ネットワークプログラミング第6回（ネットワークとプログラミング（1））

２００９春学期　中村　修

授業Webページ

- SFC-SFS
  https://vu2.sfc.keio.ac.jp/sfc-sfs/

- 課題・授業資料などの情報を掲示します
  - **課題は毎回こちらに提出してください**！！
  - 今日の課題締め切り
    - 6/1(月)23:59まで！
    - 遅れて提出する方は要連絡

今期の授業スケジュール（予定）

- 第１回：イントロダクション (4/14)
- 第２回：C言語の基礎～関数・ポインタ (4/21)
- 第３回：C言語の基礎～コマンドライン引数 (4/28)
- 第４回：C言語の基礎～構造体・リスト構造 (5/12)
- 第５回：FileI/Oとシステムコール (5/19)
- **第６回：ネットワークとプログラミング（1） (5/26)**
- 第７回：ネットワークとプログラミング（2） (6/2)
- 第８回：ネットワークプログラミング実践（1） (6/9)
- 第９回：応用ネットワークプログラミング（１） (6/16)
- 第１０回：応用ネットワークプログラミング（２） (6/23)
- 第１１回以降：ミニプロ (6/30, 7/7, 7/14)

第5回課題

- コマンド引数で2つのファイル名を与え、その1つ目の引数のファイルを開き、2つ目の引数のファイルへコピーするプログラムを作成せよ.

実行例

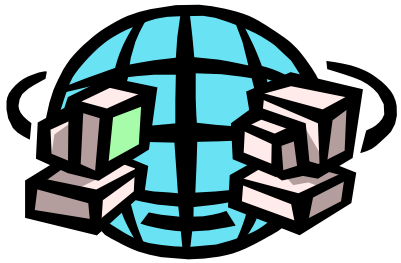
```
% cat src.txt
1,2,3
this is a pen.
dada
% ./a.out src.txt dst.txt
% cat dst.txt
1,2,3
this is a pen.
dada
```

```c
#include <stdio.h>
#include <fcntl.h>
#include <stdlib.h>

int main(int argc, char *argv[]){
   int fdin, fdout;
   char buf[512];
   int cnt;

   if(argc != 3){
       fprintf(stderr, "%s [input
file] [output file]¥n", argv[0]);
       exit(-1);
   }
   fdin = open (argv[1], O_RDONLY, 0);
   if(fdin < 0){
       perror("open fdin");
       exit(-1);
   }
   fdout = open (argv[2], O_WRONLY | O_CREAT |
O_EXCL, 0644);
   if(fdout < 0){
       perror("open fdout");
       exit(-1);
   }

 while ((cnt = read(fdin, buf, sizeof(buf))) >
0) {
   if (write (fdout, buf, cnt) < 0) {
       perror ("write error");
       exit (-1);
   }
 }
 if (cnt < 0) {
   perror ("read error");
   exit (-1);
 }

 close(fdin);
 close(fdout);
 return;
}
```

今日のお題

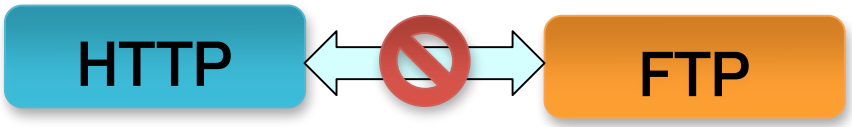
- 講義
  - 7　layer modelのおさらい
  - ネットワークプログラミング基本手順
  - IPv6のみを扱います
- 実習/課題：データの送受信
  - UDP echo client 作成

コンピュータ・プロトコル



- 通信の手順をきめた約束事
- 通信規約と訳される
- 例:
  IP、HTTP、TCP、FTP、UDP、ICMP、etc…

- それぞれ対応しているプロトコルが違うとお互いに通信できない



同じ決まりで通信するということ

- 相手が英語，自分が日本語で話しても会話は成立しない
- 電話では始めに「もしもし，xxです」

- コンピュータの世界でも同じ
  - UNIX−Windows間の通信
  - 無線−有線間の通信

1） 受話器を上げる

2） 電話番号を入力

5） 会話する

6） どちらかが受話器を置く

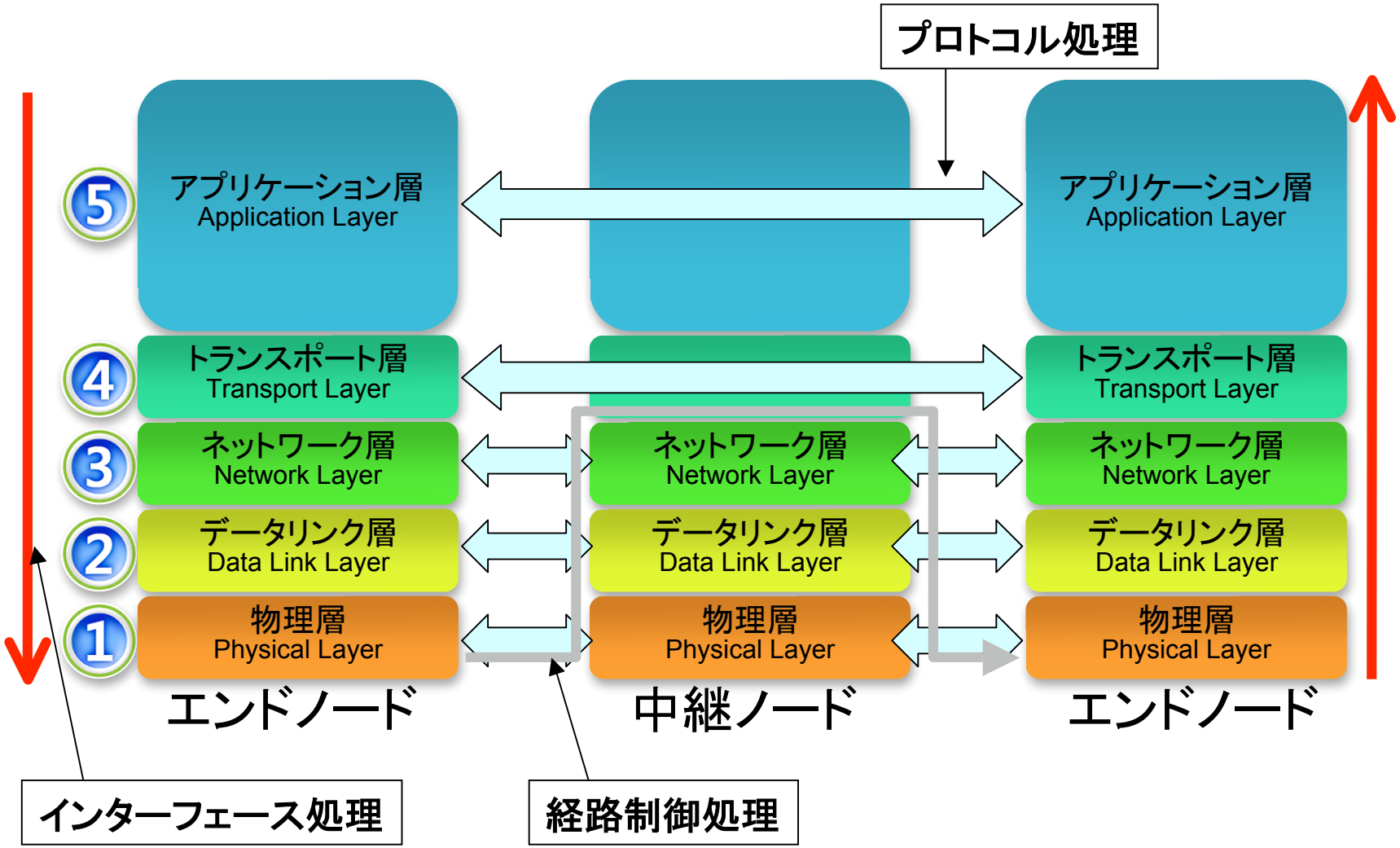
7） 回線が切れる

3） 着信を通知する

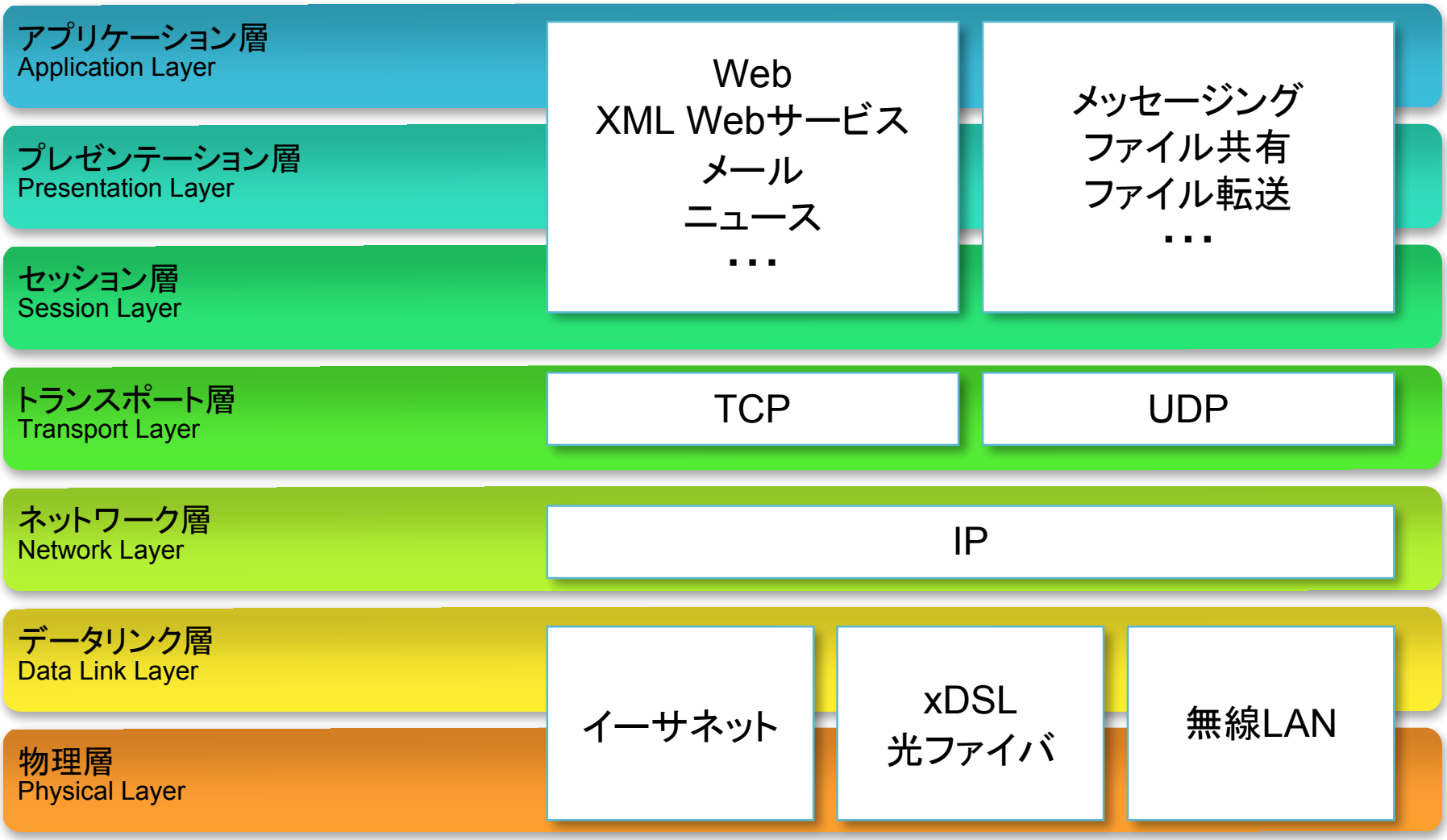
4） 受話器を上げる

電話の例

インターネットの階層モデル

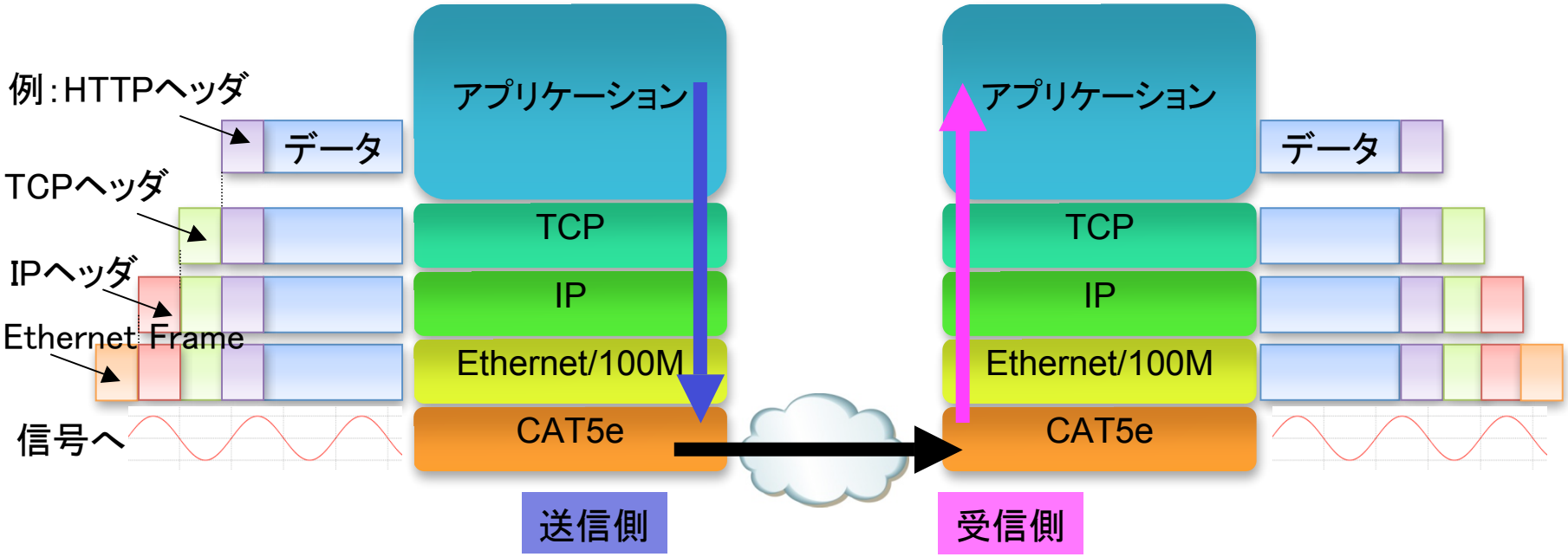


OSIモデルとインターネットアーキテクチャ



プロトコルスタックとカプセル化

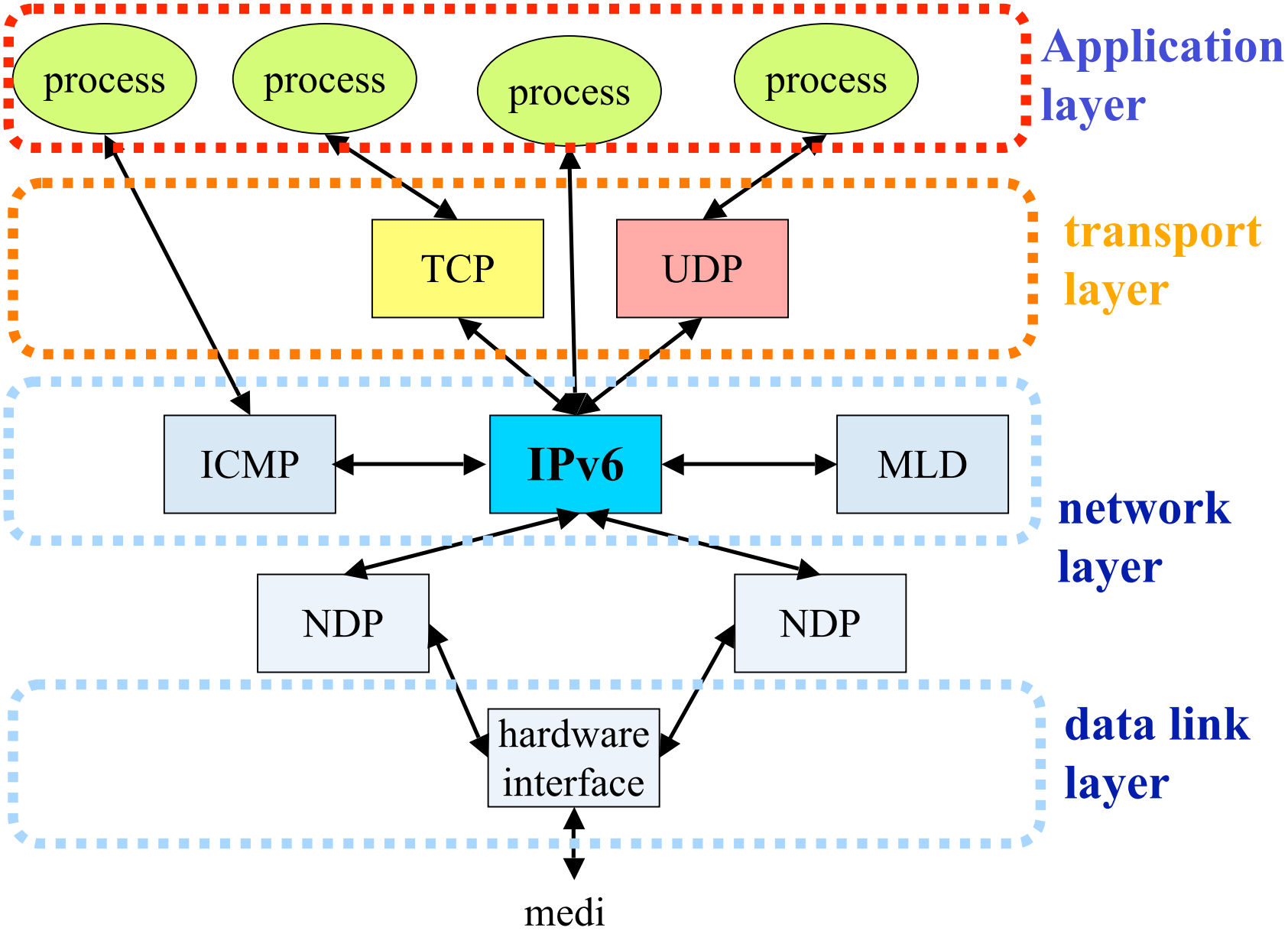
- **送信側**
  - 各層がそれぞれ必要な情報（ヘッダ）を付加して下層へ渡す
- **受信側**
  - 各層はヘッダの情報をもとに処理を行い，そのヘッダ部分を取り除いて上層へ渡す



階層化の特徴

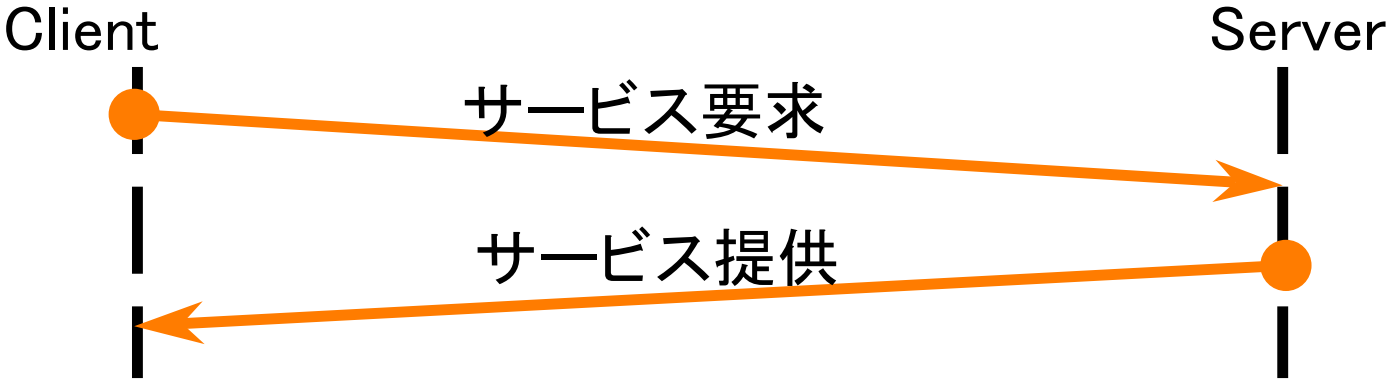
- 責任範囲(役割)の規定，限定
- 各階層が独立
  - 上位のプロトコルは，自分のすぐ下のプロトコルの使い方(インターフェース)さえ知っていれば，それより下で何が起きているかをまったく気にする必要がない
  - 階層毎に共通のインターフェイスを定義
- スケーラビリティ
  - 処理を各層に分散できる
- 新しい技術への柔軟性
  - 同一レベルの階層同士を交換出来る
    - 新しく技術が開発された部分だけ交換すれば進化できる
  - ISDN→ADSL→FTTH，IPv4→IPv6
- 階層化していないと・・・
  - 規格が変わるとシステムすべてを更新する必要がある

ネットワークアプリケーションとは？



クライアント・サーバモデル

- ネットワークを介したサービスにおける通信モデル
- サーバ
  - 受動的にサービス提供する側、待っててくれる
  - 反復サーバ/平行サーバ
- クライアント
  - 能動的にサービス提供を促す側、接続しに行く

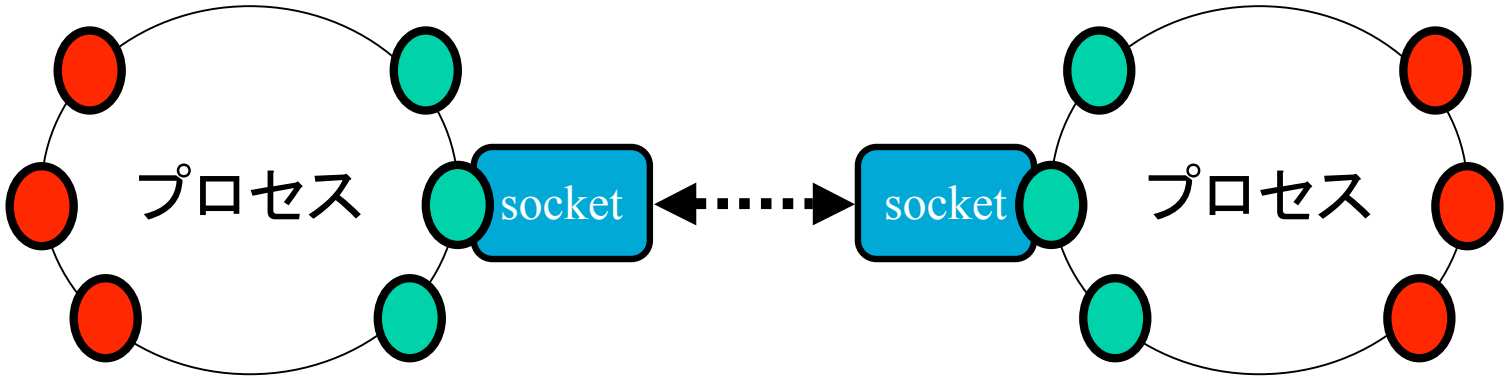


ポートとソケット

- ポート
  - トランスポート層のアクセスポイント
  - TCP/UDP毎に持っている
- ソケット
  - プロセスとポートを繋ぐアダプタ

ソケット(Socket)

- プロセス間通信を行う為のデータの出入り口
- プロセスからはファイルディスクプリタを用いてアクセス
- <u>プロセスにとってはプロセス間通信もファイル入出力も同じインターフェイス</u>



socket()システムコール

- int socket(int family, int type, int proto)
  - familyにはプロトコルファミリを指定
    - AF_INET　IPv4プロトコル
    - AF_INET6　IPv6プロトコル
    - AF_LOCAL　UNIX Domain Socket
    - AF_ROUTE　経路制御ソケット
  - Typeにはソケットのタイプ（以下のどれか）
    - SOCK_STREAM　ストリームソケット
    - SOCK_DGRAM　データグラムソケット
    - SOCK_RAW　rawソケット
  - Protoにはrawソケット以外、通常0

socket()システムコール

- 返り値
  - 成功: ソケットディスクリプタが返る
  - 失敗: −1が返る
    - ソケットディスクリプタはファイルディスクリプタの友達
- 実際のコードでは…

  listenfd = socket(AF_INET6, SOCK_STREAM, 0)

- AF_INET6の場合の利用されるIPv6の上位層
  - SOCK_STREAM　　TCP
  - SOCK_DGRAM　　UDP
  - SOCK_RAW　　ICMPv6, OSPFv3, etc

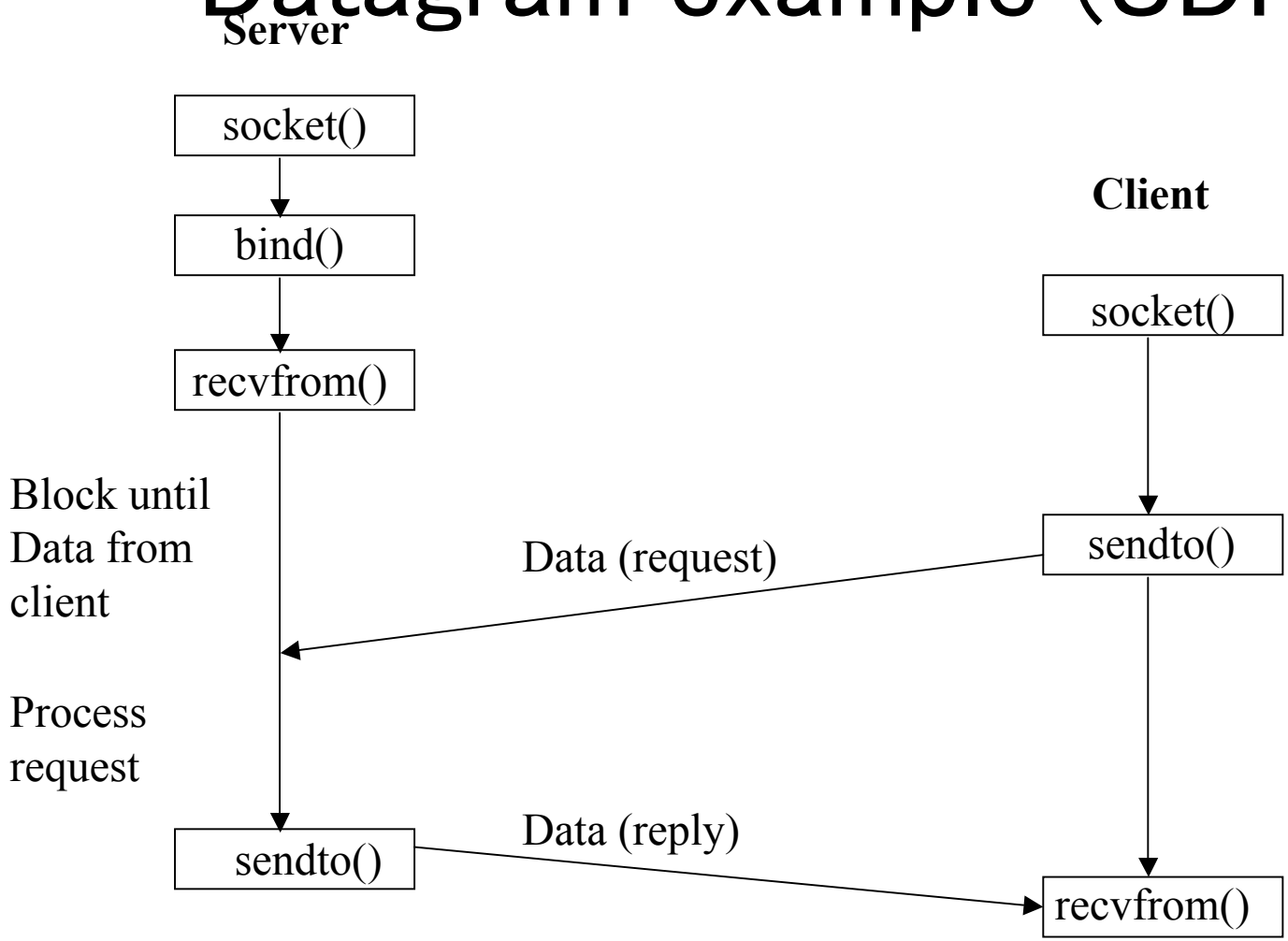
sendto()システムコール

- ssize_t sendto(int socket,
  const void *buffer,
  size_t length,
  int flags,
  const struct sockaddr *dest_addr,
  socklen_t dest_len);
- 開いたsocketに対して、データを送信する
- あて先として、struct sockaddr形式でアドレスを指定

recvfrom()システムコール

- ssize_t recvfrom(int socket,
  void *restrict buffer,
  size_t length,
  int flags,
  struct sockaddr *restrict address,
  socklen_t *restrict address_len);
- 開いたsocketに対して送信してきたデータを受信
- 送信元を知るには、address引数に通知されるsockaddrを読み出し

Datagram example (UDP)

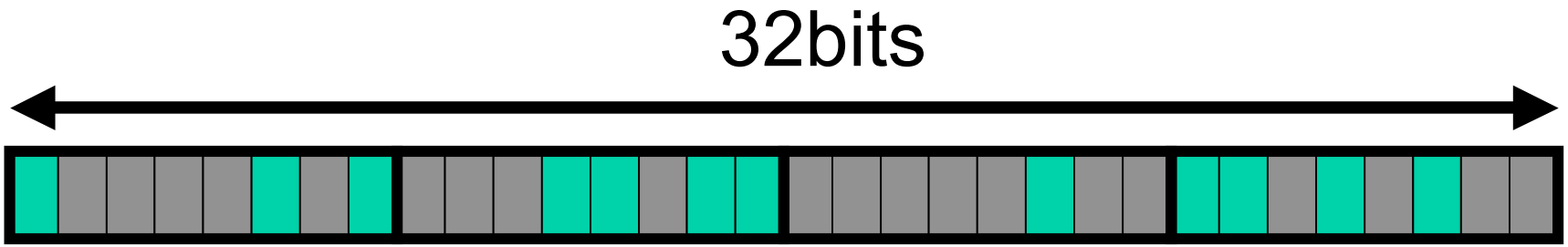


IPv4アドレスとIPv6アドレス

- IPv4アドレス32bit
- IPv6アドレス128bit

IPv4アドレス

- アドレスの数は2の32乗(約42億)
  - 世界の人口の数より少ない(約64億)
  - アドレスはコンピュータだけ!?
- IPアドレスの表記



133. 27. 4. 212

IPv6アドレス

- アドレスの数は2の128乗（約340澗(カン)）
  (340,282,366,920,938,463,463,374,607,431,768,211,456)
  - 1人あたり5垓(ガイ)個
  - アドレスはコンピュータ以外のものにも使える
- IPアドレスの表記
  - 128bit を 16進数で表す
  - 4桁(16bit)ごとに「：（コロン）」で区切る

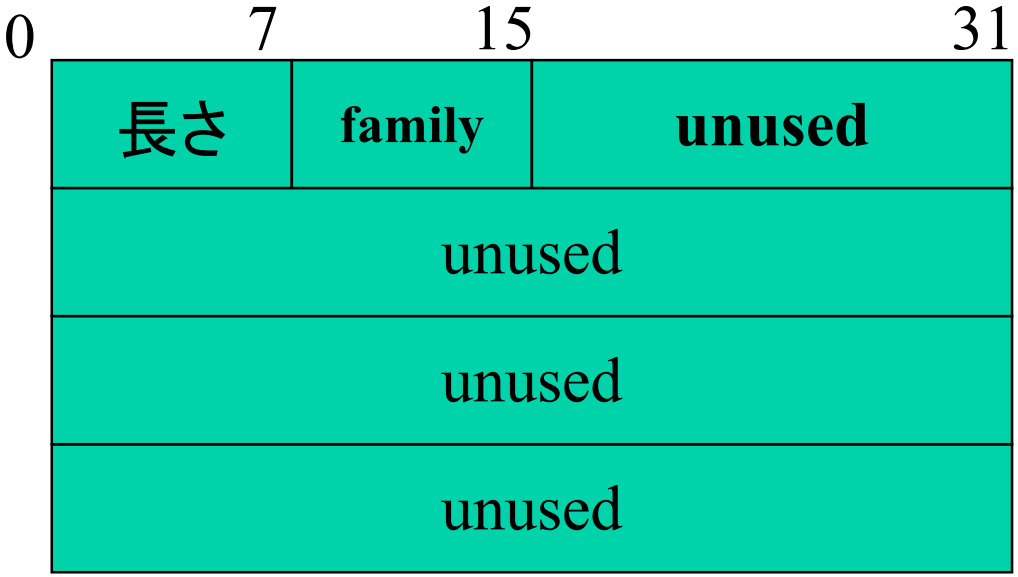
2001:200:1c0:1100:21a:64ff:fe8e:2e40

sockaddr_in6構造体

- ソケットの情報
  - アドレス … 128bit
  - ポート番号 … 16bit
  - プロトコルファミリー
    - AF_INET6

sockaddr構造体

- ソケットの情報を一般化した形
  - ソケットを使った通信のためのテンプレート
  - 利用するプロトコルに依存しない
  - 共通： 長さ・プロトコルファミリ(AF_XXX)



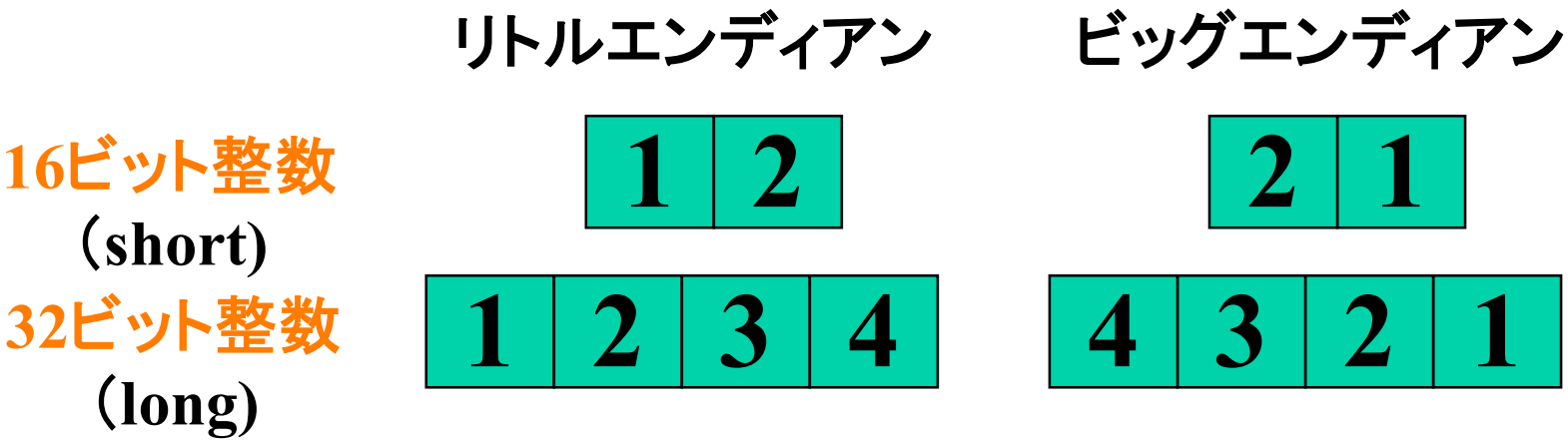
キャスト

- ある変数・構造体を無理やり違う型の変数や構造体として扱う方法
  - 変数を使う時に扱いたい型をカッコで括る
    - (int)no_int_variable; ← int型にキャスト
  - 関数の引数を一般化するのに便利
    - sockaddrの例
    - struct sockaddr_in6 sin6; (struct sockaddr)sin6;
  - 型やサイズに依存せず1バイトずつ読みたいときにも使う

```
long addr = 1234567;  char *cp = (char *)&addr;
for(j = 0; j < 4; j++) {
        printf("%c ", *cp++);
}
```

ネットワーク・バイト・オーダ

- Network Byte Order
- CPUアーキテクチャによって、バイトの並びが違う
  - 一般にBig Endian(sparc等)とLittle Endian(Intel等)の二つ
- ネットワーク上に流すバイト順を統一しなくてはならない
  - Big Endianに統一
- htons()/htonl()/ntohs()/ntohl()を利用



エンディアン変換

- u_long htonl(u_long hostlong);
- u_short htons(u_short hostshort);
- u_long ntohl(u_long netlong);
- u_short ntohs(u_short netshort);

inet_ntop()

- const char * **inet_ntop(int af, const void * restrict src, char * restrict dst, socklen_t size);**
- ネットワークバイト順序のバイナリ値を、アドレスを表す文字列に変換
- 「2001:200:1c0:1100:21a:64ff:fe8e:2e40 」という文字列は人間には分かりやすいが，コンピュータには分かりにくい
- 返り値は，指定したアドレスが正当ならばその文字列，エラー発生時ならNULL.

inet_ntop() 使用例

```
char buf[64];
structn in6_addr addr;

printf (“%s ¥n”,
 inet_ntop (AF_INET6, &addr, buf, sizeof (buf)));
```

IPv6、IPv4両方に対応するには

- 以下の構造体のメンバの情報を
  - struct addrinfo hints, *ai ;
- 以下の関数を用いてあつめ
  - getaddrinfo(hostname, port, &hints, &ai));
- 以下のように使う
  - socket(ai->ai_family, ai->ai_socktype, ai->ai_protocol);
  - connect(socketfd, ai->ai_addr, ai->ai_addrlen) ;

getaddrinfo

- **getaddrinfo(const char *nodename,**
  const char *servname,
  const struct addrinfo *hints,
  struct addrinfo **res);
- nodename=ホスト名, servname=ポート, hintsは情報のタイプに関するヒント, resは結果のaddrinfoのリストが格納される
- 返り値:成功すると 0 を返し、失敗すると以下の非 0 のエラーコードのいずれかを返す。
- サーバの場合は
  - nodenameはNULL
  - hintsにAI_PASSIVEフラグを設定する

addrinfo構造体

・ <netdb.h>

```
struct addrinfo {
    int      ai_flags;        /* AI_PASSIVE, AI_CANONNAME */
    int      ai_family;       /* PF_xxx */
    int      ai_socktype;     /* SOCK_xxx */
    int      ai_protocol;     /* 0 or IPPROTO_xxx for IPv4 and IPv6 */
    size_t   ai_addrlen;      /* length of ai_addr */
    char     *ai_canonname;   /* canonical name for hostname */
    struct   sockaddr *ai_addr;      /* binary address */
    struct   addrinfo *ai_next;      /* next structure in linked list */
};
```

getaddrinfoの取り扱い方

- 複数のアドレスが戻ってくる（RFC3493）
  - 帰って来る順番はDNSの実装に依存
  - 帰ってきたそれぞれに対してconnectしてみる

```
for(ai = ai_save; ai != NULL; ai->ai_next){
   sockfd = socket(ai->ai_family, ai->ai_socktype,ai->ai_protocol);
   if(sockfd < 0){continue;}
   if(connect(sockfd, ai->ai_addr, ai->ai_addrlen)){
      break;
   }
}
```

freeaddrinfo

- void freeaddrinfo(struct addrinfo *ai)
- getaddrinfoで動的に割り当てられた領域を開放する

gaistrerror

- char * gai_strerror(int ecode);
- getaddrinfoの返り値のエラーコードを人間に可読な文字列に変換する

実習/課題:UDP-echoクライアント作成

- echoクライアントとは？
  - あるサーバと通信し、送信したメッセージと同じものが受信される(エコーされる)プログラム
- echoサーバは以下
  - IP address: **long.sfc.wide.ad.jp**
  - Port : **5525**
- 第1引数にIPアドレス，第2引数にポート番号を指定しよう
  - **./a.out long.sfc.wide.ad.jp　5525**
- 結果は以下のWebページにて、参照可能
  - http://long.sfc.wide.ad.jp/~tatsurou/09a-npro/

プログラミング環境

- CNS環境(推奨)
  - Linux:ccx00,ccx01

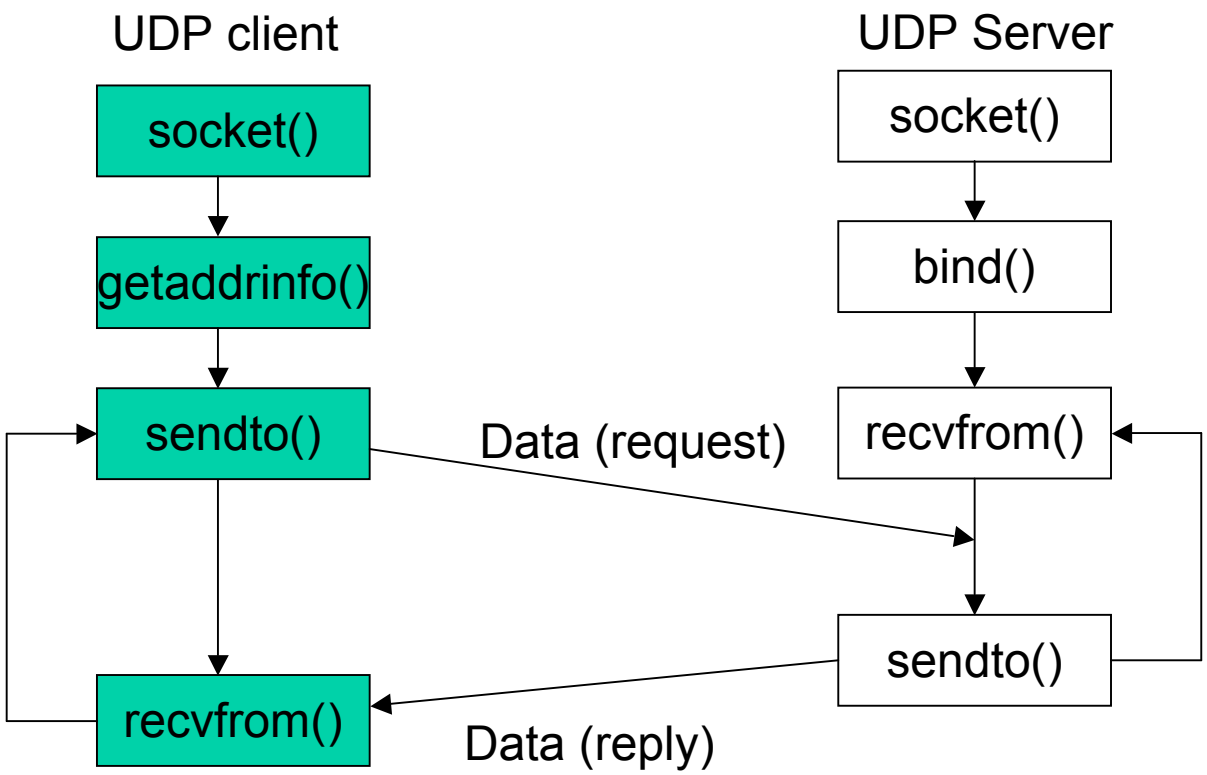
- sshして上記のホストにログイン
  - http://itc.sfc.keio.ac.jp/ より
    Top / 技術情報 / リモートログイン

環境

- MacOSX
  - Xcodeをインストール
  - 「システム環境設定」=>「ネットワーク」=>「詳細」=>「TCP/IP」=>「IPv6の構成」 自動に設定
- Linux
  - IPv6設定を有効に
- Windows
  - cygwinはIPv6サポート不十分(CNS環境を推奨します)

必要な関数

- socket
- getaddrinfo
- sendto
- recvfrom



socket

- **int socket(int domain,**
  int type,
  int protocol);

(例)

```
int sd;

sd = socket(AF_INET6,SOCK_DGRAM,IPPROTO_UDP)
```

getaddrinfo

- **getaddrinfo(const char *nodename,**
  const char *servname,
  const struct addrinfo *hints,
  struct addrinfo **res);
- （例）

```
struct addrinfo hint = {0, PF_INET6, SOCK_DGRAM, IPPROTO_UDP, 0, NULL,
NULL, NULL};
struct addrinfo *res, *ai;
int ret = getaddrinfo("argv[1]", "5525", &hint, &res);
if (ret == 0) {
 for(ai = res; ai != NULL; ai->ai_next){
    sd = socket(ai->ai_family, ai->ai_socktype,ai->ai_protocol);
    /* break if found*/
    break;
  }
```

必要な構造体

・ #include <netdb.h>

```
struct addrinfo {
    int      ai_flags;        /* AI_PASSIVE, AI_CANONNAME */
    int      ai_family;       /* PF_xxx */
    int      ai_socktype;     /* SOCK_xxx */
    int      ai_protocol;     /* 0 or IPPROTO_xxx for IPv4 and IPv6 */
    size_t   ai_addrlen;      /* length of ai_addr */
    char     *ai_canonname;   /* canonical name for hostname */
    struct   sockaddr *ai_addr;      /* binary address */
    struct   addrinfo *ai_next;      /* next structure in linked list */
};
```

sendto

- ssize_t sendto(int s,
  const void *msg,
  size_t len,
  int flags,
  const struct sockaddr *to,
  int tolen);

（例）

```
if (sendto(sd, (char *)&msg, sizeof(msg), 0 , (struct
   sockaddr *)&sv_addr, sizeof(sv_addr)) < 0) {
   perror("sendto");
   exit(-1);
}
```

recvfrom

- **ssize_t recvfrom(int s,**
  void *buf,
  size_t len,
  int flags,
  struct sockaddr *from,
  int *fromlen);

(例)

```
recvlen = recvfrom(sd, (void *)buf, 1024, 0, (struct
    sockaddr *)&sv_addr, &svadlen);
```