CSE 344 – 2022 System Programming

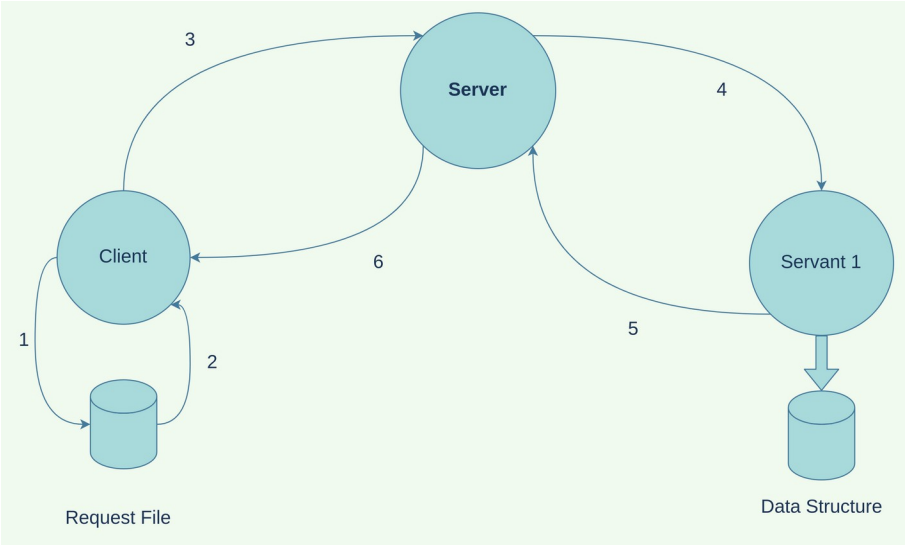
Final Project Report

Muhammed Bedir ULUCAY

1901042697

Problem:

The task wanted to be done by us. Implement three different processes (client, server, servant) these processes' tasks will be like below.



1. Read request line from the request file
2. Get line parse it and prepare to valid structure to send server
3. Get request from the client
4. Find related servant about getting request and prepare for sending this request to servant
5. Get the match number in data which we keep in the servant.
6. Forward getting result to client's thread.

Client:

Read the request File line by line and create a thread for each request. Then send these requests to the server process and then wait for the response from the server. The wanted thing from the server number of match data in the data sets. After the server forwards the response to the client's thread per request the thread will exit. Threads get their response the client will be terminated.

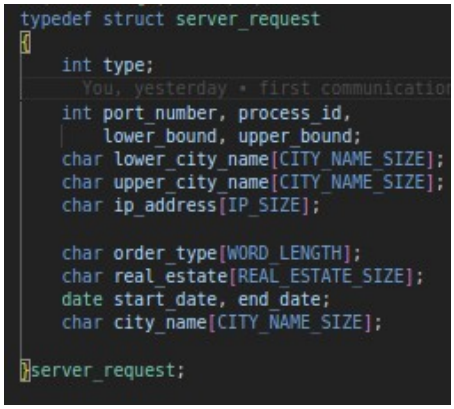
Example request;

- Column 0: Always same for our assigned task. It's wanted the match count in dataset.
- Column 1: Real Estate type of the property.
- Column 2: Start date for the searching.
- Column 3: End date for the search.
- Column 4: City name for search if this column is blank that means. You need to search all the cities in the dataset.

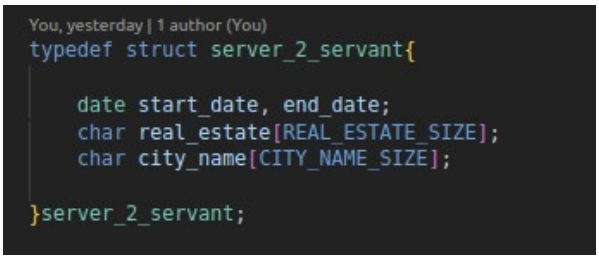
Server:

Firstly, create a thread pool to get requests in the wait condition to get requests from the servant and client process. There are 2 types of requests for the server. The first one is the coming from the servant process. Another one comes from the client process.

Servant sends its properties first like unique new port number, process id, and boundaries. In the beginning server and servant communicate at the same port with the client but when the servant sends the unique port number it will create a new socket for each servant's different port and communicate over this unique port. After getting the servant boundaries store them in a variable and when a client request arrives check for the Is there is a valid servant for the client request. If there is a match in boundaries or no city name in the request this request getting and preparing to send the servant in the server_2_servant variable. Then the servant takes the request and one of the server's thread waits for the response that will come from the servant process. After getting the response from the servant process. It will forward this request to the client's related thread. And then the server's thread waits for another request on the requested port.

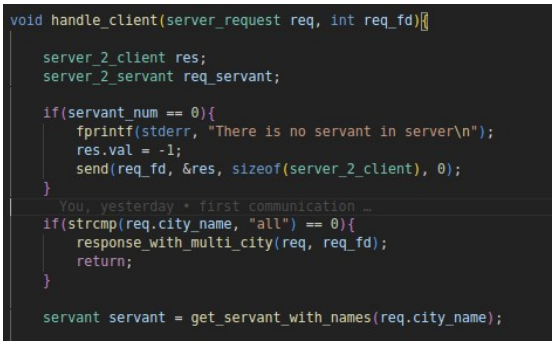


```c
typedef struct server_request
{
    int type;
      You, yesterday • first communicatio
    int port_number, process_id,
        lower_bound, upper_bound;
    char lower_city_name[CITY_NAME_SIZE];
    char upper_city_name[CITY_NAME_SIZE];
    char ip_address[IP_SIZE];

    char order_type[WORD_LENGTH];
    char real_estate[REAL_ESTATE_SIZE];
    date start_date, end_date;
    char city_name[CITY_NAME_SIZE];

}server_request;
```



```c
You, yesterday | 1 author (You)
typedef struct server_2_servant{

    date start_date, end_date;
    char real_estate[REAL_ESTATE_SIZE];
    char city_name[CITY_NAME_SIZE];

}server_2_servant;
```

```c
void handle_request(int req_fd){

    server_request req;
    memset(&req, 0, sizeof(server_request));
    recv(req_fd, &req, sizeof(server_request), 0);

    pthread_mutex_lock(&global_mutex);
    (req.type == 0) ? servant_counter++ : client_counter++;
    pthread_mutex_unlock(&global_mutex);

    if(req.type == 0){
        create_servant_connection(req);
    }      You, last week • [ADD]dir …
    else if(req.type == 1){
        handle_client(req, req_fd);
    }
}
```



```c
void handle_client(server_request req, int req_fd){

    server_2_client res;
    server_2_servant req_servant;

    if(servant_num == 0){
        fprintf(stderr, "There is no servant in server\n");
        res.val = -1;
        send(req_fd, &res, sizeof(server_2_client), 0);
    }
      You, yesterday • first communication …
    if(strcmp(req.city_name, "all") == 0){
        response_with_multi_city(req, req_fd);
        return;
    }

    servant servant = get_servant_with_names(req.city_name);
```

```c
void create_servant_connection(server_request req){

    servant s;
    s.port_number = req.port_number;
    s.lower_bound = req.lower_bound;
    s.upper_bound = req.upper_bound;      You, 5 days ago • servant properties …
    s.process_id = req.process_id;
    strcpy(s.lower_city_name, req.lower_city_name);
    strcpy(s.upper_city_name, req.upper_city_name);
    strcpy(s.ip_address, req.ip_address);

    fprintf(stdout, "Servant %d present at port %d handling cities %s-%s\n", s.process_id, s.port_number, s.lower_city_name, s.upper_city_name);

    pthread_mutex_lock(&global_mutex);
    servants[servant_num++] = s;
    pthread_mutex_unlock(&global_mutex);

}
```

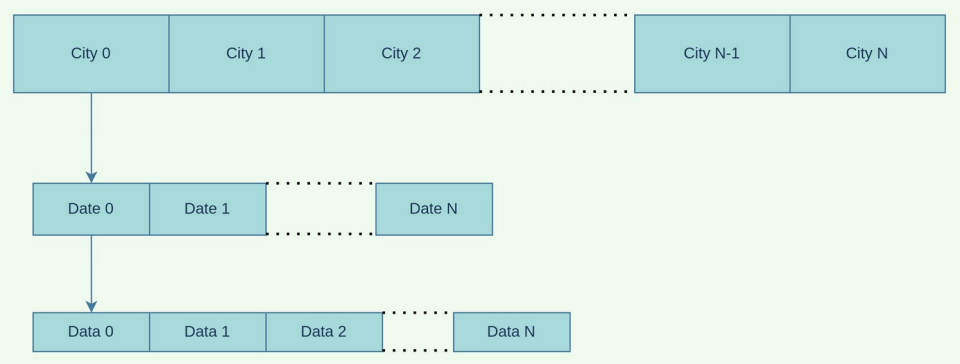
Servant:

Servant process reading related data in data folder according to its border values which are getting as an argument. Making a client connection for the server to send its unique properties. And sending it port address, Ip address, boundaries etc. Then create a server connection of its own in the unique port number. And wait for the request to come from the server. In the servant process, there is no thread pool to get requests instead of creating a thread for each request. In the request, the thread calculates the number of matches and returns the match number to the server.

```c
void* handle_client(void* arg){

    int client_fd = *(int*)arg;
    free(arg);

    server_2_servant s2s;
    recv(client_fd, &s2s, sizeof(server_2_servant), 0);

    fprintf(stdout, "Received request from client: %s\n", s2s.city_name);
    int result = calculate_transaction(cities, responsible_directories_counter, s2s);
      You, 2 days ago • start of base connection between 3 process
    fprintf(stdout, "Servant is sending %d\n", result);
    send(client_fd, &result, sizeof(int), 0);

    pthread_exit(NULL);
}
```

Data Structure for storing data:

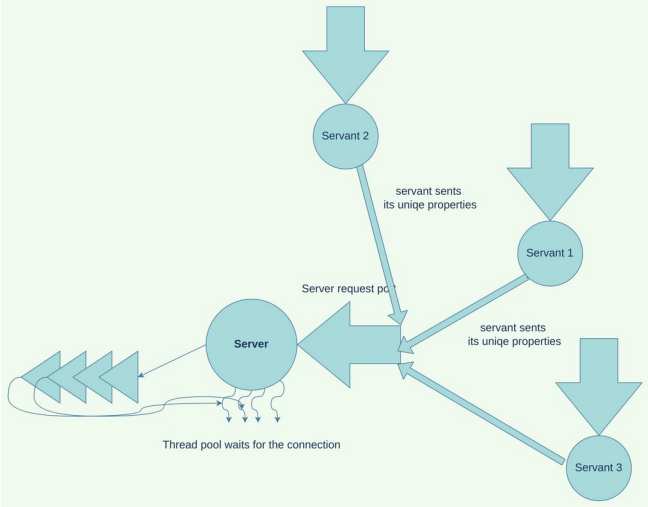
It is like a hybrid list. In the main array stores the city name. The first sub-list keeps the date value, and each date list element keeps request in the list.



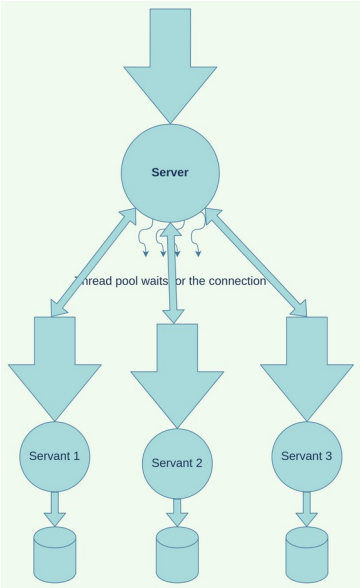
For an efficient search algorithm, we are eliminating the city by its name, then eliminating with the date of the data. Then we are looking for the data in the file. In this way, our search algorithm works much better than just keeping a linear search.

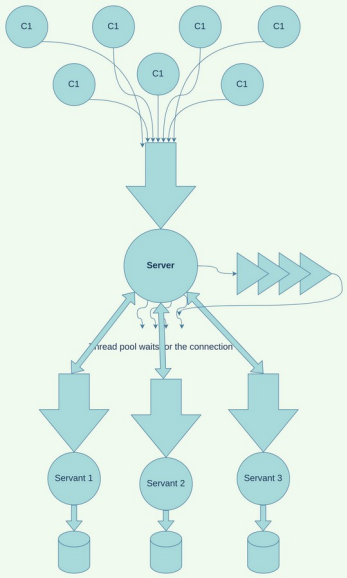
Socket Connection Between Process:

- ./myserver -p $PORT -t 11 &
- ./myservant -d ../dataset -c 1-9 -r 127.0.0.1 -p $PORT &
- ./myservant -d ../dataset -c 10-19 -r 127.0.0.1 -p $PORT &
- ./myservant -d ../dataset -c 20-29 -r 127.0.0.1 -p $PORT &
- ./myclient -r requestFile -q $PORT -s 127.0.0.1 &



The main server creates the main port for the request coming from the main port. Wait for the request then the servant sends its unique properties to open a port.





Create unique port with servants and wait for the client request to answer.

Path of Client Request:

The client reads the request file and creates a thread for each line. And sending the request from the server port. The server gets the request and puts it in the queue of the file descriptor. A valid thread gets for the calculation. The server decides which servant related getting request. So, servant starts being server for the main server. Get request and return the calculated value to main server the main server forwarding this value to client then client thread get the value print it and terminate the thread. After all thread gets the response terminate the whole program.

Muhammed Bedir ULUCAY

1901042697