Design : Each node running as a process.

Each process maintains a message queue.

Every other node sends its adj list to all its adj nodes

#define INF 1000000 //some largest num

#define MAX\_LEN 50 // max queue len

/\*\*\*\*\*\*\*\*\*\*\*\* All edges maintained in below graph\*\*\*\*\*\*\*\*\*/

int graphmatrix[50][50] = {0}; // all edges which are not defined will have a cost of 0

semaphore s1; // semaphore to protect access to graphmatrix while updating/reading

/\*\*\*\*\*\*\*\* label enum used in dijkstras algo\*\*\*\*\*\*\*\*\*\*/

enum lbl { black, gray, white };

/\*\*\*\*\*\*\*\*\* fun to update matrix\*\*\*\*\*\*\*\*

returns 0 on success

\*/

int updategraph(int \*\*graph, int s, int d, int cost)

{

graph[s][d] = cost;

return 0;

}

struct adjlist

{

int node;

int code;

};

/\*\*\*\*\* enum may be used to distinguish the type of the packetsend through message queue\*\*\*\*/

enum buffertype {upd\_graph};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Strucutre used to define the format of the buffer exchanged between processes

Info exchanged is the cost of all adj nodes

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

struct queuebuffer //buffer to be used in queue

{

buffertype type;

int sending\_node;

int adj\_cost[50];

};

/\*\*\* utility func to update an edge details in graph safely\*\*\*\*\*\*\*\*\*\*\*/

int updategraph(int s, int d, int cost)

{

sem\_wait(s1);

graphmatrix[s][d] = cost;

sem\_signal(s1);

}

int q = msg\_creat("\queuename", MAX\_LEN, sizeof(struct queuebuffer)); //store queuename as a global variable

int process\_alive =0;

sem\_init(s1, 1);

int main()

{

nodeprocess(); // need to create menu and keep in loop

}

int nodeprocess()

{

//create msg queue --- to receive updates for other processes ---

int cost , n, node;

int thread t1;

char \*queuename[20];

process\_alive = 1;

//configure current node deatils for user

printf("Enter current node:");

scanf("%d", &node); // can recieve queue name also and store in a common file in form of (key, value) or map -- nodeno, queuename

printf("Enter num of adj nodes:");

scanf("%d", &n);

for(int i =0 ; i<n ; i++)

{

printf("Enter cost of adj node identifier and cost of a node:");

scanf("%d %d", &adj, &cost);

//printf("Enter the queue name identifier for the node (if not maintained as DB) :");

//scanf("%s", queuename[i]);

updategraph(node, adj, cost);

// can update adjacency list here

}

t1 = pthread\_create(&queue\_events,NULL, NULL);

// send updated msgqueue buffer to all adj nodes for updation

// retrieve the queuename from the DB(stored as map) based on adj node or read along with node data

struct msg\_buffer \*msg;

for(int i =0; i<n; i++)

{

sem\_wait(s1);

if(graphmatrix[node][i] != 0)

msg = malloc(sizeof(strct msg\_buffer));

q = msg\_open("\name\_node[i]");

msg->sending\_node = node;

msg->adj\_cost = graphmatrix[node]; //syntax wrong --- copy properly all costs

sem\_signal(s1);

}

pthread\_join(t1);

queue\_close(q);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* utility thread to process the events received from other threads\*\*\*\*\*\*\*\*\*/

void queue\_events(void \*param)

{

// wait for events in queue

struct queuebuffer \*msg = NULL;

msg\_open(q);

while(!msg\_empty(q))

{

msg = msg\_rcv(q);

for(i=0; i<total\_nodes; i++)

{

updategraph(msg->sending\_node, i, msg->cost[i]);

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* implementation dijkstras algo (need to take input from user) and the path returned in path array\*\*\*\*\*\*\*/

int shortestpath(int s, int d, int total\_nodes,int path[])

{

//Dijkstras algorithm (use min heap for better complexity)

int dist[50];

int predecessor[50];

int node;

lbl label[50];

int i;

int min = INF;

for(int i=0; i<total\_nodes; i++)

{

label[i] = white;

dist[i] = INF;

predecessor[i] = -1;

}

dist[s] = 0;

sem\_wait(s1); // since we are reading the graphmatrix , prevent update in between

while(s!= d)

{

for(int i =0 ; i< total\_nodes; i++) //better use adjacency list implementation or else have to travese O(Vertices)

{

if(graphmatrix[node][i] != 0 && label[i] == white)

{

if(dist[i] < dist[node] + graphmatrix[node][i])

{

dist[i] = dist[node] + graphmatrix[node][i];

predecessor[i] = node;

}

}

}

// find node with least distance in graph

for(int i =0 ; i< total\_nodes; i++)

{

if(dist[i] < min)

{

min = dist[i];

node = i;

}

}

label[node] = black;

}

sem\_signal(s1); //release the lock

// predecessor array gives the predecessor node to reach through shortest path

while(node != s)

{

path[i++] = node;

node = predecessor[node];

}

//reverse path

}

/\*\*\*\*\*\*\*\*\*\*\*\* Need to implement prisms algo to get mst\*\*\*\*\*\*\*\*\*\*\*\*/

int minSpanningTree()

{

// Use prims algorithm

}