


See also the code to be commented in class.




**Week 10 - Friday**

---

## Network Programming

---

School of Information Technology and Electrical Engineering  
The University of Queensland




**Week 10**

---

- Today
  - Network Programming
    - Creatin
    - TCP client and server
- Assignment 3
  - Due Saturday 16 May @ 11pm (plus any available grace days)
  - No submissions after 11pm Thursday 21 May
  - See course website for clarifications and bug fixes
  - Test script released

2




**Creating a Socket**

---

- socket (...) creates a socket descriptor
  - AF\_INET: indicates that the socket is associated with Internet protocols
  - SOCK\_STREAM: selects a reliable byte stream connection (TCP)

```
int fd; /* socket descriptor */
/* Create a TCP socket descriptor */
if ((fd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
{
    perror("Socket creation failed");
    exit(1);
}
```

3



**Socket Address Structures**


---

- Generic socket address:
  - For address arguments to connect, bind, and accept.
  - Necessary only because C did not have generic (void \*) pointers when the sockets interface was designed

```
struct sockaddr {
    unsigned short sa_family; /* protocol family */
    char sa_data[14]; /* address data. */
};
```

- Internet-specific socket address:
  - Must cast (struct sockaddr\_in \*) to (struct sockaddr \*) for connect, bind, and accept

```
struct sockaddr_in {
    unsigned short sin_family; /* address family (always AF_INET) */
    unsigned short sin_port; /* port num in network byte order */
    struct in_addr sin_addr; /* IP addr in network byte order */
    unsigned char sin_zero[8]; /* pad to sizeof(struct sockaddr) */
};
```




**Socket Address Structures**

---

```
struct sockaddr {
    unsigned short sa_family; /* protocol family */
    char sa_data[14]; /* address data. */
};
```

```
struct sockaddr_in {
    unsigned short sin_family; /* address family (always AF_INET) */
    unsigned short sin_port; /* port num in network byte order */
    struct in_addr sin_addr; /* IP addr in network byte order */
    unsigned char sin_zero[8]; /* pad to sizeof(struct sockaddr) */
};
```



**Typical Client (Stream based)**

---

<ul style="list-style-type: none"> <li>● Create socket</li> <li>● Connect to server (at a particular address)</li> <li>● Send/receive data as necessary</li> <li>● Close connection</li> </ul>	<pre>socket(...) connect(...) send(...) or write(...) recv(...) or read(...) close(...)</pre>
--	---

6

COMP2303  
COMP7306

## Sample Client Code

- To be discussed in class

7

COMP2303  
COMP7306

## Short Break

- Stand up and stretch

8

COMP2303  
COMP7306

## Typical Server (Stream based)

- Create socket `socket(...)`
- Bind to address/port `bind(...)`
- Specify willingness to accept connections `listen(...)`
- Block waiting for connection `accept(...)`
  - `accept(...)` returns a new socket
  - Original socket continues to listen
- Deal with request
  - e.g. spawn process `send(...)` or `write(...)`
  - or `recv(...)` or `read(...)`
- Continue

9

COMP2303  
COMP7306

## accept ( )

- `accept(...)` blocks waiting for a connection request

```
int listenfd; /* listening descriptor */
int connfd; /* connected descriptor */
struct sockaddr_in clientaddr;
int clientlen;

clientlen = sizeof(clientaddr);
connfd = accept(listenfd, (struct sockaddr*)&clientaddr, &clientlen);
```

- `accept(...)` returns a *connected descriptor* (`connfd`) with the same properties as the *listening descriptor* (`listenfd`)
  - Returns when the connection between client and server is created and ready for I/O transfers
  - All I/O with the client will be done via the connected socket
- `accept(...)` also fills in client's IP address.

10

COMP2303  
COMP7306

## Socket Options: setsockopt

- The socket can be given some attributes
  - Many are integers

```
int optval = 1;
...
/* Eliminates "Address already in use" error from bind(). */
if (setsockopt(fd, SOL_SOCKET, SO_REUSEADDR,
              (const void *)&optval , sizeof(int)) < 0)
{
    perror("Unable to set socket option");
    exit(1);
}
```

- Handy trick that allows us to rerun a server immediately after we kill it
  - Otherwise we would have to wait about 30 secs+
  - Eliminates "Address already in use" error from `bind()`
  - Useful when debugging

11

COMP2303  
COMP7306

## accept ( ) Illustrated

- Server blocks in `accept`, waiting for connection request on listening descriptor `listenfd`.
- Client makes connection request by calling and blocking in `connect`.
- Server returns `connfd` from `accept`. Client returns from `connect`. Connection is now established between `clientfd` and `connfd`.

12

COMP2303  
COMP7306

## Connected vs. Listening Descriptors

- **Listening** descriptor
  - End point for client connection requests
  - Created once and exists for lifetime of the server
- **Connected** descriptor
  - End point of the connection between client and server
  - A new descriptor is created each time the server accepts a connection request from a client
  - Exists only as long as it takes to service client
- Why the distinction?
  - Allows for concurrent servers that can communicate over many client connections simultaneously
    - e.g., each time we receive a new request, we fork a child process to handle the request

13

COMP2303  
COMP7306

## Echo Server: Identifying the Client

- The server can determine the domain name and IP address of the client

```

struct hostent *hp; /* pointer to DNS host entry */
char *haddrp;      /* pointer to dotted decimal string */

hp = gethostbyaddr((const char *)&clientaddr.sin_addr.s_addr,
                  sizeof(clientaddr.sin_addr.s_addr), AF_INET);
haddrp = inet_ntoa(clientaddr.sin_addr);
printf("server connected to %s (%s)\n", hp->h_name, haddrp);
    
```

14

COMP2303  
COMP7306

## Testing Servers Using telnet

- The telnet program is invaluable for testing servers that transmit ASCII strings over Internet connections
  - Our simple server
  - Web servers
  - Mail servers
- Usage:
  - `unix> telnet <host> <portnumber>`
  - Creates a connection with a server running on <host> and listening on port <portnumber>
  - Ctrl-] to get telnet prompt, "quit" to exit

15

COMP2303  
COMP7306

## Sample Server Code

- *To be discussed in class*

16

COMP2303  
COMP7306

## Iterative Servers

- Iterative servers process one request at a time

```

sequenceDiagram
    participant C1 as client 1
    participant S as server
    participant C2 as client 2

    C1->>S: call connect
    S-->>C1: ret connect
    C1->>S: call read
    S-->>C1: ret read
    C1->>S: close
    S->>S: call accept
    S-->>C2: ret accept
    C2->>S: call connect
    S-->>C2: ret connect
    S->>C2: write
    C2->>S: ret read
    S->>S: close
    
```

17

COMP2303  
COMP7306

## Resources

- Beej's Guide to Network Programming
  - <http://beej.us/guide/bgnet/>
- UNIX manual pages
  - On Solaris: `man -s 3socket <name>`
  - where <name> is socket, bind, connect, listen, accept, recv, send, ...
- Glass & Ables, "UNIX for Programmers and Users"
- Rockkind, "Advanced UNIX Programming"
- Bryant and O'Halloran, "Computer Systems: A Programmer's Perspective"
- Other UNIX Programming books...
  - See Reference text list in course profile

18

COMP2303  
COMP7306

## Coming Up

- Week 11
  - Tuesday – Network Applications
  - Friday – Network Programming (cont.)
- Week 12
  - Tuesday – Network Programming (cont.)
  - Ethernet
- Week 13
  - Internet Layer
  - Review

19

20

21

22

23

24