CS414 Prelim 2. 10:15 to 11:25.
Five problems, 20 points each. Closed book.

Your name: _____Your ID : _____

1. Suppose that a program P is downloading a web page from a web server S, and has two TCP connections open to that server. For each of the following scenarios, indicate whether or not it can happen and in one line, why. When we say "P crashes" we mean that the computer on which P is running halts and hence P halts, and that it doesn't restart again.

   a) P crashes and S remains operational, but S never sees either TCP connection to P "break" and hence doesn't know that P has crashed.

      This can't happen. TCP will notice that the connection is broken because the ACKs it uses to keep connections alive won't get through and it will eventually time out.

   b) Neither P nor S crashes but both of them see both TCP connections break and hence each believes that the other may have crashed.

      This can happen if the network between P and S fails or temporarily becomes extremely overloaded, so that messages can't get through. Eventually the TCP protocol times out and breaks the connection in such situations.

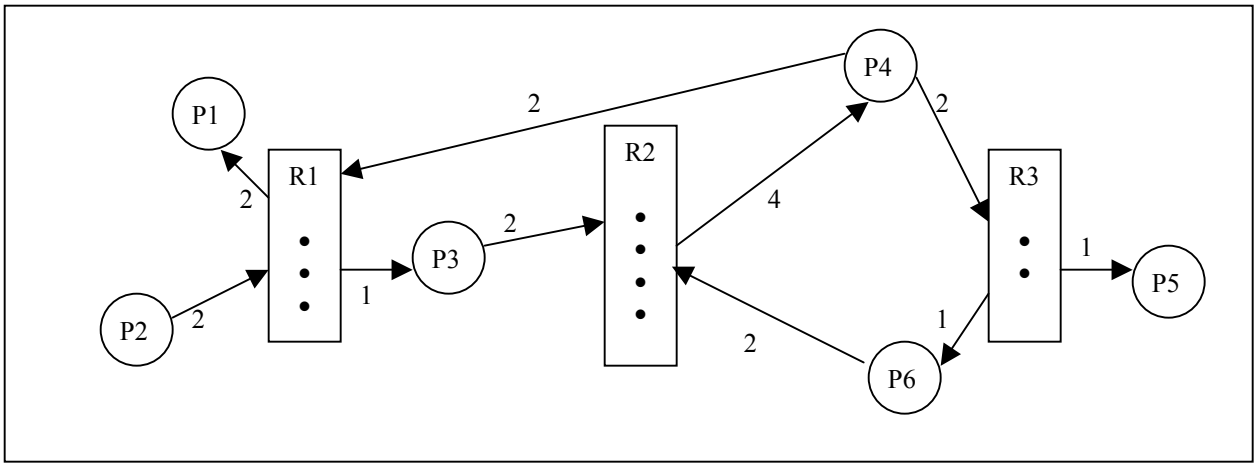
   c) Neither P nor S crashes but one of the two TCP connections breaks. The other remains connected.

      This can happen if the network between P and S fails for a short time. Since the connections aren't synchronized, they will probably time out and break at different times over a period of as much as 30 seconds. During that period if the network recovers, one might be broken and yet the other would remain connected.

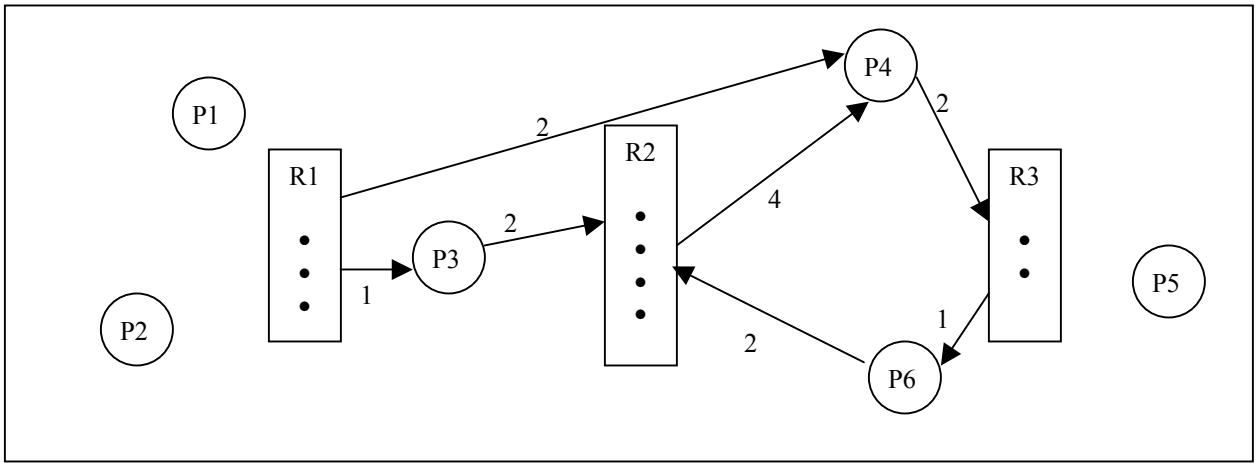
   d) P has a wireless link to the Internet and this link is flaky; about half the packets get lost. Yet even at "half capacity" the link can still deliver 500kbytes per second. Nonetheless, the web page from S downloads as slowly as if the link was a telephone connection with 33kbit (hence about 4kbyte) bandwidth.

      This is a common problem with wireless connections. The high loss rate fools TCP into thinking that the network is overloaded and hence it "backs off" by reducing the sending rate drastically. In fact TCP should have retransmitted even more aggressively, but it doesn't know that the link is wireless and hence can't do the smart thing.

2. Consider the resource wait graph draw below. Does this represent a deadlock? If so, give a reduction order that results in an irreducible component and draw the irreducible component for us. Otherwise, show us a reduction order that fully reduces the graph. (Notation: a process is denoted by it's name in a circle, e.g. P3. An arrow represents a resource request or an allocation, as we saw in class and in the book. The label on the arrow gives the number of units. A resource is represented by a box with the name of the resource and, for each unit available, a black dot – thus, below, there are a total of 4 units of resource R2.)

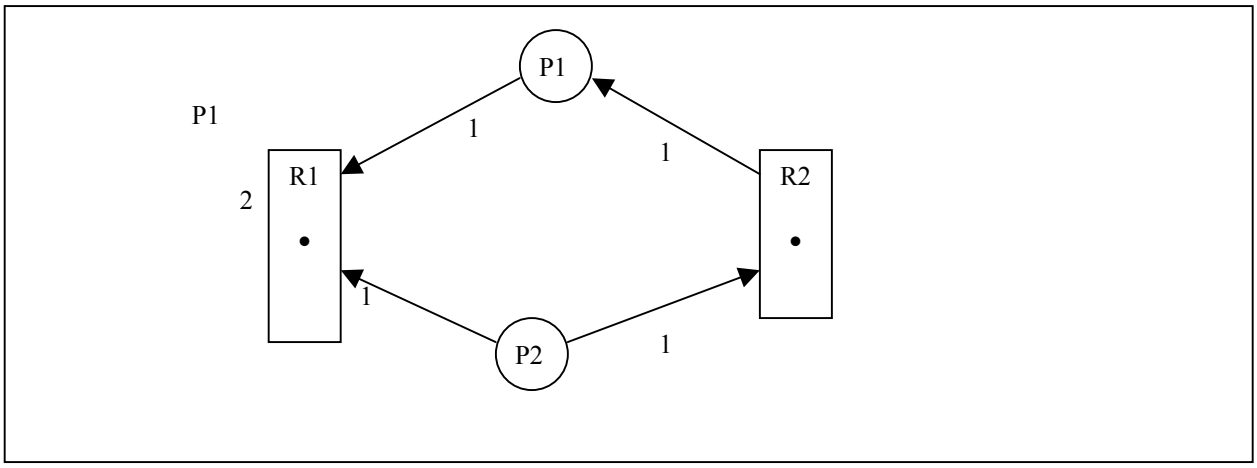


The scenario shown is a deadlock. We can reduce the graph by reducing P1, then P2, then P5. But this leaves the following irreducible component:



3. Deadlocks…

a) Draw a resource-wait graph that does not represent a deadlock, and yet where there is some request such that if that request is granted, the system will be in a deadlocked state. Indicate which pending request has the risk of causing a deadlock.



If we grant one unit of R1 to P2, the system will deadlock.

b) Suppose that the Banker's Algorithm was in use in the system. The algorithm guarantees deadlock freedom. In what way would it prevent the scenario you identified in part (a)? E.g. would it be impossible to get into the state your resource-wait graph depicts? Either explain why, or (if this state could still arise), explain why the Banker's Algorithm will prevent a deadlock from occurring. (You can assume that the Banker's Algorithm has access to the maximum resource need information for each process, even though this isn't shown on the resource-wait graph.)

The Banker's Algorithm only allows the system to enter states in which there is some way to allocate the maximum possible amount needed for each resource to some process, so that it will certainly terminate, releasing its resources after which the maximum need of some other process can be granted, etc, until all processes have terminated. Thus it would potentially allow a graph such as the one above, but would never allocate a resource in such a way that the resulting state represents a deadlock or even "potentially" leads to a deadlock.

4. Many computations involve what we call a "barrier synchronization" in which a set of threads must all reach some point before any of them advances. For example, in a weather simulation, the thread doing each part of the computation for the weather state at time *t* before the simulation starts computing the state at time *t+1*. Suppose that there are *n* threads. Write a monitor **barrier** with entry point **pause(t).** Each thread should call **barrier.pause(t)** as it finishes step *t* of the computation, and the monitor will prevent progress until it is safe to start step *t+1*. You may assume that *n* is a global constant.

```
monitor barrier {
        /* Counts the number of threads that have reached the barrier */
        int             count = 0;

        /* The barrier itself */
        condition       wall;

        /* This implementation won't actually need to look at t */
        pause(t)
        {
                /* Increase the count.  If this isn't the last thread, wait */
                if(++count < n)
                        wall.wait;

                /*  On the way out, wake up one waiting thread and decrement the count */
                wall.signal;
                –count;
        }
}
```

5. On homework 4 we compared the behavior of LRU and WS using the same value of Δ for the size of the memory partition in the case of LRU, and the size of the working set window in the case of WS.

a) In general, how would the hit ratio for LRU and WS be related, if we look at a variety of reference strings and a variety of values for Δ? Assume that we always "warm start" the algorithms and compute the hit ratio starting with Δ distinct pages in memory.

LRU would normally have a hit ration as high as for WS, and sometimes, higher. Basically, they page in the same pages but WS may chose to page things out that LRU would retain. Thus LRU may later avoid a page fault (if one of these is referenced), where WS would need to page it back in again.

b) We saw in class that WSOPT is the "optimal" paging algorithm in the working set model. We also saw that WS and WSOPT have the identical hit ratio, although WSOPT pages pages out sooner than WS. Again referring back to your answer on part a, indicate two ways in which we might think of WS as not being optimal. A single line answer should be sufficient in each case, if your answer is clear.

1. *WS keeps more pages around than WSOPT. This is because it makes paging decisions later than WSOPT. So it is less optimal in this respect than WSOPT.*
2. *WS may have a higher page fault rate (and hence a lower hit ratio) than LRU under the conditions we looked at in part a. Of course LRU always would keep Δ pages in memory, whereas WS only has Δ in memory as a maximum case that arises rarely. Still, if we focus on hit ratio, WS would perform worse than LRU.*