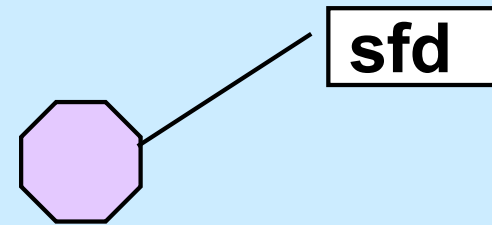# CS 33

Network Programming (2)

Streams in the Inet Domain (1)

- **Server steps**
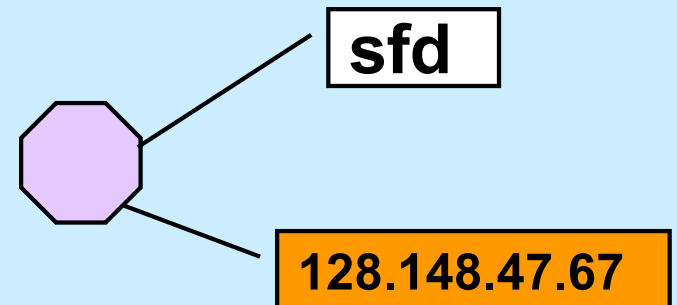 **1) create socket**

```
sfd = socket(AF_INET, SOCK_STREAM, 0);
```

sfd

Streams in the Inet Domain (2)

- **Server steps**

  2) bind name to socket

  ```
  bind(sfd,
      (struct sockaddr *)&my_addr, sizeof(my_addr));
  ```
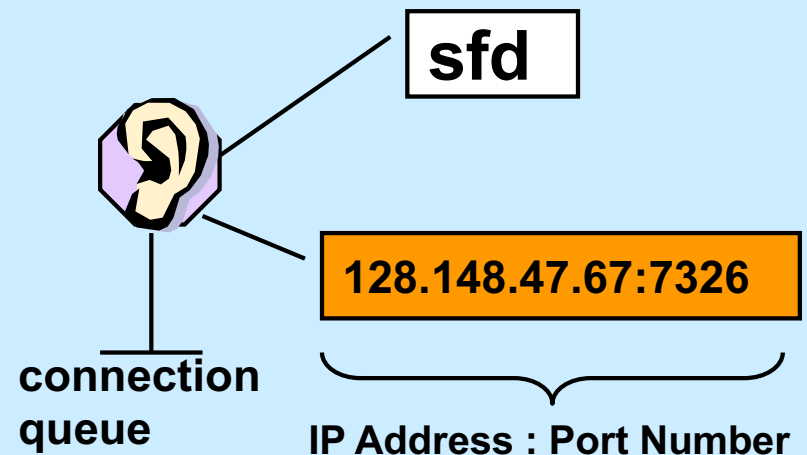

sfd

128.148.47.67

Streams in the Inet Domain (3)

- ## Server steps

  ### 3) put socket in "listening mode"

  ```
  int listen(int sfd, int MaxQueueLength);
  ```

  sfd

  128.148.47.67:7326

  connection queue

  IP Address : Port Number

Streams in the Inet Domain (4)

- ## Client steps

    ### 1) create socket

    ```
    cfd = socket(AF_INET, SOCK_STREAM, 0);
    ```
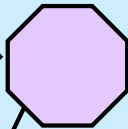
cfd
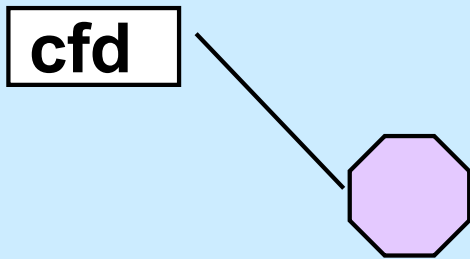
Streams in the Inet Domain (5)

- **Client steps**

    2) bind name to socket

    ```
    bind(cfd,
        (struct sockaddr *)&my_addr, sizeof(my_addr));
    ```
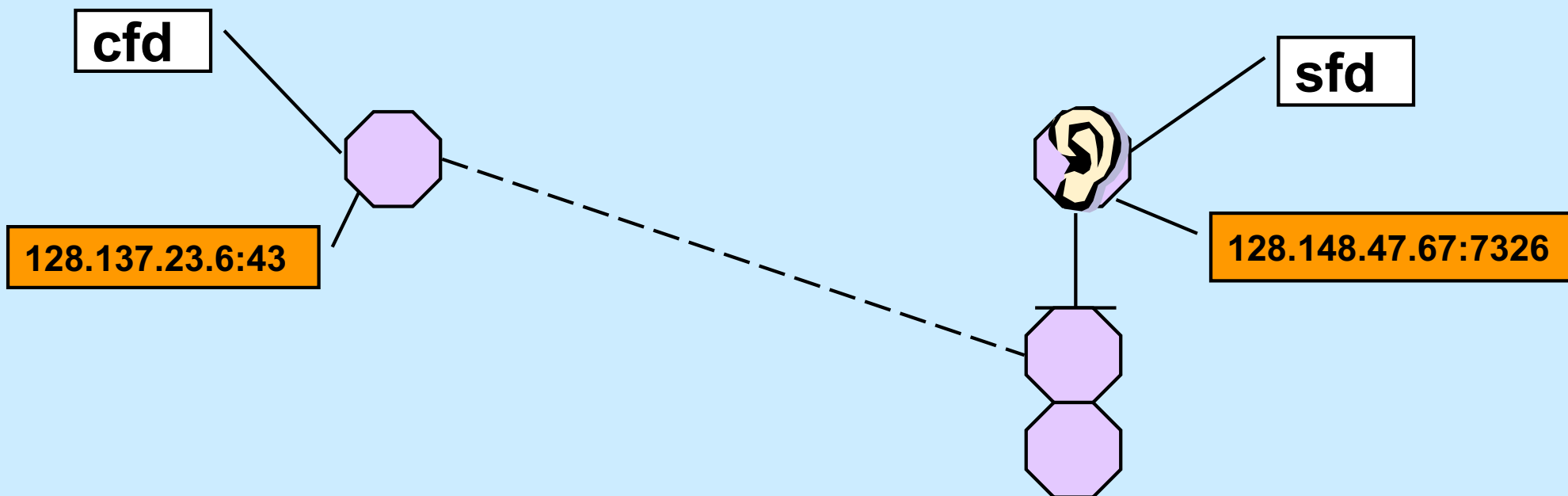
cfd

128.137.23.6:43

Streams in the Inet Domain (6)

- **Client steps**

  3) connect to server

  ```
  connect(cfd, (struct sockaddr *)&server_addr,
          sizeof(server_addr));
  ```


cfd

sfd

128.137.23.6:43

128.148.47.67:7326

Streams in the Inet Domain (7)

- **Server steps**

    4) accept connection

```
fd = accept((int)sfd, (struct sockaddr *)addr,
            (int *)&addrlen);
```

cfd

fd

sfd

128.137.23.6:43

128.148.47.67:7326
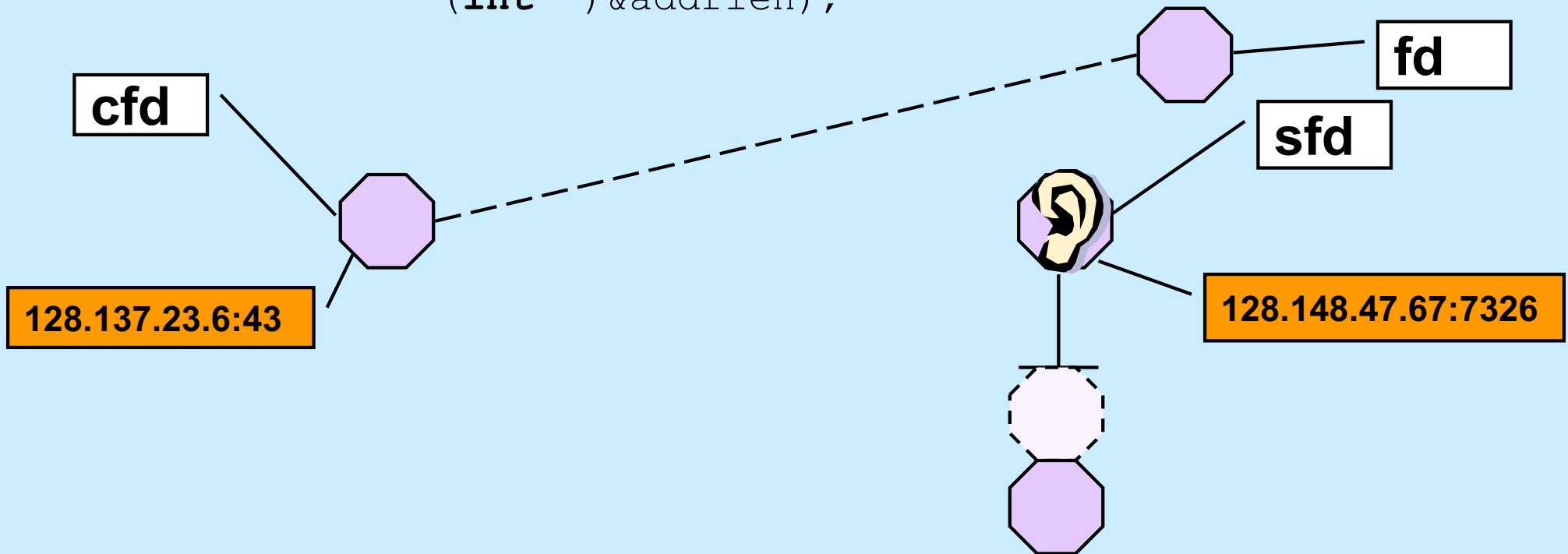
TCP Server (1)

```c
int main(int argc, char *argv[ ]) {
    if (argc != 2) {
        fprintf(stderr, "Usage: port\n");
        exit(1);
    }

    int lsocket;
    struct addrinfo tcp_hints;
    struct addrinfo *result;
```

TCP Server (2)

```
memset(&tcp_hints, 0, sizeof(tcp_hints));
tcp_hints.ai_family = AF_INET;
tcp_hints.ai_socktype = SOCK_STREAM;
tcp_hints.ai_flags = AI_PASSIVE;

int err;
if ((err = getaddrinfo(NULL, argv[1], &tcp_hints,
        &result)) != 0) {
    fprintf(stderr,"%s\n", gai_strerror(err));
    exit(1);
}
```

TCP Server (3)

```
struct addrinfo *r;
for (r = result; r != NULL; r = r->ai_next) {
    if ((lsocket =
            socket(r->ai_family, r->ai_socktype,
            r->ai_protocol)) < 0) {
        continue;
    }
    if (bind(lsocket, r->ai_addr, r->ai_addrlen) >= 0) {
        break;
    }
    close(lsocket);
}
```

TCP Server (4)

```
if (r == NULL) {
    fprintf(stderr, "Could not find local interface %s\n");
    exit(1);
}
freeaddrinfo(result);



if (listen(lsocket, 50) < 0) {
    perror("listen");
    exit(1);
}
```

TCP Server (5)

```c
while (1) {
    int csock;
    struct sockaddr client_addr;
    int client_len = sizeof(client_addr);

    csock = accept(lsocket, &client_addr, &client_len);
    if (csock == -1) {
        perror("accept");
        exit(1);
    }
```

TCP Server (6)

```c
char host_name[256];
char serv_name[256];
int err;
if ((err = getnameinfo(&client_addr,
        client_len, host_name, sizeof(host_name),
        serv_name, sizeof(serv_name), 0))) {
    fprintf(stderr, "%s/n", gai_strerror(err));
    exit(1);
}
printf("received connection from %s port %s\n",
        host_name, serv_name);
```

TCP Server (7)

```c
    switch (fork()) {
    case -1:
        perror("fork");
        exit(1);
    case 0:
        serve(csock);
        exit(0);
    default:
        close(csock);
        break;
    }
  }
  return 0;
}
```

TCP Server (8)

```c
void serve(int fd) {
    char buf[1024];
    int count;

    while ((count = read(fd, buf, 1024)) > 0) {
        write(1, buf, count);
    }
    if (count == -1) {
        perror("read");
        exit(1);
    }
    printf("connection terminated\n");
}
```

TCP Client (1)

```c
int main(int argc, char *argv[]) {
    int s;
    int sock;
    struct addrinfo hints;
    struct addrinfo *result;
    struct addrinfo *rp;
    char buf[1024];

    if (argc != 3) {
        fprintf(stderr, "Usage: tcpClient host port\n");
        exit(1);
    }
```

TCP Client (2)

```
memset(&hints, 0, sizeof(hints));
hints.ai_family = AF_INET;
hints.ai_socktype = SOCK_STREAM;

if ((s=getaddrinfo(argv[1], argv[2], &hints, &result))
        != 0) {
    fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(s));
    exit(1);
}
```

TCP Client (3)

```
for (rp = result; rp != NULL; rp = rp->ai_next) {
    if ((sock = socket(rp->ai_family, rp->ai_socktype,
            rp->ai_protocol)) < 0) {
        continue;
    }
    if (connect(sock, rp->ai_addr, rp->ai_addrlen) >= 0) {
        break;
    }
    close(sock);
}
```

TCP Client (4)

```
if (rp == NULL) {
    fprintf(stderr, "Could not connect to %s\n", argv[1]);
    exit(1);
}
freeaddrinfo(result);
```

TCP Client (5)

```
while(fgets(buf, 1024, stdin) != 0) {
    if (write(sock, buf, strlen(buf)) < 0) {
        perror("write");
        exit(1);
    }
}
return 0;
}
```

Quiz 2

The previous slide contains
`write(sock, buf, strlen(buf))`

If data is lost and must be retransmitted

a) **write returns an error so the caller can retransmit the data.**

b) **nothing happens as far as the application code is concerned, the data is retransmitted automatically.**

Quiz 3
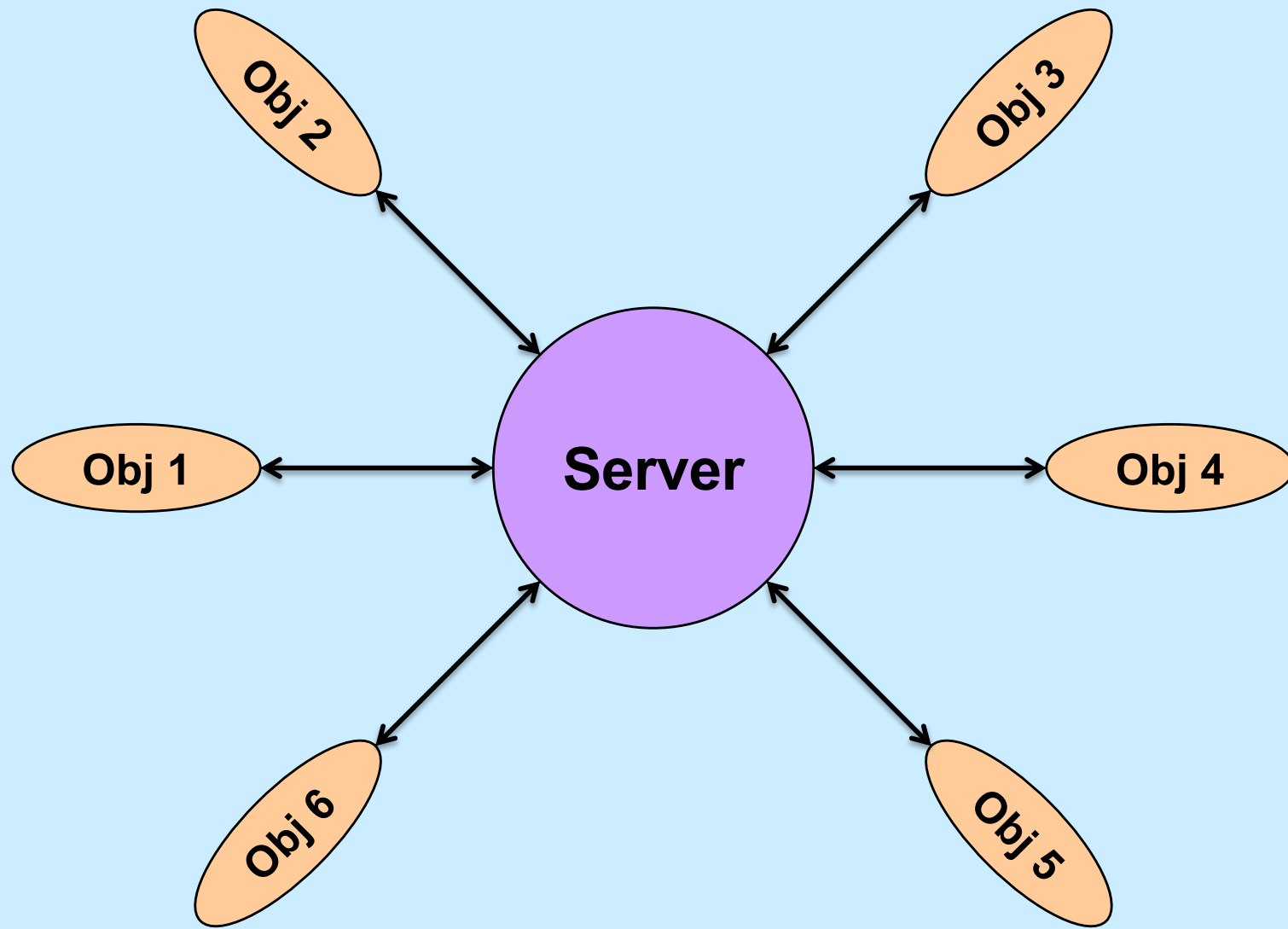
A previous slide contains
`write(sock, buf, strlen(buf))`

We lose the connection to the other party (perhaps a network cable is cut).

a) write returns an error so the caller can reconnect, if desired.

b) nothing happens as far as the application code is concerned, the connection is reestablished automatically.

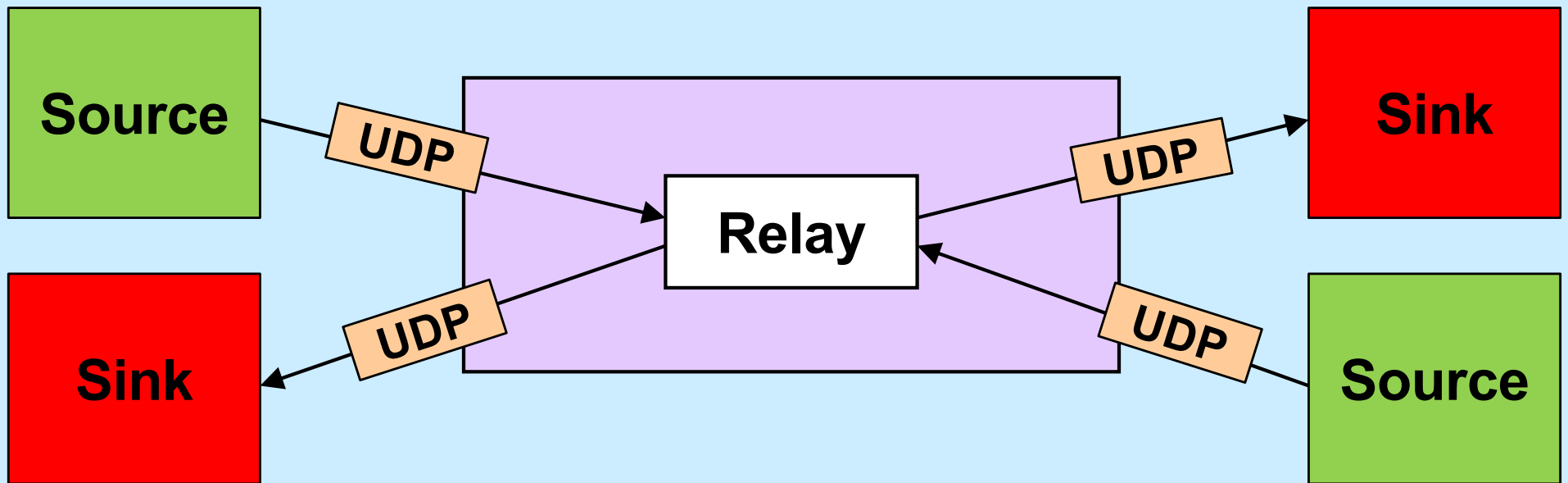# CS 33

Event-Based Programming

Event Handling

Stream Relay

Solution?

```
while(…) {
    size = read(left, buf, sizeof(buf));
    write(right, buf, size);
    size = read(right, buf, sizeof(buf));
    write(left, buf, size);
}
```

Select System Call

```
int select(
   int nfds,          // size of fd_sets
   fd_set *readfds,   // descriptors of interest
                      // for reading
   fd_set *writefds,  // descriptors of interest
                      // for writing
   fd_set *excpfds,   // descriptors of interest
                      // for exceptional events
   struct timeval *timeout
                      // max time to wait
   );
```

Relay Sketch

```c
void relay(int left, int right) {
    fd_set rd, wr;
    int maxFD = max(left, right) + 1;
    FD_ZERO(&rd); FD_SET(left, &rd); FD_SET(right, &rd);
    FD_ZERO(&wr); FD_SET(left, &wr); FD_SET(right, &wr);
    while (1) {
        select(maxFD, &rd, &wr, 0, 0);
        if (FD_ISSET(left, &rd))
            read(left, bufLR, sizeof(message_t));
        if (FD_ISSET(right, &rd))
            read(right, bufRL, sizeof(message_t));
        if (FD_ISSET(right, &wr))
            write(right, bufLR, sizeof(message_t));
        if (FD_ISSET(left, &rd))
            write(left, bufRL, sizeof(message_t));
    }
}
```

Relay (1)

```c
void relay(int left, int right) {
  fd_set rd, wr;
  int left_read = 1, right_write = 0;
  int right_read = 1, left_write = 0;
  message_t bufLR;
  message_t bufRL;
  int maxFD = max(left, right) + 1;
```

Relay (2)

```
while(1) {
  FD_ZERO(&rd);
  FD_ZERO(&wr);
  if (left_read)
    FD_SET(left, &rd);
  if (right_read)
    FD_SET(right, &rd);
  if (left_write)
    FD_SET(left, &wr);
  if (right_write)
    FD_SET(right, &wr);

  select(maxFD, &rd, &wr, 0, 0);
```

Relay (3)

```
if (FD_ISSET(left, &rd)) {
  read(left, bufLR, sizeof(message_t));
  left_read = 0;
  right_write = 1;
}
if (FD_ISSET(right, &rd)) {
  read(right, bufRL, sizeof(message_t));
  right_read = 0;
  left_write = 1;
}
```

Relay (4)

```
    if (FD_ISSET(right, &wr)) {
      write(right, bufLR, sizeof(message_t));
      left_read = 1;
      right_write = 0;
    }
    if (FD_ISSET(left, &wr)) {
      write(left, bufRL, sizeof(message_t));
      right_read = 1;
      left_write = 0;
    }
  }
  return 0;
}
```

CS 33

Linking and Libraries

Libraries

- **Collections of useful stuff**
- **Allow you to:**
  - incorporate items into your program
  - substitute new stuff for existing items
- **Often ugly …**

Creating a Library

```
$ gcc -c sub1.c sub2.c sub3.c
$ ls
sub1.c          sub2.c          sub3.c
sub1.o          sub2.o          sub3.o
$ ar cr libpriv1.a sub1.o sub2.o sub3.o
$ ar t libpriv1.a
sub1.o
sub2.o
sub3.o
$
```

Using a Library

```
$ cat prog.c
int main() {
  sub1();
  sub2();
  sub3();
}
$ cat sub1.c
void sub1() {
  puts("sub1");
}
```

```
$ gcc -o prog prog.c -L. -lpriv1
$ ./prog
sub1
sub2
sub3
```

Where does *puts* come from?

```
$ gcc -o prog prog.c -L. \
    -lpriv1 \
    -L/lib/x86_64-linux-gnu -lc
```

Static-Linking: What's in the Executable

- **ld puts in the executable:**
    - » (assuming all .c files have been compiled into .o files)
    - – all .o files from argument list (including those newly compiled)
    - – .o files from archives as needed to satisfy unresolved references
        - » some may have their own unresolved references that may need to be resolved from additional .o files from archives
        - » each archive processed just once (as ordered in argument list)
            - order matters!

Example

```
$ cat prog2.c
int main() {
  void func1();
  func1();
  return 0;
}
$ cat func1.c
void func1() {
  void func2();
  func2();
}
$ cat func2.c
void func2() {
}
```

Order Matters ...

```
$ ar t libf1.a
func1.o
$ ar t libf2.a
func2.o
$ gcc -o prog2 prog2.c -L. -lf1 -lf2
$
$ gcc -o prog2 prog2.c -L. -lf2 -lf1
./libf1.a(sub1.o): In function `func1':
func1.c:(.text+0xa): undefined reference to `func2'
collect2: error: ld returned 1 exit status
```

Substitution

```
$ cat myputs.c
int puts(char *s) {
  write(1, "My puts: ", 9);
  write(1, s, strlen(s));
  write(1, "\n", 1);
  return 1;
}
$ gcc -c myputs.c
$ ar cr libmyputs.a myputs.o
$ gcc -o prog prog.c -L. -lpriv1 -lmyputs
$ ./prog
My puts: sub1
My puts: sub2
My puts: sub3
```

An Urgent Problem

- **printf is found to have a bug**
    - perhaps a security problem
- **All existing instances must be replaced**
    - there are zillions of instances ...
- **Do we have to re-link all programs that use printf?**

Dynamic Linking

- **Executable is not fully linked**
  - **contains list of needed libraries**
- **Linkages set up when executable is run**

Benefits

- **Without dynamic linking**
  - **every executable contains copy of printf (and other stuff)**
    - » **waste of disk space**
    - » **waste of primary memory**

- **With dynamic linking**
  - **just one copy of printf**
    - » **shared by all**

Shared Objects: Unix's Dynamic Linking

1 **Compile program**

2 **Track down references with *ld***

- *archives* (containing *relocatable objects*) in ".a" files are statically linked

- *shared objects* in ".so" files are dynamically linked
  - » names of needed .so files included with executable

3 **Run program**

- *ld-linux.so* is invoked first to complete the linking and relocation steps, if necessary