Department Computer Science	**Distributed Systems**
VU University	**16.02.2011**

MAKE SURE THAT YOUR HANDWRITING IS READABLE

1a Give a simple, centralized algorithm for mutual exclusion. *4pt*

The simplest one is described as Fig. 6-14 on page 253.

1b Give a fully decentralized algorithm for mutual exclusion. *6pt*

The algorithm discussed in Section 6.3.3 is definitely OK. As an alternative, a good explanation of Ricart-Agrawala (Section 6.3.4) would also suffice, and to a certain extent the token-based one, but one could argue that this is not fully decentralized.

1c Compare the two algorithms with respect the number of messages that a process needs to exchange before it can proceed as only one. *6pt*

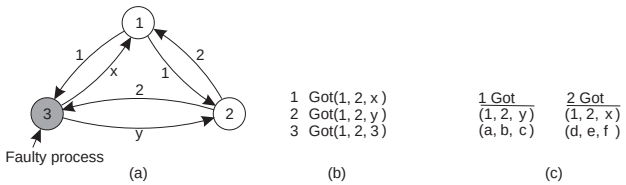
The answer is in Fig 6-17, but watch out for the decentralized solution: it should $2mk+m$ instead of the $3mk$. In any case, be sure to motivate your answer!

2a Explain the difference between fault, error, and failure by means of a compelling example. *6pt*

As explained in class: a faulty programmer may lead to introducing errors (bugs), which may in turn lead to a failure in the execution of a program.

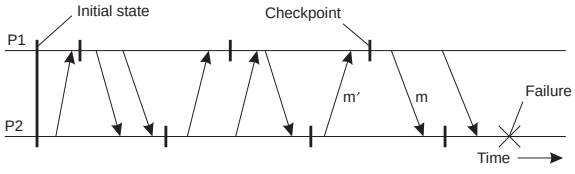
2b Show that having three processes of which one is faulty, is not enough to guarantee agreement between the two nonfaulty ones in a Byzantine setting. *6pt*

The easiest thing to do is to prove this by contradiction: let the three processes pass their information in two rounds, and see whether you can get a majority. Essentially, this means drawing Figure 8-6:



2c Illustrate by means of an example how independent checkpointing can lead to a cascaded rollback. Be sure to explain your answer! *6pt*

You need to explain where the rollback actually comes from: you cannot have recorded the receipt of a message m without having recorded that it was sent. Taking Fig 8-25 will then complete the answer:



3a Why is the following data store not sequentially consistent? Is it causally consistent? Be sure to explain your answer. *6pt*

P1:	W(x)a			
P2:		W(x)b		
P3:			R(x)b	R(x)a
P4:			R(x)a	R(x)b

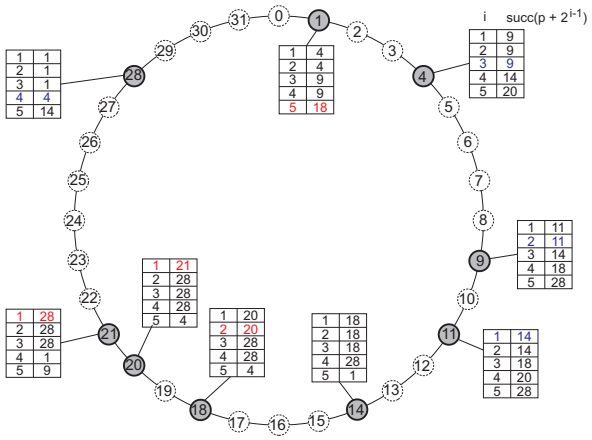
It is not sequentially consistent because P3 and P4 are reading the effects of concurrent writes (by respectively P1 and P2) in different orders. It is causally consistent because there are no causal relationships that need to be obeyed.

3b Consider a system that combines read-your-writes consistency with writes-follow-reads consistency. Is this system also sequentially consistent? Explain your answer. *6pt*

No, it is not sequentially consistent. Although the combination effectively provides location-independent consistent behavior for a single process, it does not guarantee that when there are two concurrent write operations at different locations, that the effect of those writes will be seen everywhere in the same order.

4a Resolve the following key lookups for the shown Chord-based P2P system: *6pt*

source	key
4	2
4	4
14	15
14	19
21	2
21	20



2@4: 20–4; 4@4: None; 15@14: 18; 19@14: 18–20; 2@21: 1–4; 20@21: 9–18–20

4b Which finger tables will be affected if a node with ID 7 enters the ring, and how? *6pt*

In this case, we need to consider the nodes who were pointing to 9 as successor, which are 1, 4, and 21. Their finger tables now become: [4, 4, 7, 9, 18], [7, 7, 9, 14, 20], and [28, 28, 28, 1, 7], respectively.

4c In Chord, there is a difference between iterative and recursive lookups. Which one performs better? *6pt*

If we assume that Chord does not optimize its finger tables to take network proximity into account, there is no performance difference: all operations take place on a logical overlay anyway. With network proximity, recursive lookup will generally be cheaper as the distance to the target will gradually decrease. Note that such a decrease need not always take place.

4d What would be a simple extension to Chord so that network proximity is taken into account when doing a key lookup? Give an example using the previous figure. *6pt*

A simple solution would be to allow nodes to register more than only the successor. In the current setup, a node p keeps track of the first in the range $p+2^{i-1}, p+2^i-1]$. It can also register other nodes if it happens to discover them. For example, at present, $FT_4[4] = 14$, but there is no reason why node 4 couldn't also register that $FT_4[4] = \{14, 18\}$. In that case, if it needs to lookup key 19, it can forward that request to node 18 instead of 14, if the network distance to 18 happens to be shorter than to 14.

```
(01) main(int argc, char* argv[]) {
(02)    Ice::Communicator   ic;
(03)    Ice::ObjectAdapter  adapter;
(04)    Ice::Object         object;
(05)    ...
(06)    ic = Ice::initialize(argc, argv);
(07)    adapter = ic->createObjectAdapterWithEnd Points( "MyAdapter","tcp -p 10000");
(08)    object  = new MyObject;
(09)    adapter->add(object, objectID);
(10)    adapter->activate();
(11)    ic->waitForShutdown(); }
```

5a Explain what is happening in the (incomplete) code fragment given above. *6pt*

In this example, we see that an adapter is created (07), after which we place a newly created object under its regime (09), activate it, meaning that messages can now be sent to that object (10), and then subsequently wait until the server is shut down. Overall, the code fragment shows the minimal code for creating an object server.

5b Which information will the objectID parameter contain in line (09)? *4pt*

That ID will most likely contain (1) the IP address of the server, (2) the port to which the adapter containing the object is listening to (10000), as well as a unique index for the object itself.

5c What is meant by an *activation policy* for an object adapter? *6pt*

An activation policy specifies how an object under the regime of a specific adapter is invoked by a remote client. There are various arlternatives: the thread running the adapter invokes the object; a new thread is created, or incoming requests are queued to wait for their turn.

5d Suppose we would add the following lines (10x meaning a line inserted after line 10). How would this affect the original code? *4pt*

```
         ...
(07x)    adapter2 = ic->createObjectAdapterWithEnd Points( "MyAdapter1","tcp -p 10001");
         ...
(09x)    adapter2->add(object, objectID2);
         ...
(10x)    adapter2->activate();
         ...
```

The only thing we'd be doing is adding allowing the newly created object to be accessed through a second adapter that had been added to the object server. However, it is unclear from the code itself whether we would be getting ourselves into trouble, as we would still have only a single instance of object*. Only the way that we access it, has been extended.*

***Grading:** The final grade is calculated by accumulating the scores per question (maximum: 90 points), and adding 10 bonus points. The maximum total is therefore 100 points.*