Hacettepe University Department of Computer Engineering

BBM203 Programming Lab.

Assignment 2

Name-Surname: Mehmet Emin Tuncer

Number: 21591022

Advisor : Res. Assist. Burçak ASAL

Programming Language : C

1.PROBLEM

In this assignment we wanted to design a basic client-server architecture. It has given us only one server and at least one client. Server and each client had its own stack and queues. Applying the commands in the input files we expect from us to stacks and queues. Two input files were given. The first line of the first input file specifies the total number of clients and servers. In the remaining lines, stack and tail dimensions are specified respectively.

The second input file gave us commands. The total number of commands is given on the first line of the second input file.

1.1.Commands:

- Process Command (’A’, ’Client Number’ ,’Process Character Name’):

-Add the third character to the queue of the client specified in the second character.

-Interrupt Command(’I’, ’Client/Server Number’ ,’Interrupt Character Name’):

-Add the third character to the stack of the client or server specified in the second character.

Send Command(’S’, ’Client Number’ ,’G’):

-‘G’ was given as irrelevant character.

-Add the client’s process or interrupt specified in the second character to server’s queue.

Operate Command(’O’ ,’G’ ,’G’):

-‘G’ was given as irrelevant character.

- Server will start to run its current process or interrupt.

1.2.Error Codes:

-If the queue of a particular client or server is full: Error Code “1”.

- If the stack of a particular client or server is full: Error Code “2”.

-If both stack and queue are empty : Error Code “3”.

2. MY SOLUTION:

I created a struct named client. I defined stack, queue and history in this struct. I found the total number of clients and servers by reading the first line of the first input file.

I read the rest of the lines, and I've assigned the clients and server stack and queue sizes. I read the first line of the second input file and set the client's command array size. I read the rest of the lines and called functions according to commands. I printed the server and the each client’s histories of in the output file. I was appointed as an argument files with the file path.

3.MY CODE & FUNCTIONS:

3.1.My Struct:

I created a struct named client. This structure has its own stack, queue, command array, stack size, queue size, top, front, rear.

Arrays-> Stack,Queue,Commandc.

Integers->top,rear,front,queue_size,stack_size,command.

3.2.Stack Functions:

-**int isFull():**It takes the client pointer as an argument. Returns 1 if the client's top value is one less than the stack size.

-int isEmpty(): It takes the client pointer as an argument. Returns 1 if the client's top value is equal to -1.

-void push(): It takes as arguments the client's pointer and the character to be sent to the client’s stack.

-void pop(): It takes the client pointer as an argument. It reduces the top value of the client.

-char top(): It takes the client pointer as an argument. returns the top value of the client's stack.

3.3.Queue Functions:

-void enqueue(): It takes as arguments the client's pointer and the character to be sent to the client's queue.

-void dequeue(): It takes the client pointer as an argument. The remainder of the portion of the client's front value to the queue size of a surplus makes the new value.

-int emptyqueue(): It takes the client pointer as an argument. Returns 1 if the client's front value is equal to -1.

-int fullqueue(): It takes the client pointer as an argument.Return 1 if: client's rear+1 equal to client's front or client's front is equal to zero and rear is equal to client's queue size.

3.4. Main():

I defined integers 'i' and 'j' for my "for loops". I set the first line of the first input file to a integer("y"). Defined a two-dimensional array with malloc to hold stack and queue sizes(qs_size). The rest of the first input file is assigned to the 'qs_size'. Defined a client array containing clients up to 'y' value.

Each client's stack, queue, command arrays and top, rear, front, command values are assigned in the for loop. Each client in for loop has been assigned to the client array. Last client in the array was 'server'.

The second input file is read using the function 'fcanf'. Commands were specified according to the first character, and command functions were called. According to the commands, error codes or sent items are assigned to each client's command list.

Finally, each user's history is printed in the output file.