INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

DEPARTMENT
OF
COMPUTER SCIENCE AND
ENGINEERING

NETWORKS LAB
(CS39006)

Vivek Jaiswal (20CS10077)
Shah Dhruv Rajendrabhai (20CS10088)

March, 2023

Contents

1 Datastructure:

1.1 Queue Node

```
typedef struct my_n
{
    char str[5000];
    int sz;
    struct my_n* next;
} node;
```

This is the structure of the node for the queue list.
str stores the data of the node.
sz stores the length of the data.
next stores the address of the next node.

1.2 My Table:

```
typedef struct my_tab
{
    node *head;
    node *tail;
    int sz;
    int sockfd;
}my_table;
```

This is the structure of the my_table using queue list.
head stores the address of the front node of the queue.
tail stores the address of the back node of the queue.
sz stores the size of the queue.
sockfd stores the socket data.

1.3 Functions:

```
int my_socket(int domain, int type, int protocol);
int my_bind(int sockfd, const struct sockaddr *addr,socklen_t addrlen);
int my_listen(int sockfd, int backlog);
int my_accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
int my_connect(int sockfd, const struct sockaddr *addr,socklen_t addrlen);
ssize_t my_send(int sockfd, const char *buf, size_t len, int flags);
ssize_t my_recv(int sockfd, char *buf, size_t len, int flags);
int my_close(int fd);
void my_push(my_table* s,node *a);
node* my_del(my_table* s);
```

These are all the functions required in the library.

my_socket function takes domain, type and protocol as arguments

- It then create the socket of the given domain, type and protocol and store it in send and receive table sockfd.
- It also intializes the send and receive table.
- It returns the sockfd.

my_bind function takes sockfd, socket address and address len as arguments

- It then binds the given sockfd with the socket address using bind function.
- It returns the error handling integer.

my_accept function takes sockfd, socket address and address len as arguments

- It then modifies the table sockfd with newsockfd and calls accept function.
- It returns the error handling integer.

my_connect function takes sockfd, socket address and address len as arguments

- It calls connect function.
- It returns the error handling integer.

my_send function takes sockfd, buffer, length of buffer and as arguments

- It create a node with buffer as the data and push it in the send table queue.
- It returns the size of the sent string.

my_recv function takes sockfd, buffer, length of buffer and flag as arguments

- It pop the front element from the receive queue and copy the data to buffer.
- It returns the size of the received string.

my_close function takes sockfd as arguments

- It sleeps for 5 seconds first.
- It will check if sockfd == **fd**
- If Yes, then free up the spaces and cancel all the threads
- Then close the fd.
- It returns the size of the sent string.

my_push function takes the table's pointer and node's pointer as arguments

- It pushes a node, having the string and its size stored in it, into the queue of the table.

my_del function takes the table's pointer as argument

- It pops a node from the front of the queue of the table.
- It returns the pointer of the node which is popped from the queue of the table.

2 Explanation:

2.1 Sending Workflow [my_send call is made]

- First, my_send call is made.
- In the my_send function, it will create the node with the buffer as data, and length of the buffer as sz of node.
- Then It pushes the node in the Send Table Queue S and return the length of sending data.
- Then In the send_thread, as the size of the Send Queue Table is not 0 so it will pop the node.
- Then it will modify the string as "len\r\n data"
- Then it will send the modified string by using send function which will be parsed by recv_thread.

2.2 Recieving Workflow [my_recv call is made]

- First, my_recv call is made.
- In the my_recv function, it will first pop the front node from the queue of the R table and store it as node *p.
- After this we will store the string into the buffer until the full string is stored or until the buffer size is full.
- And then return the number of bytes from the my_recv.
- Now we come to the recv_thread
- Here it will wait until the server gets an accept call or client gets a connect call or the size of R's table is full.
- Then it will allocate a new node's space dynamically.
- After this it will go to a recieve call, extract the total length of the data from the header and then store the remaining data into the node's buffer and also store the number of bytes in the node's size.
- It will do the recieve call till it is having the full data.
- The header here we are using is the "len\r\n" and using this only we will get to know till how many bytes we have to do the recv call.
- After this it will push the new node to the R table's queue and go to the next iteration.