

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

- プログラムを書かないで済ますには
- クライアントアプリケーションの書き方
 - socket()
 - connect()
 - read(), write()
 - ネットワークバイトオーダー
- ユーティリティ
 - gettimeofday()
 - tcpdump
 - wireshark

プログラムを書かないで済ますには(1)

- リードアウトモジュールがTCPでデータを出す場合には `nc` コマンド が使える
- 読む: `nc 192.168.0.16 24`
- 読んで書く: `nc 192.168.0.16 24 > datafile`
- モニター: `nc 192.168.0.16 24 | monitor_prog`
- モニターしながら書く:
`nc 192.168.0.16 | tee datafile | monitor_prog`

プログラムを書かないで済ますには(2)

- サーバー側もできることがある
 - nc -l 1234 (port 24で接続を待つ)
 - nc -l 1234 < datafile

アルファベットのエル
(数字の1ではない)

参考書 (軽量型)

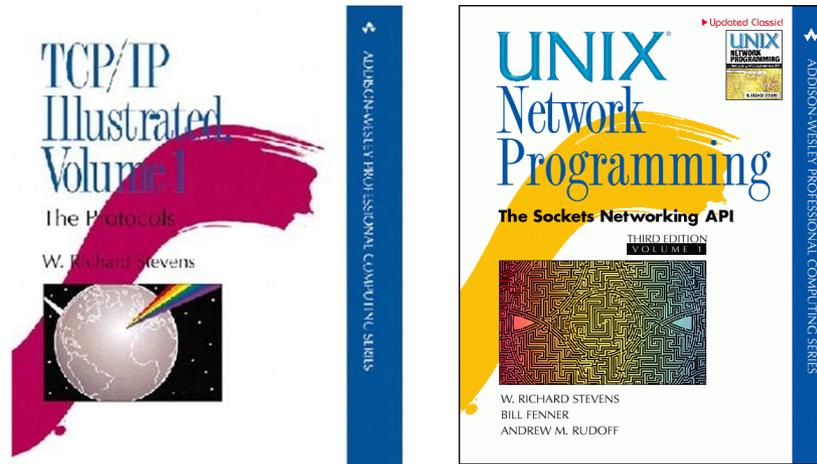


TCP/IP ソケットプログラミングC言語編
Michael J. Donahoo, L. Calvert
小高知宏監訳
オーム社
ISBN4-274-06519-7

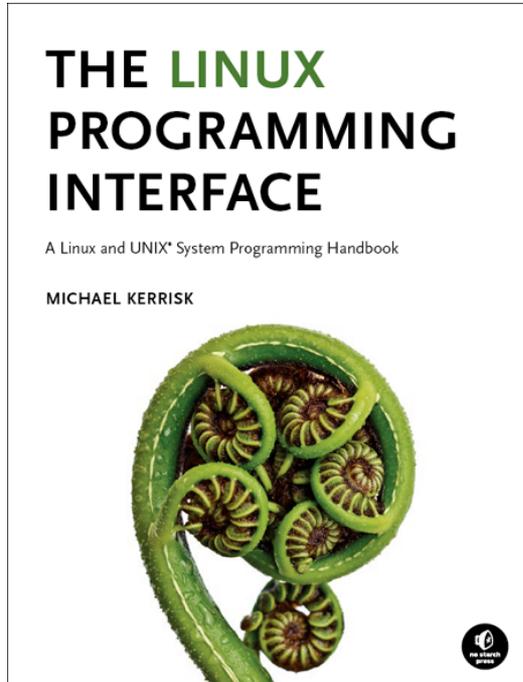
<http://ssl.ohmsha.co.jp/cgi-bin/menu.cgi?ISBN=4-274-06519-7>
38ページまで読めばクライアントが書けるようになる。

参考書 (本格的)

- Protocol
 - TCP/IP Illustrated, Volume 1 2nd edition (Fall, 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/>



翻訳

Linuxプログラミングインターフェイス

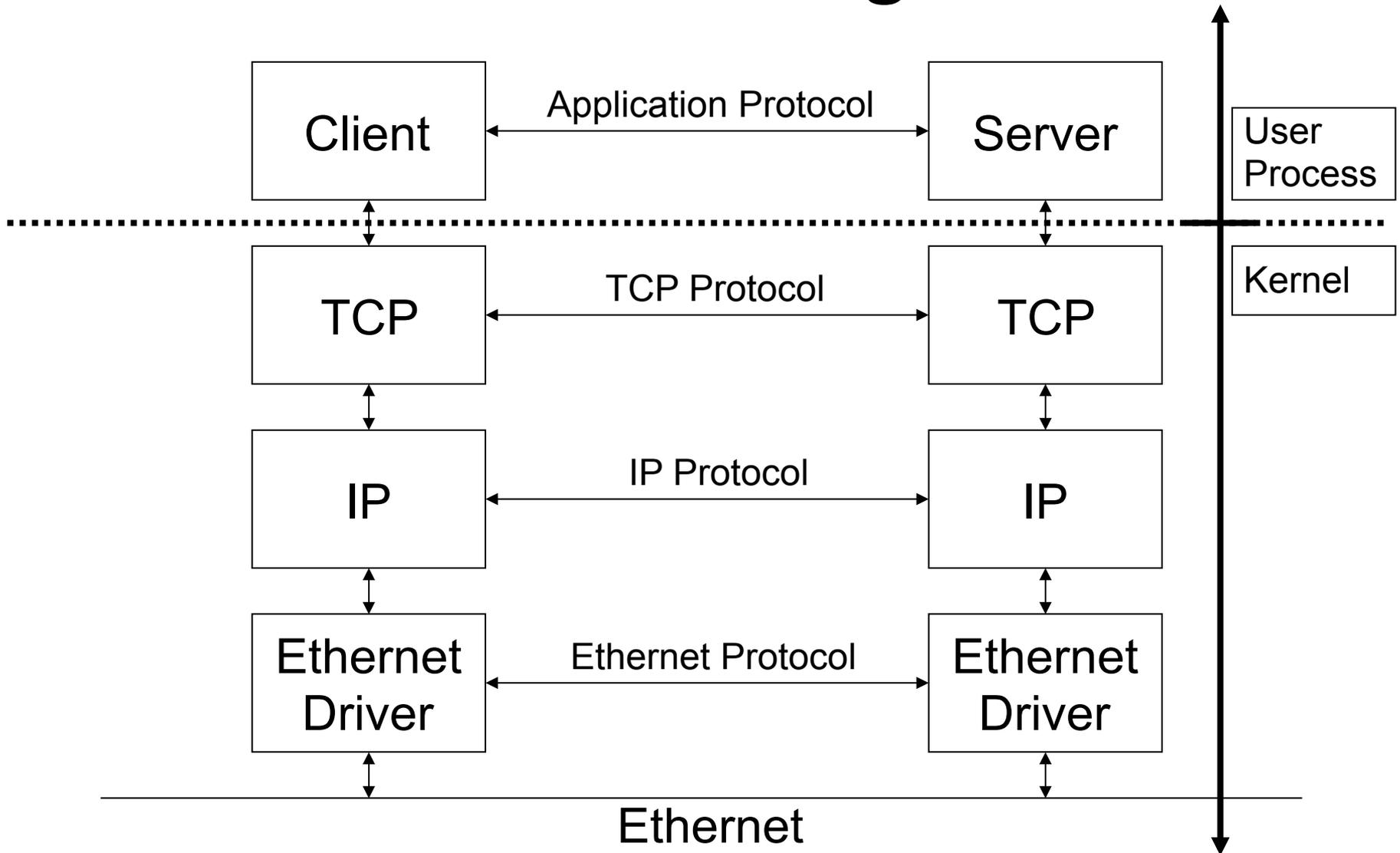
Michael Kerrisk 著、千住 治郎 訳

ISBN978-4-87311-585-6

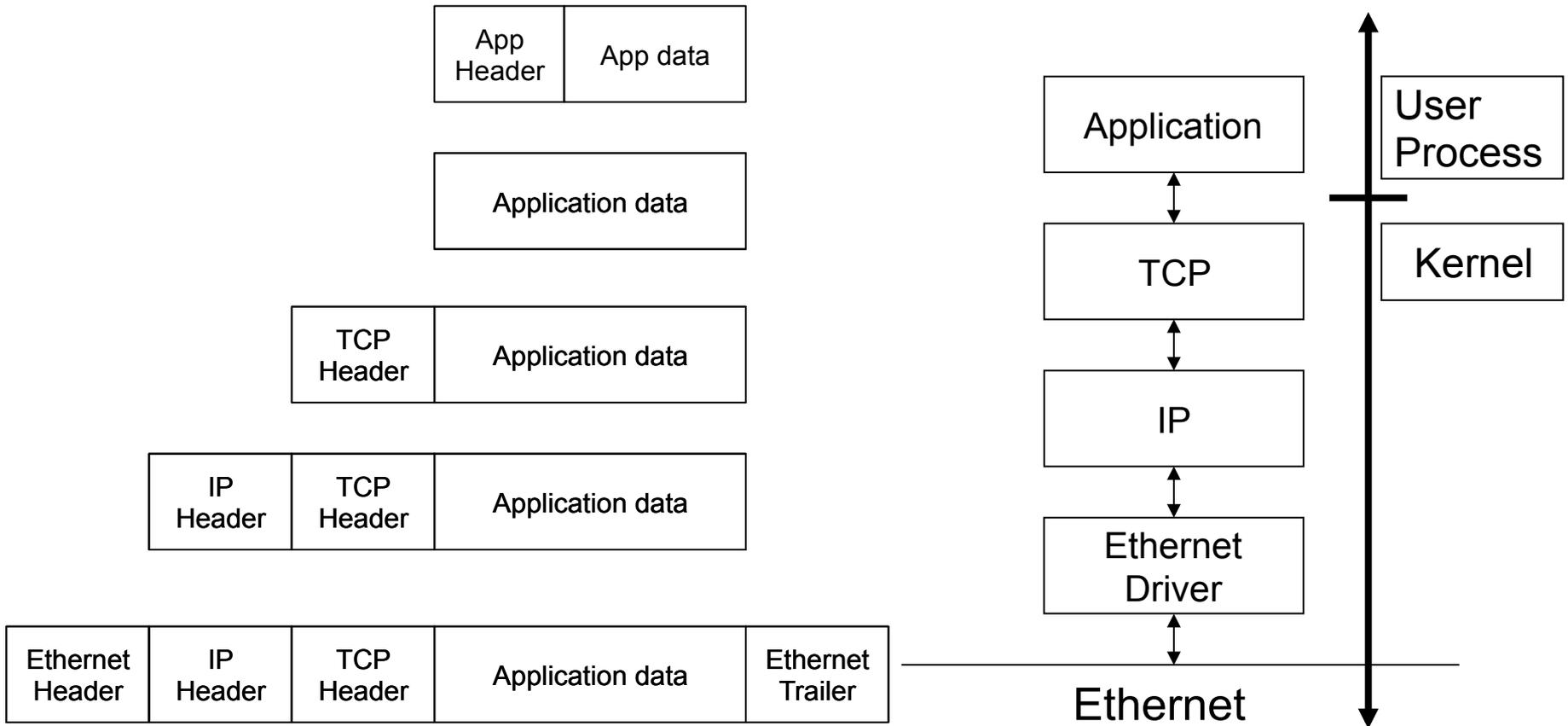
1604 ページ

システムコールプログラミングの話だけではなくたとえばシェアードライブラリの作り方およびsonameなどの話も書かれています。

Ethernet Using TCP

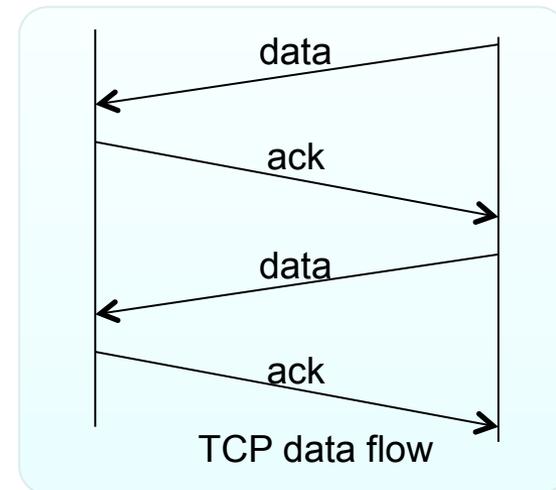
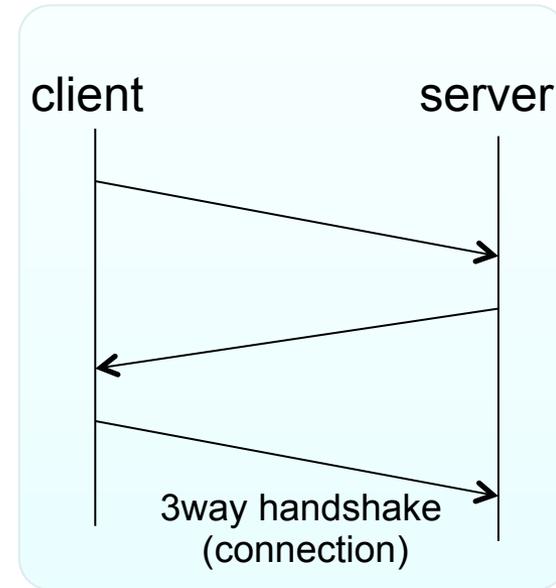


プロトコルスタック縦断



TCPとUDP

- TCP (Transmission Control Protocol)
 - コネクション型通信(まず最初に接続を確立)
 - データが届いたか確認しながら通信する
 - 届いていなければ再送する
 - SiTCPではデータ転送に使う
- UDP (User Datagram Protocol)
 - コネクションレス型通信(接続を確立せずデータを送る)
 - データが届いたかどうかの確認が必要ならそれはユーザーが行う
 - SiTCPではスローコントロールに使う



Network Application: Client - Server



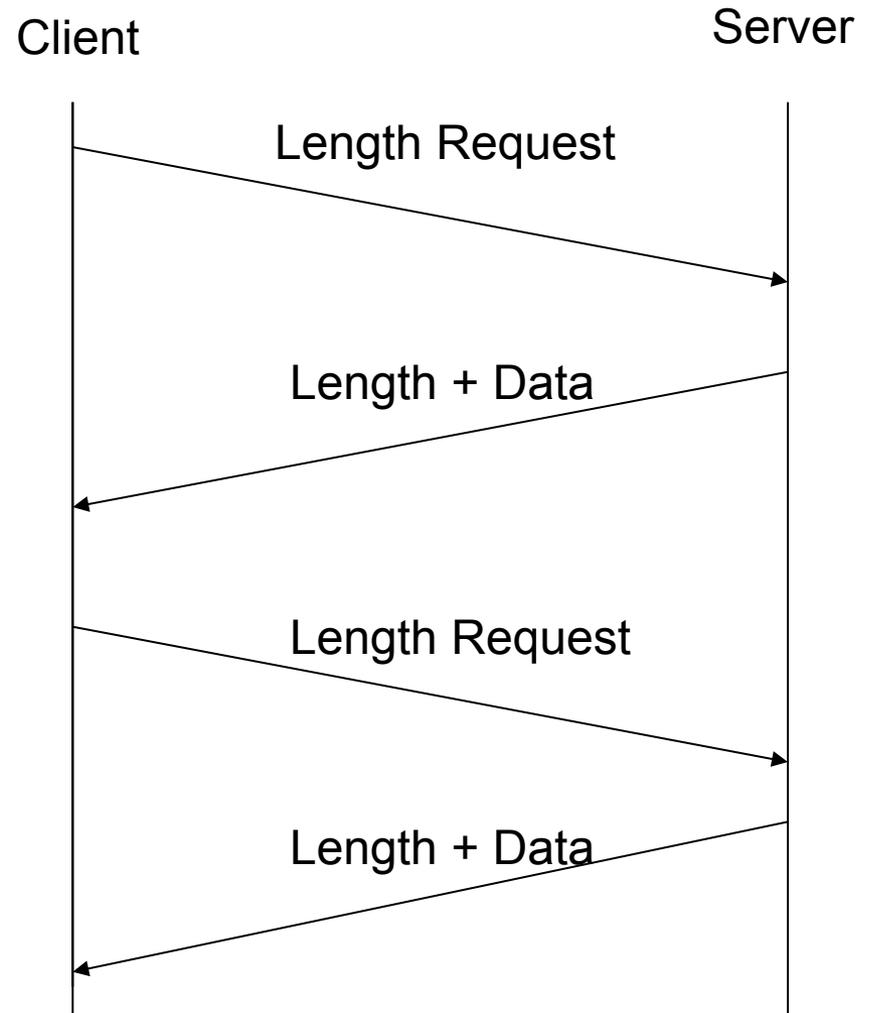
ネットワークを通じて通信するにはまずクライアントおよびサーバー間で通信プロトコルを策定する必要がある。

通信プロトコルの例

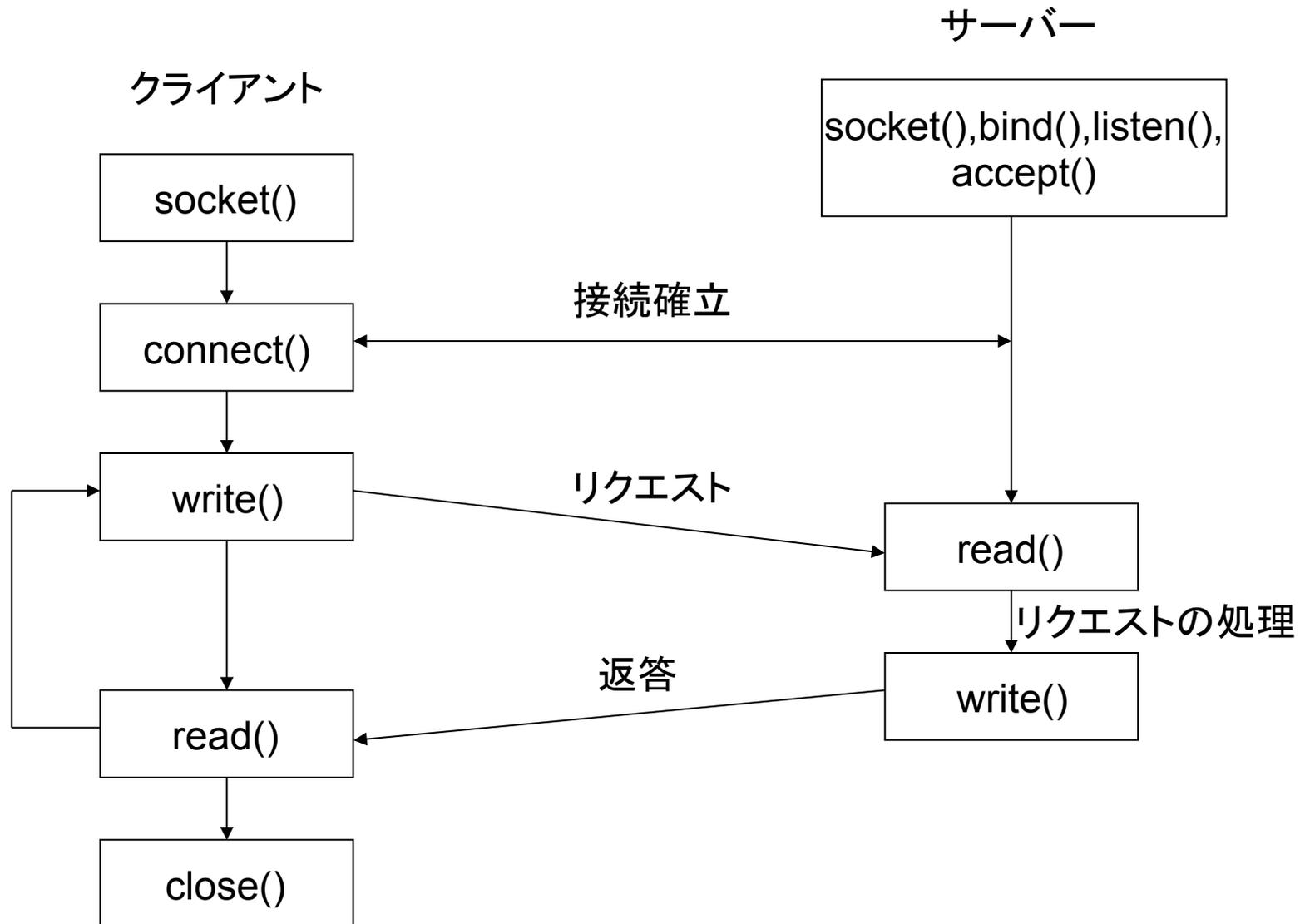
- SMTP (メール)
- HTTP (ウェブ)
- その他いろいろ

通信プロトコルの例：実験システム

- 垂れ流し
- ポーリングで読み取り



TCPクライアント、サーバーの流れ



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)。

```
FILE *fp;  
fp = fopen("my_file", "r")
```

システムコールのエラーの捕捉(1)

- エラーの捕捉は必須
 - さぼるとあまりよいことはない
- 大部分のシステムコールはエラーだと -1 を返す
- 大域変数errnoが設定される
 - #include <errno.h>
 - エラーが起きたときに設定される。エラーがおこる前は前のerrnoが残っている
- どんなエラーがあるかはマニュアルページのERRORSに書いてある。

man socketで出てくる例:

RETURN VALUE

On success, a file descriptor for the new socket is returned. On error, -1 is returned, and errno is set appropriately.

ERRORS

EACCES Permission to create a socket of the specified type and/or protocol is denied.

EAFNOSUPPORT

The implementation does not support the specified address family.

EINVAL Unknown protocol, or protocol family not available.

EMFILE Process file table overflow.

ENFILE The system limit on the total number of open files has been reached.

ENOBUFS or ENOMEM

Insufficient memory is available. The socket cannot be created until sufficient resources are freed.

EPROTONOSUPPORT

The protocol type or the specified protocol is not supported within this domain.

Other errors may be generated by the underlying protocol modules.

システムコールのエラーの捕捉(2)

- errnoから文字列へ変換する関数

- perror()

- err()

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

エラー時にはperror()で指定した文字列 + ": " と、errnoに対応する文字列が表示される。

システムコールのエラーの捕捉(3)

```
#include <err.h>
```

```
err(int eval, const char *fmt, ...)
```

prognome: fmtの文字列 : errnoに対応する文字列
と表示してexit(eval)する。

fmtはprintf()と同じ感じで書ける

```
char *ip_address = "192.168.0.16";
```

```
if ( (sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {  
    err(1, "socket error for %s", ip_address);  
}
```

TCP接続

サーバー

クライアント

socket()
connect()
(blocks)

connect()
returns

ここでread()、
write()できるようになる。

socket()
bind()
listen()

accept()
(blocks)

accept()
returns

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)

```
#include <netinet/in.h>
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) {
    perror("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);
    exit(1);
}

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

長過ぎるので普通はなにかしたいところ

connect_tcp()

```
if ( (sockfd = connect_tcp(ip_address, port)) < 0) {  
    fprintf("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 Sockライブラリでは

```
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");  
}
```

パケットの流れをしてみる

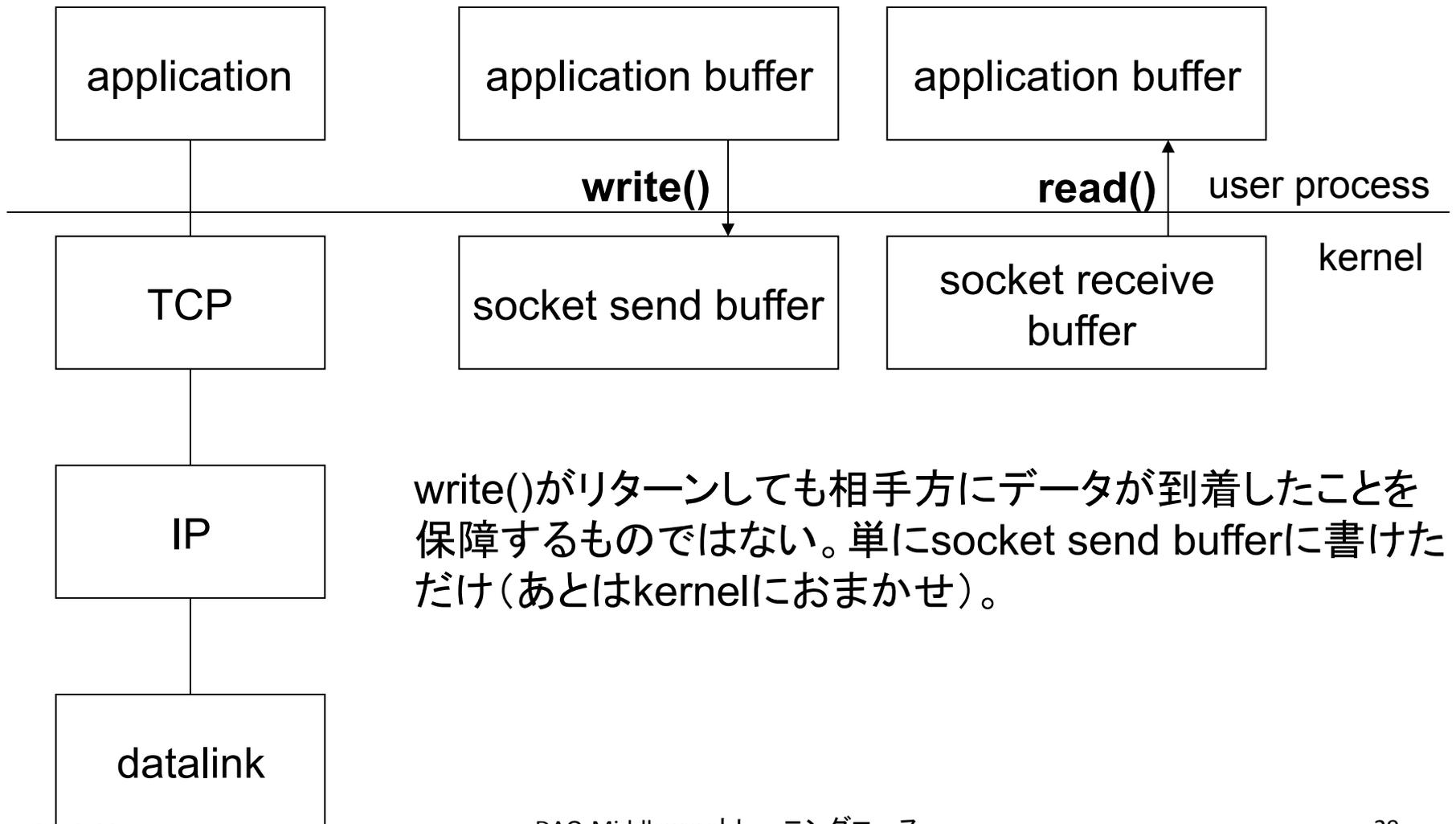
```
0.000000 0.000000 connect start (プログラムの出力)
0.000363 0.000363 IP 192.168.0.100.35005 > 192.168.0.101.13: S
0.000489 0.000126 IP 192.168.0.101.13 > 192.168.0.100.35005: S
0.000536 0.000047 IP 192.168.0.100.35005 > 192.168.0.101.13: . ack 1 win 1460
0.000583 0.000047 connect returns (プログラムの出力)

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() (1)

```
#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() (2)

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

注意点

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。

デフォルトでは読むデータがない場合は読めるまでブロックする
(そこでプログラムの動作が止まる)

ブロックと無限ループ

```
int ch;  
ch = getchar();  
/* CPU を消費しない */
```

```
for (;;) {  
    ;  
}  
/* CPU を消費する */
```

```

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 {
                return -1;
            }
        }
        else if (nread == 0) { /* EOF */
            break;
        }
        nleft -= nread;
        buf_ptr += nread;
    }
    return (nbytes - nleft);
}

```

readn()

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);`
使えるOSは限られる(Linuxでは使える)

write()

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

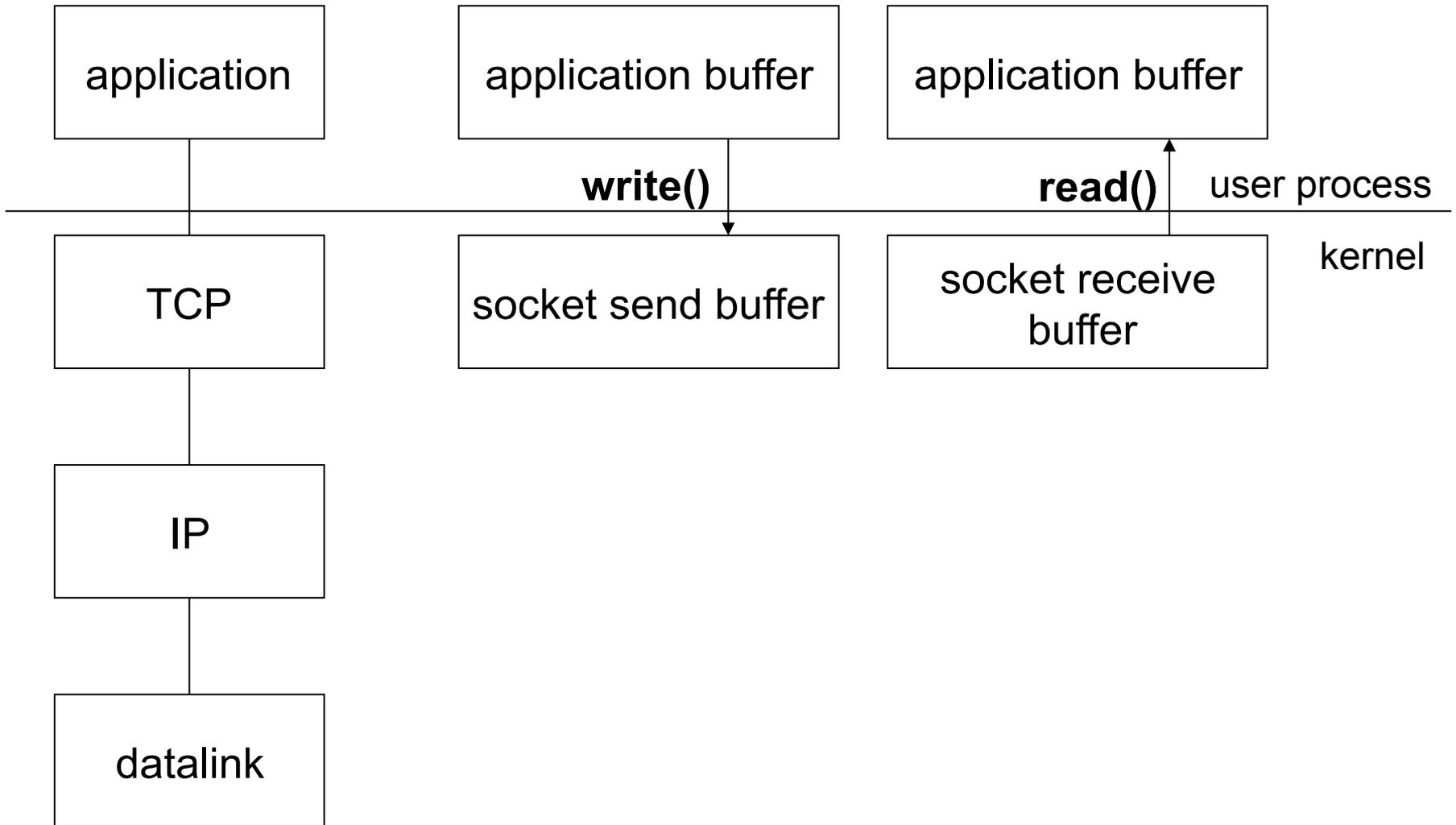
```
unsigned char buf[4];
ssize_t n;
```

```
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 SEND 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)。

socket send/receive bufferの大きさ



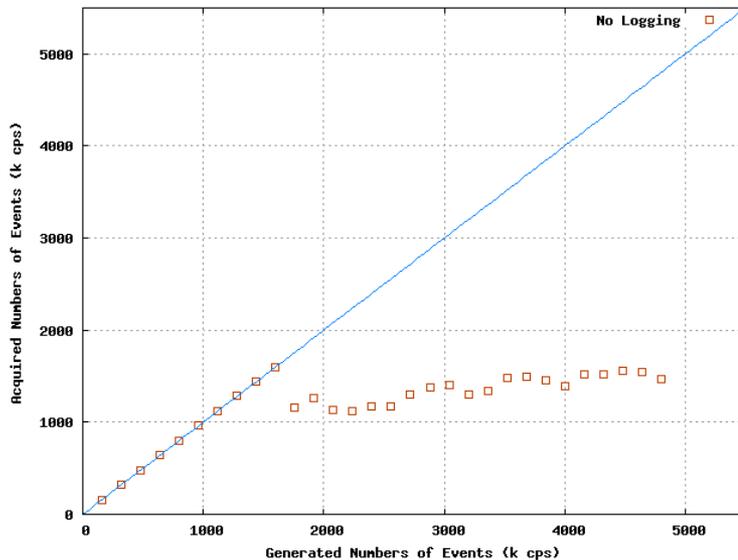
socket send/receive bufferの大きさの調整

- 受信に関してはLinuxでは自動調節機能がある
- 多重読み出しを行うときにはあらかじめ大きくしておかないと性能がでないことが多い

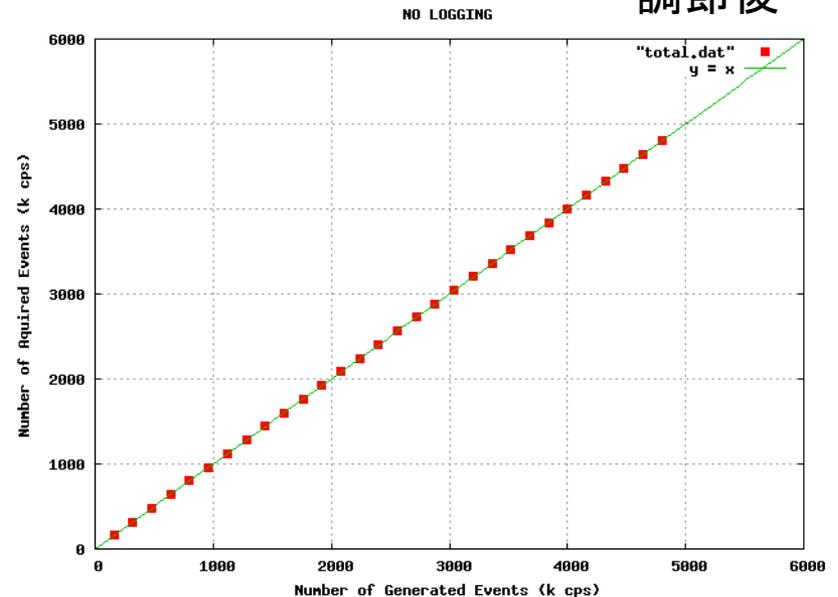
```
echo 0 > /proc/sys/net/ipv4/tcp_timestamps
echo 1 > /proc/sys/net/ipv4/tcp_moderate_rcvbuf
echo 4194304 > /proc/sys/net/core/wmem_max
echo 4194304 > /proc/sys/net/core/rmem_max
echo 4194304 > /proc/sys/net/core/wmem_default
echo 4194304 > /proc/sys/net/core/rmem_default
echo 4096 131072 4194304 > /proc/sys/net/ipv4/tcp_rmem
echo 4096 131072 4194304 > /proc/sys/net/ipv4/tcp_wmem
# 131072 128kB がデフォルト値。もっと大きくしておかないとだめな場合もある
# あるいはソケット個別に大きくすることもできる(setsockopt())
```

ソケットレシーブバッファの大きさ調節による改善例

調節前



調節後



多重読み出しで複数モジュールから読み出し
各モジュールは同一レートでデータを送ってくるようにセット
読むモジュール数を1, 2, 3, と増加させていった。

ここまでのまとめ

- ソケットファイルディスクリプタを取得するとあとは通常のファイルの読み書きと同様
- ファイルを読むときとは違って指定したサイズが必ずしも読めるとは限らない。指定したサイズ必ず読みたければそのような関数を作る必要がある。

ネットワークバイトオーダー (1)

- `unsigned char buf[10];`
アドレスは`buf[0]`, `buf[1]`, `buf[2]`の順に大きくなる
- `unsigned char buf[10];`
`write(sockfd, buf, 10);`
とすると`buf[0]`, `buf[1]`, `buf[2]` ...の順に送られる。
- `read(sockfd, buf, 10);`
きた順に`buf[0]`, `buf[1]`, `buf[2]`に格納される。

ネットワークバイトオーダー (2)

```
// intがどういう順番でメモリーに  
// 入っているか調べるプログラム
```

```
#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;
```

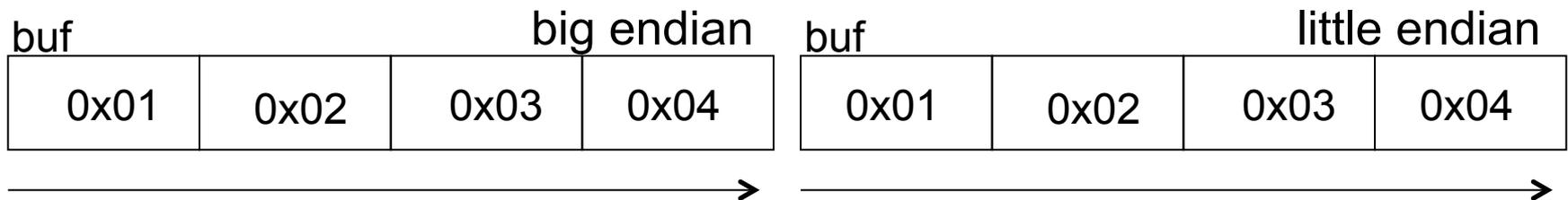
```
}
```

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(3)

0x 01 02 03 04 の順に送られてきたデータをread(sockfd, buf, 4)で読んだ場合



big endianでは $0x\ 01020304 = 16909060$

little endianでは $0x\ 04030201 = 67305985$

ネットワークバイトオーダーはbig endian

ネットワークバイトオーダー(4)

- ホストオーダー \leftrightarrow ネットワークバイトオーダー変換関数
 - htonl (host to network long)
 - htons (host to network short)
 - ntohl (network to host long)
 - ntohs (network to host short)

daytime client (1)

- xinetd内蔵サーバー daytime (port 13)
- /etc/xinetd.d/daytime-streamにて
disable = no
に変更して service xinetd restart
- telnet localhost 13
すると現在日時が表示される

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

#include <arpa/inet.h>
#include <netinet/in.h>

#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#define MAXLINE 1024

int main(int argc, char *argv[])
{
    unsigned char line[MAXLINE + 1];
    struct sockaddr_in servaddr;
    char *ip_address = "127.0.0.1";
    int port = 13;
    int sockfd, n;
```

```
servaddr.sin_family = AF_INET;
servaddr.sin_port   = htons(port);

n = inet_pton(AF_INET, ip_address, &servaddr.sin_addr);
if (n < 0) {
    perror("inet_pton");
    exit(1);
}
else if (n == 0) {
    fprintf(stderr, "invalid address %s", ip_address);
    exit(1);
}

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

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

```

for ( ; ; ) {
    n = read(sockfd, line, MAXLINE);
    if (n < 0) {
        perror("read");
        exit(1);
    }
    else if (n == 0) {
        printf("EOF\n");
        break;
    }
    line[n] = '\0'; /* string termination */
    printf("%s\n", line);
}

if (close(sockfd) < 0) {
    perror("close");
    exit(1);
}

return 0;
}

```

```

0123456789012345678901234 5 6
07 AUG 2012 13:02:10 JST\r\n\0

```

```

[daq@localhost daytimeclient]$ ./daytimeclient | hexdump -vC
00000000  30 37 20 41 55 47 20 32  30 31 32 20 31 33 3a 30  |07 AUG 2012 13:0|
00000010  32 3a 31 30 20 4a 53 54  0d 0a 0a 45 4f 46 0a    |2:10 JST...EOF.|

```

情報のありか

- Manual Page
- 本

Manual Pages

- セクション

- 1 (Utility Program) Linuxだとこの他
- 2 (System call) – 3P (Posix)
- 3 (Library)
- 4 (Device)
- 5 (File format)
- 6 (Game)
- 7 (Misc.)
- 8 (Administration)

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。

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)
NAME

Linux Programmer's Manual

READ(2)

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(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

Utility

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

gettimeofday()で現在時刻の取得

```
#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 (gettimeofday(&end, NULL) < 0) {
    err(1, "gettimeofday");
}
```

```
/* 時間差をとるには引き算してもよいし、timersub()関数を使ってもよい */
timersub(&end, &start, &diff);
printf("%ld.%06ld\n", result.tv_sec, result.tv_usec);
```

```
struct timeval {
    time_t      tv_sec;      /* seconds */
    suseconds_t tv_usec;    /* microseconds */
};
```

Linuxではgettimeofday()を1,000,000回繰り返して1秒以下(CPUに依存する)

ナノ秒まで必要なとき

```
clock_gettime(CLOCK_REALTIME, &ts);
```

コンパイル時に-lrtが必要

```
struct timespec {  
    time_t    tv_sec;           /* seconds */  
    long      tv_nsec;         /* nanoseconds */  
};
```

余談: 最近のファイルシステムのタイムスタンプはナノ秒まで記録されている

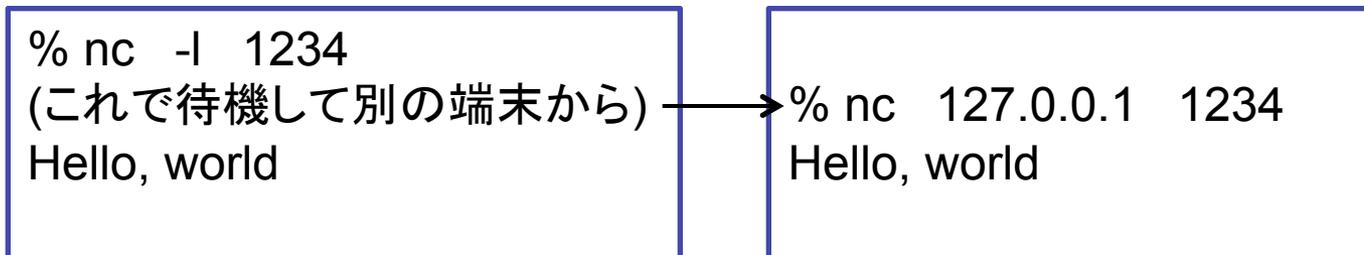
```
% touch X
```

```
% ls -l --full X
```

```
-rw-rw-r-- 1 sendai sendai 0 2012-08-02 15:02:55.362116699 +0900 X
```

nc (netcat)

- nc - arbitrary TCP and UDP connections and listens



- `nc 192.168.0.16 > datafile` で接続してデータをとってみる
- `nc 192.168.0.16 | tee log.dat | prog_histo`

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

- 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おことがある。

```
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

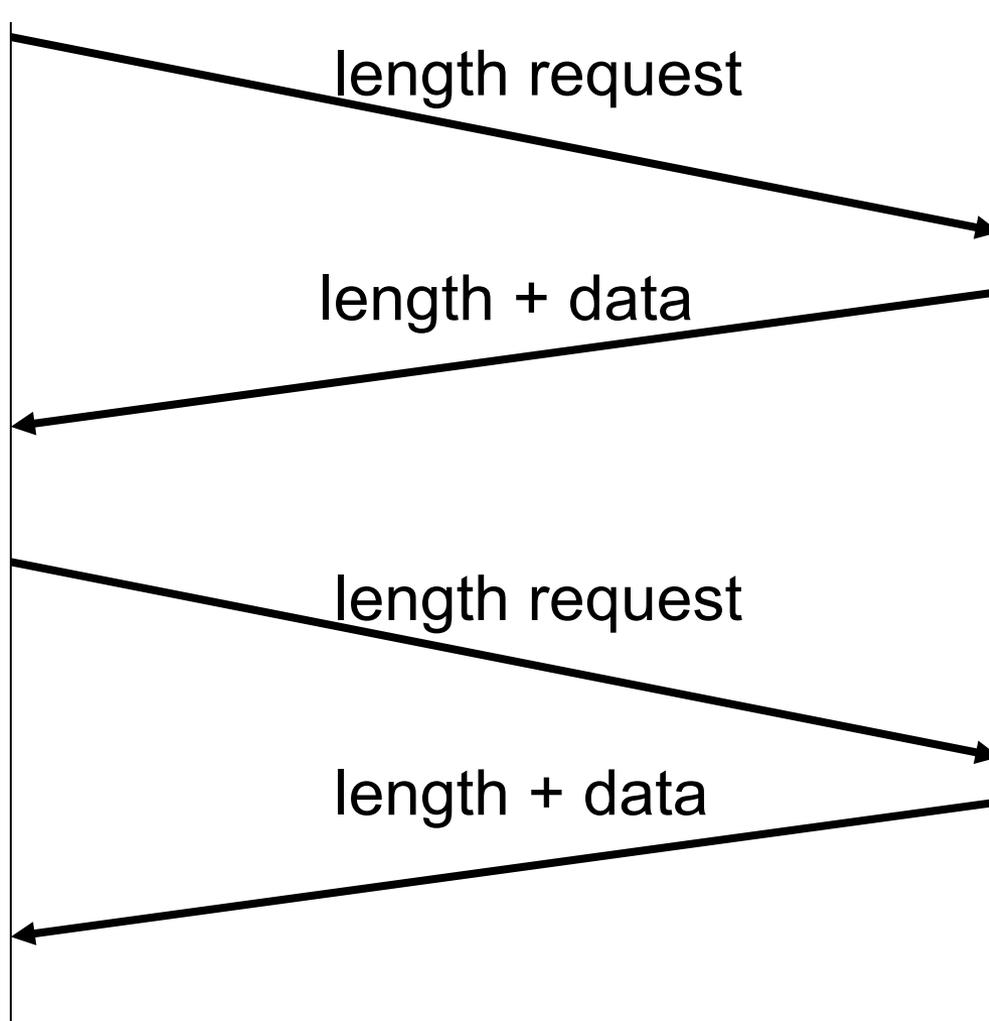
- tcpdumpの時刻情報と同じ時刻フォーマットでログを出すようにしておいてtcpdumpをとりつつプログラムを走らせあとからマージする:

```
(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)
```

wireshark

- yum install wireshark-gnome (GUIつきののをインストールする)
- Ethernet、IP、TCPのヘッダがどこか色つきで表示してくれるので便利

File Edit View Go Capture Analyze Statistics Telephony Tools Help

Filter: Expression... Clear Apply

No. .	Time	Source	Destination	Protocol	Info
1	0.000000	IntelCor_8b:60:14	Broadcast	ARP	Who has 192.168.0.32
2	0.000160	02:00:c0:a8:00:20	IntelCor_8b:60:14	ARP	192.168.0.32 j
3	0.000169	192.168.0.101	192.168.0.32	TCP	44779 > telnet
4	0.000224	192.168.0.32	192.168.0.101	TCP	telnet > 44779
5	0.000253	192.168.0.101	192.168.0.32	TCP	44779 > telnet
6	0.000281	192.168.0.101	192.168.0.32	TELNET	Telnet Data ..
7	0.000349	192.168.0.32	192.168.0.101	TCP	telnet > 44779
8	0.002032	192.168.0.32	192.168.0.101	TELNET	Telnet Data ..
9	0.002068	192.168.0.101	192.168.0.32	TCP	44779 > telnet
10	0.002139	192.168.0.101	192.168.0.32	TELNET	Telnet Data ..
11	0.002175	192.168.0.32	192.168.0.101	TCP	telnet > 44779
12	0.004013	192.168.0.32	192.168.0.101	TELNET	Telnet Data ..
13	0.004179	192.168.0.101	192.168.0.32	TCP	44779 > telnet
14	0.004280	192.168.0.32	192.168.0.101	TCP	telnet > 44779
15	0.004298	192.168.0.101	192.168.0.32	TCP	44779 > telnet

Frame 1 (42 bytes on wire, 42 bytes captured)
 Ethernet II, Src: IntelCor_8b:60:14 (00:1b:21:8b:60:14), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
 Address Resolution Protocol (request)

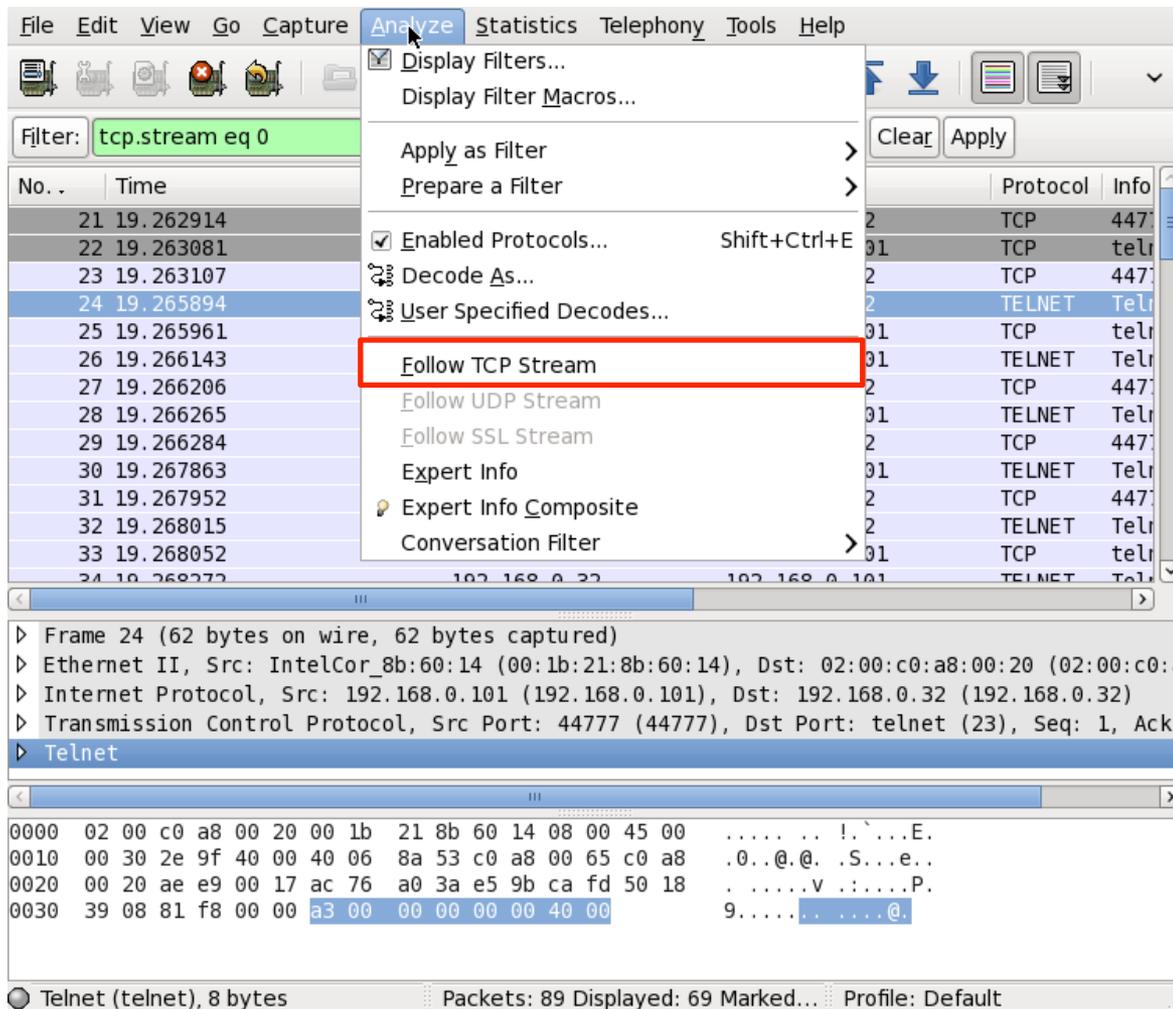
```

0000  ff ff ff ff ff ff 00 1b 21 8b 60 14 08 06 00 01  ..... !\.....
0010  08 00 06 04 00 01 00 1b 21 8b 60 14 c0 a8 00 65  ..... !\.....e
0020  00 00 00 00 00 00 c0 a8 00 20  .....
  
```

eth1: <live capture in progress> ... Packets: 15 Displayed: 15 Marked: 0 Profile: Default

MACアドレスからベンダーを調べて表示する(らしい)

wireshark



- 複数のTCPセッションがあっても
Analyze→Follow TCP Streamで追跡可能

File Edit View Go Capture Analyze Statistics Telephony Tools Help

Filter: tcp.stream eq 0 Expression... Clear Apply

No. .	Time	Source	Destination	Protocol	Info
21	19.262914	192.168.0.101	192.168.0.32	TCP	44777
22	19.263081	192.168.0.32	192.168.0.101	TCP	telnet
23	19.263107	192.168.0.101	192.168.0.32	TCP	44777
24	19.265894	192.168.0.101	192.168.0.32	TELNET	Telnet
25	19.265961	192.168.0.32	192.168.0.101	TCP	telnet
26	19.266143	192.168.0.32	192.168.0.101	TELNET	Telnet
27	19.266206	192.168.0.101	192.168.0.32	TCP	44777
28	19.266265	192.168.0.32	192.168.0.101	TELNET	Telnet
29	19.266284	192.168.0.101	192.168.0.32	TCP	44777
30	19.267863	192.168.0.32	192.168.0.101	TELNET	Telnet
31	19.267952	192.168.0.101	192.168.0.32	TCP	44777
32	19.268015	192.168.0.101	192.168.0.32	TELNET	Telnet
33	19.268052	192.168.0.32	192.168.0.101	TCP	telnet
34	19.268272	192.168.0.32	192.168.0.101	TELNET	Telnet

Frame 24 (62 bytes on wire, 62 bytes captured)

- Ethernet II, Src: IntelCor_8b:60:14 (00:1b:21:8b:60:14), Dst: 02:00:c0:a8:00:20 (02:00:c0:a8:00:20)
- Internet Protocol, Src: 192.168.0.101 (192.168.0.101), Dst: 192.168.0.32 (192.168.0.32)
- Transmission Control Protocol, Src Port: 44777 (44777), Dst Port: telnet (23), Seq: 1, Ack: 1
- Telnet

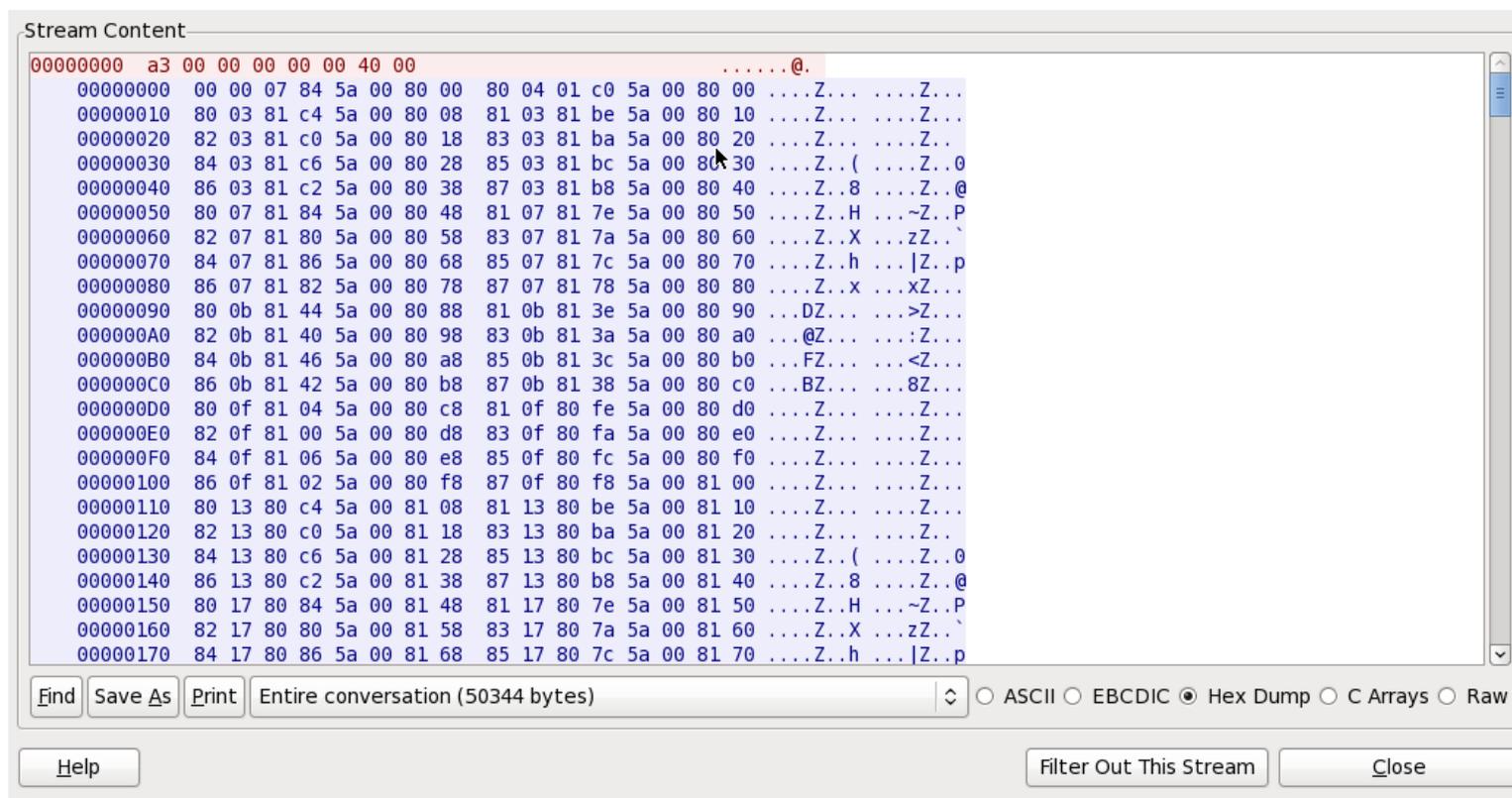
```

0000  02 00 c0 a8 00 20 00 1b 21 8b 60 14 08 00 45 00  .... .. !. ...E.
0010  00 30 2e 9f 40 00 40 06 8a 53 c0 a8 00 65 c0 a8  .0..@.@. .S...e..
0020  00 20 ae e9 00 17 ac 76 a0 3a e5 9b ca fd 50 18  . ....v :....P.
0030  39 08 81 f8 00 00 a3 00 00 00 00 00 40 00      9..... .. @.
  
```

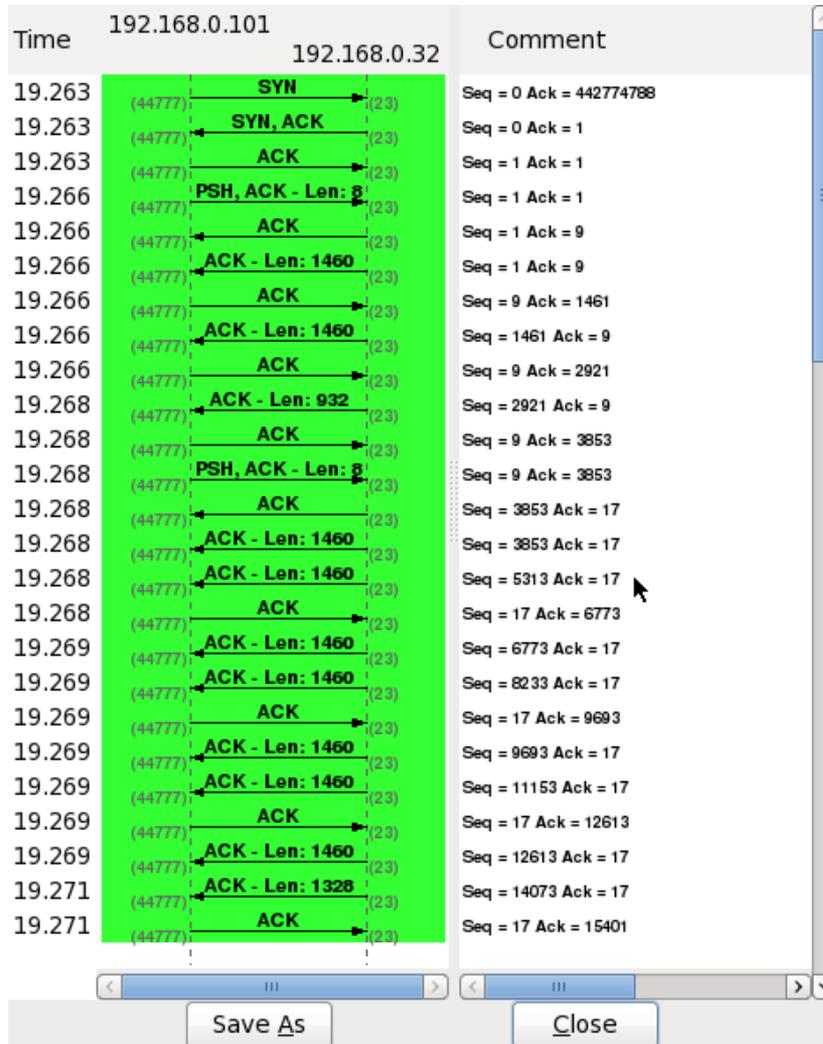
Telnet (telnet), 8 bytes Packets: 89 Displayed: 69 Marked... Profile: Default

wireshark

- データのダンプもできる



wireshark



- フローグラフ

tcpflow

- tcpdump、wireshark等でキャプチャしたファイルからデータフローを取り出すことができる
- 同様なソフトウェアは他にもある

多重読み出し

```
read(sockfd_0, buf_0, sizeof(buf_0));  
read(sockfd_1, buf_1, sizeof(buf_1));  
read(sockfd_2, buf_2, sizeof(buf_2));
```

とするとsockfd_0で止まると、sockfd_1が読めるようになっていてもプログラムが進行しない。

読めるようになったものをどんどん読むにはselect()あるいはLinuxならepoll()を使う。
あるいはpthreadを使う。