Programming Assignment 1
ECEN602: Computer Communication and Networking

Mahesh Kallelil (UIN: 226001692)
Ankit Yadav (UIN: 726006435)
Dr. Pierce Cantrell

September 22, 2017

1 Design

1.1 Client

Sockets are the endpoint for communication between client and server. Every client who has to talk to any node in the network needs to have a socket descriptor. Therefore, while implementing the client side of Echo-Server and Client model we need to create a socket. Socket is created using the SOCK() function. This function takes Address family, type and protocol as input. As we are comunicating over internet using TCP sockets. Hence, we pass AF_INET and SOCK_STREAM as input. SOCKET() function returns a negative value if the socket is not created sucessfully, else a positive value. Now, client has successfully created the socket and it needs to connect to the server that is other point of communication. SOCKADDR_IN structure holds the information of the other end e.g. Medium of communication(internet), Identification of the other host(IP), port(process ID). Client uses the information packed in the SOCKADDR_IN structure to identify and connect to the server. CONNECT() function is used to connect to the server. It returns 0 if the connection is successful else it returns -1. Once the client is connected to the server, it can start communicating. Client records it's message into a buffer which is then transferred to the server using WRITE() function. As it is an echo server, server echoes back the same message to the client. READ() function is used to read the reply from the server. Which is then printed onto the screen. When the client encounters EOF(end of file). It closes the connection using close() function.

1.2 Server

The server is implemented independent of address family (IPv6 and IPv4). The server uses the `getaddrinfo()` to populate the data structures for the socket functions. It later binds to the port specified as the command line argument. When a client connects to the server, it forks and creates a child process to handle the incoming client. When the client disconnects, the child process exits and the parent receives a `SIGCHLD` signal. This is handled by a signal handler [`void sigchld_handler(int signo)`] which makes sure that the system doesn't create any zombie processes. The EOF character is processed in the client side and whenever the user enters the EOF character, the client closes the connection with the server.

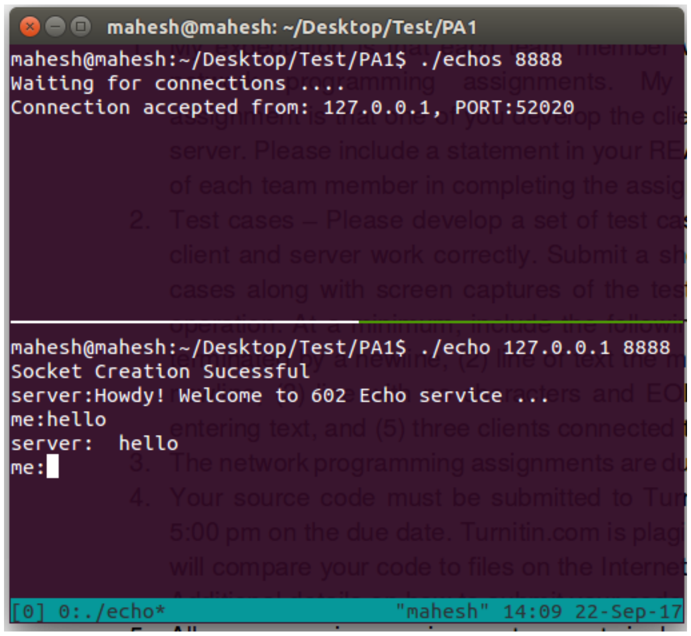
1.3 Usage

The echo-server can be used as follows:

```
$ make #For building the binaries
$ ./echos 8888 #For executing the server
$ ./echo 127.0.0.1 8888 #For executing the client
```

2 Testcases

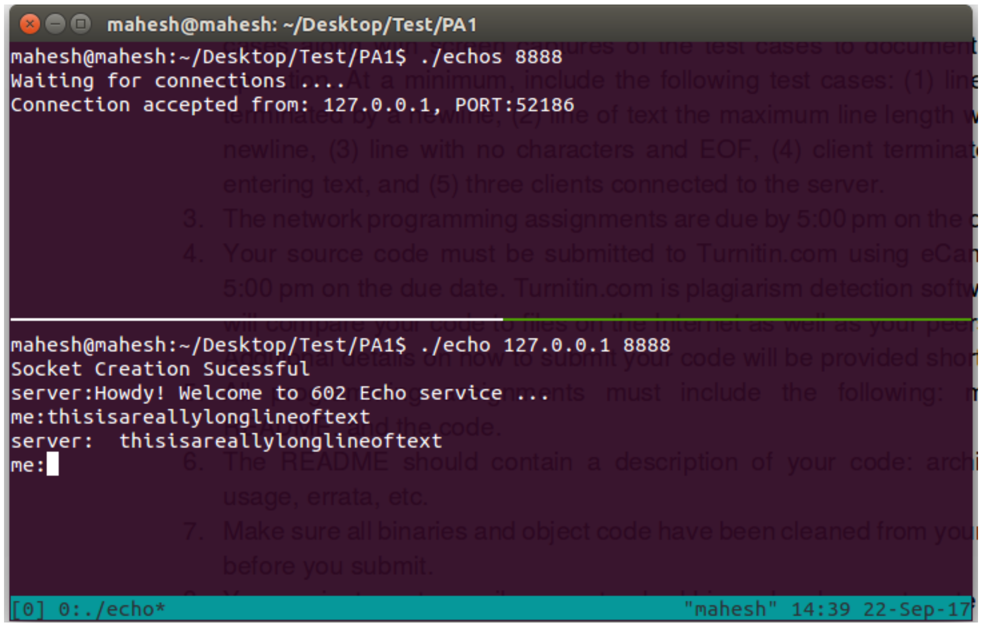
Figure 1: Line of Text terminated by newline



```
mahesh@mahesh:~/Desktop/Test/PA1$ ./echos 8888
Waiting for connections ....
Connection accepted from: 127.0.0.1, PORT:52020
```

```
mahesh@mahesh:~/Desktop/Test/PA1$ ./echo 127.0.0.1 8888
Socket Creation Sucessful
server:Howdy! Welcome to 602 Echo service ...
me:hello
server:  hello
me:
[0] 0:./echo*                         "mahesh" 14:09 22-Sep-17
```

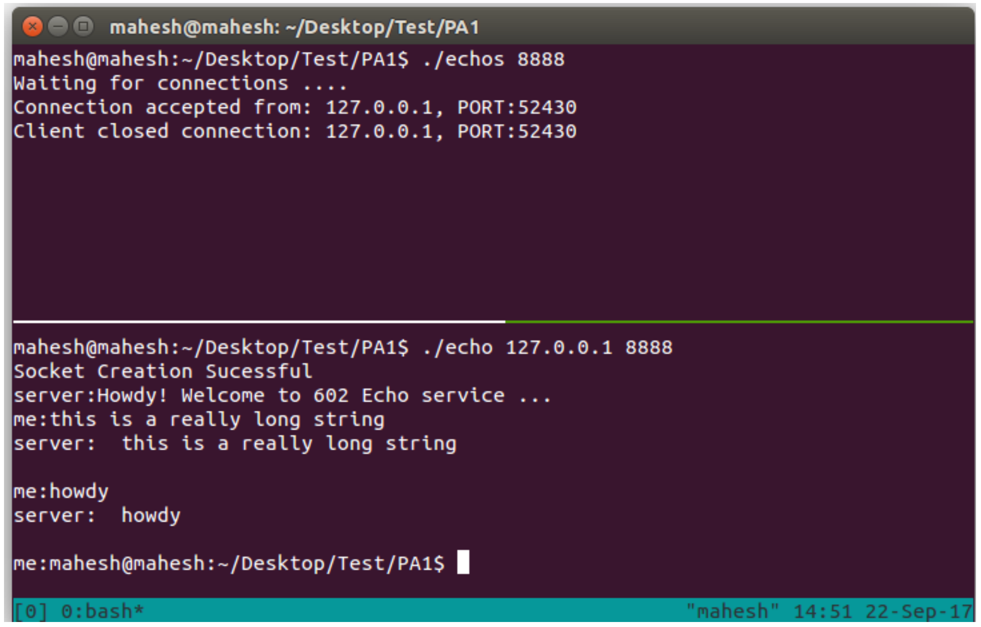
Figure 2: Line of Text maximum line length without NL



```
mahesh@mahesh: ~/Desktop/Test/PA1
mahesh@mahesh:~/Desktop/Test/PA1$ ./echos 8888
Waiting for connections ....
Connection accepted from: 127.0.0.1, PORT:52186

mahesh@mahesh:~/Desktop/Test/PA1$ ./echo 127.0.0.1 8888
Socket Creation Sucessful
server:Howdy! Welcome to 602 Echo service ...
me:thisisareallylonglineoftext
server:  thisisareallylonglineoftext
me:
[0] 0:./echo*                                        "mahesh" 14:39 22-Sep-17
```

Figure 3: Line of text with only EOF



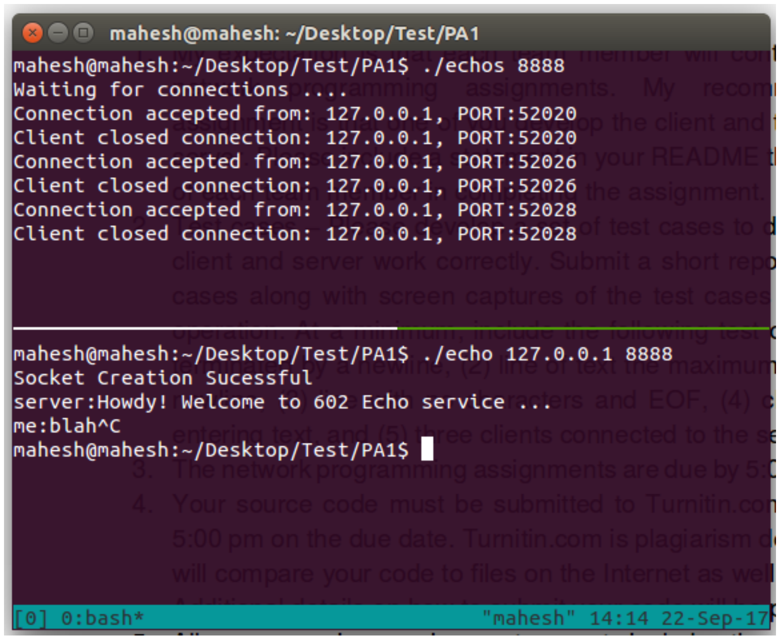
```
mahesh@mahesh: ~/Desktop/Test/PA1
mahesh@mahesh:~/Desktop/Test/PA1$ ./echos 8888
Waiting for connections ....
Connection accepted from: 127.0.0.1, PORT:52430
Client closed connection: 127.0.0.1, PORT:52430

mahesh@mahesh:~/Desktop/Test/PA1$ ./echo 127.0.0.1 8888
Socket Creation Sucessful
server:Howdy! Welcome to 602 Echo service ...
me:this is a really long string
server:  this is a really long string

me:howdy
server:  howdy

me:mahesh@mahesh:~/Desktop/Test/PA1$
[0] 0:bash*                                          "mahesh" 14:51 22-Sep-17
```

Figure 4: Client terminated after entering text

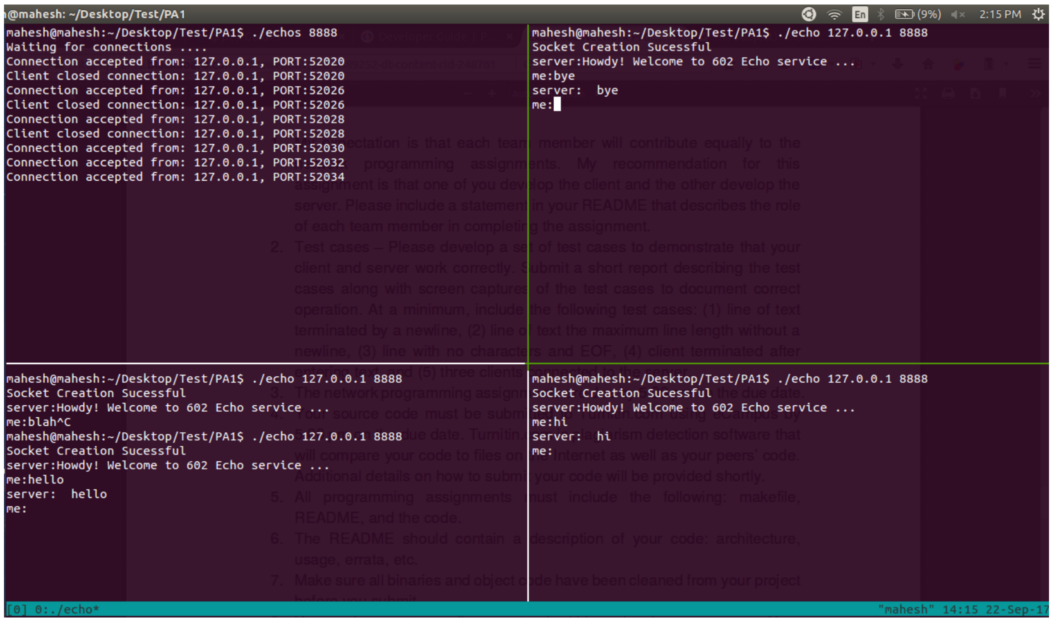


```
mahesh@mahesh:~/Desktop/Test/PA1$ ./echos 8888
Waiting for connections ....
Connection accepted from: 127.0.0.1, PORT:52020
Client closed connection: 127.0.0.1, PORT:52020
Connection accepted from: 127.0.0.1, PORT:52026
Client closed connection: 127.0.0.1, PORT:52026
Connection accepted from: 127.0.0.1, PORT:52028
Client closed connection: 127.0.0.1, PORT:52028
```

```
mahesh@mahesh:~/Desktop/Test/PA1$ ./echo 127.0.0.1 8888
Socket Creation Sucessful
server:Howdy! Welcome to 602 Echo service ...
me:blah^C
mahesh@mahesh:~/Desktop/Test/PA1$
```

```
[0] 0:bash*                          "mahesh" 14:14 22-Sep-17
```

Figure 5: Three clients connected to server



```
mahesh@mahesh:~/Desktop/Test/PA1$ ./echos 8888
Waiting for connections ....
Connection accepted from: 127.0.0.1, PORT:52020
Client closed connection: 127.0.0.1, PORT:52020
Connection accepted from: 127.0.0.1, PORT:52026
Client closed connection: 127.0.0.1, PORT:52026
Connection accepted from: 127.0.0.1, PORT:52028
Client closed connection: 127.0.0.1, PORT:52028
Connection accepted from: 127.0.0.1, PORT:52030
Connection accepted from: 127.0.0.1, PORT:52032
Connection accepted from: 127.0.0.1, PORT:52034
```

```
mahesh@mahesh:~/Desktop/Test/PA1$ ./echo 127.0.0.1 8888
Socket Creation Sucessful
server:Howdy! Welcome to 602 Echo service ...
me:bye
server:  bye
me:
```

```
mahesh@mahesh:~/Desktop/Test/PA1$ ./echo 127.0.0.1 8888
Socket Creation Sucessful
server:Howdy! Welcome to 602 Echo service ...
me:blah^C
mahesh@mahesh:~/Desktop/Test/PA1$ ./echo 127.0.0.1 8888
Socket Creation Sucessful
server:Howdy! Welcome to 602 Echo service ...
me:hello
server:  hello
me:
```

```
mahesh@mahesh:~/Desktop/Test/PA1$ ./echo 127.0.0.1 8888
Socket Creation Sucessful
server:Howdy! Welcome to 602 Echo service ...
me:hi
server:  hi
me:
```

```
[0] 0:./echo*                        "mahesh" 14:15 22-Sep-17
```

A Source Code

A.1 Server Source Code

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <signal.h>
#include <netdb.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/wait.h>
#include <netinet/in.h>
#include <arpa/inet.h>

// Max clients connected to the server
#define BACKLOGS 10
#define MAX_CLIENT_MSG_LEN 100

// Signal Handler for Child Process
void sigchld_handler(int signo);
// Sends n bytes of msg to socket
int writen(int socket, char * msg, unsigned int n);

int main(int argc, char const *argv[])
{
  int sd, new_sd;
  int err;
  int yes=1;
  int client_port;
  int bytes_recv;
  struct addrinfo addr_info, *server_info, *it;
  struct sigaction sig;
  struct sockaddr_storage client_addr;
  socklen_t sockaddr_size;
  char ip_addr[INET6_ADDRSTRLEN];
  char *msg, client_msg[MAX_CLIENT_MSG_LEN];
  void *client_ip;

  // Populate the address structure
  memset(&addr_info, 0, sizeof(addr_info));
  // IPv4 IPv6 independent
  addr_info.ai_family = AF_UNSPEC;
  // Use TCP
  addr_info.ai_socktype = SOCK_STREAM;
addr_info.ai_flags = AI_PASSIVE;

// Check command line arguments
if (argc < 2) {
  printf("Invalid usage. Try: \n");
  printf("$ ./echos <port_no>\n");
  return 1;
}

if ((err = getaddrinfo(NULL, argv[1], &addr_info, &server_info)) !=
    0) {
  fprintf(stderr, "getaddrinfo() failed: %s\n", gai_strerror(err));
  return 1;
}

// Iterate through output of getaddrinfo() and find a port to bind
   to
for (it = server_info; it != NULL; it = it->ai_next) {
  sd = socket(it->ai_family, it->ai_socktype, it->ai_protocol);
  if (sd == -1) {
    perror("socket()");
    continue;
  }
  if (setsockopt(sd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(int)) ==
       -1) {
    perror("setsockopt()");
    return 1;
  }
  if (bind(sd, it->ai_addr, it->ai_addrlen) == -1) {
    close(sd);
    perror("bind()");
    // If bind was unsuccesssful, try the next port
    // Note: In this case, there happens to be only one port and
    // so the list server_info only has one element
    continue;
  }
  break;
}

freeaddrinfo(server_info);

// Check if server successfully binded to the given port
if (it == NULL) {
  fprintf(stderr, "Server failed to bind to given port.\n");
  return 1;
}
```

```
// Listen on the port
if (listen(sd, BACKLOGS) == -1) {
  perror("listen()");
  return 1;
}

// Register the signal handler for SIGCHLD event
// so that the resulting process is not a zombie while exiting
sig.sa_handler = sigchld_handler;
sigemptyset(&sig.sa_mask);
sig.sa_flags = SA_RESTART;
if (sigaction(SIGCHLD, &sig, NULL) == -1) {
  perror("sigaction()");
  return 1;
}

printf("Waiting for connections ....\n");

while (1) {
  sockaddr_size = sizeof(client_addr);
  // Accept incoming connection
  new_sd = accept(sd, (struct sockaddr *)&client_addr, &
     sockaddr_size);
  if (new_sd == -1) {
    perror("accept()");
    continue;
  }

  // Print the client IP Address and Port Number
  if ((*(struct sockaddr *)&client_addr).sa_family == AF_INET) {
    client_ip = &(((struct sockaddr_in *)&client_addr)->sin_addr);
  } else {
    client_ip = &(((struct sockaddr_in6 *)&client_addr)->sin6_addr);
  }
  client_port = ntohs((*(struct sockaddr_in *)&client_addr).sin_port
     );
  inet_ntop(client_addr.ss_family, client_ip, ip_addr,
     INET6_ADDRSTRLEN);
  printf("Connection accepted from: %s, PORT:%d\n", ip_addr,
     client_port);

  // Create a child process to handle the client
  if (!fork()) {
    close(sd);
    msg = "Howdy! Welcome to 602 Echo service ...\n";
    if (send(new_sd, msg, strlen(msg), 0) == -1)
      perror("send()");
      // Wait for client to send data
      while ((bytes_recv = recv(new_sd, client_msg, MAX_CLIENT_MSG_LEN
         , 0)) > 0) {
        client_msg[bytes_recv] = '\0';
        // Write back the received data
        writen(new_sd, client_msg, bytes_recv);
      }
      // If client closes connection, print it on standard output
      if (bytes_recv == 0) {
        printf("Client closed connection: %s, PORT:%d\n", ip_addr,
           client_port);
        fflush(stdout);
      }
      // If recv() fails, output with error code
      else {
        perror("recv()");
      }
      close(new_sd);
      exit(0);
    }

    close(new_sd);
  }
}

// SIGCHLD handler
void sigchld_handler(int signo)
{
  int saved_errno = errno;
  while(waitpid(-1, NULL, WNOHANG) > 0);
  errno = saved_errno;
}

// Writes n bytes of msg to socket
// Returns -1 on error and 0 if successful
int writen(int socket, char *msg, unsigned int n)
{
  if (send(socket, msg, n, 0) == -1) {
    perror("send()");
    return -1;
  }
  return 0;
}
```

A.2 Client Source Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <arpa/inet.h>

#define MAX_MSG_LEN 100

int main(int argc, char const *argv[])
{
  int ank_sock,a,n;
  struct hostent *hostname;
  // Creation of a socket
  ank_sock = socket(AF_INET, SOCK_STREAM, 0);

  // Check if the creation was successful
  if(ank_sock<=0)
     printf("Socket Creation was Unsucessful\n");
  else
     printf("Socket Creation Sucessful\n");

  if (argc < 3) {
    printf("Invalid usage. Try ..\n");
    printf("$ ./echo IP_Addr Port_No\n");
    return 1;
  }

  // connecting to a remote server
  struct sockaddr_in rem_serv;
  memset(&rem_serv,0,sizeof(rem_serv));
  rem_serv.sin_addr.s_addr = inet_addr(argv[1]);
  rem_serv.sin_family=AF_INET;
  rem_serv.sin_port=htons(atoi(argv[2]));

  if((a=connect(ank_sock,(struct sockaddr*)&rem_serv,sizeof(rem_serv))
     )<0)  {
   perror("couldn't connect");
   exit(1);
  }
  //printf("%d",a);
  char mess[100]="anit";
  char ankit[100]="hello";


   // while(1){
         printf("server:");
        // reading the reply from the server
        if((n=read(ank_sock,&mess,sizeof(mess)))<0)
          perror("message couldn't be recei");
        else
          printf("%s",mess);

        while (1) {

        // receiving the messsage from the user
        printf("me:");
        fgets (ankit, MAX_MSG_LEN, stdin);
//        scanf("%s", ankit);
        // When the user enters ^D (EOF), close the connection and exit
           loop
        if (strlen(ankit) == 0)
          break;

        // writing to the server
        if((n=write(ank_sock,&ankit,100)<0))
          perror("message couldn't be sent");

        memset(mess,0,strlen(mess));
        memset(ankit,0,strlen(ankit));

        printf("server:  ");
        // reading the reply from the server
        if((n=read(ank_sock,&mess,sizeof(mess)))<0)
          perror("message couldn't be recei");
        else
          printf("%s\n",mess);
      // }
      }

        close(ank_sock);


   //printf("port no %d",rem_serv.sin_port);
   //printf("%d",ntohs(rem_serv.sin_port));
}
```