EXAM
(Tentamen)

Observe Question 3 has been updated. This is marked in green.

TDDI11
Embedded Software

2022-06-03

On-call (jour):
Ahmed Rezine 013-28 19 38.

General instructions:

- The questions are **not ordered** according to difficulty.
- You may answer in either English or Swedish.
- It is recommended to read all questions completely before you begin.
- Use a new sheet for each question and use only one side.
- Before you hand in, order the sheets according to the questions, number each sheet, and fill in your AID-number, date and course code at the top of the page.
- Write clearly. **Unreadable text will be ignored**.
- Be precise in your statements.
- **Clearly motivate** all statements and reasoning.
- **Explain** calculations and solution procedures.
- If in doubt about the question, write down your interpretation and assumptions.
- Grading: U, 3, 4, 5. The **preliminary** grading thresholds for p points are:

$0 \leq p < 20$:	U
$20 \leq p \leq 30$:	3
$31 \leq p \leq 35$:	4
$36 \leq p \leq 40$:	5

Question 1. (10 points)
Use the answer sheet at the end of the exam. No motivation or explanation is required for this 10-points question. **Zero or more statements may be correct for each sub-question. Tick each statement if and only if it is correct**. **Ticking a wrong statement or missing to tick a correct statement gives 0 points for that sub-question**.

1a) Memory-mapped I/O communication with peripherals

1. Can be used in polling-based I/O communication (i.e., Programmed I/O).
2. Uses usual memory assembly instructions to communicate with peripherals.
3. Uses special I/O instructions to communicate with peripherals.

1b) An I2C bus:

1. Is a serial bus.
2. Can make use of parity bits.
3. Can only be used between two components.

1c) A soft-real-time embedded system

1. Uses software instead of hardware as in hard-real-time systems.
2. May be correct even if it misses some deadlines.
3. Is not a real time system.

1d) What will be the output from the following C program?

```
int main() {
      int a[4];
      for(int i=0; i < 4; ++i)
            a[i]= 2*i;
      printf("%d \n", *(a + *(a + 1)));
}
```

1. 2
2. 4
3. 8

1e) What will be the output from the following C program?

```
int main() {
      printf("%d \n", (1 + 2) || 4);
}
```

1. 0
2. 1
3. 7

1f) What will be the output from the following C program?

```
int main() {
      printf("%d \n", (1 + 2) | 4);
}
```

1. 0
2. 1
3. 7

1g) The background loop (or super-loop) in a foreground/background model

1. Is an infinite loop that calls the different modules of the system.
2. Cannot be used in the presence of interrupts.
3. Should not include, for a given iteration, more than one call to a given module.

1h) Using a programming model based on finite state machines to capture embedded programs:

1. Is recommended because it encourages the programmer to systematically think about the possible transitions.
2. Cannot be done with a sequential programming language like C and is therefore never used in practice.
3. Requires a programming environment that supports drawing circles for states and arrows for transitions.

1i) Using a message passing approach to concurrency:

1. Can be used even for processes running on the same machine.
2. Can still result in deadlocks.
3. Avoids simultaneous writes to the same variable.

1j) Concurrent engineering:

1. Makes several teams compete to deliver the best product.
2. Builds cross-functional teams.
3. Eases information sharing.

Question 2. (5 points)
The multi-tasking based approach and the foreground/background based approach are two different approaches to concurrency. Explain the two approaches and give an advantage and a disadvantage of each one of them.

Question 3. (5 points)
You are asked to add labels to the transitions of the Mealy machine given in Figure 1 at the end of the exam according to the following description. Read all paragraphs in this Question 3 before attempting to complete the Mealy machine. The Mealy machine is used to detect deadlocks involving two tasks `T1` and `T2`. You can think of the Mealy machine as a finite state machine describing a program running in a `monitor` task that executes concurrently with `T1` and `T2`. You are not asked to describe the tasks `T1` and `T2`. They may run different code, may have nested loops, if-statements and I/O instructions.

The tasks `T1` and `T2` share two resources: `Ra` and `Rb`. The tasks need exclusive accesses to `Ra` and `Rb`. The `monitor`-task monitors acquisitions and releases of resources `Ra` and `Rb` by the tasks `T1` and `T2`. The `monitor`-task does not influence the behaviors of `T1` and `T2`, it only "follows" the calls to acquire and release the resources to check if a deadlock occurred. A deadlock occurs when both tasks try each to acquire a resource already acquired by the other task. The Mealy machine has an input alphabet Σ and an output alphabet Γ, where:

- $\Sigma = \{\text{acq}_1(a), \text{acq}_1(b), \text{acq}_2(a), \text{acq}_2(b), rel_1(a), rel_1(b), rel_2(a), rel_2(b)\}$, with $\text{acq}_1(a)$ stands for `T1` acquiring resource `Ra` and $rel_2(b)$ stands for `T2` releasing resource `Rb`, and
- $\Gamma = \{\text{ok}, \text{dk}\}$: ok stands for "ok behavior" and dk stands for "detected deadlock".

A task needs to acquire a resource before it can use it. More specifically, `T1` needs to call the method `T1.acq(Ra)` to acquire resource `Ra` (in this case, the input $\text{acq}_1(a)$ is sent to the Mealy machine at the beginning of the call to the `acq` method). If the resource is available (i.e., not used by another task), the method `T1.acq(Ra)` succeeds and `T1` is considered to have the resource. At this point, if `T2` wants to acquire `Ra`, it will call `T2.acq(Ra)` (hence sending the input $\text{acq}_2(a)$ to the Mealy machine). In this scenario, the method `T2.acq(Ra)`would block (i.e., place `T2` in a FIFO queue associated to `Ra` and "hang" waiting for the resource `Ra` to be released). The resource `Ra` is released when `T1` calls `T1.rel(Ra)` (sending the input $rel_1(a)$ to the Mealy machine). After the release, the oldest request for the resource returns (in this scenario giving `T2` access to `Ra`). We say that `acq` is blocking (since it will block until the resource is acquired to the calling task) and that `rel` is not blocking. The inputs in Σ are generated in a similar manner for the other resource. We assume tasks do not try to acquire a resource if they already have it, and that they do not try to release a resource if they do not have it. You do not need to add transitions for these situations. Both resources are initially available.

Intuitively, the Mealy machine is meant to monitor any such tasks `T1` and `T2` and output "ok" as long as it did not witness a deadlock. As soon as it witnesses a deadlock (even after an arbitrary number of "ok" steps), the Mealy machine should output "*dk*" for deadlock and get to a state that does not have any outgoing transitions[1]. In Figure 1, there are 4 such states that correspond to detected deadlocks. The machine has a unique initial state and may change state only when one of the two tasks tries to acquire or to release a resource. The machine does not change state when the tasks execute other instructions. Again, you should not care about the codes of tasks `T1` and `T2`, just about possible input sequences over Σ and corresponding sequences of outputs over Γ.
Your Mealy machine should capture possible sequences over Σ that could be generated by arbitrary tasks `T1` and `T2` satisfying these descriptions. Recall the tasks do not try to acquire a resource if they already have it, and do not try to release a resource if they do not have it. Observe the sequences can be arbitrary long. You should decorate all transitions. The Machine should be deterministic. It is ok if the machine misses some sequences. Possible runs of your solution (need to be captured by your machine):

Input sequence	Output sequence
$\text{acq}_2(b)\ rel_2(b)\ \text{acq}_2(b)\ \text{acq}_2(a)\ rel_2(a)\ rel_2(b)\ldots$	ok ok ok ok ok ok. . .
$\text{acq}_1(b)\ rel_1(b)\ \text{acq}_2(a)\ \text{acq}_1(b)\ rel_2(a)\ rel_1(b)\ldots$	ok ok ok ok ok ok. . .
$\text{acq}_1(a)\ rel_1(a)\ \text{acq}_1(a)\ \text{acq}_2(b)\ \text{acq}_2(a)\ \text{acq}_1(b)\ldots$	ok ok ok ok ok dk. . .

[1]In a real system, the deadlock would be handled by restarting some of the tasks, but this is not considered here.

Question 4. (5 points)
Assume an embedded system where an "unsigned int" with 32 bits $b_{31}...b_2b_1b_0$ is used to represent the hours, minutes and seconds: the first 6 bits $b_5b_4b_3b_2b_1b_0$ are used for the seconds (in 0-59), the 6 bits $b_{11}b_{10}...b_7b_6$ for the minutes (in 0-59) and bits $b_{16}b_{15}...b_{13}b_{12}$ for the hours (in 0-23). Write a C function:

```
unsigned int tick (unsigned int current)
```

that returns an unsigned short that captures adding one second to the input "current". Of course, when the number of seconds is 59, then adding one second results in 0 seconds and one more minutes. Adding one minute when the number of minutes is 59 results in 0 minutes and one more hour. Adding one hour when the number of hours is 23 results in 0 hours.

Question 5. (5 points)

Explain why maintainability and upgrades are simpler for interrupt based I/O compared to programmed I/O.

Question 6. (10 points)

Consider a task set with three periodic tasks: task T1 with execution time 1 and period 3; task T2 with execution time 2 and period 12; and task T3 with execution time 4 and period 8. All three tasks are ready to run from the beginning and are supposed to run on the same single processor using some scheduling algorithm.

1. Give the processor utilization ratio in case the tasks are scheduled. (1pt)
2. Can the tasks be scheduled using Preemptive RMS? Explain with **a diagram using 24 time-units from 0 (inclusive) to 24 (exclusive)** (3pts)
3. Can the tasks be scheduled using Non-Preemptive Earliest Deadline First (EDF)? Explain with **a diagram using 24 time-units from 0 (inclusive) to 24 (exclusive)** (3pts)
4. Can the tasks be scheduled using Preemptive Earliest Deadline First (EDF)? Explain with **a diagram using 24 time-units from 0 (inclusive) to 24 (exclusive)** (3pts)

Answer sheet for question 1. Please hand this paper in together with the answers for the other questions (numbered and with AID number).

1a)	( ) 1	( ) 2	( ) 3
1b)	( ) 1	( ) 2	( ) 3
1c)	( ) 1	( ) 2	( ) 3
1d)	( ) 1	( ) 2	( ) 3
1e)	( ) 1	( ) 2	( ) 3
1f)	( ) 1	( ) 2	( ) 3
1g)	( ) 1	( ) 2	( ) 3
1h)	( ) 1	( ) 2	( ) 3
1i)	( ) 1	( ) 2	( ) 3
1j)	( ) 1	( ) 2	( ) 3

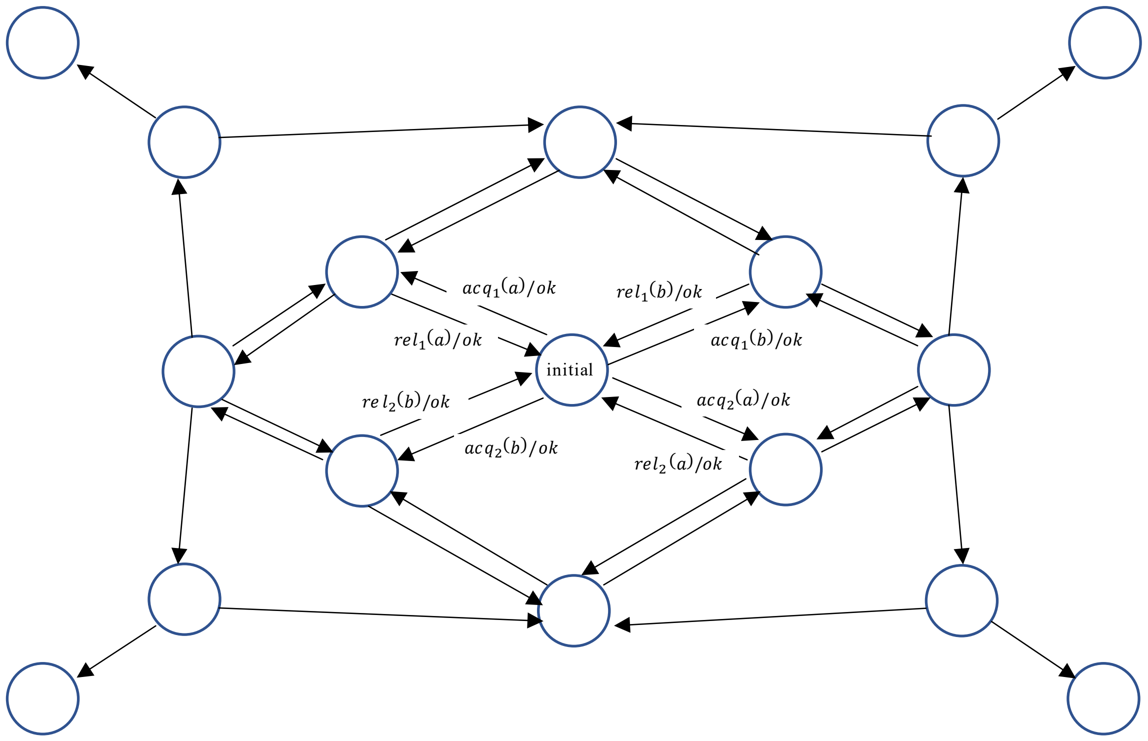

Figure 1. Sketch of a Mealy Machine for Question 6.