UNIXシステムプログラミング

第9回 ネットワークプログラミング(2)

2019年11月29日

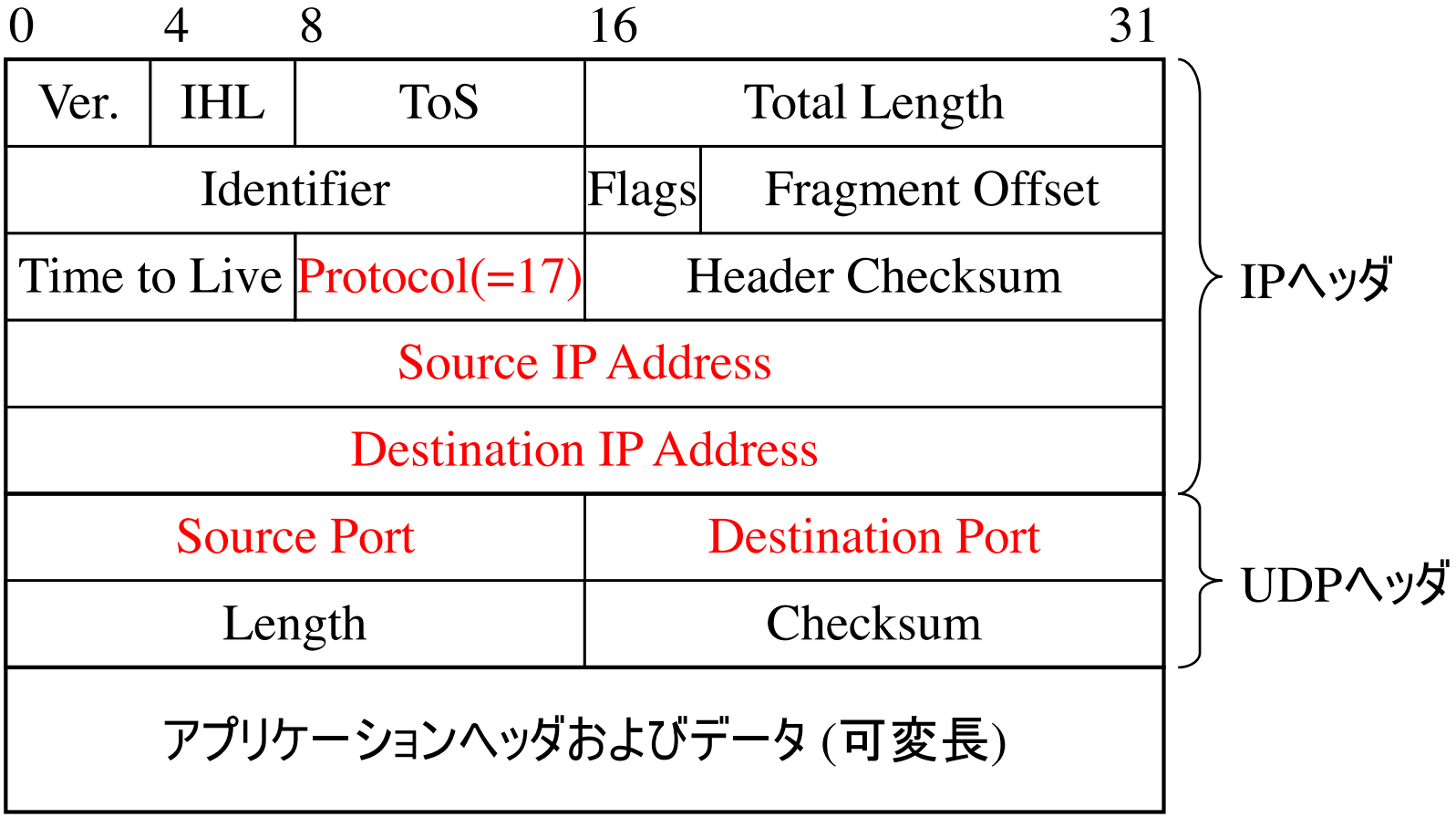
情報工学科

寺岡文男

復習：ネットワークプログラミングの注意点

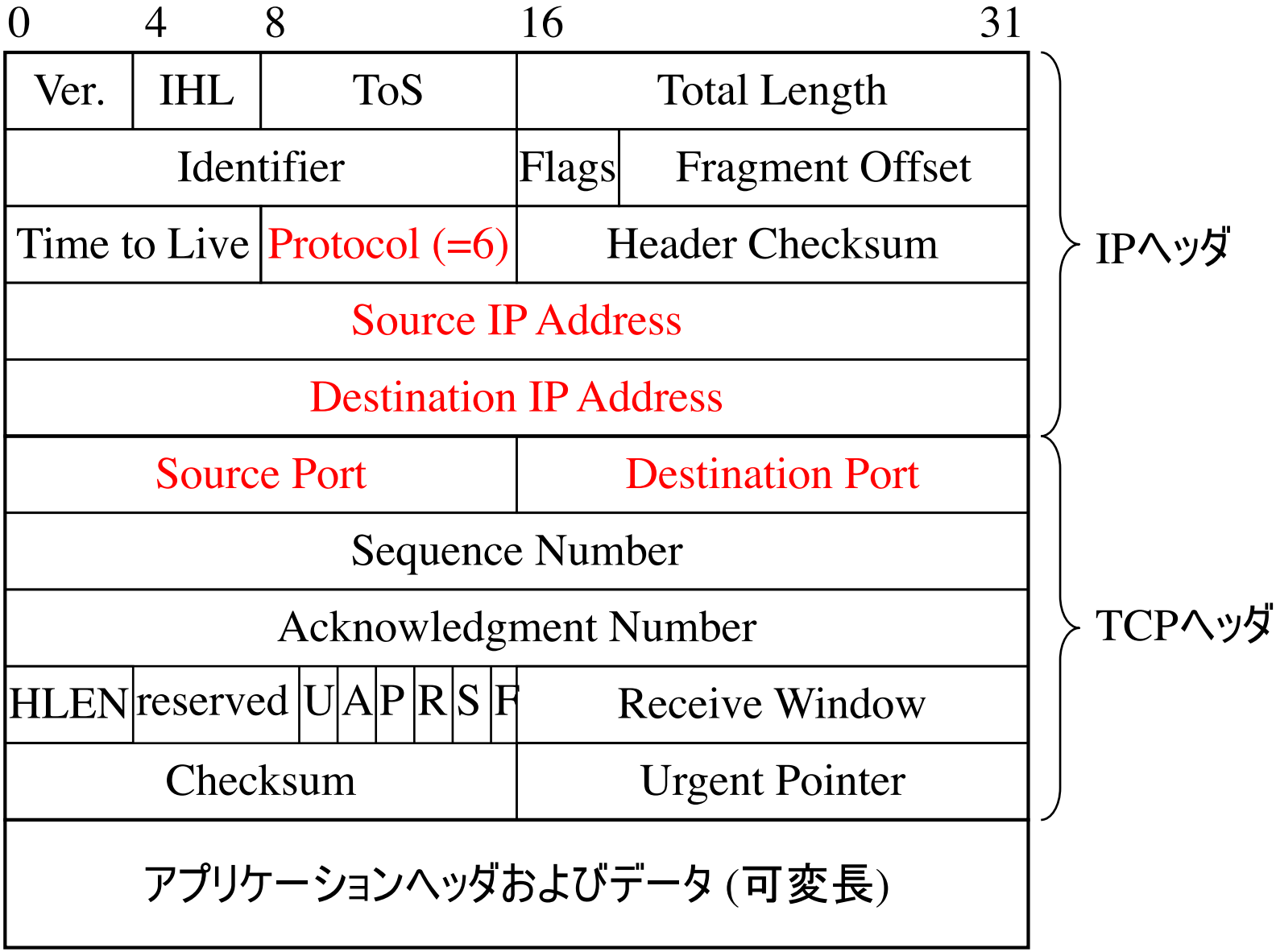
- 通信相手との相互動作であることを念頭に
- ソケットインタフェースは汎用的に設計されている
  - インターネットであることを明示的に示す必要がある
  - e.g., PF_INET, AF_INET, struct sockaddr_in
- ビット長を明示した型を使う
- バイトオーダーに注意する
- 用途に応じてUDPとTCPを使い分ける

復習：UDP / IP パケット

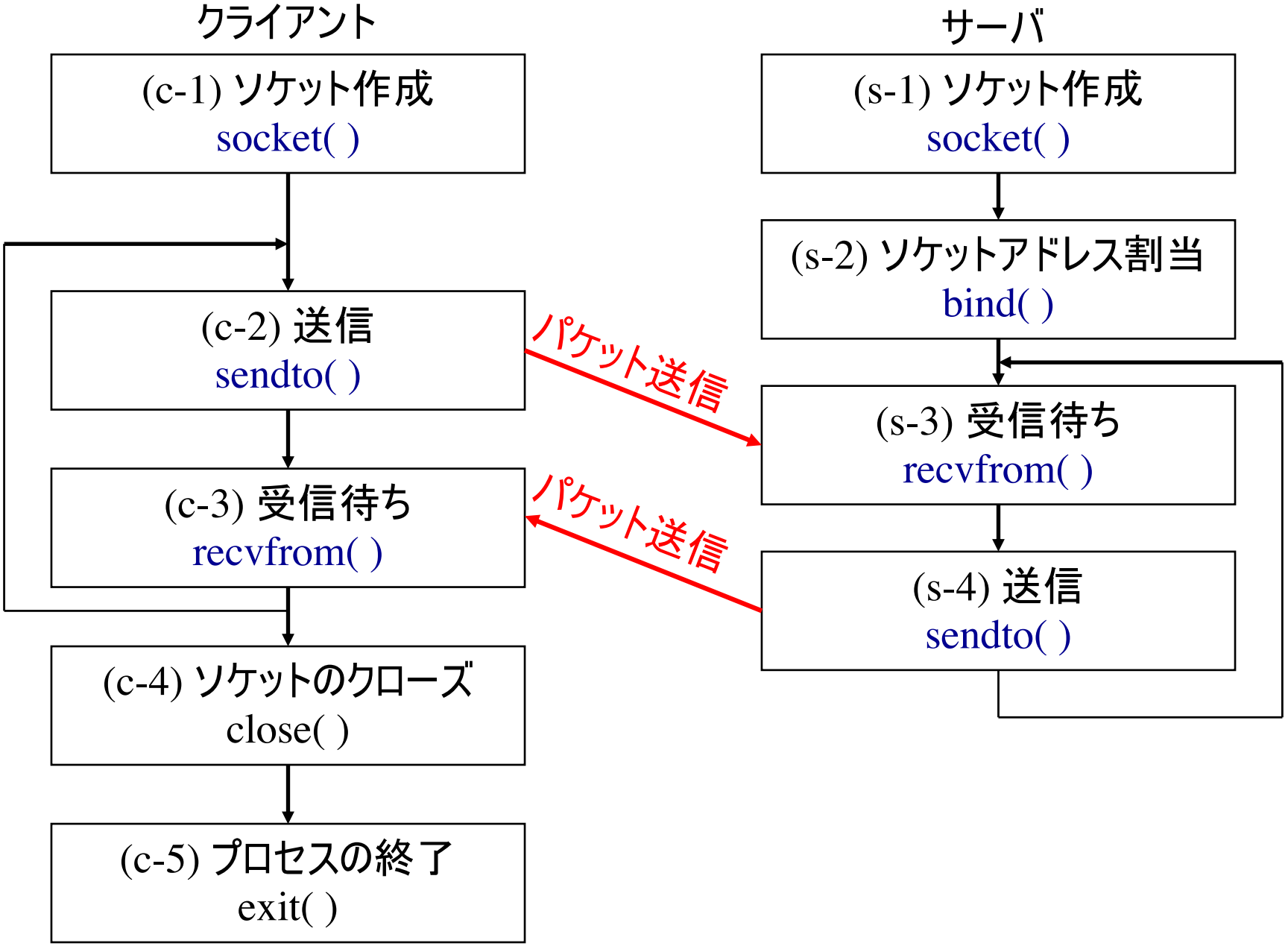


- 通信相手をソケットで指定 (TCPも同様)
- ソケット = IPアドレス + ポート番号

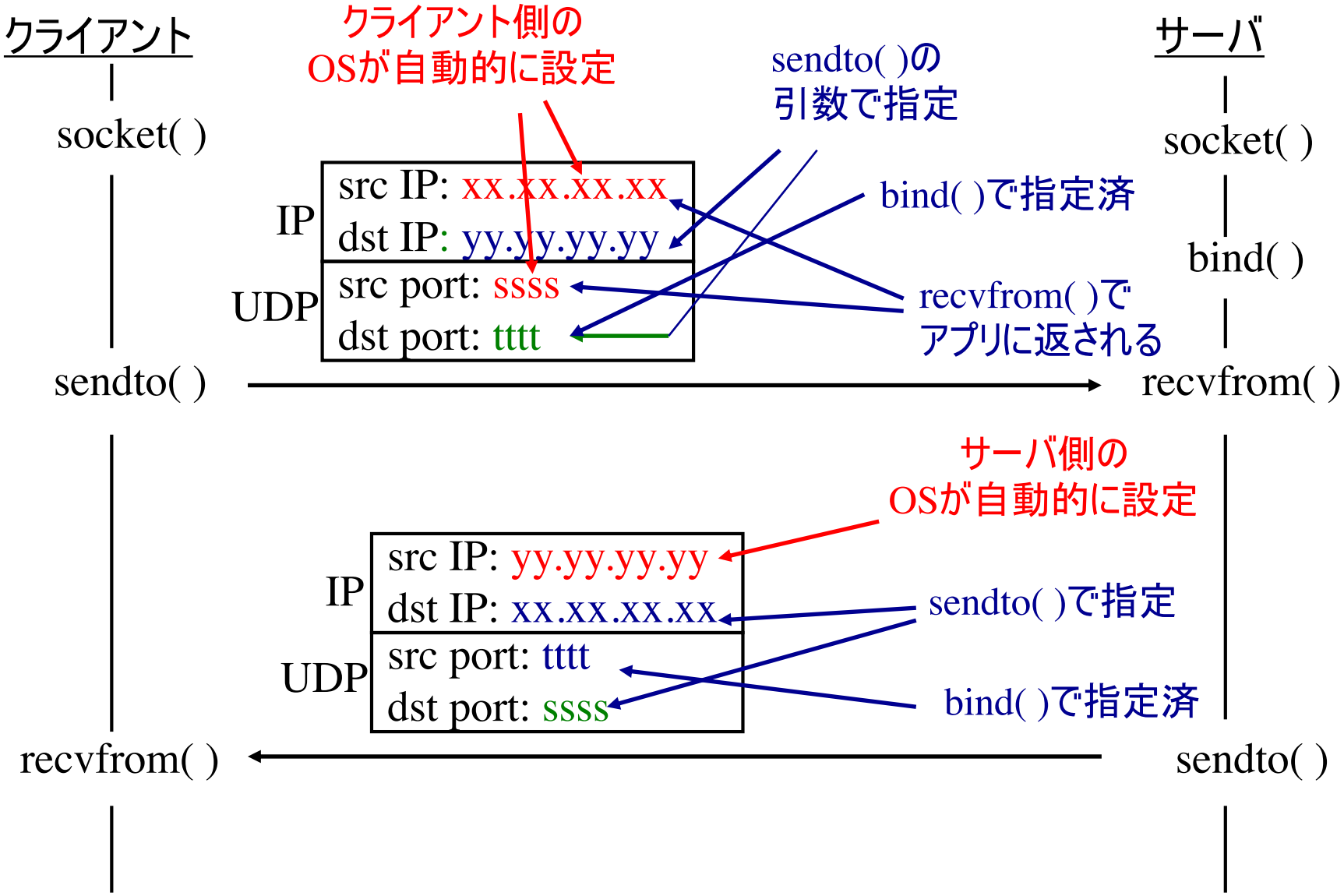
復習：TCP / IP パケット



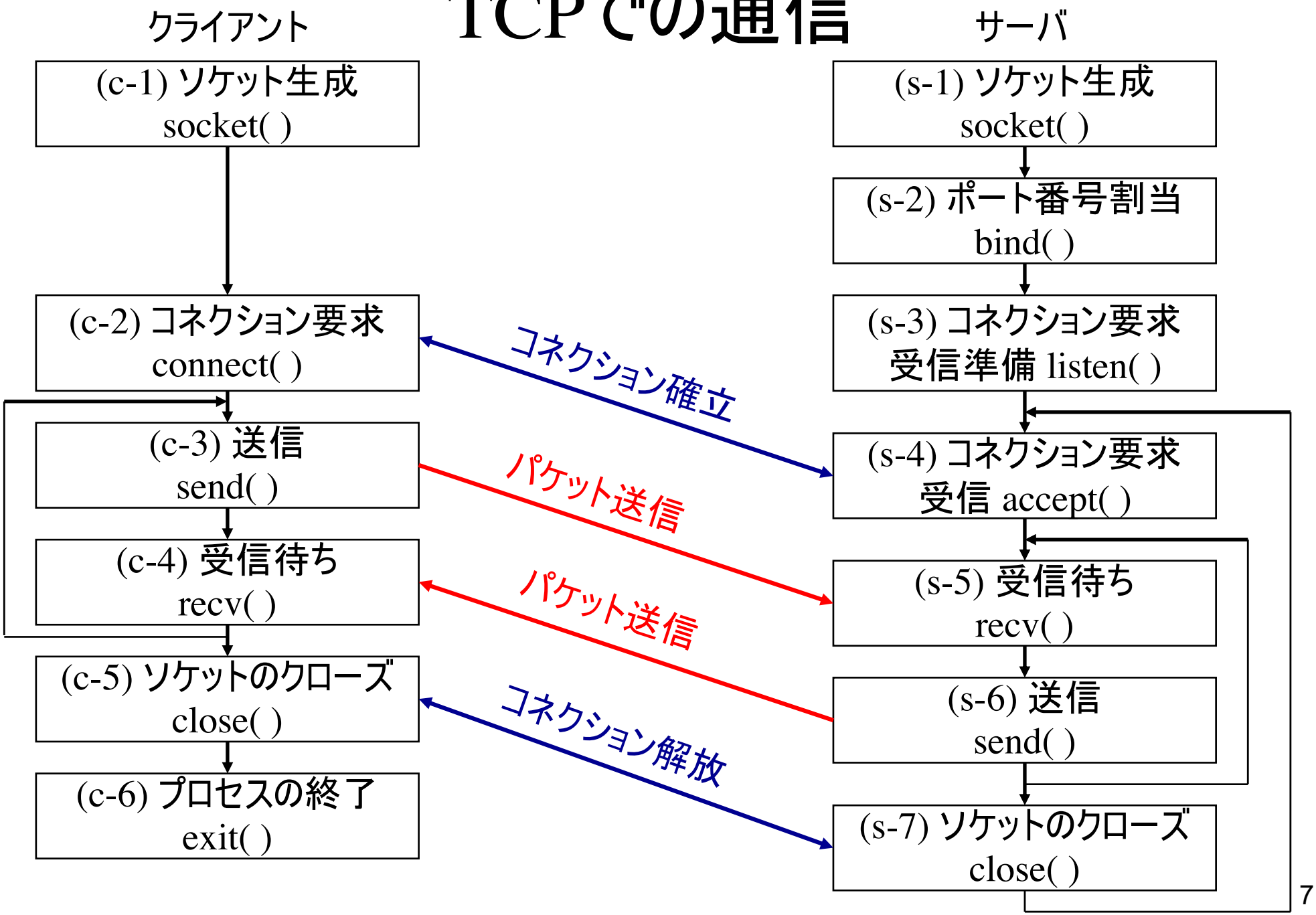
復習：UDPでの通信


クライアント
(c-1) ソケット作成
socket( )
(c-2) 送信
sendto( )
(c-3) 受信待ち
recvfrom( )
(c-4) ソケットのクローズ
close( )
(c-5) プロセスの終了
exit( )
パケット送信
パケット送信
サーバ
(s-1) ソケット作成
socket( )
(s-2) ソケットアドレス割当
bind( )
(s-3) 受信待ち
recvfrom( )
(s-4) 送信
sendto( )

復習: システムコールとパケット送受信の関係(UDP)



TCPでの通信


クライアント
サーバ
(c-1) ソケット生成
socket( )
(s-1) ソケット生成
socket( )
(s-2) ポート番号割当
bind( )
(c-2) コネクション要求
connect( )
(s-3) コネクション要求
受信準備 listen( )
コネクション確立
(c-3) 送信
send( )
(s-4) コネクション要求
受信 accept( )
パケット送信
(c-4) 受信待ち
recv( )
(s-5) 受信待ち
recv( )
パケット送信
(c-5) ソケットのクローズ
close( )
(s-6) 送信
send( )
コネクション解放
(c-6) プロセスの終了
exit( )
(s-7) ソケットのクローズ
close( )

サーバの動作 (1)

1. socket( )で送受信のためのソケットを開く
2. bind( )で自分のソケットにポート番号を割り当てる
   - クライアントはこのポート番号を指定してコネクション確立要求を送信する
   - 自分のソケットのIPアドレスはOSが自動的に割り当てる
3. listen( )でクライアントからのコネクション確立要求の受信準備をする
4. accept( )でクライアントからのコネクション確立要求を受信し，応答を返す → コネクションが確立する
   - accept( )は新しいソケット記述子を返す
   - 以降のrecv( )やsend( )には新しいソケット記述子を使用する

サーバの動作 (2)

5. recv( )でクライアントからのデータ受信を待つ
   - コネクション確立済なので，相手側のソケットアドレスは返らない
6. send( )でクライアントにデータを送信する
   - コネクション確立済なので，相手側のソケットアドレスを指定する必要はない
- 以降，必要なだけ 5 と 6 を繰り返す

7. 1つのクライアントへのサービスが終了したらaccept( )で得られたソケットを close( )でクローズし，4 に戻る
   - 次のクライアントからのコネクション要求を待つ

クライアントの動作

1. socket( )で送受信のためのソケットを開く
2. connect( )でコネクション確立要求を送信する
   - 正常に終了するとコネクションが確立される
3. send( )でデータを送信する
   - コネクション確立済なので，相手側のソケットアドレスを指定する必要はない
4. recv( )でデータを受信する
   - コネクション確立済なので，相手側のソケットアドレスは返らない

- 必要なだけ 3 と 4 を繰り返す

5. 処理が終わったら close( )でコネクションを解放する
6. exit( )でプロセスを終了する

listen( ): コネクション確立要求受信準備

```
#include <sys/types.h>
#include <sys/socket.h>

int listen(int s, int backlog);
```

- 引数
  - s: ソケット記述子
  - backlog: コネクション確立要求パケットの待ち行列のサイズ
    - 通常は 5 程度にすればよい
- 返り値
  - 正常の場合 0 が返る
  - エラーの場合 –1 が返り，errnoに理由が設定される

accept( ): コネクション確立要求(passive) (1)

```
#include <sys/types.h>
#include <sys/socket.h>

int accept(int s, struct sockaddr *addr,
                                socklen_t *addrlen);
```

- 引数
  - s: ソケット記述子
  - addr: コネクション要求側ソケットの情報(i.e., IPアドレス、ポート番号)が格納される領域のポインタ
  - addrlen:上記の領域のサイズを格納したsocklen_t領域のポインタ. 受信後には実際のサイズがこの領域に設定される

(次ページへ)

accept( ): コネクション確立要求(passive) (2)

- 返り値
  - 正常の場合，新しいソケット記述子を返す
  - 以降のrecv( ), send( )にはこのソケット記述子を使用する
  - エラーの場合 –1 が返り，errnoに理由が設定される

connect( ): コネクション確立要求(active)

```
#include <sys/types.h>
#include <sys/socket.h>

int connect(int s, struct sockaddr *name,
                                socklen_t namelen);
```

- 引数
  - s: ソケット記述子
  - name: サーバ側のソケットアドレス構造体へのポインタ
  - namelen: 上記構造体のサイズ
- 返り値
  - 正常の場合 0 が返る
  - エラーの場合 –1 が返り，errnoに理由が設定される

サーバ側のプログラム例

```
int s, s2;
struct sockaddr_in myskt;  // 自ソケットアドレス
struct sockaddr_in skt;    // クライアント側のソケットアドレス
socklen_t sktlen;

if ((s = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
        . . .
}
// myskの各メンバの値を設定しておく
if (bind(s, (strcut sockaddr *)&mkskt, sizeof myskt)) < 0) {
        . . .
}
if (listen(s, 5) < 0) {
        . . .
}
sktlen = sizeof skt;
if ((s2 = accept(s, (struct sockaddr *)&skt, &sktlen)) < 0) {
        . . .
}
// sktにクライアント側のソケットアドレスが返される
```

クライアント側のプログラム例

```
int s;
struct sockaddr_in skt;     // サーバ側のソケットアドレス
in_port_t port;             // サーバ側のポート番号
struct in_addr ipaddr;      // サーバのIPアドレス

if ((s = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
        . . .
}

// port, ipaddrを設定しておく

memset(&skt, 0, sizeof skt);
skt.sin_family = AF_INET;
skt.sin_port = htons(port);
skt.sin_addr.s_addr = htonl(ipaddr.s_addr);
if (connect(s, (struct sockaddr *)&skt, sizeof skt) < 0) {
        . . .
}
```

recv( ): パケットの受信

```
#include <sys/types.h>
#include <sys/socket.h>

ssize_t recv(int s, void *buf, size_t len, int flags);
```

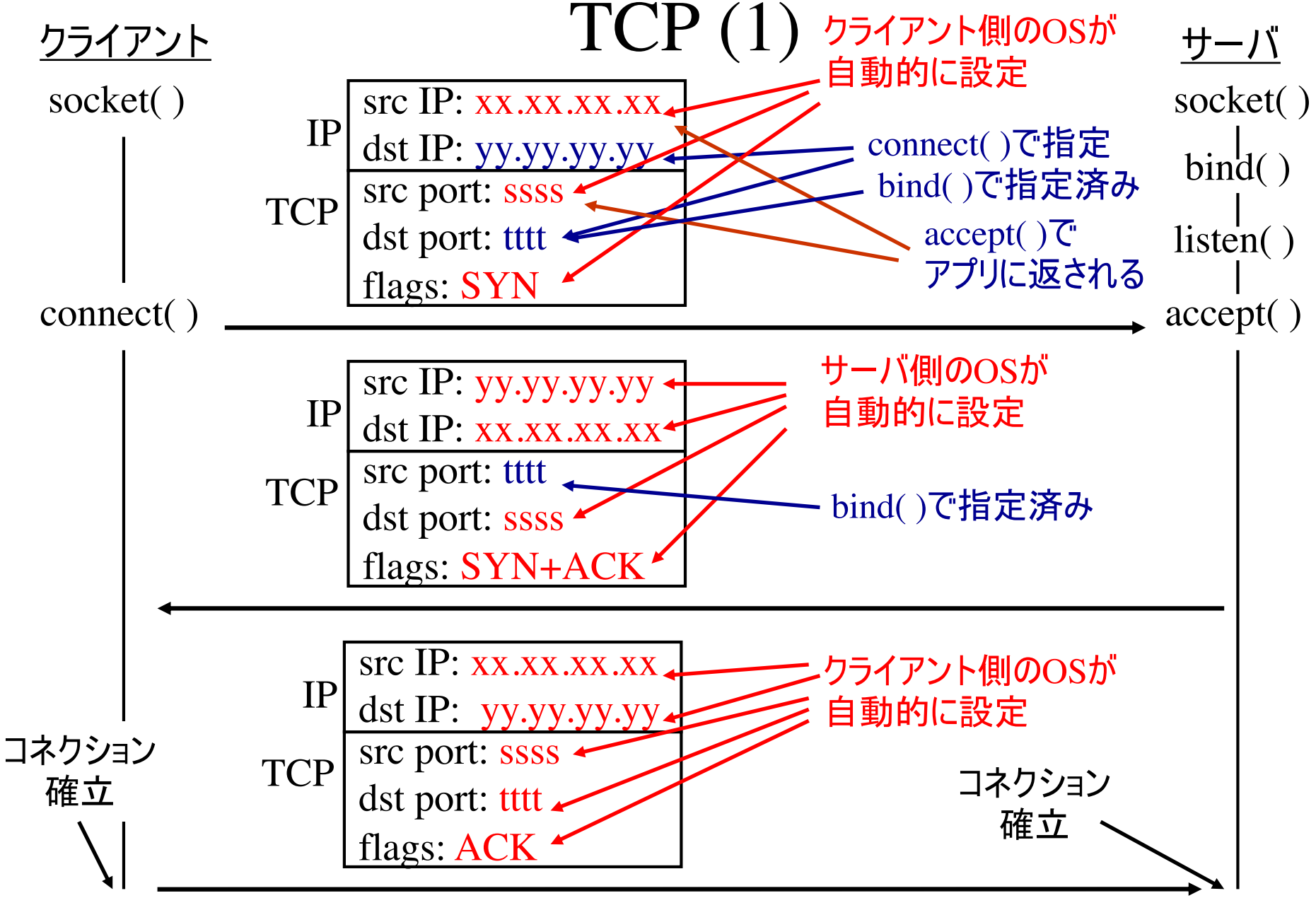
- 引数
  - s: ソケット記述子 (socket( ) または accept( )が返したもの)
  - buf: 受信データを格納する領域へのポインタ
  - len: 上記領域のサイズ
  - flags: 通常は 0 を指定する
- 返り値
  - 正常の場合，実際に受信したデータサイズが返る
  - エラーの場合 –1 が返り，errnoに理由が設定される

send( ): パケットの送信

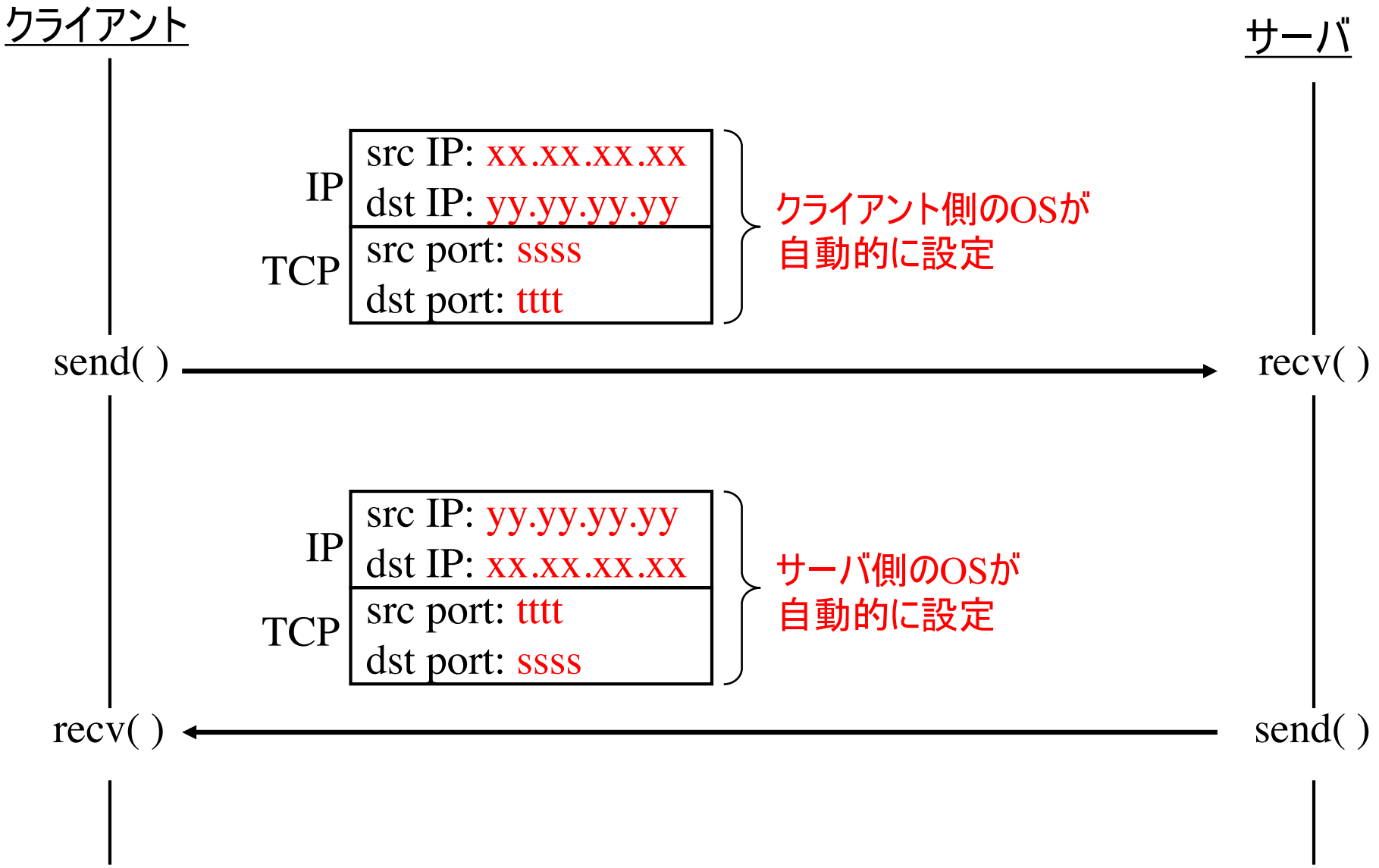
```
#include <sys/types.h>
#include <sys/socket.h>

ssize_t send(int s, void *msg, size_t len, int flags);
```

- 引数
  - s: ソケット記述子 (socket( ) または accept( )が返したもの)
  - msg: 送信データを格納した領域へのポインタ
  - len: 送信データサイズ
  - flags: 通常は 0 を指定する
- 返り値
  - 正常の場合，実際に送信したデータサイズが返る
  - エラーの場合 –1 が返り，errnoに理由が設定される

システムコールとパケット送受信の関係：TCP (1)



システムコールとパケット送受信の関係：TCP (2)



TCPとUDPでの受信データの取り扱い

- TCPは上位層に(信頼性のある)バイトストリームを提供
  - 送信側のメッセージ境界は保存されない
- UDPは上位層に(信頼性を保証しない)データグラム通信を提供
  - 送信側のメッセージ境界は保存される
- 例：送信側は500バイト, 1,000バイト, 500バイトを送信
  → 受信側のバッファには2,000バイトのデータ
  - TCP: メッセージ境界のない2,000バイトのデータとして見える
    - 何バイト単位で recv( ) してもよい
  - UDP： 500B, 1000B, 500Bという3つのデータとして見える
    - 1,024Bでrecvfrom( )すると，最初は500Bが返り…となる

ホスト名からIPアドレスへの変換

- ユーザは “www.inl.ics.keio.ac.jp” のようなホスト名で通信相手を指定
- IPはIPアドレスで通信相手を認識

- クライアントがDNS (Domain Name System)サーバに問い合わせることにより，ホスト名をIPアドレスに変換
  - 各組織がホスト名とIPアドレスの対応を管理
  - 世界規模の階層的な分散データベース
  - DNSサーバへの問合せにはUDPが使用される
  - 詳細は省略 (ネットワーク工学I参照)

ホスト名やサービス名からソケットアドレスへ

```
#include <sys/socket.h>
#include <netdb.h>

int getaddrinfo(char *nodename, char *servname,
      struct addrinfo *hints, struct addrinfo **res);

void freeaddrinfo(struct addrinfo *ai);
```

- 引数
  - nodename: ノード名の文字列 or IPアドレスの文字列
  - servname: サービス名の文字列 or ポート番号の文字列
    - nodename, servicename はどちらかがNULLでもよい
  - hints: ノード名やサービス名をソケットアドレスに変換するためのヒント (or NULLポインタ)
  - res: 結果としてのaddrinfo構造体のリストの先頭
    (関数内で領域が確保される → 使用後に要解放)

ホスト名やサービス名からソケットアドレスへ

```
struct addrinfo {
    int ai_flags;                /* input flag */
    int ai_family;               /* protocol family */
    int ai_socketype;            /* socket type */
    int ai_protocol;             /* protocol for socket */
    socklen_t ai_addrlen;        /* length of sock addr */
    struct sockaddr *ai_addr;    /* socket address */
    char *ai_canonname;          /* canonical name */
    struct addrinfo *ai_next;    /* pointer to next */
}
```

- getaddrinfo() で返された addrinfo構造体(のリスト)の領域を解放するには freeaddrinfo() を呼び出す

getaddrinfo()の使用例

```
struct addrinfo hints, *res;
char *host, *serv;
int err;

// host が指す領域にホスト名を格納しておく (e.g.,"www.ics.keio.ac.jp"
//     or "131.113.126.136"
// serv が指す領域にサービス名を格納しておく (e.g., "http" or "80")

memset(&hints, 0, sizeof hints);
hints.ai_socktype = SOCK_STREAM;        // TCPを使用
if ((err = getaddrinfo(host, serv, &hints, &res)) < 0) {
        fprintf(stderr, "getaddrinfo: %s¥n", gai_strerror(err));
        exit(1);
}
// res->ai_addr がソケットアドレスの領域を指す (struct sockaddr *)
. . .

freeaddrinfo(res);          // getaddrinfo()が確保した領域を解放
```

参考：IPv4/v6対応：サーバ側の例 (1)

```
struct addrinfo hints, *res;
struct sockaddr_storage sin;
char *serv, buf[BUFSIZE];
int sd, sd1, err, sktlen;

// servが指す領域にサービス名(e.g, “http” or “80”)を格納しておく.
memset(&hists, 0, sizeof hints);
hints.ai_flags = AI_PASSIVE;      // accept()で使用することを示す
hints.ai_socktype = SOCK_STREAM; // TCPを使用することを示す
if ((err = getaddrinfo(NULL, serv, &hints, &res)) < 0) {
        fprintf(stderr, “getaddrinfo: %s¥n”, gai_strerror(err));
        exit(1);
}
if ((sd = socket(res->ai_family, res->ai_socktype,
                                  res->ai_protocol)) < 0) {
        perror(“socket”);
        exit(1);
}
// 次ページへ
```

参考：IPv4/v6対応：サーバ側の例 (2)

```
// 前ページから
if (bind(sd, res->ai_addr, res->ai_addrlen) < 0) {
        perror("bind");
        exit(1);
}
freeaddrinfo(res);
if (listen(sd, 5) < 0) {
        perror("listen");
        exit(1);
}
sktlen = sizeof (struct sockaddr_storage);
if ((sd1 = accept(sd, &sin, &sktlen)) < 0) {
        perror("accept");
        exit(1);
}
if ((cnt = recv(sd1, buf, sizeof buf, 0)) < 0) {
        perror("recv");
        exit(1);
}
. . .
```

参考：IPv4/v6対応：クライアント側の例 (1)

```
struct addrinfo hints, *res;
char *host, *serv;
int sd, err;

// hostが指す領域にホスト名(e.g., “www.ics.keio.ac.jp” or
//     “131.113.126.136”)を格納しておく.
// servが指す領域にサービス名(e.g, “http” or “80”)を格納しておく.

memset(&hists, 0, sizeof hints);
hints.ai_socktype = SOCK_STREAM;          // TCPを使用

if ((err = getaddrinfo(host, serv, &hints, &res)) < 0) {
        fprintf(stderr, “getaddrinfo: %s¥n”, gai_strerror(err));
        exit(1);
}
// 次ページへ
```

参考：IPv4/v6対応：クライアント側の例 (2)

```
// 前ページから

if ((sd = socket(res->ai_family, res->ai_socktype,
                                    res->ai_protocol)) < 0) {
        perror("socket");
        exit(1);
}

if (connect(sd, res->ai_addr, res->ai_addrlen) < 0) {
        perror("connect");
        exit(1);
}

// buf[]に送信するデータを格納する.
if ((cnt = send(sd, buf, sizeof buf, 0)) < 0) {
        perror("send");
        exit(1);
}


. . .
```

演習

- テキスト6章の演習問題2を解いてみよう.
  - クライアント側ではホスト名でサーバを指定し，getaddrinfo( ) を利用してサーバのIPアドレスを得るようにする.