

ITEE



What's New:

There are fewer questions this time, to allow more time to work through each question thoroughly.

Learning objectives for this week:

- Understand the major concepts we've covered, including:
 - what deadlocks are
 - how they are caused
 - how they are dealt with
- Apply these concepts to a specific situation, to tell whether a deadlock has or could occur
- Apply these concepts to choosing an appropriate strategy for a given situation
- Understand where and how deadlocks are likely to occur, and their significance in real systems

As before, you should aim to answer all of these questions by the end of the course. Being able to answer them by the end of week 7 is a useful goal. Be prepared: if you know what you can't do yourself, you will make most efficient use of the tutorial.

1. Concepts

- a. Why is each of the 4 conditions for deadlock *necessary* before a deadlock could occur? For the non-mathematicians, a *necessary* condition is one which must hold before something occurs, but the fact that the condition holds doesn't mean it *will* happen.
- b. Show that circular wait could not occur unless hold and wait had occurred.
- c. Explain why it is useful to have both conditions, even though circular wait implies hold and wait.

2. Safety

- a. Work through the example (slides 27–29) of the Banker's algorithm in the notes (slides 24–26) showing the values of each data structure at each step, showing that:
 - i. the sequence $\langle P_1, P_3, P_4, P_2, P_0 \rangle$ satisfies the safety criteria
 - ii. if the request (1,0,2) is made by P_1 , the sequence $\langle P_1, P_3, P_4, P_2, P_0 \rangle$ satisfies the safety criteria
- b. In general terms, is checking for safety a viable strategy? When might you use this strategy?
- c. How much simpler is the case where there is only one of each resource type? Would that scenario change your answer to (b)?

3. Detection

- a. Single instance of each resource type:
 - i. Why is it not generally practical to check for deadlock every time a resource is requested, using the approach for a single resource type covered in lectures?
 - ii. Would it be practical to check for a deadlock periodically? Explain difficulties which could arise.
- b. Multiple instances of each resource type:
 - i. Work through the given example of the detection algorithm, writing out contents of all the data structures at each step, for the example of slides 36–37.
 - ii. Based on this example, how easy is it to back out of a deadlock?
- c. You are writing some code using a database in which records can be locked. Your code has to be able to roll back to a safe state if a deadlock is detected by the system. Outline steps you would take in designing your program to make it possible to undo calculations. You need to consider points at which a resource (in this case, a database record) is requested, and how to record what you've done since the last successful request of a resource. Assume that once you've released a resource you will not have to roll back. The algorithm without handling deadlocks looks like this:

```
lock customer record // nothing else locked at this point
authorize credit card payment
if authorization failed
    unlock customer record
    report failure
    exit
lock flight record
book seat
unlock flight record
debit credit card
unlock customer record
```

4. Think of any situation (real-world or computer) where deadlocks could occur

- a. Show that the 4 conditions required for a deadlock all hold.
- b. How would you prevent the deadlock by changing the rules which lead to it?
- c. If instead you choose to do deadlock detection, will rolling back work?