LUND UNIVERSITY, Faculty of Engineering Department of Computer Science

Exam, EDAF85 Realtime Systems

2019–11–01, 14.00–19.00

You are allowed to use

- Java quick reference (Java snabbreferens)
- dictionary from English to your language
- calculator

Answers may be given in Swedish or English.

The exam has two parts:

- a theory part (problems 1–5, 14 points in total)
- a construction part (problems 6–7, 16 points in total).

The maximal score is 30 points. The preliminary requirement for grade 3 is 15 points, with at least 6 points from the theory part, and at least 8 points from the construction part.

Formulary

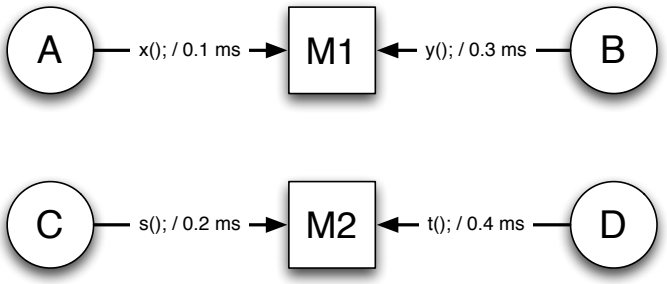
$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{1/n} - 1)$$

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

1. What is a context switch? Explain in one or a few sentences. *(1p)*

2. What do we mean with the term *race condition*? Explain in one or a few sentences. *(1p)*

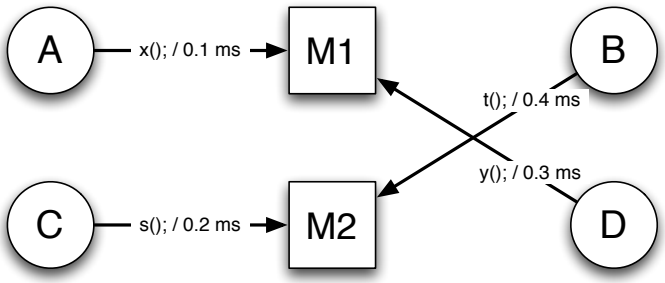
3. The four threads A, B, C, and D with highest priority in a real-time system communicate with each other using the monitors M1 and M2 according to the figure below:



The monitor operations `x`, `y`, `s`, and `t` are called by the threads (once each thread invocation) as illustrated in the figure. For each monitor operation, the maximum required execution time is shown in the figure. Thread A has the highest priority and D has the lowest priority among the three threads. Thread B has higher priority than thread C. The threads do not communicate or interact with lower priority threads other than as illustrated in the figure.

a) What are *the maximum blocking time* for each of the four threads above? *(2p)*

b) Imagine we modify the call graph according to the following figure:



What are the maximum blocking time for each thread now? *(2p)*

4. Consider a system consisting of three independent threads (i.e., no blocking) executing periodically with the following characteristics (C = worst-case execution time, D = deadline, T = period).

Thread	C (ms)	D (ms)	T (ms)
A	3	4	10
B	1	8	8
C	2	6	6

a) Is the system schedulable, i.e., will the system meet all of its deadlines, if we apply Rate Monotonic Scheduling? To get any points you must motivate your answer (a simple yes/no will not be enough). *(2p)*

b) What if we instead apply Deadline Monotonic Scheduling? Is the system schedulable in this case? To get any points you must again motivate your answer. *(2p)*

5. Jim Hacker has decided to try mutex semaphores. In his schematic program below, five resources are protected by mutex semphores (denoted A, B, C, D, E), and accessed by three threads (denoted R, S, T).

   a) Draw a resource allocation graph. Is there a risk for deadlock? Motivate your answer. *(2p)*

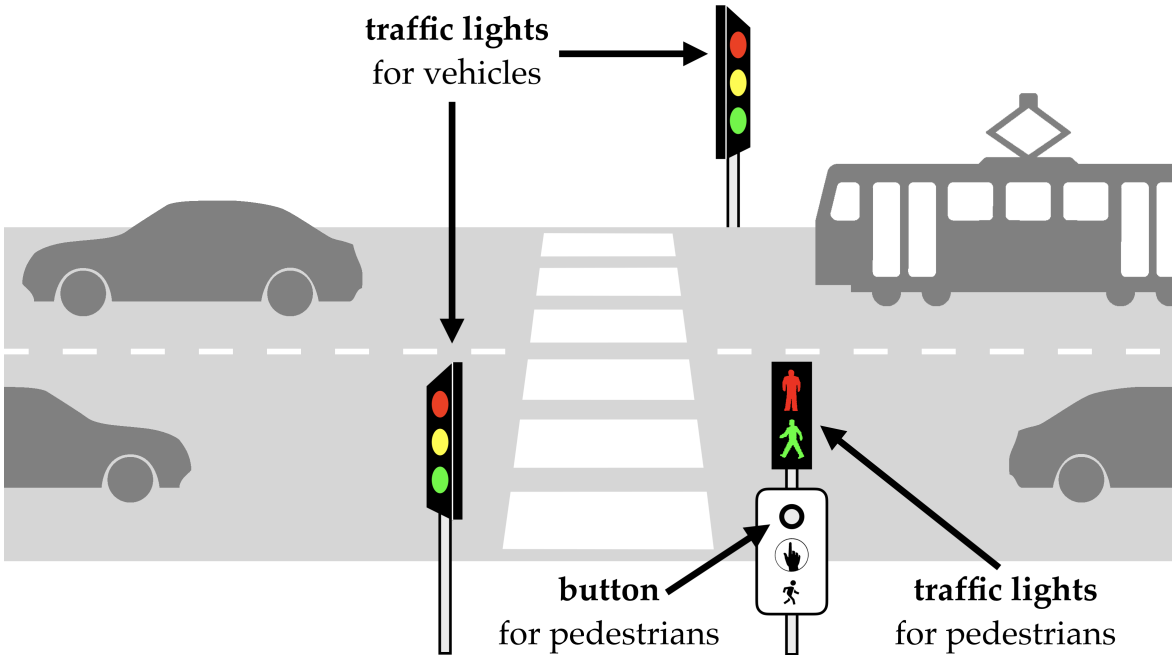
   b) Suggest changes in the allocation order to avoid deadlock. Note, however, that functions `useA()` and `useB()` take long time, so a thread calling them should not hold any unnecessary resource. *(2p)*

```
 1  THREAD T
 2  ========
 3  C.take();
 4  D.take();
 5  A.take();
 6  useCDA();
 7  A.give();
 8  D.give();
 9  C.give();
10
11  THREAD S
12  ========
13  B.take();
14  useB();
15  D.take();
16  useBD();
17  B.give();
18  E.take();
19  useDE();
20  E.give();
21  D.give();
22
23  THREAD R
24  ========
25  E.take();
26  D.take();
27  useED();
28  D.give();
29  E.give();
30  A.take();
31  useA();
32  B.take();
33  useAB();
34  B.give();
35  A.give();
```

6. Pedestrian crossing

In this task, you will extend an existing system for traffic lights in a pedestrian crossing. The crossing, sketched below, has vehicle traffic flowing in both directions. Pedestrians wishing to cross press a button, wait for green light, and then cross. After some time, the pedestrians' light turns red again, and vehicle traffic is allowed to resume.

The existing system is presented in section 6.1, and your task in section 6.2.



6.1 The existing system

The traffic lights are controlled using the `LightIO` class. This class includes access to the pedestrians' button, as well as a sensor for detecting trams and other vehicles.

```
public class LightIO {
  public static final int  RED = 1,  YELLOW = 2,  RED_AND_YELLOW = 3,  GREEN = 4;

  /**
   * Set traffic lights to value v for vehicles (RED, YELLOW, RED_AND_YELLOW, or GREEN),
   * and p for pedestrians (RED or GREEN).
   */
  public void lights(int v, int p)                                          { ... }

  /**
   * Blocks until the pedestrians' button is pressed. If the button was
   * pressed before awaitButton() was called, the method returns immediately.
   */
  public void awaitButton() throws InterruptedException                     { ... }

  /**
   * Blocks until an approaching vehicle is detected. If a vehicle passed before
   * awaitVehicle() was called, the method returns immediately.
   * Returns true if the detected vehicle is a tram, false otherwise.
   */
  public boolean awaitVehicle() throws InterruptedException                 { ... }
}
```

We also have a program `LightController`, shown below. It cycles through five steps:

S1. All lights (for both vehicles and pedestrians) are set to red.
S2. When a vehicle is detected, vehicles' lights turn red and yellow for 3 seconds.
S3. Vehicles' lights turn green for 120 seconds.
S4. Vehicles' lights then turn yellow for 5 seconds, and then red for another 5 seconds.
S5. Pedestrians' light turns green, and stays green for 30 seconds.
The program then starts over with step S1.

Note: this program is **flawed**, as we will soon see, as it ignores the pedestrians' button.

```
1  public class LightController {
2    public static void main(String[] args) throws InterruptedException {
3      LightIO io = new LightIO();
4
5      while (true) {
6        io.lights(LightIO.RED, LightIO.RED);              // S1
7
8        io.awaitVehicle();
9
10       io.lights(LightIO.RED_AND_YELLOW, LightIO.RED);   // S2
11       Thread.sleep(3000);
12
13       io.lights(LightIO.GREEN, LightIO.RED);            // S3
14       Thread.sleep(120000);
15
16       io.lights(LightIO.YELLOW, LightIO.RED);           // S4
17       Thread.sleep(5000);
18       io.lights(LightIO.RED, LightIO.RED);
19       Thread.sleep(5000);
20
21       io.lights(LightIO.RED, LightIO.GREEN);            // S5
22       Thread.sleep(30000);
23     }
24   }
25 }
```

6.2 Your task

After step S1, we wait for traffic to arrive before we continue. This means that pedestrians cannot (legally) cross an idle road – they must wait for some traffic to arrive first. This is clearly unpractical.

Moreover, to facilitate efficient public transport, we want the green time in step S3 (120 seconds) to be extended when trams are detected. A tram requires up to 30 seconds to pass the crossing.

Your task is to design and implement a better `LightController`, according to the following:

- After step S1, the program should wait for **either** vehicles **or** pedestrians to arrive.
  - If the pedestrians' button was pressed, step S5 is executed.
  - Otherwise, if vehicle traffic was detected, steps S2, S3' (see below), and S4 are executed.

  After selecting **one** of these alternatives, the program starts over in state S1.

- Step S3 is revised to prioritize trams, unless pedestrians want to pass:

  S3'. Vehicles' lights turn green for 120 seconds, and then stay green until **either**

  (a) the button has been pressed **or**
  (b) no tram has been detected in the last 30 seconds.

You will need to introduce new threads for detecting the independent inputs (button/traffic). The information from these threads should be shared with your updated `LightController` using a **monitor**.

The monitor should provide monitor operations for the following:

- Notify the monitor that a button has been pressed.
- Notify the monitor that traffic has been detected (tram or other traffic).
- Wait until either a button is pressed or traffic is detected and return information about which event occured. Prioritize pedestrians! Use this to implement the first change described above.
- Wait until it is time to leave state S3', i.e. until the button is pressed or 30 seconds has passed since the last tram was detected. Use this to modify state S3 according to the second change request.

Your solution should thus include:

- An updated `LightController` (or a clear and complete description of how it should be modified).
- New threads for handling button/traffic detection. Create and start the threads in `LightController`.
- A monitor according to the description above. Create the monitor object in `LightController`.

Hints:

- Use `System.currentTimeMillis()` to get the current time.
- Remember that there is a variant of `wait()` that times out by throwing an `InterruptedException` after a given number of milliseconds if `notify()`/`notifyAll()` is not called: `wait(long milliseconds)`.

(13p)

7. Temperature control

The problem is to implement the part of a control program responsible for the temperature control in an industrial process. The temperature can be increased by the use of a microwave source wheras it can be decreased only by natural cooling. The temperature should be kept within a given interval and the limits of the interval should not be exceeded (other than during startup). The following static methods are available for sampling and control:

```
public class ProcessControl {

  /**
   * Returns the current temperature (degrees Celsius).
   * @return the current temperature
  */
  public static double readTemp();

  /**
   * Switch the microwave source on or off.
   * @param on true if the microwave source should be switched on, false to switch off
  */
  public static void heating(boolean on);

  /**
   * Returns the target temperature.
   * @return the target temperature
  */
  public static double targetTemp();

}
```

Task

Implement a thread controlling the temperature. The temperature should not be allowed to get higher than the target temperature returned by `targetTemp()`, and it must neither be allowed to get lower than $5°C$ under the temperature returned by `targetTemp()`. When the heating is enabled, the temperature will increase by $5°C$ per second and when it is disabled the temperature will decrease by $1°C$ per second. Delays in the system and errors in the measurement causes a small error but we choose to ignore this aspect of the problem.

Your solution must be fairly efficient in the sense that the control thread should not execute significantly more often than is reasonable to keep the temperature within the limits. It is also desired to avoid switching the heating on and off much more often than necessary in order to avoid wearing down the mechanical relay controlling power to the microwave source. Make a reasonable compromise between sampling time and optimal usage of the available temperature interval. For maximum point for the problem you are expected to motivate why your solution fulfills the demands above. *(3p)*