EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Mathematics and Computer Science

Written examination Real-time software engineering (2IN70) - Part II: Real-time Architectures on Wednesday, January 20th 2016, 13:30h-16:30h.

Hand-in the answers to this 2nd part of the examination on separate sheets of paper, and make sure to put your name and identification number on the paper.

Read all the questions first. There are 4 questions in total. Points for each question are indicated between parentheses and sum up to 6 points. Many questions have a word limit, which must be observed. Good luck!

1. Imagine a real-time application consisting of three non-preemptive periodic tasks. The following table provides the characteristics of the task set $\mathcal{T}$.

	T	D	C
`Task1()`	10ms	3ms	2ms
`Task2()`	10ms	5ms	3ms
`Task3()`	20ms	15ms	5ms

(a) (0.5) Which cyclic executive approach is appropriate for multiplexing these tasks, in order for all tasks to meet their deadlines? (max 10 words)

Answer: *Multi-rate periodic.* Note that *multi-rate time-driven AFAP* will not work, because with worst-case (rather than *fixed*) computation times, task `Task1` may be activated with an inter-arrival time that is *shorter* than 10ms, i.e. when $C_1+C_2+C_3 < 10$.

(b) (0.5) Provide the C code for the `main` loop of the selected cyclic executive for the task set $\mathcal{T}$, making sure that all tasks meet their deadline. You can assume that (*i*) the hardware timer is set to expire periodically every 10ms and (*ii*) you can wait for the timer expiration using the `wai` assembly instruction.

Answer:

```
while (1) {
    asm wai;
    Task1();
    Task2();
    Task3();
    asm wai;
    Task1();
    Task2();
}
```

(c) (0.5) Name two general shortcomings of the selected cyclic executive approach (max 30 words).

Answer: Any two of the following:

- Large release jitter.
- All tasks must complete before next timer expiration.
- Careful calibration may be needed to make sure tasks complete before next timer expiration, e.g. splitting tasks is sub-tasks.
- Polling for events is the only option (interrupt handling is not supported).
- Non-preemptive execution, may cause lower schedulability.
- Timer must fire at the greatest common divisor of task periods.
- All tasks are treated alike, i.e. there is no way to associate either a priority or an importance level to a task.

2. Fixed-priority pre-emptive scheduling is the de facto standard used in industry.

(a) (0.5) Describe the concept of preemption (max 50 words).

Answer: When during the execution of a task a higher priority task arrives, e.g. due to an external or timer interrupt, the higher priority task is allowed to run. The state of the currently executing task is stored to allow resumption at a later time.

(b) (0.5) Explain when preemption is needed (max 20 words).

Answer: When the non-pre-emptive execution of a lower priority task causes a higher priority task to miss its deadline. A special case is when the worst-case computation time of the lowest priority task cannot be bounded.

(c) (0.5) Describe how preemption can be supported by an operating system (max 30 words).

Answer: Two main aspects (1 is sufficient):

- Interrupt handling (external and timer interrupts activate tasks).
- The *state* of a task needs to be stored upon pre-emption and later restored upon resumption.

Note: several aspects of an operating system are not specific to pre-emption, e.g. a *ready-queue* or *priority*.

3. Different car functions will monitor different sensors to perform their work.

   (a) (0.5) What are the advantages of interrupt-based sensing compared to polling-based sensing? Name at least one (max 20 words).

   Answer: There are two main aspects: (*i*) reduced worst-case latency between an external state change and the detection by the system and (*ii*) improved efficiency, i.e. the overhead of polling is removed.

   (b) (0.5) Sometimes it may be necessary to guarantee mutually exclusive access to a resource, e.g. if a preemption would corrupt an ongoing analog-to-digital conversion of a sensor reading. Which mutual exclusion primitive and resource access protocol should be used in order to (*i*) avoid deadlock, (*ii*) avoid "unbounded" priority inversions, and (*iii*) avoid blocking high priority tasks not using the resource?

   Answer: SRP based mutex. Mutex only yields half of the points.

   (c) (0.5) Study the excerpt from a program in appendix A. Is a deadlock between these tasks possible? Motivate your answer (max 30 words). *Note*: no points are given for a correct answer with a wrong motivation.

   Answer: No, because there are no *nested* mutexes.

   (d) (0.5) Why should critical sections implemented by disabling the scheduler be as short as possible? (max 20 words)

   Answer: Because disabling the scheduler will cause blocking of all tasks with a higher priority than the task executing a critical section.

4. Imagine three ECUs communicating via a CAN bus attempt to send messages m_1, m_2, and m_3 at the same time. The following table shows the binary representation of the identifiers and the data of the three messages.

	Identifier	Data
m_1	01100101100	000000111111000000
m_2	01100100100	011111000011110000
m_3	10001000100	011111111100000000

   (a) (0.5) Assuming 0 is the dominant bit, which message will be sent? (The bits are read from left to right)

   Answer: m_2.

   (b) (0.5) What is the exact bit sequence that will be written to the CAN bus for transmitting the data of message m_2?

   Answer: $011111\underline{\mathbf{0}}0000\underline{\mathbf{1}}1111\underline{\mathbf{0}}0000\underline{\mathbf{1}}$

A Code listing for question 3c

```
...

OS_EVENT* Mutex1;
OS_EVENT* Mutex2;
OS_EVENT* Mutex3;

void Task1(void) {
  SetLed(LED_D26, ON);
  OSMutexPend(Mutex1, 0, 0);
  SetLed(LED_D26, OFF);
  OSMutexPost(Mutex1);
  ToggleLed(LED_D27, ON);
  OSMutexPend(Mutex2, 0, 0);
  SetLed(LED_D26, ON);
  OSMutexPost(Mutex2);
  SetLed(LED_D26, ON);
}

void Task2(void) {
  OSMutexPend(Mutex2, 0, 0);
  ToggleLed(LED_D27, ON);
  OSMutexPost(Mutex2);
  ToggleLed(LED_D27, ON);
  OSMutexPend(Mutex3, 0, 0);
  SetLed(LED_D27, OFF);
  OSMutexPost(Mutex3);
}

void Task3(void) {
  OSMutexPend(Mutex3, 0, 0);
  SetLed(LED_D24, OFF);
  OSMutexPost(Mutex3);
  ToggleLed(LED_D24, ON);
  OSMutexPend(Mutex1, 0, 0);
  ToggleLed(LED_D24, ON);
  OSMutexPost(Mutex1);
}

...
```