Design Document for Q1

Name: Samina Shiraj Mulani **Name:** Jithin Kallukalam Sojan

ID: 2018A7PS0314P **ID:** 2017A7PS0163P

Deliverables:

coordinator.c

node.c

To run:

$ *gcc node.c -o node*

$ *gcc coordinator.c*

$ *./a.out <total-num-of-nodes> <root-node-id> <input-file>*

Assumptions:

- The CLA input to the coordinator(root) executable has to be the number of nodes and the root ID, which has to lie between 0 and N-1(The numbering of all the other nodes are relative to the root ID specified in the CLA).
- The coordinator creates N-1 child processes. These are the processes that correspond to the N-1 nodes other than the root node (coordinator.c). Communication between the nodes happens through TCP sockets.
- The node.c file is compiled with the executable name as node, this name is used in execv() when the coordinator creates the N-1 child processes.

Design

1. **Sockets**

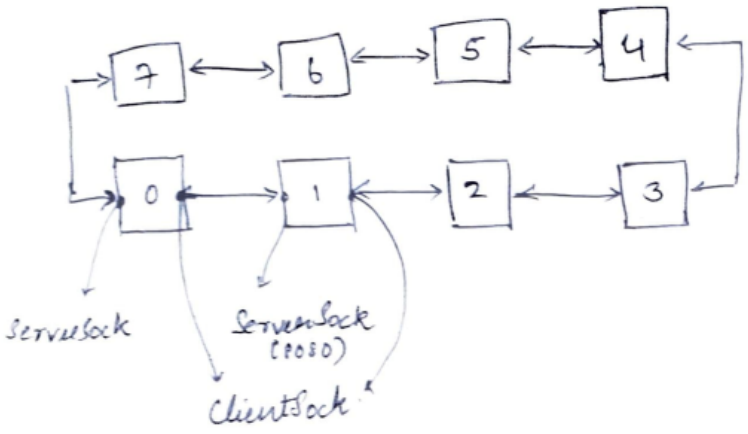
   Each node has two sockets. **One is bound to a particular port number and it acts like a server socket. The other socket acts like a client socket that connects to the server socket of the**

next node in the ring topology (for example, the client socket of node 3 connects to the server socket of node 4).

The port number that each node binds its server socket to is known to the coordinator and hence the other nodes. This is necessary for every node in the ring topology to connect to its next node.

Note that the IP and PORT number of the destination/source address that gets printed by each node on the console is the actual destination/ source of the data, not the next/previous node in the ring topology. The ring topology only acts as the carrier of the data to its true destination from its true source.

Since the IP and PORT of the sockets that data is being sent to and received from have to be printed in the console, the client sockets' IP and PORT numbers are sent along with the unsorted data, which is sent from the parent to the child.



2. **Format of data transfer**

   There are two types of data transfers in this system. One is a downwards data transfer, the parent sending unsorted sub-arrays to its children. First the destination node id is transferred, along with the source node id. **The destination node id is used for a node in the ring topology to understand that the data that it is receiving right now is meant for itself and should not be transmitted further.**

Also, the client socket IP address and PORT number is sent for reference as mentioned above. Finally, the unsorted sub-array is sent.

There is also an upwards transfer of data, when the children send sorted sub-arrays back to their respective parent. Here, the server port address is not necessary, therefore only the destination node, source node and the sorted sub-array is sent. The destination node id is used in the same manner as stated above.

Downward Data Transfer Stream:

<dest-id> <src-id> <src-client-addr> <sub-array>

Upward Data Transfer Stream:

<dest-id> <src-id> <sub-array>

3. Algorithm for data transfer

Since every node knows its relative id and the total number of nodes, it can figure out its relative position in the tree structure for the merge sort algorithm. **On receiving an unsorted array/sub-array from the file or from the parent on its server socket, a node keeps one element for itself and divides the sub-array to be sent to its immediate children. These sub-arrays with different destinations are sent through its client socket.** The sub-array is divided according to the size of the sub-tree with the immediate child as root.

For example, at node 0(root node, 16 node tree), the unsorted array is read from a file. Node 0 keeps one element with it, and sends one element to node 1, two elements to node 2, 4 elements to node 4, and 8 elements to node 8.

The node then receives the sorted sub-arrays from the same children. These sub-arrays are received on the client socket of the node. Following the previous example, node 0 gets a sorted sub-array of size 1 from node 1(basically the element itself), a sorted sub-array of size 2 from node2, a sorted array of size 4 from node 4 and the same of size 8 from node 8.

4. Algorithm for Merge Sort

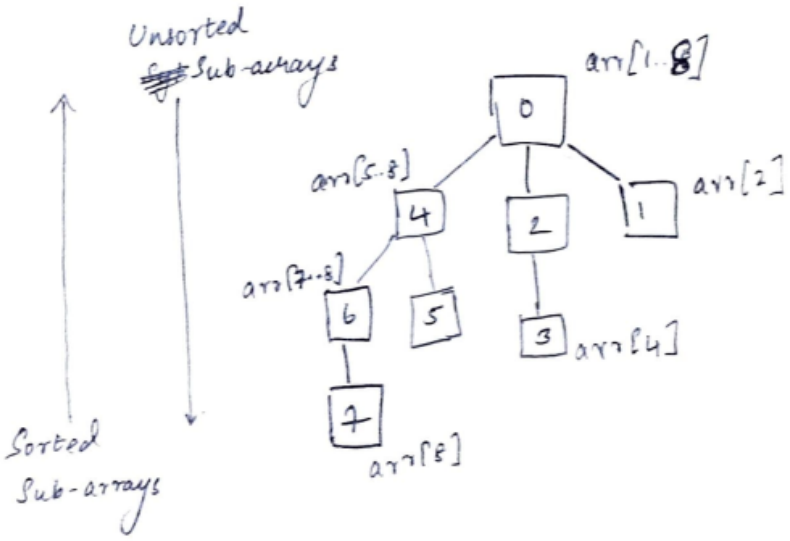
The merge sort algorithm is very involved with the manner in which the data transfer is done as discussed above. Once all the sorted sub-

arrays are received at the parent, merge sort is done on the sub-arrays.

Following from the previous example, the element at node 0 and the element received from node 1 are merged to form a sub-array of size 2.

The sorted sub-array from node 2 and the sorted sub-array stated above are merged to form a sorted sub-array of size 4. Similarly, a sorted sub-array of size 8 and finally a completely sorted array of size 16 is formed. Since the example is about node 0, the sorted array is now written to the output file. **In-case this was any other parent node than node 0, the sorted array would now be sent back to its own parent.**

This is the complete process in how an unsorted array is taken from the input file and the sorted array is placed in the output file.



Design Choices:

- Since there are two sockets on each node both of which can get packets at any time, I/O multiplexing is required. For this, select() was chosen. Both the client and server socket fds are placed in the read set.