CSE 344 FINAL PROJECT REPORT

EMRE SEZER
1901042640

Client:

First, Client reads the requestFile as the name of the file is given as an argument input. It stores each line in a 2d char array. For each line a thread is created. Each thread, creates its socket and waits for the broadcast() function to be called inside a monitor.

When, it is called each thread starts to begin communicating with the server through the socket. Threads put ‘c’ to the end of the lines read from the requestFile. This string is sent to the server.

After writing to the server, each thread waits for server to the respond back to the client. When it receives the response from the server. Ir prints the response message and ends. After creating threads, main process waits for the threads to end. When threads are ended, it frees all of the dynamically allocated space.

As the thread variable thread id is givent to the each thread. Response from server includes the response given to server from the servant.

Server:

Process creates required variables, SIGINT Signal handler, mainThread, poolThreads. It waits for the threads to end then, it frees all dynamically allocated space.
MainThread creates main socket to communicate with the client and servants through the port. Port is given as an input.

When it creates the socket it goes in an infinite loop then accepts a new socket and adds that socket to the queue. Adding operation happens inside a mutex then sends cond_broadcast to all poolThreads. poolThreads wait at the cond_wait.

poolThreads are always in an infinite loop that waits in a monitor. When each poolThread gets awakened it gets and removes from the queue. This is socketID. Then, reads what is sent to buffer. If last element of buffer is ‘c’ that means message is came from the client. Splits buffer to 4 or 5 elements. If that is 5 city is specified, if that is 4 city is not specified.
If city is specified finds the proper city has that city. There may not be any servant that looks to that city, in that case writes “no_servant” to the buffer and sends it to client. If there was a servant then poolThread communicates with the servant using its socket. It tries to connect until it happens. Then, sends clients request with that socket. Then waits for the response from the servant, sends that response to the client and goes to the monitor.

If city is not specified poolThreads, sends the message to all of the servants and waits for the response from each of them. If last element of buffer is ‘s’ message came from Servant. In that case locks mutexServant, allocates new ServantData struct and reads servants uniquePort, ip numberOfCities, pid and cities. Each Servant data is stored inside the ServantData struct. Unlocks mutexServant and goes to the monitor for waiting.

mainThread has SIGUSR1 signal handler. When process gets SIGINT signal, in SIGINTHANDLER function SIGINTHappened becomes 1 and sends SIGUSR1 to the mainThread . When mainThread gets SIGUSR1 sends SIGUSR1 to poolThreads to inform them and make them exit. Also, before exiting it sends SIGINT to all of the servants. It sends SIGINT to servant with servantData.pid for each servant.

ServantData is a struct for keepin data of each servant required for operations of server. Its content can be viewed below.

Queue is a struct for keeping socket ids. These socket ids are for poolThreads to be used. Each poolThread tries to call dequeue function and uses that socket id to read from socket. I said try, because synchronization is provided with the monitor.

```
typedef struct ServantData
{
    int uniquePort;
    char ** cities;
    int numberOfCities;
    char ip[30];
    int pid;
}ServantData;
```

```
typedef struct node
{
    int item;
    int index;
    struct node* next;
}node;

typedef struct Queue
{
    node* front;
    node* rear;
}Queue;
```

Servant:

Process finds process id through “/proc/self/stat” path. Defines its unique port according to lower limit and upper limit argument .
For example 23-31. In Microsoft Teams it is said that lower and upper limits won’t overlap. So, all of them will be unique. I add lower limit to 16000 and made a unique port. If there is an overlap with port argument it checks and add 2 to the uniquePort. Then through scandir() i get all of the directories inside the dataset and store the required ones in a Linked List according to lower limit and upper limit argument. When all of them are ready i prepare a buffer which includes uniquePort, ip and cities. Then, send it to server. Then, Create a thread, wait for it to end and free all dynamically allocated space.

servantThread in an infinite loop creates new socket with its own unique port and waits for the server to send client request. Reads, spkits it and if city is specified looks into that city path and starts sending filenames inside that city paths files. It uses checkTime() function, if it returns 1 it calls checkPath() function. Return value is written into the result buffer. servantThread sends that buffer to the server through the socket. If city is not specified then does all of that to all cities that servant is responsible for and returns sum of these results. As at the server, if servant process receives a SIGINT, sends SIGUSR1 to servantThread and end the thread.

I wrote a linked list data structure for keeping city names. You can see the detail below.

```
typedef struct node
{
    char data[30];
    struct node * next;
}node;

typedef struct LinkedList
{
    node * head;
}LinkedList;
```

HELPER FUNCTIONS FROM servant.h:

checkTime():
This function takes 3 parameters as input. These 3 parameters are 3 dates. It checks if the third date is between first and second dates. If so, it returns 1, else 0.

checkPath():
This function which checks if that desired file contains any lines that contains type input and returns how many are there.

fileNameSelect:
Uses scandir to ignore system files starting with '.'