Sunday Drivers Network

# Purpose:

The main purpose of the project is to create a small intelligent network without any central controller. the network should have the below intelligence.

1. Each process in the network reads neighbour information and cost from a local file.

2. Each process builds a network graph using the neighbour information received from all the other processes in the network.

3. The network can reconfigure itself, if any node dies or edge is lost.

# Assumptions:

1.A process is a node.

2.At any point of time, a node can be connected to minimum one node and maximum "n" number of nodes.

# High level design:

Each node is having a local database where the graph data is stored.

Each node is a multithreading system. Below threads run in all the nodes in the network.

Thread1: This is to listen the 'keep alive' data from all the other connected nodes. If node A fails to send the 'alive info' to node B for a particular timeframe, node B marks the direct path to node A is lost and makes the edge from 'active' to 'inactive'.

Thread2: This is to send the 'keep alive' data to other directly connected nodes. There will be a keep alive timer. and when the timer expires, the 'alive info' is sent to all the other directly connected nodes.

Thread3: This is to make the minimum spanning tree with the 'active' edges.

# Low level design:

We have a intelligent network where each node can detect the neighbour. To achieve the intelligence all the nodes open sockets to read data from the other directly connected nodes and to send data to other directly connected nodes.

Each node behaves like a server when it receives the 'alive info' from other directly connected nodes and as a client when it sends 'alive info' to other directly connected nodes.

The thread1 of each node is for opening a socket as a server. This thread opens a socket and waits for the 'alive info' coming from the other nodes. Maximum "n" number sockets can be created for each node to connect to adjacent nodes.

The thread2 acts as a client and connects to its adjacent nodes so that it can share its neighbor adjacency information with the other nodes. It can connect to maximum "n" number of nodes.

# Functional level design:

/\* Node Structure Example \*/

Members : 1. File pointer to a file of its neighbor info

2. Number of neighbors it is having

/\* Function Example \*/

//The main function

main()

// The thread function to create server socket to read data from the adjacent nodes

read\_adjacent\_node()

//The thread function to send own neighbor information to the adjacent clients

send\_neighbor\_info()

// The main function to construct MST using Kruskal's algorithm

KruskalMST()