ネットワークプログラミング

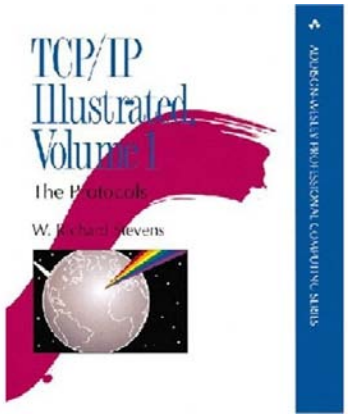
千代浩司
高エネルギー加速器研究機構
素粒子原子核研究所

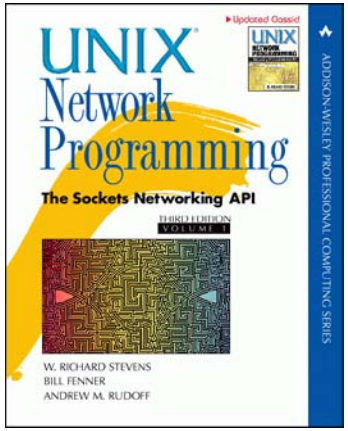
内容

- クライアントアプリケーションを書けるようになろう
- 情報のありか
- 各種ユーティリティー

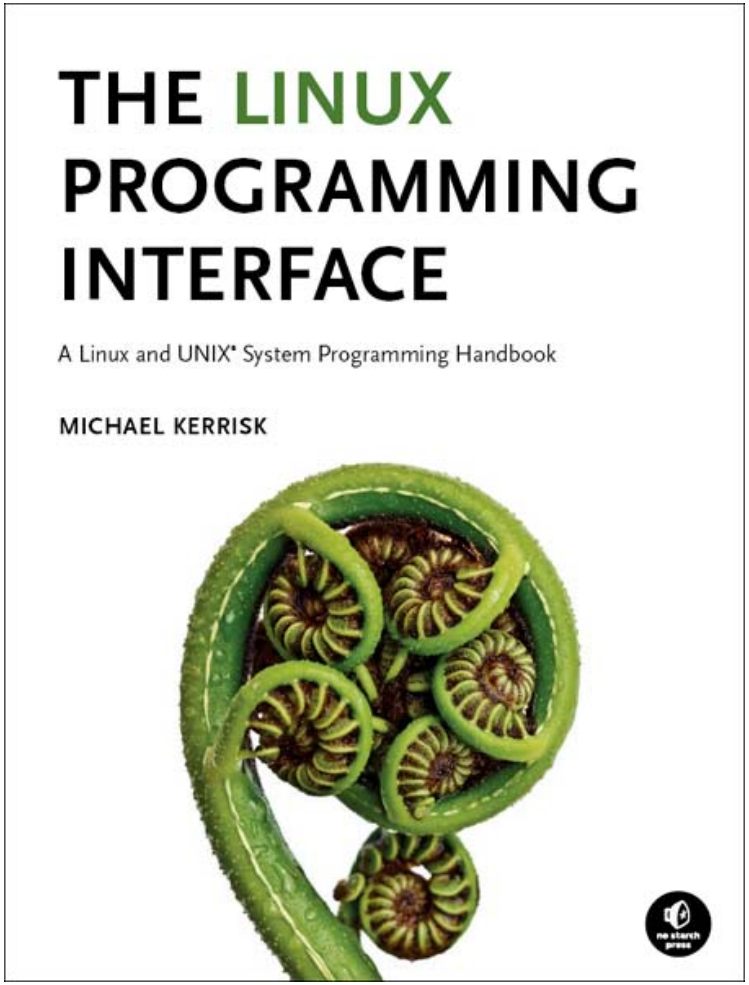
参考書

- Protocol
  - TCP/IP Illustrated, Volume 1 (Stevens)
- Programming
  - Unix Network Programming Volume 1 (3rd edition) (Stevens, Fenner, Rudoff)





Linux System Programming

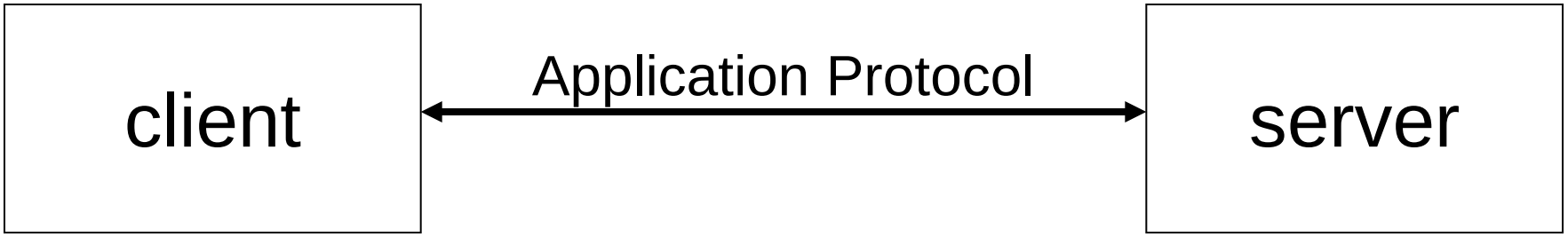


The Linux Programming Interface
Michael Kerrisk
No Starch Press
ISBN 978-1-59327-220-3
1552 pages
published in October 2010
http://man7.org/tlpi/

system call programmingの話だけではなくたとえばshared libraryの作り方、sonameとかの話も書かれています。

1552ページもあって重たいです（電子版もあります）。

Network Application: Client - Server



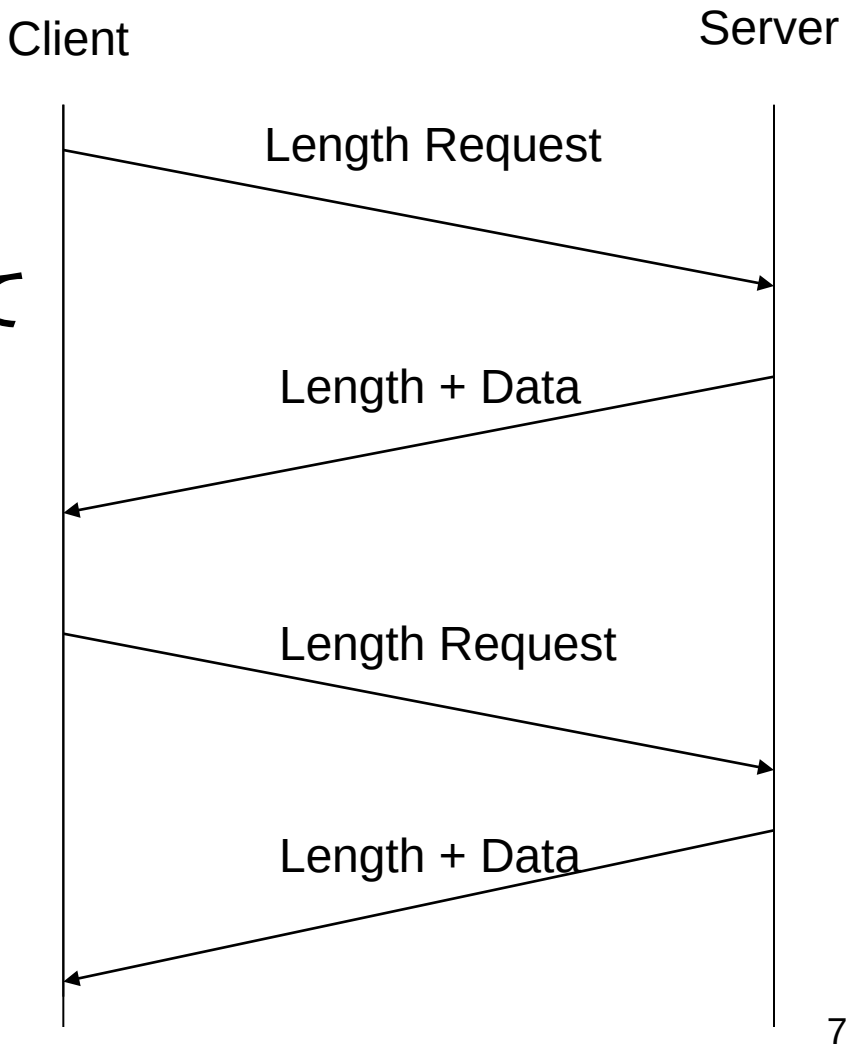
ネットワークを通じて通信するにはまずクライアントおよびサーバー間で通信プロトコルを策定する必要がある。

アプリケーションプロトコルの例

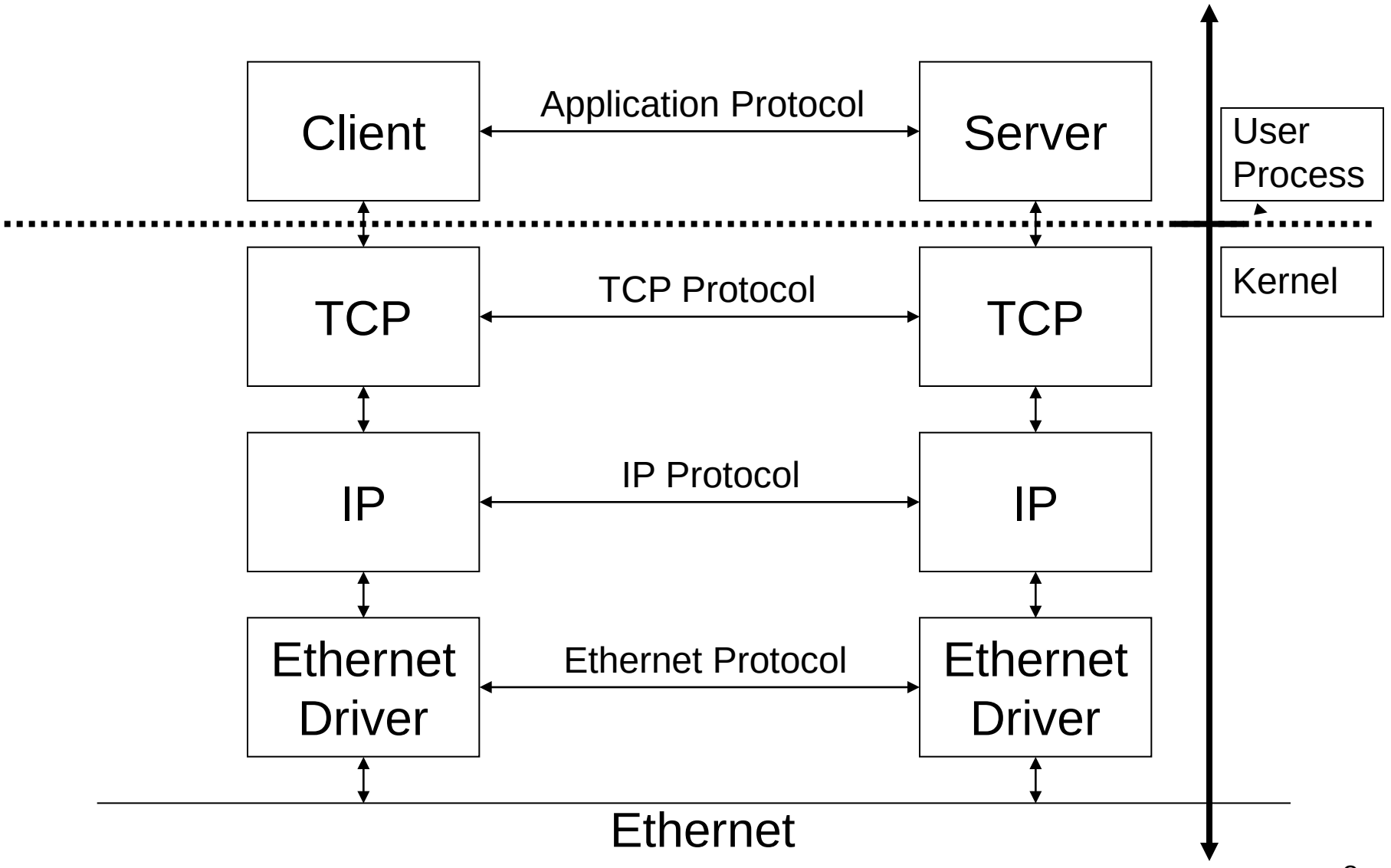
- SMTP (メール）
- HTTP （ウェブ）
- その他いろいろ

アプリケーションプロトコル例

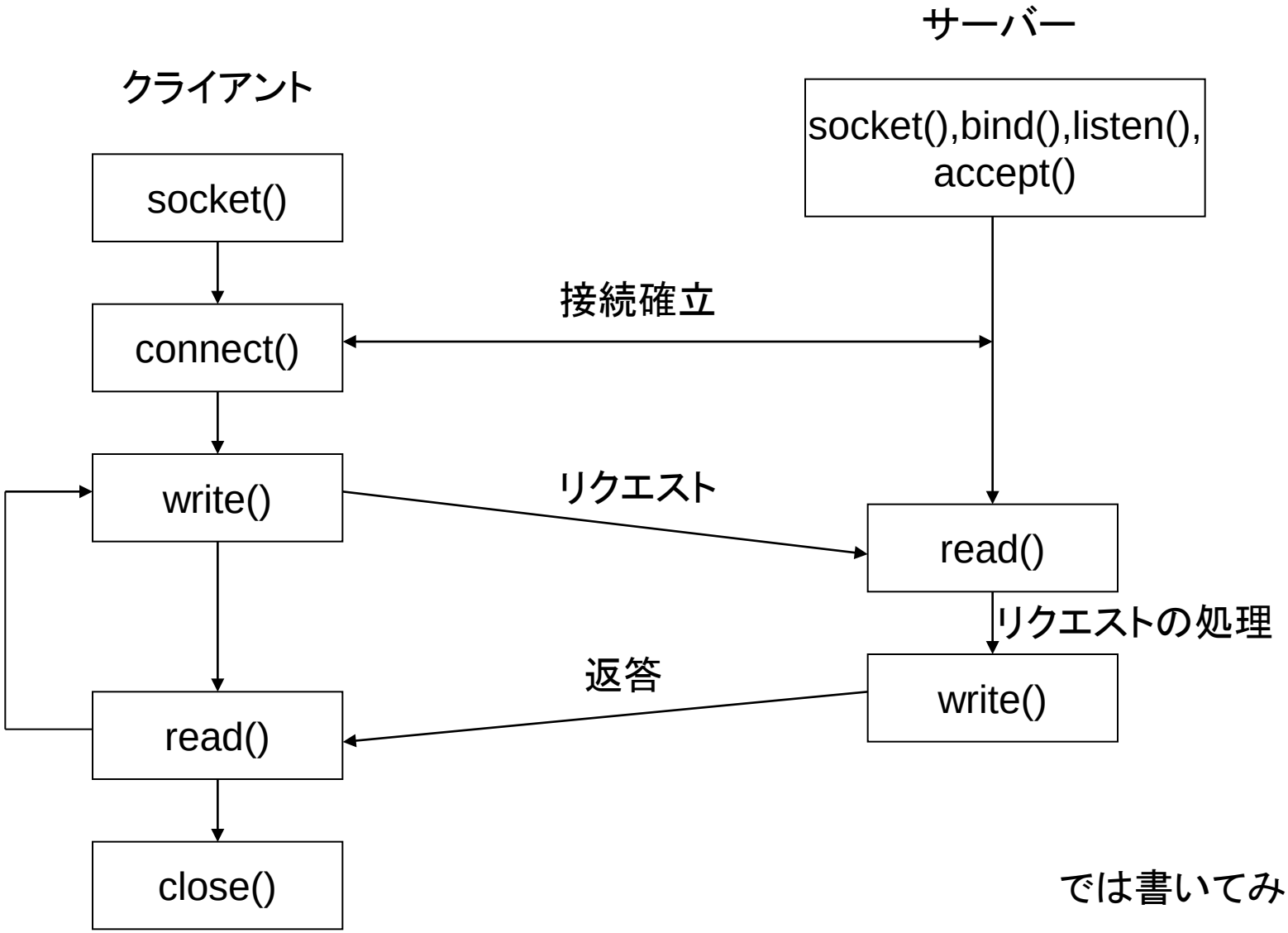
- 垂れ流し
  - クライアントが接続するとデータがだーっと送られてくる
- ポーリングで読み取り
  - 右の図がその一例



Ethernet Using TCP



TCPクライアント、サーバーの流れ



では書いてみよう

話の順序

- クライアントプログラム
- エラーの取得方法
- 各システムコール
  - socket()
  - connect()
  - read()
  - write()
- ネットワークバイトオーダー

クライアントプログラム

```
int sockfd;
sockfd = socket(AF_INET, SOCK_STREAM, 0);
connect(sockfd, (struct sockaddr *)&servaddr, sizeof(servaddr));
```

接続を確保できればあとはファイルディスクリプタを使って
read()したりwrite()すれば通信できる
（普通のファイルの読み書きと同様）。

理想的にはkernel側の話はまったく知らなくてもよいはずだが、
そうはいかないこともある。

kernel側も理解しておくと勉強が進む（こともある）。

エラーの取得

- システムコールがエラーを起こした場合
  - 多くは -1 を返す
  - 大域変数 errno にエラーの理由を示す番号がセットされる。
  - エラーが起きたときに何を返すかはmanual pageのRETURN VALUE、あるいはERRORSセクションを見る。

エラー原因の文字列への変換

#include <errno.h> /* errno の定義 */

人間が読める文字列に変換するには、perror(), err()等を使う

- perror("user string")
  errnoを見て
  user string: errnoに対応する文字列
  を表示する。
- err(int eval, const char *fmt, ...) /* #include <err.h> */
  progname: fmtの文字列 : errnoに対応する文字列
  と表示してexit(eval)する。non-standard BSD extensions. Linuxにもある。
  fmtはprintf()と同じ感じで書ける:
  err(1, "error on file %s", filename);
- その他

エラーの取得方法例題

```
int sockfd;
if ( (sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    perror("socket error");
    exit(1);
}
```

```
int sockfd;
char *ip_address = “192.168.0.16”;
if ( (sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    err(1, “socket: %s” ip_address); // exitする
}
```

socket()

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

int socket(int domain, int type, int protocol);
```

domain
- IPv4: AF_INET
- Unix: AF_UNIX (X11などで使われている）

type
- SOCK_STREAM (TCP)
- SOCK_DGRAM (UDP)

protocol
- 0
- その他

```
int sockfd;
if ( (sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    perror("socket error");
    exit(1);
}
```

connect() [1]

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

int connect(int sockfd, const struct sockaddr *serv_addr,
                socklen_t addrlen);
```

struct sockaddr: 総称ソケットアドレス構造体

- アドレス、ポートの情報を格納する構造体

```
struct sockaddr {
    uint8_t         sa_len;
    sa_family_t     sa_family;      /* address family: AF_XXX value */
    char            sa_data[14];    /* protocol-specific address
};
```

connect()では通信相手を指定するためにsockaddrを使用する。

connect() [2]（IPv4の場合）

```
struct sockaddr_in {
    sa_family_t     sin_family;         /* AF_INET */
    in_port_t       sin_port;           /* 16 bit TCP or UDP port number
    struct in_addr  sin_addr;           /* 32 bit IPv4 address */
    char            sin_zero[8]         /* unused */
};
struct in_addr {
    in_addr_t       s_addr;
};
```

Example:

```
struct sockaddr_in servaddr;
char       *ip_address = "192.168.0.16";
int        port = 13;                                   /* daytime */
servaddr.sin_family  = AF_INET;
servaddr.sin_port    = htons(port);
inet_pton(AF_INET, ip_address, &servaddr.sin_addr); /* need error check */
```

socket() + connect()

```
struct sockaddr_in servaddr;
int         sockfd;
char        *ip_address = "192.168.0.16";
int         port = 13;                                  /* daytime */

if ( (sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    peror("socket");
    exit(1);
}

servaddr.sin_family  = AF_INET;
servaddr.sin_port    = htons(port);
if (inet_pton(AF_INET, ip_address, &servaddr.sin_addr) <=0) {
    fprintf(stderr, "inet_pton error for %s¥n", ip_address);
}

if (connect(sockfd, (struct sockaddr *) &servaddr,
            sizeof(servaddr)) < 0) {
    perror("connect");
    exit(1);
}
```

connect_tcp()

```
if (   (sockfd = connect_tcp(ip_address, port)) < 0) {
    fprintf(stderr, "connect error");
    exit(1);
}
```

と書けるようにまとめておくと使いまわしがきく（かもしれない）。

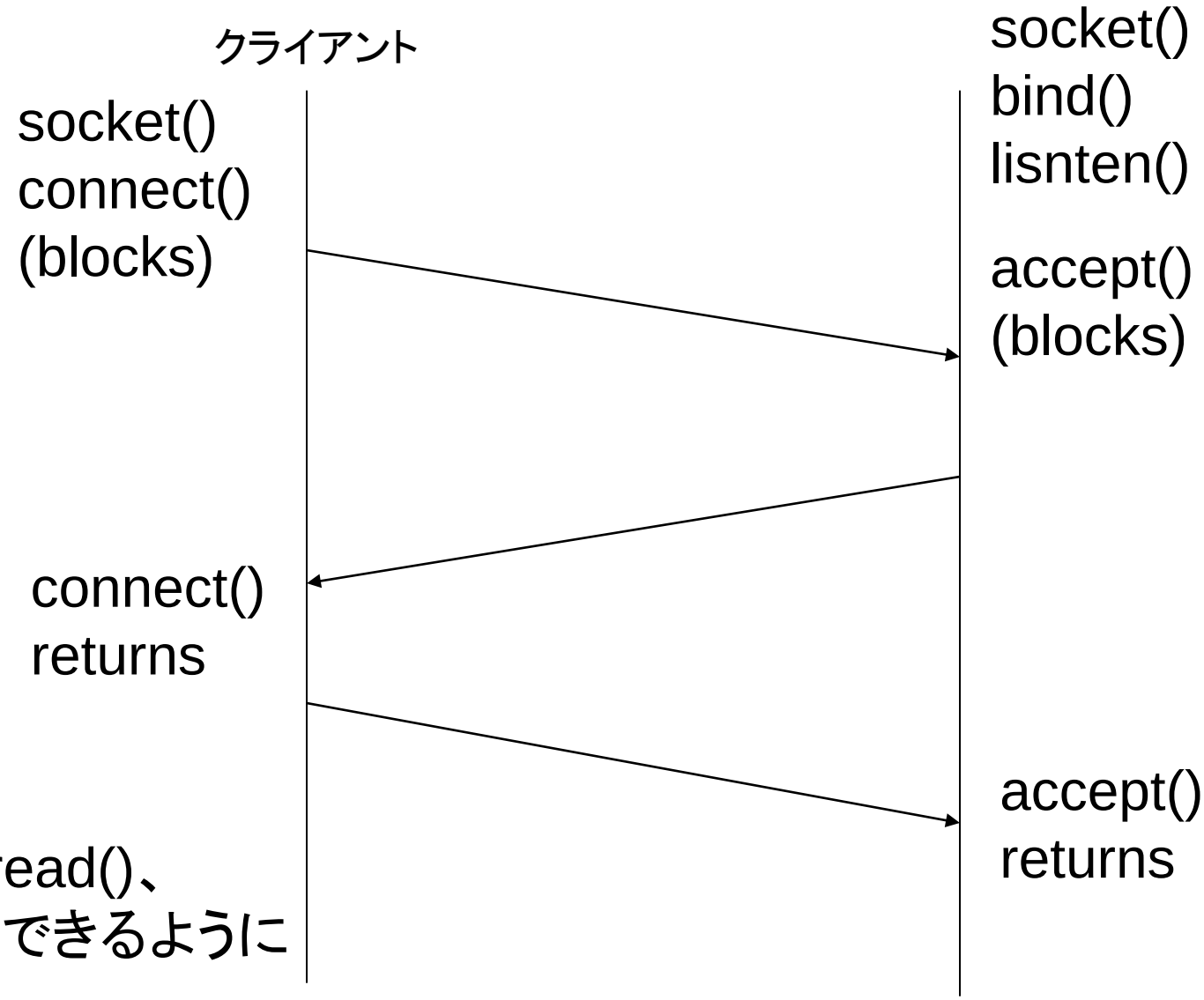
その他
　ソケットアドレス構造体の取り扱いにgetaddrinfo()を使う。

DAQ-MiddlewareでのSockライブラリ

```
try {
        // Create socket and connect to data server.
        m_sock = new DAQMW::Sock();
        m_sock->connect(m_srcAddr, m_srcPort);
} catch (DAQMW::SockException& e) {
        std::cerr << "Sock Fatal Error : " << e.what() << std::endl;
        fatal_error_report(USER_DEFINED_ERROR1, "SOCKET FATAL ERROR");
} catch (...) {
        std::cerr << "Sock Fatal Error : Unknown" << std::endl;
        fatal_error_report(USER_DEFINED_ERROR1, "SOCKET FATAL ERROR");
}
```

/usr/share/daqmw/examples/SampleReader/SampleReader.cpp

TCP接続 サーバー


クライアント
socket()
bind()
lisnten()
socket()
connect()
(blocks)
accept()
(blocks)
connect()
returns
accept()
returns
ここでread()、
write()できるように
なる。

パケットの流れを見てみる

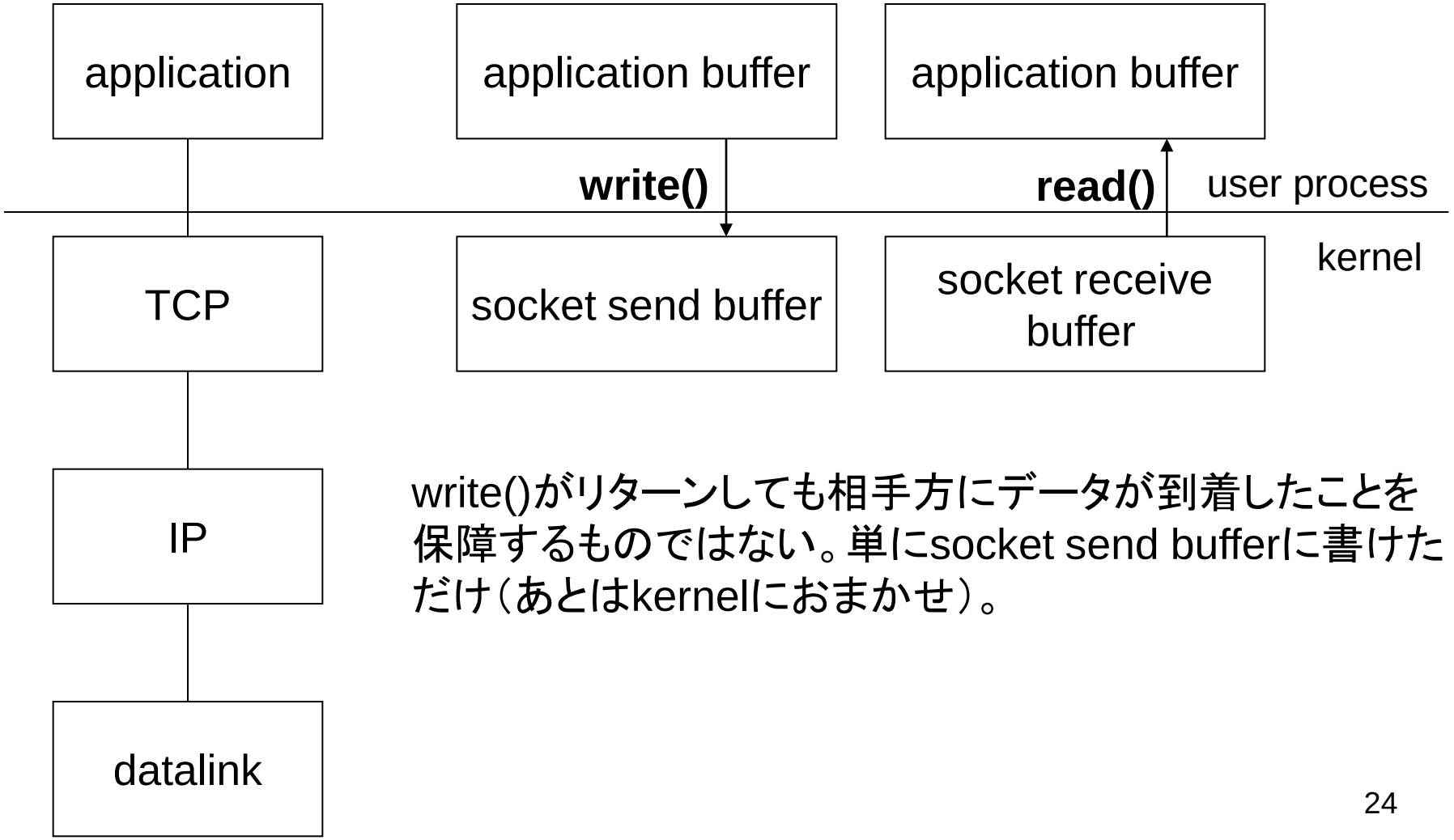
```
0.000000   0.000000   connect start
0.000363   0.000363   IP 192.168.0.100.35005 > 192.168.0.101.13: S
0.000489   0.000126   IP 192.168.0.101.13    > 192.168.0.100.35005: S
0.000536   0.000047   IP 192.168.0.100.35005 > 192.168.0.101.13: . ack 1 win 1460
0.000583   0.000047   connect returns

0.004302   0.003719   IP 192.168.0.101.13    > 192.168.0.100.35005: FP 1:27(26) ack 1
0.004718   0.000416   IP 192.168.0.100.35005 > 192.168.0.101.13: F 1:1(0) ack 28
0.004917   0.000199   IP 192.168.0.101.13    > 192.168.0.100.35005: . ack 2 win 33303
```

read()、write()

- ソケットファイルディスクリプタをread(), write()するとデータの受信、送信ができる。

- read()
  - 通信相手方からのデータがソケットレシーブバッファに入っている。そのデータを読む。
- write()
  - ソケットセンドバッファにデータを書く。書いたデータが通信相手方に送られる。

TCP Input/Output



write()がリターンしても相手方にデータが到着したことを保障するものではない。単にsocket send bufferに書けただけ（あとはkernelにおまかせ）。

read()

```
#include <unistd.h>
ssize_t read(int fd, void *buf, size_t count);
```

```
#define MAX_BUF_SIZE 1024

ssize_t n;
unsigned char buf[MAX_BUF_SIZE];
n = read(sockfd, buf, sizeof(buf));
if (n < 0) {
    perror("read error");
    exit(1);
}
```

戻り値
n > 0: 読んだバイト数
n==0: EOF
n== -1: エラー

read()

```
#include <unistd.h>
ssize_t read(int fd, void *buf, size_t count);
```

read()がリターンしたときにbufにcountバイトのデータが入っているとは限らない。
(データが要求したぶんだけまだ到着していないなど）

必ずcountバイト読んだあとリターンするようにしたければそのようにプログラムする必要がある。

readn()

```
int readn(int sockfd, unsigned char *buf, int nbytes)
{
    int nleft;
    int nread;
    unsigned char *buf_ptr;

    buf_ptr = buf;
    nleft   = nbytes;

    while (nleft > 0) {
        nread = read(sockfd, buf_ptr, nleft);
        if (nread < 0) {
            if (errno == EINTR) {
                nread = 0; /* read again */
            }
        }
        else if (nread == 0) { /* EOF */
            break;
        }
        nleft   -= nread;
        buf_ptr += nread;
    }

    return (nbytes - nleft);
}
```

ソケットレシーブバッファに
何バイトのデータがあるか調べる方法

- nbytes = recv(sockfd, buf, sizeof(buf),
  MSG_PEEK|MSG_DONTWAIT);

  データはbufにコピーされる

- ioctl(sockfd, FIONREAD &nbytes);

write()

```
#include <unistd.h>
ssize_t write(int fd, const void *buf, size_t count);
```

```
unsigned char buf[4];
ssize_t n;

buf[0] = 0x5a;
buf[1] = 0x5b;
buf[2] = 0x5c;
buf[3] = 0x5b;

if (write(sockfd, buf, 4) == -1) {
    perror("write error");
    exit(1);
}
```

ソケットセンドバッファに余裕がないときにはブロックする（エラーにはならない）。
ブロックしないようにするにはノンブロックキグソケットオプションを使う（ノンブロッキングにするとエラー処理とかでだいぶ行数が増える）。

ネットワークバイトオーダー

```
#include <stdio.h>

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

        union num_tag {
                unsigned char c[sizeof(int)];
                unsigned int    num;
        } u_num;

        u_num.num = 0x01020304;

        for (i = 0; i < sizeof(int); i++) {
                printf("u_num.c[%d]: %p 0x%02x ¥n", i, &u_num.c[i], u_num.c[i]);
        }
        return 0;
}
```

出力 (i386)

```
u_num.c[0]: 0xbfbfe850 0x04
u_num.c[1]: 0xbfbfe851 0x03
u_num.c[2]: 0xbfbfe852 0x02
u_num.c[3]: 0xbfbfe853 0x01
```

ネットワークバイトオーダー

int a = 0x01020304;

big endian

0x01	0x02	0x03	0x04

little endian

0x04	0x03	0x02	0x01

- htonl (host to network long)
- htons (host to network short)
- ntohl (network to host long)
- ntohs (network to host short)

情報のありか

- Manual Page
- 本

Manual Pages

- セクション
  - 1 (Utility Program)
  - 2 (System call)
  - 3 (Library)
  - 4 (Device)
  - 5 (File format)
  - 6 (Game)
  - 7 (Misc.)
  - 8 (Administration)

Linuxだとこの他

- 3P (Posix)

表示される順番はMANSECT
環境変数で指定する。
RHEL4のデフォルトでは（2）
より(3P)が先にでる。
RHEL 5,6では(2)が先にでる。
3Pを読むには man 3p read

Manual Pages

- manコマンド
- Linuxのマニュアルページは
  - http://www.kernel.org/doc/man-pages/
  - 最新のマニュアルはここで読める。
  - 利用しているkernel、library等のバージョンに注意する必要がある（こともある）。

Manual Pages

- Header

  READ(3P) POSIX Programmer's Manual READ(3P)

  READ(2) Linux Programmer's Manual READ(2)
- SYNOPSIS
- DESCRIPTION
- RETURN VALUE
- SEE ALSO

- EXAMPLE

Manual Pages(例題)

READ(2) Linux Programmer's Manual READ(2)

NAME

read - read from a file descriptor

SYNOPSIS

#include <unistd.h>

ssize_t read(int fd, void *buf, size_t count);

DESCRIPTION

read() attempts to read up to count bytes from file descriptor fd into the buffer starting at buf.

:

RETURN VALUE

:

ERRORS

:

CONFORMING TO

SVr4, 4.3BSD, POSIX.1-2001.

NOTES

:

SEE ALSO

Manual Pages

struct sockaddr_in servaddr;
&servaddr.sin_port

&と構造体の.(ドット)ってどっちが強い？

man operator

This manual page lists C operators and their precedence in evaluation.

Operator	Associativity
() [] -> .	left to right
! ~ ++ -- + - (type) * & sizeof	right to left
* / %	left to right

以下略。
最初の()は関数の()。n = (i + j) * k の()ではない。

Utility

- gettimeofday()
- tcpdump、wireshark (ex. ethereal)

gettimeofday()

```
#include <sys/time.h>
int gettimeofday(struct timeval *tv, struct timezone *tz);
```

```
struct timeval start, end, diff;
if (gettimeofday(&start, NULL) < 0) {
    err(1, "gettimeofday");
}
/* ... */
if (getimeofday(&end, NULL) < 0) {
    err(1, "gettimeofday");
}
timersub(&end, &start, &diff);
printf("%ld.%06ld¥n", result.tv_sec, result.tv_usec);
```

Linuxではgettimeofday()を1,000,000回繰り返して1秒以下
(Pentium 4 2.4 GHzで0.85秒、Core2Quad 2.5GHzで0.45秒程度)

tcpdump

- ネットワーク上を流れているパケットを見るコマンド
  - 接続できないんだけどパケットはでているのか？
  - データが読めないんだけど向こうからパケットはきているんでしょうか？
- rootにならないと使えない（ことが多い）
- 起動方法

```
# tcpdump -n -w dumpfile -i eth0
# tcpdump -n -r dumpfile
```

- Selector

```
# tcpdump -n -r host 192.168.0.16
# tcpdump -n -r src 192.168.0.16 and dst 192.168.0.17
```

tcpdump出力例

TCPの3wayハンドシェイク付近:

11:27:55.137827 IP **192.168.0.16.59448 > 192.168.0.17.http:** S 153443204: 153443204(0) win 5840 <mss 1460,sackOK,timestamp 587094474 0,nop,wscale 7>
11:27:55.139573 IP **192.168.0.17.http > 192.168.0.16.59448**: S 4091282933: 4091282933(0) ack 153443205 win 65535 <mss 1460,nop,wscale 1,nop,nop,timestamp 3029380287 587094474,sackOK,eol>
11:27:55.139591 IP **192.168.0.16.59448 > 192.168.0.17.http**: . ack 1 win 46 <nop,nop,timestamp 587094479 3029380287>
11:27:55.139751 IP **192.168.0.16.59448 > 192.168.0.17.http**: P 1:103(102) ack 1 win 46 <nop,nop,timestamp 587094479 3029380287>
11:27:55.143520 IP **192.168.0.17.http > 192.168.0.16.59448**: P 1:252(251) ack103 win 33304 <nop,nop,timestamp 3029380290 587094479>

tcpdump - 時刻情報

- 絶対時刻ではなくて相対的な時間に変換するプログラムを作っておくと便利なことがある。

0.000000 0.000000 IP **192.168.0.16.59448 > 192.168.0.17.http**: S 153443204:1534432
0.001746 0.001746 IP **192.168.0.17.http > 192.168.0.16.59448**: S 4091282933:409128
0.001764 0.000018 IP **192.168.0.16.59448 > 192.168.0.17.http**: . ack 1 win 46 <nop
0.001924 0.000160 IP **192.168.0.16.59448 > 192.168.0.17.http**: P 1:103(102) ack 1
0.005693 0.003769 IP **192.168.0.17.http > 192.168.0.16.59448**: P 1:252(251) ack 10
0.005703 0.000010 IP **192.168.0.16.59448 > 192.168.0.17.http**: . ack 252 win 54 <n
1.107822 1.102119 IP **192.168.0.16.59448 > 192.168.0.17.http**: F 103:103(0) ack 25
1.108482 0.000660 IP **192.168.0.17.http > 192.168.0.16.59448**: . ack 104 win 33304
1.109608 0.001126 IP **192.168.0.17.http > 192.168.0.16.59448**: F 252:252(0) ack 10
1.109618 0.000010 IP **192.168.0.16.59448 > 192.168.0.17.http**: . ack 253 win 54 <n

最初の欄はSYNを送ってからの経過時間
2番目の欄は直前の行との時間差を示すもの

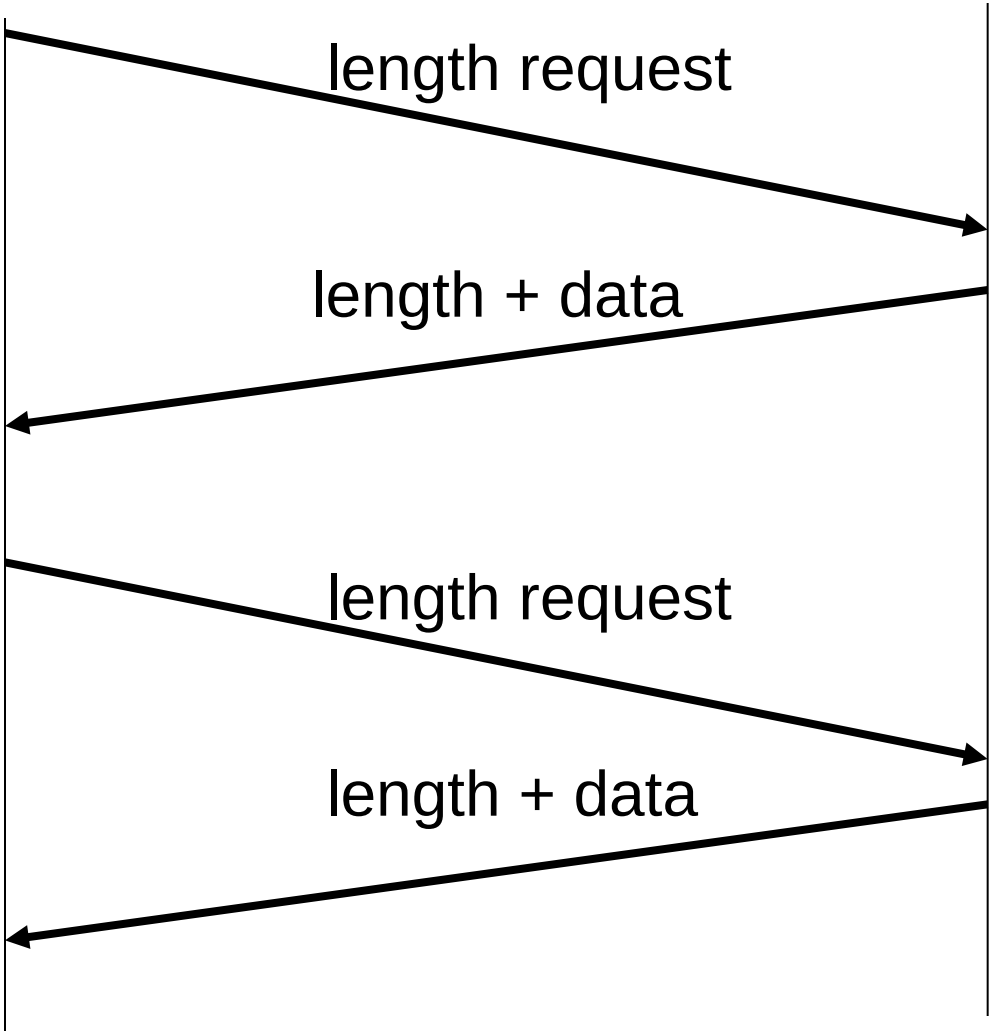
tcpdump + program log

- tcpdumpの時刻情報と同じ時刻フォーマットでログを出すようにしておいてtcpdumpをとりつつプログラムを走らせあとからマージする:

  (tcpdump -n -r tcpdump.out; cat log) | sort -n

NEUNET Protocol

クライアント　　サーバー（検出器モジュール）

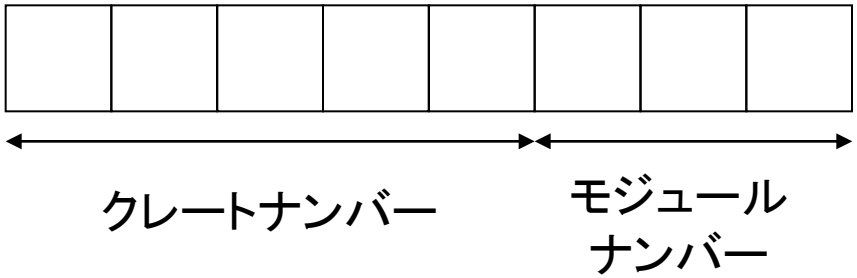


tcpdump + program log

0.000000 0.000000 connect start
0.000063 0.000063 IP 192.168.0.204.57447 > 192.168.0.20.telnet: S 4076228960:407
0.000128 0.000065 IP 192.168.0.20.telnet > 192.168.0.204.57447: S 3718362368:371
0.000159 0.000031 IP 192.168.0.204.57447 > 192.168.0.20.telnet: . ack 1 win 5840
0.000215 0.000056 write length
0.000227 0.000012 IP 192.168.0.204.57447 > 192.168.0.20.telnet: P 1:9(8) ack 1 w
0.000234 0.000007 read length + data
0.000275 0.000041 IP 192.168.0.20.telnet > 192.168.0.204.57447: . ack 9 win 6551
0.002269 0.001994 IP 192.168.0.20.telnet > 192.168.0.204.57447: . 1:5(4) ack 9 w
0.002284 0.000015 IP 192.168.0.204.57447 > 192.168.0.20.telnet: . ack 5 win 5840
0.002300 0.000016 write length
0.002306 0.000006 IP 192.168.0.204.57447 > 192.168.0.20.telnet: P 9:17(8) ack 5
0.002312 0.000006 read length + data
0.002369 0.000057 IP 192.168.0.20.telnet > 192.168.0.204.57447: . ack 17 win 655
0.002568 0.000199 IP 192.168.0.20.telnet > 192.168.0.204.57447: . 5:1465(1460) a
0.002583 0.000015 IP 192.168.0.204.57447 > 192.168.0.20.telnet: . ack 1465 win 8
0.002717 0.000134 IP 192.168.0.20.telnet > 192.168.0.204.57447: . 1465:2925(1460

ビットシフト、マスク

- ネットワークとは直接は関係しない。
- データのデコードの際に必要になることがある。
- 通常の計算ではあまり使用することはない？
- ビットを節約するため等の理由により、１バイト内に意味が違うデータが入っている場合にビットシフト、マスク等を使用してデータを取り出すことが必要である場合がある。



- 取り出したあとは構造体メンバーに代入するなどしたほうがよい。

ビットシフト、マスク

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    int i, j, k;

    i = 1;
    j = (i << 1);
    k = (i << 2);

    printf("i: %d j: %d¥n", i, j);
    printf("i: %d k: %d¥n", i, k);
    return 0;
}
```

```
i: 1 j: 2
i: 1 k: 4
```

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    int i, j, k;

    i = 0x0f;
    j = (i >> 1);
    k = (i >> 2);

    printf("i: %d j: %d¥n", i, j);
    printf("i: %d k: %d¥n", i, k);
    return 0;
}
```

```
i: 15 j: 7
i: 15 k: 3
```

ビットシフト、マスク

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    int i = 0xff;
    int j = 0x0f;
    int k;

    k = (i & j);
    printf("i: %02x k: %02x¥n", i, k);
    return 0;
}
```

```
i: ff k: 0f
```