

CSE 120 Winter 2002

Midterm Examination

This midterm contains 4 questions and is worth 70 points. You have 80 minutes to complete the exam.

1. (10 points) One can implement semaphores using busy waiting or via the kernel (we described the latter in class). Give a concrete example in which using busy waiting semaphores is a poor choice and a concrete example in which using busy waiting semaphores is a good choice. Be as explicit as possible; e.g., simply saying "Busy waiting wastes CPU time" is not concrete.

Busy waiting is bad if the process p that is spinning can not make progress until another process q executes, and there is only one processor. If this is the case, then p continuing to execute has no benefit at all. Some of you used the example of a priority inversion from the book, which is an extreme example of this kind of behavior.

Busy waiting can be good if that same process p is waiting for some event to occur that (1) will happen even if p maintains control of the CPU and (2) the expected time to the event will occur within a few δ where δ is the time needed to execute a context switch. Such a situation might arise in multiprocessor systems waiting for a fast interrupt to return, or for a processor controlling a physical device.

2. (15 points) You are reviewing a paper in which the author proposes a new kind of semaphore called *symmetric semaphores*. A symmetric semaphore is initialized by giving it two values: its initial value and its maximum value max . The semaphore can take on values between 0 and max . The initial value must, of course, be between 0 and max . A P operation will block if the value of the semaphore is 0 and a V operation will block if the value of the semaphore is max :

$$P(s) \text{ is } \langle \mathbf{when} (s > 0) \mathbf{do} s = s - 1 \rangle$$
$$V(s) \text{ is } \langle \mathbf{when} (s < max) \mathbf{do} s = s + 1 \rangle$$

The author claims that symmetric semaphores are more powerful than general semaphores, but the article contains no proof of this claim.

- a) Give a monitor solution of symmetric semaphores. You can use a Hoare-style, Mesa-style, Java-style, or Brinch-Hansen style monitor: the choice is up to you. The constructor will take as parameters the initial and the maximum value of the semaphores. Give the monitor invariant and the condition associated with any condition variable you use. Explain why you chose to use the style of monitor that you use to express the solution.

I use a Brinch-Hansen style monitor because signals are only executed when a process is about to leave the monitor.

```
monitor symsem {
private:
    int s, max;          // invariant:  $0 \leq s \leq \text{max}$ 
    condition gZero;    //  $s > 0$ 
    condition lMax;     //  $s < \text{max}$ 
public:

    symsem (int init, mx) {
        assert(0 <= init && init <= mx);
        s = init;
        max = mx;
    }

    entry void P(void) {
        if (s == 0) gZero.wait();
        s--;
        lMax.signal();
    }

    entry void V(void) {
        if (s == max) lMax.wait();
        s++;
        gZero.signal();
    }
}
```

- b) Prove or disprove (informally) the author's claim that symmetric semaphores are more powerful than general semaphores.

They're not more powerful. The monitor above can be implemented with regular semaphores. Thus, we can translate any program that uses symmetric semaphores into one that uses regular semaphores.

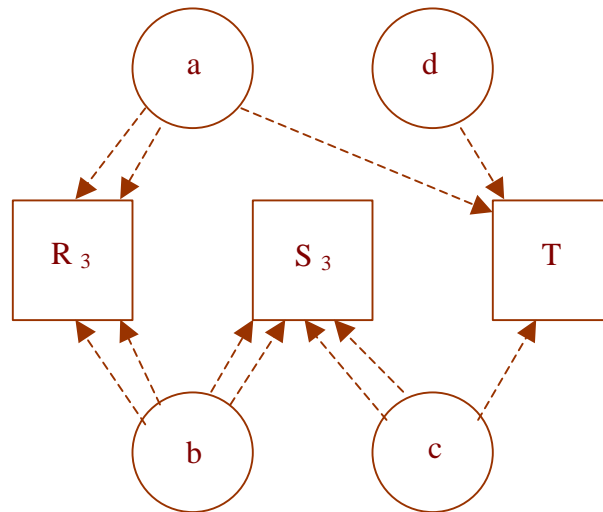
3. (25 points) Consider a system with three resources:

- Resource R has three units;
- Resource S has three units;
- Resource T has one unit.

There are four programs that utilize these resources:

- Program a can use up to two units of R and one unit of T ;
- Program b can use up to two units of R and two units of S ;
- Program c can use up to two units of S and one unit of T ;
- Program d can use up to one unit of T .

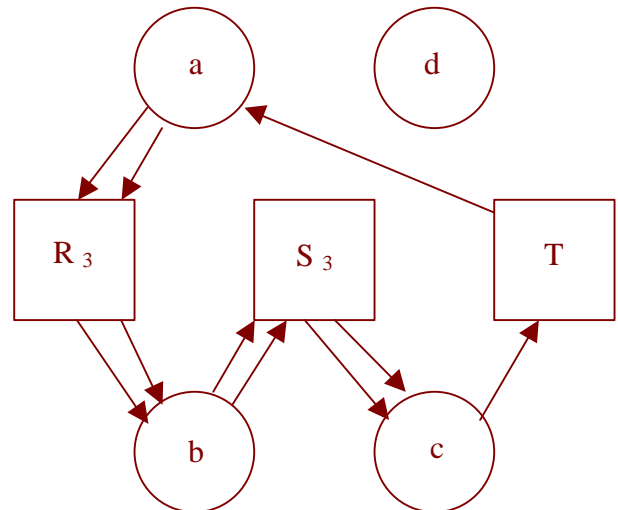
a) Draw the initial maximum claims graph for this system.



b) Give a sequence of *allocate* requests that leads to a deadlock. During the examination I asked you to also show that the resulting state was indeed a deadlock state.

```
allocate(a, T, 1)
allocate(b, R, 2)
allocate(c, S, 2)
allocate(a, R, 2)
allocate(b, S, 2)
allocate(c, T, 1)
```

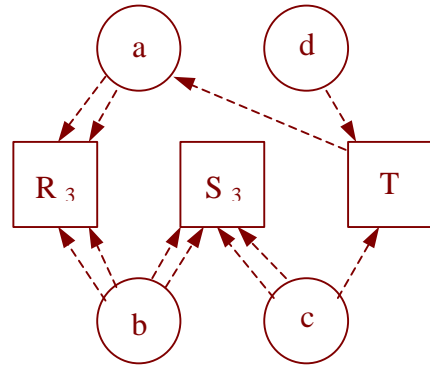
The resource allocation graph for the resulting state is to the right. All processes are blocked, and so it is not fully reducible.



c) Show how, if the Banker's algorithm were to be used for the same sequence of allocate requests, then deadlock would have been avoided.

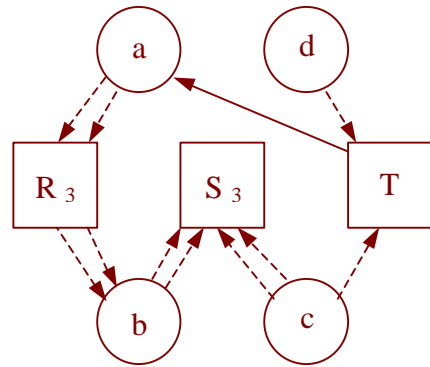
`allocate(a, T, 1)`

MCG is fully reducible; request granted.



`allocate(b, R, 2)`

MCG is fully reducible; request granted.

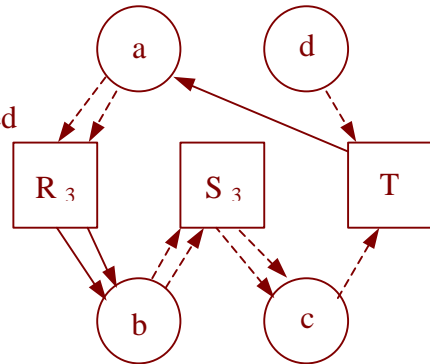


`allocate(c, S, 2)`

MCG is not fully reducible; request delayed.

`allocate(a, R, 2)`

sufficient resources not available; request delayed

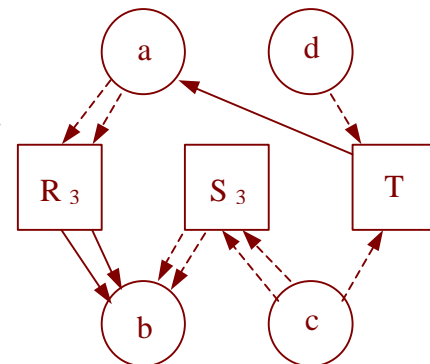


`allocate(b, S, 2)`

MCG is fully reducible; request granted.

`allocate(c, T, 1)`

not issued because c previous request is delayed.



(20 points) Some short questions, each worth five points:

- a) A colleague tries to convince you that the scheduling quantum should be as small as possible. He points out, correctly, that such a scheduler would be quite fair. Are you convinced by the argument? How should the size of the quantum be chosen?

No; the overhead of taking a context switch will become proportionally larger as the quantum decreases: the quantum should be significantly larger than the context switch time. Too long, and the latency to respond to input actions can be unacceptably long.

- b) Consider a vending machine and a person making a purchase as a concurrent system. Give five properties of the system: two safety, two liveness, and one of your own choice. Be sure to identify each property you give as a safety or liveness property.

The machine is not operational iff the red light is on. (safety).

The customer never breaks the glass window on the front of the machine (safety).

When a customer puts a coin in the machine, it eventually either registers the fact that the coin has been entered or it returns the coin. (liveness)

There will be a nonzero number of my favorite candy bar infinitely often. (liveness)

The machine never accepts pennies. (safety).

- c) Consider a buffer pool that contains ten buffers, each 100 bytes long. The first buffer starts at address 1000, the second at address 1100, and so on.

Give a scenario in which, for purposes of deadlock, you would want to model this as a resource of ten units. Give another scenario in which you would want to model this as ten separate resources of one unit each.

If the buffers are interchangeable, then they should be modeled as a single resource; otherwise, they should be modeled as separate resources. The first case arises with a buffer managers (similar to *malloc*); the latter would arise if, for example, the data to be loaded contains pointers to itself (and so the absolute address matters; this can arise when processes share memory).

- d) The only operations that one can execute on a semaphore are *initialization*, *P* and *V*. The value of the semaphore can not be directly read by a process. Why is this the case: why is it not useful to have an operation that returns the current value of a semaphore?

Semaphores are objects that are manipulated in a concurrent manner. Reading the value of a semaphore usually gives no information on what the value of the semaphore will be even by the time the reading process executes its next instruction. Hence, it isn't a useful operation to implement.