**Advanced Operating System**

**Project 3 – Reference Document**

Submitted By

Aron Sajan Philip

**Information**

Run with the switch ConfigMode to configure participant systems. This needs to be done only once and the participant system file will be shared across by all systems

**Background**

1. Every node addition operation works on a set. Assume nodes (A,B) is a set. Now when C joins with either A or B then the set will become (A,B,C)
2. A node cannot belong to two groups at the same time. So if two groups (A,B) and (C,D) exists then when C joins with either A or B then two sets are formed (A,B,C) and (D). Note :- While performing merging/repairing, a node should join the group having the recent data. Since repairing is done with user interaction, it is assumed, that the user will take care of this.

**Commands**

These are the list of commands used by the systems to interact with each another. The commands are implemented as java class objects which are then subject to binary serialization to convert to byte array. This data is then de-serialized at the received and based upon the command necessary action is taken

1. I\_WANT\_TO\_JOIN 🡪 Joins with a system; The data object of the destination will be synced with the source
2. CASCADE\_JOIN 🡪 When a node receives I\_WANT\_TO\_JOIN from a new node then the node informs other nodes in its NEIGHBORHOOD\_LIST of the new comer using this command
3. JOIN\_NETWORK\_ACK 🡪 The destination system which receives “I\_WANT\_TO\_JOIN” from source send this command along with its list of neighbors and SharedData to the source. The source will then update its data based on this.
4. DISCONNET\_ME 🡪 The destination will remove the source from its neighbor list (NEIGHBORHOOD\_LIST) after which the source does the same for destination. This will simulate loss of network connection. This will not affect the NODE\_LIST data stored in the shared data.
5. CASCADE\_DISCONNECT 🡪 When a node receives DISCONNECT\_ME from one of its neighbors then it cascades this disconnect command to all its other neighbors.
6. CAN\_I\_UPDATE 🡪 Voting request to update the data
7. GO\_AHEAD\_UPDATE 🡪 Confirmation message for update
8. EXECUTE\_UPDATE 🡪 Updates the data without further voting (use after confirmation from all systems)

**Shared data contains:**

VN;RU;DS\_ID; NODE\_LIST(List of systems that makes the RU count) to be considered in further communication

**Network Simulation**

Each system maintains a list of neighborhood systems - NEIGHBORHOOD\_LIST. When a system disconnects from a group its entry will be removed from the NEIGHBORHOOD\_LIST of all systems in that group. In other words NEIGHBORHOOD\_LIST is a data-structure that emulates active systems to which one is connected in the view of a participant system.

**Voting Algorithm:**

1. System A issues CAN\_I\_UPDATE command. All systems except H responded.
2. A checks 2xCOUNT(Systems responded)>RU, if true then A performs UPDATE();
3. If 2xCOUNT(Systems responded)=RU then A checks if DS\_ID is in the list of systems responded; If true then A performs UPDATE()
4. 2xCOUNT(Systems responded)<RU then a discards the operation

**UPDATE()**

* + - “A” updates the data as VN=2; RU=6; DS\_ID=G;
    - A Removes H from NODE\_LIST and NEIGHBORHOOD\_LIST
    - A re-computes DS

1. Sends this information [Shared Data (VN;RU;DS\_ID;NODE\_LIST)] as “GO\_AHEAD\_UPDATE” to all systems except H.
2. On receiving GO\_AHEAD\_UPDATE from A, all systems B through G updates their shared data with that received from A
3. Note: H will still have its VN=1, RU=7 and DS\_IG=H; When H does an election no system respond and since RU=7, H will not get majority of vote and so it cannot update its data