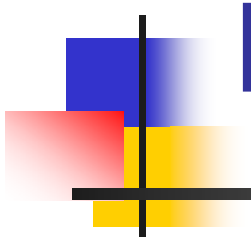


Chapter 16. Network IPC: Sockets

System Programming

http://www.cs.ccu.edu.tw/~pahsiung/courses/sp

熊博安

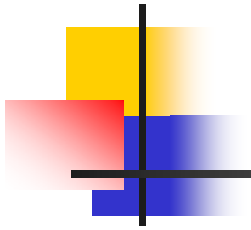
國立中正大學資訊工程學系

pahsiung@cs.ccu.edu.tw

(05)2720411 ext. 33119

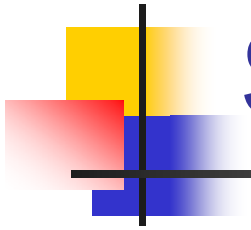
Class: EA-104

Office: EA-512



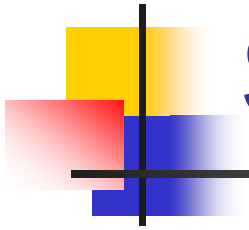
Introduction

- Communication between processes
  - On the same machine, OR
  - On different machines
- Many different network protocols
  - TCP/IP: de facto standard over Internet
- POSIX.1
  - Based on 4.4BSD socket interface



Socket Descriptors

- Socket
  - An abstraction of a communication endpoint
- To access files: file descriptors
- To access sockets: socket descriptors
- File descriptor functions: read(), write() work with socket descriptors, too!



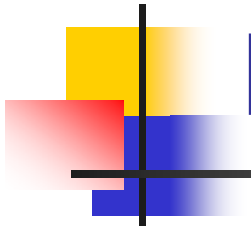
Socket Descriptors

- To create a socket

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

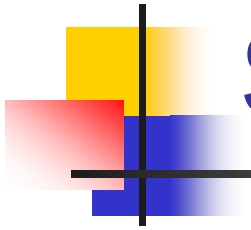
- Returns: file (socket) descriptor if OK, -1 on error

Socket Communication Domains



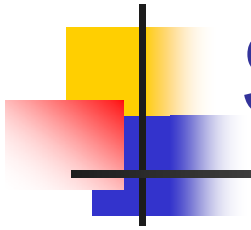
- domain
  - Nature of communication, including address format
  - AF_INET: IPv4 Internet domain
  - AF_INET6: IPv6 Internet domain
  - AF_UNIX: UNIX domain (AF_LOCAL)
  - AF_UNSPEC: unspecified (i.e., any)
  - AF_IPX: Netware protocol (not in POSIX.1)

Address Family



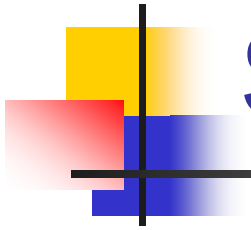
Socket Types

- type
  - Communication characteristics
  - SOCK_DGRAM: fixed-length, connectionless, unreliable messages
  - SOCK_RAW: datagram interface to IP (optional in POSIX.1)
  - SOCK_SEQPACKET: fixed-length, sequenced, reliable, connection-oriented messages
  - SOCK_STREAM: sequenced, reliable, bidirectional, connection-oriented byte streams



Socket Protocol

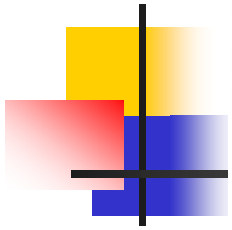
- protocol
  - Usually zero
    - The default protocol for the given domain and socket type
      - SOCK_STREAM socket in the AF_INET domain ➔ TCP (Transmission Control Protocol)
      - SOCK_DGRAM socket in the AF_INET domain ➔ UDP (User Datagram Protocol)
  - When multiple protocols are supported for the same domain and socket type, we can use protocol argument to specify the specific protocol.



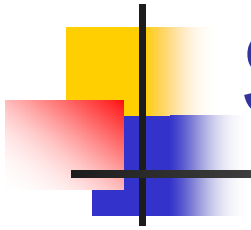
Socket Communication

- Datagram (SOCK_DGRAM)
  - Connectionless, like sending mails
- Connection-oriented
  - Establish connection, like phone call
  - SOCK_STREAM: connection-oriented, byte stream, no message boundary
  - SOCK_SEQPACKET: message-based
- Raw (SOCK_RAW)
  - Applications build their own protocol headers, need superuser privilege

Functions on Socket Descriptors



Function	Behavior with socket
`close` (Section 3.3)	deallocates the socket
`dup`, `dup2` (Section 3.12)	duplicates the file descriptor as normal
`fchdir` (Section 4.22)	fails with `errno` set to `ENOTDIR`
`fchmod` (Section 4.9)	unspecified
`fchown` (Section 4.11)	implementation defined
`fcntl` (Section 3.14)	some commands supported, including `F_DUPFD`, `F_GETFD`, `F_GETFL`, `F_GETOWN`, `F_SETFD`, `F_SETFL`, and `F_SETOWN`
`fdatasync`, `fsync` (Section 3.13)	implementation defined
`fstat` (Section 4.2)	some `stat` structure members supported, but how left up to the implementation
`ftruncate` (Section 4.13)	unspecified
`getmsg`, `getpmsg` (Section 14.4)	works if sockets are implemented with STREAMS (i.e., on Solaris)
`ioctl` (Section 3.15)	some commands work, depending on underlying device driver
`lseek` (Section 3.6)	implementation defined (usually fails with `errno` set to `ESPIPE`)
`mmap` (Section 14.9)	unspecified
`poll` (Section 14.5.2)	works as expected
`putmsg`, `putpmsg` (Section 14.4)	works if sockets are implemented with STREAMS (i.e., on Solaris)
`read` (Section 3.7) and `readv` (Section 14.7)	equivalent to `recv` (Section 16.5) without any flags
`select` (Section 14.5.1)	works as expected
`write` (Section 3.8) and `writev` (Section 14.7)	equivalent to `send` (Section 16.5) without any flags

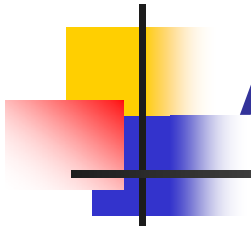


Socket Shutdown

- Communication on a socket is bidirectional
- Disable I/O on socket with shutdown:

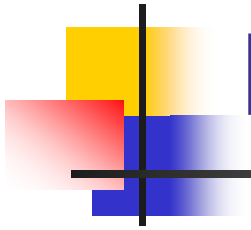
  #include <sys/socket.h>

  int shutdown(int sockfd, int how);
- Returns: 0 if OK, -1 on error
- how:
  - SHUT_RD: disable read
  - SHUT_WR: disable write
  - SHUT_RDWR: disable read/write



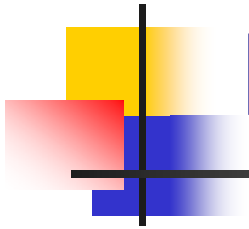
Addressing

- How to identify target process for communication?
  - Machine's network address
    - 140.123.xx.xx
  - Service
    - ftp, telnet, …



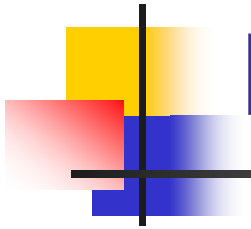
Byte Ordering

- Characteristic of processor architecture
- Arrangement of bytes in larger data types
- Little-endian
  - Lowest byte address = LSB
  - Highest byte address = MSB
- Big-endian
  - Lowest byte address = MSB
  - Highest byte address = LSB



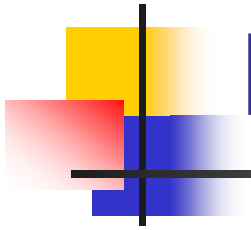
Byte Ordering

- char *cp = 0x4030201
  - MSB = 4
    - Little-endian: cp[3] = 4
    - Big-endian: cp[0] = 4
  - LSB = 1
    - Little-endian: cp[0] = 1
    - Big-endian: cp[3] = 1



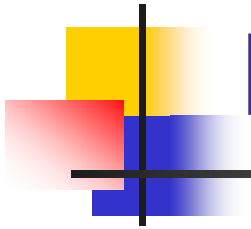
Byte Ordering

- Little-Endian
  - FreeBSD 5.2.1, Intel Pentium
  - Linux 2.4.22, Intel Pentium
- Big Endian
  - Mac OS X 10.3, PowerPC
  - Solaris 9, Sun SPARC
- Some processors can be configured for little-endian or big-endian operation



Byte Ordering

- Host Byte Order
  - Processor Architecture
- Network Byte Order
  - Protocol Address
    - TCP/IP: Big-endian byte order
- Need to translate between host and network byte orders



Byte Ordering Translations

h: host
n: network
l: long
s: short

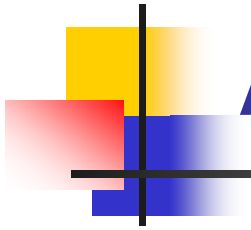
#include <arpa/inet.h>

uint32_t htonl(uint32_t *hostint32*);
  Returns: 32-bit integer in network byte order

uint16_t htons(uint16_t *hostint16*);
  Returns: 16-bit integer in network byte order

uint32_t ntohl(uint32_t *netint32*);
  Returns: 32-bit integer in host byte order

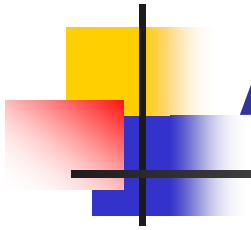
uint16_t ntohs(uint16_t *netint16*);
  Returns: 16-bit integer in host byte order



Address Formats

- Address
  - Socket endpoint in a communication domain
  - Different formats in different communication domain are cast to a generic **sockaddr** address structure

```
struct sockaddr {
  sa_family_t sa_family;/*addr family*/
  char sa_data[]; /* var-len address */
  …
};
```



Address Formats

- On Linux

```
struct sockaddr {
  sa_family_t sa_family; /*addr family */
  char sa_data[14]; /* var-len address */
  …
};
```

- On FreeBSD

```
struct sockaddr {
  unsigned sa_len;   /* total length */
  sa_family_t sa_family; /*addr family */
  char sa_data[14]; /* var-len address */
  …
};
```

Internet Addresses (AF_INET)

- In the IPv4 Internet domain (AF_INET)

```
struct in_addr {
    in_addr_t s_addr; /* IPv4 address */
};
struct sockaddr_in {
    sa_family_t sin_family; /* address family */
    in_port_t sin_port; /* port number */
    struct in_addr sin_addr; /* IPv4 address */
};
```

uint32_t

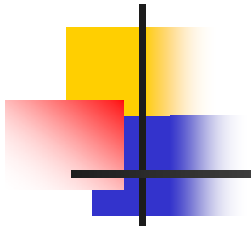
uint16_t

Internet Addresses (AF_INET6)

- In the IPv6 Internet domain (AF_INET6)

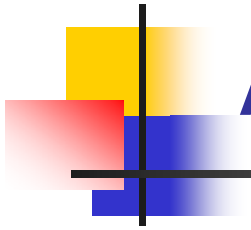
```
struct in6_addr {
    uint8_t s6_addr[16]; /* IPv6 address */
};
struct sockaddr_in6 {
    sa_family_t sin6_family; /* address family */
    in_port_t sin6_port; /* port number */
    uint32_t sin6_flowinfo; /* traffic class, flow info */
    struct in6_addr sin6_addr; /* IPv6 address */
    uint32_t sin6_scope_id; /* interfaces for scope */
};
```

uint16_t



Internet Address (Linux)

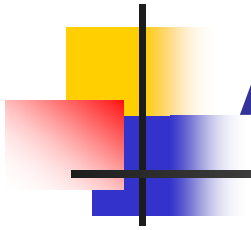
```
struct sockaddr_in {
   sa_family_t sin_family; /* address family */
   in_port_t sin_port; /* port number */
   struct in_addr sin_addr; /* IPv4 address */
   unsigned char sin_zero[8]; /* filler */
};
```



Address Formats

- sockaddr_in
- sockaddr_in6

{ Both cast to **sockaddr** generic structure }



Address Formats

- Binary address format ←→ Dotted-decimal notation (a.b.c.d)

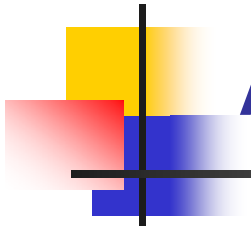
#include <arpa/inet.h>

```
const char *inet_ntop(int domain, const
void *restrict addr, char *restrict str,
socklen_t size);
```

- Returns: pointer to address, NULL on error

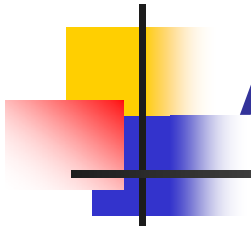
```
int inet_pton(int domain, const char
*restrict str, void *restrict addr);
```

- Returns: 1 on success, 0 if format invalid, or -1 on error



Address Format

- **`inet_ntop`**
  - Binary (network) ➔ text (pointer)
- **`inet_pton`**
  - Text (pointer) ➔ binary (network)
- **`domain`**
  - **`AF_INET`**, **`AF_INET6`**
- **`size`** in inet_ntop
  - **`INET_ADDRSTRLEN`**, **`INET6_ADDRSTRLEN`**
- **`addr`** in inet_pton
  - **`AF_INET`**: 32 bit address
  - **`AF_INET6`**: 128 bit address



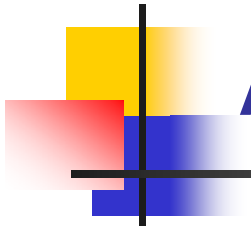
Address Lookup

- To lookup a host
- #include <netdb.h>

```
struct hostent
  *gethostent(void);
```

Returns: pointer if OK, NULL on error

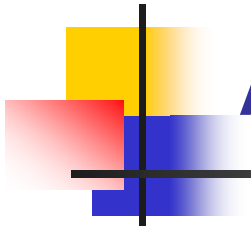
```
void sethostent(int stayopen);
void endhostent(void);
```



Address Lookup

```
struct hostent {
  char *h_name;      /* name of host */
  char **h_aliases; /* alternate hosts */
  int h_addrtype;    /* address type */
  int h_length;      /*add len in bytes */
  char **h_addr_list;/* net addr */
  …
};
```

Addresses are in network byte order.



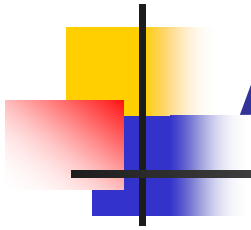
Address Lookup

- gethostbyname, gethostbyaddr
  - Obsolete, replaced by getaddrinfo, getnameinfo

```
#include <sys/socket.h>
#include <netdb.h>
int getaddrinfo( const char *restrict host,
    const char *restrict service,
    const struct addrinfo *restrict hint,
    struct addrinfo **restrict res );
```

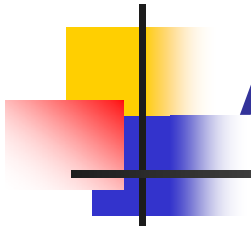
- Returns: 0 if OK, nonzero error code on error

```
void freeaddrinfo(struct addrinfo *ai);
```



Address Lookup

```
struct addrinfo {
    int ai_flags;                   /* custom */
    int ai_family;                  /* address family */
    int ai_socktype;                /* socket type */
    int ai_protocol;                /* protocol */
    socklen_t ai_addrlen;           /* addr len in bytes */
    struct sockaddr *ai_addr;       /* address */
    char *ai_canonname;             /* canonical name */
    struct addrinfo *ai_next;       /* next in list */
    …
};
```

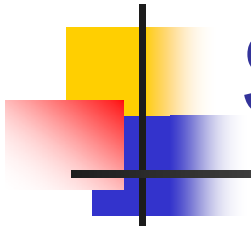


Address Lookup

- Check Figure 16.18 for an example on how to use getaddrinfo()
  - Pages 557 ~ 560
- Results

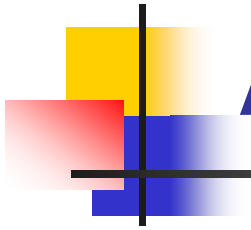
```
./a.out harry nfs
flags canon family inet type stream
  protocol TCP
host harry address 192.168.1.105
  port 2049
```

Associating Addresses with Sockets



- To associate an address with a socket

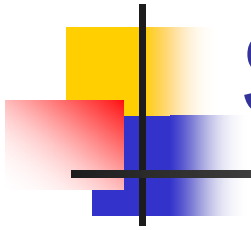
```
#include <sys/socket.h>
int bind(int sockfd,
  const struct sockaddr *addr,
  socklen_t len);
Returns: 0 if OK, -1 on error
```



Address Restrictions

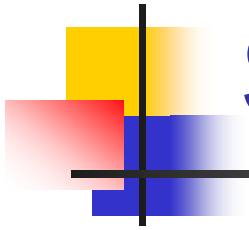
- Address must be valid on the machine (cannot belong to another machine)
- Address must match format supported by address family used to create socket
- Port number cannot be less than 1,024, unless process have privilege (root)
- Usually, only one socket can be bound to one address (some protocols allow duplicate bindings)

Associating Addresses with Sockets



- In AF_INET
  - **`INADDR_ANY`** will bind the socket to all system's network interfaces (cards)
  - Used when we start using the socket (connect, listen), without binding an address to socket,
    - system will choose an address and bind it to our socket

Associating Addresses with Sockets

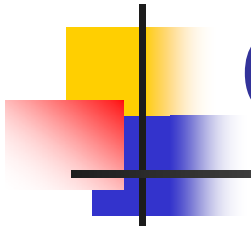


- To get address bound to a socket

```
#include <sys/socket.h>
int getsockname(int sockfd,
  struct sockaddr *restrict
  addr,
  socklen_t *restrict alenp);
```

- Returns: 0 if OK, -1 on error

size of sockaddr buffer changed on return

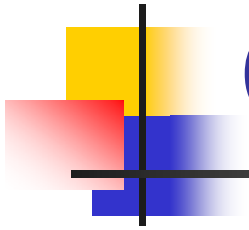


Getting peer's address

- If socket is connected to a peer, we can get the peer's address using:

```
#include <sys/socket.h>
int getpeername(int sockfd,
  struct sockaddr *restrict
  addr,
  socklen_t *restrict alenp);
```

- Returns: 0 if OK, -1 on error

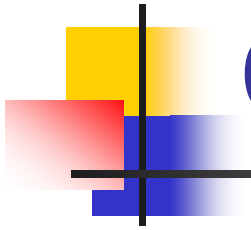


Connection Establishment

- For connection-oriented network service (SOCK_STREAM, SOCK_SEQPACKET)

```
#include <sys/socket.h>
int connect(int sockfd, const
  struct sockaddr *addr,
  socklen_t len);
```

- Returns: 0 if OK, -1 on error



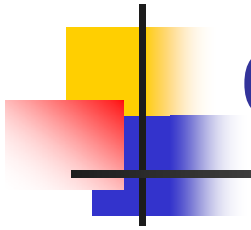
Connection Establishment

- Connect will fail because
  - Machine is down
  - Server not bound to the address we are connecting
  - Server's pending queue full
- Retry connection establishment

Figure 16.9: Retry Connect

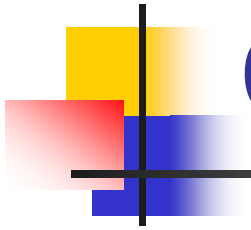
```
#include "apue.h"
#include <sys/socket.h>
#define MAXSLEEP 128

int connect_retry(int sockfd,
        const struct sockaddr *addr, socklen_t alen) {
   int nsec;

   /* Try to connect with exponential backoff.  */
   for (nsec = 1; nsec <= MAXSLEEP; nsec <<= 1) {
      if (connect(sockfd, addr, alen) == 0)
            return(0); /* Connection accepted. */
      /*  Delay before trying again. */
      if (nsec <= MAXSLEEP/2) sleep(nsec);
   }
   return(-1);
}
```

Exponential backoff algorithm



connect with SOCK_DGRAM

- Connect can be used with SOCK_DGRAM as an OPTIMIZATION
- Destination address of all messages is set to the address in connect()
- No need to provide address every time we transmit a message
- Receive datagrams only from that address



Connection Establishment

- Server can announce that it is willing to accept connect requests

```
#include <sys/socket.h>
int listen(int sockfd, int
  backlog);
```

- Returns: 0 if OK, -1 on error
- backlog: #outstanding connect requests

Connection Establishment

- To retrieve a connect request

```
#include <sys/socket.h>
int accept(int sockfd, struct
  sockaddr *restrict addr, socklen_t
  *restrict len);
```

Client's address

- Returns: file descriptor if OK, -1 on error
- Will block if no connect requests
- Sockfd in nonblocking mode ➔ accept will return -1, errno:= EAGAIN or EWOULDBLOCK

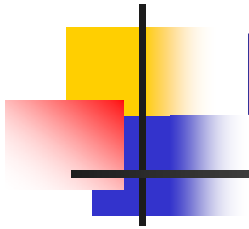
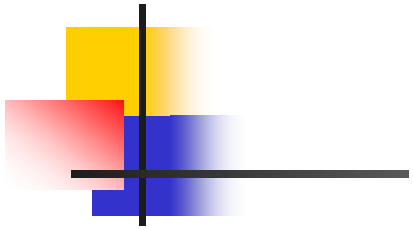


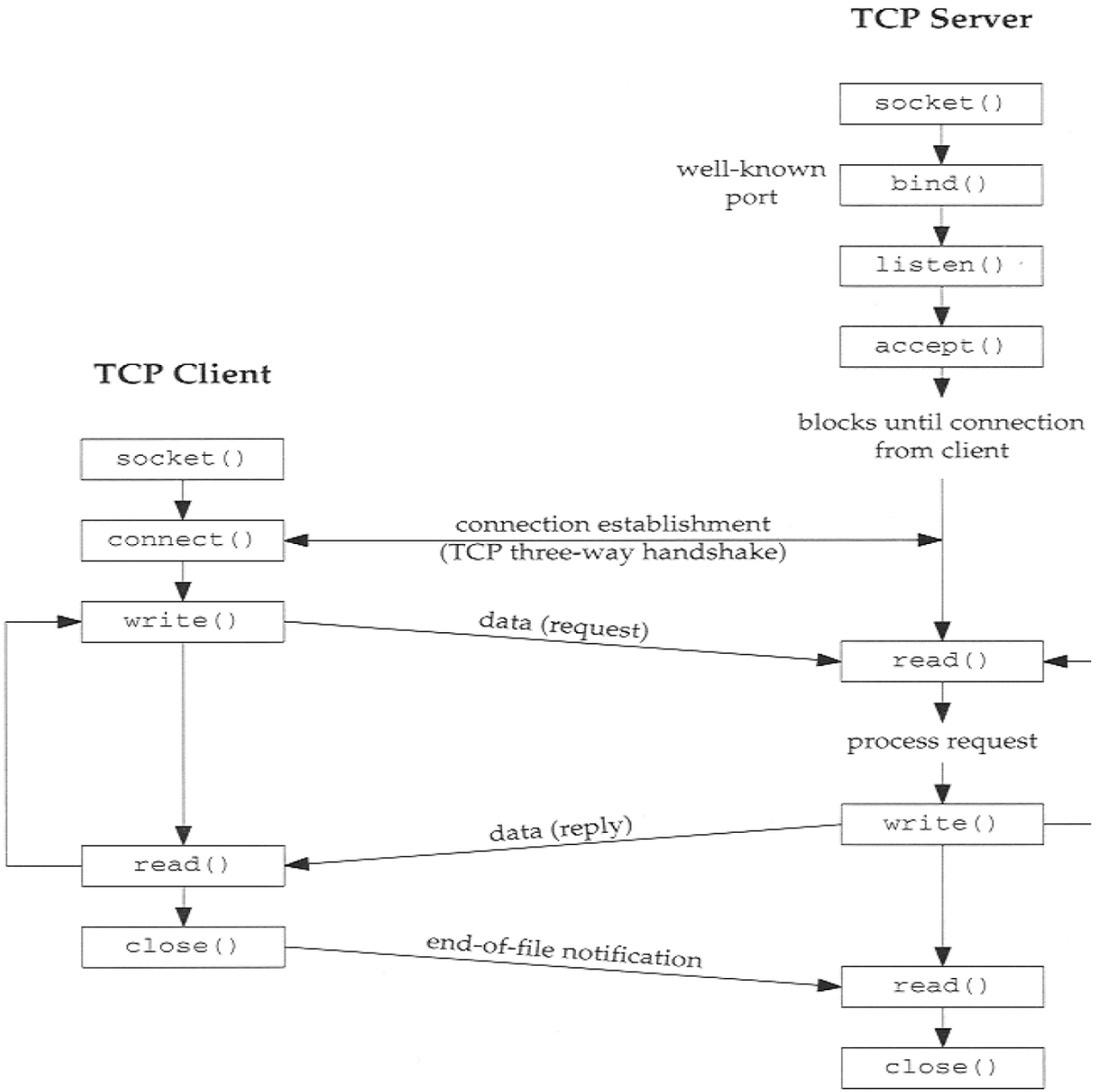
Figure 16.10: Socket Server

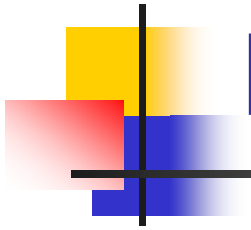
```
#include "apue.h"
#include <errno.h>
#include <sys/socket.h>

int initserver(int type, const struct sockaddr *addr,
   socklen_t alen, int qlen)
{
   int fd;
   int err = 0;

   if ((fd = socket(addr->sa_family, type, 0)) < 0)
       return(-1);
   if (bind(fd, addr, alen) < 0) {
       err = errno;
       goto errout;
   }
```

Figure 16.10: Socket Server

```
    if (type == SOCK_STREAM || type ==
    SOCK_SEQPACKET) {
        if (listen(fd, qlen) < 0) {
            err = errno;
            goto errout;
        }
    }
    return(fd);

errout:
    close(fd);
    errno = err;
    return(-1);
}
```

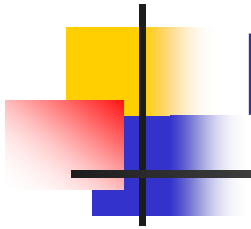



TCP Server
socket()
well-known port
bind()
listen()
accept()
blocks until connection from client
TCP Client
socket()
connect()
connection establishment (TCP three-way handshake)
write()
data (request)
read()
process request
write()
data (reply)
read()
close()
end-of-file notification
read()
close()



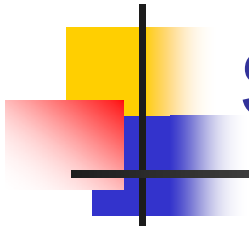
Data Transfer

- Socket is same as file descriptor
  - read()
  - write()
- However, we need to set some options while sending or receiving data
  - Nonblocking, out-of-band data, packet routing, etc.



Data Transfer

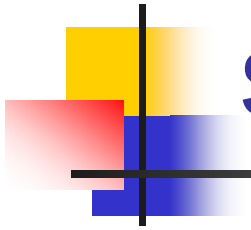
- 3 functions to send data
  - send()
  - sendto()
  - sendmsg()
- 3 functions to receive data
  - recv()
  - recvfrom()
  - recvmsg()



send()

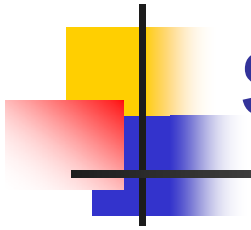
```
#include <sys/socket.h>
ssize_t send(int sockfd, const
  void *buf, size_t nbytes, int
  flags);
```

- Returns: #bytes sent if OK, -1 on error
- After send() returns
  - data has been delivered to network drivers without error (not peer has received!)



send() flags

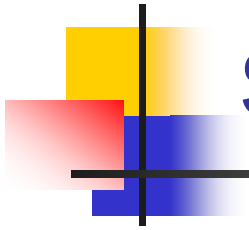
Flag	Description	POSIX.1	FreeBSD 5.2.1	Linux 2.4.22	Mac OS X 10.3	Solaris 9
MSG_DONTROUTE	Don't route packet outside of local network.		•	•	•	•
MSG_DONTWAIT	Enable nonblocking operation (equivalent to using O_NONBLOCK).		•	•	•	
MSG_EOR	This is the end of record if supported by protocol.	•	•	•	•	
MSG_OOB	Send out-of-band data if supported by protocol (see Section 16.7).	•	•	•	•	•



sendto()

```
#include <sys/socket.h>
ssize_t sendto(int sockfd, const
  void *buf, size_t nbytes, int
  flags, const struct sockaddr
  *destaddr, socklen_t destlen);
```

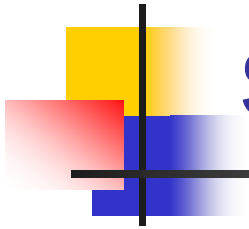
- Returns: #bytes sent if OK, -1 on error
- For connectionless socket, without first calling connect()



sendmsg()

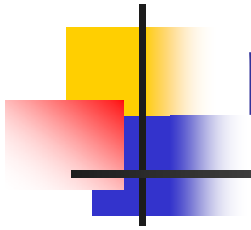
```
#include <sys/socket.h>
ssize_t sendmsg(int sockfd,
  const struct msghdr *msg, int
  flags);
```

- Returns: #bytes sent if OK, -1 on error
- Multiple buffers to send
- Similar to writev() (Section 14.7)



sendmsg()

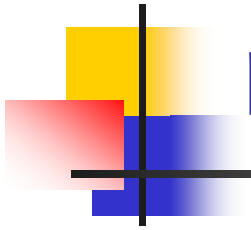
```
struct msghdr {
  void *msg_name; /* opt addr */
  socklen_t msg_namelen; /* #bytes in addr */
  struct iovec *msg_iov; /*array of I/O bufs */
  int msg_iovlen; /* #elements in array */
  void *msg_control; /* ancillary data */
  socklen_t msg_controllen; /* #bytes ancill */
  int msg_flags; /* flags for recd msg */
  …
};
```



recv()

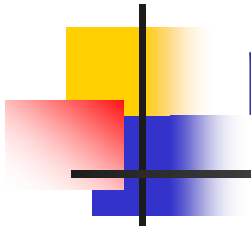
```
#include <sys/socket.h>
ssize_t recv(int sockfd, void
  *buf, size_t nbytes, int
  flags);
```

- Returns: #bytes in msg, 0 if no msg and peer shutdown, or -1 on error



recv() flags

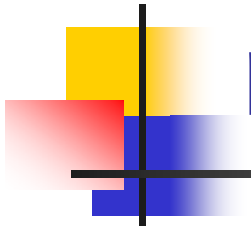
Flag	Description	POSIX.1	FreeBSD 5.2.1	Linux 2.4.22	Mac OS X 10.3	Solaris 9
MSG_OOB	Retrieve out-of-band data if supported by protocol (see Section 16.7).	•	•	•	•	•
MSG_PEEK	Return packet contents without consuming packet.	•	•	•	•	•
MSG_TRUNC	Request that the real length of the packet be returned, even if it was truncated.			•		
MSG_WAITALL	Wait until all data is available (SOCK_STREAM only).	•	•	•	•	•



recvfrom()

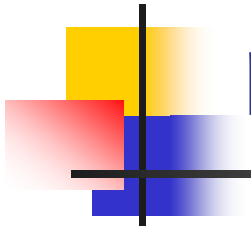
```
#include <sys/socket.h>
ssize_t recvfrom(int sockfd,
  void *restrict buf,
  size_t len, int flags,
  struct sockaddr *restrict addr,
  socklen_t *restrict addrlen);
```

- Returns: #bytes in msg, 0 if no msg and peer shutdown, or -1 on error
- Used by connectionless socket



recvmsg()

- #include <sys/socket.h>
- ssize_t recvmsg(int sockfd, struct msghdr *msg, int flags);
- Returns: #bytes in msg, 0 if no msg and peer shutdown, or -1 on error



recvmsg() flags

Flag	Description	POSIX.1	FreeBSD 5.2.1	Linux 2.4.22	Mac OS X 10.3	Solaris 9
MSG_CTRUNC	Control data was truncated.	•	•	•	•	•
MSG_DONTWAIT	recvmsg was called in nonblocking mode.			•		•
MSG_EOR	End of record was received.	•	•	•	•	•
MSG_OOB	Out-of-band data was received.	•	•	•	•	•
MSG_TRUNC	Normal data was truncated.	•	•	•	•	•

Figure 16.14: Remote Uptime

```
#include "apue.h"
#include <netdb.h>
#include <errno.h>
#include <sys/socket.h>

#define MAXADDRLEN    256
#define BUFLEN        128
extern int connect_retry(int, const struct sockaddr *,
   socklen_t);

void print_uptime(int sockfd)
{
   int  n;
   char buf[BUFLEN];

   while ((n = recv(sockfd, buf, BUFLEN, 0)) > 0)
        write(STDOUT_FILENO, buf, n);
   if (n < 0)  err_sys("recv error");
}
```

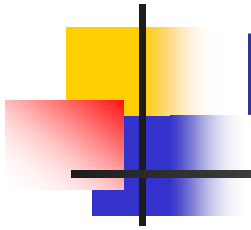


Figure 16.14: Remote Uptime

```
int main(int argc, char *argv[])
{
    struct addrinfo         *ailist, *aip;
    struct addrinfo         hint;
    int                     sockfd, err;
    if (argc != 2) err_quit("usage: ruptime hostname");
    hint.ai_flags = 0;
    hint.ai_family = 0;
    hint.ai_socktype = SOCK_STREAM;
    hint.ai_protocol = 0;
    hint.ai_addrlen = 0;
    hint.ai_canonname = NULL;
    hint.ai_addr = NULL;
    hint.ai_next = NULL;
    if ((err = getaddrinfo(argv[1], "ruptime", &hint, &ailist)) != 0)
```

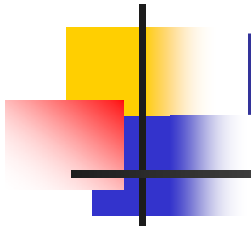


Figure 16.14: Remote Uptime

```
        err_quit("getaddrinfo error: %s", gai_strerror(err));
    for (aip = ailist; aip != NULL; aip = aip->ai_next) {
        if ((sockfd = socket(aip->ai_family, SOCK_STREAM, 0)) < 0)
            err = errno;
        if (connect_retry(sockfd, aip->ai_addr, aip->ai_addrlen) < 0) {
            err = errno;
        } else {
            print_uptime(sockfd);
            exit(0);
        }
    }
    fprintf(stderr, "can't connect to %s: %s\n", argv[1],
      strerror(err));
    exit(1);
}
```

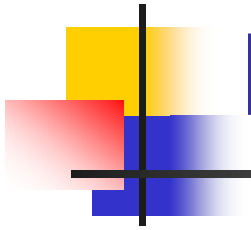


Figure 16.14: Remote Uptime

- Connects to a server
- Reads string from server
- Prints string on **`stdout`**
- **`SOCK_STREAM`** ➔ one recv() might not get the full string ➔ repeat until 0
- If server supports multiple network interfaces or protocols, try each in turn using **`connect_retry()`** (Fig. 16.9)

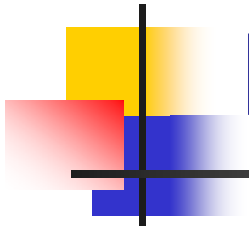


Figure 16.15: Uptime Server

```
#include "apue.h"
#include <netdb.h>
#include <errno.h>
#include <syslog.h>
#include <sys/socket.h>

#define BUFLEN 128
#define QLEN 10
#ifndef HOST_NAME_MAX
#define HOST_NAME_MAX 256
#endif
extern int initserver(int, struct sockaddr *, socklen_t, int);

void serve(int sockfd)
{
   int  clfd;
   FILE *fp;
   char buf[BUFLEN];
```

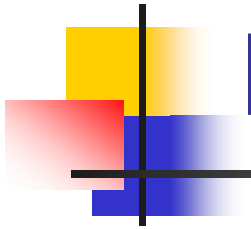


Figure 16.15: Uptime Server

```
        for (;;) {
                clfd = accept(sockfd, NULL, NULL);
                if (clfd < 0) {
                        syslog(LOG_ERR, "ruptimed: accept error: %s",
                          strerror(errno));
                        exit(1);
                }
                if ((fp = popen("/usr/bin/uptime", "r")) == NULL) {
                        sprintf(buf, "error: %s\n", strerror(errno));
                        send(clfd, buf, strlen(buf), 0);
                } else {
                        while (fgets(buf, BUFLEN, fp) != NULL)
                                send(clfd, buf, strlen(buf), 0);
                        pclose(fp);
                }
                close(clfd);
        }
}
```

Figure 16.15: Uptime Server

```
int main(int argc, char *argv[])
{
   struct addrinfo     *ailist, *aip;
   struct addrinfo     hint;
   int                 sockfd, err, n;
   char                *host;

   if (argc != 1)  err_quit("usage: ruptimed");
#ifdef _SC_HOST_NAME_MAX
   n = sysconf(_SC_HOST_NAME_MAX);
   if (n < 0)  /* best guess */
#endif
   n = HOST_NAME_MAX;
   host = malloc(n);
   if (host == NULL)   err_sys("malloc error");
   if (gethostname(host, n) < 0)     err_sys("gethostname
   error");
   daemonize("ruptimed");
```

Fig. 13.1

Figure 16.15: Uptime Server

```
        hint.ai_flags = AI_CANONNAME;
        hint.ai_family = 0;
        hint.ai_socktype = SOCK_STREAM;
        hint.ai_protocol = 0;
        hint.ai_addrlen = 0;
        hint.ai_canonname = NULL;
        hint.ai_addr = NULL;
        hint.ai_next = NULL;
        if ((err = getaddrinfo(host, "ruptime", &hint, &ailist)) != 0) {
             syslog(LOG_ERR, "ruptimed: getaddrinfo error: %s",
               gai_strerror(err));
             exit(1);
        }
        for (aip = ailist; aip != NULL; aip = aip->ai_next) {
             if ((sockfd = initserver(SOCK_STREAM, aip->ai_addr,
               aip->ai_addrlen, QLEN)) >= 0) {
                     serve(sockfd);
                     exit(0);
             }
        }
        exit(1);
}
```

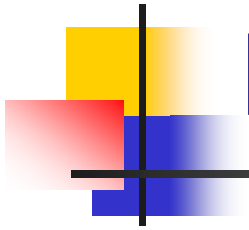


Figure 16.15: Uptime Server

- Get host name
- Lookup host address for uptime
- Initialize server
  - initserver → (socket, bind, listen)
- Wait for requests to arrive
  - (server → accept)

Figure 16.16: Server without pipe to client

```
#include "apue.h"
#include <netdb.h>
#include <errno.h>
#include <syslog.h>
#include <fcntl.h>
#include <sys/socket.h>
#include <sys/wait.h>
#define QLEN 10

#ifndef HOST_NAME_MAX
#define HOST_NAME_MAX 256
#endif

extern int initserver(int, struct sockaddr *, socklen_t, int);

void serve(int sockfd)
{
    int         clfd, status;
    pid_t       pid;
```

Figure 16.16: Server without pipe to client

```
for (;;) {
    clfd = accept(sockfd, NULL, NULL);
    if (clfd < 0) {
            syslog(LOG_ERR, "ruptimed: accept error:
%s",
              strerror(errno));
            exit(1);
    }
    if ((pid = fork()) < 0) {
          syslog(LOG_ERR, "ruptimed: fork error: %s",
              strerror(errno));
            exit(1);
```

Figure 16.16: Server without pipe to client

```
                        } else if (pid == 0) { /* child */
    /* The parent called daemonize ({Prog daemoninit}), so
    STDIN_FILENO, STDOUT_FILENO, and STDERR_FILENO are already
    open to /dev/null.  Thus, the call to close doesn't need to
    be protected by checks that clfd isn't already equal to one
    of these values. */
                if (dup2(clfd, STDOUT_FILENO) != STDOUT_FILENO
    ||              dup2(clfd, STDERR_FILENO) != STDERR_FILENO) {
                    syslog(LOG_ERR, "ruptimed: unexpected error");
                          exit(1);
                }
                close(clfd);
                execl("/usr/bin/uptime", "uptime", (char *)0);
                syslog(LOG_ERR, "ruptimed: unexpected return
    from exec: %s",
                    strerror(errno));
            } else {                /* parent */
                close(clfd);
                waitpid(pid, &status, 0);     }
    }
}
```

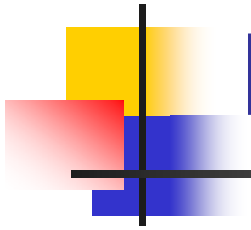
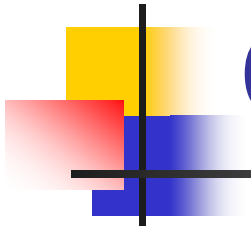


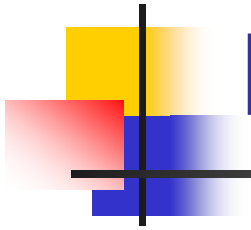
Figure 16.16: Server

- Instead of **popen**, this server forks a child, uses **dup2** to change child's:
  - STDIN_FILENO = **/dev/null**
  - STDOUT_FILENO = client socket
  - STDERR_FILENO = client socket
- Child **exec**'s uptime and results are sent to **ruptime** client directly (no pipe!)
- Parent closes client socket, waits for child to finish (uptime is very quick, no delay problem)



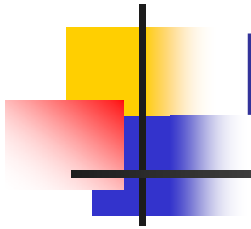
Connection-oriented vs. Connectionless Sockets

- When to use connection-oriented socket?
- When to use connectionless socket?
- Depends on
  - how much work we want to do?
  - what kind of tolerance we have for errors?
- Connectionless socket
  - packets arrive out of order
  - packets can be lost or duplicate



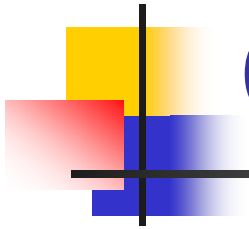
Packets arrive out of order

- Maximum packet size is a characteristic of communication protocol
- Data cannot fit in one packet
  - Packets need to be numbered
  - Worry about packet ordering in application



Packet loss

- Tolerating packet loss
  - Reliable communication
    - Request retransmission for missing packets
    - Identify duplicate packets and discard one
      - Packet delayed, requested retransmission, delayed packet and resent packet both arrive (duplicate!)
  - Unreliable communication
    - Let user retry the command



Connection-oriented vs. Connectionless Sockets

- Simple Applications
  - Connectionless sockets
    - No connection setup, can retry (delay small)
- Complex Applications
  - Connection-oriented sockets
    - Connection setup, reliable! (cannot afford delay)

Figure 16.17: Connectionless Client

```
#include "apue.h"
#include <netdb.h>
#include <errno.h>
#include <sys/socket.h>
#define BUFLEN          128
#define TIMEOUT                 20

void sigalrm(int signo){}

void print_uptime(int sockfd, struct addrinfo *aip) {
   int  n;
   char buf[BUFLEN];

   buf[0] = 0;
  if (sendto(sockfd, buf, 1, 0, aip->ai_addr, aip->ai_addrlen)
   < 0)
        err_sys("sendto error");
   alarm(TIMEOUT);
```

Figure 16.17: Connectionless Client

```
    if ((n = recvfrom(sockfd, buf, BUFLEN, 0, NULL, NULL)) < 0)
    {
         if (errno != EINTR) alarm(0);
         err_sys("recv error");
    }
    alarm(0);
    write(STDOUT_FILENO, buf, n);
}
```

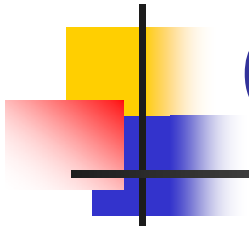
Figure 16.17: Connectionless Client

```
int main(int argc, char *argv[]) {
   struct addrinfo            *ailist, *aip;
   struct addrinfo            hint;
   int                        sockfd, err;
   struct sigaction           sa;

   if (argc != 2) err_quit("usage: ruptime hostname");
   sa.sa_handler = sigalrm;
   sa.sa_flags = 0;
   sigemptyset(&sa.sa_mask);
   if (sigaction(SIGALRM, &sa, NULL) < 0) err_sys("sigaction
   error");
   hint.ai_flags = 0;
   hint.ai_family = 0;
   hint.ai_socktype = SOCK_DGRAM;
   hint.ai_protocol = 0;
   hint.ai_addrlen = 0;
```

Figure 16.17: Connectionless Client

```
    hint.ai_canonname = NULL;
    hint.ai_addr = NULL;
    hint.ai_next = NULL;
    if ((err = getaddrinfo(argv[1], "ruptime", &hint,
    &ailist)) != 0)
         err_quit("getaddrinfo error: %s", gai_strerror(err));

    for (aip = ailist; aip != NULL; aip = aip->ai_next) {
         if ((sockfd=socket(aip->ai_family, SOCK_DGRAM, 0))<0){
                 err = errno;
         } else {
                 print_uptime(sockfd, aip);
                 exit(0); }
    }

    fprintf(stderr, "can't contact %s: %s\n", argv[1],
    strerror(err));
    exit(1);
}
```

Figure 16.17: Connectionless Client



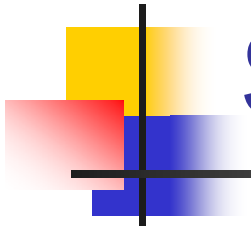
- main: installing handler for **`SIGALRM`**
  - to avoid indefinite blocking in **`recvfrom`**()
- Connectionless ➔ need to notify server we need its server
  - Send a single byte first!
  - Server receives the byte
  - Server gets our address
  - Server uses this address for responding

Figure 16.18: Connectionless Server

```
#include "apue.h"
#include <netdb.h>
#include <errno.h>
#include <syslog.h>
#include <sys/socket.h>
#define BUFLEN          128
#define MAXADDRLEN      256

#ifndef HOST_NAME_MAX
#define HOST_NAME_MAX 256
#endif
extern int initserver(int, struct sockaddr *, socklen_t, int);

void serve(int sockfd) {
   int          n;
   socklen_t    alen;
   FILE         *fp;
   char         buf[BUFLEN];
   char         abuf[MAXADDRLEN];
```

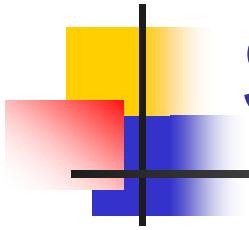
Figure 16.18: Connectionless Server

```
    for (;;) {
        alen = MAXADDRLEN;
        if ((n = recvfrom(sockfd, buf, BUFLEN, 0,
          (struct sockaddr *)abuf, &alen)) < 0) {
                syslog(LOG_ERR, "ruptimed: recvfrom error: %s",
                  strerror(errno));
                exit(1);        }
        if ((fp = popen("/usr/bin/uptime", "r")) == NULL) {
                sprintf(buf, "error: %s\n", strerror(errno));
                sendto(sockfd, buf, strlen(buf), 0,
                  (struct sockaddr *)abuf, alen);
        } else {
                if (fgets(buf, BUFLEN, fp) != NULL)
                        sendto(sockfd, buf, strlen(buf), 0,
                          (struct sockaddr *)abuf, alen);
                pclose(fp);     }
    }
}
```



Socket Options

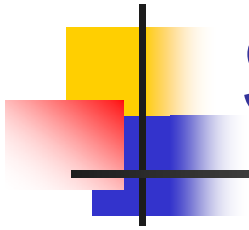
- Three kinds of socket options
  - Generic options: all socket types
  - Socket level options: depend on protocol support
  - Protocol level options
- Single UNIX Specification
  - Socket-layer options (first two above)



Socket Options

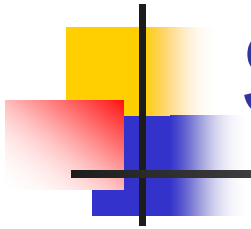
```
#include <sys/socket.h>
int setsockopt(int sockfd, int
  level, int option, const void
  *val, socklen_t len);
```

- Returns: 0 if OK, -1 on error



Socket Options

- ***level*** : protocol number
  - Generic option: SOL_SOCKET
  - TCP option: IPPROTO_TCP
  - IP option: IPPROTO_IP
- ***val*** : pointer or integer
  - nonzero integer: option is enabled
  - zero integer: option is disabled
- ***len*** : size of object pointed to by ***val***



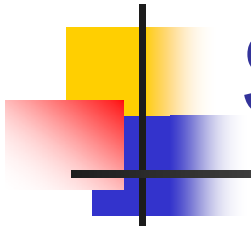
Socket Options

```
#include <sys/socket.h>
int getsockopt(int sockfd, int
  level, int option, void *restrict
  val, socklen_t *restrict lenp);
```

- Returns: 0 if OK, -1 on error
- ***lenp*** : size of buffer for option copy
- data size > buffer size
  - silent truncation
- data size ≤ buffer size
  - ***lenp*** := data size

Figure 16.19: Socket Options

Option	Type of *val* argument	Description
SO_ACCEPTCONN	int	Return whether a socket is enabled for listening (getsockopt only).
SO_BROADCAST	int	Broadcast datagrams if *val is nonzero.
SO_DEBUG	int	Debugging in network drivers enabled if *val is nonzero.
SO_DONTROUTE	int	Bypass normal routing if *val is nonzero.
SO_ERROR	int	Return and clear pending socket error (getsockopt only).
SO_KEEPALIVE	int	Periodic keep-alive messages enabled if *val is nonzero.
SO_LINGER	struct linger	Delay time when unsent messages exist and socket is closed.
SO_OOBINLINE	int	Out-of-band data placed inline with normal data if *val is nonzero.
SO_RCVBUF	int	The size in bytes of the receive buffer.
SO_RCVLOWAT	int	The minimum amount of data in bytes to return on a receive call.
SO_RCVTIMEO	struct timeval	The timeout value for a socket receive call.
SO_REUSEADDR	int	Reuse addresses in bind if *val is nonzero.
SO_SNDBUF	int	The size in bytes of the send buffer.
SO_SNDLOWAT	int	The minimum amount of data in bytes to transmit in a send call.
SO_SNDTIMEO	struct timeval	The timeout value for a socket send call.
SO_TYPE	int	Identify the socket type (getsockopt only).



Socket Address Reuse

- **SO_REUSEADDR**: reuse socket address
- When a server terminates, cannot restart it immediately
  - TCP implementation prevents us from binding the same address until a timeout expires (several minutes!)
  - Use **SO_REUSEADDR** to bypass restriction

Figure 16.20: Initialize socket

```
#include "apue.h"
#include <errno.h>
#include <sys/socket.h>

int initserver(int type, const struct sockaddr *addr, socklen_t alen,
  int qlen)
{
    int fd, err;
    int reuse = 1;

    if ((fd = socket(addr->sa_family, type, 0)) < 0)
        return(-1);
    if (setsockopt(fd, SOL_SOCKET, SO_REUSEADDR, &reuse,
      sizeof(int)) < 0) {
        err = errno;
        goto errout;
    }
```

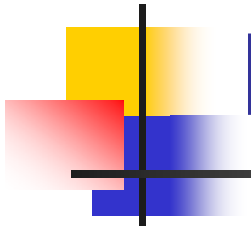
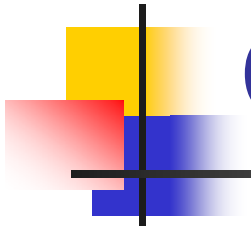


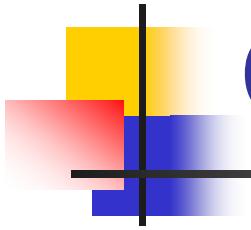
Figure 16.20: Initialize socket

```
    if (bind(fd, addr, alen) < 0) {
        err = errno;
        goto errout;
    }
    if (type == SOCK_STREAM || type == SOCK_SEQPACKET) {
        if (listen(fd, qlen) < 0) {
            err = errno;
            goto errout;
        }
    }
    return(fd);

errout:
    close(fd);
    errno = err;
    return(-1);
}
```



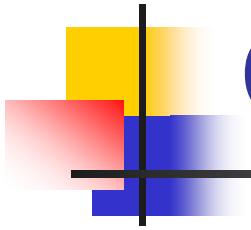
Out-of-Band Data

- Optional feature of communication protocol
  - Allows higher-priority delivery of data
    - Out-of-band data is sent ahead of other data queued for transmission
  - TCP: “urgent” data (single byte!)
    - Specify **`MSG_OOB`** flag in any **`send`** function
    - More than one byte: last byte is urgent



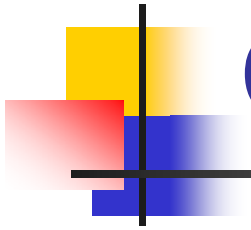
Out-of-Band Data

- When urgent data received, SIGURG signal is received
  - Need to set ownership of socket
    - fcntl(sockfd, F_SETOWN, pid);
  - Get ownership of socket
    - owner = fcntl(sockfd, F_GETOWN, 0);



Out-of-Band Data

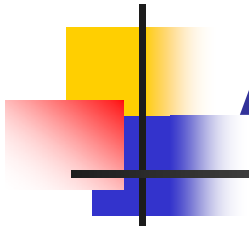
- TCP supports “urgent mark”
  - point in normal data stream where urgent data would go
- To receive urgent data inline with normal data
  - use SO_OOBINLINE socket option
- To identify urgent mark
  - sockatmark()



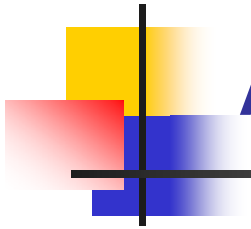
Out-of-Band Data

- #include <sys/socket.h>
- int sockatmark(int sockfd);
- Returns: 1 if at mark 0, -1 on error
- Next byte is urgent mark ➔ sockatmark returns 1
- TCP queues only one urgent data
  - Current urgent byte not received ➔ discard next urgent byte

Nonblocking and Asynchronous I/O



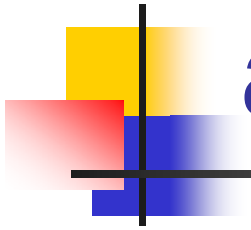
- Blocking I/O
  - **`send`**, **`recv`** block when no queue space or no data
- Nonblocking I/O
  - **`send`**, **`recv`** fail, **`errno`** := **`EWOULDBLOCK`** or **`EAGAIN`**
  - use **`poll`** or **`select`** to determine if we can receive or transmit data



Nonblocking and Asynchronous I/O

- Real-time extensions of Single UNIX Specification
  - Generic asynchronous I/O
- Socket-based asynchronous I/O (signal-based I/O)
  - uses **`SIGIO`**

Socket-based (signal-based) asynchronous I/O



- Two steps to enable asynchronous I/O
  - Establish socket ownership
  - Arrange signal
- Establish socket ownership
  - fcntl: **`F_SETOWN`**
  - ioctl: **`FIOSETOWN`**, **`SIOCSPGRP`**
- Arrange signal
  - fcntl: **`F_SETFL`** command with **`O_ASYNC`** flag
  - ioctl: **`FIOASYNC`** command

Socket Asynchronous I/O

Mechanism	POSIX.1	FreeBSD 5.2.1	Linux 2.4.22	Mac OS X 10.3	Solaris 9
`fcntl(fd, F_SETOWN, pid)`	•	•	•	•	•
`ioctl(fd, FIOSETOWN, pid)`		•	†	•	•
`ioctl(fd, SIOCSPGRP, pid)`		•	†	•	•
`fcntl(fd, F_SETFL, flags|O_ASYNC)`		•	•	•	
`ioctl(fd, FIOASYNC, &n);`		•	•	•	•