CSE 42 DIS: DISTRIBUTED COMPUTATION

Assignment No. 2. Total Marks: 20.
Due at 9.30 a.m. on Friday, 23rd Sept. 2005. USUAL PENALTIES FOR LATE SUBMISSION.
This is an individual assignment. Usual checking and penalties for plagiarism apply.

1. In the first assignment, you solved the problem of simulating many processes which communicate with each other, and log the events on a common file. The objective of this next exercise, is to implement a set of processes that are able to carry out a checkpoint and restart, using the method discussed in the lectures. For this, you will use the first assignment as a starting point.

2. The present assignment requires that you write a program that creates a number of processes (at least 4), each of which will send messages to one of the others chosen at **random**. The process should sleep for a **random** amount of time after sending a message, and then **wake up** and look for any received messages, before sending another message and so on. This sending of messages is considered to be part of the the "normal" work of the distributed system.

3. The messages will be sent and received by invoking a "service" from a **"lower layer" (function call, or method).** It is the job of the lower layer to also receive the messages sent to this process and then pass them on after storing them if necessary, for the purposes of a check–point. Similarly, it will be responsible for the roll-back (restart) operation as well, by "delivering" to the upper layer the messages it has buffered on the previous checkpoint. We will not concern ourselves with saving the "state" of the process during the check-point, since this would depend intimately on the nature of the distributed application that was being implemented. We will assume that this state is correctly saved and restored.

4. As explained above, it is the lower layer, which will carry out all the functions required for check-pointing and restart. Since the messages are dummies anyway, the actual delivery to the higher layer can be a "dummy" operation in the lower layer itself, which simply destroys the relevant message. However, this event can be "logged" as a message delivery in the log-file for testing purposes. Once the program is running, you will only need to log those events which are relevant to the check-point or the restart.

5. Each lower layer should print out a statement on the common log file, to record that it has taken a checkpoint and list the messages that it has buffered and at which point the checkpoint is complete. It should also log messages which arrived during s checkpoint but were NOT buffered as they were not a valid part of the state of the system. Similarly, when doing a restart, it should print out the messages it "delivered" to the upper layer, as well as any that it has discarded, and the point at which the restart is complete. It should also state how these events came to pass i.e. whether it was selfinitiated or initiated by a message from another process.

6. Notice that, no matter how you have implemented the processes,, the lower layer will need to buffer the messages at a check-point, while also delivering a copy to the upper layer. These buffered messages will then be delivered again to the upper layer when a restart is carried out.

7. The only question that remains is: when should the check-point and restart be carried out?. Recall, that in your previous assignment, every message had a local sequence number and source and destination, so that it was uniquely identified. Which ever process reaches the local sequence snumber 10, should initiate a checkpoint (unless they have received such a request from another), and which ever reaches local sequence number 20 should initiate a restart. They should all continue until sequence number 30 and then terminate. This is not quite as simple as it sounds, and you will need to think about it a bit.

8. The final version of the assignment **MUST** run on the RedHat linux system on latcs6 in the department. This is necessary for obvious reasons of standardisation. The program should be executable by me to check that it does indeed work as claimed. A minimal "user manual" giving program names, files, directories, etc. must accompany the submitted assignment in a README file.

9. The assignment should be **submitted electronically**, using the **submit command**, which functions as follows:

Create a special directory in your home directory called CheckPoint. Put all your work (the files to be submitted) into this directory, and then type: **submit CSE42DIS** CheckPoint <CR>. You can verify after this what are all the items that have been submitted by typing **verify** <CR>. Note that a mistake in submission can be overwritten by repeating the submit process. The time-stamp of the electronic submission will be taken as the time of submission. This makes it possible to submit from a remote location, as long as you can log into latcs6.

10. **The electronically submitted assignment** should contain the following files:

(i) the README file: a *very brief* description of the program design, (descriptions of the main data-structures and the structure of the message) along with the minimal "user manual". This should describe how to compile and run the program and get the required results. **This is all the written material needed (about a page at most). This needs to be an ASCII file. i.e. no word processing is required.**

(ii) the source program files including header files etc. and
(iii) the log-file, clearly showing where messages have been saved when the checkpoint is initiated and again when the restart is carried out (as explained in paragraph 5 above).

11. **NO hard copy of the source program** is required. **Caution:** Keep the program simple, since the algorithm is complex enough. An "over-elaborate" construction using C/C++ /Java can lead to a program that is hard to debug in the limited time available.

12. **Marking Scheme: Total Marks: 20.** (You may find it useful to **incrementally enhance and test** your program along the lines indicated below. At each stage you can be sure how many marks you will get, and what remains to be done. Once again, since there is a substantial amount of learning that is involved in the production of this program, the incremental approach is recommended. A "waterfall" model of development is not recommended)

(a) checkpoint inititiation and completion by every process as logged in the file. (no message storage)
5 marks.
(b) maintaining a buffer for stored messages, which correctly stores the required messages and logs them. Also discards inappropriate messages and logs those.
5 marks
(c) restart correctly and events logged, including delivery of stored messages.
5 marks.
(d) Correctly rejecting (discarding) messages which arrive after a restart and logging these events.
5 marks

References:

[1] Lecture Notes for CSE 42 DIS, 2005.

IMPORTANT: Return of assignments: Feedback will be via e-mail, sent individually to each student. In addition, the marks will be displayed against your student number on the subject web-site. **Please verify that your marks have been correctly entered in the list, and report any errors or omissions to me before the end of the semester.** Exam marks will be similary displayed after the exams as "unofficial" marks. **If anyone has privacy concerns about this procedure, please e-mail me beforehand and your marks will not be displayed.**