Final Project System Programming

TARIK KILIÇ - 151044010

June 2020

1 Client Socket

I assign the client socket from the request to the variable which is the type int that I keep globally. Threads in the thread pool can access this global socket. I used mutex because the threads in both the main thread and the pool were accessible. I also used one cond variable. When the main thread receives a new client request, it sends a signal to the threads. One thread gets that client socket.

2 Resizer

For dynamic pool, I used 2 global variables to keep the thread counts idle and busy. One of the threads increases the busy variable when the client receives the socket, and increases the idle variable when finished. In the Resizer thread, I calculate the percentage using these variables. I increase the pool according to him. When I increase the Busy variable, I send the signal to the resizer thread with the special cond variable I hold for the resizer. If 0.75 percent has not exceeded, Resizer wait with condwait.

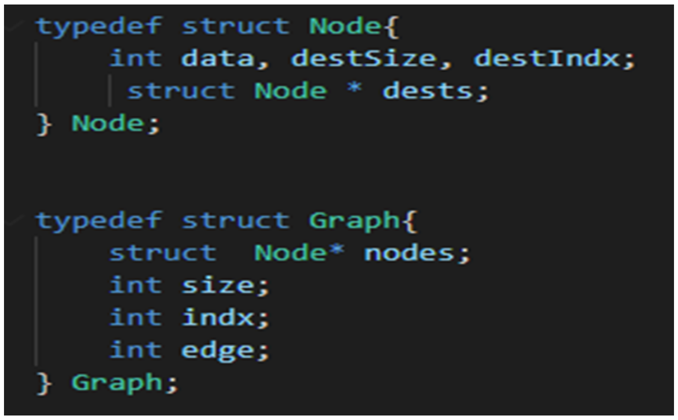
3 Database

All threads can access the database data structure. They both read and write. So when I access the database, I locked. When I finished, unlocked. So that race condition does not occur. No deadlock.

4 My Own Data Structures

The loop is up to number of clients. I calculated the distance from the customer to all florists. Then I returned the florist's index at the shortest distance. Using Mutex and Signals, I put the id of the customer in the request queue of the selected florist and increased the request number.

5 Graph

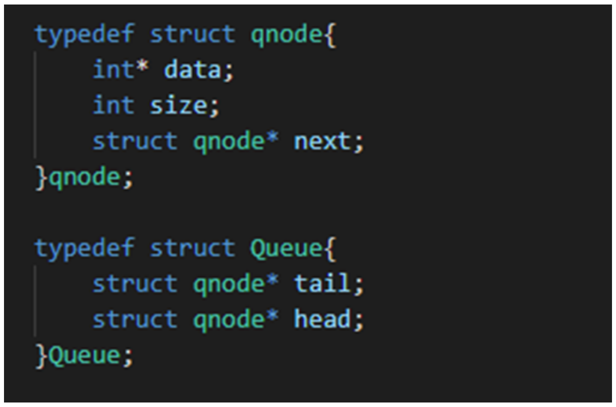


```c
typedef struct Node{
    int data, destSize, destIndx;
     struct Node * dests;
} Node;


typedef struct Graph{
    struct  Node* nodes;
    int size;
    int indx;
    int edge;
} Graph;
```

I have one graph struct. It contains capacity, number of nodes, number of edges and an array in which these nodes are held. In the node struct, there is an array that holds the nodes it is adjacent to. Basically, this data structure is an adjacency list.

6 Queue

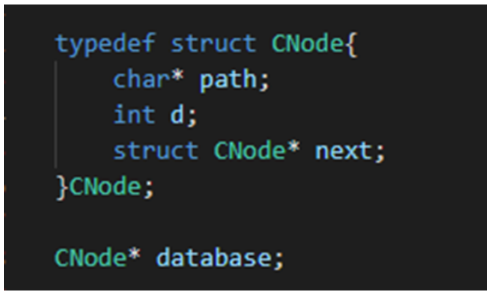


```c
typedef struct qnode{
    int* data;
    int size;
    struct qnode* next;
}qnode;

typedef struct Queue{
    struct qnode* tail;
    struct qnode* head;
}Queue;
```

I used the queue structure by implementing a double linked list. I implement it to use in BFS algorithm.

7 Cache



```
typedef struct CNode{
    char* path;
    int d;
    struct CNode* next;
}CNode;

CNode* database;
```

I was in the database as much as the number of nodes in the CNode type. For quick access, the node and index number are the same. In this node, there are paths that start with that index. If there is a path that starts with the same index, I add end to end in the linked list structure.

8 SIGINT

When I send a signal in the form of Kill -2, I clean all resources and close the threads. There is no leak left.

9 Daemon

The server is working properly in the background.