456: Network-Centric Programming

Yingying (Jennifer) Chen

Web: https://www.winlab.rutgers.edu/~yychen/
Email: yingche@scarletmail.rutgers.edu

Department of Electrical and Computer Engineering
Rutgers University

Review: UDP vs. TCP

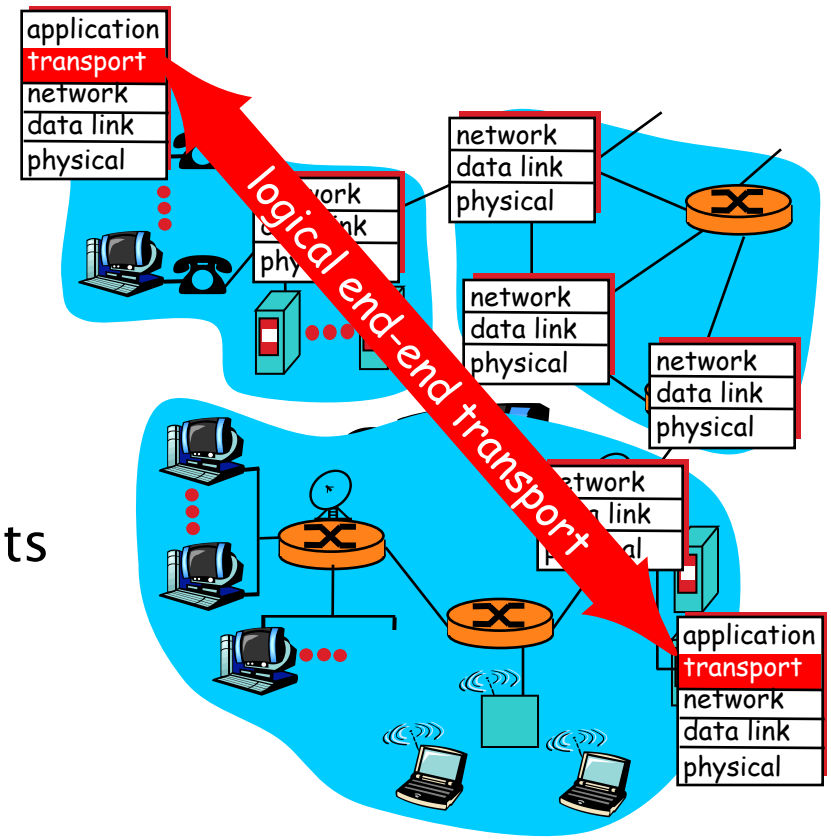
- **Transport-layer protocols** in the Internet
  - User Datagram Protocol (UDP)
  - Transmission Control Protocol (TCP)

Role of Transport Layer

- Application layer
  - Communication for specific applications
  - E.g., HyperText Transfer Protocol (HTTP), File Transfer Protocol (FTP), Network News Transfer Protocol (NNTP)
- Transport layer
  - Communication between processes (e.g., socket)
  - Relies on network layer and serves the application layer
  - E.g., TCP and UDP
- Network layer
  - Logical communication between nodes
  - Hides details of the link technology
  - E.g., IP

Transport Protocols

- Provide *logical communication* between application processes running on different hosts
- Run on end hosts
  - Sender: breaks application messages into segments, and passes to network layer
  - Receiver: reassembles segments into messages, passes to application layer
- Multiple transport protocol available to applications
  - Internet: TCP and UDP



Internet Transport Protocols

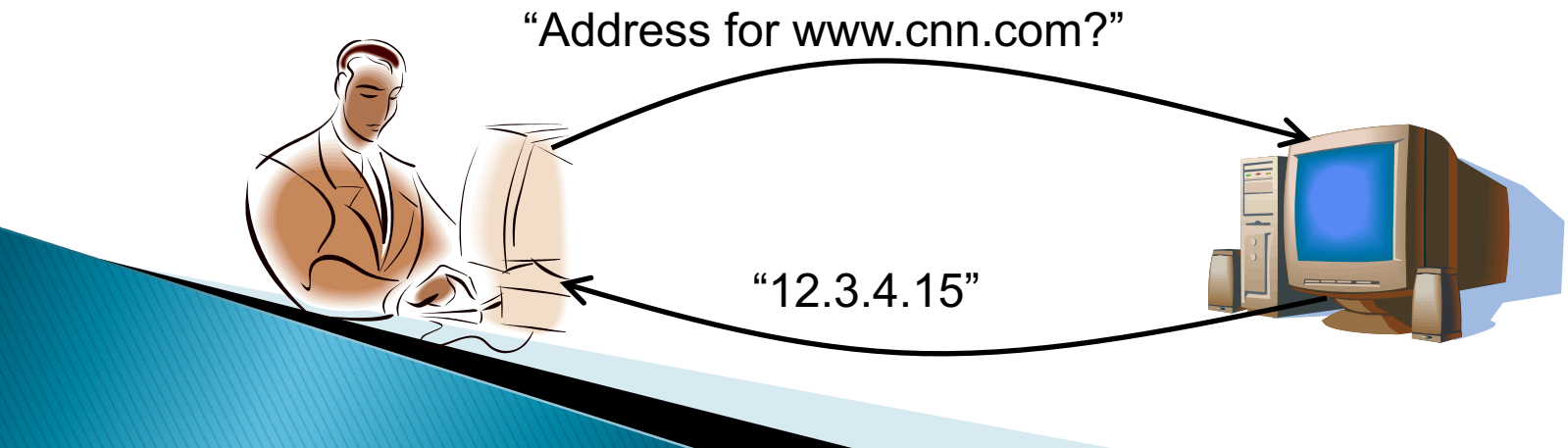
- Datagram messaging service (UDP)
  - Connectionless, unreliable, packet-oriented
- Reliable, in-order delivery (TCP)
  - Connection set-up
  - Discarding of corrupted packets
  - Retransmission of lost packets
  - Flow control
  - Congestion control
- Other services not available
  - Delay guarantees
  - Bandwidth guarantees

Why Would Anyone Use UDP?

- Finer control over what data is sent and when
  - As soon as an application process writes into the socket
    - UDP will package the data and send the packet
- No delay for connection establishment
  - UDP just blasts away without any formal preliminaries
    - which avoids introducing any unnecessary delays
- No connection state
  - No allocation of buffers, parameters, sequence #s, etc.
    - making it easier to handle many active clients at once
- Small packet header overhead
  - UDP header is only eight-bytes long

Popular Applications That Use UDP

- Multimedia streaming
  - Retransmitting lost/corrupted packets is not worthwhile
  - By the time the packet is retransmitted, it's too late
  - E.g., Internet phone calls, video conferencing, gaming
- Simple query protocols like Domain Name System
  - Overhead of connection establishment is overkill
  - Easier to have application retransmit if needed



Transmission Control Protocol (TCP)

- Connection oriented
  - Explicit set-up and tear-down of TCP session
- Stream-of-bytes service
  - Sends and receives a stream of bytes, not messages
- Reliable, in-order delivery
  - Checksums to detect corrupted data
  - Acknowledgments & retransmissions for reliable delivery
  - Sequence numbers to detect losses and reorder data
- Flow control
  - Prevent overflow of the receiver's buffer space
- Congestion control
  - Adapt to network congestion for the greater good

An Analogy: Talking on a Cell Phone

- Alice and Bob on their cell phones
  - Both Alice and Bob are talking
- What if Alice couldn't understand Bob?
  - Alice asks Bob to repeat what he said
- What if Bob hasn't heard Alice for a while?
  - Is Alice just being quiet?
  - Or, have Bob and Alice lost reception?
  - How long should Bob just keep on talking?
  - Maybe Alice should periodically say "uh huh"
  - … or Bob should ask "Can you hear me now?" ☺

Some Take-Aways from the Example

- Acknowledgments from receiver
  - Positive: “okay” or “ACK”
  - Negative: “please repeat that” or “NACK”
- Timeout by the sender (“stop and wait”)
  - Don’t wait indefinitely without receiving some response
  - … whether a positive or a negative acknowledgment
- Retransmission by the sender
  - After receiving a “NACK” from the receiver
  - After receiving no feedback from the receiver

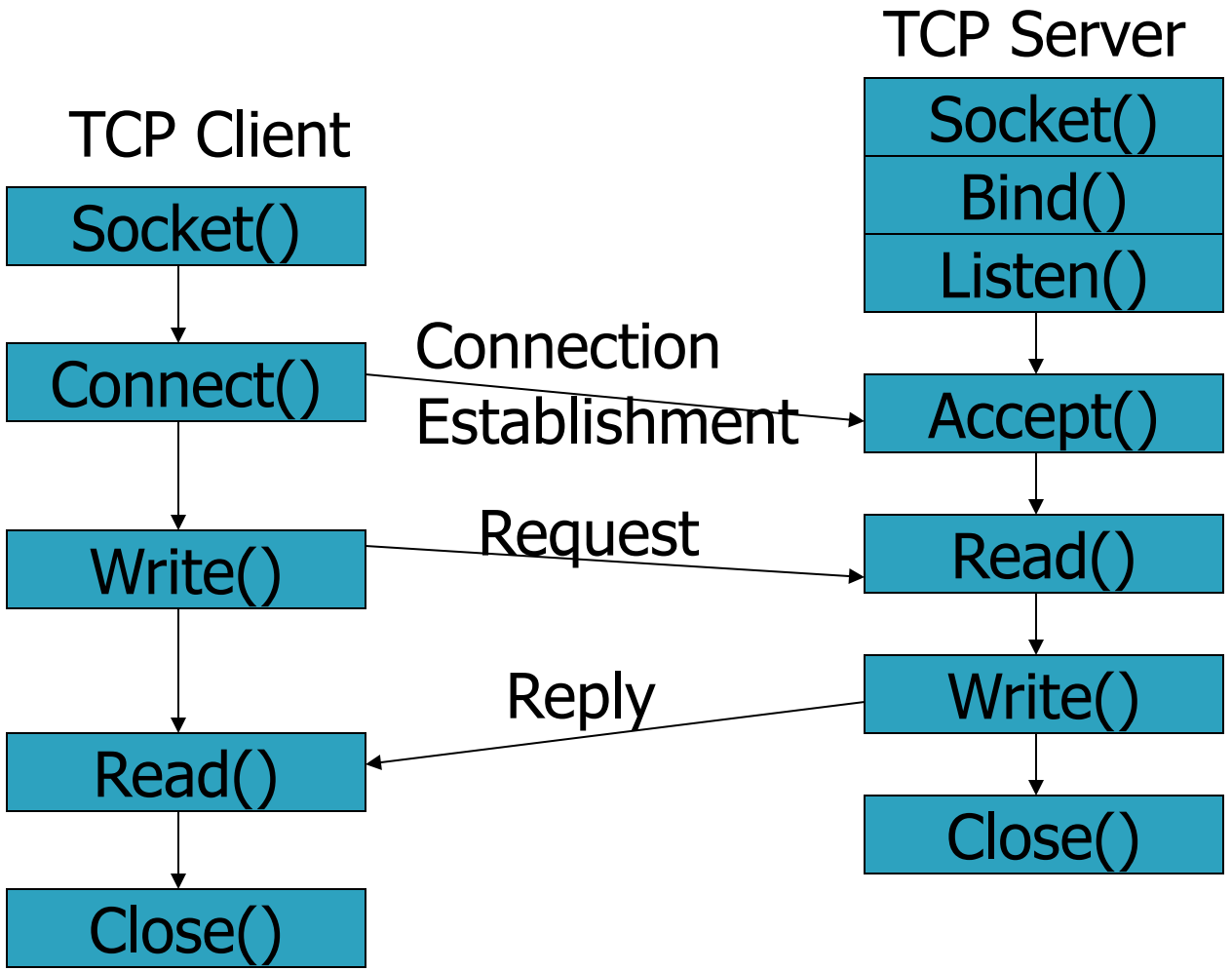
Challenges of Reliable Data Transfer

- Over a perfectly reliable channel
  - All of the data arrives in order, just as it was sent
  - Simple: sender sends data, and receiver receives data
- Over a channel with bit errors
  - All of the data arrives in order, but some bits corrupted
  - Receiver detects errors and says “please repeat that”
  - Sender retransmits the data that were corrupted
- Over a lossy channel with bit errors
  - Some data are missing, and some bits are corrupted
  - Receiver detects errors but cannot always detect loss
  - Sender must wait for acknowledgment (“ACK” or “OK”)
  - … and retransmit data after some time if no ACK arrives

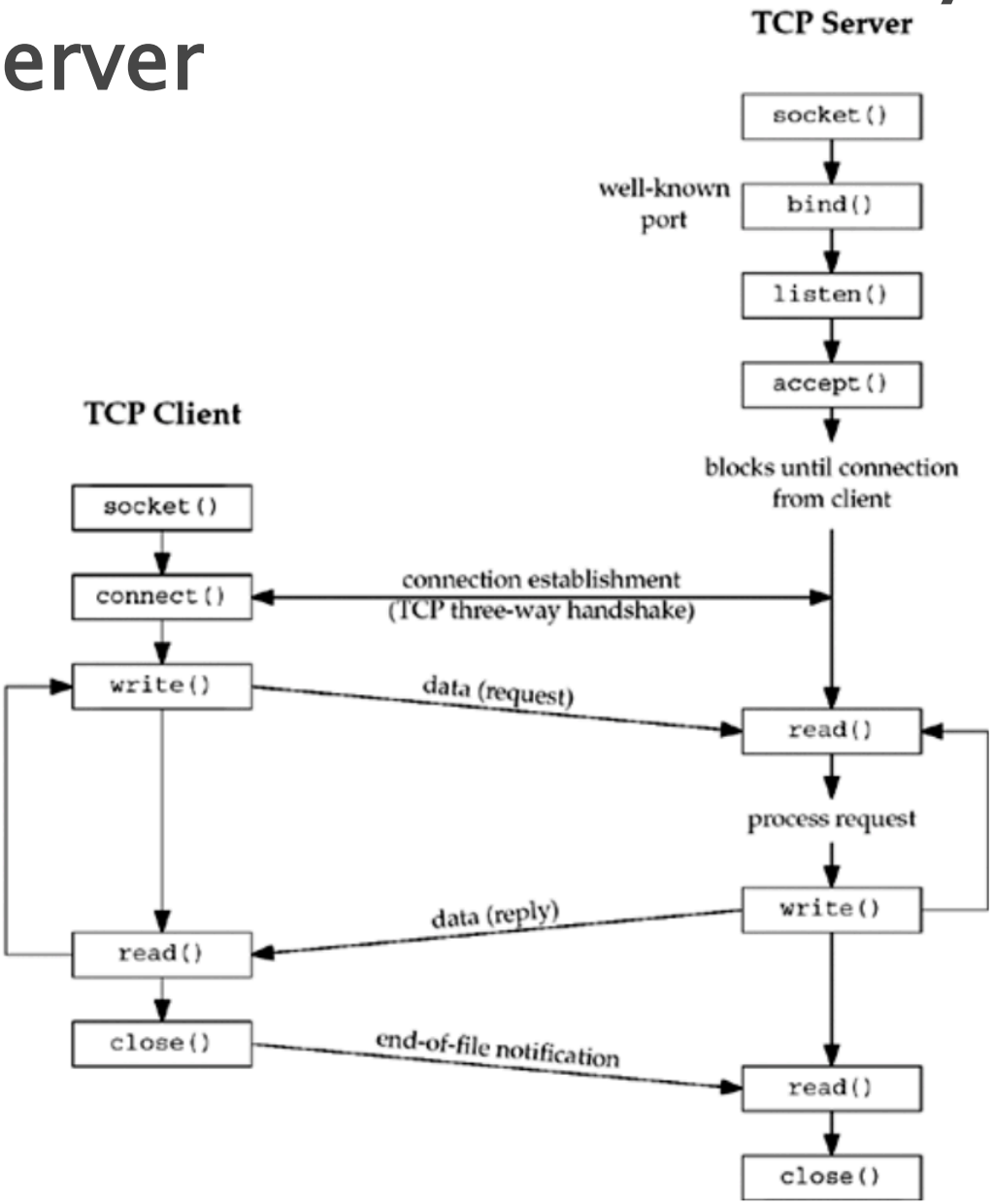
TCP Support for Reliable Delivery

- Checksum
  - Used to detect corrupted data at the receiver
  - …leading the receiver to drop the packet
- Sequence numbers
  - Used to detect missing data
  - … and for putting the data back in order
- Retransmission
  - Sender retransmits lost or corrupted data
  - Timeout based on estimates of round-trip time
  - Fast retransmit algorithm for rapid retransmission

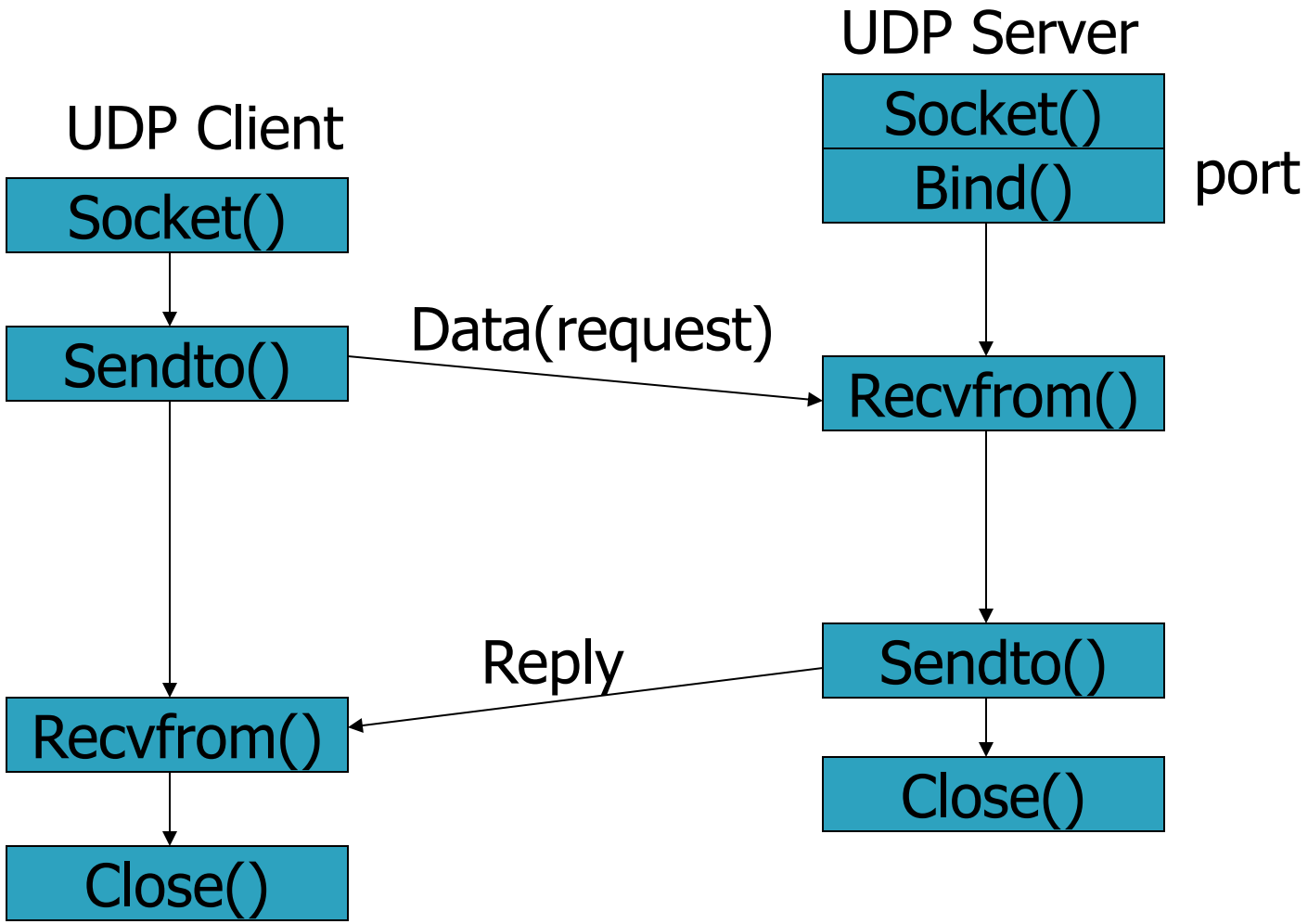
Socket Functions (TCP)



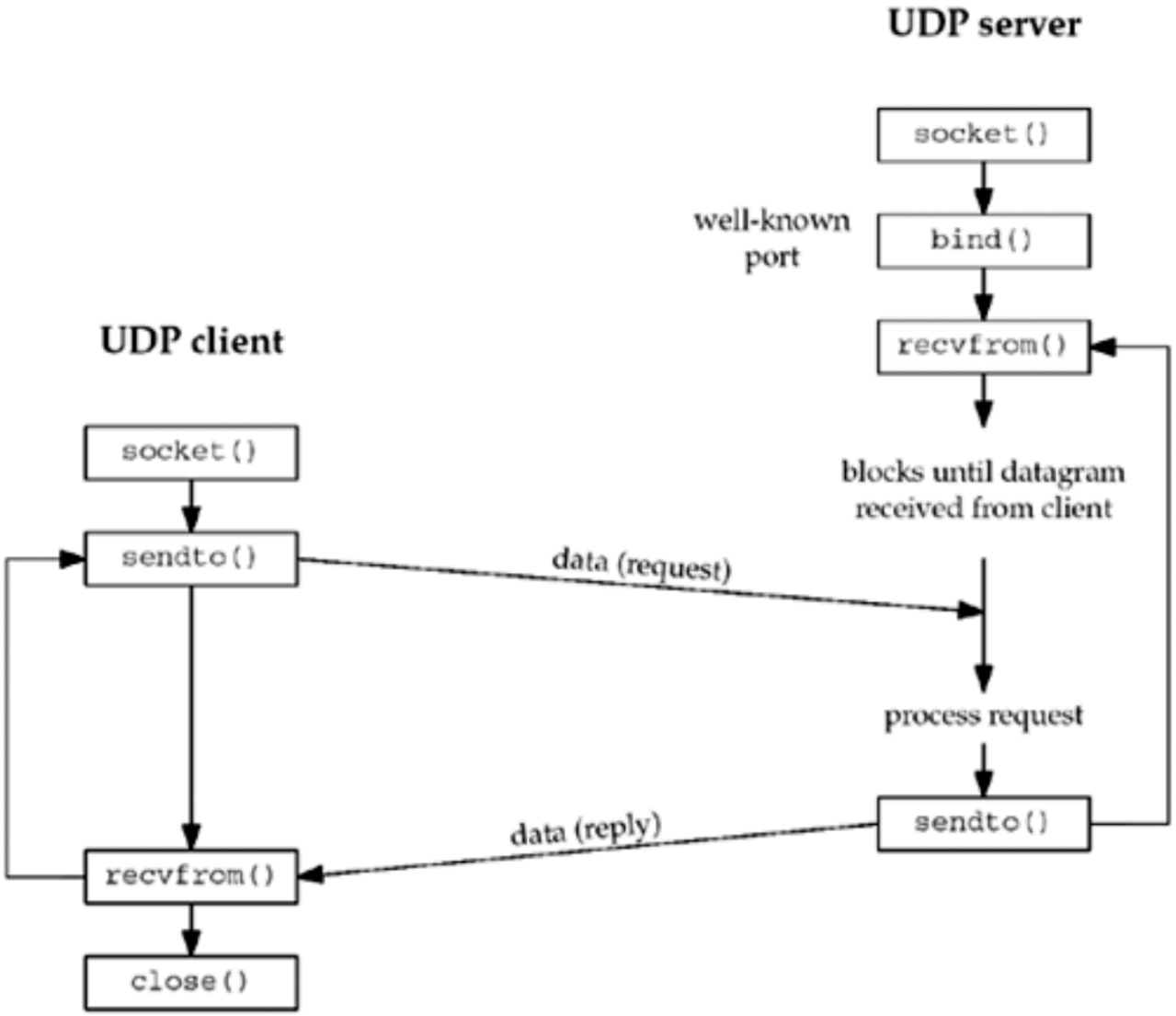
Socket functions for elementary TCP client/server



Socket Functions (UDP)



Socket functions for UDP client/server



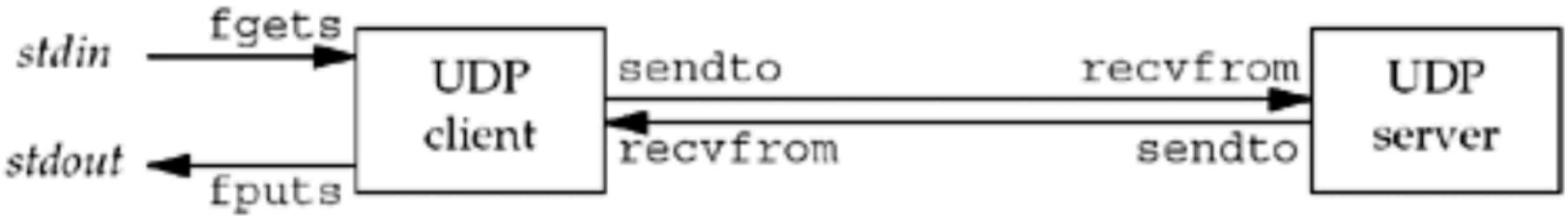
recvfrom() / sendto()

- #include <sys/socket.h>
- ssize_t recvfrom(int sockfd, void *buff, size_t nbytes, int flags, struct sockaddr* from, socklen_t *addrlen)

- ssize_t sendto(int sockfd, const void* buff, size_t nbytes, int flags, const struct sockaddr *to, socklen_t addrlen)

- Arguments
  - sockfd: socket file descriptor
  - buff: Allocated buffer
  - nbytes: Number of bytes to read or write
  - flags: 0, MSG_DONTWAIT, …
    (For now, we will always set the flags to 0)
  - from/to: sender/destination address
  - addrlen: socket address length

Example (UDP echo server)



Simple echo client/server using UDP

1. The client reads a line of text from its standard input and writes the line to the server.
2. The server reads the line from its network input and echoes the line back to the client.
3. The client reads the echoed line and prints it on its standard output.

Example (UDP echo server): lecture_10_code/udp_echo_server/server.c

```
#include <stdio.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <string.h>
#include <unistd.h>

#define PORT 8080
#define MAXLINE 4096

int main(int argc, char const *argv[])
{
    int sockfd;
    struct sockaddr_in  servaddr, cliaddr;
```

Example (UDP echo server): lecture_10_code/udp_echo_server/server.c

```
    sockfd = socket(AF_INET, SOCK_DGRAM, 0); //create UDP socket

    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family      = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port        = htons(PORT);

    bind(sockfd, (struct sockaddr *) &servaddr, sizeof(servaddr));
    // bind the socket address to the socket
    printf("The UDP server is on.\n");

    udp_echo(sockfd, (struct sockaddr *) &cliaddr, sizeof(cliaddr));

}
```

Example (UDP echo server): lecture_10_code/udp_echo_server/server.c

```
void udp_echo(int sockfd, struct sockaddr *pcliaddr, socklen_t clilen)
{
    int         n;
    socklen_t   len;
    char        msg[MAXLINE+1];

    for ( ; ; ) {
        len = clilen;
        n = recvfrom(sockfd, msg, MAXLINE, 0, pcliaddr, &len);
        //receive message from the client

        sendto(sockfd, msg, n, 0, pcliaddr, len);
        //send message back to the client

        msg[n] = 0;     /* null terminate */
        printf("Received and sent: %s\n", msg); //print out the message
    }
}
```

Example (UDP echo server): lecture_10_code/udp_echo_server/client.c

```
#include <stdio.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#define PORT 8080
#define MAXLINE 4096

int main(int argc, char const *argv[])
{
    int sockfd;
    struct sockaddr_in serv_addr;
```

Example (UDP echo server): lecture_10_code/udp_echo_server/client.c

```
memset(&serv_addr, '0', sizeof(serv_addr));
serv_addr.sin_family = AF_INET;
serv_addr.sin_port = htons(PORT);
// Convert IP addresses from text to binary form
if(inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr)<=0) {
    printf("\nInvalid address/ Address not supported \n");
    return -1;
}

if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0){ //create UDP socket
    printf("\n Socket creation error \n");
    return -1;
}
printf("The UDP client is on.\n");
echo_client(stdin, sockfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr));
exit(0);
}
```

Example (UDP echo server): lecture_10_code/udp_echo_server/client.c

```
void
echo_client(FILE *fp, int sockfd, const struct sockaddr *pservaddr, socklen_t
servlen)
{
    int n;
    char    sendline[MAXLINE], recvline[MAXLINE + 1];

    while (fgets(sendline, MAXLINE, fp) != NULL) {
        // send the message to the server
        sendto(sockfd, sendline, strlen(sendline), 0, pservaddr, servlen);
        // receive the message from the server
        n = recvfrom(sockfd, recvline, MAXLINE, 0, NULL, NULL);

        recvline[n] = 0;    /* null terminate */
        fputs(recvline, stdout); //print out what the client just received
    }
}
```

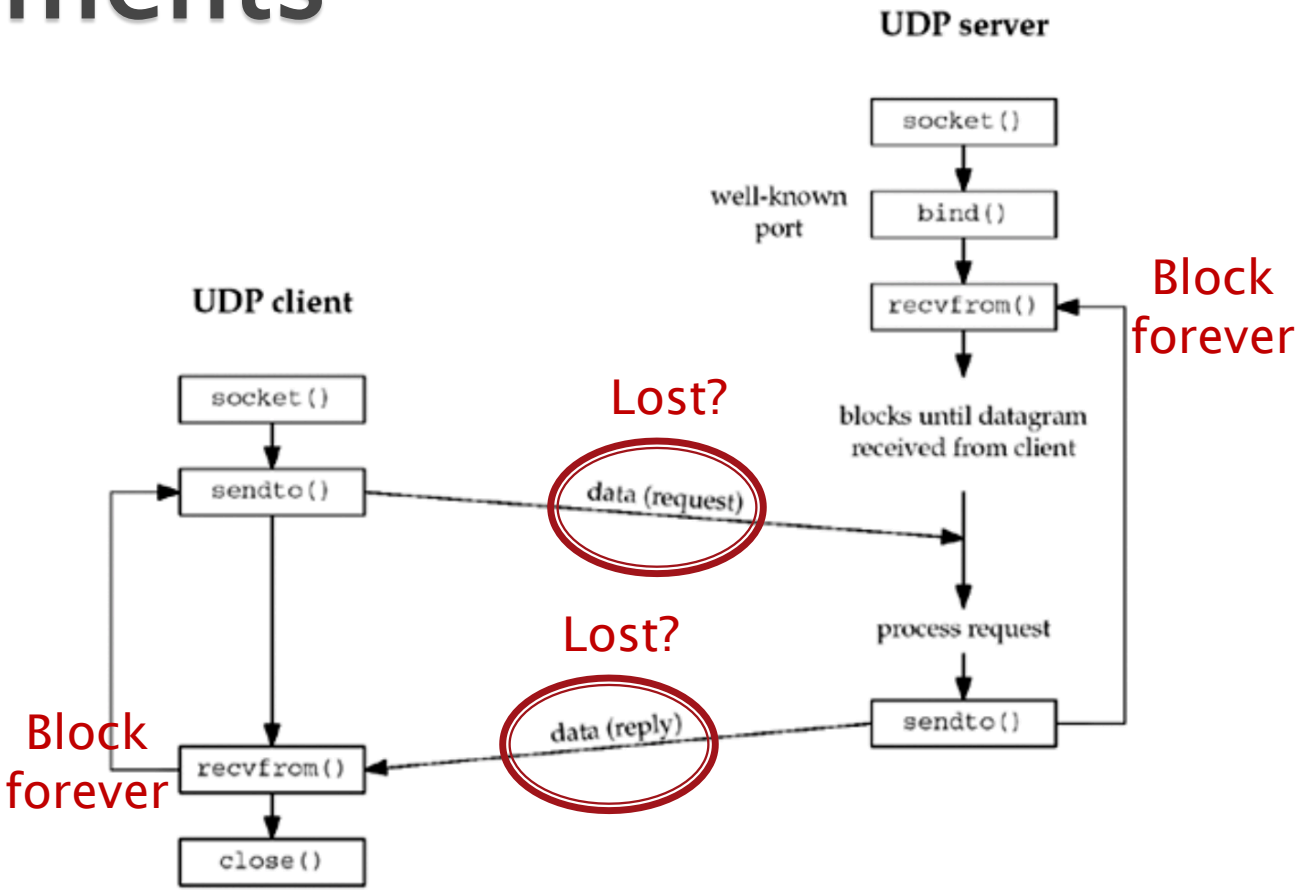
Terminal 1:
$ ./server

Terminal 2:
$ ./client

Comments

- Unreliable
  - If a client datagram is lost (say it is discarded by some router between the client and server), the server will block forever in its call to *recvfrom()*.
  - Similarly, if the client datagram arrives at the server but the server's reply is lost, the client will again block forever in its call to *recvfrom().*

Comments



Solution: A typical way to prevent this is to place a timeout on the server or client's call to `recvfrom()`.

Socket Timeouts

- Two ways to place a timeout on an I/O operation involving a socket:
  - Call *alarm()*, which generates the SIGALRM signal when the specified time has expired.
  - Block waiting for I/O in *select()*, which has a time limit built-in, instead of blocking in a call to *recvfrom()*.

recvfrom() with a Timeout Using SIGALRM

- Call *alarm()*, which generates the SIGALRM signal when the specified time has expired.

```
#include <unistd.h>

unsigned alarm(unsigned seconds);
```

- *DESCRIPTION*
  - The *alarm*() function shall cause the system to generate a SIGALRM signal for the process after the number of realtime seconds specified by *seconds* have elapsed.
  - If *seconds* is 0, a pending alarm request, if any, is canceled.

recvfrom() with a Timeout Using SIGALRM

- signal() – ANSI C signal handling

```
#include <signal.h>

typedef void (*sighandler_t)(int);

sighandler_t signal(int signum, sighandler_t handler);
```

- *DESCRIPTION*
  - signal() sets the disposition of the signal *signum* to *handler*, which could be the address of a programmer-defined function (a "signal handler").
    - *signum:* SIGALRM signal is raised when a time interval specified in a call to the alarm function expires.

recvfrom() with a Timeout Using SIGALRM

- siginterrupt() – allow signals to interrupt system calls

```
#include <signal.h>

int siginterrupt(int sig, int flag);
```

- *DESCRIPTION*
  - If the *flag* argument is false (0), then system calls will be restarted if interrupted by the specified signal *sig*. This is the default behavior in Linux.
  - If the *flag* argument is true (1), then a system call interrupted by the signal *sig* will return -1 and *errno* will be set to EINTR.

Note:

we usually use siginterrupt(SIGALRM, 1); along with the signal() to place timeout in the *recvfrom().*

siginterrupt(SIGALRM, 0); the recvfrom() will be restarted if interrupted by the signal.

Example (UDP echo server with SIGALRM timeout): lecture_10_code/udp_echo_server_alarm/server.c

```
#include <stdio.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <signal.h>

#define PORT 8080
#define MAXLINE 4096

static void sig_alrm(int);
```

Example (UDP echo server with SIGALRM timeout): lecture_10_code/udp_echo_server_alarm/server.c

```
int main(int argc, char const *argv[])
{
    int sockfd;
    struct sockaddr_in  servaddr, cliaddr;

    sockfd = socket(AF_INET, SOCK_DGRAM, 0); // create UDP socket

    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family      = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port        = htons(PORT);
    // bind the socket address to the socket file descriptor
    bind(sockfd, (struct sockaddr *) &servaddr, sizeof(servaddr));

    printf("The UDP server is on.\n");
    udp_echo(sockfd, (struct sockaddr *) &cliaddr, sizeof(cliaddr));
}
```

Example (UDP echo server with SIGALRM timeout): lecture_10_code/udp_echo_server_alarm/server.c

```
void udp_echo(int sockfd, struct sockaddr *pcliaddr, socklen_t clilen){
    int         n;
    socklen_t   len;
    char        msg[MAXLINE + 1];
    siginterrupt(SIGALRM, 1);  /* a system call interrupted by the signal will
return -1 and errno will be set to EINTR. */

    signal(SIGALRM, sig_alrm); // establish a signal handler for SIGALRM
    for ( ; ; ) {
        len = clilen;
        alarm(5); // generate a SIGALRM signal after 5 seconds
        //receive message from the client
        if((n = recvfrom(sockfd, msg, MAXLINE, 0, pcliaddr, &len)) < 0){
            if (errno == EINTR)
                fprintf(stderr, "socket timeout\n");
            else
                perror("recvfrom error");
        }else{
```

Example (UDP echo server with SIGALRM timeout): lecture_10_code/udp_echo_server_alarm/server.c

```
            alarm(0); // cancel the pending alarm request
            sendto(sockfd, msg, n, 0, pcliaddr, len);
            //send message back to the client

            msg[n] = 0;     /* null terminate */
            printf("Received and sent: %s\n", msg); //print out the message
        }
    }
}


static void sig_alrm(int signo)
{
        printf("The signal handler is working.\n");
        return; /* just interrupt the recvfrom() */
}
```

Example (UDP echo server with SIGALRM timeout): lecture_10_code/udp_echo_server_alarm/client.c

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

#define PORT 8080
#define MAXLINE 4096
```

Example (UDP echo server with SIGALRM timeout): lecture_10_code/udp_echo_server_alarm/client.c

```
int main(int argc, char const *argv[]){
    int sockfd;
    struct sockaddr_in serv_addr;
    memset(&serv_addr, '0', sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);
    // Convert IP addresses from text to binary form
    if(inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr)<=0) {
        printf("\nInvalid address/ Address not supported \n");
        return -1;
    }
    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0){
        printf("\n Socket creation error \n");
        return -1;
    }
    printf("The UDP client is on.\n");
    echo_client(stdin, sockfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr));
    exit(0);
}
```

Example (UDP echo server with SIGALRM timeout): lecture_10_code/udp_echo_server_alarm/client.c

```
void
echo_client(FILE *fp, int sockfd, const struct sockaddr *pservaddr, socklen_t servlen)
{
    int n;
    char    sendline[MAXLINE], recvline[MAXLINE + 1];

    while (fgets(sendline, MAXLINE, fp) != NULL) {

        sendto(sockfd, sendline, strlen(sendline), 0, pservaddr, servlen);

        n = recvfrom(sockfd, recvline, MAXLINE, 0, NULL, NULL);

        recvline[n] = 0;    /* null terminate */
        fputs(recvline, stdout);
    }
}
```

Terminal 1:
$ ./server

Terminal 2:
$ ./client

Example (UDP echo server with SIGALRM timeout): lecture_10_code/udp_echo_server_alarm/client1.c

Refer client1.c to see how to place timeout in the client's recvfrom().

```
void echo_client(FILE *fp, int sockfd, const struct sockaddr *pservaddr, socklen_t
servlen){
    int n;
    char    sendline[MAXLINE], recvline[MAXLINE + 1];
    siginterrupt(SIGALRM, 1);  /* a system call interrupted by the signal will return -1
and errno will be set to EINTR. */
    signal(SIGALRM, sig_alrm); // establish a signal handler for SIGALRM

    while (fgets(sendline, MAXLINE, fp) != NULL) {
        sendto(sockfd, sendline, strlen(sendline), 0, pservaddr, servlen);
        alarm(5); // generate a SIGALRM signal after 5 seconds

        if ((n = recvfrom(sockfd, recvline, MAXLINE, 0, NULL, NULL)) < 0){
            if (errno == EINTR)
                fprintf(stderr, "socket timeout\n");
            else
                perror("recvfrom error");
        }else{
```

Example (UDP echo server with SIGALRM timeout): lecture_10_code/udp_echo_server_alarm/client1.c

Refer client1.c to see how to place timeout in the client's recvfrom().

```
            alarm(0); // cancel the pending alarm request
            recvline[n] = 0;    /* null terminate */
            fputs(recvline, stdout); //print out what the client just received
        }
    }
}

static void sig_alrm(int signo)
{
    return; /* just interrupt the recvfrom() */
}
```

recvfrom() with a Timeout Using select()

- select() allows a program to monitor multiple file descriptors, waiting until one or more of the file descriptors become "ready" for some class of I/O operation (e.g., input possible)

```
int select(int nfds, fd_set *readfds, fd_set *writefds,
           fd_set *exceptfds, struct timeval *timeout);
```

  - *nfds* is the highest-numbered file descriptor in any of the three sets, plus 1.
  - *timeout* is an upper bound on the amount of time elapsed before **select**() returns.

```
struct timeval {
    long    tv_sec;         /* seconds */
    long    tv_usec;        /* microseconds */
};
```

  - *readfds*, *writefds* and *exceptfds* are three independent sets of file descriptors

Three Independent Sets of File Descriptors

- The file descriptors listed in *readfds* will be watched to see if characters become available for reading (more precisely, to see if a read will not block).
- The file descriptors in *writefds* will be watched to see if space is available for write (buffer available for write).
- The file descriptors in *exceptfds* will be watched for exceptional conditions (e.g., out-of-band data). After **select**() has returned, *exceptfds* will be cleared of all file descriptors except for those for which an exceptional condition has occurred.

out-of-band data or *expedited data* is the data transferred through a stream that is independent from the main *in-band* data stream.

Descriptor Sets

- Use these macros to manipulate descriptor sets
  - FD_ZERO(fd_set *fd_set)
    - Clear a set
  - FD_SET(int fd, fd_set *fd_set)
    - Add a given file descriptor to a set
  - FD_CLR(int fd, fd_set *fd_set)
    - Remove a given file descriptor from a set
  - FD_ISSET(int fd, fd_set *fd_set)
    - Test to see if a file descriptor is part of the set

Example (UDP echo server with timeout select()): lecture_10_code/udp_echo_server_select/server.c

```
#include <stdio.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <string.h>
#include <unistd.h>
#include <sys/select.h>
#define PORT 8080
#define MAXLINE 4096

int readable_timeout(int fd, int sec){
    fd_set rset;
    struct timeval tv;
    FD_ZERO(&rset); //clear the file descriptor set
    FD_SET(fd, &rset); // add fd to the set
    tv.tv_sec = sec; // set the timeout time in second
    tv.tv_usec = 0; // set the timeout time in microseconds
    return (select(fd + 1, &rset, NULL, NULL, &tv)); /* > 0 if descriptor is readable */
}
```

Example (UDP echo server with timeout select()): lecture_10_code/udp_echo_server_select/server.c

```
int main (int argc, char const *argv[])
{
    int sockfd;
    struct sockaddr_in  servaddr, cliaddr;

    sockfd = socket(AF_INET, SOCK_DGRAM, 0);

    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family      = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port        = htons(PORT);

    bind(sockfd, (struct sockaddr *) &servaddr, sizeof(servaddr));

    printf("The UDP server is on.\n");
    udp_echo(sockfd, (struct sockaddr *) &cliaddr, sizeof(cliaddr));

}
```

Example (UDP echo server with timeout select()): lecture_10_code/udp_echo_server_select/server.c

```
void udp_echo(int sockfd, struct sockaddr *pcliaddr, socklen_t clilen)
{
    int         n;
    socklen_t   len;
    char        msg[MAXLINE + 1];
    for ( ; ; ) {
        len = clilen;
        if (readable_timeout(sockfd, 5) == 0){
      // check whether the sockfd is readable
            fprintf(stderr, "socket timeout\n");
        }else{
            n = recvfrom(sockfd, msg, MAXLINE, 0, pcliaddr, &len);
            //receive message from the client
            sendto(sockfd, msg, n, 0, pcliaddr, len);
            //send message back to the client
            msg[n] = 0;     /* null terminate */
            printf("Received and sent: %s\n", msg); //print out the message
        }
    }
}
```

Example (UDP echo server with timeout select()): lecture_10_code/udp_echo_server_select/client.c

```
#include <stdio.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/select.h>
#define PORT 8080
#define MAXLINE 4096

int readable_timeout(int fd, int sec){
    fd_set rset;
    struct timeval tv;
    FD_ZERO(&rset); // clear the file descriptor set
    FD_SET(fd, &rset); //add fd to the set
    tv.tv_sec = sec; // set the timeout time in second
    tv.tv_usec = 0; //set the timeout time in microsecond
    return (select(fd + 1, &rset, NULL, NULL, &tv)); /* > 0 if descriptor is readable */
}
```

Example (UDP echo server with timeout select()): lecture_10_code/udp_echo_server_select/client.c

```
int main(int argc, char const *argv[]) {
    int sockfd;
    struct sockaddr_in serv_addr;

    memset(&serv_addr, '0', sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);

    // Convert IP addresses from text to binary form
    if(inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr)<=0) {
        printf("\nInvalid address/ Address not supported \n");
        return -1;
    }
    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) { //create UDP socket
        printf("\n Socket creation error \n");
        return -1;
    }
    printf("The UDP client is on.\n");
    echo_client(stdin, sockfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr));
    exit(0);
}
```

Example (UDP echo server with timeout select()): lecture_10_code/udp_echo_server_select/client.c

```
Void echo_client(FILE *fp, int sockfd, const struct sockaddr *pservaddr, socklen_t servlen) {
    int n;
    char    sendline[MAXLINE], recvline[MAXLINE + 1];

    while (fgets(sendline, MAXLINE, fp) != NULL) {

        sendto(sockfd, sendline, strlen(sendline), 0, pservaddr, servlen);

        if (readable_timeout(sockfd, 5) == 0){
            fprintf(stderr, "socket timeout\n");
        }else{
            n = recvfrom(sockfd, recvline, MAXLINE, 0, NULL, NULL);
            recvline[n] = 0;    /* null terminate */
            fputs(recvline, stdout); //print out what the client just received
        }

    }
}
```

Terminal 1:
$ ./server

Terminal 2:
$ ./client

Homework Readings

- UNP: Unix Network Programming
  - Chapter 8, 14.2