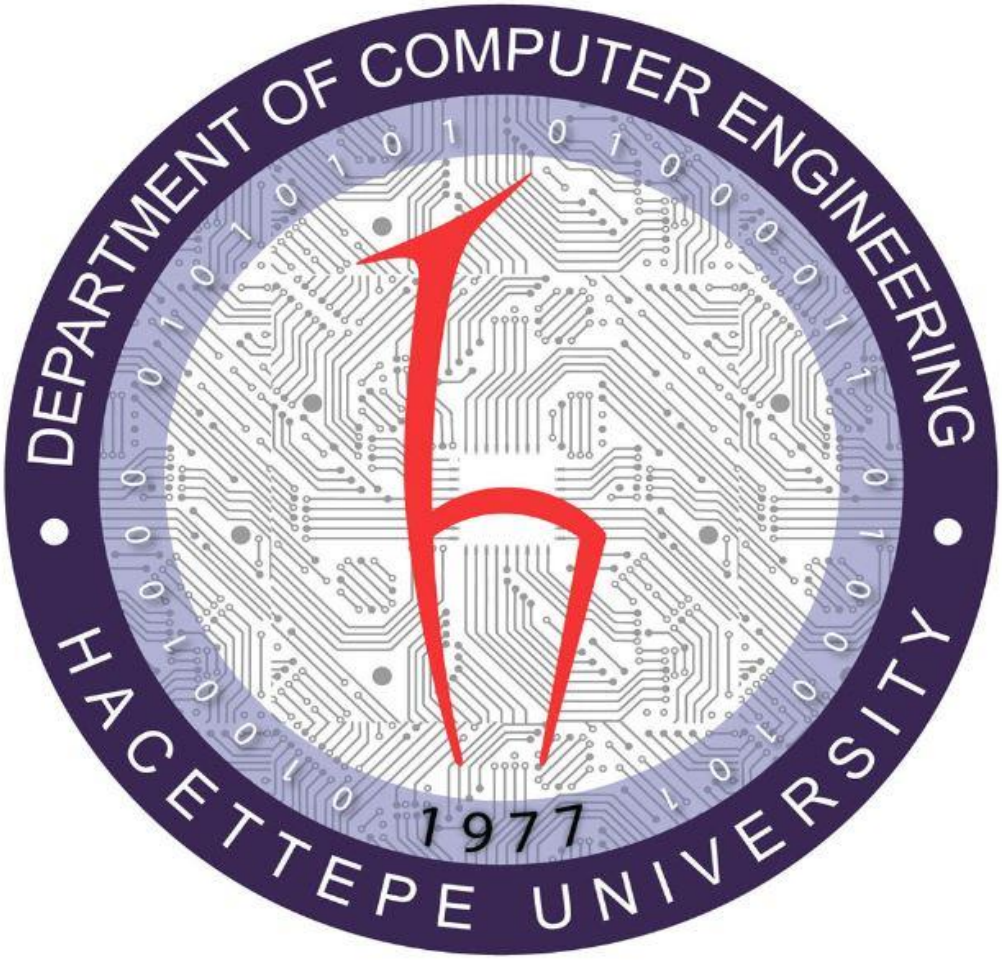

NAME AND SURNAME : BUĞRA TUNCER

SCHOOL NUMBER **:** 21527395

COURSE **:** BBM 203

EXPERIMENT **:** ASSIGNMENT 2

SUBJECT **:** CLIENT-SERVER ARCHITECTURE

ADVISORS **:** R.A. BURÇAK ASAL, DR. BURCU CAN, DR. SEVİL ŞEN, DR ADNAN ÖZSOY

PROGRAMMING LANGUAGE : C

- ASSIGNMENT OVERVIEW

In this assignment, we designed a basic client-server architecture. If i mention about that, users create requests or interrupts on the clients and send it to the servers with the same method. When the process is completed, both clients and server create a log history to show which of these process accomplished.

- ALGORITHM

The first thing i do is thinking how to read given input files and use them more effectively in my code. Then, i read the first input file and store the first line number. Because, it gives us how many server and clients are there existed. Then, i read the other lines char by char and store the values. The values are meaning that clients' and servers' stacks and queues size. At the end of reading first input file i create a struct and dynamic struct array for the store all stacks,queues and also log histories. Then after all these operations i create stack and queue functions.

In the second part i need to read second input file for the given operations. There are four main operations inside this problem:

1. Process Requests : Add a new process to the specific client's queue structure.
2. Interrupt Requests : Add a new interrupt to the specific client's or server's stack structure.

3. Send Operation : Clients' processes and interrupts send to the server, but in this operation interrupts always be sent first.Sent means added on the server's queue. For the each sent and didn't send operations it will transfer these operations to the log history of the clients or stack.
4. Operate Command : Operate command means that server have to put in the operation the interrupts and processes. Also in this case it is way like send operations. Because interrupts need to operated first before the processes. Finally this command send the value to the log history.

At last algorithm print out the log history of each clients and server then the program stop.

- DESCRIPTION OF THE FUNCTIONS

1. ***isStackFull, isStackEmpty*** : These functions check the stack array is full or empty by the help of the stack size and top value.
2. ***Push and Pop*** : Push operation add given character to the stack array and pop operation removes the last element of the stack array.
3. ***Size, isQueueEmpty, isQueueFull*** : Firstly, size operations means the current number of elements in the queue array. The others check the queue array is full or empty with the size function.
4. ***Enqueue and Dequeue*** : Enqueue operation add given character to the queue array and dequeue function deletes the last in character inside queue array.

5. ***createClientServer*** : This function is my main function to the solving of the problem. I have done all of my file operations inside of this. But two code parts are very important.

```
*itemArrayAddress = (struct itemStruct*) malloc((*totalItemAddress)*sizeof(struct itemStruct));
int i;
for(i=0;i<*totalItemAddress;i++)
{
    fscanf(file1,"%d %d",&queueLen,&stackLen);
    (*itemArrayAddress)[i]=(struct itemStruct){-1,stackLen,queueLen,-1,0,0,(char*)malloc(sizeof(char)*queueLen),(char*)malloc(sizeof(char)*stackLen),
    (char*)malloc(sizeof(char)*4*countCommands),0};
}
```

In this part, memory allocated struct array are created and initiliazed with the given input values, also my creational values. Whenever a new client or server is created it will added on the array with these initials.

```
    int x;
    int serverIndex;
    serverIndex = *totalItemAddress;
    for(x=0;x<countCommands;x++)
    {
        char slotOne;
        char slotTwo;
        char slotThree;
        fscanf(file2,"\n%c %c %c",&slotOne,&slotTwo,&slotThree);
        if(slotOne=='A')
        {
        else if(slotOne=='I')
        {
        else if(slotOne=='S')
        {
        else if(slotOne == 'O')
        {
    }
```

This part reads the second input files and taking commands inside of it. Secondly, it checks the commands with the first character of input and do the operations by the rule of the architecture.

6. ***Main:*** My main function contains only the createClientServer functions' arguments and parse to it.Also main function has argc and *argv[] argument for the file read.