ネットワークコンピューティング
第10回

中山　雅哉 (m.nakayama@cnl.t.u-tokyo.ac.jp)
関谷　勇司 (sekiya@nc.u-tokyo.ac.jp)

そろそろ最終課題を。。。

- なにかプログラムを組んでみましょう
  - チャットサーバの作成
    - select / epoll or thread を利用
  - IPv4 / IPv6 デュアルスタック対応簡易 Web サーバ
    - 多種の OS で動作するよう作成
  - Web Proxy を作成
    - 簡単な Proxy なら fork() / thread() にて作成可能
- 授業で学んだネットワークの知識を生かしたもの

課題例：IPv4/IPv6 Web サーバ

- IPv4 と IPv6 両方でサービスを提供できる簡易 Web サーバの作成
- ポイント
  - IPv4 ソケットと IPv6 ソケットの使い分け
  - 複数の OS でも動く汎用的なネットワークプログラミング
  - 多重化
  - HTTP プロトコルの実装

課題例：チャットサーバ

- 複数人でメッセージ交換を行えるチャットサーバ
  - クライアントも作成する必要があるかも
  - とりあえず telnet でつないで使える、でも OK
- ポイント
  - 複数人同時接続
  - 1対多へのメッセージ配信
  - メンバーの動的な増減
  - IPv4 / IPv6 への対応

課題例：ダウンロードソフトウェア

- http もしくは ftp でファイルをダウンロードするソフトウェアの作成
  - ./file_get http://lecture.nc.u-tokyo.ac.jp/images/XXX.pdf
- ポイント
  - そんなに難しくはない
  - URI の解釈 (http://, ftp://)
  - http プロトコル、ftp プロトコルの実装
  - つながらなかったときのリトライ

課題例 : Web プロキシ

- Web プロキシ
  - ユーザからの HTTP リクエストを一度受け止めて、Web サーバとのデータのやりとりを中継するソフトウェア
- ポイント
  - 簡単な HTTP リクエストの実装
  - IPv4/IPv6 対応
  - 多重化

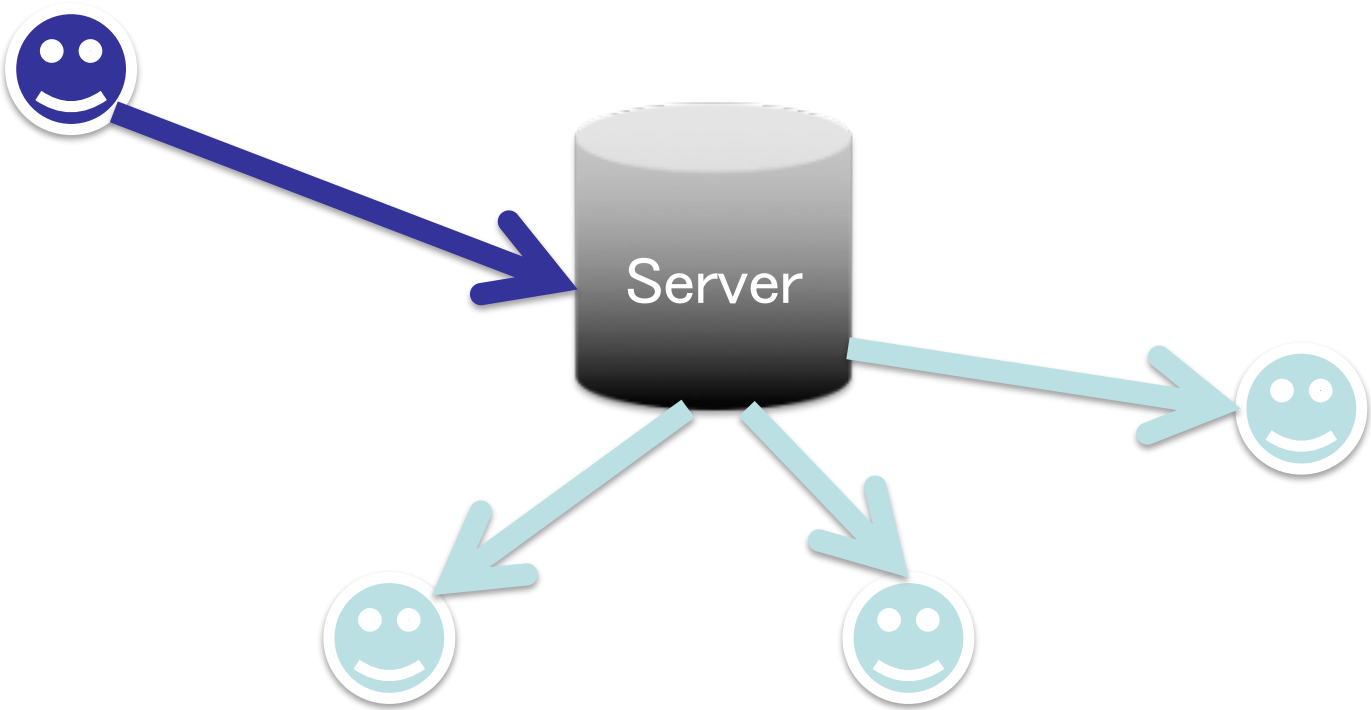
最終課題

- 締め切り
  - 7/31(火)　　23:59
  - lecture@sekiya-lab.info まで

  - 氏名 / 学生番号
  - 何を作ったのか
  - 作ったモノのソースコード
  - 動作を確認できるログもしくは画面ダンプ
  - 授業への感想、要望等

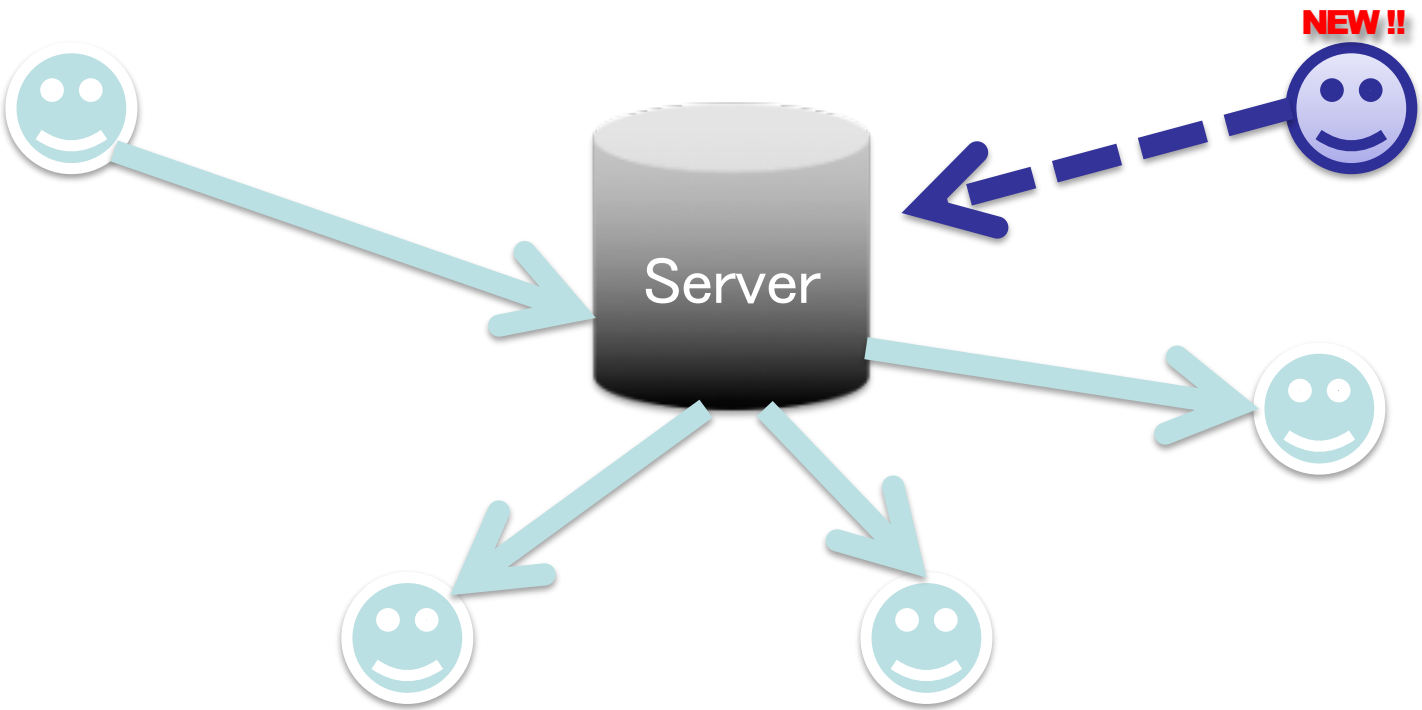
チャットサーバの場合

- 一人が送ったメッセージを複数人に伝えなければならない
  - どの socket からメッセージが届くか常に監視



チャットサーバの場合

- 当然、新たなユーザが加わる場合もあれば、ユーザが去っていく場合もある



チャットサーバを作る場合には

- 非同期書き込み / 読み込みが発生
  - fork()
    - 1ユーザに対して 1プロセスが割り当てられるので、プロセス同士でメッセージを交換する必要あり
  - thread
    - fork() と同様、1ユーザに対して 1 thread が割り当てられるが、メモリ空間は共有されるため、排他制御のみで可能
  - select() / epoll()
    - ひとつのプロセスの中で複数の socket を非同期に扱うことが可能

epoll() の使い方

- #include <sys/epoll.h>

- epoll_create()
  - epoll ファイルディスクリプタの作成
- epoll_ctl()
  - epoll で待ち受けるファイルディスクリプタの登録
- epoll_wait()
  - 登録されたファイルディスクリプタにデータが届くのを待ち受け

epoll_create()

- int epoll_create(int size);
  - size は以前は登録するファイルディスクリプタ数を与えていたが、現在 (Linux 2.6.8 以降) では 0 より大きい数を与えておけば良い
  - 成功すれば非負の値を返す

epoll_ctl()

- int epoll_ctl(int epfd, int op, int fd,
  struct epoll_event *event);
  - fd で指定されたファイルディスクリプタに対し op で指定された操作を行う
  - op
    - EPOLL_CTL_ADD / EPOLL_CTL_MOD / EPOLL_CTL_DEL
  - event
    - EPOLLIN : 対象となる fd で read 可能
    - EPOLLOUT : 対象となる fd で write 可能

epoll_wait()

- int epoll_wait(int epfd, struct epoll_event *events,
  int maxevents, int timeout);
- events
  - 利用可能なイベントが格納され返る
- maxevents
  - 0 より大きい値を指定
- timeout
  - epoll_wait のタイムアウト値
  - -1 で無限に待つ

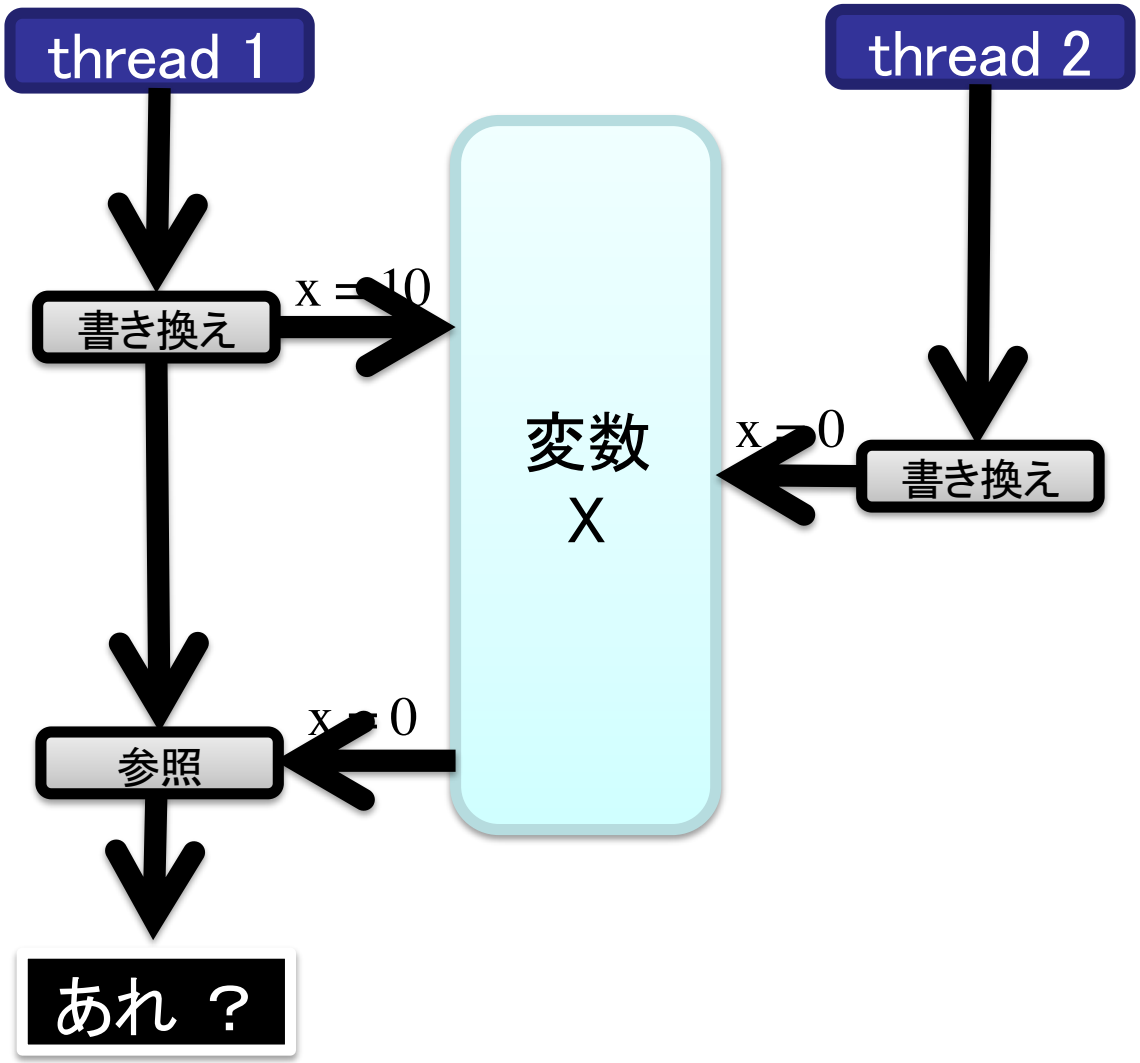
epoll() で標準入力の待受

```
epfd = epoll_create(1);
if (epfd < 0) {
  printf("epoll_create() failed¥n");
  return -1;
}

memset(&event, 0, sizeof(event));
event.events  = EPOLLIN;
event.data.fd = 0;

if (epoll_ctl(epfd, EPOLL_CTL_ADD, 0, &event) < 0) {
  printf("epoll_ctl() failed¥n");
  return -1;
}

while(1) {
  nfd = epoll_wait(epfd, &events, 1, -1);
  if (nfd < 0) {
    printf("epoll_wait() failed¥n");
  } else if (nfd == 0) {
    printf("epoll_wait() timeout¥n");
  } else if (nfd) {
    read(fd, s, sizeof(s));
    printf("%s",s);
    memset(&s, 0, sizeof(s));
}
```

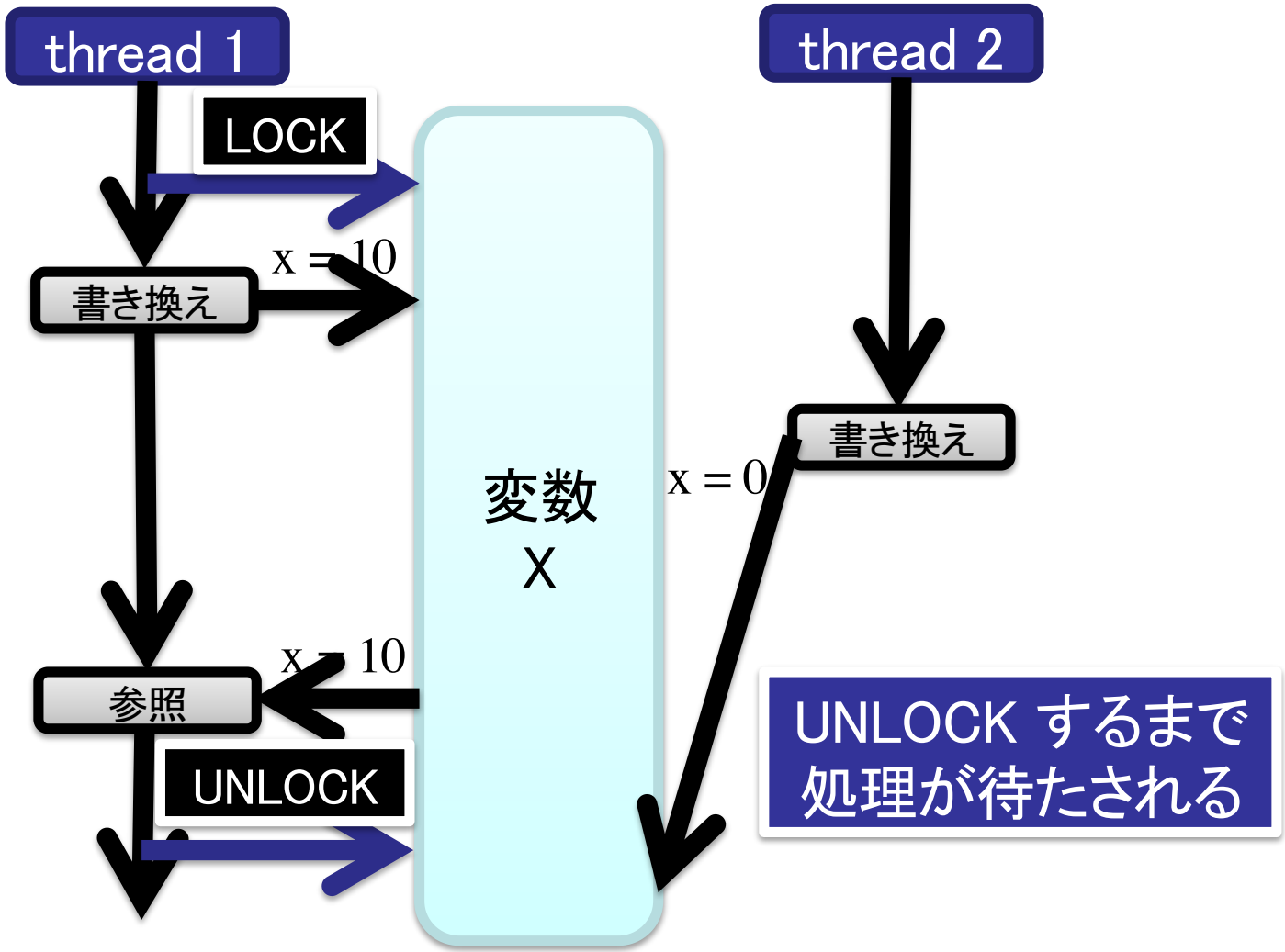
pthread での変数共有

- thread ではグローバル変数やメモリ領域は共有される
  - 複数のスレッドが同時に変数を書き換える場合も発生する
  - 意図しない動作を引き起こすかも
- mutex (mutual exclusion) という仕組みによって同時書き込みを防ぐことが可能

mutex を用いない場合



mutex でロックをかけた場合



mutex の利用例 (1)

```
pthread_mutex_t mutex;

int main(int argc, char *argv[])
{
  pthread_t id1, id2;
  pthread_mutex_init(&mutex, NULL);

  pthread_create(&id1, NULL, thread1, NULL);
  pthread_create(&id2, NULL, thread2, NULL);

}
```

mutex の利用例 (2)

```
void* thread1(void* param)
{
  int i;
  while(1){
    for(i=0;i<10;i++){
      printf("%d:",count1);
      count1++;
      sleep(1);
    }
    printf("¥n");
    sleep(1);
  }
}
```

```
void* thread2(void* param)
{
  while(1){
    count1=0;
}
```

実行結果

```
count1:0:0:0:0:0:0:0:0:0:0:
count1:0:0:0:0:0:0:0:0:0:0:
count1:0:0:0:0:0:0:0:0:0:0:
count1:0:0:0:0:0:0:0:0:0:0:
```

mutex の利用例 (3)

```
void* thread1(void* param)
{
  int i;
  while(1){
    pthread_mutex_lock(&mutex);
    for(i=0;i<10;i++){
      printf("%d:",count1);
      count1++;
      sleep(1);
    }
    pthread_mutex_unlock(&mutex);
    printf("¥n");
    sleep(1);
  }
}
```

```
void* thread2(void* param)
{
  while(1){
    count1=0;
}
```

実行結果

```
count1:0:1:2:3:4:5:6:7:8:9:
count1:0:1:2:3:4:5:6:7:8:9:
count1:0:1:2:3:4:5:6:7:8:9:
count1:0:1:2:3:4:5:6:7:8:9:
```

mutex 利用上の注意点

- mutex_lock が長いと他のスレッドの実行が止まってしまったままになる
  - 並列化の意味がなくなる
- mutex_lock したまま unlock せずに他処理に移る
  => Deadlock 状態
  - 他のスレッドが何も処理できない状態となる
  - lock したままそのスレッドが消滅したら無限に待ち状態

Curses プログラム

- GUI で作ってもいいのですが。。。
  - 授業用のサーバは CUI しか提供していない
  - printf 等の表示では 1行単位でしか表示できない
  - しかも表示する場所を自由に指定することは難しい

- CUI で自在に文字を表示
  - curses ライブラリ
  - libcurses, libncurses という名前で存在

```
#include <curses.h>
gcc XXX.c -lcurses
```

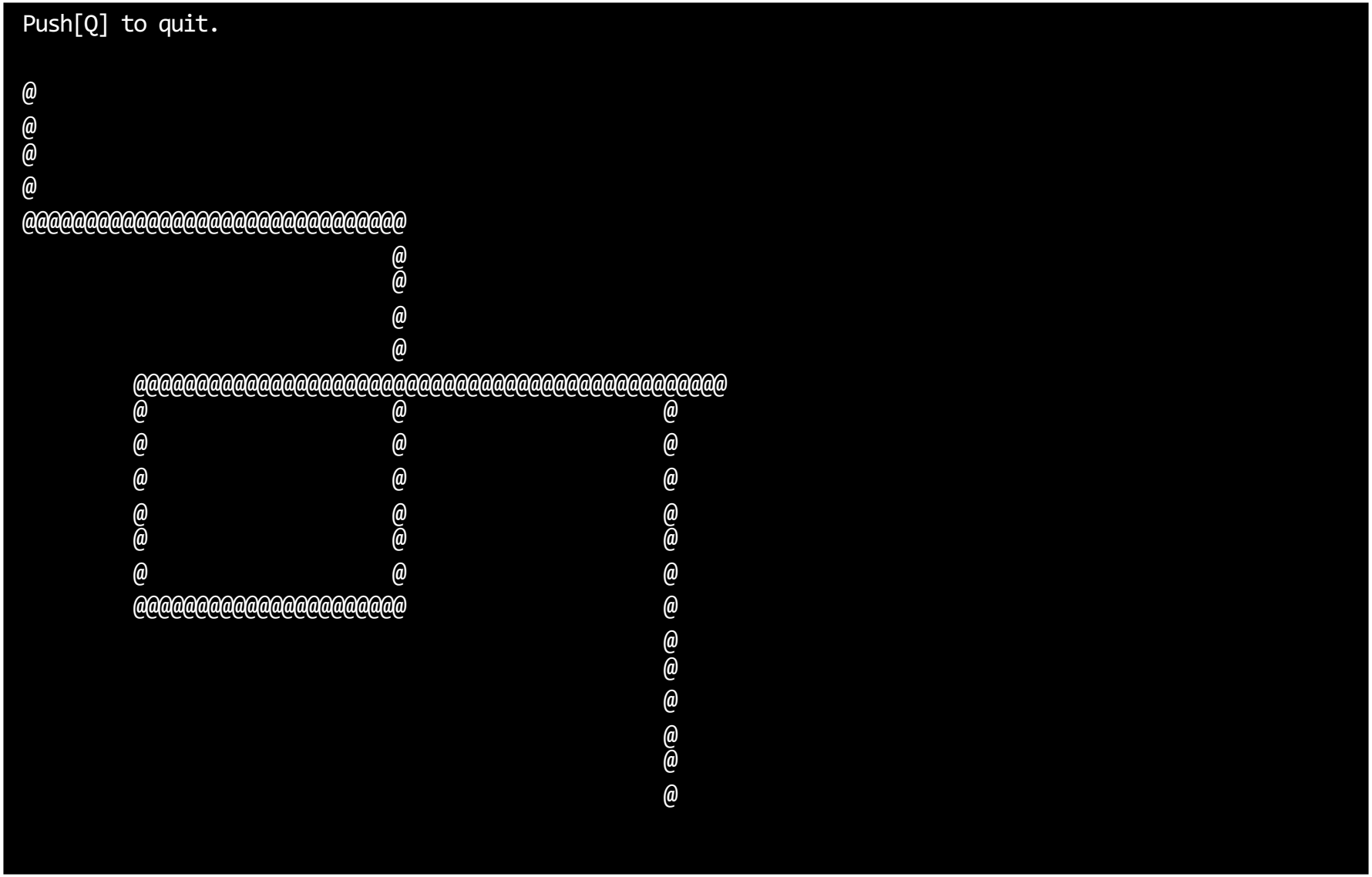
Curses の基本的な使い方

- initscr()
  - 画面の初期化 : curses を使うことを宣言
- clear()
  - 画面の消去
- cbreak()
  - ユーザからの文字入力を即座に受け取ることを指定
- mvaddstr(int y, int x, const char *str)
  - 座標 (x,y) の位置に文字列 *str を表示
- mvaddch(int y, int x, const chtype ch)
  - 座標 (x,y) の位置に一文字 ch を表示

サンプルプログラム

```
int x=0, y=1, ch;
char v='@';

int main(void) {
    initscr();
    noecho();
    clear();
    cbreak();

    mvaddstr(0, 0, "Push[Q] to quit.");
    while ((ch = getch()) != 'Q') {
            switch (ch) {
                    キー入力にあわせた処理
            }
            mvaddch(y, x, v);
    }
    endwin();
}
```

画面上でカーソル移動しながら文字を書く



while() -> getch() -> swich() での処理

```
while ((ch = getch()) != 'Q') {
        switch (ch) {
        case 2:
                if (x != 0)
                        x--;
                break;
        case 14:
                if (y < LINES)
                        y++;
                break;
        case 16:
                if (y != 1)
                        y--;
                break;
        case 6:
                if (x < COLS)
                        x++;
                break;
        default:
               break;
        }
        mvaddch(y, x, v);
    }
```

色とかもつけられる

- 色の初期化と定義

```
start_color();
init_pair(2, COLOR_YELLOW, COLOR_BLUE);
```

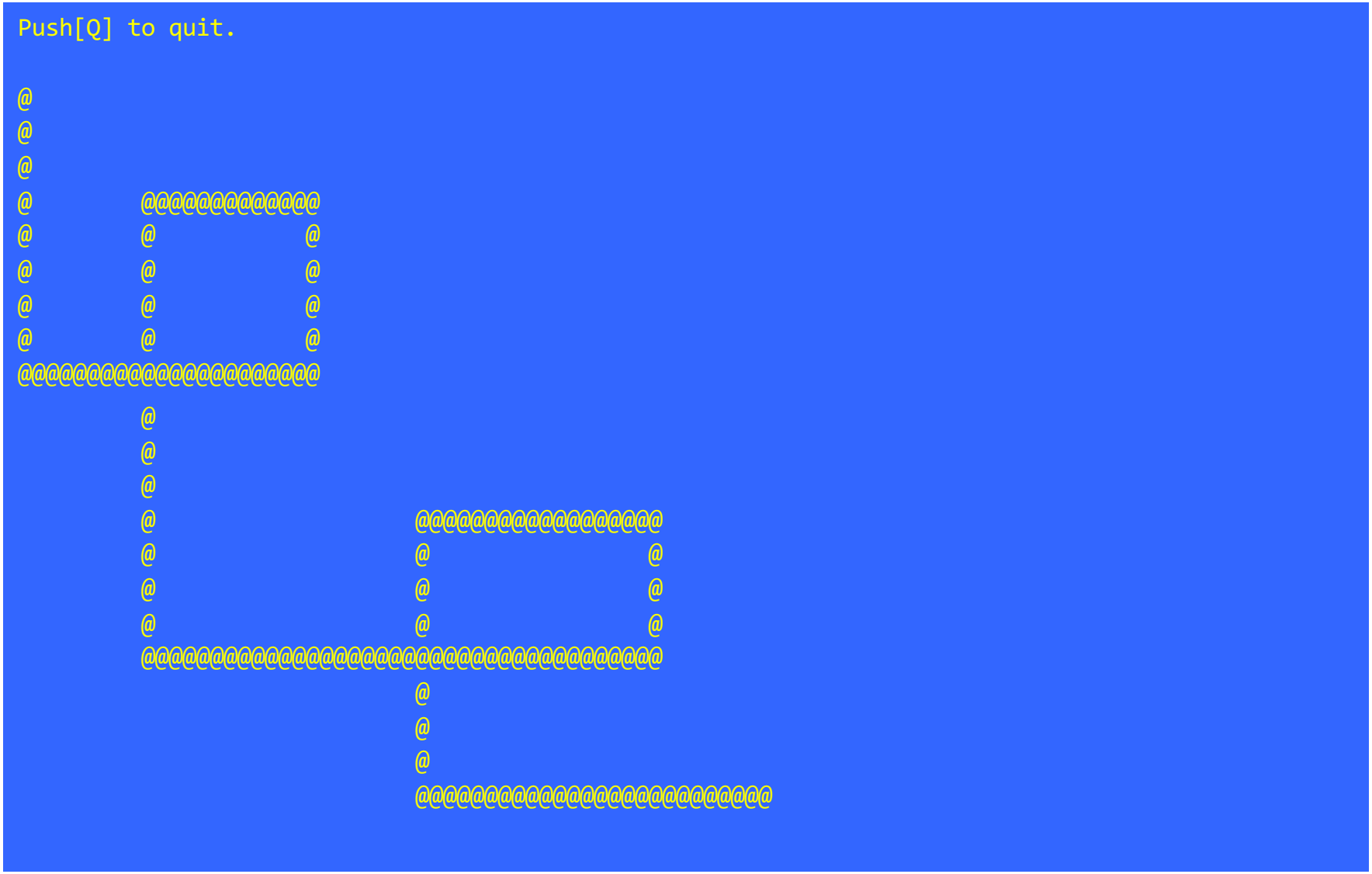
- 背景色の定義

```
bkgd(COLOR_PAIR(2));
```

- 文字色の定義

```
attrset(COLOR_PAIR(2));
```

色指定した例



利用用途

- メッセンジャークライアントでの表示に適用
  - ユーザ毎に色を変える

```
sekiya : Hello
nakayama : Good Bye
```

- ダウンロードクライアントでのメッセージ強調表示

```
./file_get   http://lecture.nc.u-tokyo.ac.jp/images/AAA.pdf
Connecting To･･･ lecture.nc.u-tokyo.ac.jp
Timed Out.
Retrying･･･
```

多重ダウンロードツールとか

```
Downloading 4 Files…

http://lecture.nc.u-tokyo.ac.jp/images/AAA.pdf
|========== 38%

http://lecture.nc.u-tokyo.ac.jp/images/BBB.pdf
|========= 27%

http://www.u-tokyo.ac.jp/ISO/CCC.iso
|==================== 46%

http://www.nc.u-tokyo.ac.jp/MOVIE/DDD.mpg
Connecting…
```

ネットワークのデバッグ

- 通信している中身をパケット(データ)レベルで解析
  - 自作アプリケーション等が期待した動作をしない場合
  - 何が起こっているのか
- パケット解析ツール
  - tcpdump
  - wireshark
- 授業用ホスト (login1, login2) では tcpdump を提供
  - 通常は root 権限を持っていないと実行できない
  - 授業用ホストでは誰でも実行可にしている

tcpdump の使い方

- 全てをダンプ
  - tcpdump -i [Interface Name] -n

- 使用例

```
sekiya@lecture-vm1:~$ sudo /usr/sbin/tcpdump -i eth0 -n
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
09:34:12.659899 IP6 2001:200:180:452:216:3eff:fe91:5a19.22 >
2001:200:180:299:217:f2ff:fe0e:d6d0.50086: Flags [P.], seq 2627729767:2627729959, ack 2463089541,
win 238, options [nop,nop,TS val 711230666 ecr 53499278], length 192
09:34:12.660704 IP6 2001:200:180:452:216:3eff:fe91:5a19.22 >
2001:200:180:299:217:f2ff:fe0e:d6d0.50086: Flags [P.], seq 192:464, ack 1, win 238, options
[nop,nop,TS val 711230667 ecr 53499278], length 272
09:34:12.660884 IP6 2001:200:180:299:217:f2ff:fe0e:d6d0.50086 >
2001:200:180:452:216:3eff:fe91:5a19.22: Flags [.], ack 192, win 65535, options [nop,nop,TS val
53499278 ecr 711230666], length 0
09:34:12.661453 IP6 2001:200:180:299:217:f2ff:fe0e:d6d0.50086 >
2001:200:180:452:216:3eff:fe91:5a19.22: Flags [.], ack 464, win 65535, options [nop,nop,TS val
53499278 ecr 711230667], length 0
```

絞った使い方

- IPv4 のみ
  - tcpdump -i eth0 -n ip
- www.u-tokyo.ac.jp に関連する通信のみ
  - tcpdump -i eth0 -n host www.u-tokyo.ac.jp
- TCP port 80 番に関連する通信のみ
  - tcpdump -i eth0 -n tcp port 80
- 全ての条件に適合する通信のみ
  - tcpdump -i eth0 -n ip and host www.u-tokyo.ac.jp and tcp port 80

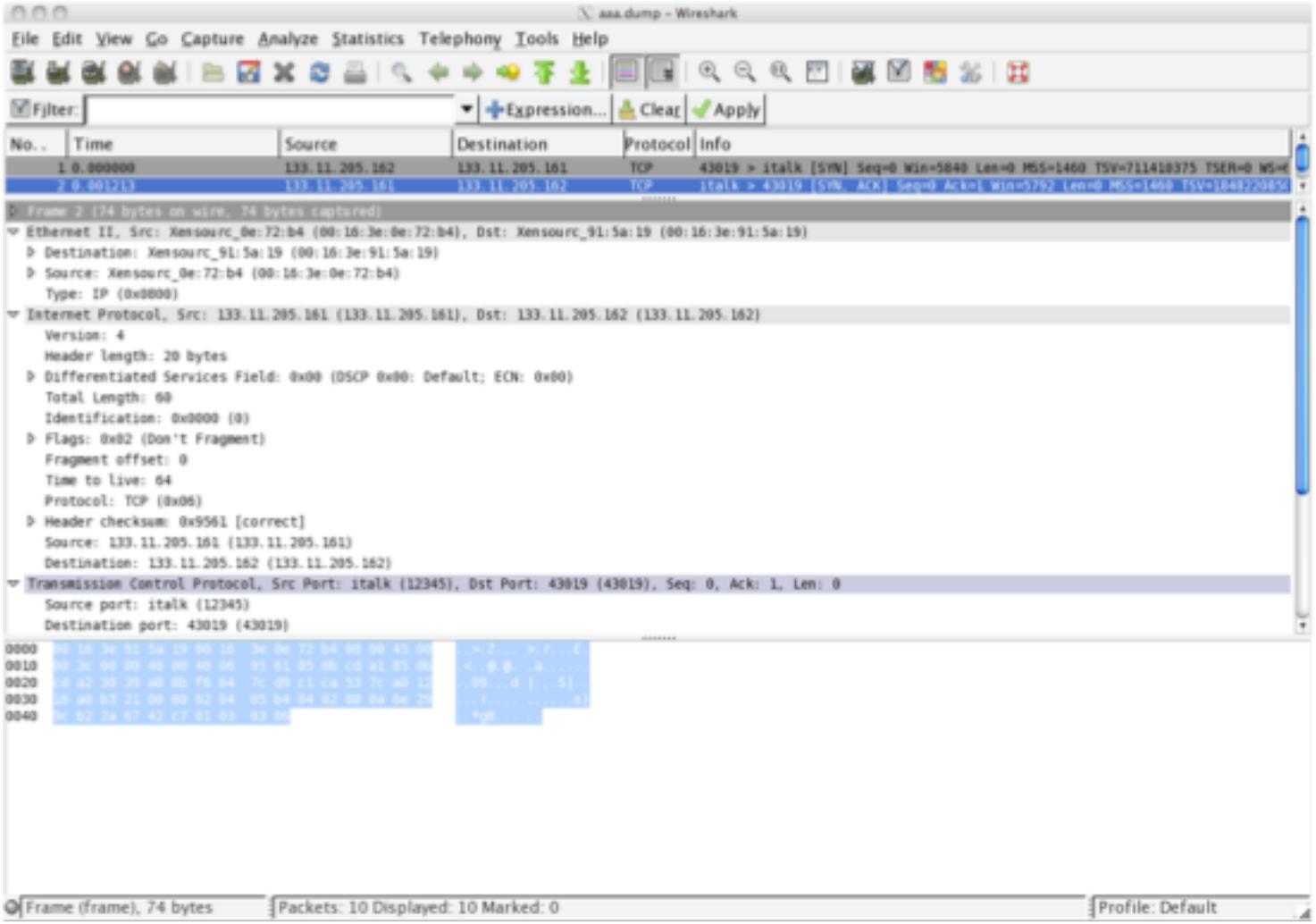
tcpdump による解析例

- login2:12345 で自作 web サーバを上げ
  login1 からアクセスした場合の通信

```
sekiya@lecture-vm1:~/EXAMPLE$ sudo tcpdump -i eth0 host lecture-vm2.nc.u-tokyo.ac.jp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
09:44:28.469265 IP lecture-vm1.nc.u-tokyo.ac.jp.43017 > lecture-vm2.nc.u-tokyo.ac.jp.12345: Flags [S],
seq 1640493337, win 5840, options [mss 1460,sackOK,TS val 711384619 ecr 0,nop,wscale 6], length 0
09:44:28.470437 IP lecture-vm2.nc.u-tokyo.ac.jp.12345 > lecture-vm1.nc.u-tokyo.ac.jp.43017: Flags [S.],
seq 2524487162, ack 1640493338, win 5792, options [mss 1460,sackOK,TS val 1848195094 ecr
711384619,nop,wscale 6], length 0
09:44:28.470456 IP lecture-vm1.nc.u-tokyo.ac.jp.43017 > lecture-vm2.nc.u-tokyo.ac.jp.12345: Flags [.],
ack 1, win 92, options [nop,nop,TS val 711384619 ecr 1848195094], length 0
09:44:28.471450 IP lecture-vm2.nc.u-tokyo.ac.jp.12345 > lecture-vm1.nc.u-tokyo.ac.jp.43017: Flags [.],
seq 1:1449, ack 1, win 91, options [nop,nop,TS val 1848195094 ecr 711384619], length 1448
09:44:28.471463 IP lecture-vm1.nc.u-tokyo.ac.jp.43017 > lecture-vm2.nc.u-tokyo.ac.jp.12345: Flags [.],
ack 1449, win 137, options [nop,nop,TS val 711384619 ecr 1848195094], length 0
09:44:28.471481 IP lecture-vm2.nc.u-tokyo.ac.jp.12345 > lecture-vm1.nc.u-tokyo.ac.jp.43017: Flags [P.],
seq 1449:2049, ack 1, win 91, options [nop,nop,TS val 1848195094 ecr 711384619], length 600
09:44:28.471485 IP lecture-vm1.nc.u-tokyo.ac.jp.43017 > lecture-vm2.nc.u-tokyo.ac.jp.12345: Flags [.],
ack 2049, win 182, options [nop,nop,TS val 711384619 ecr 1848195094], length 0
09:44:28.471489 IP lecture-vm2.nc.u-tokyo.ac.jp.12345 > lecture-vm1.nc.u-tokyo.ac.jp.43017: Flags [F.],
seq 2049, ack 1, win 91, options [nop,nop,TS val 1848195094 ecr 711384619], length 0
09:44:28.471738 IP lecture-vm1.nc.u-tokyo.ac.jp.43017 > lecture-vm2.nc.u-tokyo.ac.jp.12345: Flags [F.],
seq 1, ack 2050, win 182, options [nop,nop,TS val 711384619 ecr 1848195094], length 0
09:44:28.473323 IP lecture-vm2.nc.u-tokyo.ac.jp.12345 > lecture-vm1.nc.u-tokyo.ac.jp.43017: Flags [.],
ack 2, win 91, options [nop,nop,TS val 1848195094 ecr 711384619], length 0
```

wireshark での解析例

- tcpdump -i eth0 -w aaa.dump 等でファイルに落とす



login サーバでの tcpdump

- 使えるようにしておきました
  - 人のトラフィックも見えるので注意
  - 自分が使っているポート番号限定で
    - tcpdump -i eth0 -n tcp port 40155
  - 動いているプロセス
    - ps
    - ps auxw