Project #1: Socket Programming

CS341a Fall 2018

IMPLEMENTING A STRING ENCRYPTION SERVICE

DISCLAIMER: THESE SLIDES WERE ORIGINALLY MADE BY NMSL LAB.

WE HAVE MODIFIED IT FOR OUR USE.

Lab Session Logistics

5 lab sessions (Schedule my change, keep an eye onKLMS)

- 8/30 (TCP Socket)
- 9/20 (KENS1)
- 10/4 (KENS2)
- *10/25 (KENS3)
- 11/22 (Routing using SDN tools)

No attendance check

- Hints may be provided during lab session, but not shown in the slides
- But, a question already explained clearly in the lab session will be silently ignored…

2~3 weeks for each project.

Q&A Board in KLMS

Officially, we take questions through KLMS Q&A board

- You are highly encouraged to answer peer students' questions on Q&A board.
- You will be given additional points for your active participation online.
- HOWEVER, do not post any codes.

All questions related to projects will be **public**!

- If you post it in private, we will change it to public

Personal questions can be posted in private on the board or inquired via email

- appointment, grading, …

Other course workload

Quiz

- approximately 10~12 short online quizzes.
- consists of 5~7 short multiple choice questions.
- students will be informed beforehand.

HW Assignment (Textbook questions, wireshark)

- 3~4 homework assignments.

Reading Assignment (essay)

- 3~4 essay writing throughout this course.

Socket Programming Outline

Objectives

Recall on socket programming
- Concept of socket
- Socket APIs for connection-oriented communication

Assignment requirements

Deliverables

Submission guideline

Objectives

To review basic concepts and API of network socket

To implement a connection-oriented, client-server protocol based on a given specification

To send/receive data via socket

Learn a basic encryption scheme (Vigenère Cipher)

Extra credit (2pt):

- Implement another version of non-blocking server using select()

Assignment Overview

All protocols are implemented over TCP sockets

Implement a string encryption/decryption service

- A client sends a string, and then server encrypts/decrypts the string and returns it to the client.

Vigenère Cipher

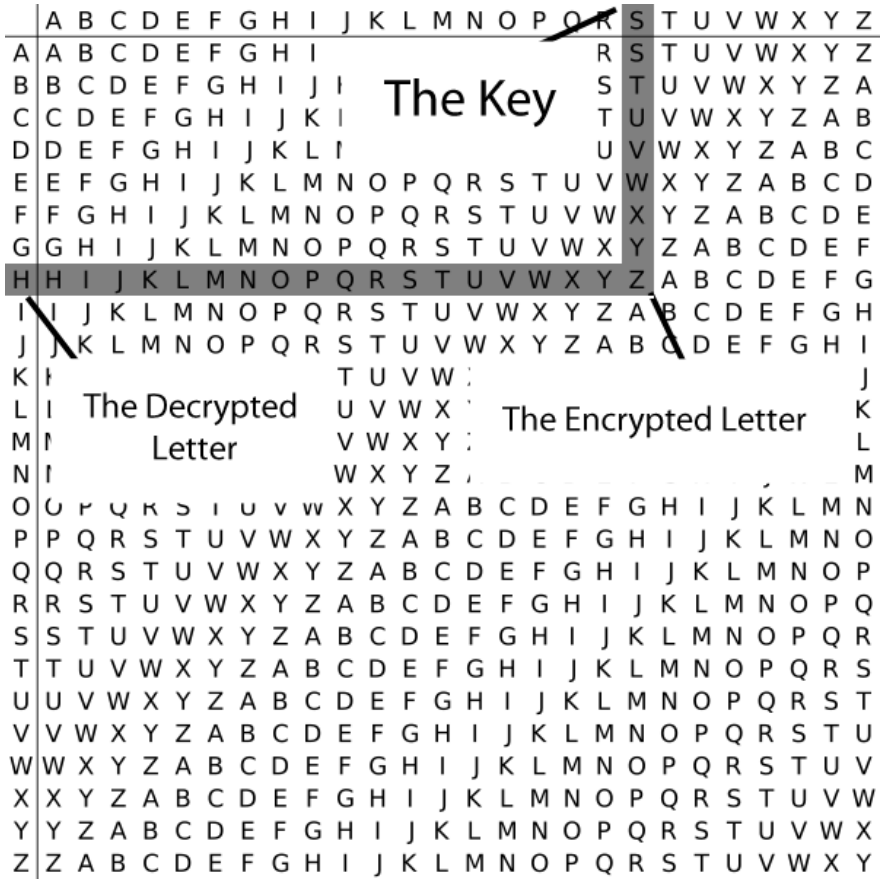
Vigenère cipher (a.k.a shift cipher): simple form of encryption

Each character is replaced with a letter corresponding to ***n*** letters up in alphabet

- E.g., a → b, when n = 1. a → d, when n = 3.

Keyword determines values of n's

- n_i = Alphabetical order of each alphabet. (e.g. a → 0 … z→ 25)
- Keyword example: **cake**
  n values: 2, 0, 10, 4




```
Example)
Keyword: cake
Plaintext: networks
Ciphertext: pedaqruw
```

For this project, there might be
characters with uppercase alphabet, or special characters.

- Convert uppercase characters to lowercase (see `int tolower(int c);`)
- After the conversion, encrypt/decrypt only lowercase characters
  - Ignore special characters (e.g., white spaces, line break, …)

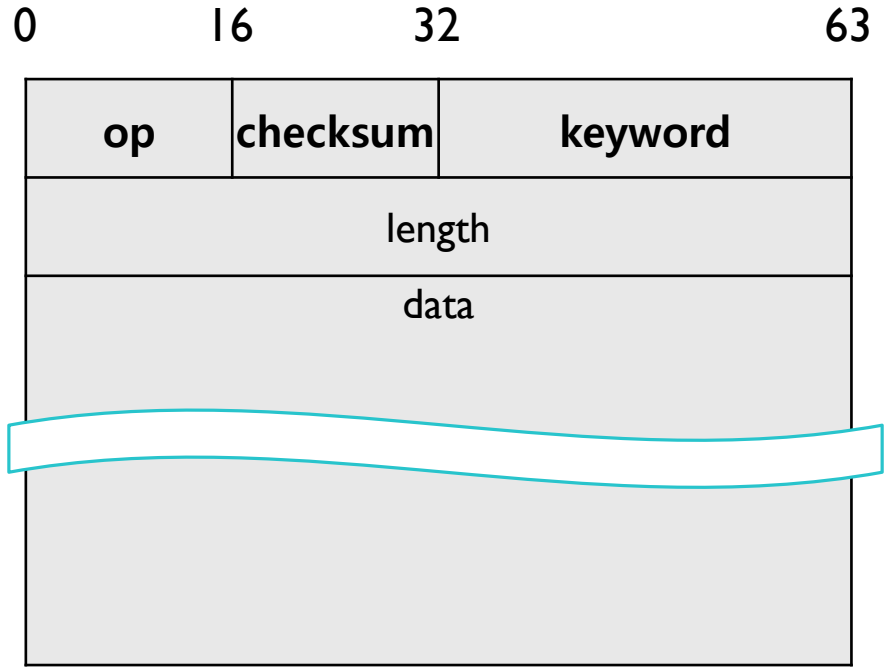
Example)
Keyword: cake
Plaintext: I love CS341! It's great!
Ciphertext: k lyzg cc341! mv's qvgad!

Keyword is always 4 characters long for this project.

Protocol Specification

- Protocol indicates the length of string by using a separate field.
- Maximum length of each message is limited to 10M

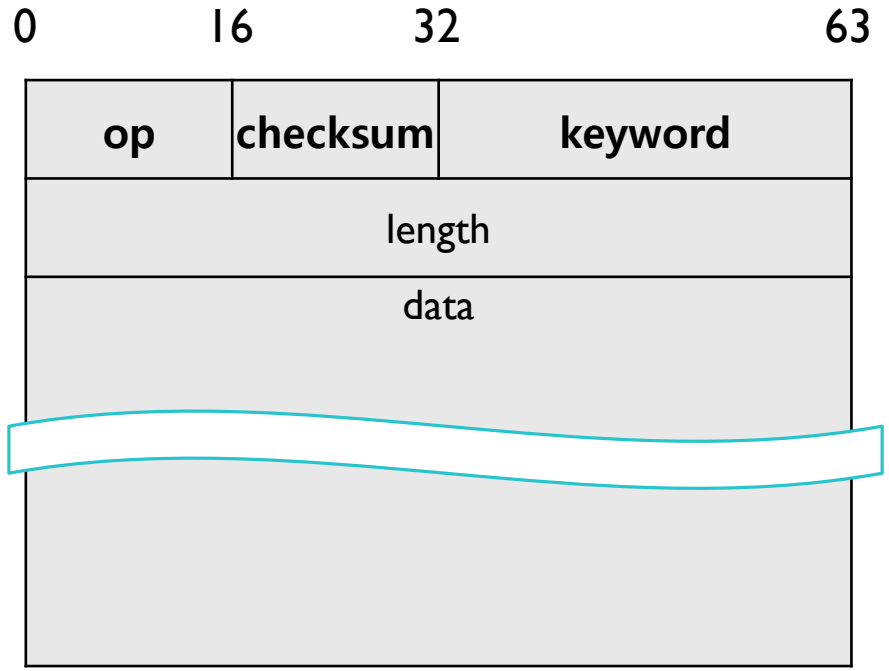
- **op** field
  - 16 bits, operation type, **network order**
  - 0: encrypt, 1: decrypt
- **checksum** field
  - 16 bits
  - Used for error-checking of protocol fields
  - Checksum is calculated in the same way as TCP checksum(1's complement)
- **keyword** field
  - 32 bits, 4 characters



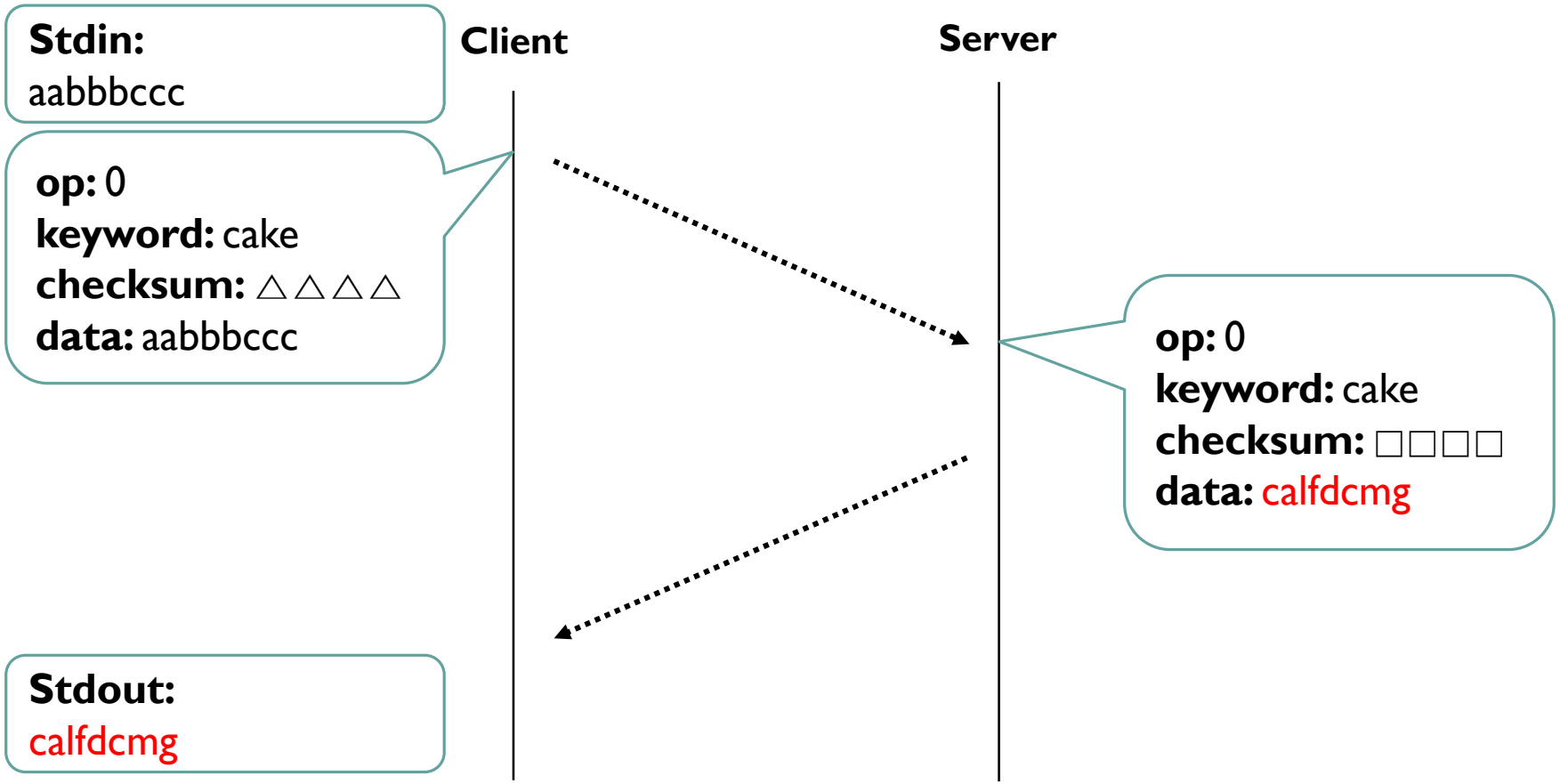
Protocol Specification

- Protocol indicates the length of string by using a separate field.
- Maximum length of each message is limited to 10M

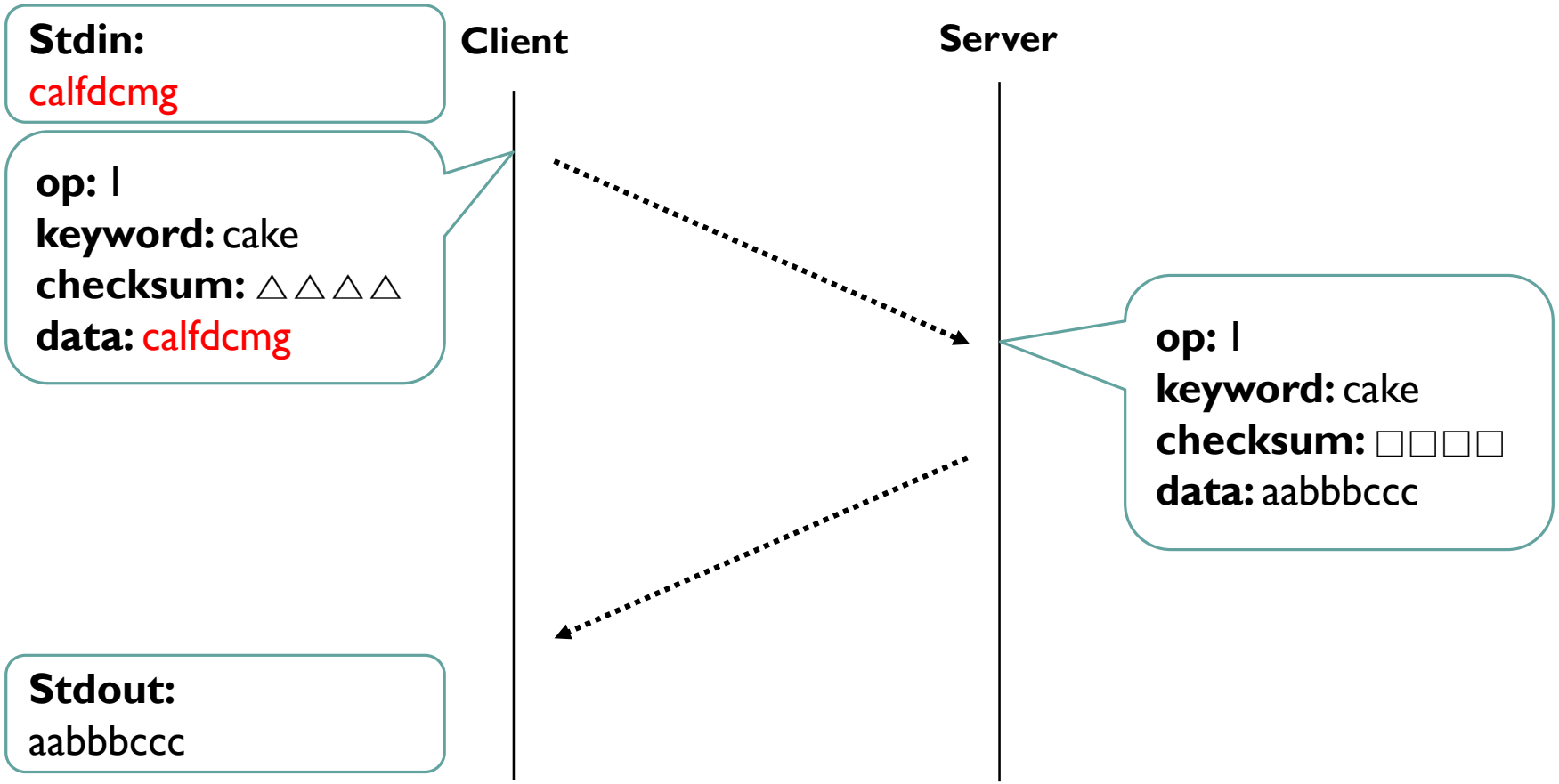
- **length** field
  - 64 bits, **network order**
  - In byte
  - Total length of a message
    - op, shift, checksum, length, data
    - Max 10MB (10,000,000 bytes)
- **data** field
  - String to be transmitted



Communication Example (Encryption)



Communication Example (Decryption)



Project #1 Part I (4pt)

Implement a client which communicates with the server based on the specification.

Basic functions of the client

- Client program should take as command-line parameters IP address and port to connect to.
  - ex: ./client –h 143.248.111.222 –p 1234 –o 0 –k cake
  - Host: 143.248.111.222, Port: 1234, Operation: encrypt, Keyword: cake
  - **You MUST follow this argument format, and also binary names**
- When a user inputs a string into stdin, client sends it to server. (Client should wrap each input using above protocols)
- Once the client receives a reply, the client prints it to stdout. **(Payload Only)**
- Must be able to handle binary data
- **Terminate when EOF is received at stdin**

Project #1 Part II (6pt + 2pt)

Implement a server which can communicate with your client you made in part I

Basic function of server

- Your server must handle **multiple** connections of clients.
- Server receives a string from client, and then server computes the string and returns it to the client.
- **Reject connections from protocol-violating clients**

Requirement

- Implement server that handles multiple clients simultaneously: **use fork()**
- Allow user to set port
  - ex: ./server –p 1234
  - **You MUST follow this argument format**

Extra credit (2pt)

- Implement a non-blocking server using select() instead of fork()
- The server must be single-threaded and serve multiple clients simultaneously
- Extra credit will be treated as a bonus point for overall project score

Testing for Project #1

The test server will be opened by next week.

- IP address, port and other information will be posted on KLMS!

We will also provide sample test inputs and results

- You could use them as test cases for the projects

Of course, when we are grading, other test cases will be used.

- Tests for multiple connections, text in different size, and other edge cases

Examples from last year

Encryption / Decryption testing

```
./client -h 127.0.0.1 -p 4000 -o 0 -k cake < test.txt > a.txt
./client -h 127.0.0.1 -p 4000 -o 1 -k cake < a.txt > b.txt
diff -i test.txt b.txt
```

Test server testing

```
./client -h 143.248.56.16 -p 4000 -o 0 -k cake < test.txt > a.txt
./client -h 127.0.0.1 -p 4000 -o 0 -k cake < test.txt > b.txt
diff a.txt b.txt
```

Requirements for the Assignments

Only in C programming language & Linux environment

Use C standard libraries & Linux system calls only

- No other 3rd party libraries

Data over the network MUST follow the network byte ordering (= Big-endian)

- Refer to the functions such as htons(), htonl(), ntohs(), ntohl()

Requirements for the Assignments

Provide a Makefile (make all) and follow the argument format
(Automated scripts will grade your program)

Ex)'$make all' generates executable files: 'client' and 'server'

Otherwise, we will deduct 2 points from your score

Strongly recommend to add **comments** on your code to avoid plagiarism

Deliverables

Report + Source code of your client & server

- Submit **both** fork() version and select() version if you did the extra assignment, otherwise you won't get extra credit.

1. client.c
2. server.c
3. server_select.c : only if you implemented the select() version
4. Makefile : *$make all* should generates executable files -'client', 'server' and 'server_select'
5. report.pdf

Above 5 materials in a single ZIP file:

- {StudentID}_{Name}_Proj1.zip
- Your name should be in English
- Ex) 20171111_HongGildong_Proj1.zip
- Other format will not be allowed

Deliverables

Content of report

- Provide the instruction to compile your programs and self-test results of your server & client
  - Again, you should write a Makefile to compile your programs.
- Explain the structure of your server & client

Format of the report

- **PDF** only, **12pt**, line-spacing **100%**
- At most **5** pages (expected around 3 pages)

Submission Guideline

This assignment is **individual**

Due: 9/20, 12:59:00pm (before lecture begins)

Report + source code in a single ZIP file:

- {StudentID}_{Name}_Proj1.zip
- Ex) 20121111_HongGildong_Proj1.zip
- Your name should be in English
- Other formats will get 0 points

Recommended Links

These links contain useful info and example codes

- Beej's Guide to Network Programming
  - http://beej.us/guide/bgnet/output/html/singlepage/bgnet.html

We recommend you to useVirtualBox or Vmware to establish a virtualized Linux environment on Windows

- VirtualBox
  - https://www.virtualbox.org/
- Vmware
  - https://my.vmware.com/en/web/vmware/free#desktop_end_user_computing/vmware_workstation_player/12_0

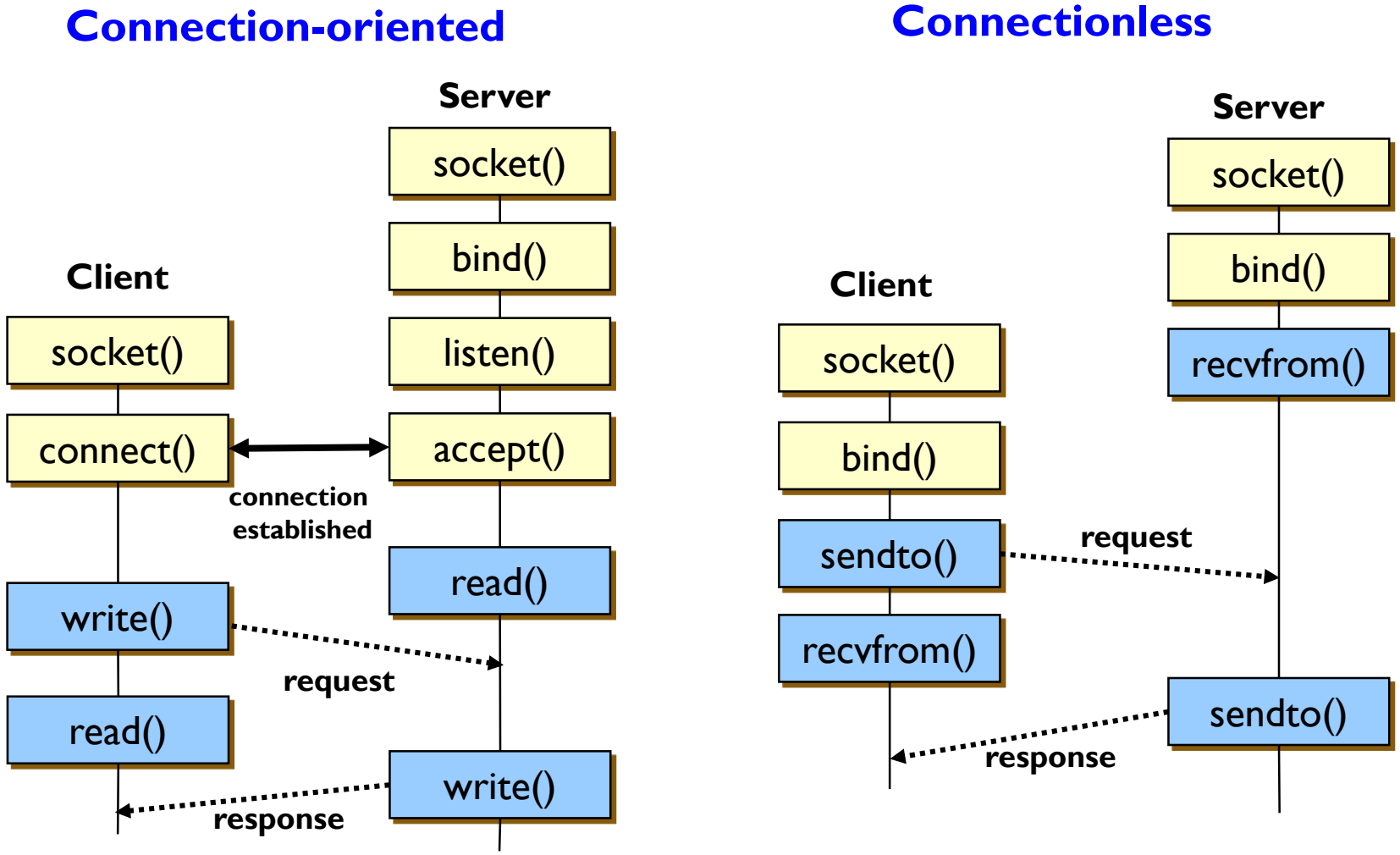
If you need, we can provide you with a linux environment. Please let us know if you need one.

Appendix: socket APIs

Socket

- An interface between application and network
- Unique identification to or from which information is transmitted in the network
- Clients and servers communicate with each other by reading from and writing to socket
- In UNIX-like systems, a socket descriptor is just another file descriptor

Two types of data tx using socket



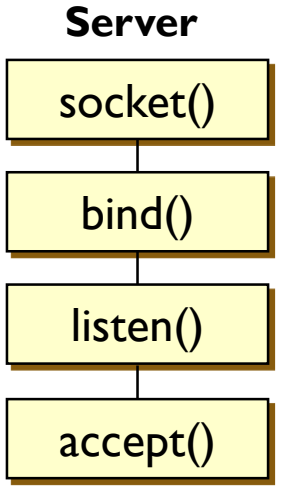
Server-side Basic Socket API

socket()
- Creates a new socket and returns its socket descriptor

bind()
- Associates a socket with a local port number and IP address

listen()
- Prepares a socket for incoming connections

accept()
- Accepts a received incoming attempt from client
- Creates a new socket associated with a new TCP connection



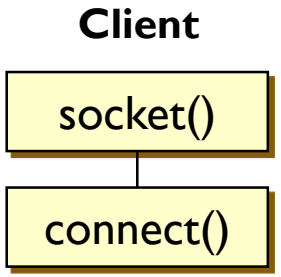
Client-side Basic Socket API

socket()

- Creates a new socket and returns its socket descriptor

connect()

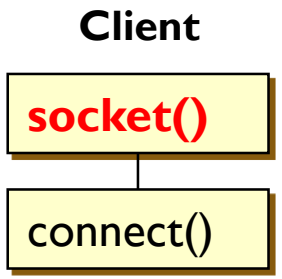
- Binds a destination to a socket or set a connection



Socket API: socket()

int socket (int family, int type, int protocol)

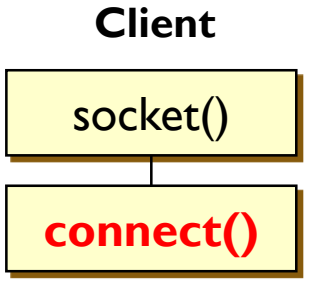
- socket() creates a socket descriptor.
- family specifies the protocol family.
  - AF_UNIX: Local Unix domain protocols
  - AF_INET: IPv4 Internet protocols
- type specifies the communication semantics.
  - SOCK_STREAM: provides sequenced, reliable, two-way, connection-based byte streams
  - SOCK_DGRAM: supports datagrams (connectionless, unreliable messages of a fixed maximum length)
  - SOCK_RAW: provides raw network protocol access
- protocol specifies a particular protocol to be used with the socket.



Socket API: connect()

int connect (int sockfd, const struct sockaddr *servaddr,

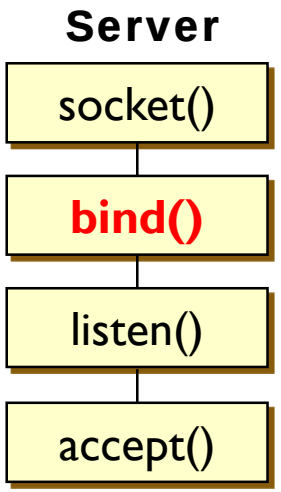
socklen_t addrlen)



- TCP client uses to establish a connection with a TCP server.
- servaddr contains *{IP address, port number}* of the server.
- The client does not have to call bind() before calling connect().
  - The kernel will choose both an ephemeral port and the source IP address if necessary.
- Client process suspends (blocks) until the connection is created.

Socket API: bind()

```
int bind ( int sockfd,
           struct sockaddr *myaddr,
           socklen_t addrlen  )
```



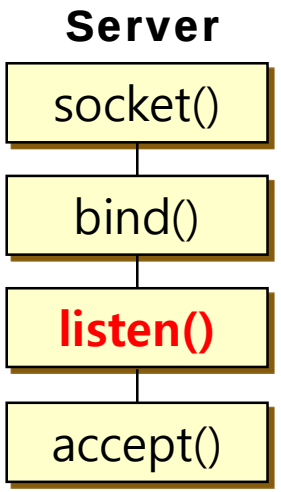
- bind() gives the socket sockfd the local address myaddr.
- myaddr is addrlen bytes long.
- Servers bind their well-known port when they start.
- If a TCP server binds a specific IP address to its socket, this restricts the socket to receive incoming client connections destined only to that IP address.
- Normally, a TCP client let the kernel choose an ephemeral port and a client IP address.

Socket API: listen()

int listen (int sockfd,

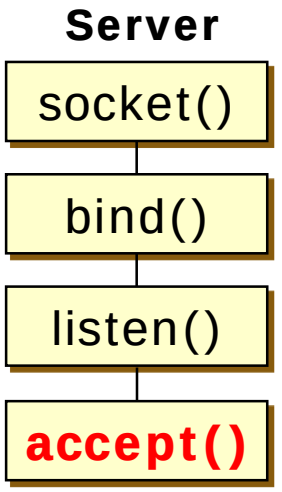
int backlog)

- listen() converts an unconnected socket into a *passive socket*, indicating that the kernel should accept incoming connection requests.
  - When a socket is created, it is assumed to be an *active socket*, that is, a client socket that will issue a connect().
- backlog specifies the maximum number of connections that the kernel should queue for this socket.
- Historically, a backlog of 5 was used, as that was the maximum value supported by 4.2BSD.
  - Busy HTTP servers must specify a much larger backlog, and newer kernels must support larger values.



Socket API: accept()

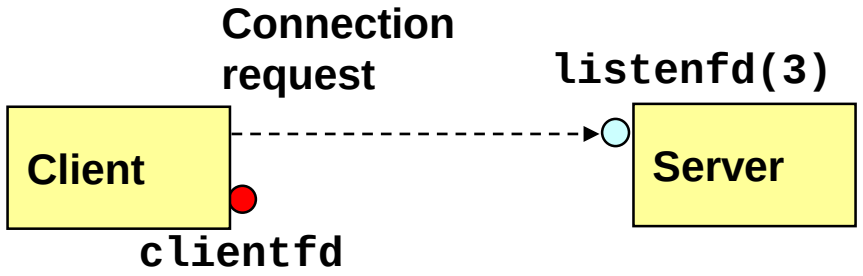
```
int accept (  int sockfd,
              struct sockaddr *cliaddr,
              socklen_t *addrlen  )
```



- accept() blocks waiting for a connection request.
- accept() returns a connected descriptor with the same properties as the listening descriptor.
  - The kernel creates one connected socket for each client connection that is accepted.
  - Returns when the connection between client and server is created and ready for I/O transfers.
  - All I/O with the client will be done via the connected socket.
- The cliaddr and addrlen arguments are used to return the address of the connected peer process (the client)

Socket API: accept()

1. Server blocks in `accept`, waiting for connection request on listening descriptor `listenfd`.



2. Client makes connection request by calling and blocking in `connect`.

3. Server returns `connfd` from `accept`. Client returns from `connect`. Connection is now established between `clientfd` and `connfd`.

Socket API: accept()

Listening descriptor

- End point for client connection requests
- Created once and exists for lifetime of the server

Connected descriptor

- End point of the connection between client and server
- A new descriptor is created each time the server accepts a connection request from a client.
- Exists only as long as it takes to provide services for client.

Why the distinction?

- Allows for concurrent servers that can communicate over many client connections simultaneously.