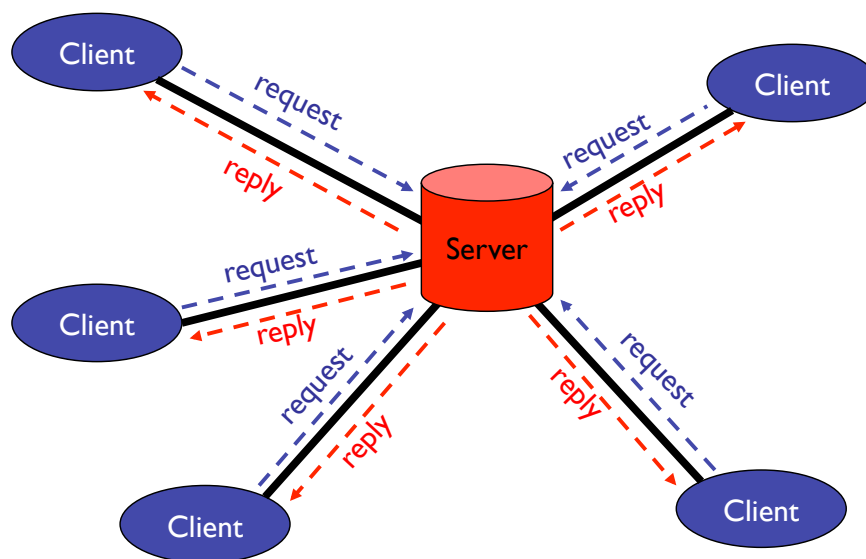


Servers



Client-Server Architecture



File I/O: File Descriptors

```
int open(char* path, int flags, ...)
```

```
int read(int fd, void* buf, int nbytes)
```

```
int write(int fd, void* buf, int nbytes)
```

```
int close(int fd)
```

Socket I/O: socket

```
int socket(int domain, int type,  
           int protocol)
```

- domain = AF_INET
- type = SOCK_STREAM
- protocol = 0 (or IPPROTO_TCP)

Socket I/O: bind

```
int bind(int sock, struct sockaddr* addr,  
        int addrlen)
```

- sock = socket file descriptor
- addr: see below
- addrlen = sizeof(addr)

```
struct sockaddr_in {  
    short sin_family; // --> AF_INET  
    u_short sin_port; // --> htons(portnum)  
    struct in_addr sin_addr;  
    // --> .s_addr = htonl(INADDR_ANY)  
}
```

Socket I/O: listen

```
int listen(int sock, int backlog)
```

- sock = socket file descriptor
- backlog = max # of unaccepted connections (e.g. 10)

Socket I/O: accept

```
int accept(int sock, struct sockaddr* addr,  
           int* addrlen)
```

- sock = listening socket
- addr gets filled in with client info
- addrlen = pointer to sizeof(struct sockaddr_in)

Blocking call!

Socket I/O: send/recv

```
int send(int sock, char* buf, size_t len,  
         int flags)
```

```
int recv(int sock, char* buf, size_t len,  
         int flags)
```

Example: send/recv

```
int conn_fd = accept(listen_fd, ..., ...);
char buf[BUF_SIZE];
int bytes = recv(conn_fd, buf, sizeof(buf) - 1, 0);
if (bytes > 0) {
    buf[bytes] = '\0'; // null terminate msg
    printf("from client: '%s'\n", buf);
    char* reply = "Hello there!\n";
    send(conn_fd, reply, strlen(reply), 0);
}
close(conn_fd);
```

Assumptions (not guaranteed!):

- 1) single **recv** gets full message
- 2) **send** delivers all bytes

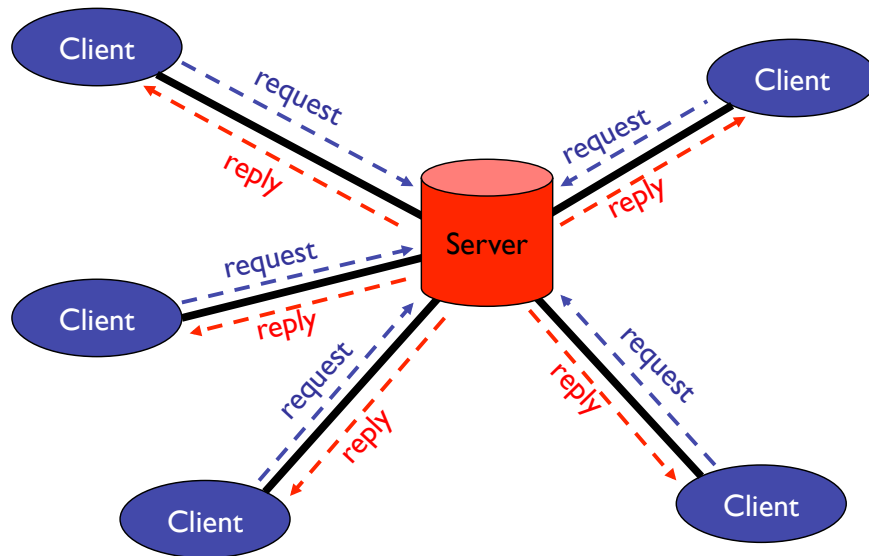
Example: Streams

Alternative: higher-level input/output functions:

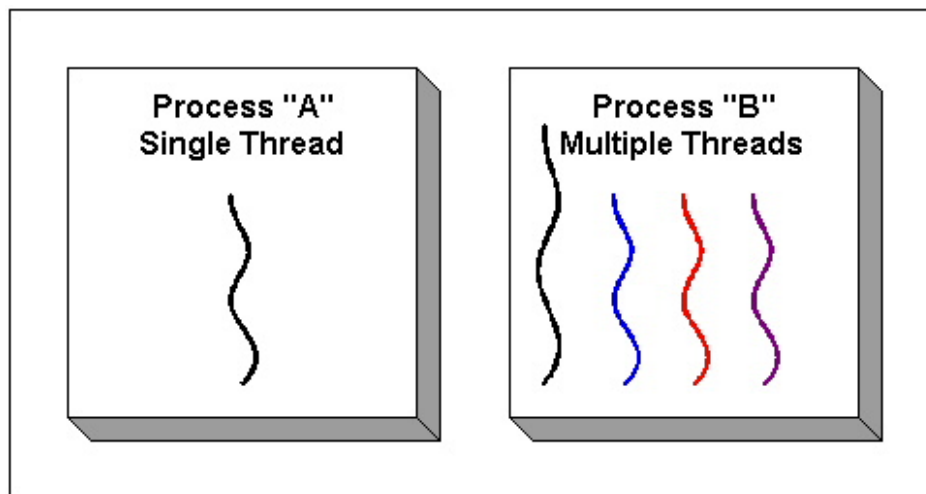
fgets, sscanf, fprintf, ...

```
int conn_fd = accept(listen_fd, ..., ...);
FILE* conn = fdopen(conn_fd, "r+"); // open stream
char buf[BUF_SIZE];
fgets(buf, sizeof(buf), conn); // read one line
printf("from client: '%s'\n", buf);
char* reply = "Hello there!\n";
fprintf(conn, "%s", reply); // write to stream
fflush(conn); // flush stream to prevent buffering
fclose(conn);
```

Client-Server Architecture



Processes and Threads



Processes and Threads

Creating Processes:

```
int fork()
```

Creating Threads:

```
int pthread_create(pthread_t* thr,  
    const pthread_attr_t* attr,  
    void* (*start_routine)(void*),  
    void* arg)
```

attr: usually NULL

start_routine: function to execute

arg: argument to function