

### CS414 Fall 2004 Homework 5. Due in class on October 21, 2004

1. In class, we discussed the Banker's Algorithm for deadlock-free resource allocation. Consider a system with two kinds of resources,  $R_1$  and  $R_2$ . Initially there are 10 units of  $R_1$  and 2 units of  $R_2$ . Process  $P_1$  has a maximum need of 3 units of  $R_1$  and 2 units of  $R_2$ , denoted  $NEED(P_1)=\{3,2\}$ ;  $NEED(P_2)=\{7,1\}$ ,  $NEED(P_3)=\{4,0\}$ ,  $NEED(P_4)=\{7,2\}$ . At the start, no resources have been allocated.
  - a) Assume that the system is presented with the following requests, in the following order:
    - i.  $P_1$  requests 1 unit of  $R_1$  and 1 unit of  $R_2$ .
    - ii.  $P_2$  requests 2 units of  $R_2$ .
    - iii.  $P_3$  requests 1 units of  $R_2$ .
    - iv.  $P_4$  requests 5 units of  $R_1$  and 2 units of  $R_2$ .Which requests will be granted, and which ones will be delayed? *Note: We've used the notation [x units of  $R_1$  and y units of  $R_2$ ] when a process requests two resources at the same time. The system won't grant the request unless it can grant the desired number of units in one "atomic" action.*
  - b) After step (a),  $P_1$  requests 2 units of  $R_1$ . Will  $P_1$ 's request be granted immediately?
  - c) Starting in the system state reached after step (b), give one example of a request that can be granted immediately and one example of a request that would be delayed.
2. You've joined the Cornell Robo-Soccer team and are implementing a new program to control this year's robots. The program is multithreaded, with one thread controlling each of the motors (a robot has several), one controlling the kick-bar, etc. In CS414 you learned that Linux and Windows support the RR scheduling discipline with multi-level feedback queues, and with a quick Google query you found the Linux code for the version used in that system. It looks very clean. Is this scheduling discipline appropriate for controlling threads in a robot? Explain briefly. A sentence or two will be fine.
3. Machines A and B are connected by a network that supports the Internet protocols. You've purchased extremely accurate GPS-based clocks, and have begun to measure the delay (latency) for messages sent from A to B, or from B to A.
  - a) You discover that it takes twice as long for messages to get from A to B as it does for them to get from B to A. List some possible explanations.
  - b) You notice that in the A to B direction, the delays vary quite a bit; some messages arrive in as little as 1ms but others need as long as 10ms. What could cause such an issue?
  - c) Suppose that you have more machines: C, D, E, etc. Is it possible that sometimes it would be faster to send a message from A to C and then from C to B (e.g. relayed through C) than to send it directly from A to B? Why?
  - d) The wireless network in your house has a weak signal and reports a lot of packet loss, but still connects machine A to B at 11Mbits/second. Yet you

test download speeds with TCP and find it running at only 100KBits/second between A and B. What could explain this very slow performance?

4. Your close friend Doug “The Bug” Crump is developing a multi-user role playing game for the Internet. In this game, when user A fires a weapon at user B, a message is sent from user A’s computer to user B’s system to determine what damage was done. While waiting for a reply, the process on machine A is waiting for action by a process on machine B. Doug is trying to correct a deadlock in which A waits for B, B waits for C, etc, and a cycle arises.
  - a) Suppose that Doug implements a protocol that “chases wait-for” edges, as follows. If machine A has been waiting for a while, it sends a special message to B that checks B’s status. Initially, this message contains a null “visited” list. On reception, if B isn’t waiting for any other process, B can discard the message. On the other hand, if B is waiting for something to happen at C, B appends its node-id (“B”) to the list, and forwards the request to C. The idea is that if a cycle is present, we’ll see it in the list: A, B, C, D, B... and on detecting a cycle, the node that notices it can do something to stop waiting. But is Doug’s wait-for edge-chasing protocol correct? Specifically: (i) will it break deadlocks, and (ii) could it detect a “false” deadlock, that doesn’t exist, and try to “break” that?
  - b) At 4:30am, Doug wakes you up to say that his protocol in part (a) had a bug, but he thinks he’s fixed it. He says that instead of assuming that a cycle represents a deadlock, he makes the message keep looping. If a message goes around the same loop twice, a deadlock is present (e.g. A, B, C, D, B, C, D, B). Ignoring the question of whether Doug had a bug in the first place, is Doug’s new and improved solution bug-free, or does your friend need some sleep?