be assumed that the wires are OK. Reading data can still be blocking without any exception if the connection is broken.
- A brake controller that has no connection with the central control should brake as much as the other brake on the same axis is braking. If there is any communication probelm with the other wheel on the same axis, that is equivalent with the case that one of the connections with the central control is broken.
- A totally disconnected brake controller should brake with half effort after 500ms. A re-established connection within that time is OK but should result in a brakeWarning.

Design the system in terms of threads and their interconnections. Show clearly in what (erroneous) states the different nodes can be. Solutions that are easy to see that they will work well/safe, are rewarded with higher credit points.

(6p)