Programming Assignment 2: Process Groups: Membership Service

Due Wednesday Oct. 31, 10 pm

1 Lab Task

In this lab, you will implement a membership service similar with the one described in slides 18, 19, 20 from topic 5. We will refer to all servers as peers.

As in the multicast service from project 1, each peer starts by knowing a list representing the maximum number of hosts that can be part of the group. You can assume that the leader is the first host in the file and that peers will know starting on who is the leader, from the file. Peer ids will be assigned based on the line in the file, starting at 1.

The goal is for peers to build and maintain a list of all alive peers. A membership consists of a unique id, referred to as *view id* and the list of the alive peers. The view id should be monotonically increasing and create a total order, even when the leader changes. Each peer should maintain a view id and a membership list.

Every time the membership list changes each peer should print the new view id and the new list of peers, the list should be printed in increasing order of peer id.

You can assume that the maximum number of peers is 10. You should test your code with minimum 5 nodes and provide the 4 testcases described below.

You can use C/C++, Java or Go.

1.1 PART 1: Add a peer to the membership list (30 points)

When a peer starts it will contact the leader. The leader will start a 2PC with existing members as follows:

- Leader sends a REQ message containing a request id, current view id, operation type, and the id of the peer to be added, to all the other peers in the existing membership list. Operation type is ADD.
- Each peer saves the operation he must perform (request id, current view id, operation type, peer id) and sends back an OK message containing the request id and the current view id.
- When receiving OK messages from all the alive peers (with matching request id and current view id), the leader will increment its view id, add the new peer to its membership list and send a NEWVIEW message that contains the new view id and new membership list to all the members including the new peer.
- When receiving the NEWVIEW message all processes update their view id and membership list and print the new view (view id and membership list).

You can assume that no peer joins before the previous one finished and that there are no crashes. Leader starts first with the membership list containing just himself.

You can optimize the protocol described above, for example send the entire list only to the new peer, and to the other peers just the view id.

Communication can be implemented with either TCP or UDP. Communication should be reliable, if you are using UDP you need to implement a reliable channel.

TESTCASE 1: Each process joins one by one, till all the processes from the configuration file joined. Leader starts first. At the end you should see for each peer how the membership list and view id changed, every time a new peer joined.

1.2 PART 2: Failure detector (10 points)

Each peer will implement a failure detector as follows.

- Each peer will broadcast a "heartbeat" message every time a timeout T expires.
- When a peer P does not hear from another peer Q within a timeframe that corresponds to twice to period with which heartbeat messages are sent, it will declare Q dead and should print on the screen "Peer Q not reachable.". (Failure of a peer here means missing two consecutive heartbeats from that peer).

All the heartbeat messages should be implemented with UDP. Heartbeat messages are not reliable.

TESTCASE 2: Crash a peer and have all other peers detect its failure. All peers should print on the screen the failure detection message as described above.

1.3 PART 3: Delete a peer from the membership list (20 points)

Once the leader has detected that a process crashed, it will initiate a 2PC to remove the member from the list – following a similar process with the one for adding a new member.

- Leader send a REQ message containing request id, current view id, operation type, and peer id of the peer to be removed. Operation type is DEL.
- All peers save the operation they must perform (request id, current view id, operation type, peer id) and send OK with matching request id and current view id.
- Once the leader receives matching OKs from all alive peers (request id and current view id match with the request), the leader will increment the view id and make the change to the list, and send to all peers a NEWVIEW message that contains the new view id and the corresponding updated list. All peers update their view id and membership list and print the new view id and membership list on the screen.

As in the case of join, you can assume that only one peer leaves at the time and that finishing the protocol to update the list will not be interrupted by another event. The leader does not leave or crash. Only the leader needs to detect the crashed peer to update the list, the other processes even if they detect the failures, still need to wait for the leader to initiate the change.

Communication should be implemented as in Part 1.

TESTCASE 3: After all the peers are part of the membership, start crashing them one by one, wait for each view change to finish before crashing another process, In the end you should be left only with the leader.

You can optimize the delete if you want, but the peers will make the changes permanent to the list, i.e. modify the view id and the list only when receiving the NEWVIEW message from the leader.

1.4 PART 4: Leader failure (30 points)

Once the leader crashed, the peer with the lowest id becomes the new leader, all members should automatically know from the membership list who is the new leader. The new leader initiates a reconciliation phase as follows:

- The new leader sends a NEWLEADER message to all peers asking if they have any pending operations – operations that they have saved when they received a request for changes from the previous leader, but they did not apply to the list yet because the leader died before sending the NEWVIEW message. NEWLEADER message should contain request id, current view id, operation type; operation type is PENDING.
- The peers will respond with a message that contains that request, current view id, the operation, ADD or DEL, and the id of the peer to be added or deleted – if they have any pending. If they do not have anything pending, the operation will be NOTHING.
- The new leader will finish the received operation (remember that we assume that there is only one event being handled and no cascading events) by restarting the add or delete protocol described in Part 1 or Part 3.

You can assume that the change was only one event, either an add or a delete. Communication is as in Part 1 and Part 3. You can optimize the protocol if you want.

TESTCASE 4: crash the current leader once he sent a request for adding a member, but before all processes received the request, specifically the peer that will be the new leader has not received this information. Your testcase should show that the new leader finishes the operation correctly and the pending peer is added to the list and all peers correctly update the view id and the membership list.

IMPORTANT: You do not need to handle more than one event, one process joins, or one leaves. You do not need to handle "cascading events" - i.e. all joins and leaves finish before another event starts. The only exception is the testcase for the leader failure as described above.

The REQ, OK, NEWVIEW, NEWLEADER are message types. ADD, DEL, PENDING are operation types. Do not use strings for them.

```
Usage: prj2 -p port -h hostfile

  -p port
    The port identifies on which port the process
    will be listening on for incoming messages.
    It can take any integer from 10000 to 65535.

  -h hostfile
    The hostfile is the path to a file that contains
    the list of hostnames that the processes are
    running on. It assumes that each host is running
    only one instance of the process.
    ...

    All the processes will listen on the same port.
    The line number indicates the identifier of the process
```

```
which starts at 1.
```

2 Submission Instructions

Your submission must include the following files (10 points):

1. The **SOURCE** and **HEADER** files (no object files or binary)
2. A **MAKEFILE** to compile and to clean your project
3. A **README** file containing your name, instructions to run your code and anything you would like us to know about your program (like errors, special conditions, etc.)
4. A **REPORT** describing the system architectures, state diagrams, design decisions, and implementation issues

Your submission should also include scripts, code, explanations on how to ran the 4 TEST-CASES described above. To receive full credit for each part, you need to provide the required testcases.

Information about submission is in post @11 in piaza. Name of the project in the submission command is prj2. Please do not submit be email.

3 Additional resources

You may find the following resources helpful

- Socket programming: `http://beej.us/guide/bgnet/`
- Unix programming links: `http://www.cse.buffalo.edu/~milun/unix.programming.html`
- C/C++ programming link: `http://www.cplusplus.com/`