ネットワークプログラミング

第1回
ネットワークアプリケーションとは
ソケットプログラミング（１）

ネットワークアプリケーション

- ネットワークを利用したソフトウェアの総称
  - WWW, Telnet, FTPなどが代表例
- プログラムの構成方法から、２～３種に大別
  - ソケットプログラミング
    - ソケットと呼ばれる通信モデルに基づく
    - 独自のプロトコルを実装できる
  - ライブラリやモジュールを用いる
    - Perl や Rubyなどのスクリプト言語では既存プロトコル用の関数モジュールが用意されている
  - WWWの機能を利用（CGIやSSI）

ソケットプログラミング

- ソケット
  - システム標準のネットワーク機能の典型例
  - TCP/UDPのポートを用いた通信をするための仕組
  - ファイル入出力と同様の方法で、
    ネットワーク入出力を扱える
  - ネットワーク機能としては最低限のもの

ソケットプログラミングを理解すれば、
ネットワークプログラミングの基本を理解できる

ソケットの概念

- ソケットは、
  - TCP/UDPのポートを実装したもの
  - ポート番号を手がかりとしてプロセス同士が通信するための仕組
  - TCPのソケット
    - 信頼性の高い全二重のTCPコネクションを設定する
  - UDPのソケット
    - コネクションレスのUDP通信モデル実装に用いる

ここでは、TCPソケットについて扱う

TCPソケット

- ２つに分類
  - サーバのソケット
    - ネットワークサーバ側に作成されるソケット
    - 特定のポートを監視し、クライアントからの接続を待ち受ける（パッシブオープン）
  - クライアントのソケット
    - あるコンピュータの特定のポート（サーバソケット）に対して接続を行うためのソケット（アクティブオープン）
- ２つのソケットは、基本的には同じ
- 設定方法が異なる

TCP/IP　おさらい

- TCP/IPのポート番号

  1～1023　は、ウェルノウン・ポート番号

  - 管理者権限がないと、この範囲のポート番号をプログラムで利用することができない。
  - システムポート番号とも呼ばれる。

  1024～65535　その他のポート番号

  - 1024～49151　登録ポート
    （一般のアプリケーションで利用）
  - 49152～65535　短命ポート
    （一時的に利用されるポート番号）

ソケット処理の流れ

クライアント側ソケット		サーバ側ソケット
ソケット作成		ソケット作成
		ポート番号の設定
		接続の待ち受け
ソケットへのサーバアドレスの設定とサーバへの接続	→	通信用ソケットの取得
データのやりとり	↔	データのやりとり
コネクションのクローズ		コネクションのクローズ

クライアントの
待ち受けに戻る

ネットワークプログラミングの実際

- ここでは、ソケットを用いたネットワークプログラムの実際を紹介する
  - UNIX系OSのソケットを対象とした、C言語のプログラムを例題として扱う
  - yahiko (Linux) にログインして実習を行う（Windowsでも、同様の実習を後日やります。）

クライアントプログラム

クライアントにおけるソケット利用手順

手順	用いる関数
ソケットの作成	**socket()**
ソケットへのアドレスの設定とサーバへの接続	**connect()**
データのやりとり	**send()** ←送信 **recv()** ←受信
コネクションのクローズ	**close()**

(1) ソケットの作成

- ソケットの作成には、socket() を呼び出す。

```
int socket(int《プロトコルファミリ》,
           int《ソケットのタイプ》,
           int《プロトコル》);
```

TCP/IPを意味する記号定数

《プロトコルファミリ》： **PF_INET** (記号定数)を指定

《ソケットのタイプ》： **SOCK_STREAM** か **SOCK_DGRAM**

TCPのとき指定
コネクション型

UDPのとき指定
コネクションレス

《プロトコル》： TCPを利用するなら **IPPROTO_TCP**
UDPを利用するなら **IPPROTO_UDP**

- socket() の戻り値
  - 失敗すると -1 が返る。
  - 成功すると、以後の通信で用いる「ソケットディスクリプタ」の値（整数）が返る。

socket descriptor : ソケットを識別するための識別子

記述例

```
int csocket;  /* ソケットのディスクリプタ */

if ((csocket = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP))<0) {
        /* エラー処理 */
}
```

(2) ソケットへのアドレスの設定とサーバへの接続

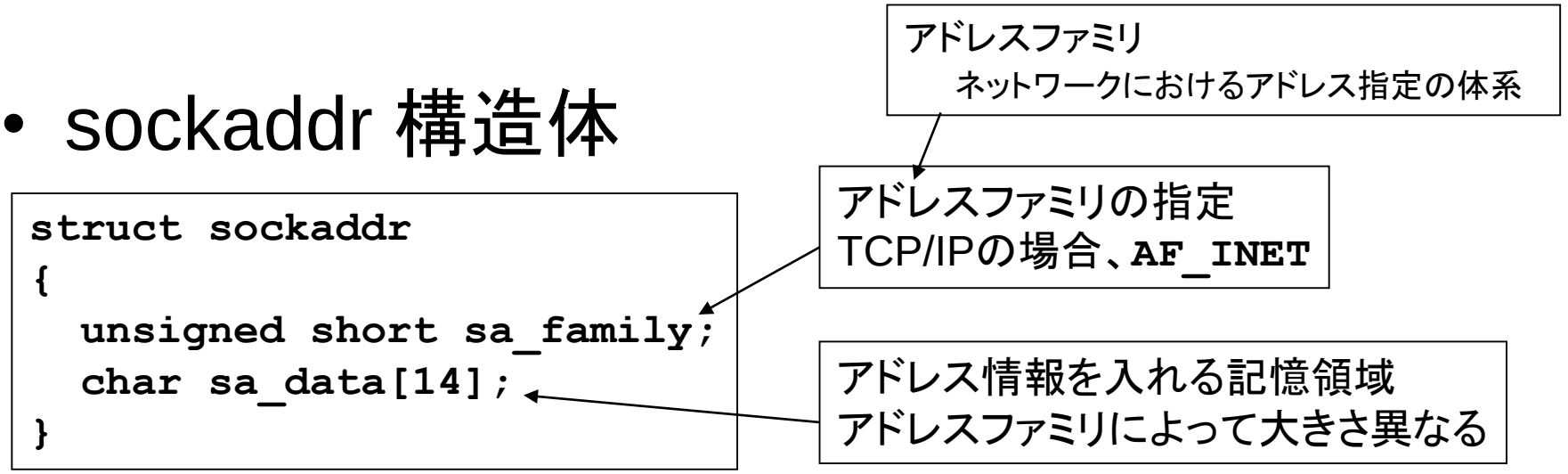
- connect() を用いる

```
int connect(int《ソケットディスクリプタ》,
            struct sockaddr *《接続先のアドレス》,
            unsigned int《第2引数のサイズ》);
```

《ソケットディスクリプタ》： socket()の戻り値を指定する
《接続先のアドレス》　： sockaddr構造体へのポインタ
　　接続先サーバのアドレスなどを指定する役割
《第2引数のサイズ》　：第2引数のサイズ（通常sizeof() を使う）

戻り値： 失敗すると -1 が返る

- sockaddr 構造体




```
struct sockaddr
{
  unsigned short sa_family;
  char sa_data[14];
}
```

- AF_INETを指定する場合、アドレス情報はポート番号とIPアドレス
- sockaddr構造体をそのまま使うと、その区別明確でない。
  ⟶ IP用の sockaddr_in 構造体を使う

- sockaddr_in 構造体

```
struct sockaddr_in
{
  unsigned short sin_family;
  unsigned short sin_port;
  struct in_addr sin_addr;
  char sin_zero[8];
};
```

AF_INET

ポート番号

IPアドレスの構造体

大きさをあわせるための詰め物

sockaddr と sockaddr_in のサイズは同じ。
キャストして使う。

- in_adrr 構造体

```
struct in_addr
{
  unsigned long s_addr;
};
```

sa_data[14]　14 byte

sin_port (2byte) + sin_addr (4byte)

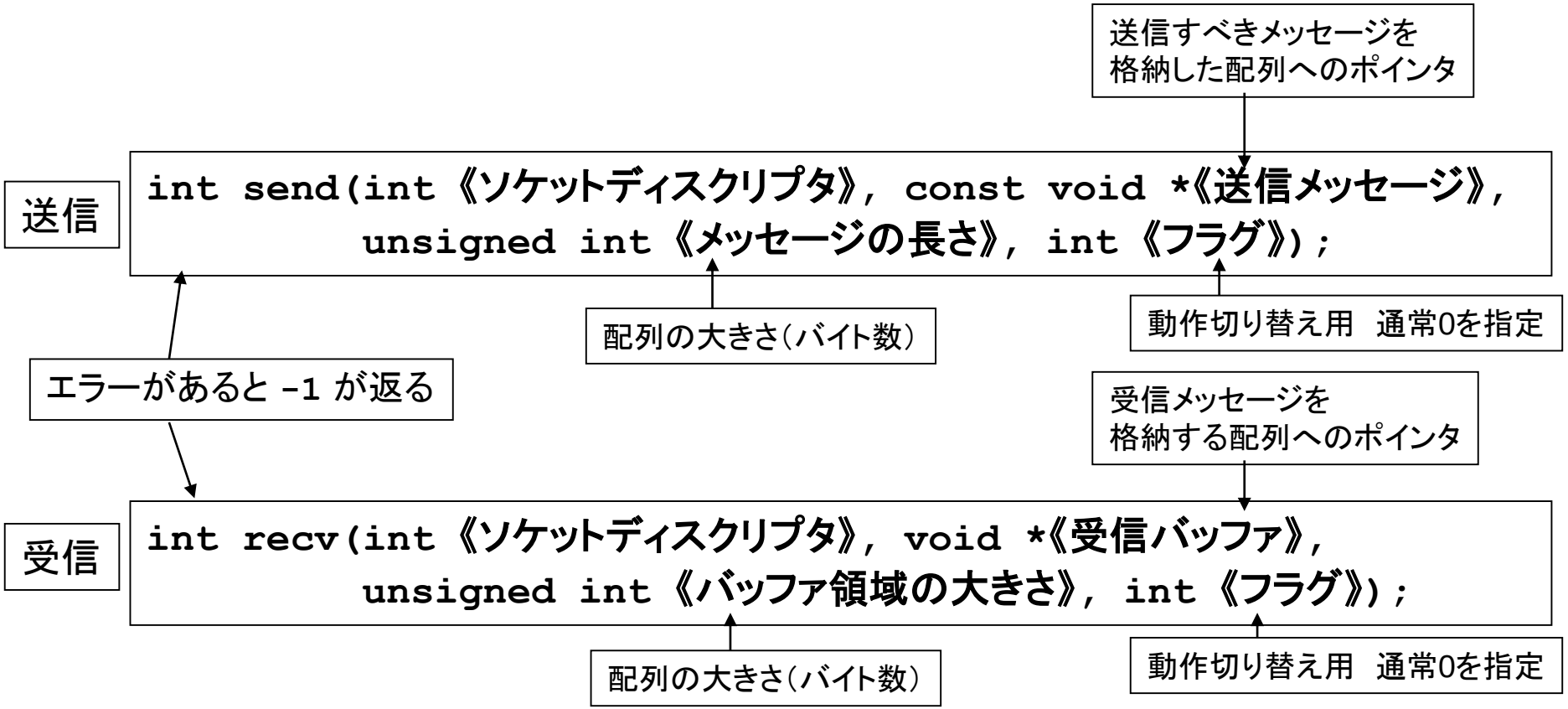
+ sin_zero[8] (8byte) = 14 byte

connect() の使用例

```
/* ① 記号定数の定義 */
#define IPADDRESS “127.0.0.1”
#define PORTNUM 80

/* ② 変数の定義 */
int csocket;  /* ソケットのディスクリプタ */
struct sockaddr_in server;  /* サーバのアドレス*/

/* ③ ソケットのセッティング (mainなどの関数内)*/
  memset(&server, 0, sizeof(server));
  server.sin_family = AF_INET;
  server.sin_addr.s_addr = inet_addr(IPADDRESS);
  server.sin_port = htons(PORTNUM);

/* ④ サーバとの接続 (この前にsocket()実行し、csocketに値入れておく*/
  connect(csocket, (struct sockaddr *)&server, sizeof(server));
```

(3) データのやりとり

- データの送受信は、send() と recv() を使う



- send()の使用例：文字列を送信

```
send(csocket, msg, strlen(msg), 0);
```

文字配列

- recv()の使用例：文字列を受信

```
recv(csocket, msg, sizeof(msg), 0);
```

(4) コネクションのクローズ

- TCPコネクションを閉じるには、close()を使う

```
int close(int《ソケットディスクリプタ》);
```

クライアントプログラムの例

- reader.c
  - サーバ上の特定のポートに対してTCPコネクションをはり、サーバから送られてくるデータを標準出力に出力するプログラム
- yahiko で、プログラムを入力、コンパイル。
  - ソースファイルは、Linuxのディレクトリをファイル共有して、TeraPadなどで作ってもよいし、SCPで転送してもよい。viなどで直接入力してもよい。
  - コンパイル： `gcc reader.c -o reader`

ファイル共有の利用

- スタート→ファイル名を指定して実行
  ¥¥yahiko¥linux
  で、yahikoのホームディレクトリが開く。
  - ここでファイルを作成すれば、
    SCPなどで転送する必要がない。
- ちなみに、
  ¥¥yahiko¥ユーザ名
  は、「マイ　ドキュメント」と同じ場所

クライアントプログラムの実行

- ./reader《サーバのIPアドレス》

- サンプルサーバにアクセスする場合、

  ./reader 202.209.3.88
  （今回の授業の間のみ、アクセス可能）

サーバプログラム

サーバにおけるソケットの利用手順

- サーバ側の利用手順は、
  クライアント側に比べ、若干複雑

手順	用いる関数
(1) ソケットの作成	socket()
(2) ポート番号の設定	bind()
(3) 接続の待ち受け	listen()
(4) 通信用ソケットの取得	accept()
(5) データのやりとり	send(), recv()
(6) コネクションのクローズ	close()

(1) ソケットの作成　socket()

- クライアントの場合と同じ。

(2) ポート番号の設定　bind()

- 特定のポート番号にソケットを関係付ける。
- 引数は、connect() のときと同じ。

```
int bind(int 《ソケットディスクリプタ》,
         struct sockaddr *《接続先のアドレス》,
         unsigned int 《第2引数のサイズ》);
```

- bind() の利用例

```
/* ① アドレスの設定 */
memset(&server, 0, sizeof(server));
server.sin_family = AF_INET;
server.sin_addr.s_addr = htonl(INADDR_ANY);
server.sin_port = htons(SERVERPORTNUM);

/* ② bind()関数の実行 */
bind(serversocket, (struct sockaddr *)&server, sizeof(server));
```

任意のアドレスという意味

htons	short型をホスト側形式からネットワーク用に変換
htonl	long型を変換

(3) 接続待ち受け状態に移行　listen()

- サーバのソケットがクライアントからの接続待ちの状態に遷移する。(プログラムの実行は先に進んでいくことに注意！ここで待っているわけではない。)

```
int listen(int 《ソケットディスクリプタ》,
           int 《接続最大数》);
```

同時に受け付ける
接続の上限値

(4) 通信用ソケットの取得　accept()

- クライアントの接続要求がきたら、実際に通信を行うためのソケットを取得する。

```
int accept(int 《ソケットディスクリプタ》,
           struct sockaddr *《アドレス構造体》,
           unsigned int *《アドレス長》);
```

- 引数

《ソケットディスクリプタ》: 待ち受けているソケットのディスクリプタを渡す

《アドレス構造体》: 接続したクライアントのアドレス等の情報が入る

《アドレス長》: 第2引数のサイズを代入した変数へのポインタ

accept呼び出し前に
変数にサイズを代入しておく

- 戻り値

クライアントとの通信に使うソケットディスクリプタの値（非負）

エラーの場合は-1

(5) データのやりとり

- send(), recv() で行う。
- クライアントの場合と同じ。

(6) コネクションのクローズ

- クライアントの場合と同じ。

サーバプログラムの例

- netclock.c
  - サーバコンピュータ上の時計を読み取って、クライアントに渡すプログラム。
  - 実行しても、そのままでは何も起こらない。
  - クライアントからの接続要求があると、
    クライアントに時刻を返答し、
    同じものをコンソール画面に表示する。
  - このプログラムを停止するには、Ctrl-Cなどの割り込み信号を入力する必要がある。

演習での注意（ポート番号）

- yahiko で全員同時に20000番ポートを使用することはできない。
- そこで、reader.c, netclock.c を実際に試すために、使用するポート番号を変更する。
- 20000＋名簿番号のポートを使用すること。
- ２つのプログラムの SERVERPORTNUM を自分に割り当てられた番号に変更し、
  再度コンパイル、実行してみよう。

netclock.c のコンパイル・実行

- コンパイル
  gcc netclock.c -o netclock
- 実行
  ./netclock
- 実行時のヒント
  - TeraTerm を2つ開き、並べて配置する。
    - 片方をサーバプログラムを実行するのに使い、もう一方をクライアントプログラムを実行するのに使うとよい。