ネットワークプログラミング講座
-UNIX, Windows & Java 環境における-

石井 秀治(NEC)
shuji@ccs.mt.nec.co.jp
日比野 洋克(オレンジソフト)
nonki@orangesoft.co.JP

目的

- ネットワークプログラムとはどのようなものであるかを理解する(初心者)
- WINDOWSのプログラム開発環境および，ネットワークプログラミングの基礎を理解する(UNIXプログラマ)
- UNIXのプログラム開発環境および，ネットワークプログラミングの基礎を理解する(WINDOWSプログラマ)

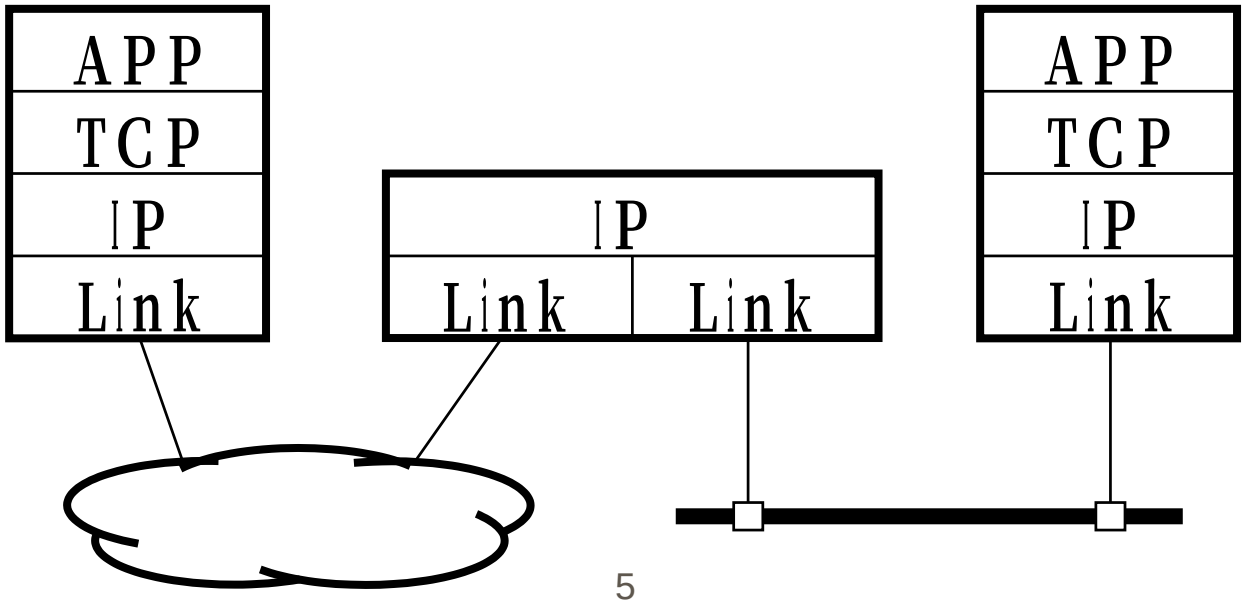
内容

- インターネット概要(石井)
- ネットワークプログラミング
  - UNIX編 (石井)
  - WINDOWS編(日比野)
- 実例紹介(日比野・石井)
  - SMTP, POP
- まとめ

インターネット概要

インターネット

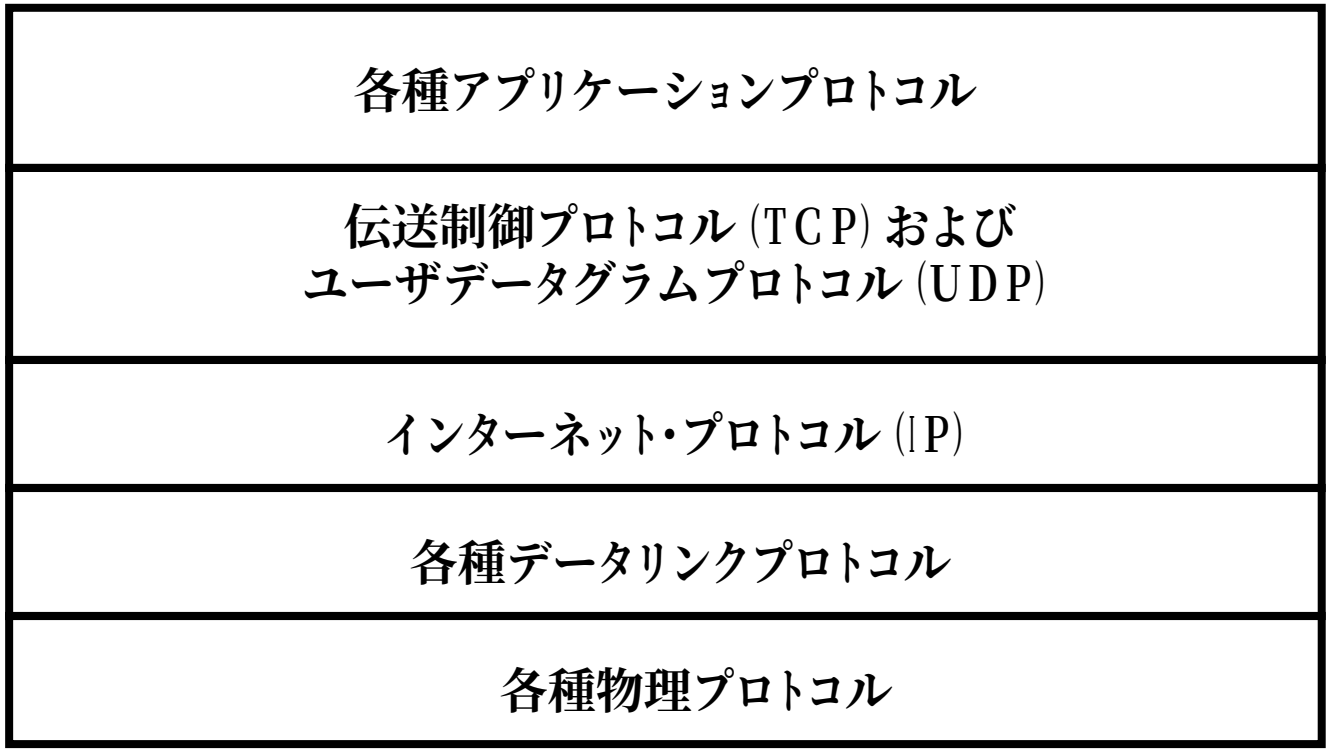
- 異なったメディアを統合して論理的なネットワークに見せる技術
  - IPアドレス
  - 経路制御



NEC

TCP/IP

- TCP, UDP とIPを中心にしたプロトコルスタック



IPアドレスとホスト名

- IP アドレス: 固定長の論理アドレス
  - IPv4: 32ビット, IPv6：128ビット
  - ホストのネットワークインターフェースを識別
  - “192.168.0.3”
- ホスト名: ホストを識別する名前
  - 機械, プログラムはアドレス(数字)を好む
  - ユーザ(人間)は名前を好む
  - www.nic.ad.jp

トランスポート

- IPではデータ配送に信頼性がない
  - パケットが届かないかもしれない
  - 送信した順序と違う順序で受信するかもしれない
  - パケットが重複して到着するかもしれない
- 上位層(トランスポート)で信頼性を確保
  - TCP
  - UDP

TCP

- 信頼性のあるデータ配送
  - ストリーム指向
  - バーチャルサーキット(コネクション型)
  - バッファ付き転送
  - 全二重
- ユーザが下位プロトコルを意識せずに通信を行う

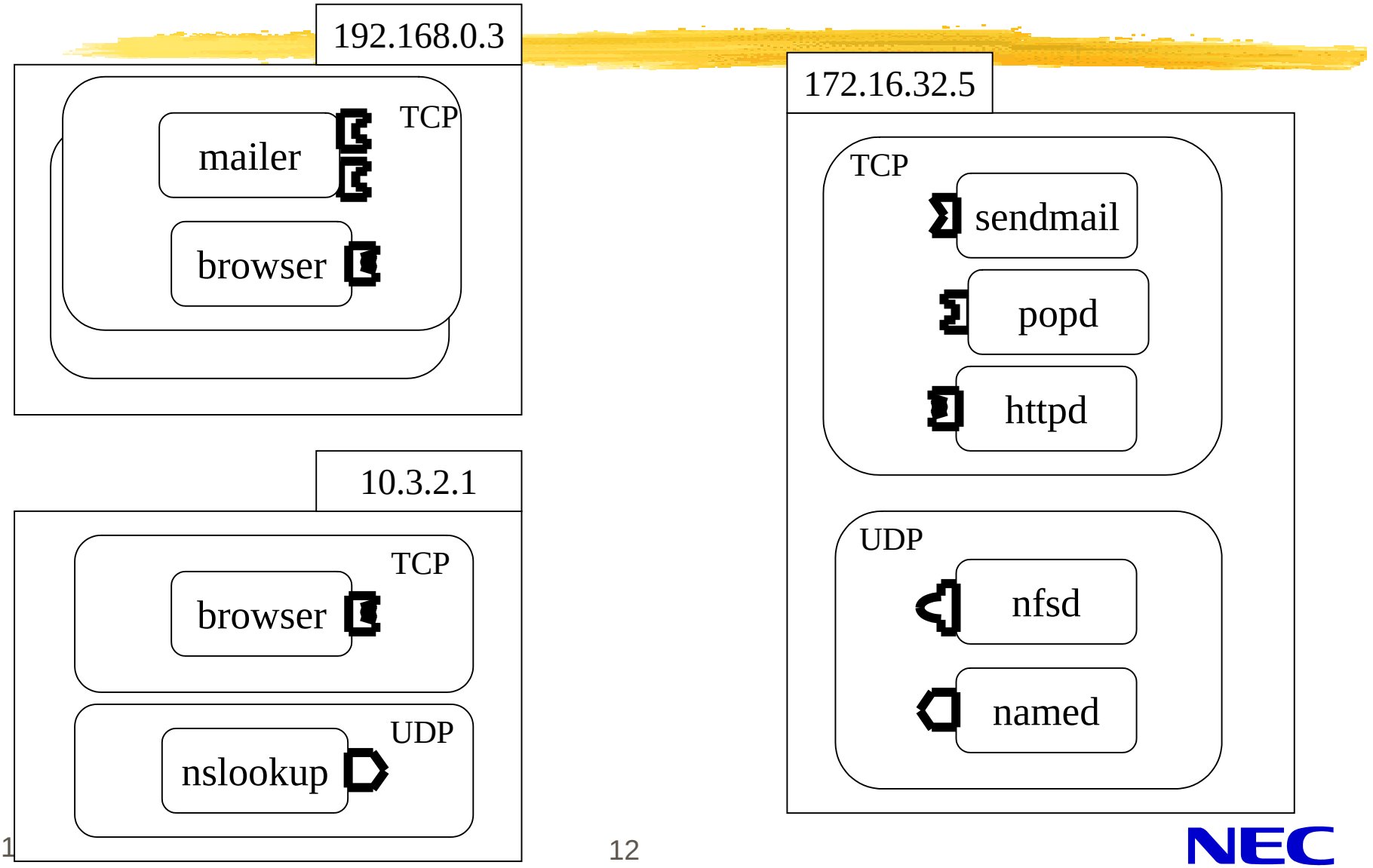
UDP

- 信頼性の無いデータグラム配送
  - ＝ IPパケット+ポート番号+チェックサム
- ユーザ(プログラマ) にプロトコルを開放
  - NFS
  - TFTP
  - DHCP
  - DNS
  - ...

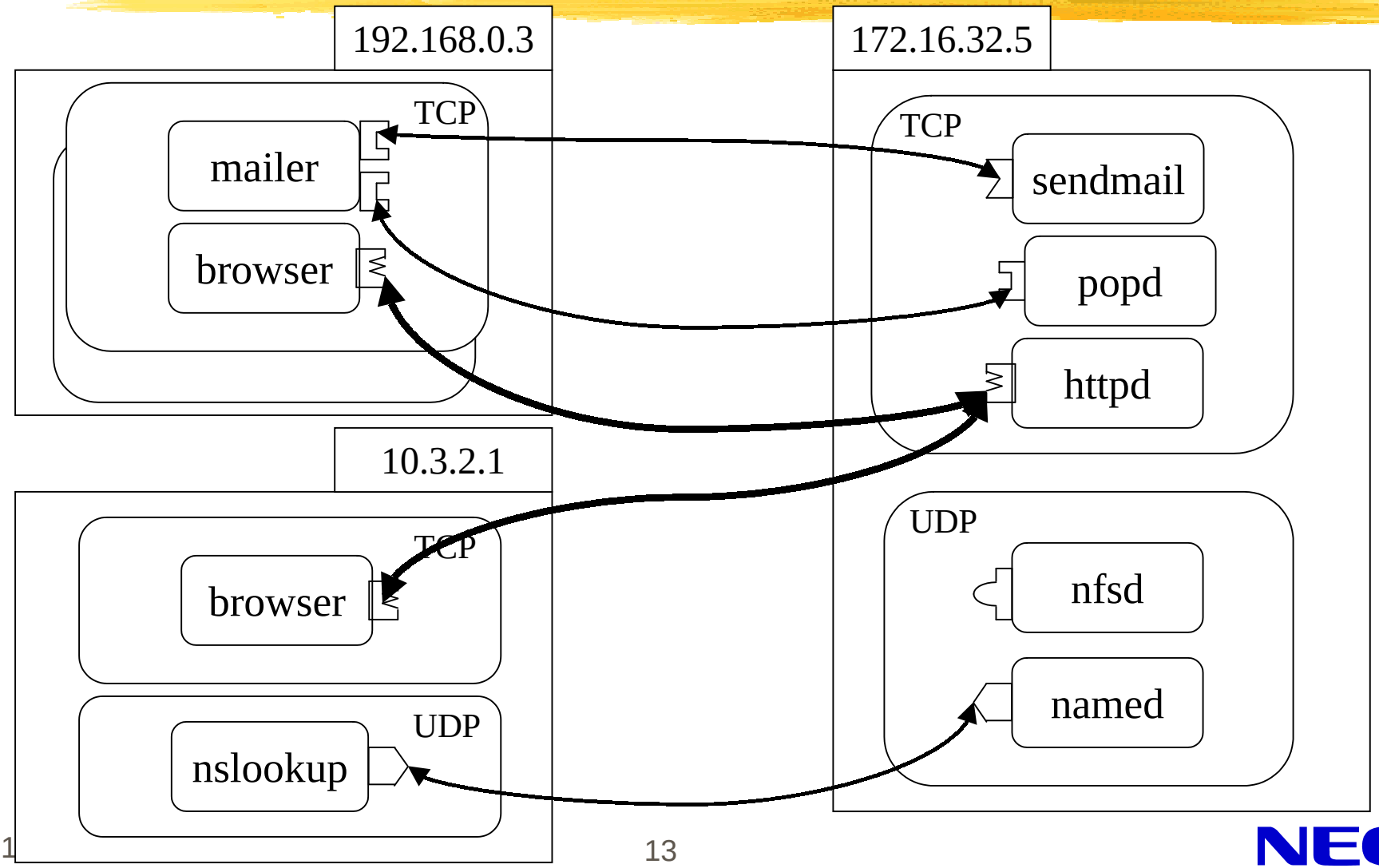
ポート

- ホスト内での通信端点
  - 同一ホスト内のサービスを区別
- ポート番号
  - ポートを区別するための16ビットの正の整数
- トランスポートごとに独立
  - TCP の80番と UDPの80番は異なる

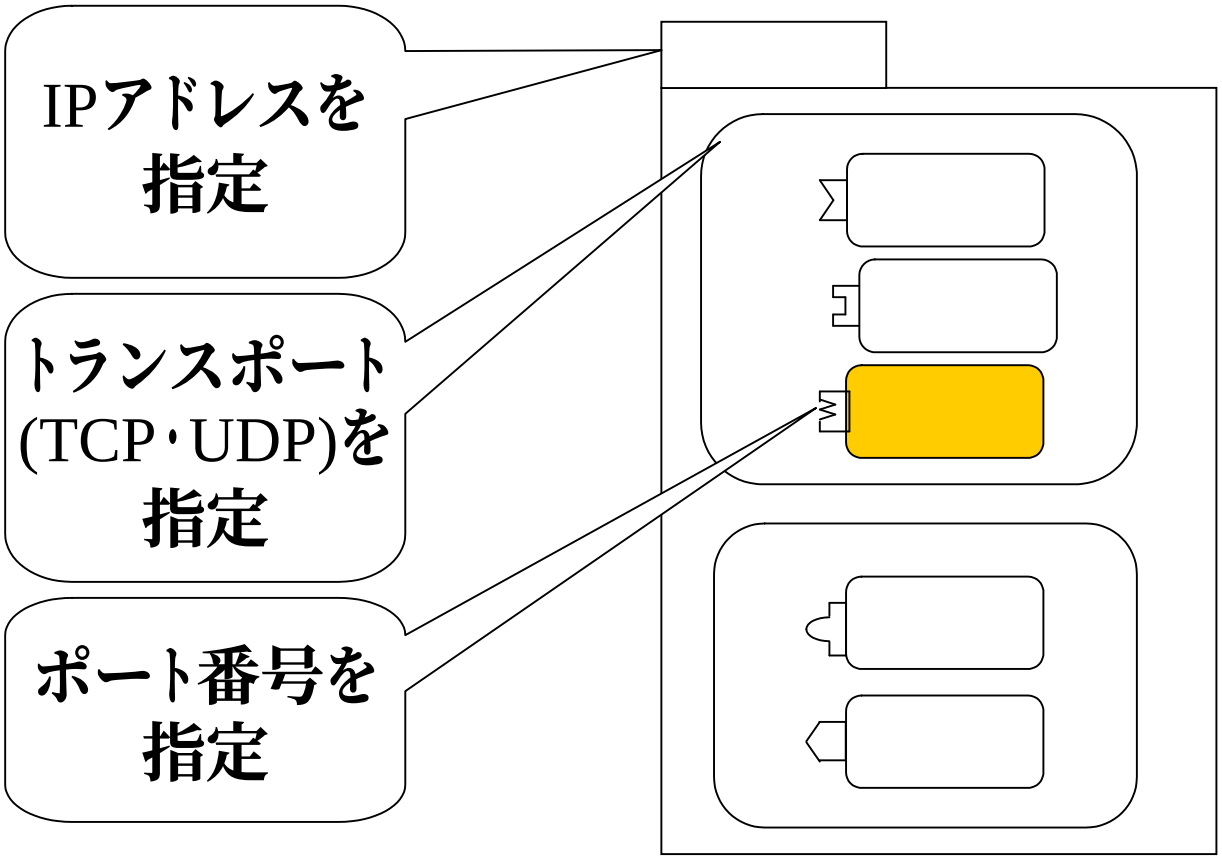
ポート(2)


192.168.0.3
TCP
mailer
browser
10.3.2.1
TCP
browser
UDP
nslookup
172.16.32.5
TCP
sendmail
popd
httpd
UDP
nfsd
named

ポート(3)


192.168.0.3
TCP
mailer
browser
10.3.2.1
TCP
browser
UDP
nslookup
172.16.32.5
TCP
sendmail
popd
httpd
UDP
nfsd
named

通信端点の指定方法



プロセス間通信 API

- Berkeley socket
- TLI/XLI
- sunrpc
- winsock
- RMI
- WinInet
- :

UNIX 編

socket interface

- BSD 系UNIXでは，socket と呼ばれる一連のシステムコール群を用いて通信する
- 特徴
  - アドレス形式に依存しない
  - サーバ･クライアントモデル
  - プロトコル非依存
    - TCP/IP
    - XNS
    - :

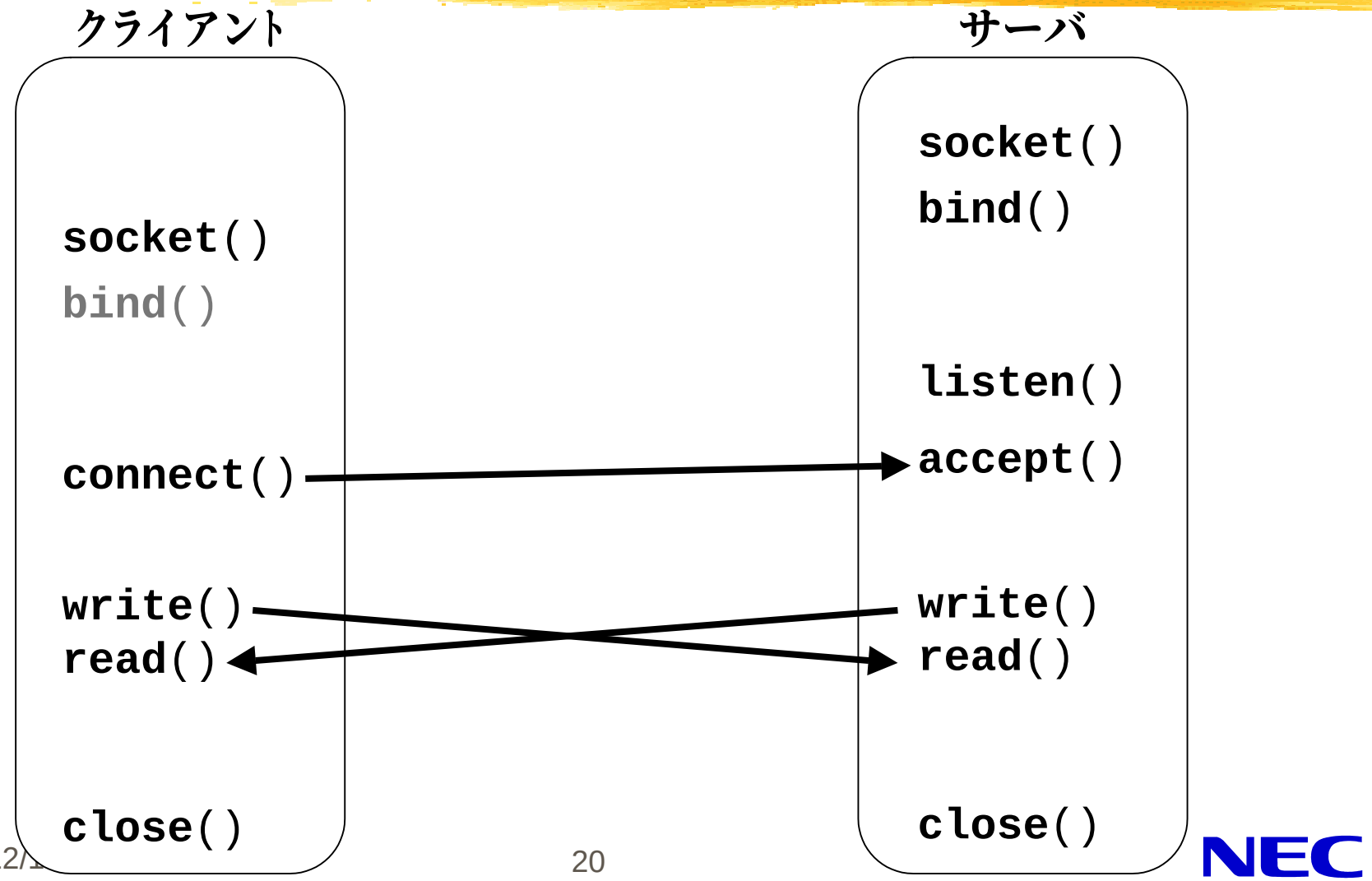
socket interface (2)

socket の生成	`socket`
名前付け	`bind`
接続受理準備	`listen`
接続要求	`connect`
接続要求受理	`accept`

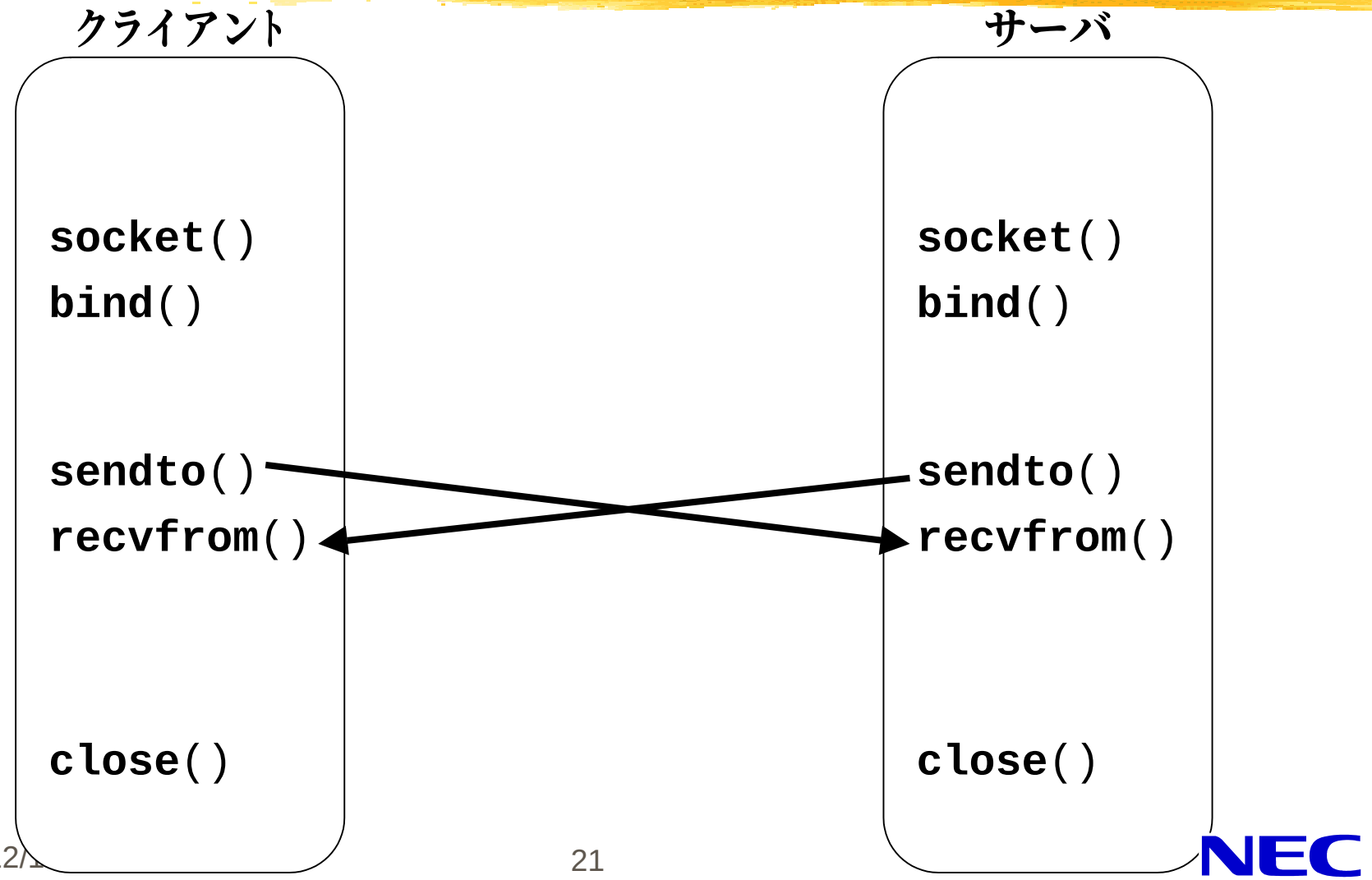
socket interface (3)

データ転送	`read, write`
	`recv, send`
	`recvfrom, sendto`
切断	`shutdown, close`
その他	`ioctl,socketpair,` `setsockopt,getsockopt,` `getsockname,getpeername`

socket を用いた通信(TCP)



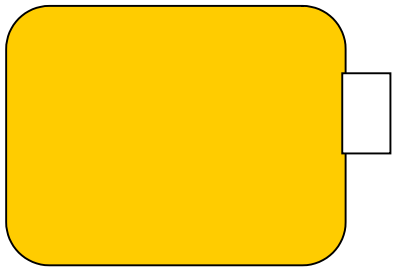
socket を用いた通信(UDP)

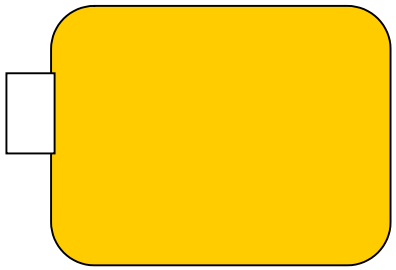


socket の生成

- 通信の口(socket)を作る

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

int socket(proto_family, type, proto);
int sd, proto_family, type, proto;
```





socket の生成 (2)

proto_family: プロトコルファミリ

PF_INET　　インターネットドメイン

他に PF_UNIX, PF_ISO, PF_INET6 などがある

type:　通信のタイプ

SOCK_STREAM　ストリーム(PF_INET の場合, TCP)

SOCK_DGRAM　データグラム(PF_INET の場合, UDP)

proto: プロトコル番号

0 にすると適当な番号に(システムが) 割り当ててくれる

sd:　　socket 記述子

NEC

socket の生成 (3)

- TCP の socket を生成

```
sd = socket(PF_INET, SOCK_STREAM, 0);
```

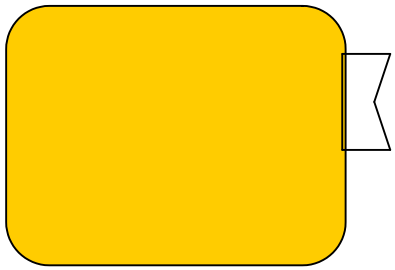
- UDP の socket を生成

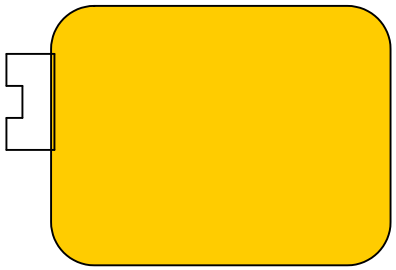
```
sd = socket(PF_INET, SOCK_DGRAM, 0);
```

名前付け

- socket に名前をつける
  - 通信端点 (IPアドレス,トランスポート,ポート番号)

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

int bind(sd, name, namelen);
int sd, namelen;
struct sockaddr *name;
```





名前付け (2)

sd	socket 記述子
name	名前へのポインタ
namelen	名前の長さ

- *name* は sockaddr_in 構造体へのポインタとなる

sockaddr_in 構造体

- IPアドレスとポート番号
  - トランスポートは，socket 生成時に指定済

```
/usr/include/netinet/in.h
struct sockaddr_in {
    u_char  sin_len;     /* 構造体の長さ */
    u_char  sin_family; /* PF_INET */
    u_short sin_port;    /* ポート番号 */
    struct  in_addr sin_addr; /*アドレス*/
    char    sin_zero[8]; /*詰物*/
};

struct in_addr {
    u_long  s_addr;      /* アドレス */
};
```

sockaddr_in 構造体 (2)

- アドレス，ポート番号はネットワークバイトオーダで指定する

```
struct sockaddr_in server;
server.sin_len = sizeof(server);
server.sin_family = PF_INET;
server.sin_port = htons(80);
server.sin_addr = htonl(0x0a030201);/*10.3.2.1*/
```

アドレス，ポート番号

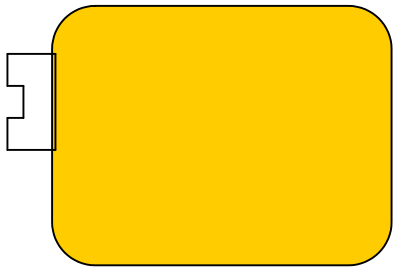
- アドレスに INADDR_ANY を指定すると，自ホストアドレスと解釈される
- ポート番号に 0 を指定するとシステムが適当なポート番号を割り当ててくれる
- 通常，クライアントは bind() を実行しなくてよい
  - システムが適当なポート番号を割り当ててくれる

接続受理の準備

- サーバは接続要求受理の準備をおこなう

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

int listen(sd, maxqueue);
int sd, maxqueue;
```

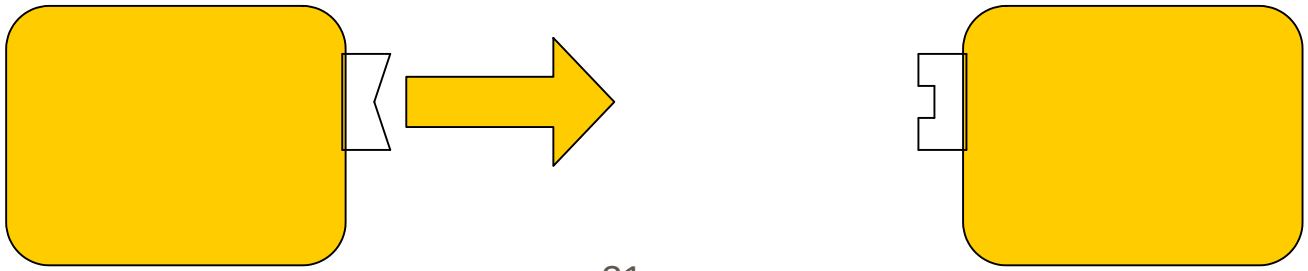
sd socket 記述子
maxqueue 受信受付キューの長さ



接続要求

- クライアントは，接続を要求する

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

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



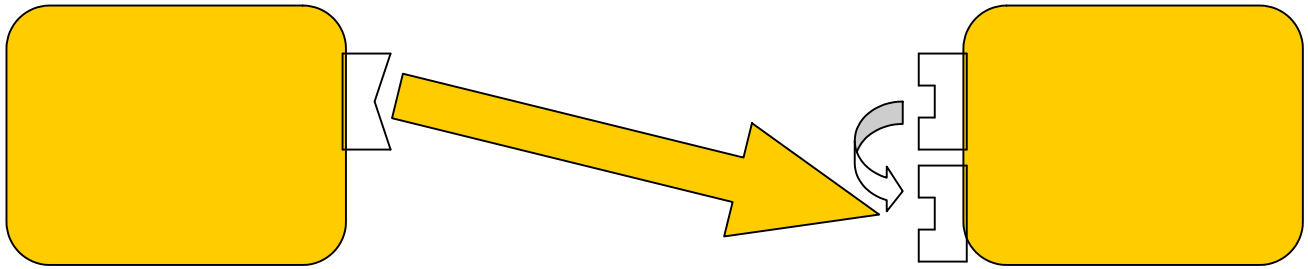
接続要求 (2)

sd	socket 記述子
name	接続相手の名前へのポインタ
namelen	接続相手の名前の長さ

接続要求の受理

- サーバは，接続要求を受け付ける

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

int accept(sd, name, namelen);
int     sd, *namelen;
struct sockaddr *name;
```



NEC

接続要求の受理 (2)

sd	socket 記述子
name	相手の名前を格納する領域へのポインタ
namelen	相手の名前の長さを格納する領域へのポインタ

- **accept** は新しい socket を作成する.
  - 以降の通信は, 新しいsocket で行う

これで, 接続が確立

NEC

データ転送

- UNIXの通常の入出力と同じ
  - `read`, `write` を使う
    - `recv`, `send`, …
  - クライアントは, socket 記述子
  - サーバは `accept()` の戻り値

データ転送-TCP

- データ受信用システムコール

```
#include <unistd.h>

ssize_t read(sd, buf, buflen);


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

ssize_t recv(sd, buf, buflen, flags);
int sd, flags;
size_t  buflen;
void    *buf;
```

データ転送-TCP (2)

- データ送信用システムコール

```
#include <unistd.h>

ssize_t write(sd, buf, buflen);


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

ssize_t send(sd, buf, buflen, flags);
int sd, flags;
size_t  buflen;
void    *buf;
```

データ転送-TCP (3)

sd	socket 記述子
buf	データバッファへのポインタ
buflen	データ長
flags	データ転送フラグ

MSG_OOB	帯域外データの処理
MSG_PEEK	バッファ中のデータを覗く
MSG_DONTROUTE	ルーティングをバイパスする
MSG_EOR	レコード境界を示す
MSG_EOF	データ転送の終了を示す
MSG_WAITALL	データが全部揃うまで待つ

データ転送-UDP

データ転送用システムコール

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

ssize_t sendto(sd, buf, buflen, flags, to,
  tolen);
ssize_t recvfrom(sd, buf, buflen, flags,
  from, fromlen);

int             sd, flags, tolen, *fromlen;
void            *buf;
ssize_t         buflen;
struct sockaddr *to, *from;
```

データ転送-UDP (2)

sd	socket記述子
buf	データバッファへのポインタ
to, from	相手の名前へのポインタ
flags	データ転送フラグ
buflen	データ転送長
tolen	名前の長さ
fromlen	名前の長さへのポインタ

切断

- 送受信の一方向，あるいは両方向のデータ転送の終了

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

int       shutdown(sd, how);
int       sd, how;
```

sd　　socket 記述子
how　　切断方法

0	データの受信を終了する
1	データの送信を終了する
2	データの送受信を終了

切断 (2)

- 接続を切断し，ソケットを消滅

```
#include <unistd.h>

int close(sd);
int sd;
```

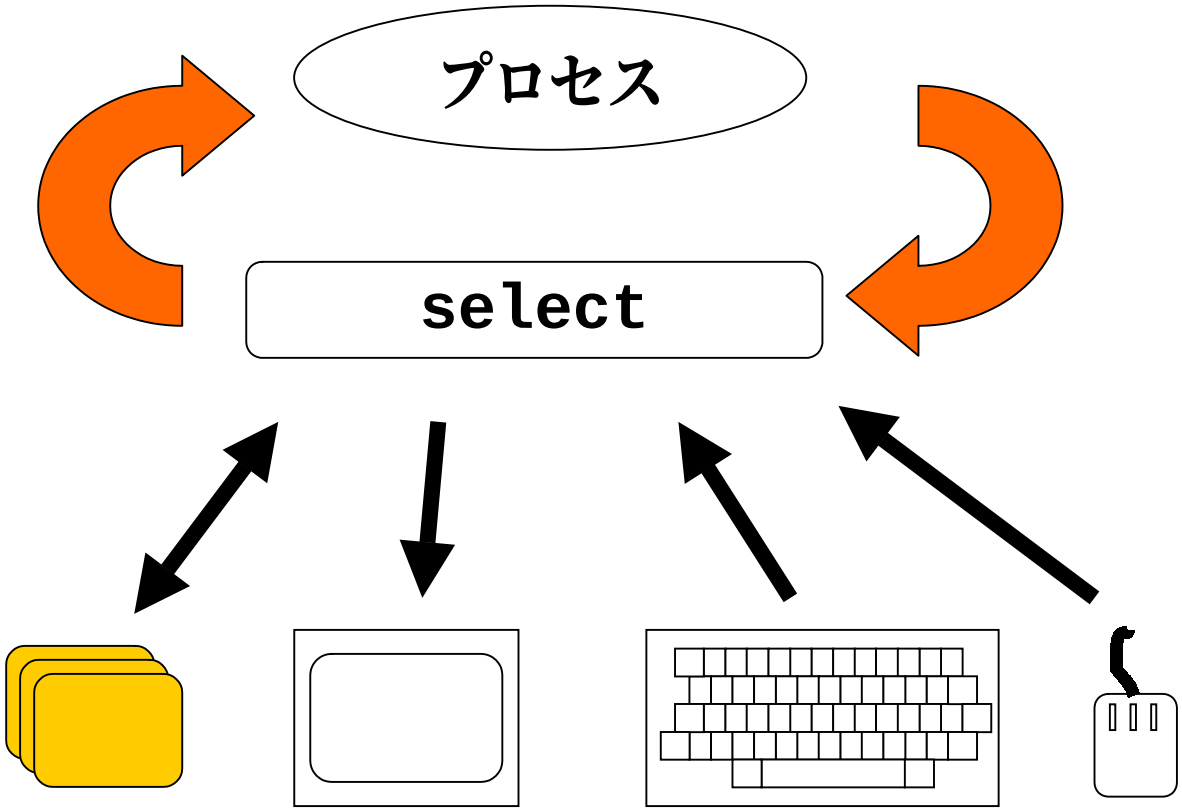
sd　　socket 記述子

- **shutdown を利用せず close してよい**

入出力の多重化

- `accept`, `read`, `recv` などのシステムコールは, データを受信するまでブロックしてしまう.
- 複数の入出力を同時に見張りたいときには, `select` システムコールを使う

select



select システムコール

- 複数デバイスの入出力待ち

```
#include <sys/types.h>
#include <sys/time.h>
#include <sys/select.h>
int select(nfds, rfds, wfds, efds, tout);
int nfds;
fd_set *rfds, *wfds, *efds;
struct timeval *tout;
```

nfds	調べるファイル記述子の数
rfds	入力用ファイル記述子の指定(ポインタ)
wfds	出力用ファイル記述子の指定(ポインタ)
efds	例外用ファイル記述子の指定(ポインタ)
tout	タイムアウトの指定(ポインタ;NULL: 無限に待つ)
返値	入出力可能なファイル記述子数

select システムコール (2)

- **FD_SETSIZE マクロ**
  - システムがサポートする記述子の最大数
- **ビット操作マクロ**

  FD_SET(*fd, &fdset*); 記述子 *fd* を *fdset* にセットする

  FD_CLR(*fd, &fdset*); 記述子 *fd* を *fdset* からクリアする

  FD_ISSET(*fd, &fdset*); *fd* が *fdset* にセットされている場合, 0 以外を返し, セットされていない場合 0 を返す

  FD_ZERO(*&fdset*); 記述子の集合を空にする

```
int fd;
fd_set fdset;
```

select システムコール (3)

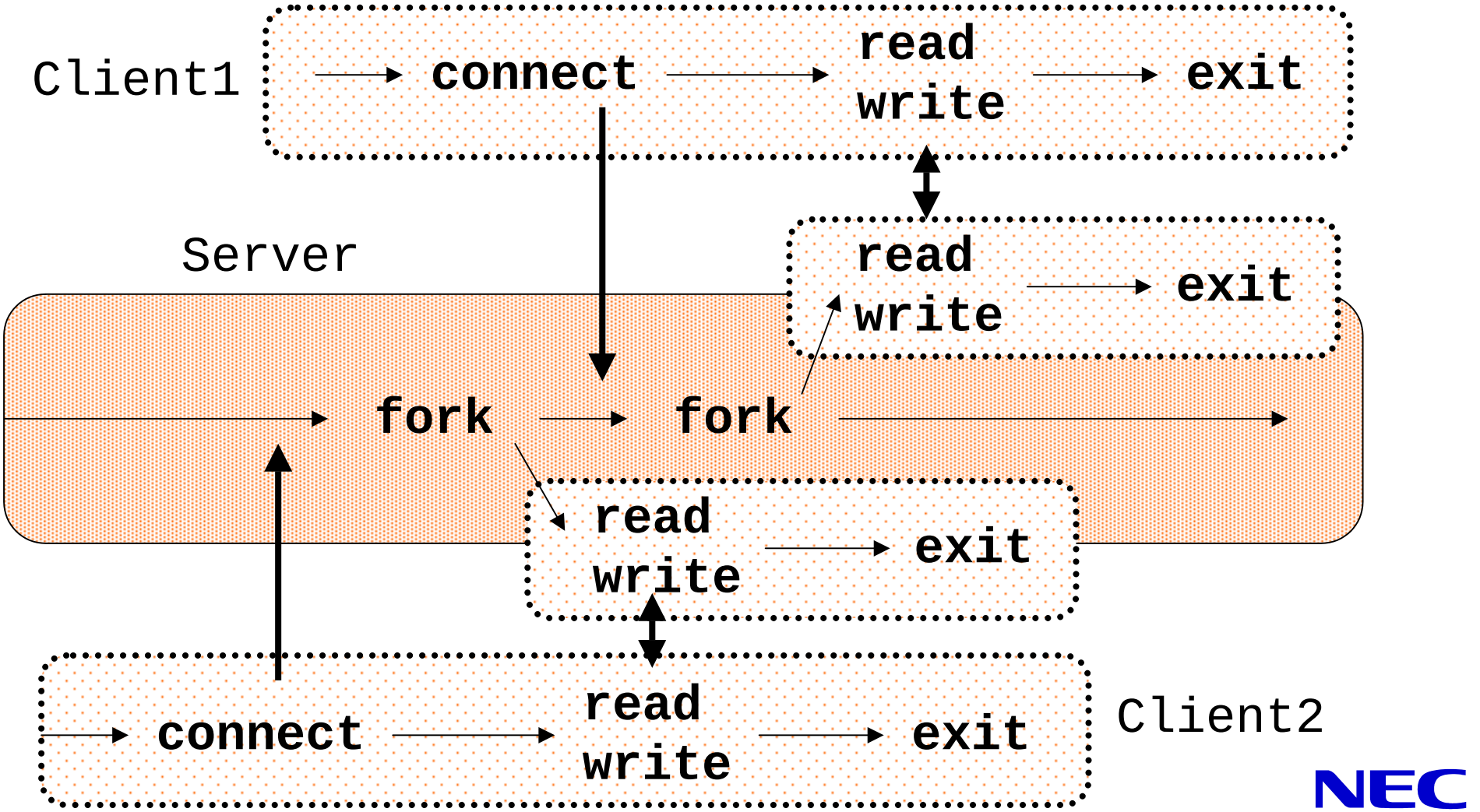
- 使用例: 標準入力と socket sd の入力を待つ

```
fd_set rd;
for (;;) {
  FD_ZERO(&rd);
  FD_SET(fileno(stdin), &rd);
  FD_SET(sd, &rd);
  select(FD_SETSIZE, &rd, NULL, NULL, NULL);
  if (FD_ISSET(fileno(stdin), &rd)) {
   /* stdin の処理*/
  }
  if (FD_ISSET(sd, &rd)) {
   /* sd の処理*/
  }
}
```

クライアント・サーバモデル

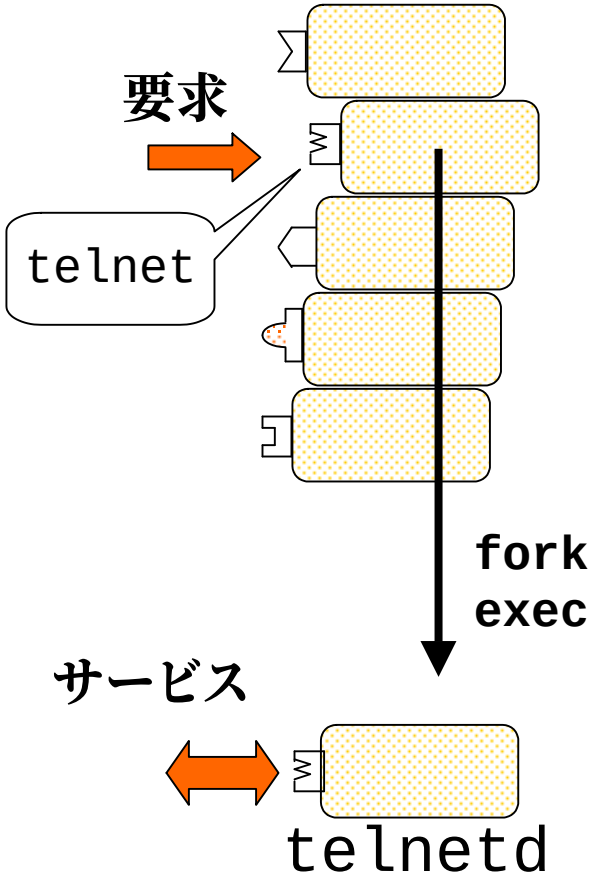
- ネットワークアプリケーションの基本的なモデル
- サーバ
  - 公示したサービスに対して要求を待ち，要求者に対してサービスを提供する（デーモンと呼ばれることも）
- クライアント
  - サーバに対してサービスを要求し，サーバからサービスを受ける

クライアント・サーバモデルの例



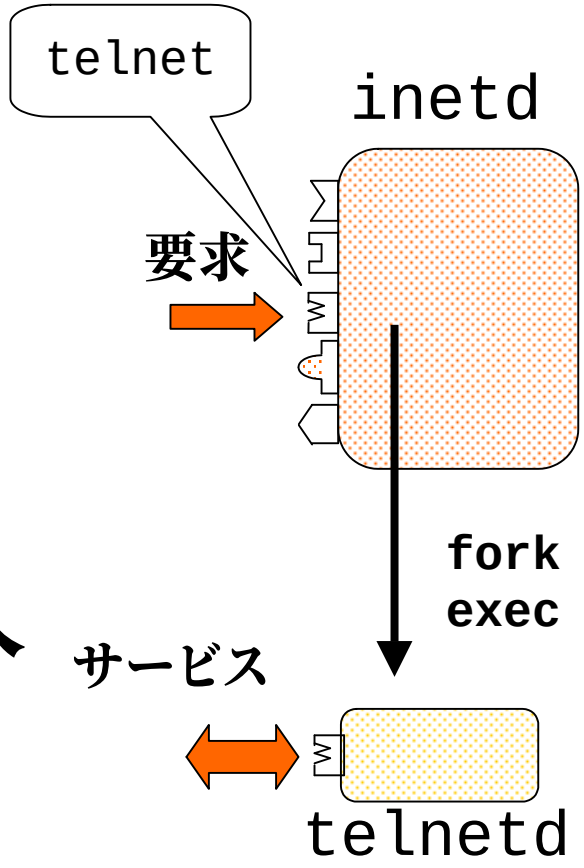
UNIX でのサーバ

- サービスの公示
  - /etc/services 中にサービス名, ポート番号, プロトコル番号で示してある.
  - サーバは公示したポートを監視する
- サービスの要求
  - クライアントはサーバが存在するホストのインターネットアドレスとポート番号の組でサービスの要求を出す



inetd

- スーパサーバ
  - 提供するサービスがたくさんあると，プロセスが，たくさん必要
    - 資源の無駄
  - 一括してポートを監視
  - /etc/inetd.conf
- クライアントとの通信は標準入出力を使う
  - サーバプログラムが (少し) 楽



簡単なサーバの例: daytimed.c

```
 1  /*
 2   * daytimed.c
 3   */
 4
 5  #include <stdio.h>
 6  #include <string.h>
 7  #include <unistd.h>
 8  #include <sys/types.h>
 9  #include <sys/socket.h>
10  #include <sys/time.h>
11  #include <sys/wait.h>
12  #include <signal.h>
13  #include <netinet/in.h>
14  #include <netdb.h>
15  #include <time.h>
16
```

daytimed.c (2)

```
17  static void print_daytime(int sd)
18  {
19          char strbuf[64];
20          struct timeval tv;
21
22          gettimeofday(&tv, NULL);
23          strftime(strbuf, sizeof(strbuf), "%a %b %d %T %Y¥r¥n",
24                   localtime(&tv.tv_sec));
25          write(sd, strbuf, strlen(strbuf));
26          exit(0);
27  }
28
29  void sigchld(int sig)
30  {
31          (void)wait(NULL);
32  }
33
```

daytimed.c (3)

```
34  main(void)
35  {
36          int sd, snew;
37          struct sockaddr_in sin;
38
39          signal(SIGCHLD, sigchld);
40
41          if ((sd = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
42                  perror("socket");
43                  exit(1);
44          }
45
46          sin.sin_len = sizeof(sin);
47          sin.sin_family = PF_INET;
48          sin.sin_port = htons(8888);
49          sin.sin_addr.s_addr = INADDR_ANY;
50
```

daytimed.c (4)

```
51              if (bind(sd, (struct sockaddr *)&sin, sizeof(sin)) < 0) {
52                      perror("bind");
53                      exit(2);
54              }
55
56              listen(sd, 5);
57              for (;;) {
58                      if ((snew = accept(sd, NULL, NULL)) < 0) {
59                              perror("accept");
60                              exit(3);
61                      }
62                      if (fork() != 0) {
63                              /* 親 */
64                              close(snew);
65                      } else {
66                              /* 子 */
67                              close(sd);
68                              print_daytime(snew);
69                      }
70              }
71      }
72      /* ----- end of daytimed.c ----- */
```


NEC

ライブラリ

- getXbyY
  - ホスト
  - ネットワーク
  - プロトコル
  - サービス
- インタネットアドレスの操作
- バイトオーダ変換

gethostbyname, gethostbyaddr

- ホスト名⇒IP アドレス

```
#include <netdb.h>
struct hostent *gethostbyname(name);
char *name;
```

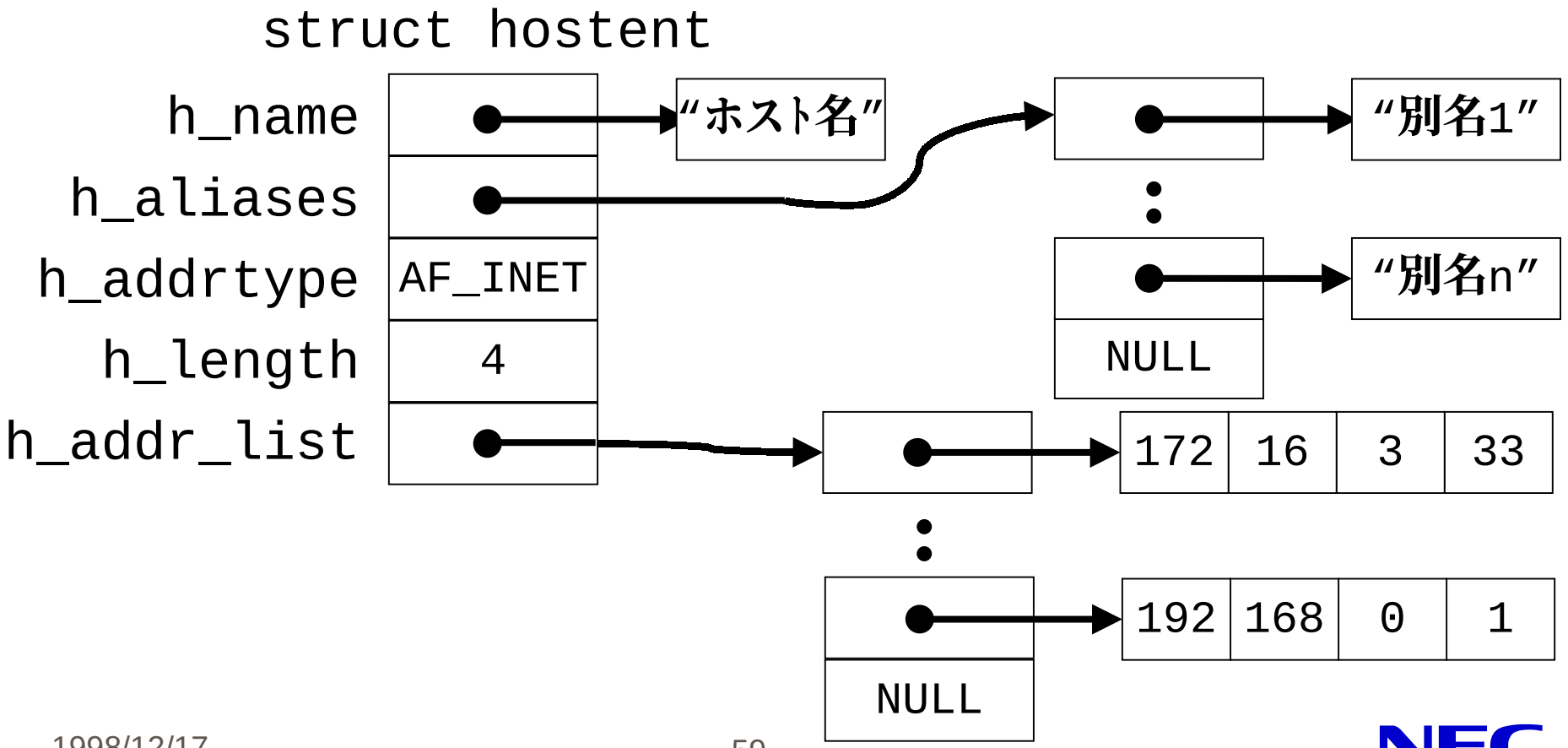
- IPアドレス⇒ホスト名

```
#include <netdb.h>
struct hostent *gethostbyaddr(addr, len,
  type);
char *addr;
int len;
int type;
```

struct hostent

```
struct hostent {
  char     *h_name;     /*ホスト名*/
  char     **h_aliases;/*ホストの別名*/
  int      h_addrtype; /*アドレスタイプ(AF_INET)*/
  int      h_length;   /*アドレス長*/
  char     **h_addr_list; /*アドレスのリスト*/
};
```

struct hostent (2)



getnetbyname, getnetbyaddr

- ネットワーク名⇒ネットワークアドレス

```
#include <netdb.h>
struct netent *getnetbyname(name);
char *name;
```

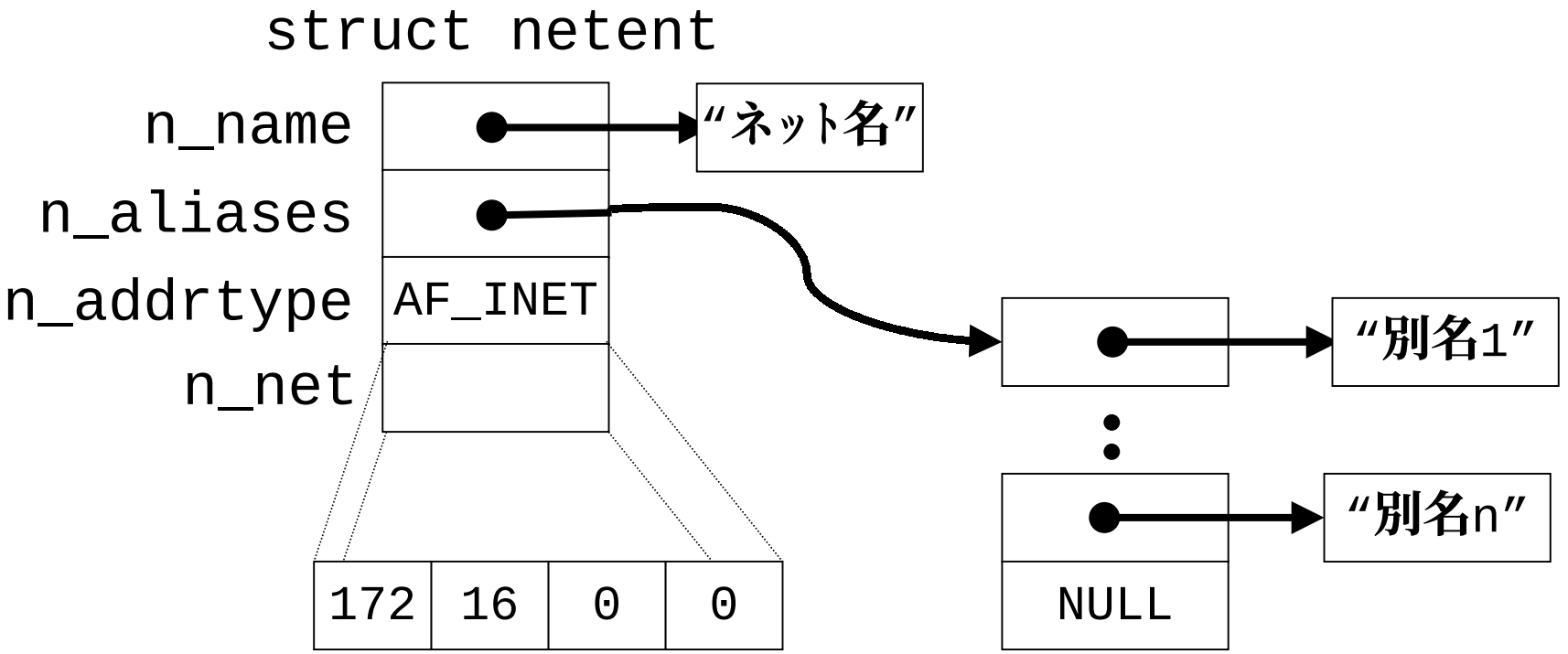
- ネットワークアドレス⇒ネットワーク名

```
#include <netdb.h>
struct netent *getnetbyaddr(net, type);
long net;
int type;
```

struct netent

```
struct netent {
  char     *n_name;      /*ネットワーク名*/
  char     **n_aliases;  /*別名のリスト*/
  int      n_addrtype;   /*アドレスタイプ*/
  unsigned long n_net;   /*ネットワークアドレス*/
};
```

struct netent (2)



getprotobyname, getprotobynumber

- サービス名⇒プロトコル番号

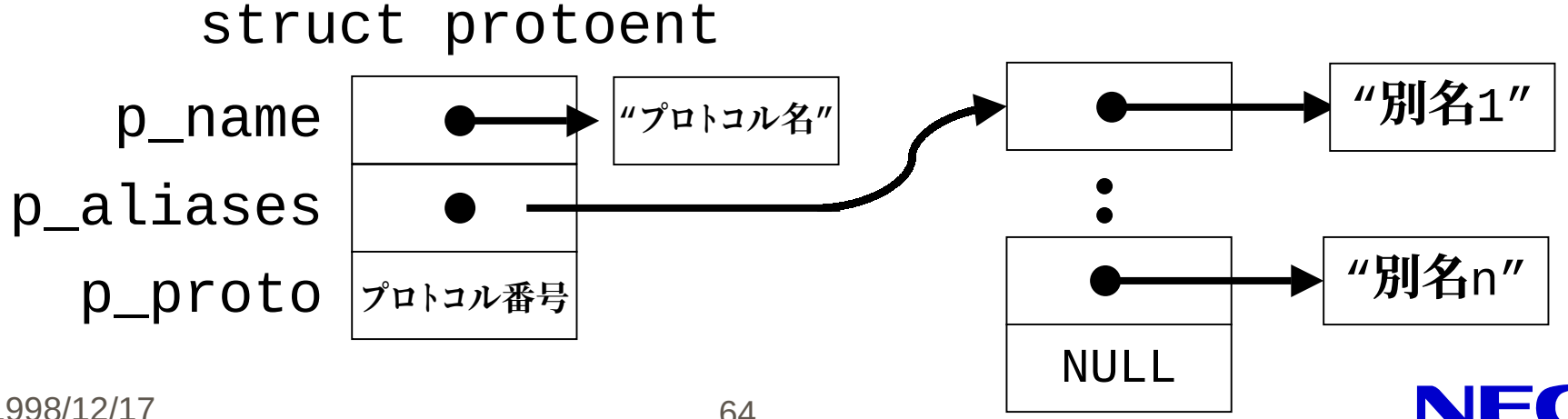
```
#include <netdb.h>
struct protoent *getprotobyname(name);
char *name;
```

- プロトコル番号⇒サービス名

```
#include <netdb.h>
struct protoent *getprotobynumber(proto);
int proto;
```

struct protoent

```
struct protoent {
  char     *p_name;      /*プロトコル名*/
  char     **p_aliases;  /*別名*/
  int      p_proto;      /*プロトコル番号*/
};
```



getservbyname, getservbyport

- サービス名⇒ポート番号

```
#include <netdb.h>

struct servent *getservbyname(name, proto);

char *name;
char *proto;
```

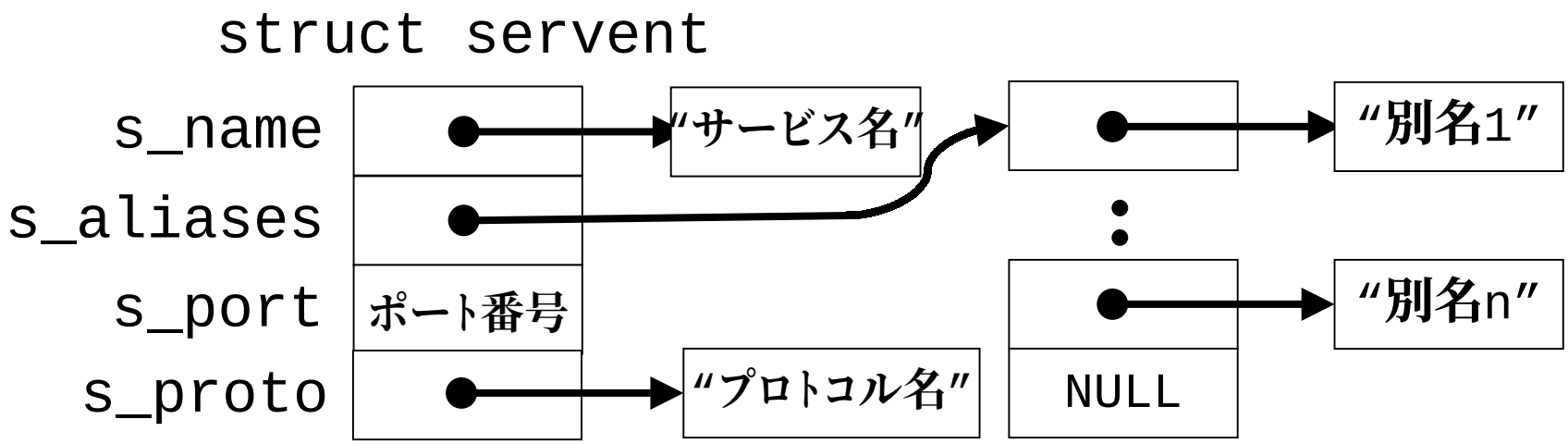
- ポート番号⇒サービス名

```
#include <netdb.h>

struct servent *getservbyport(port, proto);

int port;
char *proto;
```

struct servent

```
struct servent {
  char     *s_name;      /*サービス名*/
  char     **s_aliases;  /*別名のリスト*/
  int      s_port;       /*ポート番号*/
  char     *s_proto;     /*プロトコル名*/
};
```

struct servent (2)



インターネットアドレスの操作

- アドレス表現⇒IPアドレス

```
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>

int inet_aton(str, addr);

char *str; /* eg: “10.0.0.1” */
struct in_addr *addr;

in_addr_t inet_addr(str);

char *str; /* eg: “10.0.0.1” */
```

インターネットアドレスの操作 (2)

- IPアドレス⇒アドレス表現

```
#include <sys/types.h>
#include <netine/in.h>
#include <arpa/inet.h>

char *inet_ntoa(addr);
struct in_addr *addr;
```

インターネットアドレスの操作 (3)

- アドレス表現⇒ IPアドレス

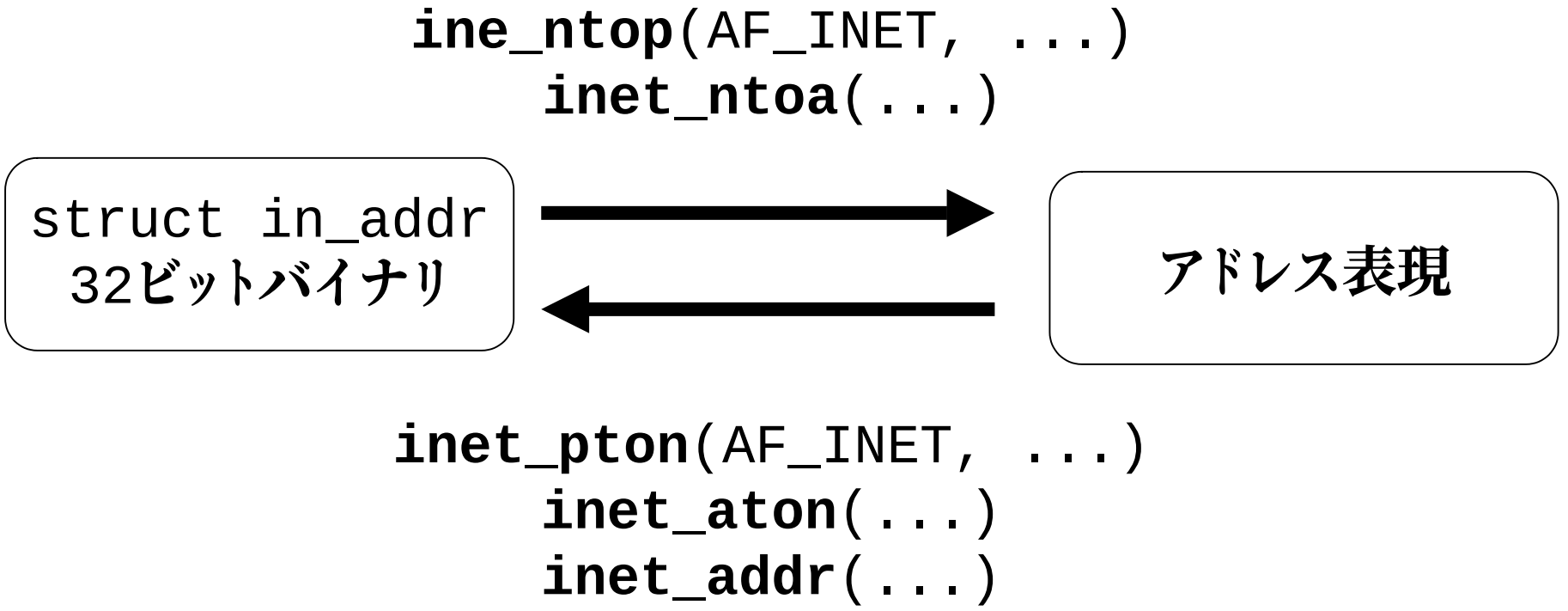
```
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>

int inet_pton(family, str, addr)

int family;  /* AF_INET */
char *str;   /* eg: “10.0.0.1” */
void *addr;
```

インターネットアドレスの操作 (4)

- IPアドレス⇒アドレス表現

```
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>

char *inet_ntop(family, addr, str, len);
int family;  /* AF_INET */
void *addr;
char *str;   /* string */
size_t len;  /* length of str */
```

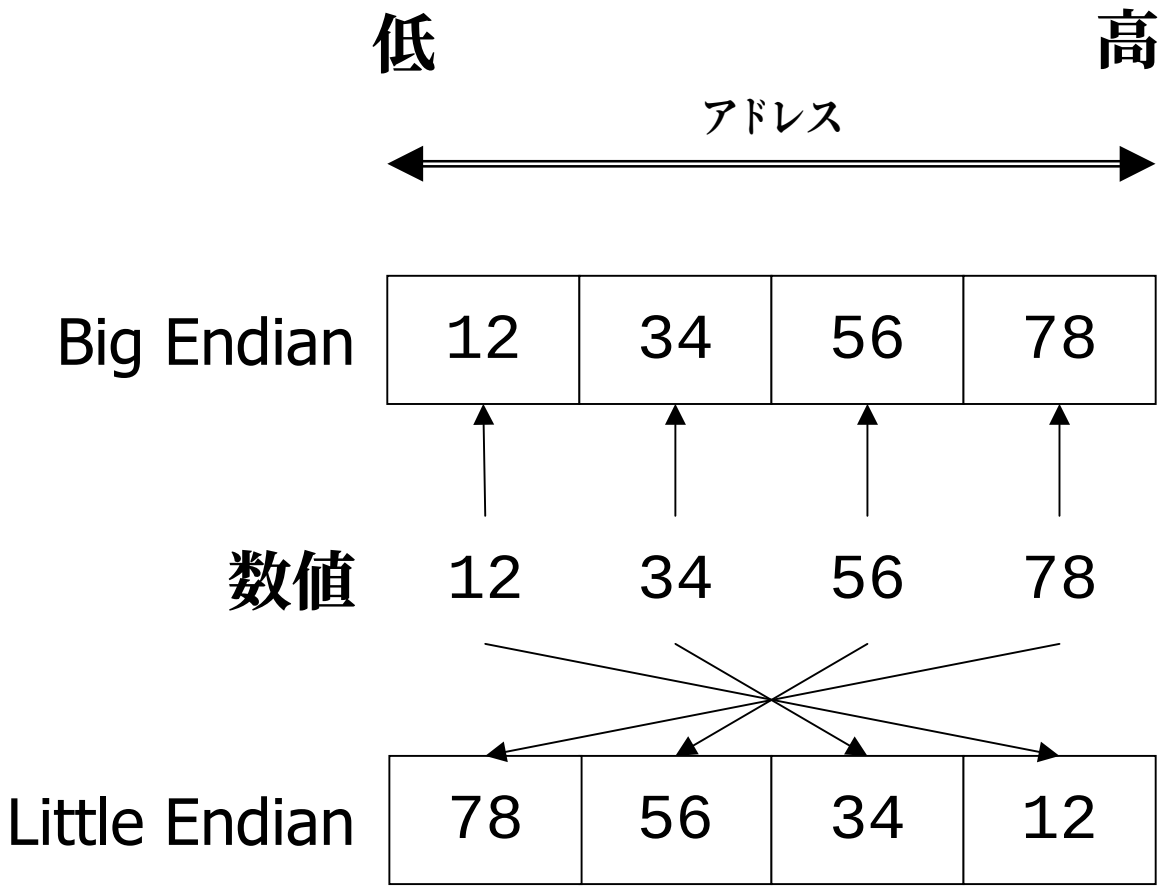
インターネットアドレスの操作 (5)



バイトオーダ

- ネットワークバイトオーダ
  - ネットワーク上を流れるデータのバイト順序
    - すべてのネットワークにおいて同一
  - Big Endian
- ホストバイトオーダ
  - CPU が扱うデータのバイト順序
  - Little Endian: VAX, 80x86, Pentium
  - Big Endian: 680x0, SPARC
  - Config で変わる: MIPS

バイトオーダ (2)


低
高
アドレス
Big Endian
12
34
56
78
数値
12
34
56
78
Little Endian
78
56
34
12

バイトオーダの変換

- ホストバイトオーダ⇒ネットワークバイトオーダ

```
#include <sys/param.h>
u_long htonl(hostlong);
u_long hostlong;

u_short htons(hostshort);
u_short hostshort;
```

バイトオーダの変換 (2)

- ネットワークバイトオーダ⇒ホストバイトオーダ

```
#include <sys/param.h>
u_long ntohl(netlong);
u_long netlong;

u_short ntohs(netshort);
u_short netshort;
```

開発環境

- BSD 系では特にライブラリの指定は不必要
- System V 系では，ライブラリの指定が必要な場合も
  - `-lsocket`
- perl

Windows 編

Windowsの開発環境

- C++(C)
  - Visual C++ (Microsoft)
  - C++ Builder (Imprise)等
- Delphi、Visual Basic等

Network API

- Winsock DLL
- WinInet
- Winsock Control
- Netscape、Internet Explorer

Windowsのライブラリ

- DLL
- COM/OCX
- DDE

DLL

- Dynamic Link library
  - 基本的に拡張子がDLL
  - Windows API(WIN32)もDLLで実装。
- コードの共有化
- バージョンの混乱
  - MFC42.DLL

COM/OCX

- ActiveX Control
- Component Object Model
  - xxxx.dll/xxxx.ocx/xxxx.exe
- ソフトウェア部品

WinInet

- 以下のプロトコルをサポートするDLL
  - HTTP
  - FTP
  - Gopher

Winsock Control

- COM(Component Object Model)
- 多くの処理系で利用可能
  - Visual Basic
  - Visual Basic for Applications(VBA)
    - Excel,Word,Access...

Netscape,Internet Explorer

- IEはWEBブラウザコントロールとして使用可能。
- NetscapeもDDE経由で使用できる。

Windowsのプログラミングスタイル

- メッセージドリブン
  - ウインドウメッセージの処理
  - 定型的な処理はクラスライブラリが隠蔽
    - MFC等
- 多くのAPIがWindowを必要とする。
  - 通知にウインドウメッセージを使う。
    - winsock等

WinSock(1)

- Windows Sockets
- Windows 3.0から実装
  - 当初はベンダがネットワークカードやネットワークソフトウェアにバンドル。
  - Windows95以降はOSに標準バンドル。
- 最新はV2.0
  - wsock32.dll
  - Windows95はV1.1が標準

WinSock(2)

- Berkeley-style API+Windows-style API+その他
  - socket(),connect()…
- Windows-style API
  - Berkeley-style APIに対応するAPI
    - WSASocket(),WSAConnect()…
    - Windows-style APIは、今回のセミナの便宜上の名称。
- その他
  - 初期化API等

初期化API

- WSAStartup()
  - 初期化、バージョンチェック
- WSACleanup()
  - winsockの解放

Windows-style API

- 非同期API
- WSAASync...
- ウインドウメッセージによる通知機構
  - 例：int WSAAsyncSelect(SOCKET s, HWND hWnd, UINT wMsg, long lEvent);
    - イベントマスク(lEvent)で指定したイベントが発生すると、ウインドウ(hWnd)にメッセージwMsgを送信する。

Hungarian notation

- prefixに型をつける
  - hWnd(h=HANDLE)
  - lEvent(l=long)
  - p(pointer)…
- クラス(構造体)のメンバーにはm_を付加
  - m_hWnd,m_nSize

イベント

- FD_ACCEPT　接続確認通知
- FD_CLOSE　ソケットが閉じられたときの通知
- FD_CONNECT　接続結果通知
- FD_OOB　帯域外のデータ到達通知
- FD_WRITE　書き込み準備完了通知
- FD_READ　読み込み準備完了通知

WSAAsyncSelect

- selectの非同期版
- 通知がWindows Message

WSAAsyncGetXByY

- YからXを取得する。
  - WSAAsyncGetHostByAddr (gethostbyaddr)
  - WSAAsyncGetHostByName (gethostbyname)
  - WSAAsyncGetServByName (getservbyname)
  - WSAAsyncGetProtoByName (getprotobyname)
  - WSAAsyncGetProtoByNumber (getprotobynumber)
  - WSAAsyncGetServByName (getservbyname)
  - WSAAsyncGetServByPort (getservbyport)

WSAAsyncGetHostByName

```
HANDLE WSAAsyncGetHostByName (
    HWND hWnd, // window handle
    unsigned int wMsg,  // message
    const char * name,  // [in]host name
    char * buf,  // [out] HOSTENT
    int buflen //  length of buf
    );
```

WSACancelAsyncRequest

- WSAAsyncGetXByYのキャンセル。

エラー処理

- WSAGetLastError()
- WSASetLastError()
- エラーコードはunixとは異なる。
  - winsock.h
  - WSAENETDOWN 等"WSAE"が先頭につく。

例：CSocket

- MFC(Microsoft Foundation Classes)で実装されているSocketクラス
- CAsycnSocket/CSocket
- 非表示のウインドウを使用。
  - Windowsプログラミングの常道
    - WM_TIMER等

CAsyncSocket

- 非同期型
  - メソッド呼び出しはただちに完了する。イベントが発生するとコールバック関数が呼ばれる。
- コールバック関数
  - OnAccept/OnClose
  - OnConnect/OnOutOfBandData
  - OnRecieve()/OnSend()

CSocket

- CAsyncSocketの派生クラス
- 同期型
  - 処理が完了するまではメソッド呼び出しは戻ってこないので、プログラミングは容易。
- CArchiveとの連動

問題点(1)

- マルチスレッド
  - スレッド間の受け渡しはできない。
    - SOCKETハンドルを受け渡して、Attach/Detachを使用する。
    - CWndも同様。

```
SOCKET hSocket;
CSocket socket;
socket.Attach(hSocket);
...
hSocket = socket.Detach();
```

問題点(2)

- VC++6.0ではマルチスレッドで動作させると落ちる。
  - http://support.microsoft.com/support/kb/articles/q193/1/01.asp
    - Q193101 BUG: Unhandled Exception Using MFC Sockets in Visual C++ 6.0
    - 対策も書いてあります。
    - SP1(Service Pack 1)を待ちましょう。

SP1(Service Pack 1)を待ちましょう。

- ...と思ったら直っていなかった。

問題点(3)

- Connect()は同期API(gethostbyaddr()等)を使用している。
  - Connect()は使わないで非同期APIを使用する。

開発スタイル

- a.Berkeley-style API+シングルスレッド
- b.Windows-style API+シングルスレッド
- c.Berkeley-style API +マルチスレッド
- d.Windows-style API +マルチスレッド

- 書籍等ではb,dを勧めることが多いが、私はcがお勧め。

Berkeley-style API+シングルスレッド

- ブロッキング中、他の操作ができない。
  - ウインドウの再描画等。
- Windowsプログラミングでは非現実的

Windows-style API+シングルスレッド

- ブロックしないので、「Berkeley-style API+シングルスレッド」の問題は回避できる。
- 複数のセッションを扱う場合などはプログラムが複雑になる傾向がある。
- unix等からの移植は面倒。
  - 全面書き直しに近い。

Berkeley-style API+マルチスレッド

- バックグラウンドで通信スレッドを動作させる。
- unix等からの移植は簡単。
  - ソースの共有化
  - OpenLDAP,PGP...
- マルチスレッド特有のプログラミングの面倒さがある。
- Windows3.1では使えない。
  - 忘れてしまうのが吉。

Windows-style API+マルチスレッド

- プログラムの構造はシンプルになる。
- CSocketを使うと結果的にそうなる。

Java編

概要

- Javaは標準でSocketサポート
- 文字列の扱いに注意
- java.net.Socket/ java.net.ServerSocket...

プログラミングスタイル

- 同期型
  - 処理が完了するまでは呼び出しは戻ってこない。
- マルチスレッドが必須。
  - Javaはマルチスレッドを標準でサポート

文字列(String)

- Unicode
- 多くの場合、Unicode<->byte[]の変換が必要。

  String#getBytes(String enc)

  String command = “RETR 1¥r¥n”;

  out_stream.write(command.getBytes());

接続

```
int POP3_PORT = 110;
Socket socket = new Socket("hostname",
   POP3_PORT);
```

読み込み/書き込み(1)

```
import java.io.*;
… // 接続処理
BufferedInputStream in_stream = BufferedInputStream(socket.
    getInputStream());
BufferedOutputStream out_stream = BufferedOutputStream
    (socket. getOutputStream());
int ch = in_stream.read();
out_stream.write(ch);
```

読み込み/書き込み(2)

```
import java.io.*;
… // 接続処理
BufferedInputReader reader = BufferedInputReader(new
    InputStreamReader(socket. getInputStream()));
BufferedOutputWriter writer = BufferedOutputWriter(new
    OutputStreamWriter(socket. getOutputStream()));
int ch = reader.read();
int ch = reader.write();
```

Reader(Writer)とStream

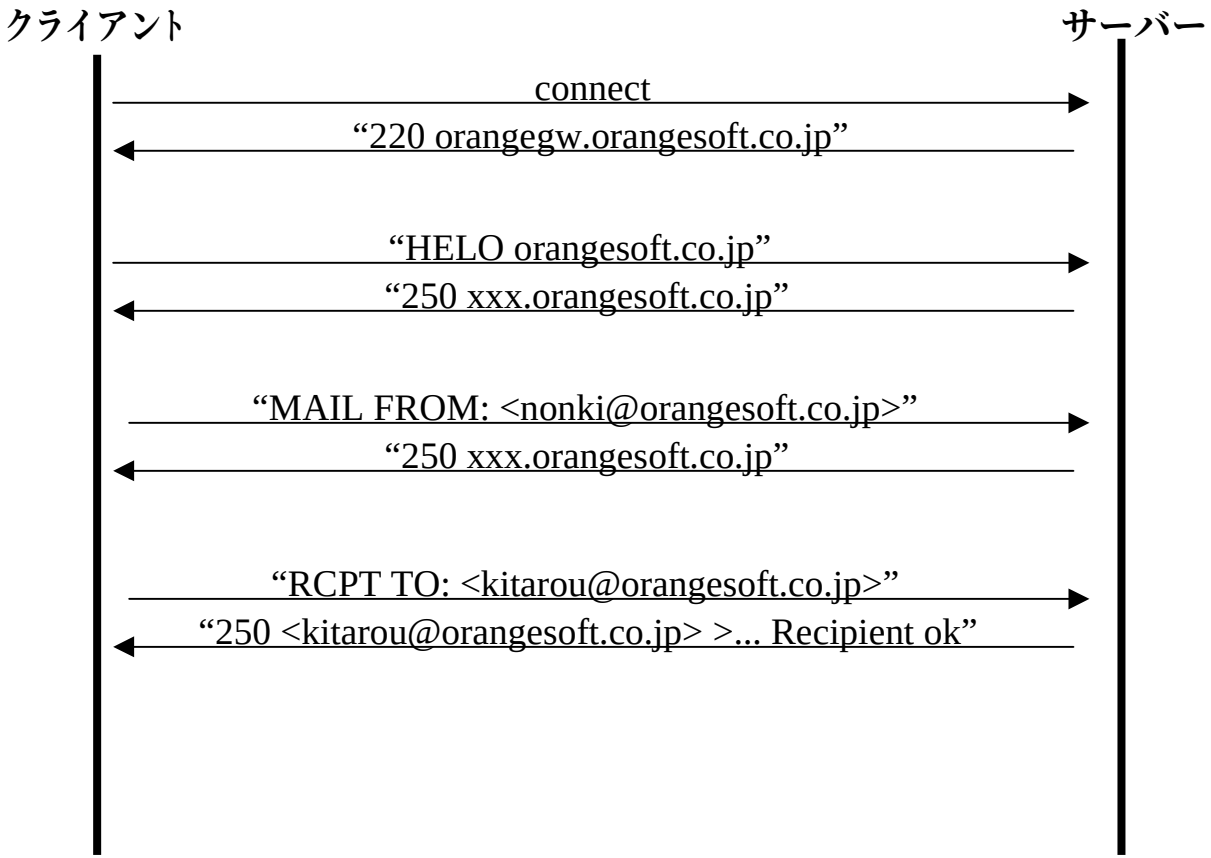
- メッセージ系の処理ではReaderは使いにくい。
- メッセージごとにcharsetは異なる。
  - MIMEマルチパートの場合、メッセージの中でcharsetは変わる。
  - 従って、メッセージを扱う場合はReaderを使った文字変換は使用できない。

実例紹介

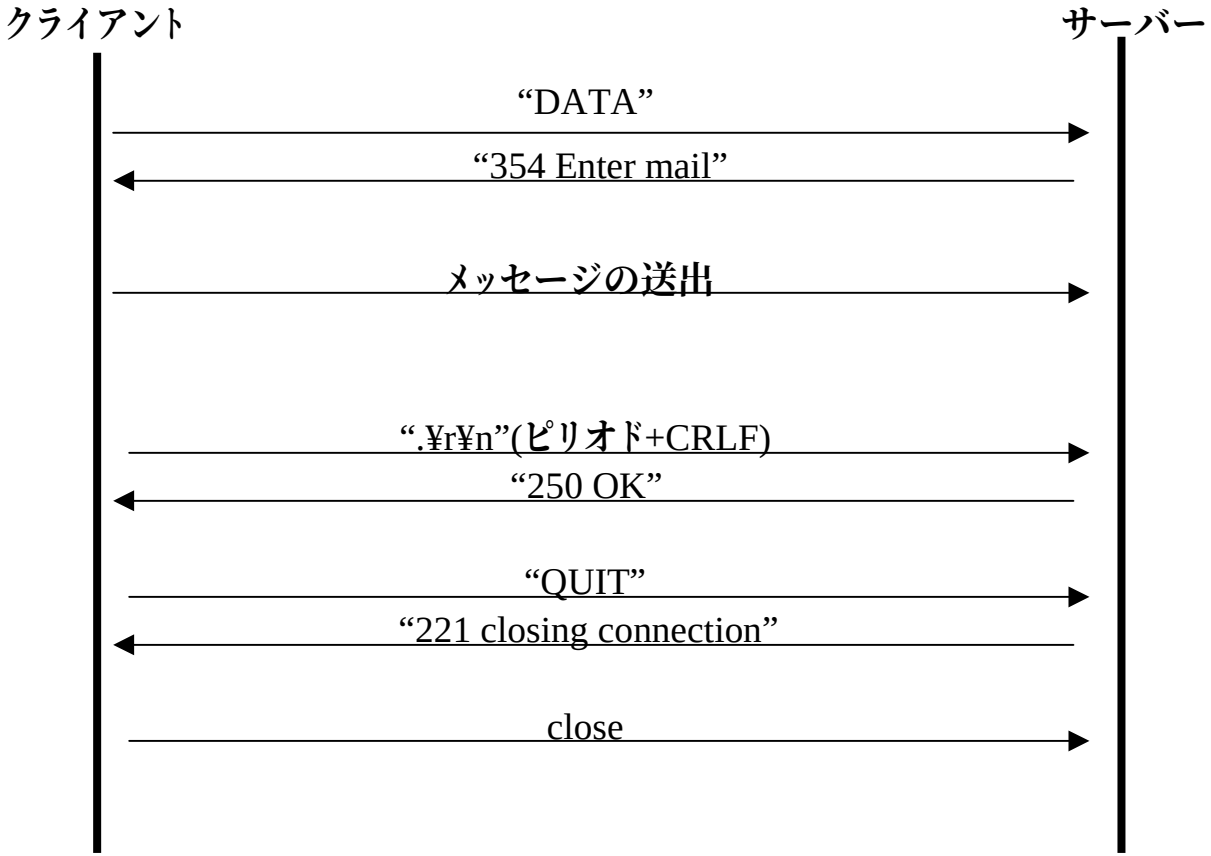
サンプル

- SMTP
  - SIMPLE MAIL TRANSFER PROTOCOL
  - RFC821
- POP3
  - POST OFFICE PROTOCOL
  - RFC1939

SMTP(1)


クライアント
サーバー
connect
“220 orangegw.orangesoft.co.jp”
“HELO orangesoft.co.jp”
“250 xxx.orangesoft.co.jp”
“MAIL FROM: <nonki@orangesoft.co.jp>”
“250 xxx.orangesoft.co.jp”
“RCPT TO: <kitarou@orangesoft.co.jp>”
“250 <kitarou@orangesoft.co.jp> >... Recipient ok”

SMTP(2)


クライアント
サーバー
“DATA”
“354 Enter mail”
メッセージの送出
“.¥r¥n”(ピリオド+CRLF)
“250 OK”
“QUIT”
“221 closing connection”
close

SMTPコマンド

コマンド一覧(抜粋)

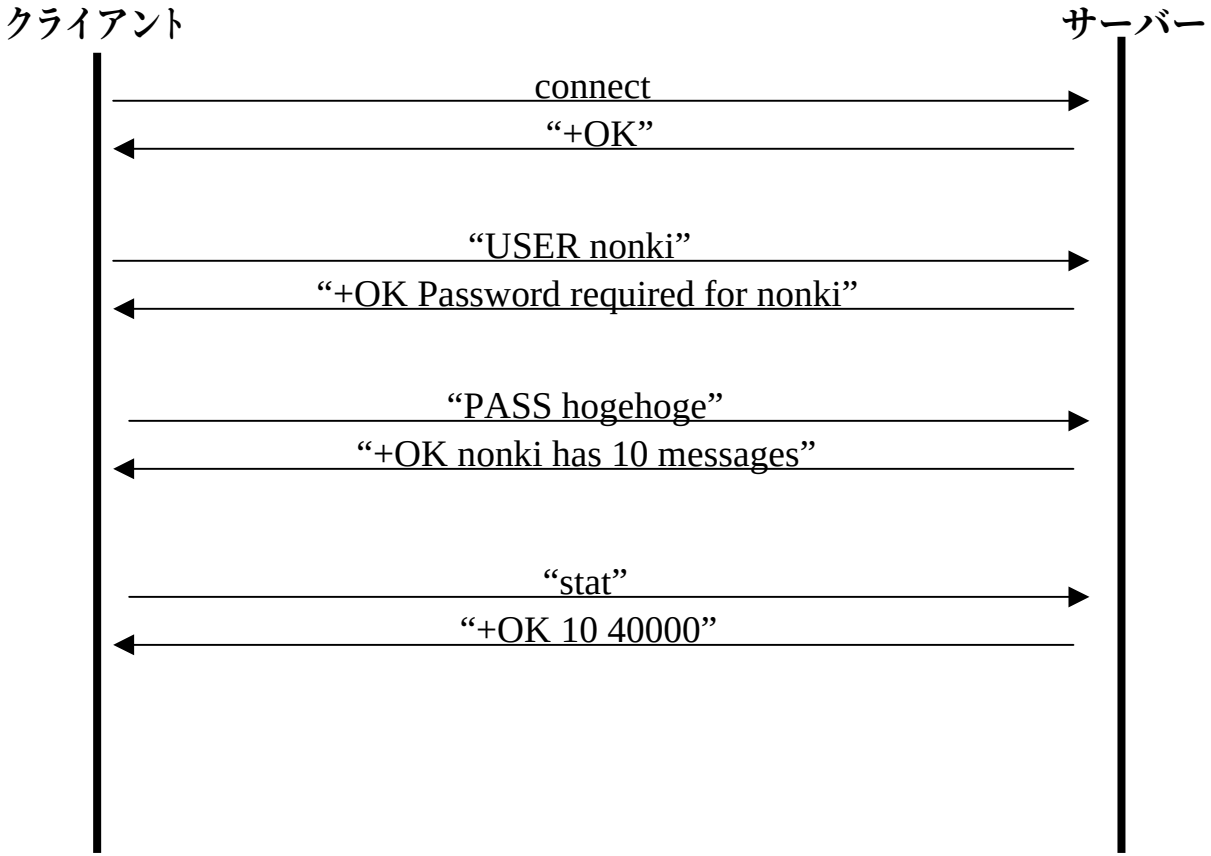
- HELO　使用開始
- MAIL　メール送信の開始
- RCPT　送り先メールアドレスの指定
- DATA　メール本文の送信開始
- RSET　リセット
- NOOP　何もしない
- QUIT　接続の終了

SMTPレスポンス

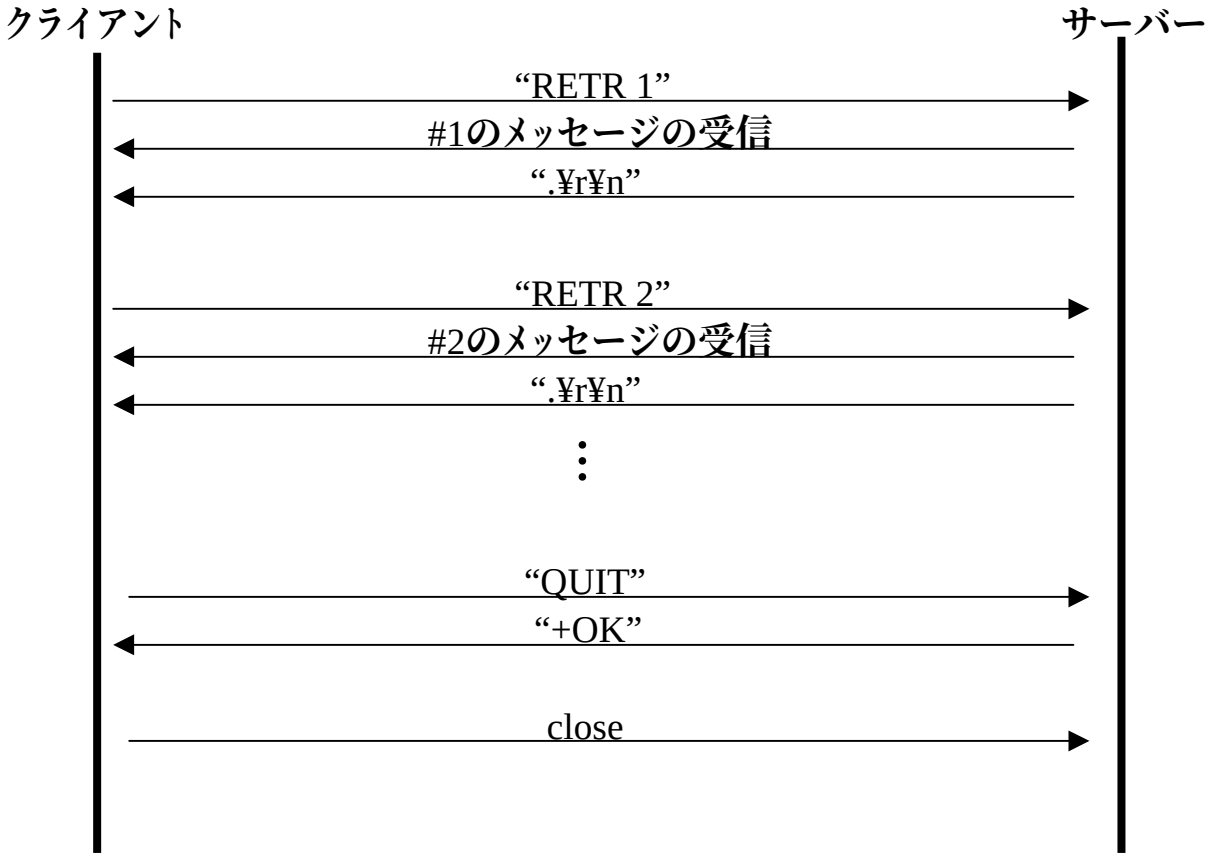
- 3桁(xyz)の数字で表現
  - 1yz 予備的な肯定
  - 2yz 肯定的な完了
  - 3yz 中間的な肯定
  - 4yz 一時的な否定
  - 5yz 永久的な否定
- 複数行の応答

```
123-aaaa
123-bbbb
123 ccccc
```

POP3(1)


クライアント
サーバー
connect
“+OK”
“USER nonki”
“+OK Password required for nonki”
“PASS hogehoge”
“+OK nonki has 10 messages”
“stat”
“+OK 10 40000”

POP3(2)


クライアント
サーバー
“RETR 1”
#1のメッセージの受信
“.¥r¥n”
“RETR 2”
#2のメッセージの受信
“.¥r¥n”
“QUIT”
“+OK”
close

POP3

- コマンド一覧
  - **STAT** メールボックスのメール数とサイズの所得
  - **LIST** ここのメールのサイズの所得
  - RETR 指定したメールの取出し
  - DELE 指定したメールの削除
  - NOOP 何もしない
  - RSET 操作の取消し
  - QUIT 接続の終了
  - TOP 指定したメッセージのヘッダと本文を指定した行数所得する
  - UIDL ここのメールのサーバ内でのIDを所得する
  - USER ユーザ認証時のユーザ名の送信
  - PASS ユーザ認証時のパスワードの送信
  - APOP MD5で暗号化されたユーザ認証
- レスポンス
  - +OK 成功、-ERR 失敗

特徴

- テキストベース
  - 行単位の処理
    - 転送データ量は予測できない。
    - CRLF(0x0d,0x0a)がセパレータ

まとめ

- インターネット
  - 異なったメディアを統合した論理的なネットワーク
  - IPアドレス、トランスポート、ポート番号
- UNIX
  - socket interface
  - 入出力の多重化:select
  - サーバ：fork,inetd

まとめ(2)

- Windows
  - Windows-style APIとunix互換のBerkeley-style APIの２種類をもつ。
  - Windows-style APIはメッセージベースの非同期API
- Java
  - Socketを標準でサポート
  - 同期型のAPI

参考文献

- “UNIX Network Programming Volume 1 Second Edition, Networking APIs: Sockets and XTI” , W.Richard Stevens 著, Prentice Hall
- “インターネットを256倍使うための本 Vol. 2”, 志村 拓, 榊 隆, 岩井 潔 共著, アスキー出版局
- マニュアルページ(man コマンド)

参考文献(2)

- WinSock2.0プログラミング
  - ソフトバンク(ISBN4-7973-0688-2)
- インターネットRFC事典
  - アスキー(ISBN4-7561-1888-7)

Resources

- Java House Mailing List
  - http://java-house.etl.go.jp/ml/
- Javaカンファレンス
  - http://www.java-fj.or.jp/

Resources(2)

- MSDN(Microsoft Developer Network)
  - http://www.microsoft.com/japan/developer/
- MSJ(J) (Microsoft Systems Journal)
- NiftyServe FWINDCフォーラム