Embedded Control Systems

Exam 2017-06-01

14:00 – 18.00: M Building

Course code: SSY190

Teachers: Knut Åkesson

The teacher will visit examination halls twice to answer questions. This will be done approximately one hour after the examination started and one hour before it ends.

The exam comprises 22 credits. At least 10 credits are needed for passing the written exam. The final grade is set by the rules published at the course syllabus.

Solutions and answers should be complete, written in English and be unambiguously and well motivated. In the case of ambiguously formulated exam questions, the suggested solution with possible assumptions must be motivated. The examiner retains the right to accept or decline the rationality of assumptions and motivations.

Exam results will be reported in Ladok. *The results* are open for review 2017-06-21 and 2017-08-30 between 12:30-13:30 at the department.

No aids are allowed on the written exam except:

- Pen and a rubber
- Standard pocket calculator (no hand computer). Erased memory.
- Essential C, Nick Parlante. No comments are allowed in the report.
- Dictionary from/to your native language to/from English

1

a) Explain the difference between casual and acasual modelling of dynamic systems.

(1p)

b) Describe how Zeno-behavior might occur and how it might affect the simulation of hybrid systems.

(1p)

c) Describe the difference between a safety specification and a liveness specification.

(1p)

d) Describe the four necessary conditions for deadlocks to occur i multi-task programs.

(2p)

2

The concurrent execution of task T_A and task T_B produces an infinite sequence of $\langle A \rangle$ and $\langle B \rangle$. Three semaphores are used to coordinate the tasks,

1. `mutex` that is initialized to 1,
2. `times_a` that is initialized to either 0, 1, or 2,
3. `times_b` that is initialized to either 0, 1, or 2.

Psuedo code for the two tasks are given below.

Task T_A:

```
for(;;)
{
    take(times_a);
    take(mutex);
    print("A");
    give(mutex);
    give(times_b);
}
```

Task T_B:

```
for(;;)
{
    take(times_b);
    take(mutex);
    print("B");
    give(mutex);
    give(times_a);
}
```

Determine for each start of sequences below if it can be printed by executing the tasks above. If a sequence can be generated also indicate to what values `times_a` and `times_b` have to be initialized to for the sequence to be generated. If it cannot be generated for any initialization explain why.

a) AABAABAAB, ... (1p)

b) ABAABABAA, ... (1p)

c) ABABABABA, ... (1p)

d) AABBABBAA, ... (1p)

3

A robot has been designed with three different tasks T_A, T_B, T_C, with increasing priority. The task T_A is a low priority thread which implements a DC-motor controller, the task T_B periodically send a "ping" through a wireless network card so that it is possible to know if the system is running. Finally the task T_C, with highest priority, is responsible to check the status of the data bus between two I/O ports. The control task is at low priority since the robot is moving very slowly in a cluttered environment. Since the data bus is a shared resource there is a semaphore (mutex) that regulates the access to the bus. The tasks have the following characteristics

Task name	Period	Execution time
T_A	9	3
T_B	6	3
T_C	2	0.1

Assuming the kernel can handle preemption, analyze the following possible working condition:

1. at time $t = 0$, the task T_A is running and acquires the bus in order to send a new control input to the DC-motors,
2. at time $t = 2$ the task T_C needs to access the bus meanwhile the control task T_A is setting the new control signal,
3. at the same time, $t = 2$, the task T_B is ready to be executed to send the "ping" signal.

a) Show graphically which tasks are running. What happens to the high priority task T_C? Compute the response time of T_C in this situation. (2p)

b) Suggest a possible way to overcome the problem in a). Compute the response time of T_C in this situation. (2p)

4

When scheduling among controller processes it is often the case that the relative deadline (D) is less than the period (T). Standard rate-monotonic scheduling assume that the relative deadline is equal to the period. In deadline montonic is is the relative deadline of the task instead of the period of the task that determine the priority. In deadline monotonic scheduling the shorter relative deadline the higher priority of the task. Consider the following task set. C is the worst case execution time.

Task	T (ms)	C (ms)
T_A	30	8
T_B	40	8
T_C	60	20

a) Assume that the deadlines are equal to the period for each task. Determine by using the Liu-Layland critera (see formula sheet) if the deadlines are guaranteed to be meet.

(2p)

b) Assume that $D_A = 30$, $D_B = 20$, and $D_C = 55$. Determine if it is possible to meet all deadlines with priorities set using the deadline monotonic principle for the task set above.

(3p)

5

Consider a system with two tasks, T_1 and T_2, and a single shared resource. Both tasks perform different activities but every now and then wants to use the shared resource. We have the following atomic propositions defined, $i \in \{1, 2\}$.

- T_i.request
- T_i.use
- T_i.release

Specify in Linear Temporal Logic (LTL) the properties below. The allowed temporal operators are Always (G), Eventually (F), Until (U), and Next (X).

a) Mutual exclusion, i.e., only one task at a time can use the resource.

(1p)

b) Finite time of usage, i.e., a task can only use the resource for a finite amount of time.

(1p)

c) Alternating access, i.e., tasks must strictly alternate in using the resource.

(2p)

Good Luck!

Formula sheet

Liu-Layland schedulability test

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

Response-time analysis

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

1/

a/ In a casual modelling languages inputs and outputs are explicitly exressed. For example, a resistance can be modelled as

$$u(t) = R \cdot i(t)$$

where $i(t)$ is the input and $u(t)$ is the output

For example in Simulink we might have a block

However, if we would like $u(t)$ to be the input and $i(t)$ the output we will have another block

.

In a casual modelling languages like Modelica/Simscape you only specify the relations between the variables (i.e. the equation).

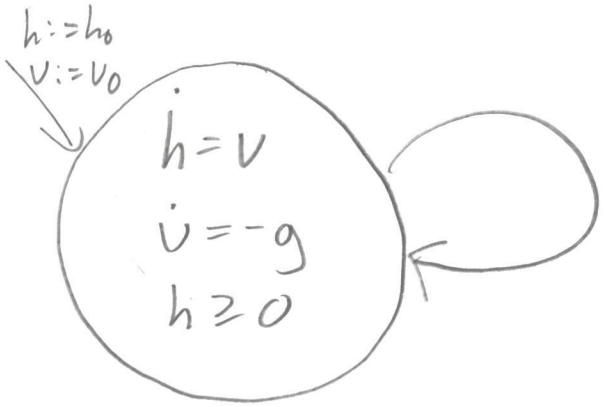
The casuality is then automatically derived from how the components are used.

This increases the reuse of the components.

1b/

Zeno-behavior occurs when the system has an infitine number of switches (state jumps) in finite time.

An example is an ideal model of a bouncing ball



$h=0 \rightarrow v := -a \cdot v \quad 0<a\leq 1$

In this model the bounces will come become more and more frequent, and as a consequence this model is not defined beyound a certain time. This is in itself a problem when simulating a system containing Zeno-behaviors because you might want to simulate other systems beyond the time for which the Zeno-model is defined.

In addition, when simulating a dynamic system a non-zero step size is always used and you might experience numerical problems identifing the exact time at which the switch should take place. In the bouncing ball example it might result in that the height at a bounce is negative and never become positive and then the simulation will never bounce again and it will look like the ball fell through the floor.

1c/

A safety specification : Nothing bad ever happens.

Violations of safety specifications are demonstrated by a finite execution (counter example)

A liveness specification express that something good should eventually happen.

Counter examples are cycles that might happen and in which something good never happens.

1d/

- Mutual exclusion (Shared resources are only used by a task at a time).
- Hold and wait (Hold some resource while waiting for others)
- No preemption (A resource can only be released by the task holding it)
- Circular wait (Circular chain where each process in the chain is waiting for a resource currently held by another process in the chain).

2/

a/ AABAABAAB

b/ ADAABABAA

c/ ADABABABAB

d/ AABBABBAA

Semaphores times_a and times_b are initialized to 0, 1, or 2.

- If both are initialized to 0 both tasks will block and we will have a deadlock.
- If times_a is initialized to 1 and times_b 0, task A might print A and then it will block and wait for task B to print a B. Task B has to wait for task A to write an A. Thus with this initialization we will print sequence C.
- If times_a is initialized to 0 and times_b to 1 the system will print BABABA...
- If times_a and times_b are both initialized to 1, we can start by printing either a A or a B, but then the other task has to print a charcter before. But after both tasks have printed a character the process starts over again and either of the two tasks can print the next character. Possible sequences:
  - ABADAB
  - ABBAAB

  However, note after an even number of characters there will always be the same number of A and B's
  Thus it might print sequence d/ but not a/ and b/. However it does not necessarily print sequence d/.

2 cont/

- If times_a are initialized to 2 and times_b to 1 we can print one extra A at any time but then we are back to the previous case since each of the semaphores are only increased after printing a character.

Sequence A,B contains 6 A's and 3 B's.

At any point in time we can have at most two more A's than B's printed. Thus, sequence c,d/ but not a) and b/ might be printed. However, neiter c or d are always printed.

- If times_a are initialized to 1 and times_b to 2 we have symmetric situation to the previous case.
Sequence c/ might be printed but not a, b or d.
None of the sequences are necessarily printed.

- If times_a and times_b are both initialized to two.
Neither of the sequences has to be printed but sequence c/ and d/ might be printed out

3/ a)

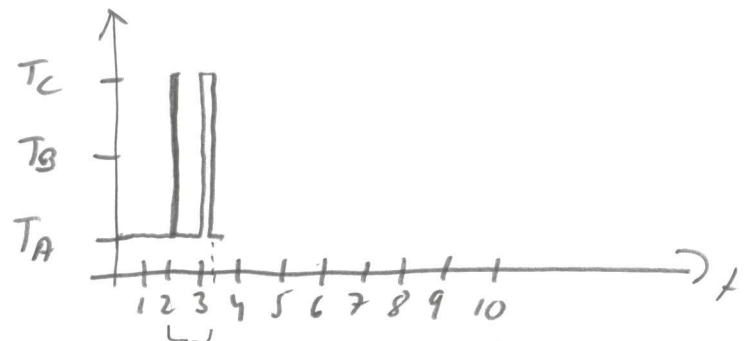
	Period	Execution time	
T_A	9	3	low prio
T_B	6	3	medium prio
T_C	2	0,1	high prio

T_A is released at 0
T_B and T_C is released at 2.

response time for task A = 4,1 seconds

This problem is called priority inversion, which occur when a low priority process has locked a shared resource and then a high priority process that would like to access the shared resource might be delayed by a completely independent medium priority process.

b/ Use the priority inheritance protocol. The low priority process will inherit the priority of the highest process that is waiting for access to the resource hold by the low priority process.



responsetime 1.1 seconds.

4)

a) Utilization is given by

$$\frac{8}{30} + \frac{8}{40} + \frac{20}{60} = 0.8 > 3(2^{1/3} - 1) \approx 0.78$$

Thus Liu-Layland may not guarantee that the system is schedulable.

b) Deadline monotonic principle $\Rightarrow$

T_B highest prio $(40, 8)$
T_A medium prio $(30, 8)$
T_C lowest prio $(60, 20)$

Use response time analysis to determine if the deadlines are possible to meet.

Start with highest priority process (T_B)

$R_B = C_B = 8 < 20$ (deadline meet)

Then compute response time for medium priority process (T_A)

$$R_A^1 = 8 + \left\lceil \frac{8}{40} \right\rceil \cdot 8 = 16$$

$$R_A^2 = 8 + \left\lceil \frac{16}{40} \right\rceil \cdot 8 = 16$$

Response time $T_A = 16$ seconds < 30 seconds, deadline meet

Now compute response time for lowest priority process

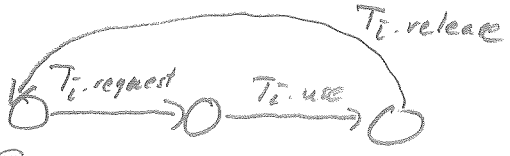
$$R_C^1 = 20 + \left\lceil \frac{20}{30} \right\rceil \cdot 8 + \left\lceil \frac{20}{60} \right\rceil \cdot 8 = 36$$

$$R_C^2 = 20 + \left\lceil \frac{36}{30} \right\rceil \cdot 8 + \left\lceil \frac{36}{40} \right\rceil \cdot 8 = 44$$

$$R_C^3 = 20 + \left\lceil \frac{44}{30} \right\rceil 8 + \left\lceil \frac{44}{40} \right\rceil \cdot 8 = 54 \Rightarrow R_C^4 = 20 + \left\lceil \frac{54}{30} \right\rceil \cdot 8 + \left\lceil \frac{54}{40} \right\rceil \cdot 8 = 54$$

$R_C = 54$ which is less than 55, thus deadline meet

5)



a) $G(\neg(T_1.use \wedge T_2.use)) \equiv Spec_A$

b) $GF(\neg T_1.use) \wedge GF(\neg T_2.use) \equiv Spec_B$

c)

$$G(T_1.release \longrightarrow (\neg T_1.use \cup T_2.use))$$

$$\wedge\ G(T_2.release \longrightarrow (\neg T_2.use \cup T_1.use))$$

$$\wedge\ Spec_A \wedge Spec_B$$