ネットワークプログラミング第7回（ネットワークとプログラミング(2)）

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
    - 6/8(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6回課題
UDP-echoクライアント作成

- 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/

```
#include <stdio.h>
#include <netdb.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[]){
 int sock;
 int ret;
 char buf[256];
 struct addrinfo *ai, *res;
 struct sockaddr_in6 server;
 struct addrinfo hints = {0, AF_INET6,
SOCK_DGRAM , 0, 0, NULL, NULL, NULL};

 ret = getaddrinfo(argv[1], argv[2],
&hints, &res);
 if (ret != 0){
   fprintf(stderr,
"getaddrinfo(%s)¥n",
    gai_strerror(ret));
  exit(-1);
 }
```

```
 for(ai = res; ai; ai = ai->ai_next){
  sock = socket(ai->ai_family, ai->ai_socktype,
ai->ai_protocol);
  if (sock < 0){
   perror("socket");
   exit(-1);
  }
  memset (&server, 0, sizeof(server));
  memcpy (&server, ai->ai_addr, ai-
>ai_addrlen);
  break;
 }

 while (1){
  ret = read (0, buf, sizeof(buf));
  if (ret < 0){
   perror("read");
   exit(-1);
  }
```

```
 ret = sendto(sock, buf, ret, 0, (struct sockaddr *)&server,
sizeof(server));
 if (ret < 0){
   perror("sendto");
   exit(-1);
 }
 memset(buf, 0, sizeof(buf));
 ret = recvfrom(sock, buf, sizeof(buf), 0, NULL, 0);
 if (ret < 0){
   perror("recvfrom");
   exit(-1);
  }
  printf("%s¥n", buf);
 }
 freeaddrinfo(res);
 return 0;
}
```

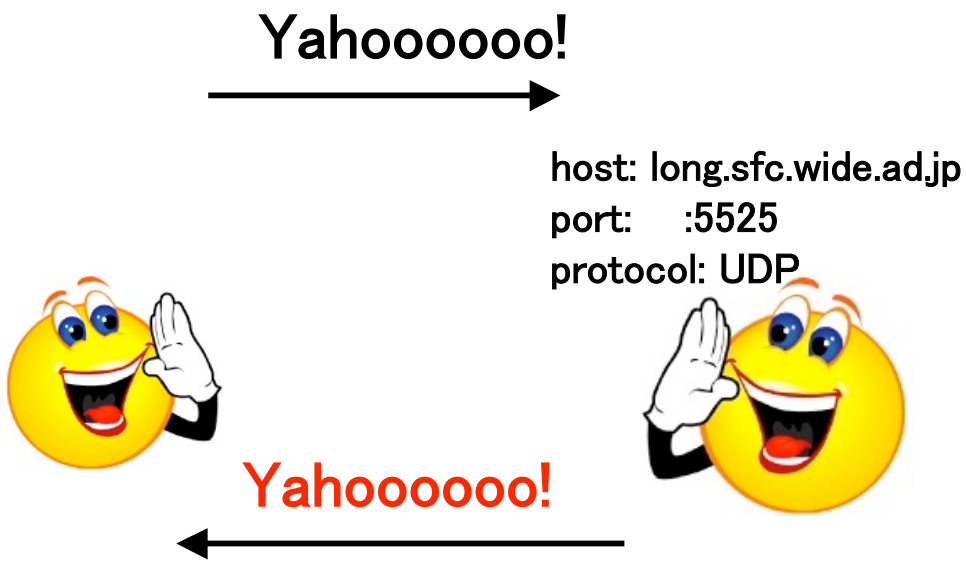
今日のお題

- 講義
  - UDPエコークライアントのおさらい
- 実習/課題：データの送受信
  - UDP echo サーバー作成

UDPクライアント(おさらい)

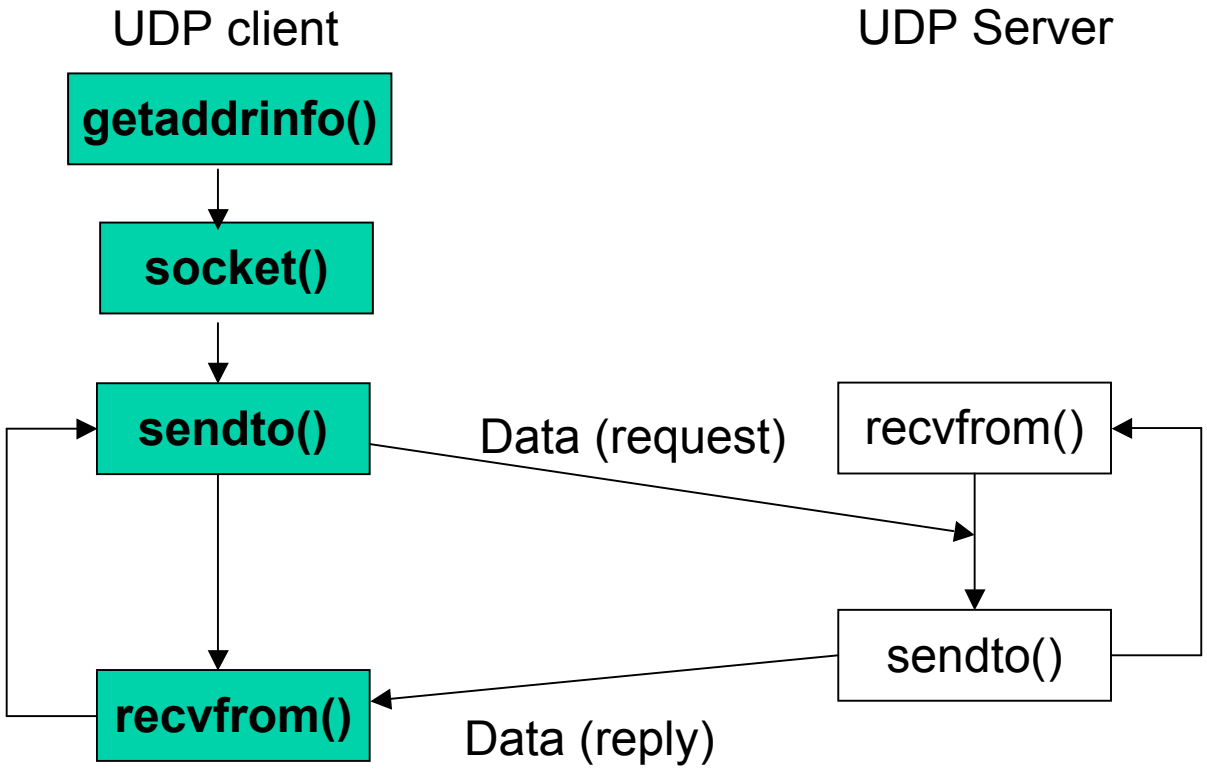
UDPエコークライアント

- サーバ・クライアント型のアプリケーション
- クライアントの送信したメッセージと同じものが、サーバから返答(エコー)される



必要な処理

- getaddrinfo
- socket
- sendto
- recvfrom



Echoクライアント処理

```
#include <stdio.h>
#include <netdb.h>

int main(int argc, char *argv[]){

  /* 利用可能なアドレスの取得 */
  getaddrinfo(argv[1], argv[2], &hints, &res);

  /* ソケットの作成 */
  sock = socket(ai->family, ai->sock_type, ai->protocol);
  /* あて先アドレスの取得 */
  memcpy (&server, ai->ai_addr, ai->ai_addrlen);

  freeaddrinfo(res);
```

Echoクライアント処理(続き)

```
  /* Ctrl-C が入力されるまで継続 */
  while (1) {
   /* キーボードデータの入力待ち受け */
   len = read (stdin, buf, sizeof(buf));

   /* エコーサーバへデータ送信 */
   sendto (sock, buf, len, 0, (struct sockaddr *)&server,
        sizeof(server));

   /* サーバからの応答を受信し、表示 */
   recvfrom (sock, buf, sizeof(buf), 0, NULL, 0);
   printf ( “%s¥n” , buf);
 }

  return 0;
}
```

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;
   }
}
```

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

実習
UDP-echoクライアント改造

- 前回作成したechoクライアントに、サーバに送信するデータにヘッダ情報(ログイン名)をつけて送信
- echoサーバは以下
  - IP address: **long.sfc.wide.ad.jp**
  - Port : **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/

```
% ./a.out long.sfc.wide.ad.jp 5525
Hello, server
[tazaki] Hello, server
How are you?
[tazaki] How are you?
```

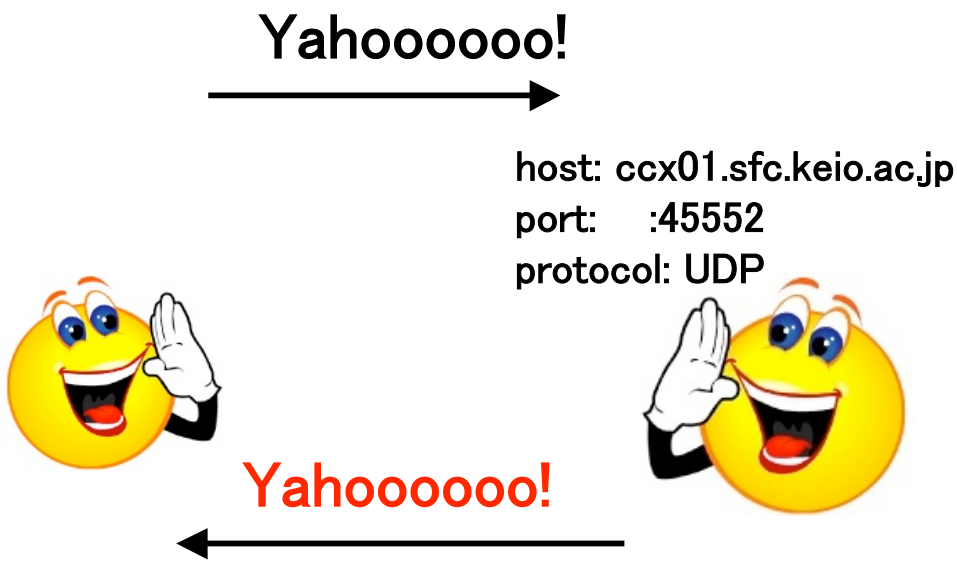
サーバプログラム

サーバプログラム

- Server：サービスを提供するプログラム
  - 例)HTTPサーバ
    - ポート80番で、HTTPのリクエストに応答し、HTMLデータをクライアントに提供(サービス)する
- 最近はクライアント・サーバの境界線はあいまいになってきている(例:P2Pアプリケーション)
  - サービスを提供しながら、自分もサービスを他人の提供したサービス利用する

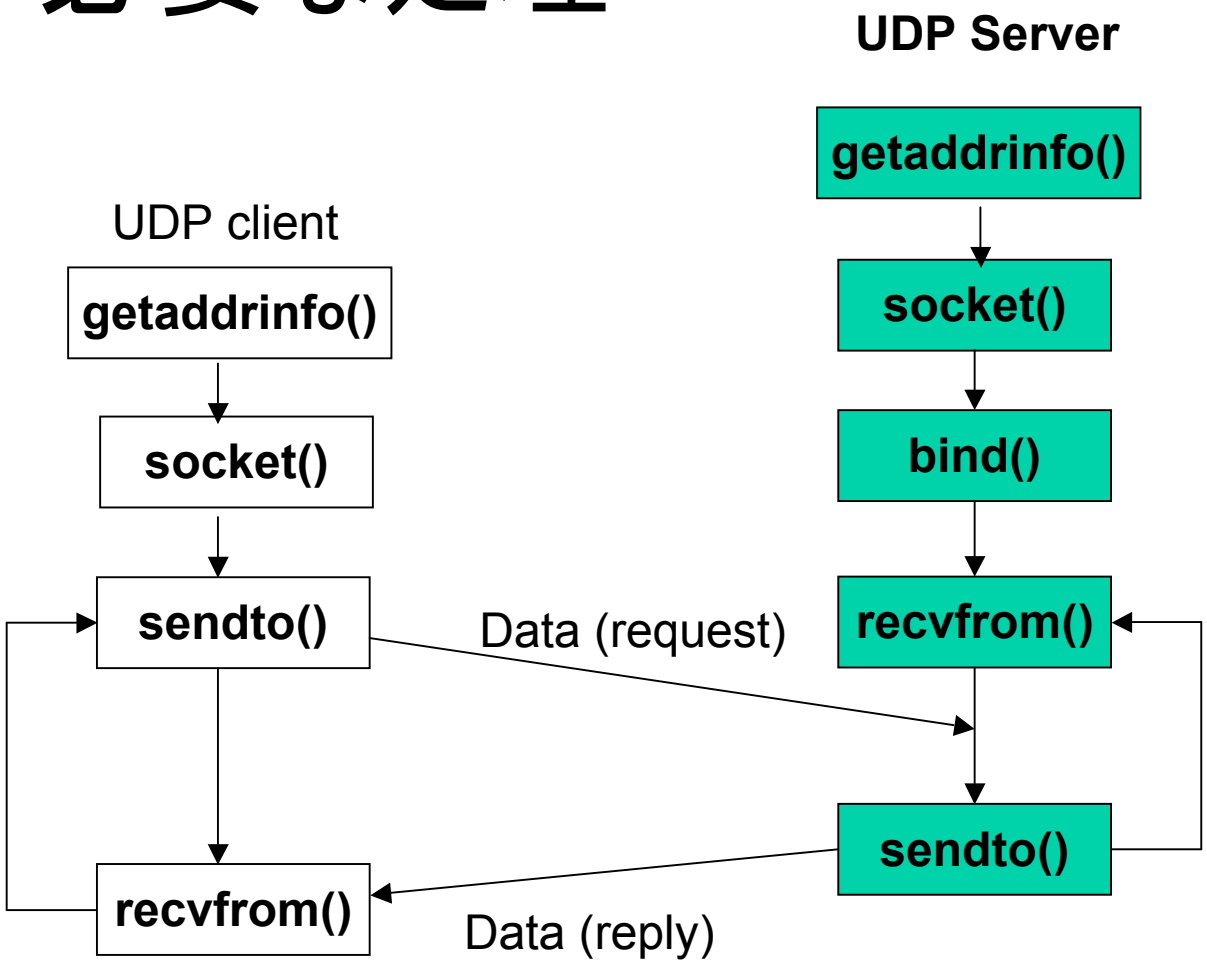
UDPエコーサーバ

- サーバ・クライアント型のアプリケーション
- クライアントの送信したメッセージと同じものが、サーバから返答(エコー)される



必要な処理

- getaddrinfo
- socket
- bind()
- sendto
- recvfrom



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

bind()システムコール

- int bind(int sockfd,struct sockaddr *addr,int addrlen);
  - 開いたソケットに名前を付ける
  - => アドレスを指定して、ソケットを利用する
    - 送信元アドレスを指定して、パケットを送信
    - 受信するアドレス・ポートを限定

(例)

```
getaddrinfo (NULL, “47583”, &hints, &res);

for (ai=res; ai; ai = ai->ai_next){
 sock = socket(ai->ai_family, ai->ai_socktype; ai->prototocol);
 bind(sock, ai->ai_addr, ai->ai_addrlen);
}
```

recvfrom()システムコール(再)

- ssize_t recvfrom(int socket,
  void *restrict buffer,
  size_t length,
  int flags,
  struct sockaddr *restrict address,
  socklen_t *restrict address_len);
- 送信元を知るには、address引数に通知されるsockaddrを読み出し

例)

```
/* 受信したデータの送信元アドレスをpeerに格納 */
readlen = recvfrom (sockfd[j], buf, sizeof (buf), 0,
                    (struct sockaddr *) &peer, &addrlen_peer);
/* peerをそのまま送信先アドレスとして利用 */
sendlen = sendto (sockfd[j], buf, readlen, 0,
                  (struct sockaddr *) &peer, addrlen_peer);
```

実習:課題(UDPエコーサーバ)

- UDPエコーサーバを作成する
- 待ち受けするポート番号(getaddrinfo時に指定するポート番号)は、学籍番号下5桁を利用
  - 例)学籍番号　80749358

    getaddrinfo(NULL, “**49358**”, &hint, &res);
    bind(sock, ai->ai_addr, ai->ai_addrlen);
- クライアントは前回作成したクライアントを使用し、送信先サーバのポートを変えデータを送信する

実習:課題(UDPエコーサーバ)

- 自作のサーバを実行させ、別のコンソールにてクライアントプログラムを動作させる。
- あて先IPアドレスには、
  - localhost(::1、ループバックアドレス)が利用可能
  - ifconfigコマンドで調べたIPv6アドレスも利用可能
- 実行例)(ccx01でのクライアント例)
  - **./a.out 2001:200:1c0:1100:21a:64ff:fe8e:2e40 49358**
- **サーバプログラムを添付して提出**