Lab 1: A Basic Web Client (Week 2)

Lab due: Monday Sep 21st @ 11:59pm

Week 2 Goals…

- Complete Part 2 of `worksheet.txt`
- Making `send`/`recv` calls with error handling
- Parsing response headers
- Saving the output file

Review of some socket stuff…

socket()

Server

socket()

bind()

listen()

accept()

recv()

send()

close()

make TCP connection

socket()

Client

Initiate a descriptor and OS data structures including receive/send buffers.

connect()

send()

recv()

close()

connect()

Server

socket()

bind()

listen()

accept()

recv()

send()

close()

make TCP connection

socket()

Client

connect()

send()

recv()

close()

connect()

Server

socket()

bind()

listen()

accept()

recv()

send()

close()

make TCP connection

socket()

Client

connect()

send()

recv()

close()

Once connect() returns on the client, socket is connected and we can proceed with send(), recv()

connect()

Server

socket()

bind()

listen()

accept()

recv()

send()

close()

Client

socket()

connect()

send()

recv()

close()

make TCP connection

On the server side:
It must call and return from accept()

Once connect() returns on the client, socket is connected and we can proceed with send(), recv()

connect() function prototype

```
int connect(int sockfd,
            const struct sockaddr *remoteAddress,
            socklen_t addressLength)
```

connect() function prototype

```
int connect(int sockfd,
            const struct sockaddr *remoteAddress,
            socklen_t addressLength)
```

How do we get the IP address?

How do you get the IP address?

Use `getaddrinfo`

Let’s look at the starter code example: `getaddrinfo_example.c`

What’s going on here?

```
inet_ntoa(((struct sockaddr_in*)result->ai_addr)->sin_addr)
```

What’s going on here?

```
inet_ntoa(((struct sockaddr_in*)result->ai_addr)->sin_addr)
```

The *addrinfo* structure used by **getaddrinfo**() contains the following fields:

```
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;
};
```

What's going on here?

```
inet_ntoa(((struct sockaddr_in*)result->ai_addr)->sin_addr)
```

type: `struct sockaddr`

The *addrinfo* structure used by **getaddrinfo**() contains the following fields:

```
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;
};
```

What's going on here?

```
inet_ntoa(((struct sockaddr_in*)result->ai_addr)->sin_addr)
```

type: `struct sockaddr`

NAME top

sockaddr, sockaddr_storage, socklen_t – socket address

LIBRARY top

Standard C library (*libc*)

SYNOPSIS top

```
#include <sys/socket.h>

struct sockaddr {
    sa_family_t     sa_family;      /* Address family */
    char            sa_data[];      /* Socket address */
};

struct sockaddr_storage {
    sa_family_t     ss_family;      /* Address family */
};

typedef /* ... */ socklen_t;
typedef /* ... */ sa_family_t;
```

Generic char pointer

DESCRIPTION top

sockaddr
Describes a socket address.

What's going on here?

```
inet_ntoa(((struct sockaddr_in*)result->ai_addr)->sin_addr)
```

type: `struct sockaddr`

NAME top

sockaddr, sockaddr_storage, socklen_t – socket address

LIBRARY top

Standard C library (*libc*)

SYNOPSIS top

```
#include <sys/socket.h>

struct sockaddr {
    sa_family_t     sa_family;      /* Address family */
    char            sa_data[];      /* Socket address */
};

struct sockaddr_storage {
    sa_family_t     ss_family;      /* Address family */
};

typedef /* ... */ socklen_t;
typedef /* ... */ sa_family_t;
```

Generic char pointer

DESCRIPTION top

sockaddr
Describes a socket address.

What's going on here?

```
inet_ntoa(((struct sockaddr_in*)result->ai_addr)->sin_addr)
```

type: `struct sockaddr_in`

NAME top

sockaddr_in, in_addr, in_addr_t, in

LIBRARY top

Standard C library (*libc*)

SYNOPSIS top

IPv4 address

```
#include <netinet/in.h>

struct sockaddr_in {
    sa_family_t     sin_family;     /* AF_INET */
    in_port_t       sin_port;       /* Port number */
    struct in_addr  sin_addr;       /* IPv4 address */
};

struct in_addr {
    in_addr_t s_addr;
};

typedef uint32_t in_addr_t;
typedef uint16_t in_port_t;
```

DESCRIPTION top

sockaddr_in

Describes an IPv4 Internet domain socket address.

What's going on here?

```
inet_ntoa(((struct sockaddr_in*)result->ai_addr)->sin_addr)
```

type: `struct in_addr`

NAME top

sockaddr_in, in_addr, in_addr_t, in_port_t – IPv4 socket address

LIBRARY top

Standard C library (*libc*)

SYNOPSIS top

```
#include <netinet/in.h>

struct sockaddr_in {
    sa_family_t     sin_family;     /* AF_INET */
    in_port_t       sin_port;       /* Port number */
    struct in_addr  sin_addr;       /* IPv4 address */
};

struct in_addr {
    in_addr_t s_addr;
};

typedef uint32_t in_addr_t;
typedef uint16_t in_port_t;
```

IPv4 address

DESCRIPTION top

sockaddr_in

Describes an IPv4 Internet domain socket address.

What's going on here?

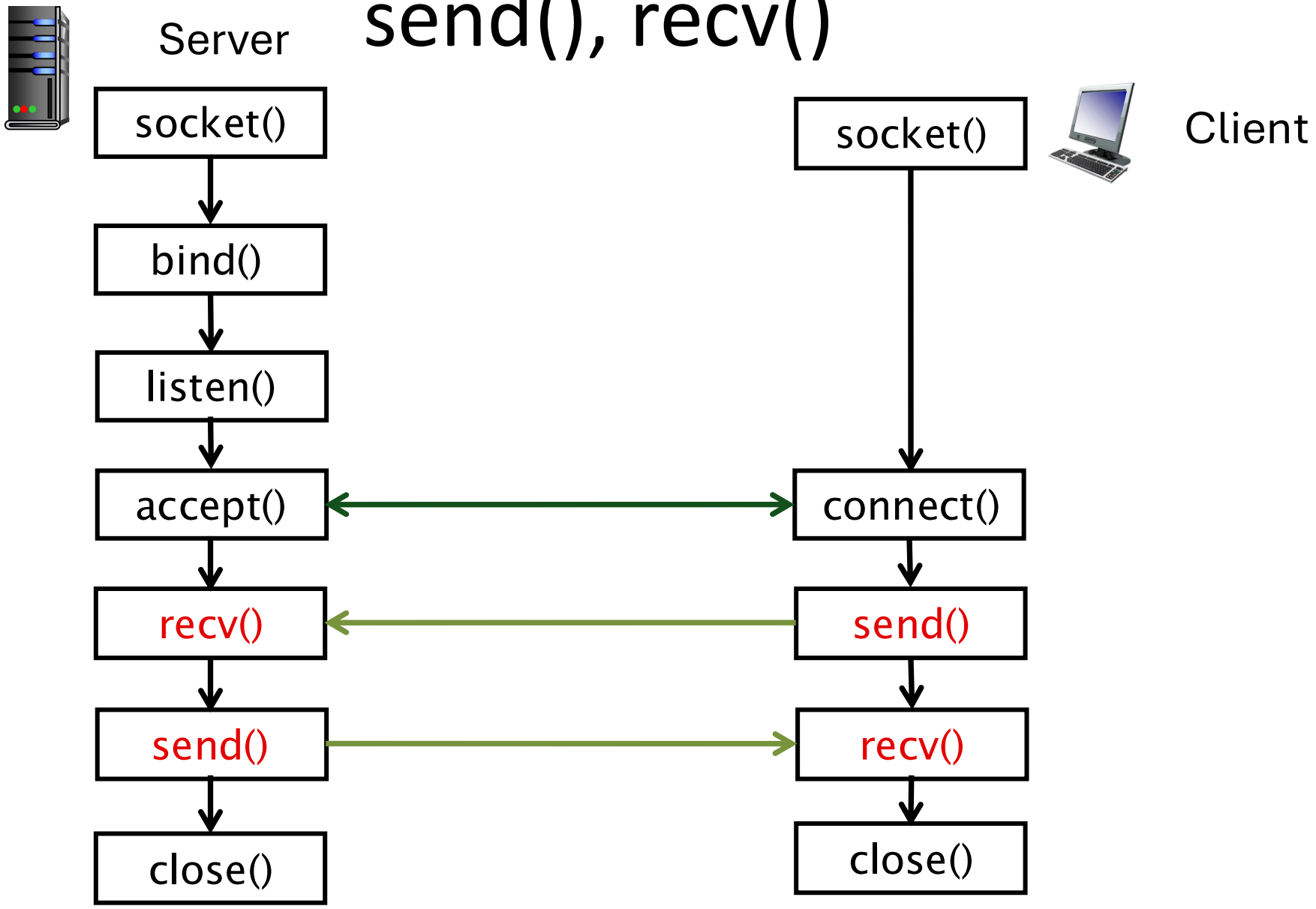
```
inet_ntoa(((struct sockaddr_in*)result->ai_addr)->sin_addr)
```

IPv4 address

From `man inet_ntoa:`

The routine **`inet_ntoa`**`()` takes an Internet address and returns an ASCII string representing the address in ‘.’ notation

send(), recv()


Server
socket()
bind()
listen()
accept()
recv()
send()
close()
socket()
Client
connect()
send()
recv()
close()

send() function prototype

```
ssize_t send(int sockfd, const void *msg, msgLength, int flags)
```

1. What does `send()` assume about the inputs?
2. What does `send()` return on success?
3. What does `send()` return if there isn't any room in the socket send buffer?
4. What does `send()` return if there was an error?
5. How do you know when you are done sending?

recv() function prototype

```
ssize_t recv (int sockfd, void *rcvBuffer, size_t bufferLength, int flags)
```

1. What does `recv()` assume about the inputs?
2. What does `recv()` return on success?
3. What does `recv()` return if there isn't anything in the socket receive buffer?
4. What does `recv()` return if there was an error?
5. How do you know when you are done receiving?

close()

Server

socket()

bind()

listen()

accept()

recv()

send()

close()

Client

socket()

connect()

send()

recv()

close()

Both sides must close the connection.

Cannot send or recv once you call close.

Important things to consider…

- The HTTP response header is interpretable as text (ASCII characters) but the HTTP response body may not be. Could just be bytes representing an image or a PDF.
- You'll need to remove the response header before saving body to a file.
- You may assume that the files you'll be retrieving are no larger than one megabyte. This means you can statically declare storage space for the response, which makes life a bit easier.
- How do you know you're done parsing the HTTP response?

Good style…

- Good error detection and handling
- Use descriptive names for variables
- Write modular code (not everything has to go into main)
- Define constants and use them (example `#define WORD 100`)
- Avoid global variables
- Code comments

Systems programming tips:

1. writing a small bit of code
2. testing w/ `valgrind` and `gdb`
3. brief comments
4. repeat step 1

Students who follow this have a much higher rate of succeeding in the course!