CS414 Summer '00: Midterm Solutions

1) Consider the following Flight Reservation Algorithm, used to book tickets from and to Ithaca, Miami, Dallas, San Diego and Seattle:
   - There is exactly one server used for reservation in each city and it **only** holds information regarding outgoing flights from that city.
   - Each server can process only a single request at a time, and the request has to be of a flight from the city where the server is located. So the server in Ithaca can only take decisions of flights leaving from Ithaca.
   - A server will not process another request till it has finished serving the current one.
   - A client could request connecting flights, for example lets suppose Mr. A requests a trip from Ithaca to San Diego via Dallas. In this case the server would issue a ticket to Mr. A only if a seat were available on both legs of his trip, i.e. from Ithaca to Dallas, and from Dallas to San Diego. So the server in Ithaca will wait till it gets a reply from the server in Dallas.
   - If Mr. B also requests a ticket from Ithaca, he will not be serviced till either Mr. A has been granted or rejected a ticket.

   a) There is a **synchronization problem** at each server:
   An incoming request is not served till the server is free and a server should not do any work unless some client process has made a request.
   Write down the client and the server process to solve this synchronization problem. Each client should have a *submit request* and *processing request* section and the server should have a *service request* section. Remember to initialize each semaphore. (2 semaphores are sufficient to solve this problem.)
   (5 points)

Client	**Server**

begin
    submit request
    V(sync);
    P(mutex);
    processing request
end
	repeat
	    P(sync);
	    service request
	    V(mutex);
	until false;

   Initialization: mutex = 1 and sync = 0;

   b) In the situation described above, a deadlock is possible!!! Describe when it could happen, and show that the 4 necessary conditions for deadlock hold in this situation. (One sentence for each condition is sufficient.) Could deadlock still occur if each city had 5 servers to process requests? Argue in favor or against it!! (Keep your answer short)
   (7 points)

   A possible deadlock situation:
   Mr. A requests for a flight from Ithaca to San Diego via Dallas, Mr. B requests a flight from Dallas to Ithaca via San Diego and Mr. C requests a ticket from San Diego to Dallas via Ithaca. If all these requests are concurrent => server at Ithaca is waiting, for the server at Dallas, to complete Mr. A's request, server at Dallas is waiting, for the server at San Diego, to complete Mr. B's request and server at C is waiting, for the server at Ithaca, to complete Mr. C's request. Here the resources are the servers and the clients are the requesting processes.
   The 4 necessary conditions are satisfied:
   - Mutual Exclusion: a server cannot simultaneously process 2 requests.
   - Hold and Wait: As in the above situation each request is holding a server (resource) and waiting for a server (resource) held by another request to be free.

- No Preemption: It is not possible to abort a request while it is waiting, hence a server can process another request only after it is done with its current one.
- Circular Wait: A holds server at Ithaca and waits for the one in Dallas to be free, B holds server at Dallas and waits for server at San Diego, held by C, to be free, and C is holding the server at San Diego and is waiting for the server at Ithaca to be free, that is held by A.

If there are 5 servers in each city, then we could add constraints to avoid deadlock. But given the conditions in the question, if requests are processed in serial order, i.e. once a server is free it serves the first request, and then the same situation of deadlock would arise.

c) What strategy could you use to remove this deadlock? Propose briefly any solution based on Deadlock Prevention that was discussed in class. Could the Resource-Allocation Graph Algorithm be used to avoid deadlock if each city had 5 servers? If no, then explain why? (Keep your answer brief)
(5 points)

If we order the servers, and requests are made only in that order, then we will be able to avoid this deadlock situation. That is we are trying to break the hold and wait condition. Suppose we order the servers alphabetically then Mr. A's request would first be made to the server in Dallas and if a seat were available only then would the request be made to the server in Ithaca.
No, the RAG algorithm cannot be used to avoid deadlock if there were 5 servers in each city. This is due to the fact that a cycle in the RAG would no longer imply that it is in an unsafe state (we did this in class). So we would have to use the Banker's Algorithm here.

d) Bankers Algorithm, as we studied in class, is successful in avoiding deadlock. Would this algorithm ever work if it were started in an unsafe state? Is deadlock guaranteed in this case?
(3 points)

The fact that you are in an unsafe state does not imply that deadlock will definitely occur. It is possible that after some time you would again be in a safe state. In such a situation Bankers Algorithm would definitely not allow further requests unless some resource has been freed!! This is because once in an unsafe state, the system cannot go into a safe state until some process has freed some resources. Once the system has reached a safe state, the Bankers Algorithm will proceed as normal, and guarantee deadlock freedom henceforth.

2) An operating system is running 3 concurrent processes, P1, P2 and P3 for 3 different users. Each process requires six units of completion time. Process P1 consists of a single thread. Process P2 consists of 2 threads, each requiring 3 units of time. Process P3 has three threads each requiring 2 units of time. Assume that the threads never block and the processes are always in main memory.

a) If the threads are kernel supported, and if the kernel implements a preemptive, round robin scheduling of threads with a quantum of one time unit. Of the first six time units, how many will be devoted to each of the three processes.
(5 points)

Process 1 receives 1 time unit, process 2 receives 2 and process 3 receives 3 time units.

b) Repeat part (a) if the threads are at the user level.
(5 points)

Each process receives 2 time units.

3) In this question we will propose another powerful design of a monitor. Suppose we permitted monitors to communicate, i.e. a process executing in a procedure in a monitor could call a procedure in some other monitor. But still we cannot have more than one process in a monitor since that is one reason why monitors were proposed, so the calling process might have to wait for its turn to get in the other monitor. At the same time it would not relinquish control of its original monitor, since it has not finished execution and is also not waiting on a condition variable. Is this construct of monitors as good and as powerful as Hoare's monitors? Present a brief argument!!
(5 points)

The modified monitor won't work since it gives rise to a potential deadlock situation. Suppose process A in monitor M calls a procedure in monitor N but is not able to enter N since B is currently executing in monitor N. If at this time B calls a procedure in M, what results is a deadlock situation, with A waiting for B to get out of the monitor N and B waiting for A to leave monitor M. So this construct for communication between monitors is incorrect!!