Internetworking with Sockets

Spring 2022

Cross-host Interprocess Communication (IPC)

- Typically client-server model over network
- Server - Provides a service
- Server - Waits for clients to connect
- Clients - Connect to utilize the service
- Clients - Possibly more than one at a time

The Internet Protocol

- Each device in a network is assigned an IP address
- IPv4 32 bit, IPv6 128 bit
  - IPv4 (in dec)
    69.89.31.226 $\Leftarrow$ 4 octets
  - IPv6 (in hex)
    2001:0db8:0a0b:12f0:0000:0000:0000:0001 $\Leftarrow$ 8 16-bit blocks
- Each device may host many services
- Accessing a service requires a (IP,port) pair
- Services you know of: ssh (port 22), http (port 80), DNS (port 53), DHCP (ports 67,68)

Common Service Use Cases

Browse the World Wide Web

- Each device has a static IP
- DNS used to translate www.google.com to 216.58.213.4
- Contact service at 216.58.213.4 and port 80 (http)

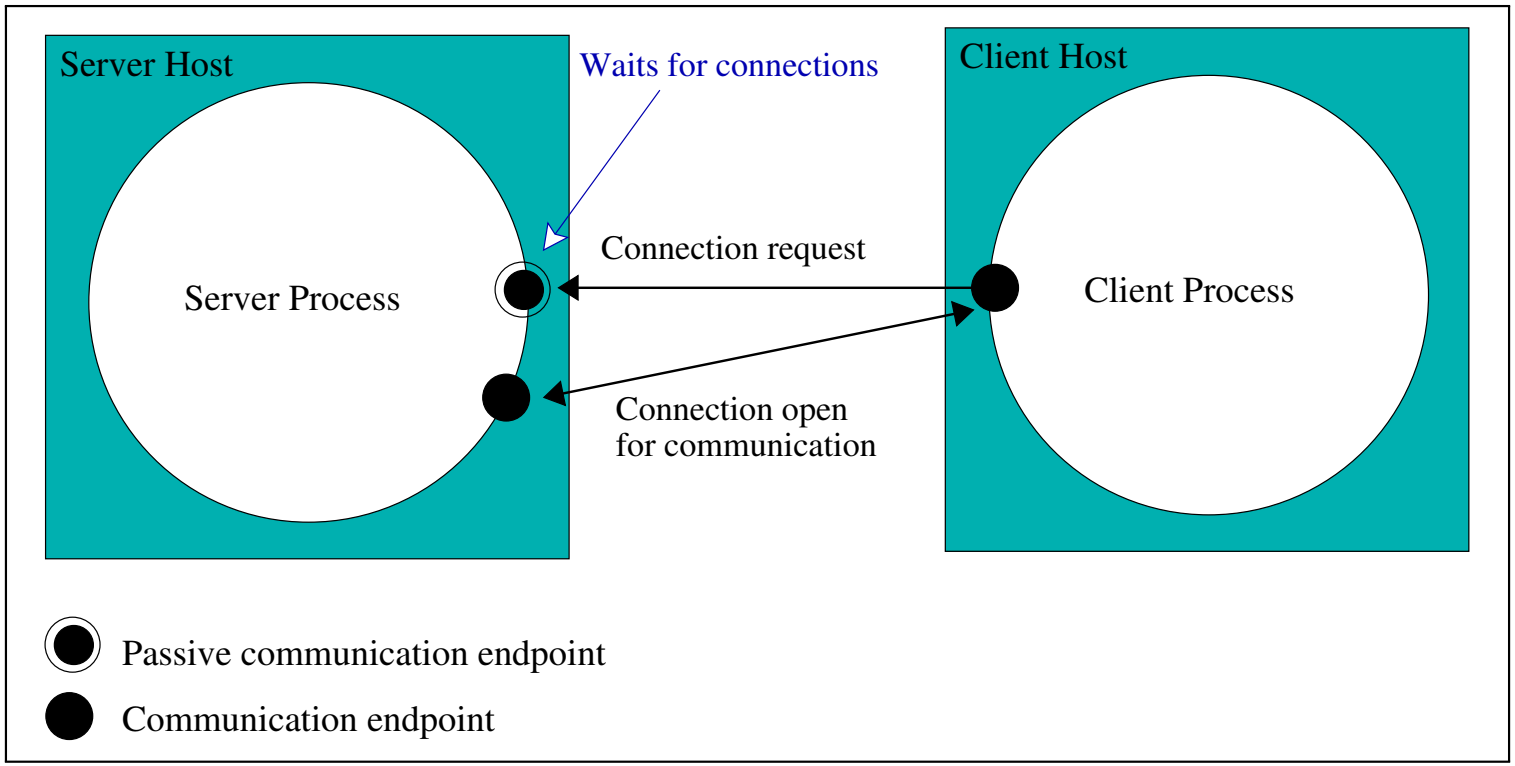
Common Service Use Cases

Your home network.

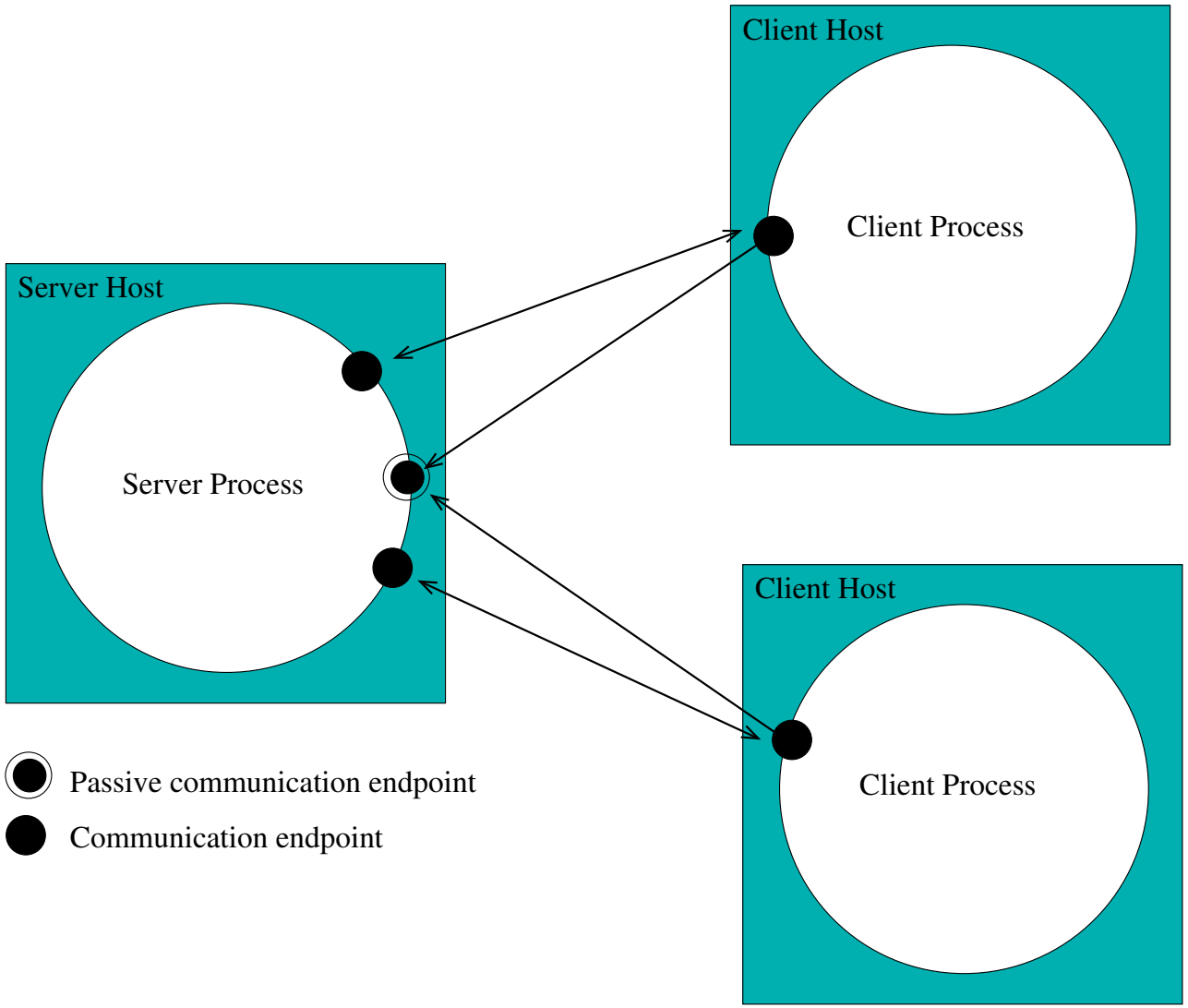
- You turn on your modem. It gets a public from you ISP (eg. 79.166.80.131)
- Your modem runs a DHCP server giving IPs in 192.168.x.y
- Your modem acts as a Internet gateway. Translates IPs from 192.168.x.y to 79.166.80.131. IP Masquerade.
- What if you need to setup a service running inside your 192.168.x.y network available to the internet?
  Do port forwarding.

The Transmission Control Protocol

- TCP Uses acknowledgments
- Non-acknowledged messages are retransmitted
- Messages re-ordered by the receiver's OS network stack
- Application sees a properly ordered *data stream*



TCP - multiple clients



Sockets

- A `socket` is a communication endpoint
- Processes refer to a socket using an *integer descriptor*
- Communication domain
  - Internet domain (over internet)
  - Unix domain (same host)
- Communication type
  - Stream (usually TCP)
  - Datagram (usually UDP)

TCP vs. UDP

	TCP	UDP
Connection Required	✓	✗
Reliability	✓	✗
Message Boundaries	✗	✓
In-Order Data Delivery	✓	✗
Socket Type	SOCK_STREAM	SOCK_DGRAM
Socket Domain	Internet	Internet
Latency	higher	lower
Flow Control	✓	✗

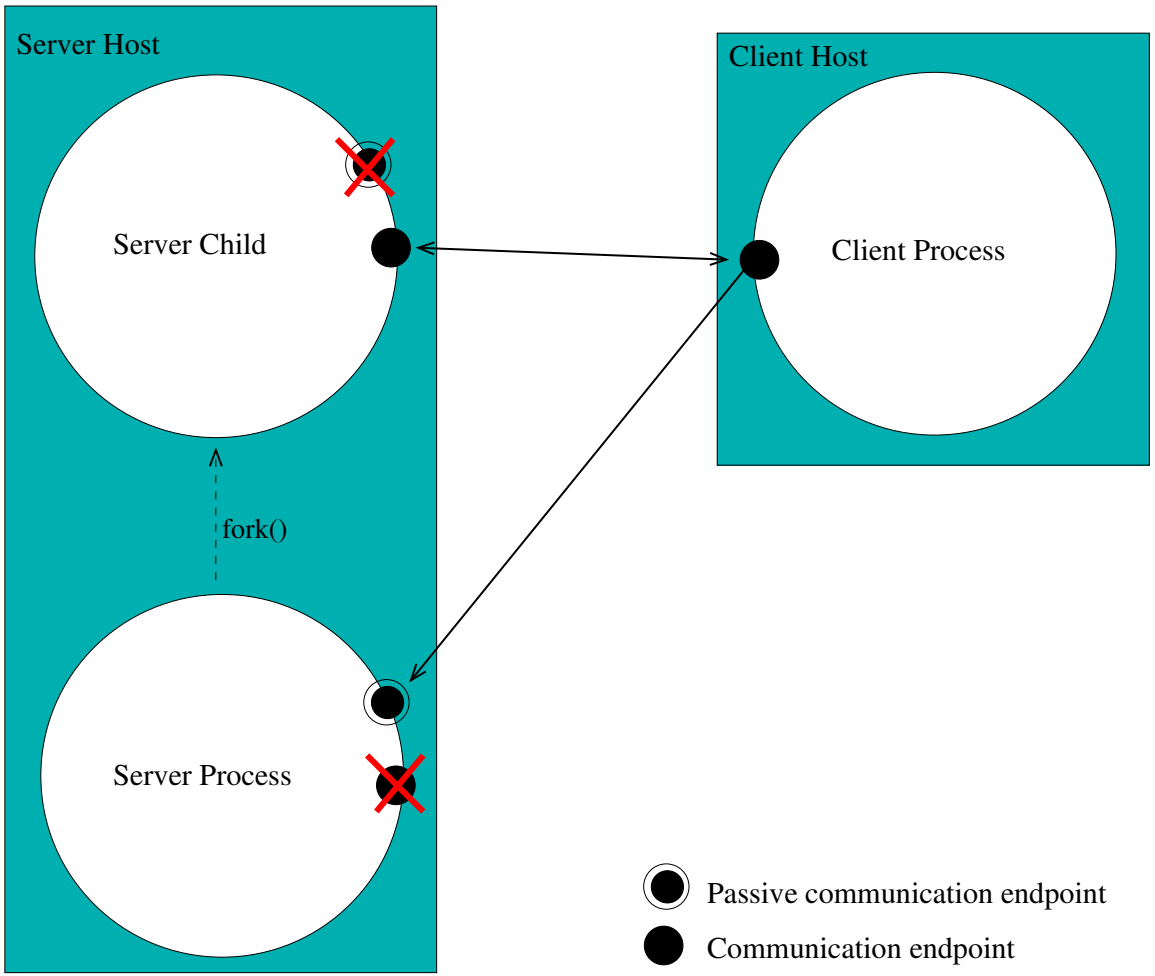
Serial Server (TCP)

Create listening socket *a*
loop
  Wait for client request on *a*
  Open two-way channel *b* with client
  while request received through *b* **do**
    Process request
    Send response through *b*
  end while
  Close file descriptor of *b*
end loop

Drawbacks:

- Serves only one client at a time
- Other clients are forced to wait or even fail

1 process per client model



- New process forked for each client
- Multiple clients served at the same time
- Inefficient, too many clients $\rightarrow$ too many processes

1 process per client model

Parent process

Create listening socket *a*
loop
Wait for client request on *a*
Create two-way channel *b* with client
Fork a child to handle the client
Close file descriptor of *b*
end loop

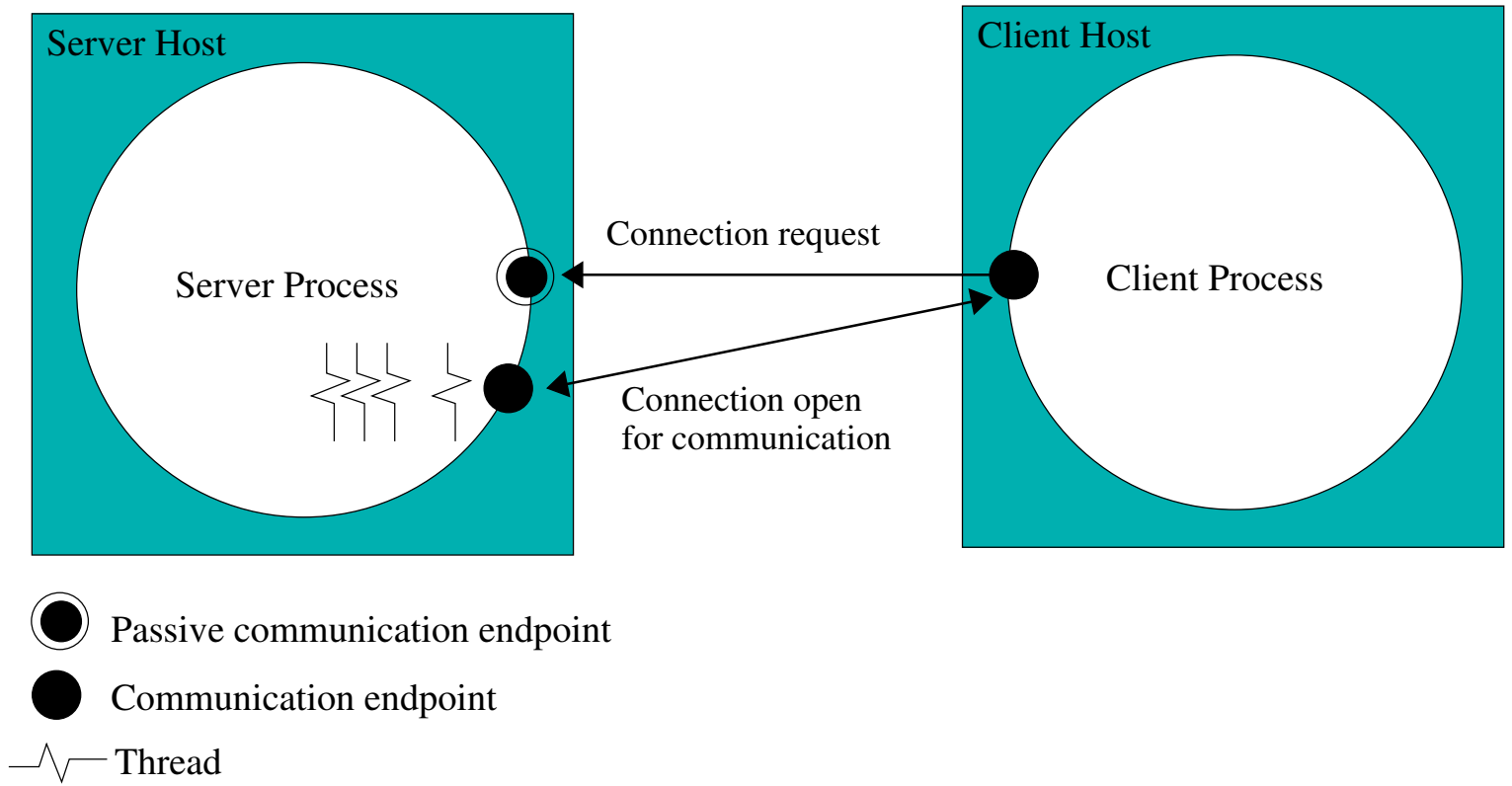
Child process

Close listening socket *a*
Serve client requests through *b*
Close private channel *b*
Exit

Parent process: why close file descriptor *b*?

- Parent doesn't need this file descriptor
- Risk of running out of file descriptors otherwise
- Enables the destruction of the channel once the other two parties (child & client) close their file descriptors
- Enables the child process to receive EOF after the client closes its end of the channel (and vice versa).

Multithreaded server model



- Multiple threads handle multiple clients concurrently
- Drawback: Requires synchronization for access to shared resources

Dealing with byte order

- Byte order poses a problem for the communication among different architectures.
- Network Protocols specify a byte ordering: ip addresses, port numbers etc. are all in what is known as *Network Byte Order*
- Convert long/short integers between *Host* and *Network Byte Order*

```
/* host to network byte order for long -32bits */
uint32_t htonl(uint32_t hostlong);
/* host to network byte order for short -16bits */
uint16_t htons(uint16_t hostshort);
/* network to host byte order for long -32bits */
uint32_t ntohl(uint32_t netlong);
/* network to host byte order for short -16bits */
uint16_t ntohs(uint16_t netshort);
```

Depicting the Byte Order ByteOrder-p16.c

```c
#include <stdio.h>
#include <arpa/inet.h>

int main(){
        uint16_t nhost = 0xD04C, nnetwork;
        unsigned char *p;
        p=(unsigned char *)&nhost;
        printf("%x %x \n", *p, *(p+1));
        /* 16-bit number from host to network byte order */
        nnetwork=htons(nhost);
        p=(unsigned char *)&nnetwork;
        printf("%x %x \n", *p, *(p+1));
        exit(1);
}
```

- Experimenting with an Intel-based (Little-Endian) machine:

```
antoulas@sazerac:~/src$ ./ByteOrder-p16
4c d0
d0 4c
antoulas@sazerac:~/src$
```

- Experimenting with a Sparc (Big-Endian/Network Byte Order) machine:

```
pubsrv1:/k24-examples>./ByteOrder-p16
d0 4c
d0 4c
pubsrv1:/k24-examples>
```

From *Domain Names* to *Addresses* and back

- An *address* is needed for network communication
- We often have to *resolve* the address from a domain name.
  ex. `spiderman.di.uoa.gr` ↔ `195.134.66.107`

```
struct hostent {
    char    *h_name;        /* official name of host */
    char    **h_aliases;    /* aliases (alt. names) */
    int     h_addrtype;     /* usually AF_INET */
    int     h_length;       /* bytelength of address */
    char    **h_addr_list;  /* pointer to array of network addresses */
};

struct hostent *gethostbyname(const char *name);

struct hostent *gethostbyaddr(const void *addr, socklen_t len, int type);
```

- For error reporting use `h_error` & `hstrerror(int err)`.
- Both calls return pointers to statically allocated `hostent` structure on sucesss and `NULL` on error.

Resolving names for machines

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

void main(int argc, char **argv){
int     i=0;
char    hostname[50], symbolicip[50];
struct  hostent *mymachine;
struct  in_addr **addr_list;

if (argc!=2 ) {printf("Usage: GetHostByName-p18 host-name\n"); exit(0);}

if ( (mymachine=gethostbyname(argv[1])) == NULL)
        printf("Could not resolved Name:  %s\n",argv[1]);
else    {
        printf("Name To Be  Resolved: %s\n", mymachine->h_name);
        printf("Name Length in Bytes: %d\n", mymachine->h_length);
        addr_list = (struct in_addr **) mymachine->h_addr_list;
        for(i = 0; addr_list[i] != NULL; i++) {
                strcpy(symbolicip , inet_ntoa(*addr_list[i]) );
                printf("%s resolved to %s \n",mymachine->h_name,symbolicip);
                }
        }
}
```

Resolving names

```
antoulas@sazerac:~/src$
antoulas@sazerac:~/src$ ./GetHostByName-p18 federal.gov.ar
Name To Be  Resolved: federal.gov.ar
Name Length in Bytes: 4
federal.gov.ar resolved to 190.210.161.110
antoulas@sazerac:~/src$
antoulas@sazerac:~/src$
antoulas@sazerac:~/src$ ./GetHostByName-p18 www.bbc.co.uk
Name To Be  Resolved: www.bbc.net.uk
Name Length in Bytes: 4
www.bbc.net.uk resolved to 212.58.246.95
www.bbc.net.uk resolved to 212.58.244.71
antoulas@sazerac:~/src$
antoulas@sazerac:~/src$ ./GetHostByName-p18 www.nytimes.com
Name To Be  Resolved: www.gtm.nytimes.com
Name Length in Bytes: 4
www.gtm.nytimes.com resolved to 170.149.161.130
antoulas@sazerac:~/src$
antoulas@sazerac:~/src$ ./GetHostByName-p18 170.149.161.130
Name To Be  Resolved: 170.149.161.130
Name Length in Bytes: 4
170.149.161.130 resolved to 170.149.161.130
antoulas@sazerac:~/src$
antoulas@sazerac:~/src$
```

Resolving IP-addresses

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

int main(int argc, char *argv[]) {
struct hostent* foundhost;
struct in_addr myaddress;

/* IPv4 dot-number into  binary form (network byte order) */
inet_aton(argv[1], &myaddress);

foundhost=gethostbyaddr((const char*)&myaddress, sizeof(myaddress), AF_INET);
if (foundhost!=NULL){
        printf("IP-address:%s Resolved to: %s\n", argv[1],foundhost->h_name);
        exit(0);
        }
else    {
        printf("IP-address:%s could not be resolved\n",argv[1]);
        exit(1);
        }
}
```

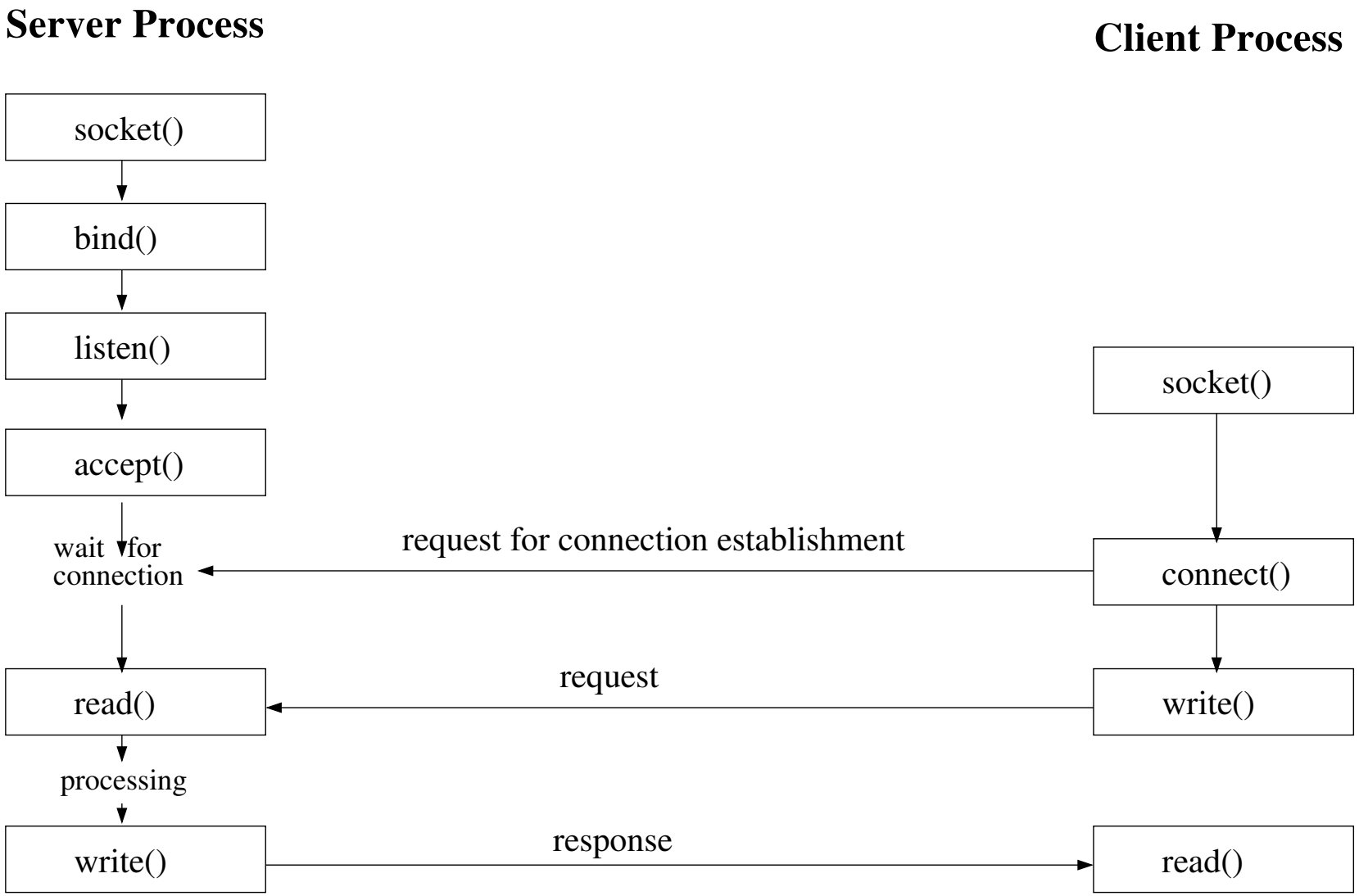
Resolving IP-addresses

```
antoulas@sazerac:~/src$
antoulas@sazerac:~/src$ ./GetHostByAddress 128.10.2.166
IP-address:128.10.2.166 Resolved to: merlin.cs.purdue.edu
antoulas@sazerac:~/src$
antoulas@sazerac:~/src$ ./GetHostByAddress 195.134.67.183
IP-address:195.134.67.183 Resolved to: sydney.di.uoa.gr
antoulas@sazerac:~/src$
```

- `gethostbyname()` and `gethostbyaddr()` have been in use.
- *POSIX.1-2001* suggests instead the use of `getnameinfo()` and `getaddrinfo()` respectively.

Our goal

Create the communication endpoint. Use it as a file descriptor.



Address Format for Sockets

- An *address* identifies a socket in a specific communication domain.
- Addresses with different formats can be passed to the socket functions – all casted to the generic `sockaddr` structure.
- Internet addresses are defined in `<netinet/in.h>`.
- Specifically in IPv4 Internet domain (`AF_INET`), a socket address is represented by the `sockaddr_in` as follows:

```
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_addt;       /* IPv4 address   */
    };
```

- `in_port_t` data type is `uint16_t` (defined in `<stdint.h>`)
- `in_addr_t` data type is `uint32_t` (defined in `<stdint.h>`)

Creating sockets

- `socket` creates an endpoint for communication
- returns a `descriptor` or `-1` on error

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

`domain` communication domain (mostly `AF_INET`)

`type` communication semantics (often `SOCK_STREAM`, `SOCK_DGRAM`)

`protocol` Use 0 as typically only one protocol is available

```
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1)
        perror("Socket creation failed!");
```

Binding sockets to addresses

- `bind` requests for an address to be assigned to a socket
- You must bind a `SOCK_STREAM` socket to a local address before receiving connections

```
int bind(int socket, const struct sockaddr *address,
         socklen_t address_len);
```

- We pass a `sockaddr_in` struct as the `address` that has at least the following members expressed in network byte-order:
  - `sin_family`: address family is `AF_INET` in the Internet domain
  - `sin_addr.s_addr`: address can be a specific IP or `INADDR_ANY`
  - `sin_port`: TCP or UDP port number

Socket binding example

```c
#include <netinet/in.h> /* for sockaddr_in */
#include <sys/socket.h>
#include <sys/types.h>
#include <arpa/inet.h>  /* for hton* */

int bind_on_port(int sock, short port) {
    struct sockaddr_in server;
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = htonl(INADDR_ANY);
    server.sin_port = htons(port);
    return bind(sock, (struct sockaddr *) &server, sizeof(server));
}
```

- `INADDR_ANY` is a special address (0.0.0.0) meaning "any address"
- `sock` will receive connections from all addresses of the host machine

listen, accept

```
int listen(int socket, int backlog);
```

- Listen for connections on a socket
- At most `backlog` connections will be queued waiting to be accepted

```
int accept(int socket, struct sockaddr *address,
           socklen_t *address_len);
```

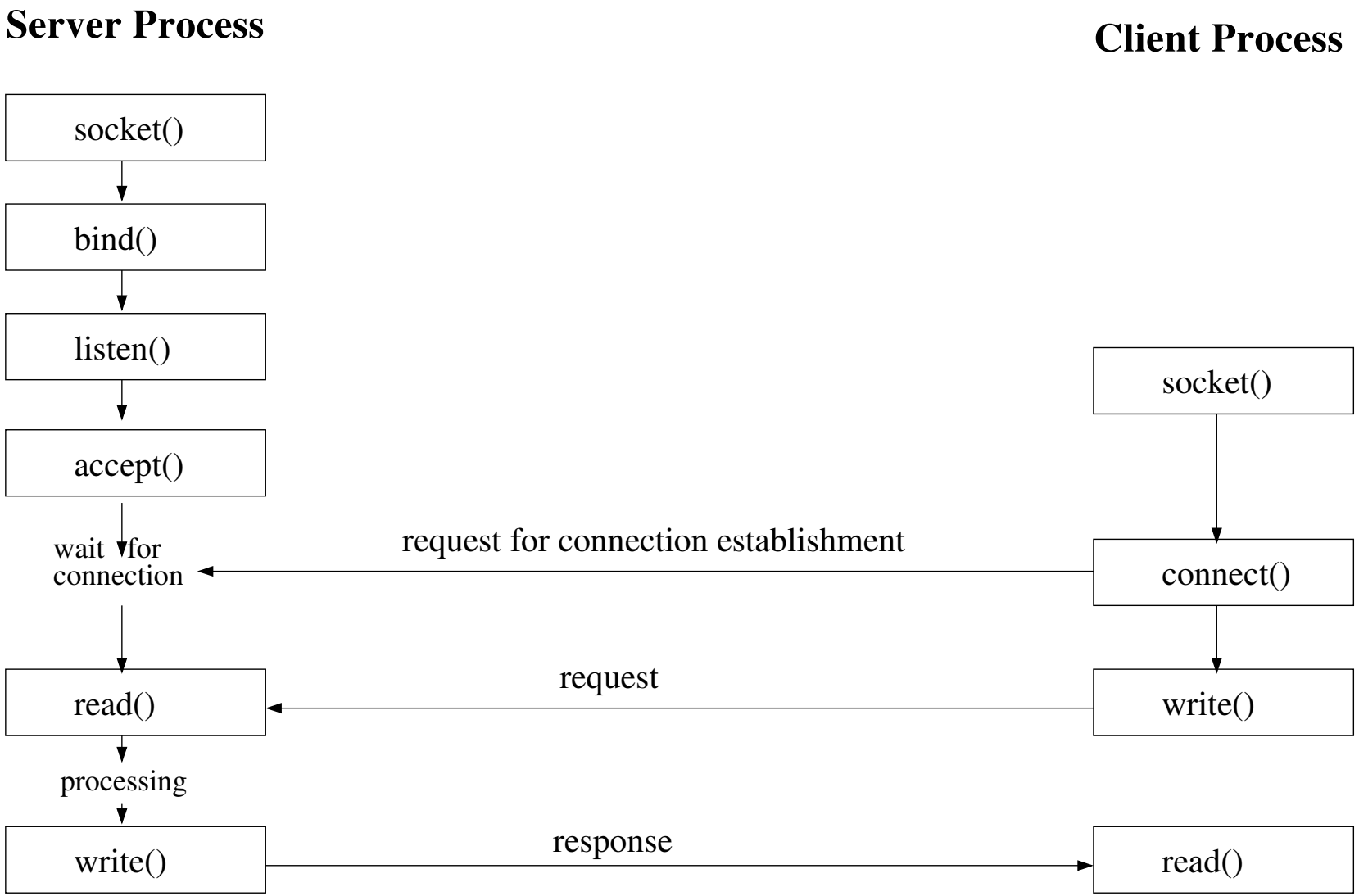
- Accepts a connection on a socket
- Blocks until a client connects/gets-interrupted by a signal
- Returns new socket descriptor to communicate with client
- Returns info on clients address through `address`.
  Pass NULL if you don't care.
- Value-result `address_len` must be set to the amount of space pointed to by `address` (or NULL).

connect

```
int connect(int socket, struct sockaddr *address,
                        socklen_t address_len);
```

- When called by a client, a connection is attempted to a listening socket on the server in `address`. Normally, the server `accepts` the connection and a communication channel is established.
- If `socket` is of type `SOCK_DGRAM`, `address` specifies the peer with which the socket is to be associated (datagrams are sent/received only to/from this peer).

TCP connection



Tips and warnings

- In Solaris compile with "`-lsocket -lnsl`"
- If a process attempts to write through a socket that has been closed by the other peer, a `SIGPIPE` signal is received.
- `SIGPIPE` is by default fatal, install a signal handler to override this.
- Use system program `netstat` to view the status of sockets.

```
antoulas@linux03:~> netstat -ant
```

- When a server quits, the listening port remains busy (state `TIME_WAIT`) for a while
- Restarting the server *fails in bind* with "`Bind: Address Already in Use`"
- To override this, use `setsockopt()` to enable `SO_REUSEADDR` before you call `bind()`.

TCP server that receives a string and replies with the string capitalized.

```c
/*inet_str_server.c: Internet stream sockets server */
#include <stdio.h>
#include <sys/wait.h>        /* sockets */
#include <sys/types.h>       /* sockets */
#include <sys/socket.h>      /* sockets */
#include <netinet/in.h>      /* internet sockets */
#include <netdb.h>           /* gethostbyaddr */
#include <unistd.h>          /* fork */
#include <stdlib.h>          /* exit */
#include <ctype.h>           /* toupper */
#include <signal.h>          /* signal */
void child_server(int newsock);
void perror_exit(char *message);
void sigchld_handler (int sig);

void main(int argc, char *argv[]) {
    int              port, sock, newsock;
    struct sockaddr_in server, client;
    socklen_t clientlen;

    struct sockaddr *serverptr=(struct sockaddr *)&server;
```

```
struct sockaddr *clientptr=(struct sockaddr *)&client;
struct hostent *rem;

if (argc != 2) {
    printf("Please give port number\n");exit(1);}
port = atoi(argv[1]);
/* Reap dead children asynchronously */
signal(SIGCHLD, sigchld_handler);
/* Create socket */
if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    perror_exit("socket");
server.sin_family = AF_INET;       /* Internet domain */
server.sin_addr.s_addr = htonl(INADDR_ANY);
server.sin_port = htons(port);      /* The given port */
/* Bind socket to address */
if (bind(sock, serverptr, sizeof(server)) < 0)
    perror_exit("bind");
```

```
/* Listen for connections */
if (listen(sock, 5) < 0) perror_exit("listen");
printf("Listening for connections to port %d\n", port);
while (1) {
    /* accept connection */
    if ((newsock = accept(sock, clientptr, &clientlen)) < 0) perror_exit("
        accept");
    /* Find client's address */
    //if ((rem = gethostbyaddr((char *) &client.sin_addr.s_addr, sizeof(
        client.sin_addr.s_addr), client.sin_family)) == NULL) {
    //herror("gethostbyaddr"); exit(1);}
    //printf("Accepted connection from %s\n", rem->h_name);
    printf("Accepted connection\n");
    switch (fork()) {    /* Create child for serving client */
    case -1:     /* Error */
        perror("fork"); break;
    case 0:      /* Child process */
        close(sock); child_server(newsock);
        exit(0);
```

```
        close(newsock); /* parent closes socket to client                */
            /* must be closed before it gets re-assigned */
    }
}

void child_server(int newsock) {
    char buf[1];
    while(read(newsock, buf, 1) > 0) {  /* Receive 1 char */
        putchar(buf[0]);               /* Print received char */
        /* Capitalize character */
        buf[0] = toupper(buf[0]);
        /* Reply */
        if (write(newsock, buf, 1) < 0)
            perror_exit("write");
    }
    printf("Closing connection.\n");
    close(newsock);   /* Close socket */
}

/* Wait for all dead child processes */
void sigchld_handler (int sig) {
    while (waitpid(-1, NULL, WNOHANG) > 0);
}

void perror_exit(char *message) {
    perror(message);
    exit(EXIT_FAILURE);
}
```

TCP client example. (definitions)

```c
/* inet_str_client.c: Internet stream sockets client */
#include <stdio.h>
#include <sys/types.h>       /* sockets */
#include <sys/socket.h>      /* sockets */
#include <netinet/in.h>      /* internet sockets */
#include <unistd.h>          /* read, write, close */
#include <netdb.h>           /* gethostbyaddr */
#include <stdlib.h>          /* exit */
#include <string.h>          /* strlen */

void perror_exit(char *message);

void main(int argc, char *argv[]) {
    int             port, sock, i;
    char            buf[256];

    struct sockaddr_in server;
    struct sockaddr *serverptr = (struct sockaddr*)&server;
    struct hostent *rem;

    if (argc != 3) {
```

TCP client example. (connection)

```
        printf("Please give host name and port number\n");
        exit(1);}
    /* Create socket */
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
        perror_exit("socket");
    /* Find server address */
    if ((rem = gethostbyname(argv[1])) == NULL) {
       herror("gethostbyname"); exit(1);
    }
    port = atoi(argv[2]); /*Convert port number to integer*/
    server.sin_family = AF_INET;       /* Internet domain */
    memcpy(&server.sin_addr, rem->h_addr, rem->h_length);
    server.sin_port = htons(port);         /* Server port */
    /* Initiate connection */
    if (connect(sock, serverptr, sizeof(server)) < 0)
```

TCP client example. (transfer loop)

```
        perror_exit("connect");
    printf("Connecting to %s port %d\n", argv[1], port);
    do {
        printf("Give input string: ");
        fgets(buf, sizeof(buf), stdin); /* Read from stdin*/
        for(i=0; buf[i] != '\0'; i++) { /* For every char */
            /* Send i-th character */
            if (write(sock, buf + i, 1) < 0)
                perror_exit("write");
            /* receive i-th character transformed */
            if (read(sock, buf + i, 1) < 0)
                perror_exit("read");
        }
        printf("Received string: %s", buf);
    } while (strcmp(buf, "END\n") != 0); /* Finish on "end" */
    close(sock);                    /* Close socket and exit */
}

void perror_exit(char *message)
{
    perror(message);
    exit(EXIT_FAILURE);
}
```

Execution

Server on linux02:

```
antoulas@linux02:~> ./server 9002
Listening for connections to port 9002
Accepted connection from linux03.di.uoa.gr
Hello world
EnD
Closing connection.
```

Client on linux03:

```
antoulas@linux03:~> ./client linux02.di.uoa.gr 9002
Connecting to linux02.di.uoa.gr port 9002
Give input string: Hello world
Received string: HELLO WORLD
Give input string: EnD
Received string: END
antoulas@linux03:~>
```