

Exam — Concurrent and Real-Time Programming — English
2005-03-30, kl 8.00-13.00

Instructions

You are allowed to use the Java quick reference. To pass the exam, grade 3, you have to solve most of the first 8 problems (which are more of a theoretical type), and you have to develop an acceptable solution to problem 8 (which means programming). To get a higher grade, you also have to provide a sufficient solution to problem 9 (which means design). To get the highest grade you also have to have a sufficiently high total score (typically 30p). Results will be published on the internal part of the course web page (id=eda040, pwd=ht04).

Problems

- 1.** What is, and what is happening during, a context switch? (1p)

- 2.** In real-time systems we use priorities to obtain bounded response times for the important threads.
 - a)** Why should we not use priorities to implement critical sections (e.g. by temporarily assigning the highest priority among all involved threads during the execution of a critical section)? (1p)
 - b)** For development of large software systems it is a good principle to keep data/information locally as far as possible (encapsulation). This is taught already in introductory courses in programming. How does the use of priorities as the foundation for dividing the CPU time among threads correspond to this basic principle of encapsulation? (1p)

- 3.** We have discussed the concept *static cyclic scheduling*.
 - a)** How do you handle execution of code that does not fit within a single time slot and thus must be interrupted by other (faster) statically scheduled activities? (1p)
 - b)** A statically scheduled system can sometimes lead to low CPU utilization. Give one reason for this and suggest how you can avoid the problem? (1p)

- 4.** Modern processors utilize various techniques such as 'pipelining' and 'caching' (for data and instructions respectively) in order to increase performance, e.g. for desktop and server applications. In this case, performance means that the average case executes efficiently. What is the problem with this type of high-performance computers in real-time systems with hard real-time demands? (1p)

5. Consider a real-time system implemented by four threads, A, B, C, and D, with the properties according to the table below (C = maximum execution time/period and T = period):

<i>Thread</i>	<i>C (ms)</i>	<i>T (ms)</i>
A	4	20
B	5	32
C	3	12
D	2	8

The deadline for each thread is equal to the thread's period. Furthermore, we assume that the threads are scheduled according to the principle RMS (Rate Monotonic Scheduling) and that the threads are completely independent of each other, i.e. they do not communicate with each other and can thus not block each other (they can of course still preempt each other). We also ignore the cost of context switches etc. The computer system in question is a single-processor system, i.e. it has only one CPU.

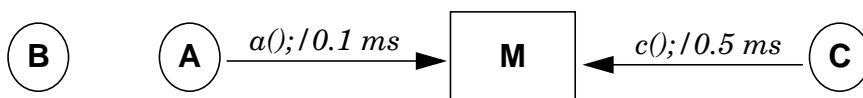
a) What is the worst-case response time for each of the four threads? (3p)

b) Assume we replace the computer with a double-processor computer, i.e. the computer has two CPUs, and assign thread D to processor number 1 and the other threads (A,B, and C) to processor number 2. The CPUs are assumed to execute completely independent of each other. From an analysis point of view, we thus get two separate systems to analyse - one consisting of just thread D and one consisting of threads A, B, and C. Show how we can in an easy way prove that all the four threads above will always meet their deadlines without computing/ knowing any worst-case response time for the individual threads. (*Hint: $5/32 < 0,157$*) (2p)

6. A real-time system is implemented using three periodic threads, A, B, and C. Thread A has the highest priority, B second-highest priority, and C the lowest priority. The worst-case execution time (C), period (T), and deadline (D) for each thread is given by the following table:

<i>Tråd</i>	<i>C (ms)</i>	<i>T (ms)</i>	<i>D (ms)</i>
A	2	40	40
B	3	50	50
C	4	60	60

Thread A communicates with thread C via a monitor called M. Each time A and C respectively executes, they call a monitor function (once and only once per period) according to the figure below. Thread B does not communicate with any other thread. The maximum execution times for the two different monitor functions are shown in the figure.



a) Assume the system is scheduled according to strict priority-based scheduling with *dynamic priority inheritance* (basic priority inheritance protocol). What will the *worst-case response times* be for the three threads? (2p)

b) Assume the system is still scheduled according to strict priority-based scheduling but *without any form of dynamic priority inheritance*. How long can in this case the three threads be blocked by lower-priority threads in the worst case, i.e. what is the *worst-case blocking times*? (2p)

7. A Java program contains three types of threads, T1, T2, and T3, whose respective `run()` method executes the following linear sequences of semaphore operations on the five mutex semaphores A, B, C, D, and E (intermediate code dependent on the semaphores is represented by function calls on the form “`useXY()`”, where “XY” denotes that the semaphores X and Y must be taken when the code is executed):

T1	T2	T3
<code>A.take();</code>	<code>D.take();</code>	<code>C.take();</code>
<code>E.take();</code>	<code>E.take();</code>	<code>D.take();</code>
<code>useAE();</code>	<code>useDE();</code>	<code>B.take();</code>
<code>E.give();</code>	<code>E.give();</code>	<code>useBCD();</code>
<code>A.give();</code>	<code>D.give();</code>	<code>B.give();</code>
<code>...</code>	<code>...</code>	<code>D.give();</code>
<code>E.take();</code>	<code>B.take();</code>	<code>C.give();</code>
<code>A.take();</code>	<code>A.take();</code>	
<code>useAE();</code>	<code>useAB();</code>	
<code>A.give();</code>	<code>A.give();</code>	
<code>E.give();</code>	<code>B.give();</code>	

a) Draw a resource allocation graph for the system. (1.5p)

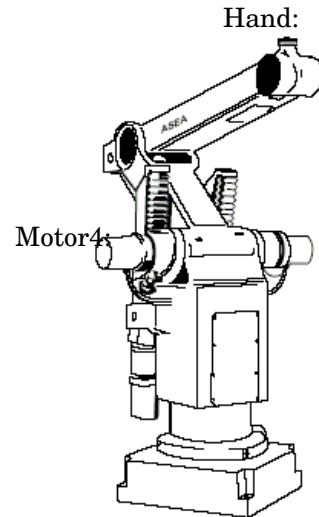
b) The program above can deadlock. How can you see this in the resource allocation graph? (1p)

c) Suggest a modification to the program such that the risk of deadlock is eliminated without jeopardizing the synchronization needs of the program. I.e., when a function on the form “`useXY()`” is called, the corresponding semaphores must still be taken. (0.5p)

8. Setpoints for motor control

In a robot arm there are a number of joints, where the motion of each joint is driven by a motor that is controlled such that it reaches the desired angle/position. Normally there is 5 or 6 such joints in one arm. As an example, the wrist angle (where Hand is mounted as depicted in the figure) is driven from Motor4 (see fig.).

For the hand to perform the desired motion, e.g. a straight line along an edge of a work-piece, reachable target positions are computed (in an available part of the control software) for each motor. These targets represent steps along the desired path, with a step size corresponding to 20ms in time. These steps (referred to as targets below) comprise setpoints that cannot be computed more frequently since every computed target value is based on extensive dynamic models and heavy computations. The feedback control of each motor, on the other hand, requires setpoints/references with a shorter period, in our case 4ms. A longer period would result in vibrations and jerky motions, so the requirements on control setpoints are



1. Every 20ms target step should be interpolated to smaller steps corresponding to 4ms.
2. The desired velocity setpoint should also be determined, corresponding to change of position per time unit.

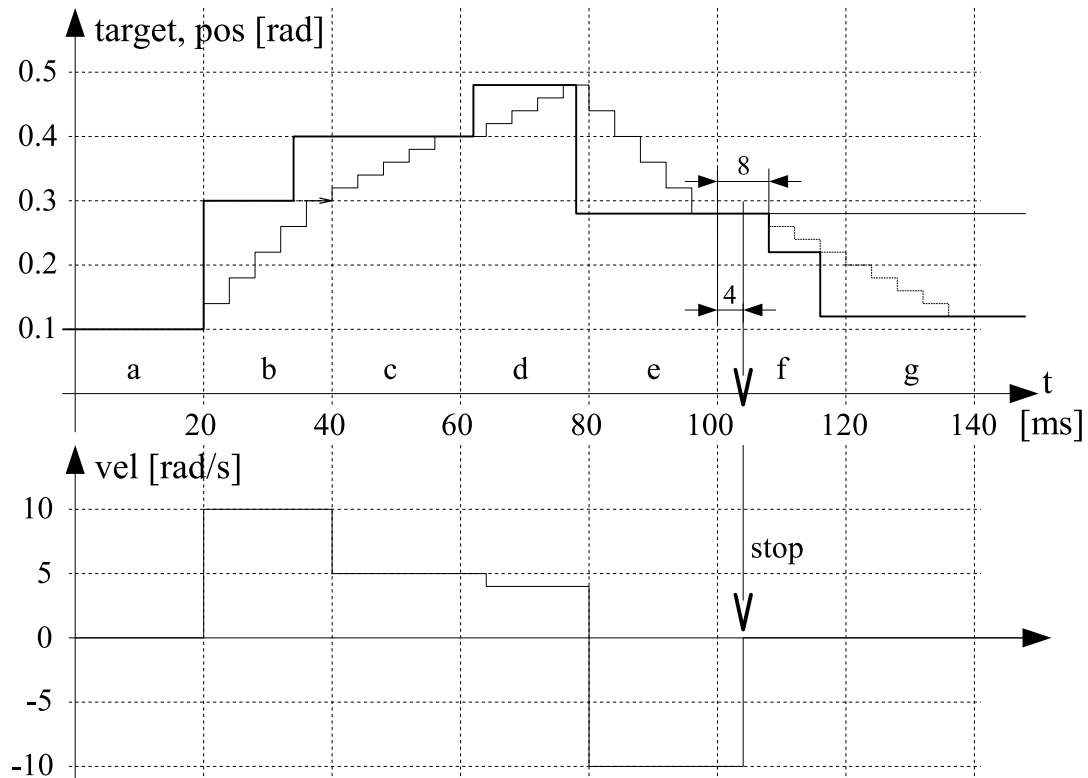
Here we limit the problem to one motor only, and we study only the quite limited part of the software that accomplishes the items 1 and 2 above. Since other parts of the system have been developed already, we also know that new target values are provided with the period of 20ms, *asynchronously* with the feedback control that is running with the period of 4ms. That means we should develop the connection between two activities/threads, which means that a thread-safe passive object should be implemented; our design is ready apart from some method types and arguments, and any additional classes you might need.

Thus, you should finish the design and implement a class that we can call `SetpointMonitor`.



If we assume we have an `SetpointMonitor` instance called `ipol`, we will by providing input target values obtain interpolated position values (`pos`), derived velocity values (`vel`), and a stop signal if any deadline should be missed. The following figure illustrates with an example how these values are to be computed, where we during the different periods a to have the following situations:

- During a, before 20ms, there is no motion commanded. The motor is at rest with the position/angle 0.1 rad.
- After 20ms a target value of 0.3 rad is set, along with a time value of 40ms, stating that the target should be reached after 40ms. When setting a target value, there is no knowledge about what period the interpolations will be done for (the 4ms in the example), but we know the output period is always shorter than that of the target values.
- During period b, while interpolated positions are obtained by the control thread with a period of 4ms, a new target 0.4 rad is set at time 34 but that is to be used from time 40.
- During period c the interpolation continues. The next target value, to be used from time 60 for reaching the position at time 80, is expected but arrives to late.



- Interval d starts without a new target been set (to be reached at 80). We do tolerate some delay (max 4ms) so the current values of pos and vel are kept during the start of the new period. Considering the definition of the velocity, the vel value should at this time go to zero (since the position is constant), but here the vel is kept constant. That is to avoid jerky motions and vibrations in case the new target is set a few ms later. This is what happens at time 62 when the next target value is provided, and interpolation continues towards position 0.48, with an updated vel value from time 64.
- At time 78 the new target to be reached at time 100 is provided. The new value is smaller so the derivative from time 80 gets less than zero.
- At time 108 the next target that should have been used already from time 100 is supplied. That is too late, since the maximum permitted delay is 4ms. Therefore, at time 104, when the controller thread obtains its next position reference, the missed deadline is detected and the `ipol` object enters a so called stop or emergency state. This means that the desired position is held constant and the desired velocity is zero.

If there would have been no missed deadline at 104, the interpolation would have continued along the dotted line. In conclusion, your task is to develop the missing code of the following class.

```
public class SetpointMonitor {

    /**
     * Maximum permitted delay of new target position. If setTarget
     * is called more than maxDelay after the time when the target
     * should be reached, this is an emergency and the motion should
     * be stopped.
     */
    public static final long maxDelay = 4;
```

```

// Your private attributes go here.....

/**
 * Provide a new target position (motor angle) and the time when that
 * position should be reached (in terms of the setpoint to the motion
 * control as returned by getPos). After the setting of one target (by
 * calling this method) and the target time, the next target (with a
 * new time) has to be set (again, by calling this method) within the
 * current target time plus the maxDelay value (in ms). Failure to
 * meet this deadline (such as the 8ms being larger than the 4ms in
 * the figure/graph of the problem description) will result in a stop
 * of the motion (equivalent to calling emergency with a true argument)
 * and false is returned from this call.
 *
 * If a motion is known in advance, the thread computing the target
 * positions (along the desired path) can compute the next target and
 * call this method in advance. However, only one extra/pending target
 * can be provided; additional calls of this method will block until
 * any pending target position has been taken care of (by the feedback
 * control calling getPos up to the previous target time).
 *
 * @param youDecide what arguments you want to have.....
 * @return the success of the setting of the new target. That is, targets
 *         provided too late or during stop/emergency results in false.
 */

public /* you choice */ setTarget(What youDecide) {
    // This is for you to implement.
}

/**
 * Obtain the commanded position to be applied at time t. The calling
 * thread (which typically is the one with the highest priority in
 * the system) is responsible for calling this method frequently
 * enough and before time t is passed. That is, the timing of the
 * thread performing feedback control is checked elsewhere, and this
 * method simply computes the desired position by interpolating
 * between the target positions. Furthermore, if the feedback control
 * would advance slower in time than expected (supplied argument t for
 * each call increases less than the real time), the calls of setTarget
 * would block longer and delay the setting of new targets, but the
 * obtained position would still be valid for the time supplied.
 *
 * @param t the time for which the computed position should apply.
 * @return the computed position to be used as the control setpoint.
 */

public /* your choice */ double getPos(long t) {
    // You also write this method
}

/**
 * Get desired velocity, determined as change of desired position
 * per time unit, which in turn is the change of target position
 * per time unit. If the target position is updated late, but before
 * stop is commanded, the obtained velocity remains unchanged (see
 * time 60-62 in the figure/graph of the problem description).
 *
 * @return the desired speed in motor-radians per second.
 */

public /* your choice */ double getVel() {
    // Another method to develop for you...
}

```

```

/**
 * Emergency stop of motion, or reset of emergency state to continue
 * after a previous stop. This method is available to all parts of
 * the system (e.g. to the operator interface) but may also be
 * called internally if deadlines are missed.
 *
 * @param on If true, stop motion by setting target position to
 *           current position, and notify any caller of awaitStop.
 *           If false, block until the velocity of the commanded
 *           motion is zero, and then permit new motions.
 */
public /* your choice */ void emergency(boolean on) {
    // Another method to develop for you...
}

/**
 * Block caller until there is an emergency stop, due to a call of
 * the emergency method and/or a missed deadline. To avoid excessive
 * context switches, considering that unblocking should be unusual,
 * this method is not synchronized. Instead, it is made thread safe
 * by synchronizing on a an internal object specifically maintained
 * for stopping only.
 */
public void awaitStop() {
    // The last one to develop...
}

// Private stuff, classes and methods, whatever you decide...
}

```

Thus, complete the implementation of the class `SetpointMonitor` such that the properties according to the comments are accomplished. If additional classes are needed, they can be put as inner classes or as external additional classes, at your own desire.

(9p)

9. Tunnel surveillance

A design (no implementation) of a surveillance system for a car-traffic tunnel needs to be carried out, to enable planning of a possible implementation and to be able to decide if such a project is reasonable. A corresponding product, which possibly can be extended by means of an available SDK, is shown on <http://www.autoscope.com> where the information to the right has been extracted. Explain by figures and text what threads, resources, communication, etc. that are needed for a system that fulfils these specifications:

1. A tunnel with bidirectional car traffic should be supervised, at both ends of the tunnel.
2. At both ends of the tunnel there should be a camera server that should do the following:
 - a) Send images by 1Hz to a central operator interface/computer.
 - b) By 5Hz detect if there is any motion in the images.
 - c) In case of motion, capture 25Hz image sequences of 1 second.
 - d) For each captures image sequence, identify car numbers and for each detected vehicle compute speed and direction.
 - e) For every vehicle, which may have been identified in several sequences, its number together with speed and direction should be sent to a central computer.
 - f) If a vehicle or motion is detected but no (license-plate) number, the image sequence is sent to the central operator.
3. Communication between different computers supports sending complete objects. All readings from the network is blocking.
4. The operator computer is located centrally; not at the tunnel.
5. An operator interface shows updated single images with the frequency 1Hz (idle), arrived image sequences, and detected problematic cars according to next item.
6. All received car data (including place a direction) is sent to a separate thread in the operator interface software. This thread eliminates vehicles that have passes through the tunnel normally. Remaining vehicles are classified as: Speeder, U-turner, Lost, Appeared. This is logged to file and also shown in the user interface.

Carry out the design of the system, but do not do any implementation. Algorithms are assumed to be available in existing classes (that you can name). (9p)

Autoscope Video Detection (product example)

In recent years, a number of aboveground technologies have emerged to complement or replace inground inductive loops, which are expensive to reconfigure, have limited capabilities, and fail frequently. These new technologies include video detection, radar, ultrasonic, infrared and laser. Of these, video detection has been the most successful, providing unsurpassed richness of data as well as video images, wider coverage areas and greater versatility for demanding applications. To determine the best technology for your applications, ask yourself these questions:

- Do you want wide area detection? Accuracy in measuring vehicle counts and speed?
- Do you want to detect stopped vehicles? Congestion? Vehicles going the wrong way?
- Do you want to be able to reconfigure the detectors easily during road construction or to reflect changes in road geometry?
- Do you want "snapshots" of traffic?

Video detection is the only technology that can provide all these functions and benefits. Video detection is now more cost-effective, accurate and reliable than ever, outperforming inground inductive loops and other aboveground detection technologies under all weather and light conditions. The Autoscope system is the leader in video vehicle detection.

Tunnels

Tunnels are safer than ever, thanks to video-based incident detection and traffic management. Autoscope systems can quickly signal an alarm when incidents are detected, enabling fast response by an operator. Autoscope systems are proving their unique value to this challenging environment by detecting: Stopped vehicles, Slow traffic, slow vehicles, Wrong way vehicles, etc.

Autoscope is being utilized with SCADA (Supervisory Control And Data Acquisition) for incident management in the Clyde Tunnel. In 1992, local authorities in Glasgow decided to adopt the new European tunnel guidelines for a highly driven tunnel and included Autoscope in their refurbishing program.

[www.autoscope.com]