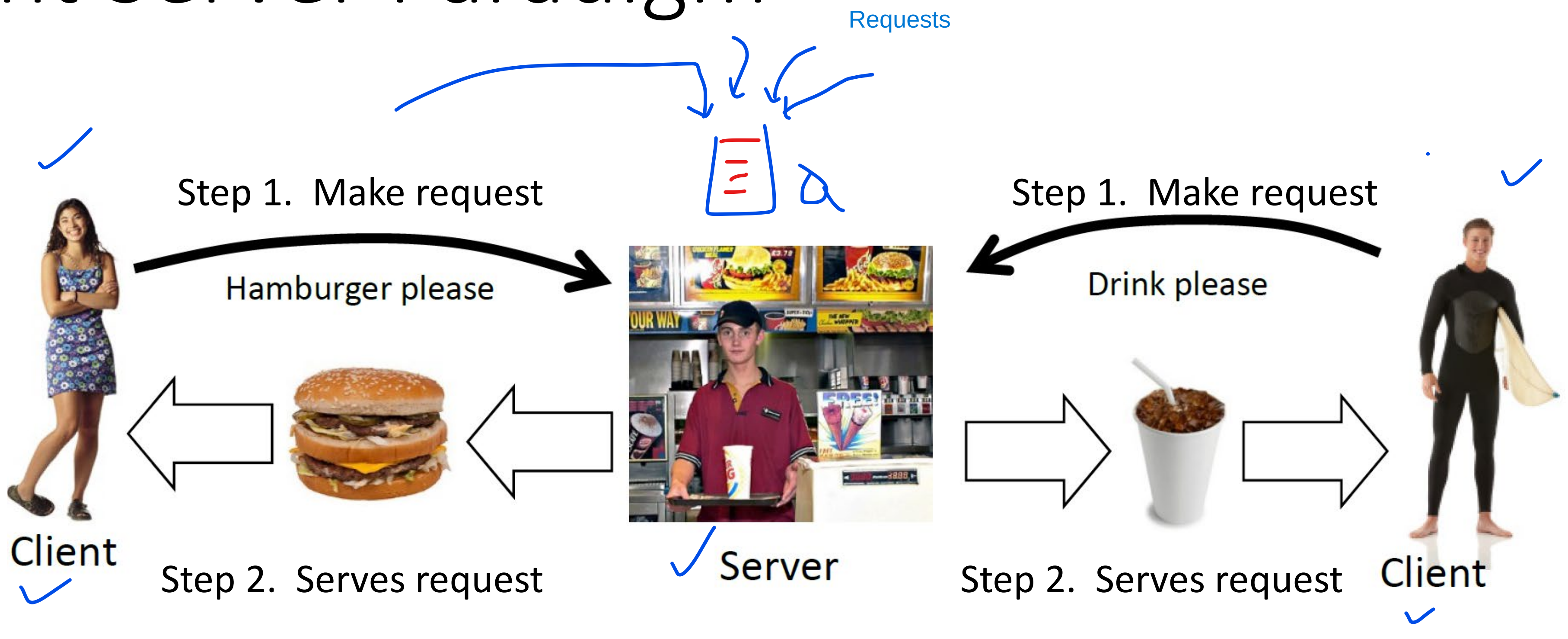
Client-Server Lab
Part 2 Internet

Outline

- Client server paradigm
- Internet
- Sockets
- Simple client-server
- Zombie processes
- TCP port assignments
- Simple exercises for the lab
- The assignment

Later

Client Server Paradigm



Requests

Step 1. Make request

Hamburger please

Step 1. Make request

Drink please

Client

Step 2. Serves request

Server

Step 2. Serves request

Client

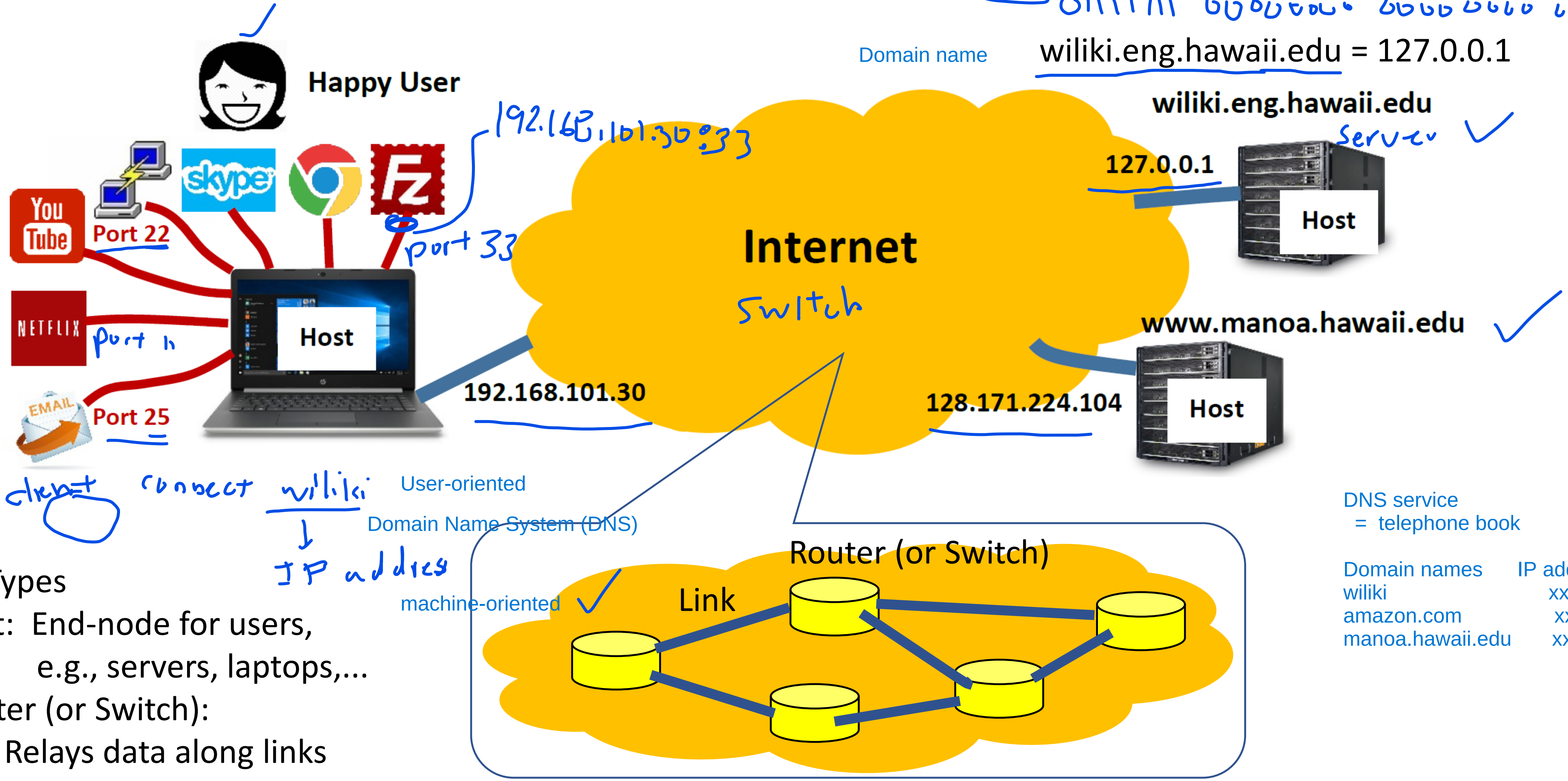
Example: client.c

Example: server.c

Internet: User point of view

IP address = 32 bit number
Written decimal-dot notation ✓
127.0.0.1, what machines use
Domain name: wiliki.eng.hawaii.edu, what humans use
01111111 00000000 00000000 00000001

Domain name wiliki.eng.hawaii.edu = 127.0.0.1



DNS service
= telephone book

Domain names	IP addresses
wiliki	xxxx
amazon.com	xxx
manoa.hawaii.edu	xxx

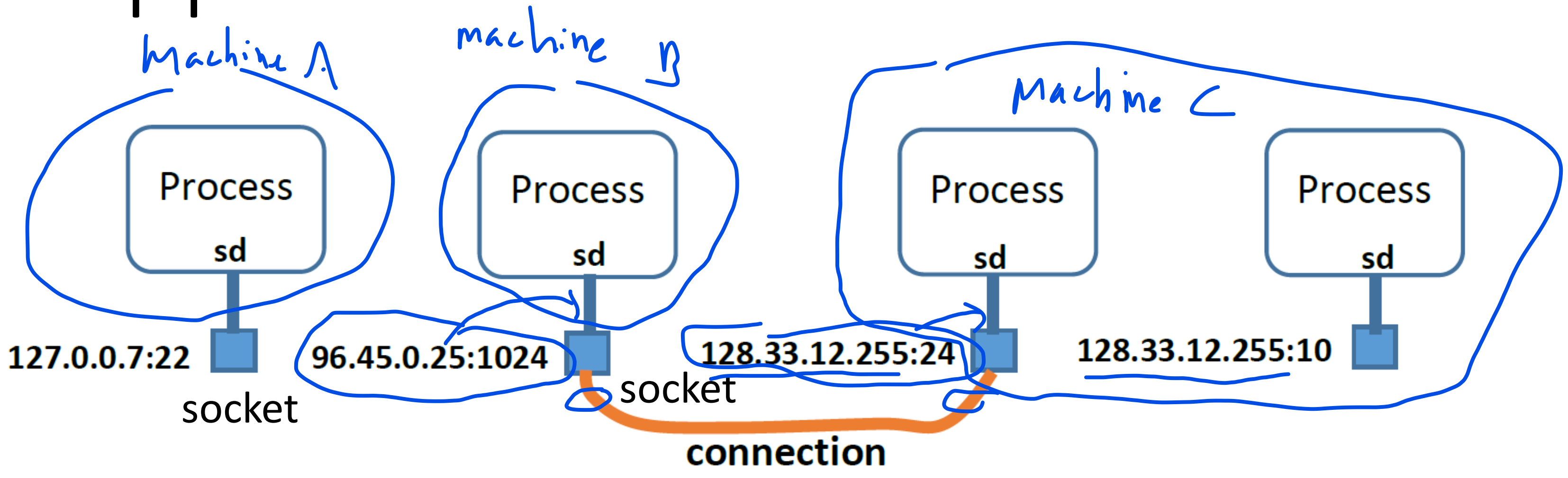
✓ Node Types

Host: End-node for users,
e.g., servers, laptops,...

Router (or Switch):
Relays data along links

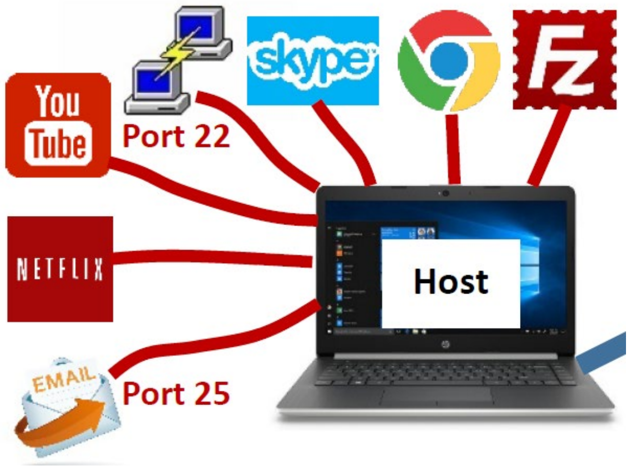
Internet: Application software

sd = socket file descriptor

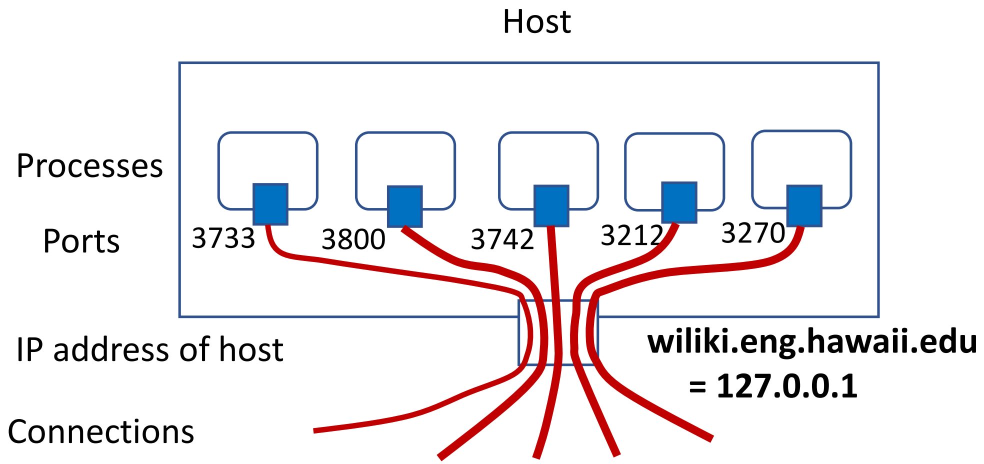


(96.45.0.25:1024 , 128.33.12.255:24)

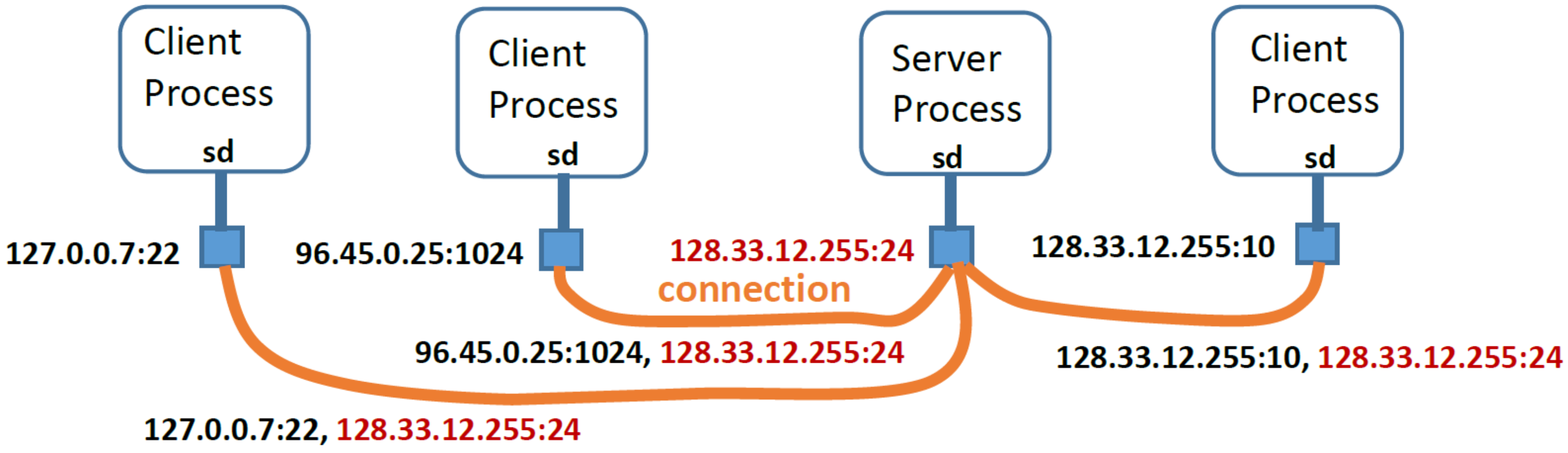
✓ Multiple applications/processes on one host



✓ Port numbers distinguish between connections to different processes on a host

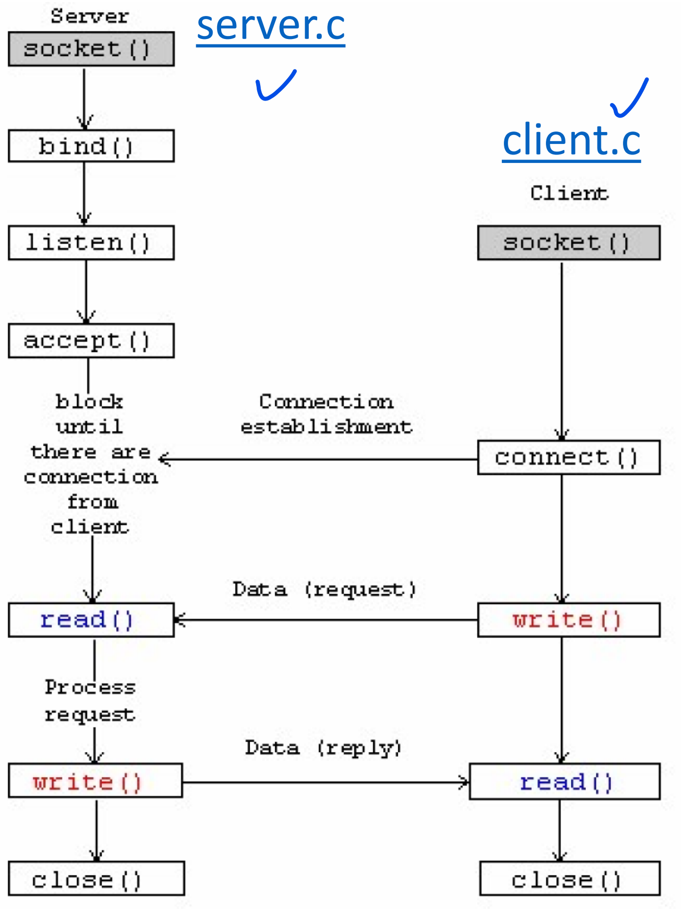


Sockets


Client Process
sd
Client Process
sd
Server Process
sd
Client Process
sd
127.0.0.7:22
96.45.0.25:1024
128.33.12.255:24
128.33.12.255:10
connection
96.45.0.25:1024, 128.33.12.255:24
128.33.12.255:10, 128.33.12.255:24
127.0.0.7:22, 128.33.12.255:24

Simple Client-Server From Beej's Guide

[Beej's Guide to Network Programming](#)



server.c ✓

client.c ✓

client wiliki...

127.0.0.1

connect to

server

listen (waiting for connections)

accept()

127.0.0.1

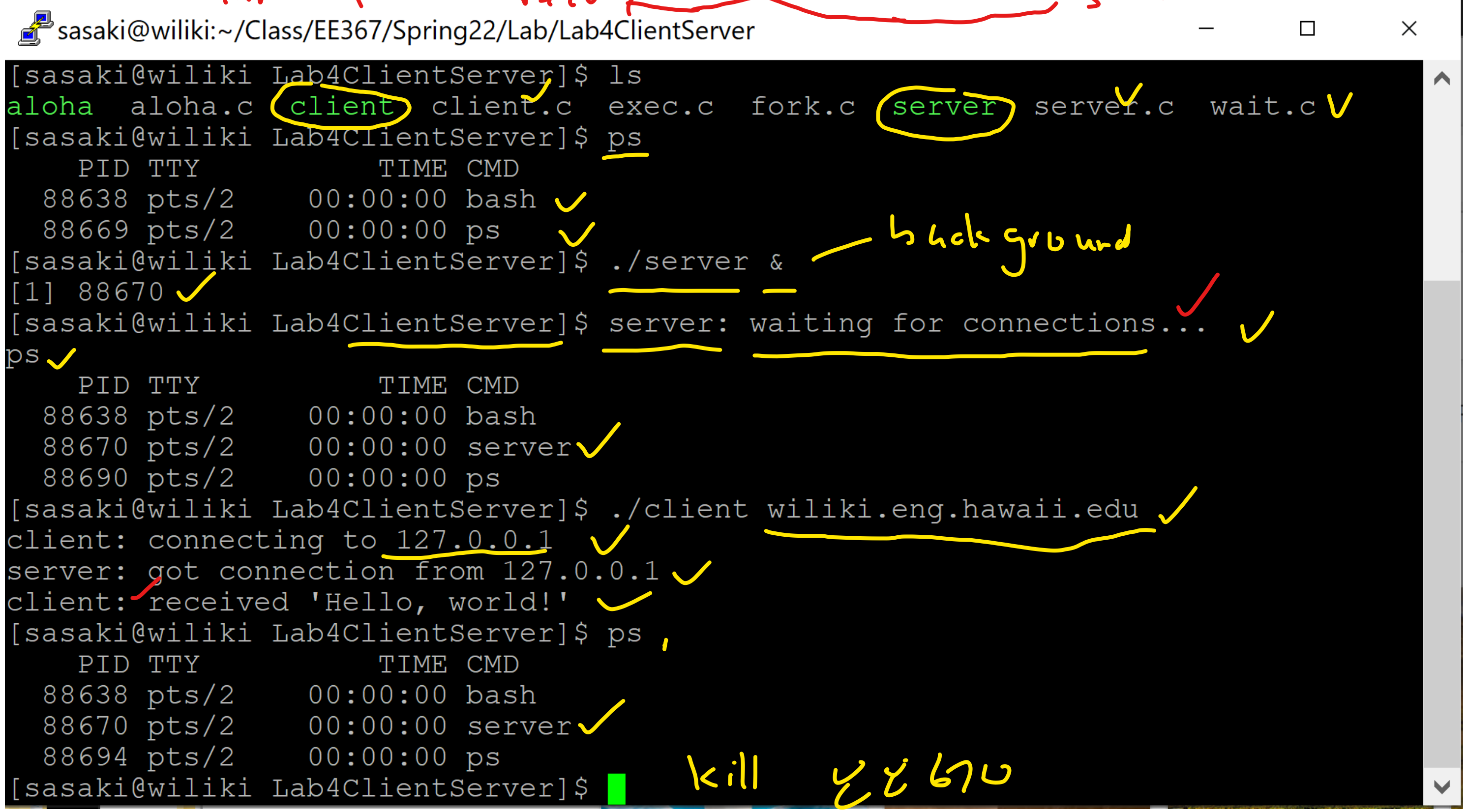
send

Hello world

recv

display

char string



sasaki@wiliki:~/Class/EE367/Spring22/Lab/Lab4ClientServer

```
[sasaki@wiliki Lab4ClientServer]$ ls
aloha   aloha.c  client  client.c  exec.c  fork.c  server  server.c  wait.c
[sasaki@wiliki Lab4ClientServer]$ ps
   PID TTY          TIME CMD
 88638 pts/2    00:00:00 bash
 88669 pts/2    00:00:00 ps
[sasaki@wiliki Lab4ClientServer]$ ./server &
[1] 88670
[sasaki@wiliki Lab4ClientServer]$ server: waiting for connections...
ps
   PID TTY          TIME CMD
 88638 pts/2    00:00:00 bash
 88670 pts/2    00:00:00 server
 88690 pts/2    00:00:00 ps
[sasaki@wiliki Lab4ClientServer]$ ./client wiliki.eng.hawaii.edu
client: connecting to 127.0.0.1
server: got connection from 127.0.0.1
client: received 'Hello, world!'
[sasaki@wiliki Lab4ClientServer]$ ps
   PID TTY          TIME CMD
 88638 pts/2    00:00:00 bash
 88670 pts/2    00:00:00 server
 88694 pts/2    00:00:00 ps
[sasaki@wiliki Lab4ClientServer]$
```

background

kill 88670

client.c

```
#define PORT "3490"
#define MAXDATASIZE 100
```

the port client will be connecting to,
i.e., port of the server

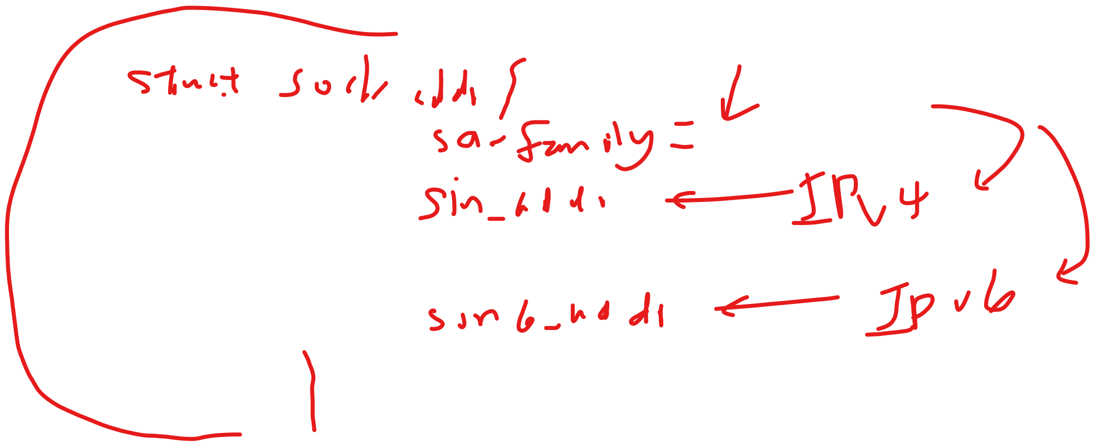
max number of bytes we can get at once

```
// get sockaddr, IPv4 or IPv6:
void *get_in_addr(struct sockaddr *sa){
        if (sa->sa_family == AF_INET) {
                return &(((struct sockaddr_in*)sa)->sin_addr);
        }
        return &(((struct sockaddr_in6*)sa)->sin6_addr);}
}
```

Two versions of IP (Internet Protocol):
- AF_INET = IPv4 (version 4)
- AF_INET6 = IPv6 (version 6)

The IP address is located in the data structure 'sa' depending on the address family (AF) it's using

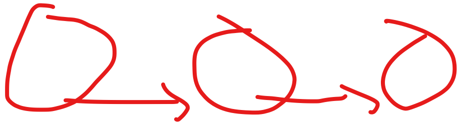
The appropriate internet address is returned



client.c – continued

```
struct addrinfo {
    int              ai_flags;
    int              ai_family;
    int              ai_socktype;
    int              ai_protocol;
    socklen_t        ai_addrlen;
    struct sockaddr *ai_addr;
    char            *ai_canonname;
    struct addrinfo *ai_next;
};
```

Node in a linked list



```
int main(int argc, char *argv[]){
```

Client connects to the server

```
        < Variable declarations >

        if (argc != 2) {
           fprintf(stderr,"usage: client hostname\n");
           exit(1);
        }
```

argv[0] argv[1]

Usage: ./client wiliki.eng.hawaii.edu
Here './client' is argv[0] and
'wiliki.eng.hawaii.edu' is argv[1]

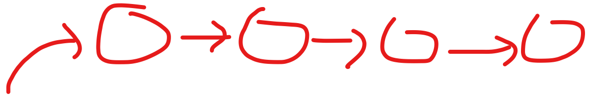
```
        memset(&hints, 0, sizeof hints);
        hints.ai_family = AF_UNSPEC;
        hints.ai_socktype = SOCK_STREAM;
```

Initializes **struct addrinfo 'hints'**, which sets up the socket

Clear **struct addrinfo 'hints'** with bytes 0

Use DNS service

5490



```
        if ((rv = getaddrinfo(argv[1], PORT, &hints, &servinfo)) != 0) {
                fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
                return 1;
        }
```

Get IP address information from the domain name in argv[1],
e.g., 'wiliki.eng.hawaii.edu'
and port # PORT

Puts information in 'hints' and 'servinfo'
servinfo is a pointer to a struct addrinfo node
servinfo is a pointer to a linked list of possible connections

Returns rv = 0 if it works, and nonzero if there is an error

client.c – continued

p → linked list of struct addrinfo nodes that has connection possibilities
Loop through all the results (linked list) and connect to the first we can

```
for(p = servinfo; p != NULL; p = p->ai_next) {
```

socket() creates and returns an end-point for a connection

```
        if ((sockfd = socket(p->ai_family, p->ai_socktype, p->ai_protocol)) == -1) {
                perror("client: socket");
                continue;
        }
```

Didn’t work so go to beginning of for-loop

connect() attempts to make a connection to server

```
        if (connect(sockfd, p->ai_addr, p->ai_addrlen) == -1) {
                close(sockfd);
                perror("client: connect");
                continue;
        }
        break;
}
if (p == NULL) {
        fprintf(stderr, "client: failed to connect\n");
        return 2;
}
```

Didn’t work so go to to beginning of for-loop

```
struct addrinfo {
    int              ai_flags;
    int              ai_family;
    int              ai_socktype;
    int              ai_protocol;
    socklen_t        ai_addrlen;
    struct sockaddr *ai_addr;
    char            *ai_canonname;
    struct addrinfo *ai_next;
};
```

Node in a linked list

Usage: ./client wiliki.eng.hawaii.edu
Here ‘./client’ is argv[0] and
‘wiliki.eng.hawaii.edu’ is argv[1]

Initializes **struct addrinfo ‘hints**’, which sets up the socket

Clear **struct addrinfo ‘hints’** with bytes 0

client.c – continued

```
    inet_ntop(p->ai_family,   get_in_addr((struct sockaddr *)p->ai_addr),   s,    sizeof s);
```

Convert IP address to char string

IP address
IP addr is void pointer

char string
s is void pointer

13 → 0 . . . 001101
↓
"0.0.0.13"

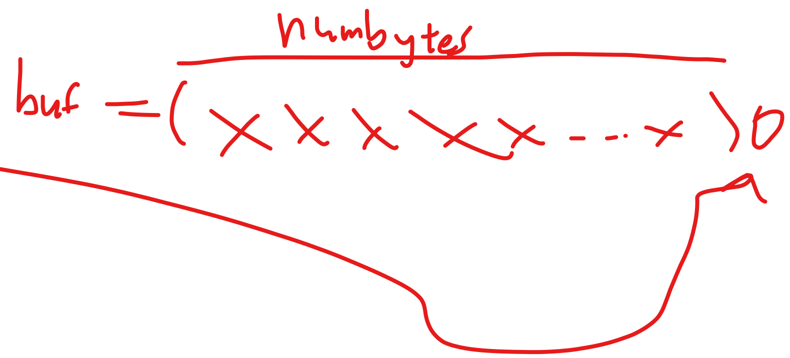
```
    printf("client: connecting to %s\n", s);

    freeaddrinfo(servinfo);
```

Free memory of the servinfo linked list

```
    if (  (numbytes  =  recv(sockfd,   buf,   MAXDATASIZE-1,   0))    == -1  ) {
        perror("recv");
        exit(1);
    }
```

Similar to read(fd, buf, length);

```
    buf[numbytes] = '\0';
    printf("client: received '%s'\n",buf);

    close(sockfd);

    return 0;
}
```



Now let’s do server.c

Zombie

```
void sigchld_handler(int s){
		while(waitpid(-1, NULL, WNOHANG) > 0);
}
```

SIgnal handler
This deletes zombie processes

```
void *get_in_addr(struct sockaddr *sa){
		if (sa->sa_family == AF_INET) {
				return &(((struct sockaddr_in*)sa)->sin_addr);
		}
		return &(((struct sockaddr_in6*)sa)->sin6_addr);
}
```

Same as in client.c
Return socket IP address, which is IPv4 or IPv6

server.c – continued

Listener:

Listens for connection requests from clients

It uses the IP address of its computer and its own port number

When it gets a request, then it creates a child process to handle the client

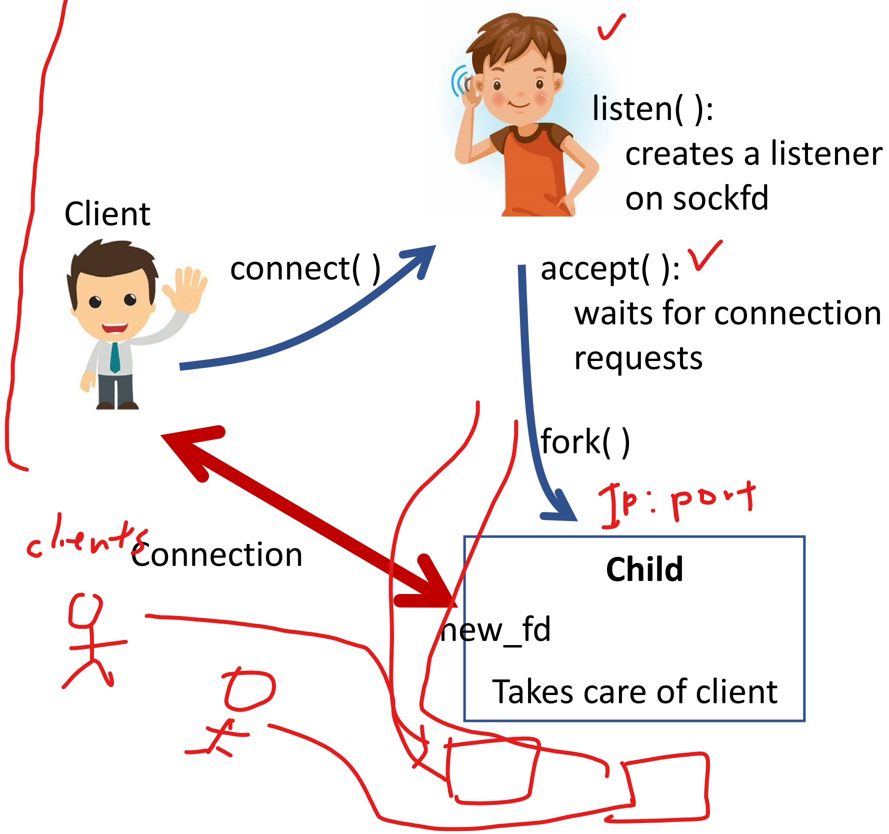
```
int main(void){

        <  Declarations of variables >

        memset(&hints, 0, sizeof hints);
        hints.ai_family = AF_UNSPEC;
        hints.ai_socktype = SOCK_STREAM;
        hints.ai_flags = AI_PASSIVE; // use my IP

        if ((rv = getaddrinfo(NULL, PORT, &hints, &servinfo)) != 0) {
                fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
                return 1;
        }
```

Set up hints for a listener

Any address family may apply

Indicates it will be used by the server

Same as client.c



Listens for connection requests from clients

listen( ): creates a listener on sockfd

Client

connect( )

accept( ): waits for connection requests

fork( )

Connection

Child

new_fd

Takes care of client

server.c – continued

```
for(p = servinfo; p != NULL; p = p->ai_next) {
        if ((sockfd = socket(p->ai_family, p->ai_socktype, p->ai_protocol)) == -1) {    perror("server: socket");
                                                                                         continue;
        }
        if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(int)) == -1) {     perror("setsockopt");
                                                                                         exit(1);
        }
        if (bind(sockfd, p->ai_addr, p->ai_addrlen) == -1) {                             close(sockfd);
                                                                                         perror("server: bind");
                                                                                         continue;
        }
        break;
}
if (p == NULL)  {        fprintf(stderr, "server: failed to bind\n");
                         return 2;
}

freeaddrinfo(servinfo);

if (listen(sockfd, BACKLOG) == -1) {           perror("listen");
                                               exit(1);
}
```

Start a socket

Allows reuse
of the socket’s
IP address

Assigns IP address
to sockfd

Free linked list of nodes

Start listening

server.c – continued

```
sa.sa_handler = sigchld_handler; // reap all dead processes
sigemptyset(&sa.sa_mask);
sa.sa_flags = SA_RESTART;
if (sigaction(SIGCHLD, &sa, NULL) == -1) {
                perror("sigaction");
                exit(1);
}
```

Mistake

Forget

~~Start a socket~~

~~Allows reuse of the socket's IP address~~

~~Assigns IP address to sockfd~~

Mistake

forget

~~Free linked list of nodes~~

~~Start listening~~

server.c – continued

Parent:
Keeps listening for connection requests from clients at sockfd
Creates children to take care of connections to clients at new_fd
Child:
Takes care of connections to clients at new_fd

```
printf("server: waiting for connections...\n");
while(1) {     // main accept() loop
        sin_size = sizeof their_addr;
        new_fd = accept(sockfd, (struct sockaddr *)&their_addr, &sin_size);
        if (new_fd == -1) {
                perror("accept");
                continue;
        }

        inet_ntop(their_addr.ss_family,   get_in_addr((struct sockaddr *)&their_addr),    s,   sizeof s);
        printf("server: got connection from %s\n", s);

        if (!fork()) {
                close(sockfd);
                if (send(new_fd, "Hello, world!", 13, 0) == -1)
                        perror("send");
                close(new_fd);
                exit(0);
        }
        close(new_fd);
}
return 0;
}
```

Waits until listener gets a connection
request from a client
newfd = fd for the new connection for the client

Prints out IP address of client, making the request

Child: takes care of request from client

Child doesn't need listener

Child sends "hello world!" to client; similar to write(fd, buf, length);

Close the connection

Parent: goes back to waiting for the next client, i.e., keeps listening

Close the connection of the client, since the child takes care of it
Listener is still alive in the parent