

CSC 4304 - Systems Programming
Fall 2008

LECTURE - XVI
SOCKETS

Tevfik Koşar

Louisiana State University
November 4th, 2008

Project 2: Simple Web Server

Basics of a Web Server

- Listen to a Network port
- Interpret incoming messages (requests)
- Serve requests
 - Read requested files
 - Send them over network
- Run consistently in the background

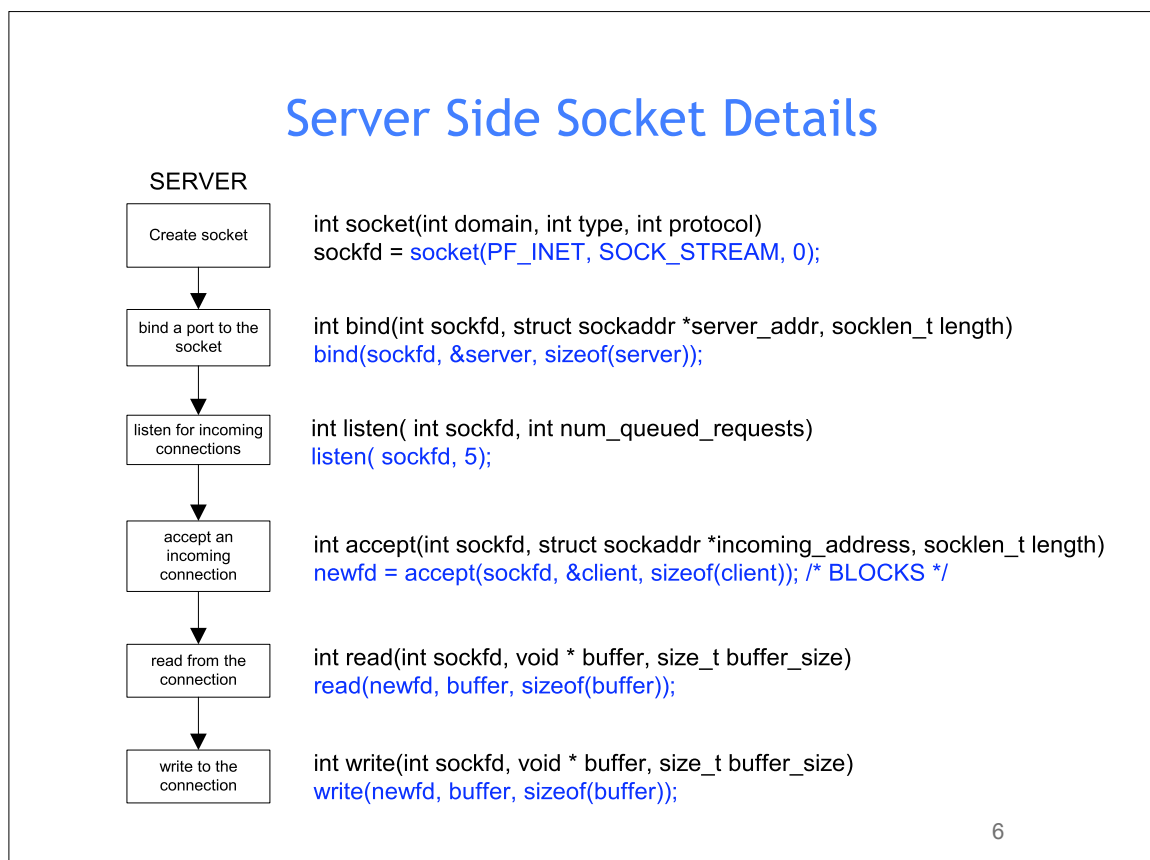
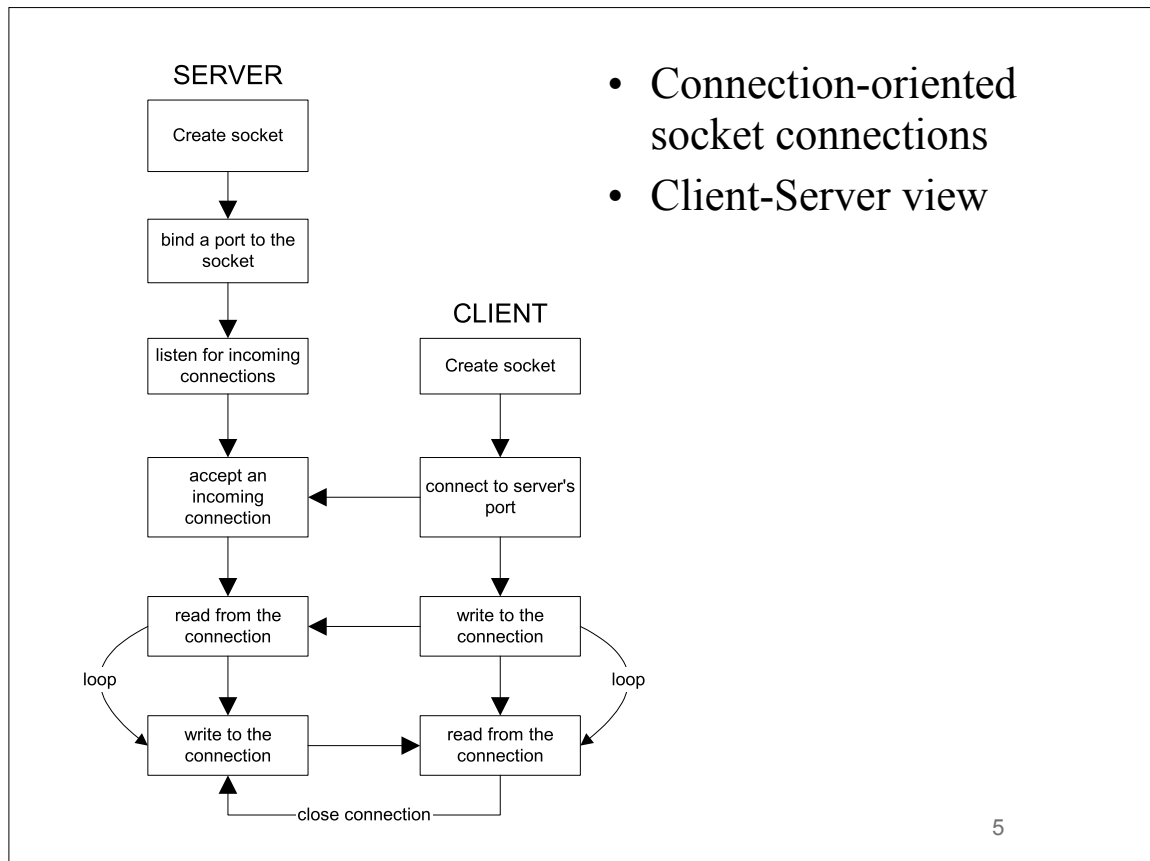
3

Creating a Socket

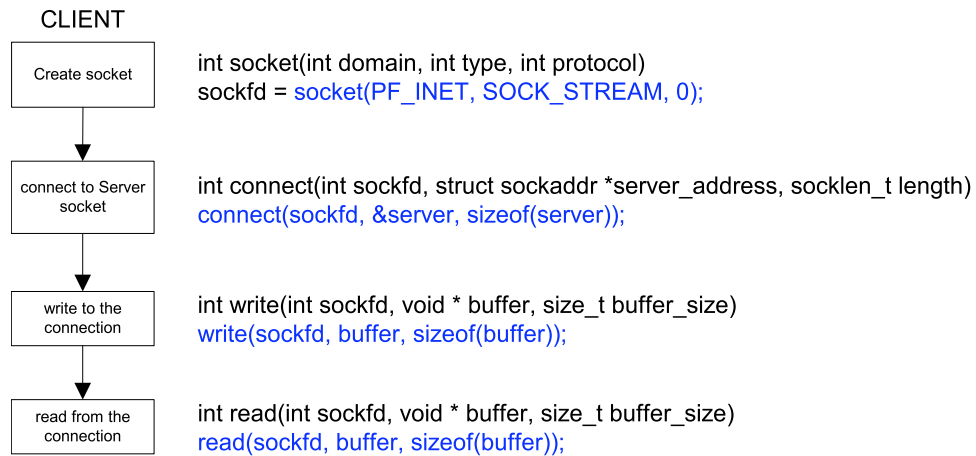
```
#include <sys/types.h>
#include <sys/socket.h>
int socket(int domain, int type, int protocol);
```

- **domain** is one of the *Address Families* (AF_INET, AF_UNIX, etc.)
- **type** defines the communication protocol semantics, usually defines either:
 - SOCK_STREAM: connection-oriented stream (TCP)
 - SOCK_DGRAM: connectionless, unreliable (UDP)
- **protocol** specifies a particular protocol, just set this to 0 to accept the default (PF_INET, PF_UNIX) based on the domain

4



Client Side Socket Details



7

Setup for an Internet Domain Socket

```
struct sockaddr_in {
    sa_family_t sin_family;
    unsigned short int sin_port;
    struct in_addr sin_addr;
    unsigned char pad[...];
};
```

- `sin_family` is set to *Address Family* `AF_INET`
- `sin_port` is set to the port number you want to *bind* to
- `sin_addr` is set to the IP address of the machine you are binding to (struct `in_addr` is a wrapper struct for an unsigned long). `INADDR_ANY` supports all interfaces (since a given machine may have multiple interface cards)
- ignore padding

8

Reading/Writing to/from Sockets

- Sockets, like everything else, are like *files*:
 - low level IO:
 - read() system call
 - write() system call
 - higher level IO:
 - int recv(int socket, char *buf, int len, int flags);
 - blocks on read
 - returns 0 when other connection has terminated
 - int send(int socket, char *buf, int len, int flags);
 - returns the number of bytes *actually sent*
 - where flags may be one of:
 - MSG_DONTROUTE (don't route out of localnet)
 - MSG_OOB (out of band data (think *interruption*))
 - MSG_PEEK (examine, but don't remove from stream)

9

Closing a Socket Session

- int close(int socket);
 - closes read/write IO, closes socket file descriptor
- int shutdown(int sockfd, int how);
 - where how is:
 - 0: no more receives allowed
 - 1: no more sends are allowed
 - 2: disables both receives and sends (but doesn't close the socket, use close() for that)
- *Example*: hangserver.c (hangman game)

10

Select()

```
int select(int numfiledescs, fd_set readfdsset, fd_set  
writefdsset, fd_set errorfdsset, struct timeval * timeout);
```

- The select() system call provides a way for a single server to wait until a set of network connections has data available for reading
- The advantage over fork() here is that no multiple processes are spawned
- The downside is that the single server must handle state management on its own for all its new clients

11

Select() cont..

- select() will return if *any* of the descriptors in readfdsset and writefdsset of file descriptors are ready for reading or writing, respectively, or, if any of the descriptors in errorfdsset are in an error condition
- The FD_SET(int fd, fd_set *set) function will add the file descriptor *fd* to the set *set*
- The FD_ISSET(int fd, fd_set *set) function will tell you if filedesc *fd* is in the modified set *set*
- select() returns the total number of descriptors in the modified sets
- If a client closes a socket whose file descriptor is in one of your watched sets, select() will return, and your next recv() will return 0, indicating the socket has been closed

12

More Socket Functions

- `int getpeername(int sockfd, struct sockaddr * addr, int *addrlen);`
 - this tells you the *hostname* of the REMOTE connection
- `int gethostname(char * hostname, size_t size);`
 - this tells you the *hostname* of your LOCAL connection
- `int inet_aton(const char * string_address, &(addr.sin_addr));`
 - converts the const ip *string_address* (“192.168.3.1”) into an acceptable numeric form
- `addr.sin_addr = inet_addr(“192.168.3.1”);`
 - does the same thing

13

More Socket Functions (*cont.*)

- `struct hostent *gethostbyname(const char *hostname);`
 - Does a DNS lookup and returns a pointer to a `hostent` structure that contains the host name, aliases, address type (AF_INET, etc.), length, and an array of IP addresses for this host (hostent.h_addr_list[0] is usually the one)
(cf. /etc/nsswitch.conf)
- ```
struct hostent {
 char *h_name; /*DNS host name*/
 char **h_aliases; /*alias list*/
 int h_addrtype; /* "AF_INET", etc*/
 int h_length; /*_length of addr*/
 char **h_addr_list; /*list of IP
adds*/
};
```

14

## Example: A Time Server

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define PORTNUM 8824
#define oops(msg) { perror(msg) ; exit(1) ; }
```

15

```
void main(int ac, char **av)
{
 struct sockaddr_in saddr; /* build our address here */
 struct hostent *hp; /* this is part of our */
 char hostname[256]; /* address */
 int slen, sock_id, sock_fd; /* line id, file desc */
 FILE *sock_fp; /* use socket as stream */
 char *ctime(); /* convert secs to string */
 long time(), thetime; /* time and the val */

 gethostname(hostname , 256); /* where am I ? */
 hp = gethostbyname(hostname); /* get info about host */
 bzero(&saddr, sizeof(saddr)); /* zero struct */
 /* fill in hostaddr */
 bcopy(hp->h_addr, &saddr.sin_addr, hp->h_length);
 saddr.sin_family = AF_INET; /* fill in socket type */
 saddr.sin_port = htons(PORTNUM); /* fill in socket port */

 sock_id = socket(AF_INET, SOCK_STREAM, 0); /* get a socket */
 if (sock_id == -1) oops("socket");

 if (bind(sock_id, &saddr, sizeof(saddr)) != 0) /* bind it to */
 oops("bind"); /* an address */

 if (listen(sock_id, 1) != 0) oops("listen");
}
```

16

```

while (1){
 sock_fd = accept(sock_id, NULL, NULL); /* wait for call */
 printf("*** Server: A new client connected!");
 if (sock_fd == -1)
 oops("accept"); /* error getting calls */

 show_my_port(sock_fd);
 sock_fp = fdopen(sock_fd,"w"); /* we'll write to the */
 if (sock_fp == NULL) /* socket as a stream */
 oops("fdopen"); /* unless we can't */

 thetime = time(NULL); /* get time */
 /* and convert to string */
 fprintf(sock_fp, "*****\n");
 fprintf(sock_fp, "*** From Server: The current time is: ");
 fprintf(sock_fp, "%s", ctime(&thetime));
 fprintf(sock_fp, "*****\n");

 fclose(sock_fp); /* release connection */
 fflush(stdout); /* force output */
}
}

```

17

## Acknowledgments

- Advanced Programming in the Unix Environment by R. Stevens
- The C Programming Language by B. Kernighan and D. Ritchie
- Understanding Unix/Linux Programming by B. Molay
- Lecture notes from B. Molay (Harvard), T. Kuo (UT-Austin), G. Pierre (Vrije), M. Matthews (SC), B. Knicki (WPI), M. Shacklette (UChicago), J. Kim (KAIST), and J. Schaumann (SIT).

18