

TCP Server

1. TCP Server

1.1 CSCI 330

CSCI 330 UNIX and Network Programming



TCP
Server Programming



1.2 Unit Overview

Unit Overview

- TCP client & server programming
 - review concepts and necessary system calls
 - illustrate client with DNS lookup
- Server fork to process client request
- Example TCP server
 - list files in a directory

1.3 TCP programming

TCP programming

- provides multiple endpoints on a single node: port
- common abstraction: socket
- socket is end-point of communication link
 - identified as IP address + port number
 - can receive data
 - can send data

1.4 Socket system calls

Socket system calls

server

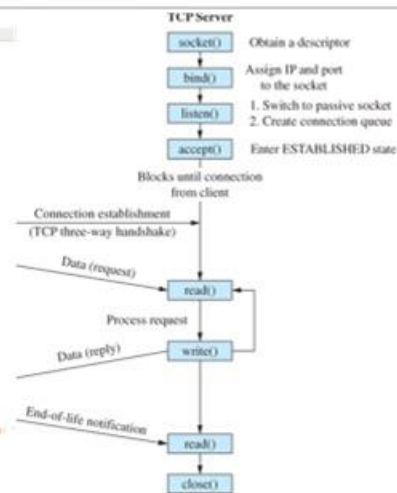
Primitive	Meaning
socket	Create a new communication endpoint
bind	Attach a local address to a socket
listen	Announce willingness to accept connections
accept	Block caller until a connection request arrives
connect	Actively attempt to establish a connection
write	Send(write) some data over the connection
read	Receive(read) some data over the connection
close	Release the connection

client

1.5 TCP server illustration

TCP server illustration

```
serverTCPserver.c
32 // Create the TCP socket
33 int sock = socket(AF_INET, SOCK_STREAM, 0);
34 if (sock < 0) {
35     perror("socket");
36     exit(EXIT_FAILURE);
37 }
38 // Create address structures
39 struct sockaddr_in server_address; // structure for address of server
40 struct sockaddr_in client_address; // structure for address of client
41 unsigned int addrlen = sizeof(client_address);
42
43 // Construct the server socket in structure
44 memset(&server_address, 0, sizeof(server_address)); // Clear struct
45 server_address.sin_family = AF_INET; // Internet IP v4
46 server_address.sin_addr.s_addr = INADDR_ANY; // Any IP address
47 server_address.sin_port = htons(8080); // server port
48
49 // Bind the socket
50 if (bind(sock, (struct sockaddr *)&server_address, sizeof(server_address)) < 0) {
51     perror("bind");
52     exit(EXIT_FAILURE);
53 }
54 // Listen: make socket passive and set length of queue
55 if (listen(sock, 5) < 0) {
56     perror("listen");
57     exit(EXIT_FAILURE);
58 }
59
60 // Run until (shutting on ctrl-c) or arg[0] == end;
61
62 // Run until (shutting on ctrl-c)
63 while (true) {
64     // Accept connection from client
65     if (newsock = accept(sock, (struct sockaddr *)&client_address, &addrlen)) {
66         // New connection
67         // Read a message from the client
68         char buffer[256];
69         int received = read(newsock, buffer, 256);
70         if (received < 0) {
71             perror("read");
72             exit(EXIT_FAILURE);
73         }
74         // Write the message back to client
75         if (write(newsock, buffer, received) < 0) {
76             perror("write");
77             exit(EXIT_FAILURE);
78         }
79         // Close the connection
80         close(newsock);
81     }
82 }
83 }
```



1.6 TCP client illustration

TCP client illustration



1.7 Improve TCP client

Improve TCP client

- Become useful as generic client to any TCP server
- Improvements:
 - accept FQDN as server address
 - read & process complete server response

1.8 Accept FQDN as server address

Accept FQDN as server address

```
// lookup FQDN
struct addrinfo *res, hints;
memset(&hints, 0, sizeof(hints));
hints.ai_family = AF_INET;
hints.ai_socktype = SOCK_STREAM;
int error = getaddrinfo(argv[1], argv[2], &hints, &res);
if (error) { ... }

// Create the TCP socket
int sock = socket(AF_INET, SOCK_STREAM, 0);
if (sock < 0) { ... }

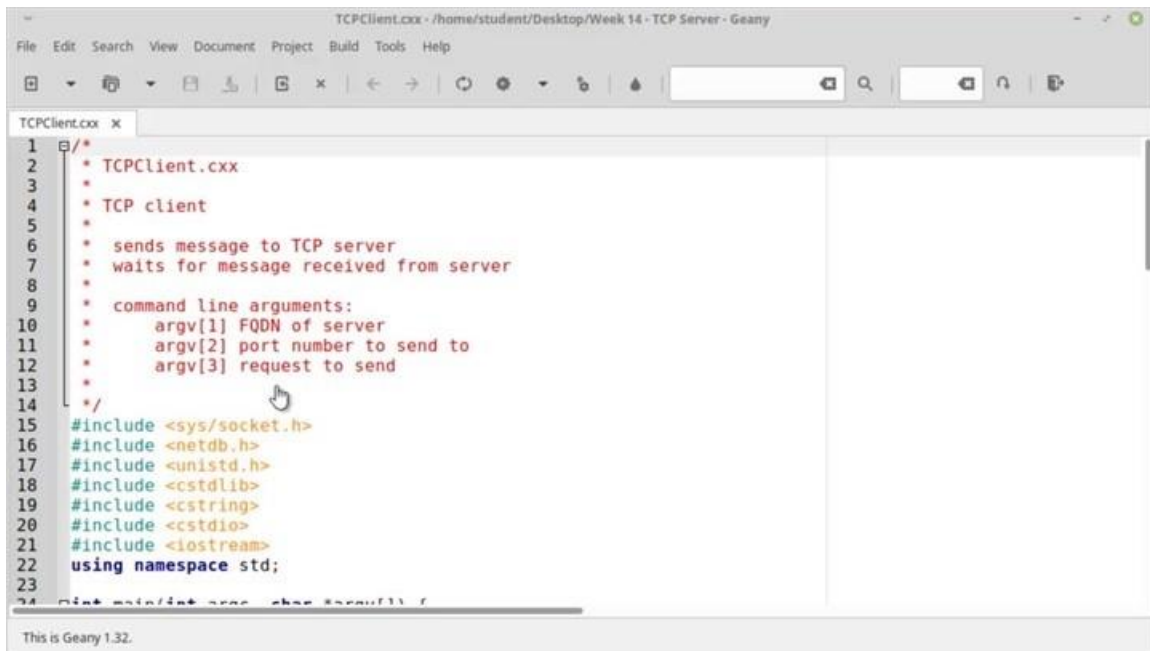
// connect to server
if (connect(sock, res->ai_addr, res->ai_addrlen) < 0) { ... }
```

1.9 Process complete server response

Process complete server response

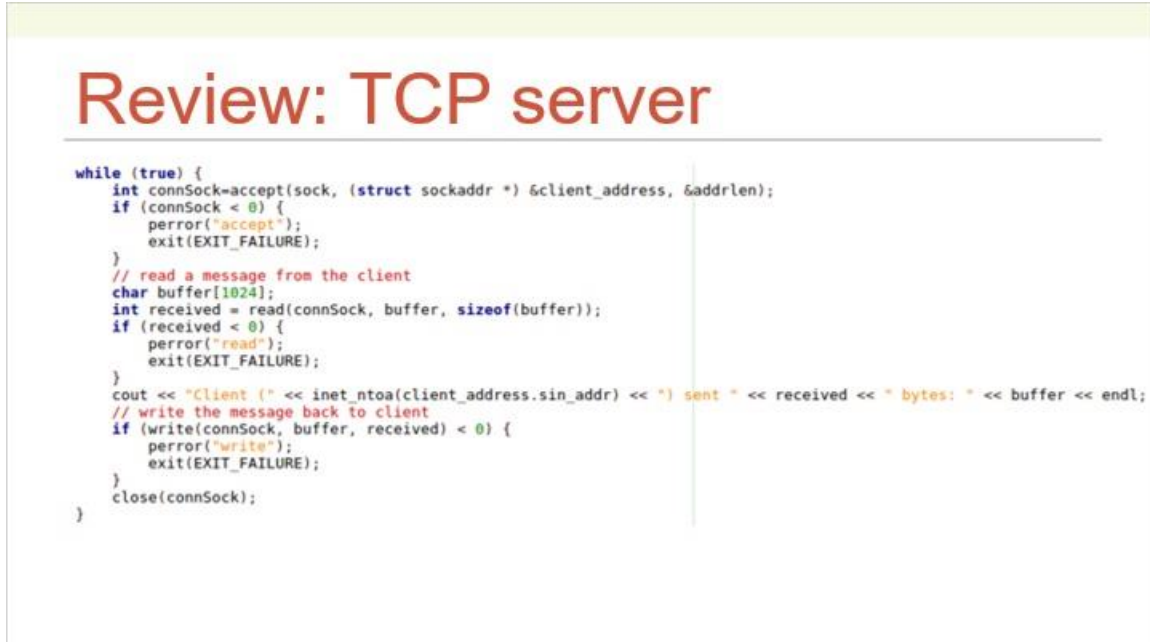
```
// Receive the message back from the server
do {
    received = read(sock, buf, sizeof(buf));
    if (received < 0) { ... }
    cout.write(buf, received);
} while (received > 0);
```

1.10 Generic TCP request client (1 of 2)



```
1 1 /*
2 2  * TCPClient.cpp
3 3  *
4 4  * TCP client
5 5  *
6 6  * sends message to TCP server
7 7  * waits for message received from server
8 8  *
9 9  * command line arguments:
10 10  *   argv[1] FQDN of server
11 11  *   argv[2] port number to send to
12 12  *   argv[3] request to send
13 13  */
14 14 */
15 15 #include <sys/socket.h>
16 16 #include <netdb.h>
17 17 #include <unistd.h>
18 18 #include <cstdlib>
19 19 #include <cstring>
20 20 #include <cstdio>
21 21 #include <iostream>
22 22 using namespace std;
23 23
24 24 int main(int argc, char *argv[]) {
```

1.11 Review: TCP server



Review: TCP server

```
while (true) {
    int connSock=accept(sock, (struct sockaddr *) &client_address, &addrlen);
    if (connSock < 0) {
        perror("accept");
        exit(EXIT_FAILURE);
    }
    // read a message from the client
    char buffer[1024];
    int received = read(connSock, buffer, sizeof(buffer));
    if (received < 0) {
        perror("read");
        exit(EXIT_FAILURE);
    }
    cout << "Client (" << inet_ntoa(client_address.sin_addr) << ") sent " << received << " bytes: " << buffer << endl;
    // write the message back to client
    if (write(connSock, buffer, received) < 0) {
        perror("write");
        exit(EXIT_FAILURE);
    }
    close(connSock);
}
```


1.12 Review: TCP Server basic logic

Review: TCP Server basic logic

```
while (true) {  
    connSock = accept(sock, ...)  
  
    // process client's request  
    //    via connSock  
  
    close(connSock) ;  
}
```

Server is busy !
No other client can connect

1.13 Better: TCP Server fork

Better: TCP Server fork

- server starts loop
 - blocks on accept for connection from client
 - after accept:
 - accept returns dedicated connection socket
 - server forks into parent and child process
- parent process
 - closes dedicated connection socket
 - continues to block for next accept
- child process
 - serves client request
 - communicates with client via dedicated connection socket



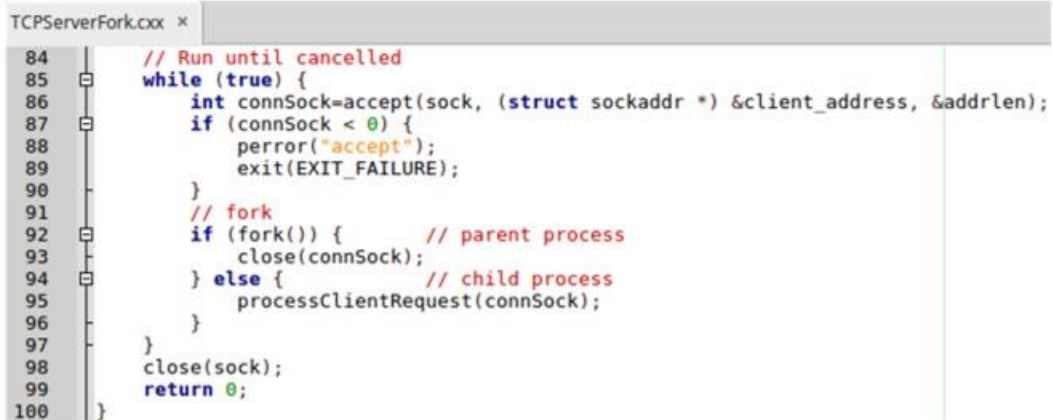
1.14 TCP Server fork: logic

TCP Server fork: logic

```
while (true) {  
    connSock = accept(sock, ...);  
    if (fork()) {        // parent process  
        close(connSock);  
    } else {            // child process  
        // process client's request via connSock  
        ...  
    }  
}
```

1.15 TCP server/fork illustration (1 of 2)

TCP server/fork illustration (1 of 2)



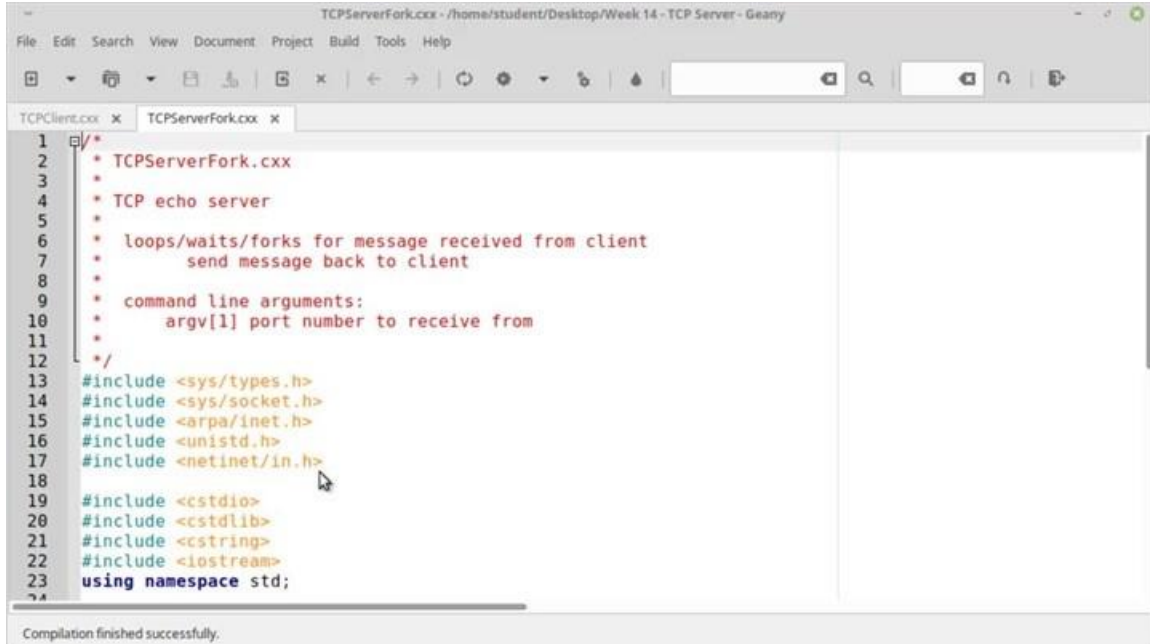
```
TCPServerFork.cxx x  
84 // Run until cancelled  
85 while (true) {  
86     int connSock=accept(sock, (struct sockaddr *) &client_address, &addrlen);  
87     if (connSock < 0) {  
88         perror("accept");  
89         exit(EXIT_FAILURE);  
90     }  
91     // fork  
92     if (fork()) {        // parent process  
93         close(connSock);  
94     } else {            // child process  
95         processClientRequest(connSock);  
96     }  
97 }  
98 close(sock);  
99 return 0;  
100 }
```


1.16 TCP server/fork illustration (2 of 2)

TCP server/fork illustration (2 of 2)

```
TCPServerFork.cxx x
25 void processClientRequest( int connSock) {
26     int received;
27     char buffer[1024];
28
29     // read a message from the client
30     if ((received = read(connSock, buffer, sizeof(buffer))) <= 0) {
31         perror("read");
32         exit(EXIT_FAILURE);
33     }
34
35     cout << "Client sent " << received << " bytes: " << buffer << endl;
36
37     // write the message back to client
38     if (write(connSock, buffer, received) < 0) {
39         perror("write");
40         exit(EXIT_FAILURE);
41     }
42     close(connSock);
43     exit(EXIT_SUCCESS);
44 }
```

1.17 TCP server/fork illustration



```
TCPServerFork.cxx - /home/student/Desktop/Week 14 - TCP Server - Geany
File Edit Search View Document Project Build Tools Help
TCPClient.cxx x TCPServerFork.cxx x
1 /*
2  * TCPServerFork.cxx
3  *
4  * TCP echo server
5  *
6  * loops/waits/forks for message received from client
7  * send message back to client
8  *
9  * command line arguments:
10  * argv[1] port number to receive from
11  */
12
13 #include <sys/types.h>
14 #include <sys/socket.h>
15 #include <arpa/inet.h>
16 #include <unistd.h>
17 #include <netinet/in.h>
18
19 #include <cstdio>
20 #include <cstdlib>
21 #include <cstring>
22 #include <iostream>
23 using namespace std;
```

Compilation finished successfully.

1.18 Server example: list directory

Server example: list directory

- after accept, server forks to service client request
 - parent process will loop to next accept
- child process serves client request
 - read directory path name from client
 - open directory
 - read directory entries, send file names to client
 - end process

1.19 Server child: processClientRequest

Server child: processClientRequest

```
TCPServerReadDir.cxx x
25 void processClientRequest(int connSock) {
26     int received;
27     char path[1024], buffer[1024];
28
29     // read a message from the client
30     if ((received = read(connSock, path, sizeof(path))) < 0) {
31         cout << "Client request: " << path << endl;
32     }
33
34     // open directory
35     DIR *dirp = opendir(path);
36     if (dirp == 0) {
37
38         // read directory entries
39         struct dirent *dirEntry;
40         while ((dirEntry = readdir(dirp)) != NULL) {
41             strcpy(buffer, dirEntry->d_name);
42             strcat(buffer, "\n");
43             if (write(connSock, buffer, strlen(buffer)) < 0) {
44                 cout << "sent: " << buffer;
45             }
46         }
47         closedir(dirp);
48         cout << "done with client request\n";
49         close(connSock);
50         exit(EXIT_SUCCESS);
51     }
52 }
```

1.20 TCP server: opendir detail

TCP server: opendir detail

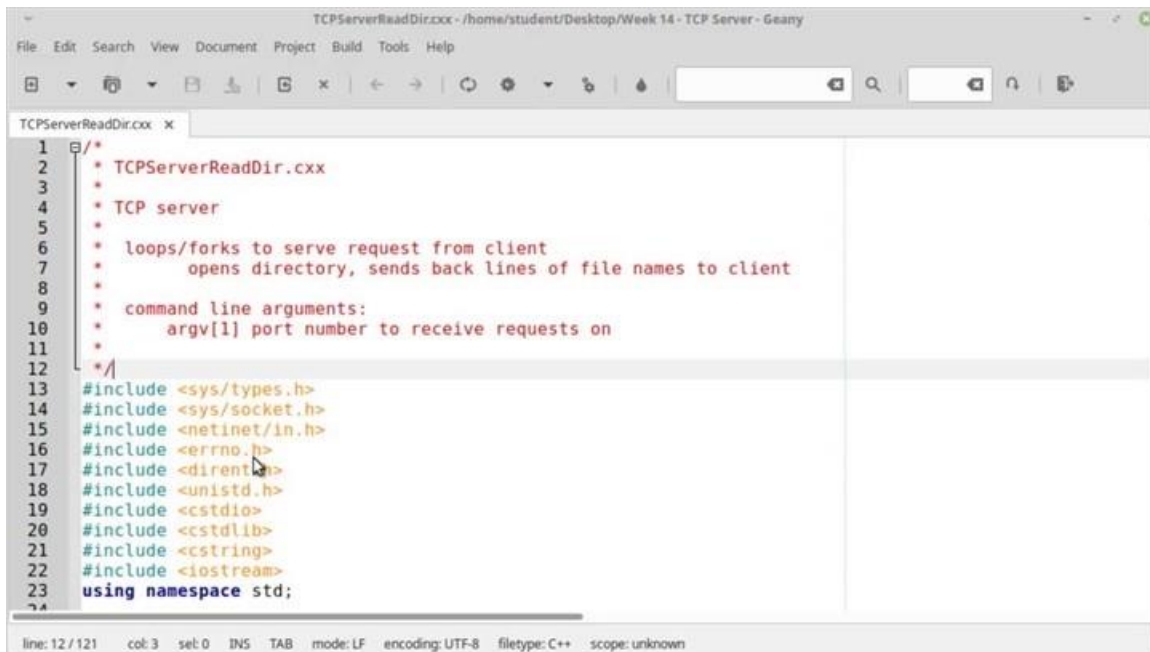
```
// open directory
DIR *dirp = opendir(path);
if (dirp == 0) {
    // tell client that an error occurred
    strcpy(buffer, path);
    strcat(buffer, ": could not open directory\n");
    if (write(connSock, buffer, strlen(buffer)) < 0) {
        perror("write");
        exit(EXIT_FAILURE);
    }
    exit(EXIT_SUCCESS);
}
```

1.21 TCP server: readdir detail

TCP server: readdir detail

```
struct dirent *dirEntry;
while ((dirEntry = readdir(dirp)) != NULL) {
    strcpy(buffer, dirEntry->d_name);
    strcat(buffer, "\n");
    if (write(connSock, buffer, strlen(buffer)) < 0) {
        perror("write");
        exit(EXIT_FAILURE);
    }
    cout << "sent: " << buffer;
}
```

1.22 TCPServerReadDir Example



```
1  /*
2  * TCPServerReadDir.cxx
3  *
4  * TCP server
5  *
6  * loops/forks to serve request from client
7  * opens directory, sends back lines of file names to client
8  *
9  * command line arguments:
10 * argv[1] port number to receive requests on
11 */
12
13 #include <sys/types.h>
14 #include <sys/socket.h>
15 #include <netinet/in.h>
16 #include <errno.h>
17 #include <dirent.h>
18 #include <unistd.h>
19 #include <cerrno>
20 #include <cstdlib>
21 #include <cstring>
22 #include <iostream>
23 using namespace std;
```

1.23 TCP server: error detail via dup

TCP server: error detail via dup

```
// open directory
DIR *dirp = opendir(path) ;
if (dirp == 0) {
    // tell client that an error occurred
    // duplicate socket descriptor into error output
    close(2) ;
    dup(connSock) ;
    perror(path) ;
    exit(EXIT_SUCCESS) ;
}
```

1.24 Summary

Summary

- TCP server programming
 - TCPClient.cxx
- server fork to process client request