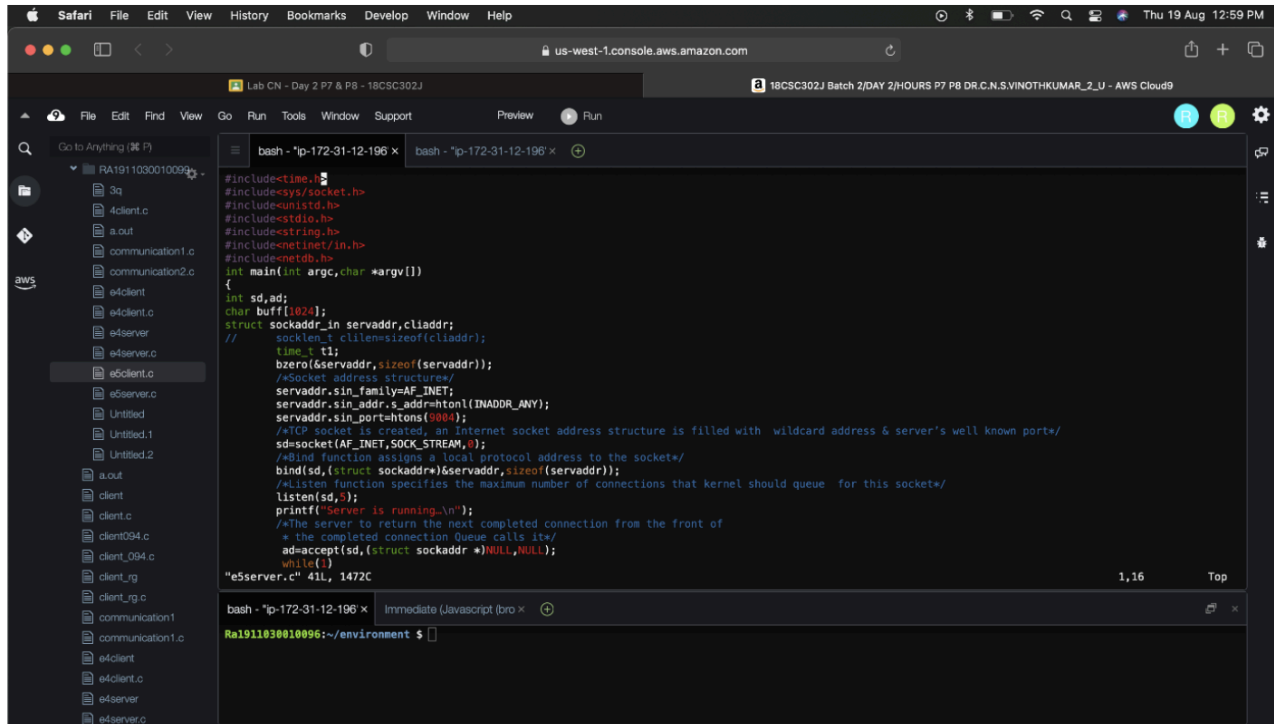


Lab Ex.5 - CONCURRENT TCP/IP DAY-TIME SERVER

Name:Rahul Goel

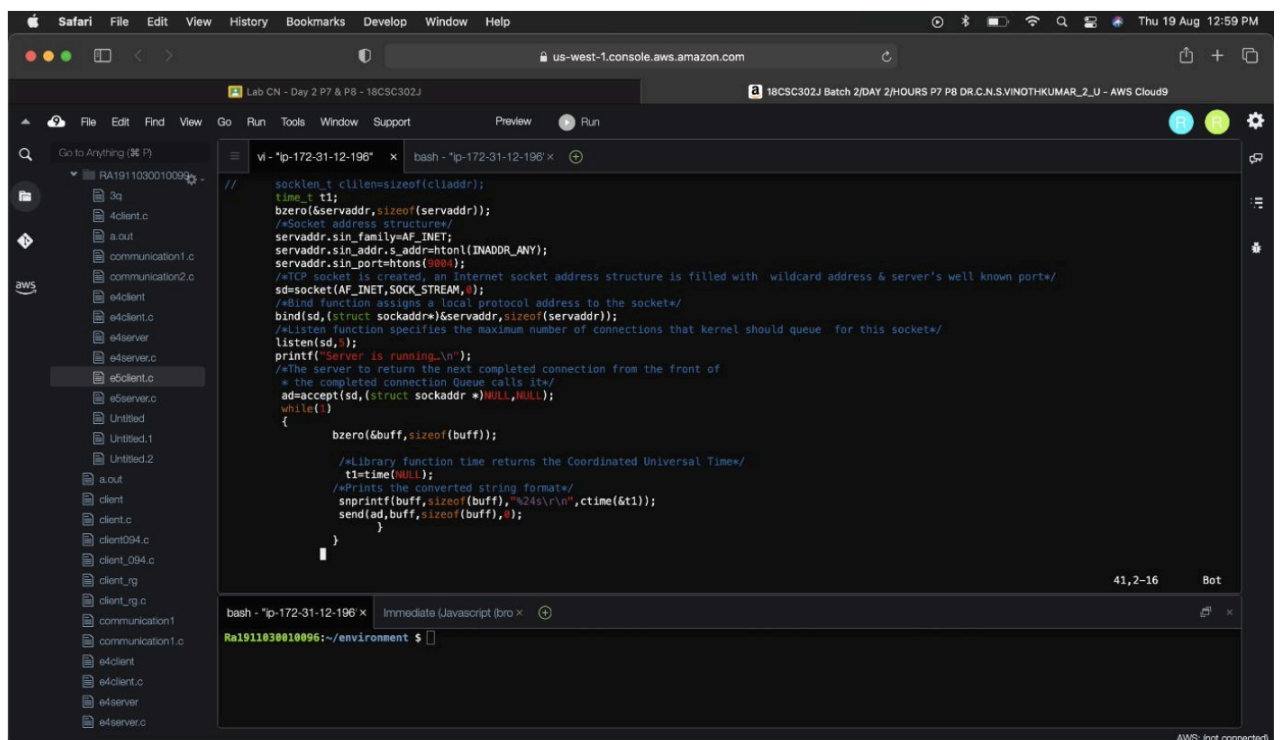
Reg.no: RA1911030010094

Server code:



The screenshot shows the AWS Cloud9 IDE interface. The left sidebar displays a file explorer with a project named 'RA1911030010094'. The main editor window shows the code for 'e4server.c'. The code includes headers for `time.h`, `sys/socket.h`, `unistd.h`, `stdio.h`, `string.h`, `netinet/in.h`, and `netdb.h`. The `main` function takes command-line arguments and sets up a server socket. It uses `socket` to create a TCP socket, `bind` to bind it to `INADDR_ANY` on port 8084, and `listen` to listen for connections. The `while` loop is currently empty, with a comment indicating where to add the logic to return the next completed connection from the front of the queue.

```
#include<time.h>
#include<sys/socket.h>
#include<unistd.h>
#include<stdio.h>
#include<string.h>
#include<netinet/in.h>
#include<netdb.h>
int main(int argc,char *argv[])
{
    int sd,ad;
    char buff[1024];
    struct sockaddr_in servaddr,cliaddr;
    // socklen_t clien=sizeof(cliaddr);
    time_t t1;
    bzero(&servaddr,sizeof(servaddr));
    /*Socket address structure*/
    servaddr.sin_family=AF_INET;
    servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
    servaddr.sin_port=htons(8084);
    /*TCP socket is created, an Internet socket address structure is filled with wildcard address & server's well known port*/
    sd=socket(AF_INET,SOCK_STREAM,0);
    /*bind function assigns a local protocol address to the socket*/
    bind(sd,(struct sockaddr*)&servaddr,sizeof(servaddr));
    /*Listen function specifies the maximum number of connections that kernel should queue for this socket*/
    listen(sd,5);
    printf("Server is running.\n");
    /*The server to return the next completed connection from the front of
    * the completed connection Queue calls it*/
    ad=accept(sd,(struct sockaddr *)&NULL,NULL);
    while(1)
    {
    }
}
```



This screenshot shows the same AWS Cloud9 IDE interface, but with the completed code for the server. The `while` loop now contains logic to receive data from the client, convert the time to a string, and send it back. The code uses `bzero` to clear the buffer, `time` to get the current time, `strftime` to format it as a string, and `send` to transmit it over the socket.

```
// socklen_t clien=sizeof(cliaddr);
time_t t1;
bzero(&servaddr,sizeof(servaddr));
/*Socket address structure*/
servaddr.sin_family=AF_INET;
servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
servaddr.sin_port=htons(8084);
/*TCP socket is created, an Internet socket address structure is filled with wildcard address & server's well known port*/
sd=socket(AF_INET,SOCK_STREAM,0);
/*bind function assigns a local protocol address to the socket*/
bind(sd,(struct sockaddr*)&servaddr,sizeof(servaddr));
/*Listen function specifies the maximum number of connections that kernel should queue for this socket*/
listen(sd,5);
printf("Server is running.\n");
/*The server to return the next completed connection from the front of
* the completed connection Queue calls it*/
ad=accept(sd,(struct sockaddr *)&NULL,NULL);
while(1)
{
    bzero(&buff,sizeof(buff));

    /*Library function time returns the Coordinated Universal Time*/
    t1=time(NULL);
    /*Prints the converted string format*/
    strftime(buff,sizeof(buff),"%24s\n",ctime(&t1));
    send(ad,buff,sizeof(buff),0);
}
}
```

