This short design document describes some key implmentation mechanism in my design:

1.UDP Socket

UDP socket is applied in my design, can support multiple clients at the same time,and

This connectionless model avoids the multiple threaded Server

2.server:Single threaded

Although the server is single threaded, it can communicate with multiple clients at the same time.It avoids complicated syncronization in multithreaded application.

3.Directed Graph

My design use one directed graph to describe the order of multiple events,one vertex represent one event.

3.1 “insert”

If the user input “insert A->B” ,one edge from ‘A’ to ‘B’ is added to the Graph.

To avoid a circle, when add edge from ‘A’ to ‘B’, the “reachbility” from ‘B’ to ‘A’ is checked, if ‘B’ can reach ‘A’, this insertion is prohbited. BFS is used to judge the reachbility.

3.2”query”

First check if the vertices exist? If no, return “Event not found”.

Then return the reachbility of two vertices. If event “A” can reach “B”,return “A happen before B”.If “A” can not reach “B”,return “A current to B”.

3.3. “reset”

Remove all edges in Graph, but keep all the vertices.

4.how to end a client

Ctrl + D can end a client.

5.redirect server’s output to one log file

./dec\_server –l log.txt

Some requirements:

1.When input at client side, not add extra spaces between “;”and “insert”,”query” or “reset”.

Below is not acceptable:

*insert A->B B->C; ˽insert C->D; ˽query A D; ˽reset;*

Below is valid:

*insert A->B B->C; insert C->D; query A D; ˽reset;*

some limitations:

1.only support 26 Events:’A’ to ‘Z’

For simplicity, this implementation only supprt 26 Events:’A’ to ‘Z’

README:

1.how to compile:

#make

2.how to clean binary:

#make clean

3.how to startup:

3.1 run server

#./dec\_server

3.2 on the same machine run

#./dec\_client

or on the another machine run

#./dec\_client -s <iphost address of server>

4.you can starup multiple clients at the same time, the server is one single thread UDP server

5.something need to take care

5.1 no spaces should exist between “;”and “insert”,”query” or “reset”.

Below is not acceptable:

insert A->B B->C; insert C->D; query A D; reset;

Below is valid:

insert A->B B->C;insert C->D;query A D;reset;

5.2 only support 26 Events:A,B,....Z.

Source code:

parameter.h

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

/\* Global constants \*/

#define ECHO\_PORT (9090)

#define MAX\_LINE (1000)

minilog.h

#ifndef \_LOG\_H\_

#define \_LOG\_H\_

#include <stdarg.h>

void minilog ( const char \* format, ... );

void open\_log(char \*logFile);

void close\_log();

void enable\_debug();

#endif /\*\_LOG\_H\_\*/

graph.h

#ifndef \_\_GRAPH\_H\_

#define \_\_GRAPH\_H\_

#include <list>

using namespace std;

// This class represents a directed graph using adjacency list representation

class Graph

{

int V; // No. of vertices

int \*Vertex\_Valid;

list<int> \*adj; // Pointer to an array containing adjacency lists

public:

Graph(int V); // Constructor

void addEdge(int v, int w); // function to add an edge to graph

bool VertexExist(int v);

bool isReachable(int s, int d); // returns true if there is a path from s to d

void reset();

~Graph();

};

#endif

analyze.h

#ifndef \_\_ANALYZE\_H\_

#define \_\_ANALYZE\_H\_

void analyze\_response(char\* buffer,char\* txbuffer);

#endif

--------------------------------------------------------------------

client.c

­­­­­­­­­­­­­­­­­­\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#include <sys/socket.h>

#include <sys/types.h>

#include <netinet/in.h>

#include <netdb.h>

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <unistd.h>

#include <errno.h>

#include <arpa/inet.h>

#include "parameter.h"

static void display\_usage()

{

printf("dec\_client [-h] [-s server-host] [-p port-number]\n");

printf("-h output online help\n");

printf("-s host name or ip address of server\n");

printf("-p listening port of server\n");

return;

}

int main(int argc, char \*argv[])

{

short int port=ECHO\_PORT;

int sockfd = 0;

char buf[MAX\_LINE];

int bytes;

struct sockaddr\_in serv\_addr;

socklen\_t serveraddrlen = sizeof(serv\_addr);

fd\_set ready;

int done=0;

int c;

char \*endptr;

char \*server\_host ="localhost";

struct hostent \*hp;

while ((c = getopt (argc, argv, "hs:p:")) != -1)

{

switch(c)

{

case 'p':

port = strtol(optarg,&endptr,0);

if(\*endptr){

printf("Invalid port number.\n");

exit(0);

}

break;

case 's':

server\_host = optarg;

break;

case 'h':

display\_usage();

exit(0);

break;

default:

display\_usage();

exit(0);

}

}

if ((hp = gethostbyname(server\_host)) == NULL) {

printf("%s unknown host\n", server\_host);

exit(1);

}

/\*if(argc != 2)

{

printf("\n Usage: %s <ip of server> \n",argv[0]);

return 1;

}\*/

memset(buf, '0',sizeof(buf));

if((sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0)

{

printf("\n Error : Could not create socket \n");

return 1;

}

memset(&serv\_addr, 0, sizeof(serv\_addr));

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_port = htons(port);

memcpy(&serv\_addr.sin\_addr, hp->h\_addr, hp->h\_length);

while(!done)

{

FD\_ZERO(&ready);

FD\_SET(sockfd,&ready);

FD\_SET(fileno(stdin), &ready);

if (select((sockfd + 1), &ready, 0, 0, 0) < 0) {

perror("select");

exit(1);

}

if (FD\_ISSET(fileno(stdin), &ready)) {

if ((bytes = read(fileno(stdin), buf, MAX\_LINE)) <= 0)

done++;

else

{

sendto(sockfd, buf, bytes, 0,(struct sockaddr \*)&serv\_addr,

sizeof(serv\_addr));

}

}

if (FD\_ISSET(sockfd, &ready)) {

if ((bytes = recvfrom(sockfd, buf, MAX\_LINE,0,

(struct sockaddr \*)&serv\_addr,

& serveraddrlen)) <= 0)

{

done++;

}

else

{

fprintf(stderr,"response from server:");

write(fileno(stdout), buf, bytes);

}

}

}

close(sockfd);

return 0;

}

server.c

#include <sys/socket.h> /\* socket definitions \*/

#include <sys/types.h> /\* socket types \*/

#include <arpa/inet.h> /\* inet (3) funtions \*/

#include <unistd.h> /\* misc. UNIX functions \*/

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include "analyze.h"

#include "parameter.h"

#include "minilog.h"

static void display\_usage()

{

printf("decserver [-h] [-l logfile] [-p port-number]\n");

printf("-h output online help\n");

printf("-l logfile\n");

printf("-p server listening port\n");

return;

}

int main(int argc, char \*argv[]) {

int list\_s; /\* listening socket \*/

short int port=ECHO\_PORT; /\* port number \*/

struct sockaddr\_in servaddr; /\* socket address structure \*/

char buffer[MAX\_LINE]; /\* character buffer \*/

char txbuffer[MAX\_LINE];

char \*endptr; /\* for strtol() \*/

struct sockaddr\_in msgfrom;

socklen\_t msgsize = sizeof(msgfrom);

union {

uint32\_t addr;

char bytes[4];

} fromaddr;

int n=0;

int c;

/\* Get port number from the command line, and

set to default port if no arguments were supplied \*/

while ((c = getopt (argc, argv, "hl:p:")) != -1)

{

switch(c)

{

case 'p':

port = strtol(optarg,&endptr,0);

if(\*endptr){

printf("Invalid port number.\n");

exit(0);

}

break;

case 'l':

open\_log(optarg);

break;

case 'h':

display\_usage();

exit(0);

break;

default:

display\_usage();

exit(0);

}

}

/\*if ( argc == 2 ) {

port = strtol(argv[1], &endptr, 0);

if ( \*endptr ) {

fprintf(stderr, "ECHOSERV: Invalid port number.\n");

exit(EXIT\_FAILURE);

}

}

else if ( argc < 2 ) {

port = ECHO\_PORT;

}

else {

fprintf(stderr, "ECHOSERV: Invalid arguments.\n");

exit(EXIT\_FAILURE);

}\*/

/\* Create the listening socket \*/

if ( (list\_s = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0 ) {

fprintf(stderr, "ECHOSERV: Error creating listening socket.\n");

exit(EXIT\_FAILURE);

}

/\* Set all bytes in socket address structure to

zero, and fill in the relevant data members \*/

memset(&servaddr, 0, sizeof(servaddr));

servaddr.sin\_family = AF\_INET;

servaddr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

servaddr.sin\_port = htons(port);

/\* Bind our socket addresss to the

listening socket, and call listen() \*/

if ( bind(list\_s, (struct sockaddr \*) &servaddr, sizeof(servaddr)) < 0 ) {

fprintf(stderr, "ECHOSERV: Error calling bind()\n");

exit(EXIT\_FAILURE);

}

/\* Enter an infinite loop to respond

to client requests and echo input \*/

/\* Retrieve an input line from the connected socket

then simply write it back to the same socket. \*/

while(1)

{

memset(buffer,0,sizeof(buffer));

n = recvfrom(list\_s, buffer, MAX\_LINE-1, 0, (struct sockaddr \*)&msgfrom, &msgsize);

if(n>0)

{

fromaddr.addr = ntohl(msgfrom.sin\_addr.s\_addr);

minilog("request received from %d.%d.%d.%d: ",

0xff & (unsigned int)fromaddr.bytes[3],

0xff & (unsigned int)fromaddr.bytes[2],

0xff & (unsigned int)fromaddr.bytes[1],

0xff & (unsigned int)fromaddr.bytes[0]);

memset(txbuffer,0,sizeof(txbuffer));

analyze\_response(buffer,txbuffer);

if(strlen(txbuffer))

{

txbuffer[strlen(txbuffer)]='\n';

}

sendto(list\_s, txbuffer, strlen(txbuffer),0,(struct sockaddr \*)&msgfrom, sizeof(msgfrom));

}

}

/\* Close the socket \*/

if ( close(list\_s) < 0 ) {

fprintf(stderr, "ECHOSERV: Error calling close()\n");

exit(EXIT\_FAILURE);

}

}

analyze.cpp

#include "graph.h"

#include "analyze.h"

#include "minilog.h"

#define IS\_CAP(a) ((a>='A')&&(a<='Z'))

#define IS\_SPAC(a) (a>=' ')

Graph g(26);

void analyze\_response(char\* buffer,char\* txbuffer)

{

char \*cmd;

char \* cmdsavedptr,\*subcmdsavedptr;

char \*insert\_pair;

char begin,end;

minilog("%s",buffer);

cmd = strtok\_r (buffer,";",&cmdsavedptr);

while (cmd != NULL)

{

//printf ("%s\n",cmd);

if(0== strncmp(cmd,"insert",6))

{

//printf("request received cmd %s\n",cmd);

{

insert\_pair = strtok\_r(cmd," ",&subcmdsavedptr);

while(NULL != insert\_pair)

{

insert\_pair= strtok\_r(NULL, " ",&subcmdsavedptr);

if(NULL != insert\_pair)

{

if((4==strlen(insert\_pair))&&('-'==insert\_pair[1])&&('>'==insert\_pair[2]))

{

begin=insert\_pair[0];

end = insert\_pair[3];

if(IS\_CAP(begin)&&IS\_CAP(end)&&(begin!=end))

{

begin = begin-'A';

end = end - 'A';

if(g.isReachable(end,begin))

{

sprintf(txbuffer+strlen(txbuffer),"CONFLICT DETECTEDINSERT FAILED ");

}

else

{

g.addEdge(begin,end);

sprintf(txbuffer+strlen(txbuffer),"INSERT DONE.");

minilog("INSERT DONE.\n");

}

}

else

{

sprintf(txbuffer+strlen(txbuffer),"UNSUPPORTED CMD ");

}

}

else

{

sprintf(txbuffer+strlen(txbuffer),"UNSUPPORTED CMD ");

}

}

}

}

}

else if(0== strncmp(cmd,"query",5))

{

//printf("request received cmd %s\n",cmd);

{

if(9==strlen(cmd))

{

if(IS\_SPAC(cmd[5])&&IS\_SPAC(cmd[7]))

{

begin=cmd[6];

end= cmd[8];

if(IS\_CAP(begin)&&IS\_CAP(end)&&(begin!=end))

{

begin = begin-'A';

end = end - 'A';

if(!g.VertexExist(begin))

{

sprintf(txbuffer+strlen(txbuffer),"Event not found: %c",begin+'A');

}

else if(!g.VertexExist(end))

{

sprintf(txbuffer+strlen(txbuffer),"Event not found: %c",end+'A');

}

else

{

if (g.isReachable(begin,end))

{

sprintf(txbuffer+strlen(txbuffer),"%c happened before %c",begin+'A',end+'A');

minilog("%c happened before %c\n",begin+'A',end+'A');

}

else if(g.isReachable(end,begin))

{

sprintf(txbuffer+strlen(txbuffer),"%c happened before %c",end+'A',begin+'A');

minilog("%c happened before %c\n",end+'A',begin+'A');

}

else

{

sprintf(txbuffer+strlen(txbuffer),"%c current to %c",begin+'A',end+'A');

minilog("%c current to %c\n",begin+'A',end+'A');

}

}

}

else

{

sprintf(txbuffer+strlen(txbuffer),"UNSUPPORTED CMD ");

}

}

else

{

sprintf(txbuffer+strlen(txbuffer),"UNSUPPORTED CMD ");

}

}

else

{

sprintf(txbuffer+strlen(txbuffer),"UNSUPPORTED CMD ");

}

}

}

else if(0== strncmp(cmd,"reset",5))

{

//printf("cmd reset\n");

g.reset();

sprintf(txbuffer+strlen(txbuffer),"RESET DONE.");

minilog("RESET DONE.\n");

}

cmd = strtok\_r (NULL, ";",&cmdsavedptr);

}

}

-----------------------------------------------------

graph.cpp

#include "graph.h"

Graph::Graph(int V){

this->V = V;

Vertex\_Valid = new int[V];

adj = new list<int>[V];}

void Graph::addEdge(int v, int w)

{

Vertex\_Valid[v]=1;

Vertex\_Valid[w]=1;

adj[v].push\_back(w); // Add w to v’s list.

}

bool Graph::VertexExist(int v)

{

return (1==Vertex\_Valid[v]);

}

// A BFS based function to check whether d is reachable from s.

bool Graph::isReachable(int s, int d)

{

// Base case

if (s == d)

return true;

// Mark all the vertices as not visited

bool \*visited = new bool[V];

for (int i = 1; i <V; i++)

visited[i] = false;

// Create a queue for BFS

list<int> queue;

// Mark the current node as visited and enqueue it

visited[s] = true;

queue.push\_back(s); // it will be used to get all adjacent vertices of a vertex

list<int>::iterator i;

while (!queue.empty()) {

// Dequeue a vertex from queue and print it

s = queue.front();

queue.pop\_front();

// Get all adjacent vertices of the dequeued vertex s

// If a adjacent has not been visited, then mark it visited

//and enqueue it

for (i = adj[s].begin(); i != adj[s].end(); ++i)

{

// If this adjacent node is the destination node, then return true

if (\*i == d)

return true;

// Else, continue to do BFS

if (!visited[\*i])

{

visited[\*i] = true;

queue.push\_back(\*i);

}

}

}

return false;

}

void Graph::reset()

{

if( adj )

{

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

{

adj[i].clear();

}

}

}

Graph::~Graph()

{

if(Vertex\_Valid)

delete[] Vertex\_Valid;

if( adj )

{

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

adj[i].clear();

delete[] adj;

}

}

-------------------------------------------------

minilog.c

#include <stdio.h>

#include <stdlib.h>

#include <stdarg.h>

#include <unistd.h>

#include "minilog.h"

FILE \*stream=NULL;

void minilog ( const char \* format, ... )

{

va\_list args;

if(NULL == stream)

{

va\_start (args, format);

vprintf (format, args);

va\_end (args);

}

else

{

// output log file

va\_start (args, format);

vfprintf(stream,format,args);

fflush(stream);

fsync(fileno(stream));

va\_end (args);

}

return;

}

void open\_log(char \*logFile)

{

stream = fopen(logFile,"w");

if (NULL == stream)

{

printf("open log file %s fail\n",logFile);

exit(0);

}

}

void close\_log()

{

if(NULL != stream)

{

fclose(stream);

}

}