

CS162 – Section # 5, 02/25/2003

Rodrigo Fonseca

Announcements

Look at clarifications to the boat problem, in the newsgroup

Next class: Chapter 6, CPU Scheduling

Project 1 code due 1 week from tomorrow

Don't forget about the midterm (3/13)

Deadlock

Chapter 8

4 conditions for deadlock:

- mutual exclusion
- hold and wait
- no preemption
- circularity in resource-allocation graph

Approaches:

- prevention
 - infinite resources, no sharing, no waiting, preemptible resources
 - request all before starting
 - banker's algorithm (see below)
 - request-in-order
- cure
 - determine that the system is deadlocked and terminate one or more processes to release resources (bad thing)
 - roll-back

Prevention: Banker's Algorithm

1. All threads state maximum resources needed in advance
2. Allocate resources if allocation will lead to **safe state**
3. Allows more resources to be requested than the sum of the total resources, provided the above sequence exists

Safe state determination:

j's are threads and k's are resources

- a) given a request let $Alloc^*(j,k)$ and $Avail^*(k)$ be the state after the request is granted
- b) if all threads can finish with no additional resources, then no deadlock, i.e. safe state
- c) find a thread x, for which for all k, $Need^*(x,k) \leq Avail^*(k)$.
If there is no such thread, then deadlock, i.e. state unsafe.
Otherwise, "mark" thread finished.
- d) If all threads are marked finished, or can finish with no additional resources, then no dealock; i.e. **state safe**.
- e) Let for all k, $Avail^*(k) = Avail^*(k) + Alloc^*(x, k)$. Goto (c).

Example of Baker's algorithm

Available: 20				
thread	Alloc	Max	Need	
A	90	100	10	
B	50	110	60	
C	30	160	130	

Is this a safe state?

What if B's line were Alloc 20 and Max 140?

Example with the dining lawyers: how could banker's algorithm apply? How could order enforcing help?

More on testing

Look at Emil's document on testing:

<http://inst.eecs.berkeley.edu/~cs162-tc/notes/week4.html>

From Microsoft:

Unit testing

"Take the smallest piece of testable software in the application, isolate it from the remainder of the code, and determine whether it behaves exactly as you expect."

<http://msdn.microsoft.com/library/en-us/vsent7/html/vxconUnitTesting.asp>

Integration testing

"two (or more) units that have already been tested are combined into a component and the interface between them is tested."

<http://msdn.microsoft.com/library/en-us/vsent7/html/vxconIntegrationTesting.asp>

Regression testing

Test whenever you modify an implementation.

<http://msdn.microsoft.com/library/en-us/vsent7/html/vxconregressiontesting.asp>

Sample test question:

Why is this code dangerous? (cs162 midterm, Fall '01)

Should be atomic, highly concurrent. Should not leave inconsistent state, such as disappearing with objects. You may assume stack1 and stack2 never refer to the same stack.

```
void AtomicSwap (Stack *stack1, *stack2) {
    Item thing1; /* First thing being transferred */
    Item thing2; /* Second thing being transferred */
    stack1->lock.Acquire();
    thing1 = stack1->Pop();
    if (thing1 != NULL) {
        stack2->lock.Acquire();
        thing2 = stack2->Pop();
        if (thing2 != NULL) {
            stack2->Push(thing1);
            stack1->Push(thing2);
            stack2->lock.Release();
            stack1->lock.Release();
        }
    }
}
```