*Write a C program, using sockets create client and server socket programs. Write a TCP iterative server program, in server program take user input for port number and bind the port address. Server waits for clients to connect. When client connects communication can happen using recv and sent functions.*

*Following is the functionalities of server program:*

1. *Client sends message to server using sent functions.*
2. *Server receives all the messages, server ignores all the consonants in the message.*
3. *All the vowels in the message are converted into upper case.*
4. *Server returns the entire message to clients (with toggled vowel cases).*
5. *For example: "This is a test and sample message." to server will be sent back to client as "ThIs Is A tEst And sAmplE mEssAgE."*

*When client closes the connection server should close the communication with that client (socket). And once again wait for new clients to connect. Server program never exits.*

Using fork function rewrite the programs, such that this server can handle multiple client connections at one time. To test this you need to run simultaneously multiple copies of client executions. Please log on server machine number of clients it is handled at **this** time.

**\* server program \***   
  
#include "unistd.h"  
#include "errno.h"  
#include "sys/types.h"  
#include "sys/socket.h"  
#include "netinet/in.h"  
#include "netdb.h"  
  
#include "stdlib.h"  
#include "stdio.h"  
#include "string.h"  
#include "strings.h"  
#include "sys/wait.h"  
  
  
//Function Prototypes  
void myabort(char \*);  
  
//Some Global Variables  
int serverport = 3000;  
char \* eptr = NULL;  
int listen\_socket, client\_socket;  
struct sockaddr\_in Server\_Address, Client\_Address;  
int result,i;  
socklen\_t csize;  
pid\_t processid;  
int childcount = 0;  
  
  
//main()  
int  
main(int argc, char \*\*argv){  
  
char buf[100];  
char tmp[100];  
char \* ptr;  
int n, sent, length;  
  
//Step 0: Process Command Line  
if (argc > 2){  
myabort("Usage: server ");  
}  
if (argc == 2){  
serverport = (int) strtol(argv[1], &eptr, 10);  
if (\*eptr != '\0') myabort("Invalid Port Number!");  
}  
  
//Step 1: Create a socket  
listen\_socket = socket(PF\_INET, SOCK\_STREAM, 0);  
if (listen\_socket == -1) myabort("socket()");  
  
  
//Step 2: Setup Address structure  
bzero(&Server\_Address, sizeof(Server\_Address));  
Server\_Address.sin\_family = AF\_INET;  
Server\_Address.sin\_port = htons(serverport);  
Server\_Address.sin\_addr.s\_addr = INADDR\_ANY;  
  
  
//Step 3: Bind the socket to the port  
result = bind(listen\_socket, (struct sockaddr \*) &Server\_Address, sizeof(Server\_Address));  
if (result == -1) myabort("bind()");  
  
//Step 4:Listen to the socket  
result = listen(listen\_socket, 1);  
if (result == -1) myabort("listen()");  
  
  
  
printf("\nThe forkserver :%d\n",ntohs(Server\_Address.sin\_port));  
fflush(stdout);  
//Step 5: Setup an infinite loop to make connections  
while(1){  
  
  
//Accept a Connection  
csize = sizeof(Client\_Address);  
client\_socket = accept( listen\_socket,(struct sockaddr \*) &Client\_Address,&csize);  
if (client\_socket == -1) myabort("accept()");  
  
printf( "\nClient Accepted!\n" );  
  
  
//fork this process into a child and parent  
processid = fork();  
  
//Check the return value given by fork(), if negative then error,  
//if 0 then it is the child.  
if ( processid == -1){  
myabort("fork()");  
}else if (processid == 0){  
/\*Child Process\*/  
  
close(listen\_socket);  
//loop until client closes  
while (1){  
  
  
  
//read string from client  
bzero(&buf, sizeof(buf));  
do{  
bzero(&tmp, sizeof(tmp));  
n = read(client\_socket,(char \*) &tmp, 100);  
//cout << "server: " << tmp;  
tmp[n] = '\0';  
if (n == -1) myabort("read()");  
if (n == 0) break;  
strncat(buf, tmp, n-1);  
buf[n-1] = ' ';  
} while (tmp[n-1] != '\n');  
  
buf[ strlen(buf) ] = '\n';  
  
printf( "From client: %s",buf);  
  
if (n == 0) break;  
  
  
//write string back to client  
sent = 0;  
ptr = buf;  
length = strlen(buf);  
  
//the vowels in the message are converted into upper case.   
for( i = 0; ptr[ i ]; i++)  
{  
if( ptr[i]=='a' || ptr[i]=='e' || ptr[i]=='i' || ptr[i]=='o' || ptr[i]=='u' )  
ptr[ i ] = toupper( ptr[ i ] );  
else  
ptr[ i ] = ptr[ i ] ;  
  
}   
  
  
printf( "To client: %s",ptr);  
while (sent < length ){  
n = write(client\_socket, ptr, strlen(ptr) );  
if ( n == -1) myabort("write()");  
sent += n;  
ptr += n;  
}  
}//end inner while  
  
close(client\_socket);  
  
//Child exits  
exit(0);  
}  
  
  
//Parent Process  
  
printf("\nChild process spawned with id number: %d",processid );  
//increment the number of children processes  
childcount++;  
while(childcount){  
processid = waitpid( (pid\_t) - 1, NULL, WNOHANG );  
if (processid < 0) myabort("waitpid()");  
else if (processid == 0) break;  
else childcount--;  
}   
  
}  
close(listen\_socket);  
  
exit(0);  
  
}  
  
  
void myabort(char \* msg){  
printf("Error!: %s" , msg);  
exit(1);  
}

**/\* client \*/**

#include "stdlib.h"  
#include "stdio.h"  
#include "string.h"  
#include "unistd.h"  
#include "sys/socket.h"  
#include "sys/types.h"  
#include "netinet/in.h"  
#include "strings.h"  
#include "arpa/inet.h"  
  
#define BUFFER 1024  
  
  
main(int argc, char \*\*argv)  
{  
struct sockaddr\_in serv;  
int sock;  
char in[BUFFER];  
char out[BUFFER];  
int len;  
  
  
if((sock = socket(AF\_INET, SOCK\_STREAM, 0)) == -1)  
{  
perror("socket");  
exit(-1);  
}  
  
serv.sin\_family = AF\_INET;  
serv.sin\_port = htons(atoi(argv[2]));  
serv.sin\_addr.s\_addr = inet\_addr(argv[1]);  
bzero(&serv.sin\_zero, 8);  
  
printf("\nThe TCPclient %d\n",ntohs(serv.sin\_port));  
fflush(stdout);  
  
  
if((connect(sock, (struct sockaddr \*)&serv, sizeof(struct sockaddr\_in))) == -1)  
{  
perror("connect");  
exit(-1);  
}  
  
while(1)  
{  
printf("\nInput: ");  
  
fgets(in, BUFFER, stdin);  
send(sock, in, strlen(in), 0);  
  
  
len = recv(sock, out, BUFFER, 0);  
out[len] = '\0';  
printf("Output: %s\n", out);  
}  
  
close(sock);  
  
  
}