Problem Set Graph Algorithms

Technical Design Document

By Rakesh Nangunuri

TABLE OF CONTENTS

1 INTRODUCTION

#1 This section should provide an overview of the problem set

1.1 PROBLEM STATEMENT

Given a set of interacting processes that exchange neighbour adjacency data. The process works like this:

- *Each process reads neighbor information and cost from a file*
- *Each process builds a network graph from all nodes using this neighbor information*

Given this information , we have to do the below :

- *Find the shortest path from any to any node given.*
- *Find the minimum spanning tree.*
- *Also, reconfigure the network connectivity graph on a node when a node (process) dies or an edge is lost*

1.2 APPROACH

#1 The approach followed is by event triggering and handling the events by functions. We start the design by creating two process. One of them acts as a client and another as server. Both the servers hold on the select system call so that if any event happens on the descriptors the client or server gets notified.

#2 In server side the nodes and edges are read from file and a graph is built. Then the server spawns 2 threads named "shortest_path_finder" and "Minimum_spanning_tree". The two threads waits on the signal for any changes in node and edge data from file. If any node death or edge break down happens then this threads get executed and finds shortest path and Minimum spanning tree.

#3 Then the server hangs on select call for any descriptor changes. The descriptor can be a :

- *file descriptor which tells if node data or edge data change.(During UT we modify this file to give inputs i.e node death or edge break down)*
- *socket descriptor which gets information from remote process which acts as client.*

2 DESIGN

#1 This section should briefly introduce the system design, and discuss the background to the problem set.

2.1 SYSTEM ARCHITECTURE

#1 The architecture designed for this problem set is Client Server model.

#2 The Server will read the node data from the file and build the graph. Then for the first time it creates two threads “shortest_path” and “Minimum_spanning_tree” which waits on a signal for calculating shortest path and Minimum spanning tree.

#3 This threads are signalled by the action code after select system call.

#4 So the select system call scans for the descriptors and when an event occurred on the descriptors the call becomes unblocked. After unblocking based on what type of descriptor the action will takeout. This can be seen in flowchart below in next chapter. So if descriptor is file descriptor then local node graph will be rebuilt. If the descriptor is socket descriptor then filedata from remote process is read and the graph is rebuilt.

2.2 DATA STRUCTURES

#1 Each node is defined with following data structure:

```
typedef struct Node_t  {
    char node_data;
    struct Node_t *    next;
    }* Node;
```

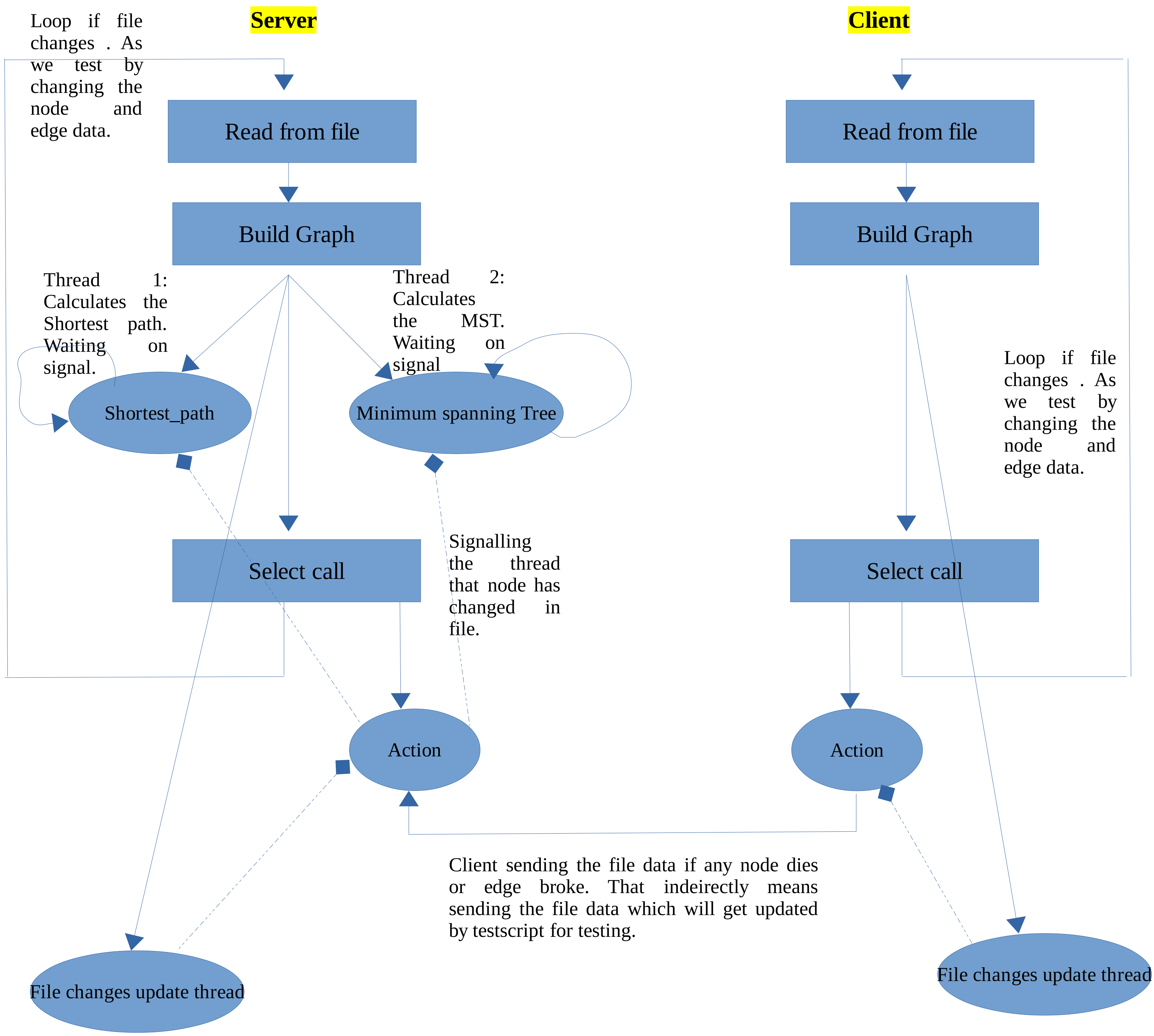
#2 Message from the remote client :

```
typedef struct Remote_Mess_t  {
        int len_of_mess;                //For len of message each time from client
        char * filedata;
        char tag;                   /* “R”  this implies remote, for distingushing
                                    between remote and local message in server*/
  } * Remote_Mess;
```

#3 Data format in file is:

Ex:A,0,A-B,1 : where A is data. 0 is node is active or dead(dead here). A-B is link . 1 is the status of the link (active here).

2.3 FLOW CHART



3 UNIT TESTING

#1 This section should define the testcases required to test.

3.1 TESTING WITH FILE UPDATE

The file from which client or server loads the node data helps us in testing the code.

The node , status , edge and edge weight is the format which we follow to specify the node specific data in file. Ex: A,0,A-B,1 means A is dead.

So we try to manually change the data or through the testscript in this file. When the file is changed the file descriptor in select call gets signalled and we proceed with reconfiguring the node graph.

4 OTHER MISCELLANEOUS

4.1 CORNER CASES

#1 During this if a process gets killed then there should be a mechanism to save context and restart the process with same context.

#2 As accessing the files is slow process it will effect the performance . So we should implement cache type of memory mechanism to store this type of node data.

4.2 EFFICIENCY IMPROVEMENT

#1 As we are taking data every time from file for node changes, the accessing file creates latency in the program . Hence it is recommended to save node data in shared-memory and update the changes.

#2 Through select call we achieve synchronous method of synchronization . The program will be effective if there is asynchronous method of synchronization.