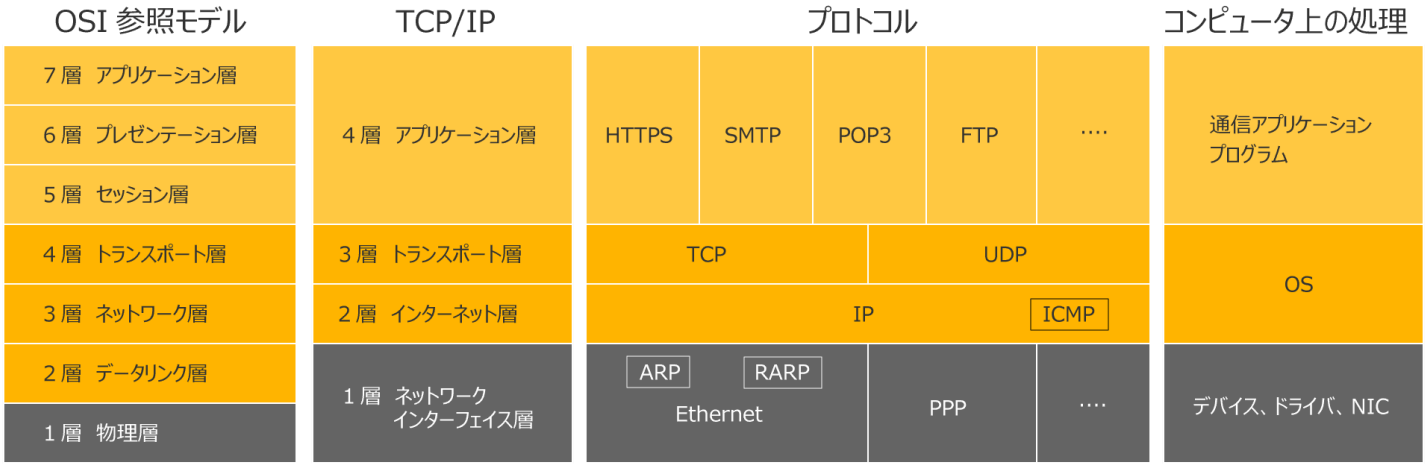
ネットワークプログラミング 1

OSI参照モデルとTCP/IP階層モデル

- OSI参照モデルとは何かを確認・調査してください。
- TCP/IP階層モデルは、どのような構成になっているか確認・調査してください。
- OSI参照モデルとTCP/IP階層モデルの関係を確認・調査してください。

> https://www.itmanage.co.jp/column/osi-reference-model/



簡単な説明

> https://thinkit.co.jp/article/20787

TCP/IPの仕様書

RFC(Request For Comments)とは？

- IETFで議論された内容が標準化されRFCというドキュメントになり、公開される

  > https://www.rfc-editor.org/rfc/

  ここに、すべてのRFCが公開されている。
- RFCにはすべて番号が付けられている。
  - IPに関する仕様：RFC791　https://www.rfc-editor.org/rfc/rfc791.txt
    - STD5
  - UDPに関する仕様：RFC768　https://www.rfc-editor.org/rfc/rfc768.txt
    - STD6
  - TCPに関する仕様：RFC793　https://www.rfc-editor.org/rfc/rfc793.txt
    - STD7

IPv4ヘッダ

RFC791より

```
3.1.  Internet Header Format

  A summary of the contents of the internet header follows:


    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |Version|  IHL  |Type of Service|          Total Length         |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |         Identification        |Flags|      Fragment Offset    |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |  Time to Live |    Protocol   |         Header Checksum       |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                       Source Address                          |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                    Destination Address                        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                    Options                    |    Padding    |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

```
                    Example Internet Datagram Header
```

UDPヘッダ

RFC768より

```
Format
------

                 0      7 8     15 16    23 24    31
                +--------+--------+--------+--------+
                |     Source      |   Destination   |
                |      Port       |      Port       |
                +--------+--------+--------+--------+
                |                 |                 |
                |     Length      |    Checksum     |
                +--------+--------+--------+--------+
                |
                |          data octets ...
                +---------------- ...

                     User Datagram Header Format
```

TCPヘッダ

RFC793より

```
   A TCP header, followed by any user data in the segment, is formatted
   as follows, using the style from [66]:

    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |          Source Port          |       Destination Port        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                        Sequence Number                        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                    Acknowledgment Number                      |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |  Data |       |C|E|U|A|P|R|S|F|                               |
   | Offset| Rsrvd |W|C|R|C|S|S|Y|I|            Window             |
   |       |       |R|E|G|K|H|T|N|N|                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |           Checksum            |         Urgent Pointer        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                           [Options]                           |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               :
```

```
:                             Data                              :
:                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

        Note that one tick mark represents one bit position.
```

- 各フィールドの働き、意味は、RFCもしくは解説ドキュメントをよく読み理解する必要がある。
  - お薦め本

  - マスタリングTCP/IP—入門編—　オーム社
    - 井上 直也 , 村山 公保 , 竹下 隆史 , 荒井 透 , 苅田 幸雄

ソケット通信

- 上記のパケットを自前で構築し、送受信を行うのは、理解していれば難しくはないが面倒。（raw socketという）
  - 例としていくつか紹介する。

> Wikipedia - Raw_socket
>
> https://www.geekpage.jp/programming/linux-network/simple-ping.php
>
> https://53ningen.com/ping-impl1

- そこで、socket APIを使用して、送受信を行う（標準socket通信という）

    - 送信するペイロードは選択したトランスポート層のプロトコル（例: TCP、UDP）によってカプセル化される。

- socket通信の流れを、まずはPython版で理解してみよう

> https://envader.plus/article/27

【server_tcp.py】

```
import socket

# socket.socket(): ソケットオブジェクトの作成
# AF_INET: アドレスファミリ（使うアドレスの種類がまとめられたもの）のうち、IPv4を指定する
# SOCK_STREAM: TCPプロトコルを指定
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# socket.bind(): IPアドレスとポート番号を作成したソケットオブジェクトに紐づける
# 127.0.0.1: ローカルホストを指定する
# 60001: ポート番号のうち60001を指定する（待ち受けポート）
s.bind(('127.0.0.1', 60001))

# socket.listen(): クライアントからの入力待ち状態になる
# 1: 並列的に処理できるリクエスト数を1つに指定する
s.listen(1)

count = 0
while count < 5:
    # socket.accept(): クライアントからの接続を受け付ける
    # conn: 新しく作成したソケットオブジェクト
    # addr: 受信したIPアドレス
    conn, addr = s.accept()

    # サーバの標準出力に文字列を出す
    # addr: (クライアントのIPアドレス, クライアントのポート番号)
    print(f'Source IP Address: {addr}')

    # クライアントのソケットにデータを送信する
    conn.send(b'Hello World!')

    # クライアントのソケットオブジェクトを削除する
    conn.close()
    count += 1

# サーバのソケットを削除する
s.close()
```

server版

- 上記Python版をCで書き換えた例です。
    - クライアント版（後述）と一緒に使用しないと動作検証できません。

```c
// 【図解あり】ソケット通信の仕組みについてわかりやすく解説(python)
// https://envader.plus/article/27
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>

// 追加
// inet_ntoa() / inet_addr() は arpa/inet.h
#include <arpa/inet.h>

// close()
#include <unistd.h>

#define SERVER_ADDR "127.0.0.1"
#define SERVER_PORT 60001
#define BUF_SIZE 4096

int main(int argc, char const *argv[]) {
    // ソケットオブジェクトの作成
    int sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0) {
        perror("socket");
        exit(1);
    }

    // IPアドレスとポート番号を作成したソケットオブジェクトに紐づける
    struct sockaddr_in addr;
    memset(&addr, 0, sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_port = htons(SERVER_PORT);
    addr.sin_addr.s_addr = inet_addr(SERVER_ADDR);
    if (bind(sockfd, (struct sockaddr *)&addr, sizeof(addr)) < 0) {
        perror("bind");
        exit(1);
    }

    // クライアントからの入力待ち状態になる
    if (listen(sockfd, 1) < 0) {
        perror("listen");
        exit(1);
    }
```

```
    int count = 0;
    while (count < 5) {

        // クライアントからの接続を受け付ける
        int connfd;
        struct sockaddr_in client_addr;
        socklen_t client_addr_len = sizeof(client_addr);
        if ((connfd = accept(sockfd, (struct sockaddr *)&client_addr, &client_addr_len))
< 0) {
            perror("accept");
            exit(1);
        }

        // サーバの標準出力に文字列を出す
        char buffer[BUF_SIZE];
        snprintf(buffer, sizeof(buffer), "Source IP Address: %s",
inet_ntoa(client_addr.sin_addr));
        printf("%s\n", buffer);

        // クライアントのソケットにデータを送信する
        char message[] = "Hello World!";
        send(connfd, message, strlen(message), 0);

        // クライアントのソケットオブジェクトを削除する
        close(connfd);
        count++;
    }
    // サーバのソケットを削除する
    close(sockfd);

    return 0;
}
```

- 見慣れない関数は、調べて使い方を確認してください。
    - これまでに説明していないものを上げておきます。

関数名	引数や戻り値	働き・仕様など詳細
`socket( )`		
`perror( )`		
`memset( )`		
`htons( )`		
`inet_addr( )`		
`bind( )`		
`listen( )`		
`accept( )`		
`snprintf( )`		
`inet_ntoa( )`		
`send( )`		

- 構造体に関して調べてください

構造体	メンバ変数	値など備考
`struct sockaddr_in`		

- 使用しているヘッダファイルについて調べてまとめてください。

header file	主に利用される目的・定義・関数など
`netinet/in.h`	
`sys/socket.h`	
`arpa/inet.h`	
`unistd.h`	

- これらのファイルは、各マシンの以下のフォルダに保存されている。確認してください。
    - `/usr/include/`

headerファイル内の定義

- C言語で、よく使用される記述
    - `#define`
        - 単なる置換
        - マクロ

    - `#include <hoge.h>`
    - `#if` 〜 `#endif`
    - `#ifdef`
    - `#ifndef`
    - `enum`
- それぞれが、どのように働くのか、調べてください。
    - プリプロセッサの働き

headerファイルを読み解く

- 上記、`struct socket_in` は、`/usr/include/netinet/in.h` に記述されている。
    - 実際に確認してみよう
    - include 以下を コピーするか、直接 vscodeで開き確認する。
        - vscodeでは、ソースファイルから順に辿れるようになっている。（右クリックメニュー）

        - 244行目あたりに記載されている

```
/* Structure describing an Internet socket address.  */
struct sockaddr_in
  {
    __SOCKADDR_COMMON (sin_);
    in_port_t sin_port;      /* Port number.  */
    struct in_addr sin_addr;    /* Internet address.  */

    /* Pad to size of `struct sockaddr'.  */
    unsigned char sin_zero[sizeof (struct sockaddr)
          - __SOCKADDR_COMMON_SIZE
          - sizeof (in_port_t)
          - sizeof (struct in_addr)];
  };
```

- `__SOCKADDR_COMMON (sin_);`
  - マクロの呼び出し
  - `/usr/include/x86_64-linux-gnu/bits/sockaddr.h`（34行目）に定義されている

```
#define __SOCKADDR_COMMON(sa_prefix) \
  sa_family_t sa_prefix##family
```

  - この部分は以下のようになる

```
sa_family_t sin_family
```

    - `sa_family_t` 型とは何か？ 同ファイル（27行目）に定義されている

```
/* POSIX.1g specifies this type name for the `sa_family' member.  */
typedef unsigned short int sa_family_t;
```

  - 最終的には、次のようになる

```
__SOCKADDR_COMMON (sin_);
  ↓
unsigned short int sin_family;
```

- `in_port_t sin_port`
  - `in_port_t` 型とは？ `in.h`（122行目）に定義がある

```
/* Type to represent a port.  */
typedef uint16_t in_port_t;
```

  - `uint16_t` 型とは？ `/usr/include/x86_64-linux-gnu/bits/stdint-uintn.h`（25行目）

```
typedef __uint16_t uint16_t;
```

- `__uint16_t;` 型とは？ `/usr/include/x86_64-linux-gnu/bits/types.h` （40行目）

```
typedef unsigned short int __uint16_t;
```

- 最終的には、次のようになる

```
in_port_t sin_port;
   ↓
unsigned short int sin_port;
```

- これを繰り返し確認していく

```
/* Structure describing an Internet socket address.  */
struct sockaddr_in
  {
    unsigned short int sin_family;
    unsigned short int sin_port;      /* Port number.  */
    ⋮
};
```

↓

- 一般的には以下のように考える

```
struct sockaddr_in {
  sa_family_t sin_family;
  in_port_t sin_port;
  struct in_addr sin_addr;
  char sin_zero[8];
};
```

- `sin_family` フィールドは、アドレスファミリを表す。IPv4 の場合は `AF_INET` が使用さる
- `sin_port` フィールドは、ポート番号を表す。
- `sin_addr` フィールドは、IP アドレスを表す。
- `sin_zero` フィールドは、未使用。

ソケット通信2

- socket通信の流れを、Python版で理解してみよう

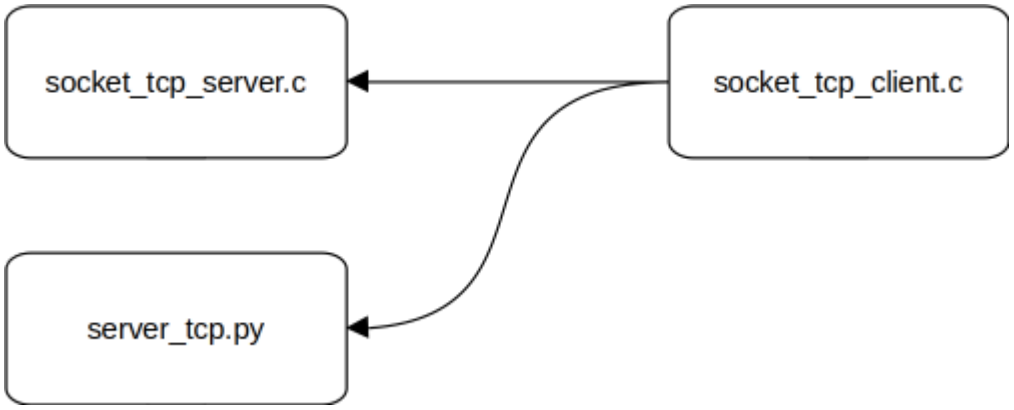
> https://envader.plus/article/27

【client_tcp.py】

```
import socket

# ホストのソケットオブジェクトの作成
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# サーバにリクエストを送る
# 127.0.0.1: サーバのIPアドレス
# 60001: サーバが待ち受けするポートを指定する
s.connect(('127.0.0.1', 60001))

# サーバからデータを受信する
data = s.recv(4096)

# クライアントの標準出力に受信したデータを表示する
print(data)

# クライアントのソケットを削除する
s.close()
```

client版

- 前回のServer版を参考に、Client（C版）を作成してください。
  - 前回作成した `socket_tcp_server.c` および `server_tcp.py` と通信ができるのを確認してください。



調査事項

- python版client（`tcp_client.py`）を使用した場合、出力が次のようになる。

```
$ python3 client_tcp.py
b'Hello World!'
```

    - この`b'Hello World!'`とは何を意味しているか？