ELEC-E8001 Embedded Real-Time Systems

EXAM 14.12.2018 **MODEL ANSWERS**

---

1. Consider the following techniques that may be used for improving the performance of CPUs:

A. Increase the CPU clock frequency
B. Use a superscalar architecture
C. Use a two-level data cache
D. Use a Very Long Instruction Word (VLIW) architecture
E. Use a wider address bus
F. Use a wider data bus
G. Use an instruction cache
H. Use an instruction pipeline
I. Use the enhanced von Neumann architecture instead of the Princeton architecture
J. Use the Harvard architecture instead of the traditional von Neumann architecture

Which of these are *well-behaving* since they do not lead to increasing uncertainty (i.e., random variation) in real-time system's response times? (6 p)

A, **D**, **E**, **F**, **I**, **J** are well-behaving.

+1 p for each correct answer
–1.5 p for each incorrect one (B, C, G, H)
Maximum 6 p and minimum 0 p

Somewhat similar to **Homework #6A.**

2. A motor control system running on a single-core CPU has five parallel tasks with the following specifications (p_i = *execution period* and e_i = *execution time*; i = 1, 2, 3, 4, 5):

CAN Communications	p_1 = 100 ms	e_1 = 5 ms
Maintenance Tool	p_2 = 20 ms	e_2 = 2 ms
Self-Diagnostics	p_3 = 500 ms	e_3 = 25 ms
Torque Control Loop	p_4 = ?	e_4 = 0.05 ms
Velocity Control Loop	p_5 = 1 ms	e_5 = 0.1 ms

a) How shall the execution period of the Torque Control Loop (= p_4) be chosen to have a 95% CPU utilization factor? (4 p)

b) How would you assign the priorities for these tasks according to the Rate-Monotonic principle? (2 p)

-----

a)

U = 5 ms/100 ms + 2 ms/20 ms + 25 ms/500 ms + 0.05 ms/p_4 + 0.1 ms/1 ms = 0.95

U = 0.3 + 0.05 ms/p_4 = 0.95

p_4 = **0.077 ms**

Similar to **Homework #2B.**

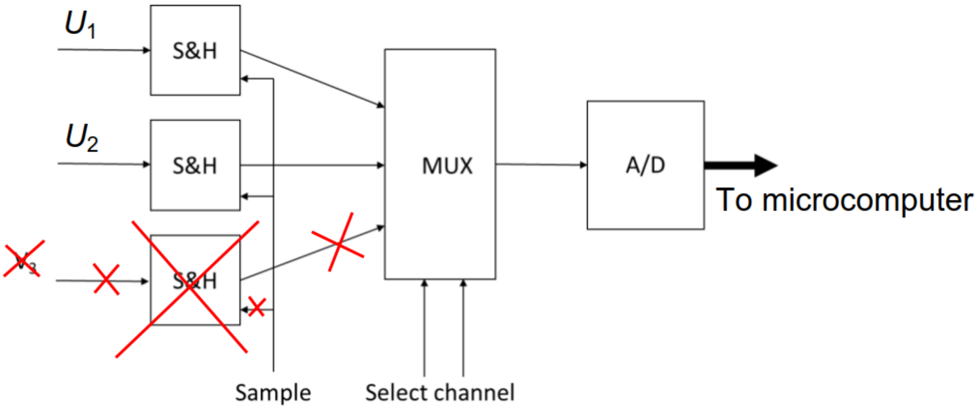
b)

1. Torque Control Loop (Highest priority)
2. Velocity Control Loop
3. Maintenance Tool
4. CAN Communications
5. Self-Diagnostics (Lowest priority)

Textbook pp. 102–104.

3. You have available one 12-bit analog-to-digital (A/D) converter, one four-channel analog multiplexer (MUX), and a few sample-and-hold (S&H) circuits. In addition, your 16-bit microcomputer has several free I/O lines that could be used, for instance, for controlling the MUX.

How could you implement a real-time measurement system with these components, which could perform the measurement of two analog voltages *exactly at the same time instant*? Assumption: the two voltages, U_1 and U_2, are readily scaled to correspond to the input range of the MUX and A/D converter. Draw a block diagram of your measurement system and explain its operating sequence. (6 p)



It is necessary to use individual S&H circuits in the two measurement channels that require simultaneous sampling. The microcomputer gives a concurrent “Sample” command through one of its output lines to these S&H circuits that memorize their analog inputs for a short period of time. After this, the S&H outputs are converted sequentially to the digital form by the A/D converter. This is accomplished by connecting one S&H output at a time to the A/D converter by selecting the desired channel of the MUX (two output lines “Select channel” are needed for this). Although the digital samples become available one after another, they still correspond to the same sampling instant.

Highly similar to **Homework #7B.**

4. *Priority inversion* may occur in a multitasking real-time system under certain conditions.

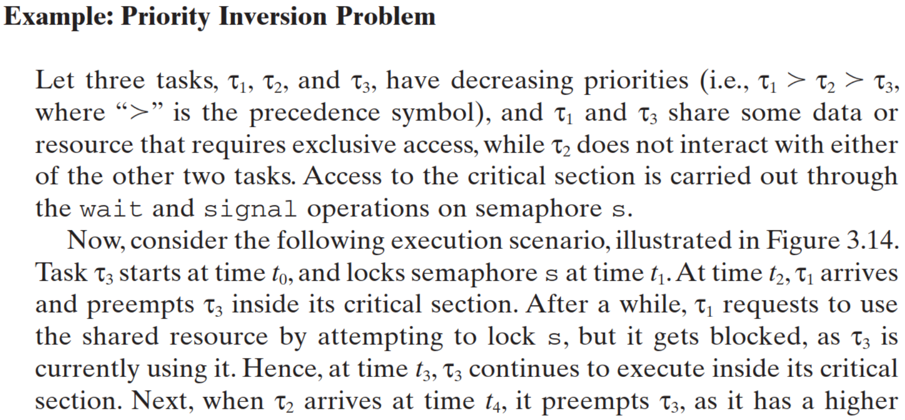
a) Why is it harmful? (1 p)

b) Show with an execution scenario how the priority inversion occurs in a three-task system under the control of an RTOS with preemptive-priority scheduling. (3 p)

c) What is the common solution to such a priority inversion problem, and how would it work in your scenario? (2 p)

-----

b)



Example: Priority Inversion Problem

Let three tasks, τ_1, τ_2, and τ_3, have decreasing priorities (i.e., $\tau_1 > \tau_2 > \tau_3$, where “>” is the precedence symbol), and τ_1 and τ_3 share some data or resource that requires exclusive access, while τ_2 does not interact with either of the other two tasks. Access to the critical section is carried out through the `wait` and `signal` operations on semaphore `s`.

Now, consider the following execution scenario, illustrated in Figure 3.14. Task τ_3 starts at time t_0, and locks semaphore `s` at time t_1. At time t_2, τ_1 arrives and preempts τ_3 inside its critical section. After a while, τ_1 requests to use the shared resource by attempting to lock `s`, but it gets blocked, as τ_3 is currently using it. Hence, at time t_3, τ_3 continues to execute inside its critical section. Next, when τ_2 arrives at time t_4, it preempts τ_3, as it has a higher

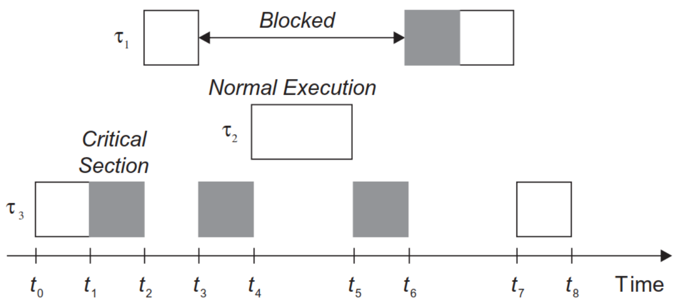
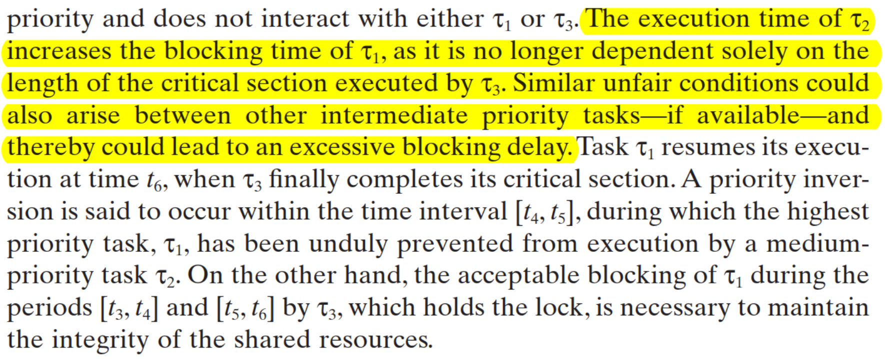
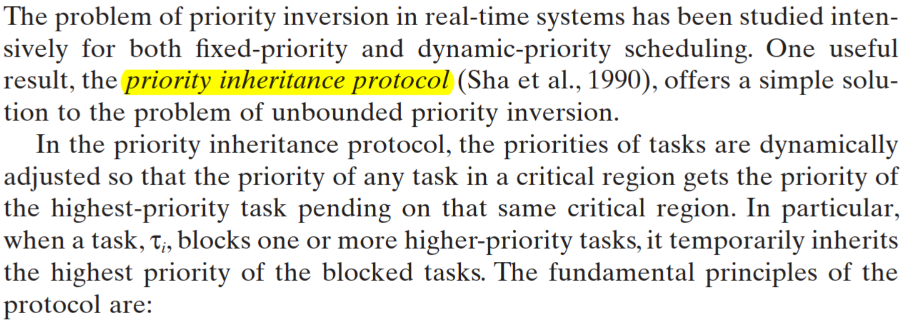


Figure 3.14. A typical priority-inversion scenario.



priority and does not interact with either τ_1 or τ_3. The execution time of τ_2 increases the blocking time of τ_1, as it is no longer dependent solely on the
a) length of the critical section executed by τ_3. Similar unfair conditions could also arise between other intermediate priority tasks—if available—and thereby could lead to an excessive blocking delay. Task τ_1 resumes its execution at time t_6, when τ_3 finally completes its critical section. A priority inversion is said to occur within the time interval $[t_4, t_5]$, during which the highest priority task, τ_1, has been unduly prevented from execution by a medium-priority task τ_2. On the other hand, the acceptable blocking of τ_1 during the periods $[t_3, t_4]$ and $[t_5, t_6]$ by τ_3, which holds the lock, is necessary to maintain the integrity of the shared resources.



The problem of priority inversion in real-time systems has been studied intensively for both fixed-priority and dynamic-priority scheduling. One useful
c) result, the *priority inheritance protocol* (Sha et al., 1990), offers a simple solution to the problem of unbounded priority inversion.

In the priority inheritance protocol, the priorities of tasks are dynamically adjusted so that the priority of any task in a critical region gets the priority of the highest-priority task pending on that same critical region. In particular, when a task, τ_i, blocks one or more higher-priority tasks, it temporarily inherits the highest priority of the blocked tasks. The fundamental principles of the protocol are:

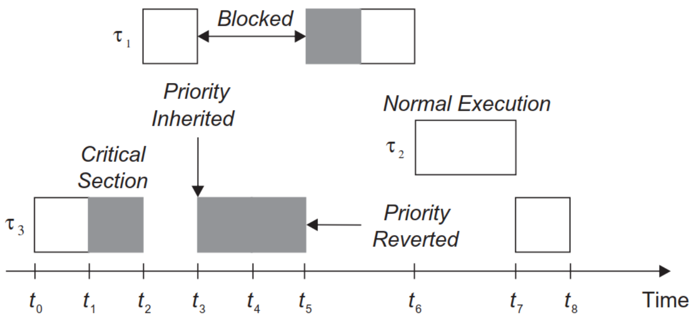
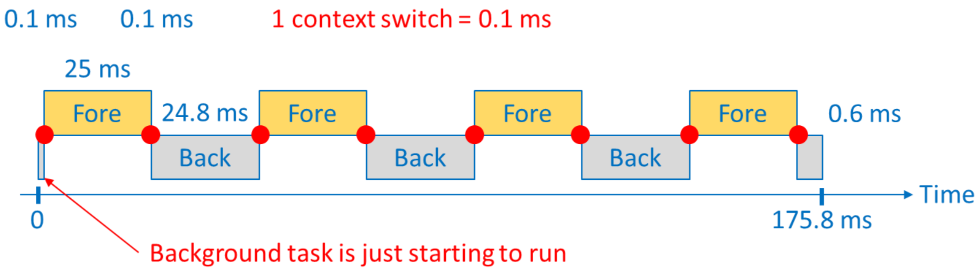


Figure 3.15. Illustration of the priority-inheritance protocol.

Textbook example pp. 118–120.

5. What is the *worst case* response time for the background task in a simple foreground/background system in which the background task requires 75 ms to complete, the foreground task executes every 50 ms (periodical timer interrupt) and requires 25 ms to complete, and each context switching takes 0.1 ms. Illustrate the execution sequence by a timeline. (6 p)



The worst case is shown above: The background task is just starting to run, but it is preempted by the foreground task that executes to completion in 25 ms, then the background task executes 24.8 ms because it is preempted again due to the 50 ms execution period of the foreground task (0.1 ms + 25 ms + 0.1 ms + 24.8 ms = 50 ms), etc. Hence, the worst case response time is 4 * 25 ms + 3 * 24.8 ms + 0.6 ms + 8 * 0.1 ms = **175.8 ms**.

Highly similar to **Homework #11A.**