UNIXシステムプログラミング

第10回 クライアント・サーバのための プログラミング(1)

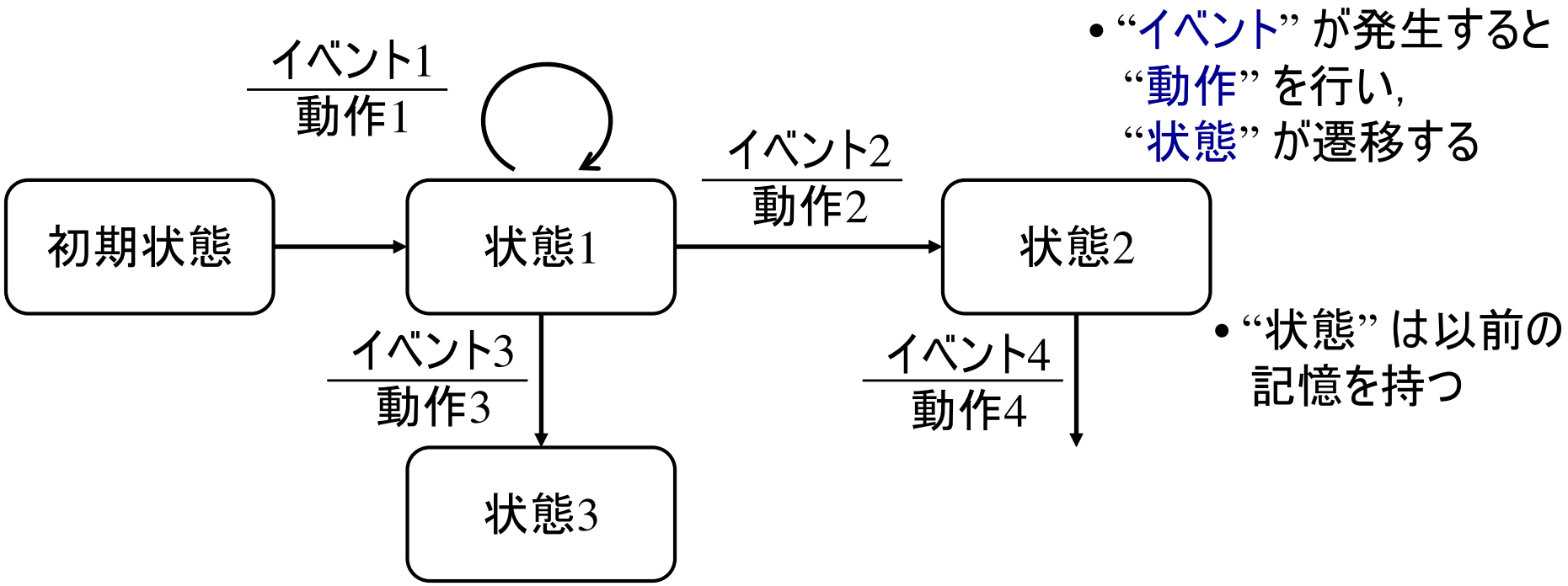
2019年12月13日

情報工学科

寺岡文男

有限状態機械

- 有限状態機械 (finite state machine: FSM)
- 有限オートマトン (finite automaton: FA)
  - 有限個の状態，状態の遷移，動作の組合せからなるモデル
- クライアント/サーバの動作はFSMで表せることが多い



FSMをプログラムへ

- “動作” の部分をそれぞれひとかたまりの処理とする
  - e.g., 関数にする
- メインループの処理
  - イベント待ち (e.g., `select()`で受信待ち，後述)
  - 現在の状態と発生したイベントから処理関数を決定
  - 処理関数内では “動作” を実行し，状態を更新
  - イベント待ちへ戻る

- 処理のロジックはFSMを書くときに考える
- 次にFSMを黙々とプログラムに落とす

FSMをプログラムへ：例1

```
for (;;) {
  // イベント待ち（例：受信待ち）
  switch (状態) {
  case 状態1:
    switch (イベント) {
    case イベント1:
      動作1();
      break;
    case イベント2:
      動作2();
      状態 = 状態2;
      break;
    case イベント3:
      動作3();
      状態 = 状態3;
      break;
    . . .
    default:
      エラー処理;
      break;
    }  // switch(イベント)
    break;
  case 状態2:
    switch (イベント) {
    case イベント4:
      動作4();
      状態 = ...;
      ...
    } // switch(イベント)
    break;
  case 状態3:
    ...
  }   // switch(状態)
}     // for()
```

FSMをプログラムへ: 例2 (1)

```
void f_act1(...), f_act2(...), f_act3(...), ...;

struct proctable {
  int status;
  int event;
  void (*func)(...);
} ptab[ ] = {
  {stat1, event1, f_act1},
  {stat1, event2, f_act2},
  {stat1, event3, f_act3},
  {stat2, event4, f_act4},
  {stat3, event5, f_act5},
  {stat3, event6, f_act6},
  ...,
  {0, 0, NULL}
};

int status;
```

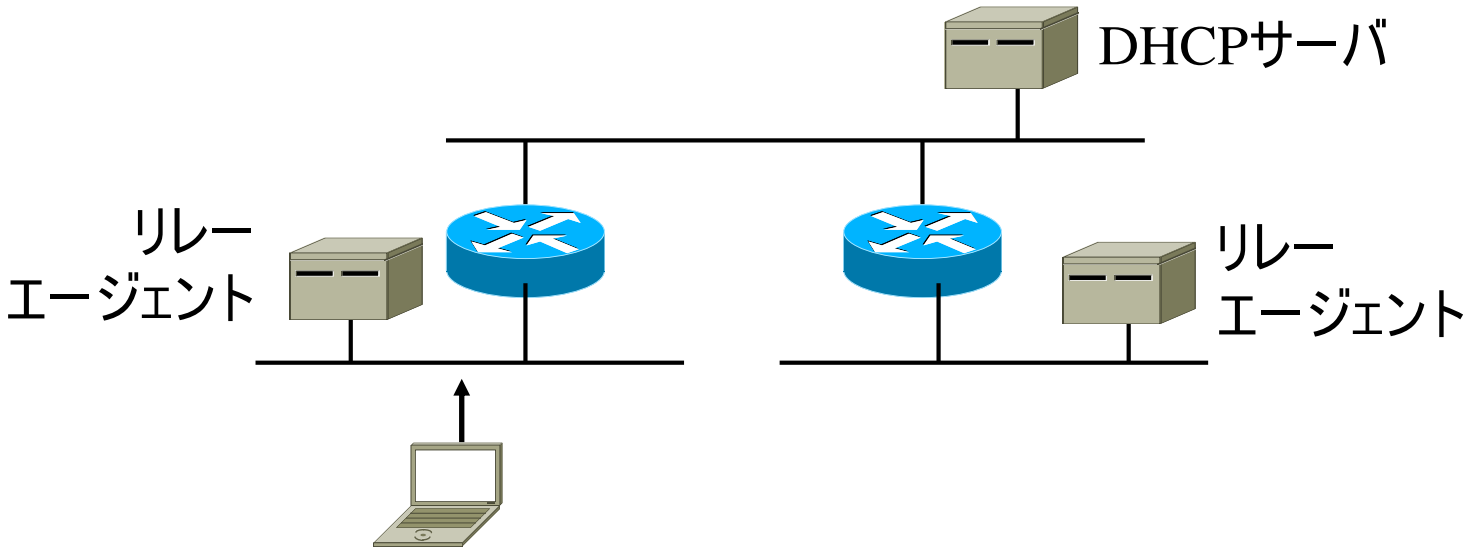
FSMをプログラムへ: 例2 (2)

```
int main( )
{
  struct proctable *pt;
  int event;

  for (;;) {
    event = wait_event(...);
    for (pt = ptab; pt->status; pt++) {
      if (pt->status == status &&
                    pt->event == event) {
        (*pt->func)(...);
        break;
      }
    if (pt->status == 0)
      エラー処理;
  }
}
```

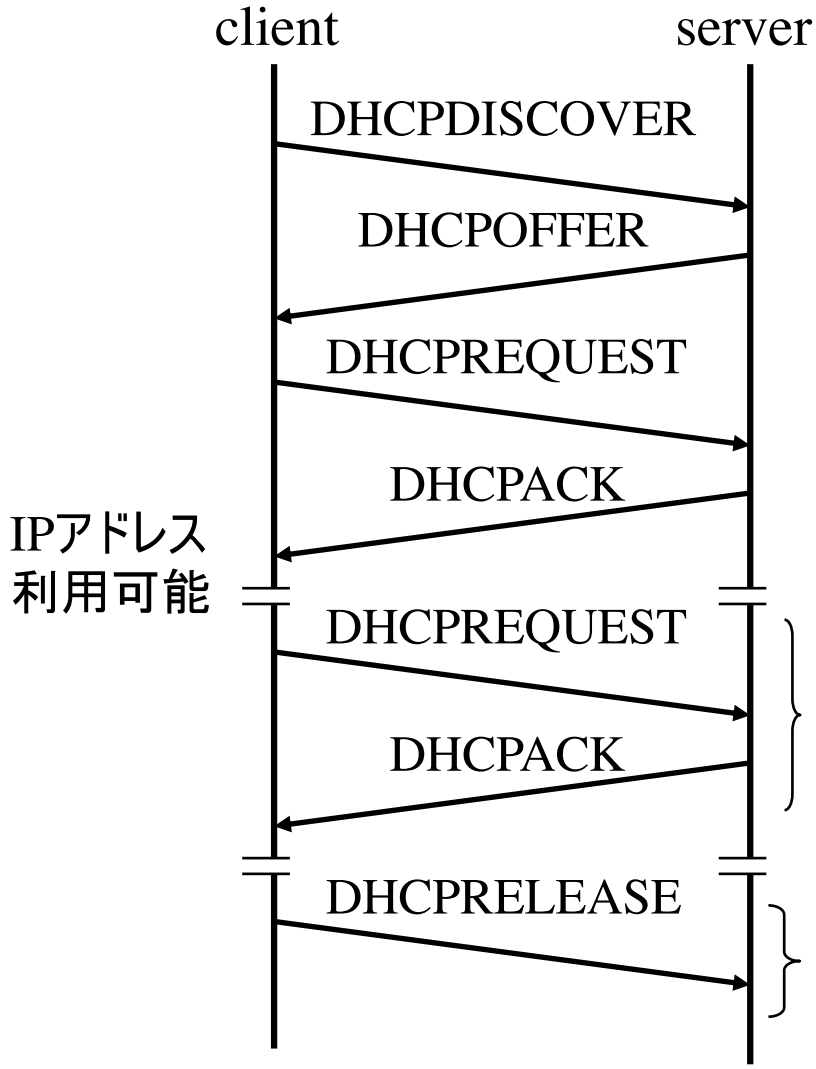
```
void f_act1(...)
{
  ...;
}

void f_act2(...)
{
  ...;
  status = stat2;
}

...
```

例：DHCP

- DHCP: Dynamic Host Configuration Protocol
  - ネットワークに接続したコンピュータにIPアドレスを自動的に配布するプロトコル
- クライアント・サーバモデルに基づく
  - リレーエージェントを配置することにより，複数セグメントを1つのサーバで管理可能

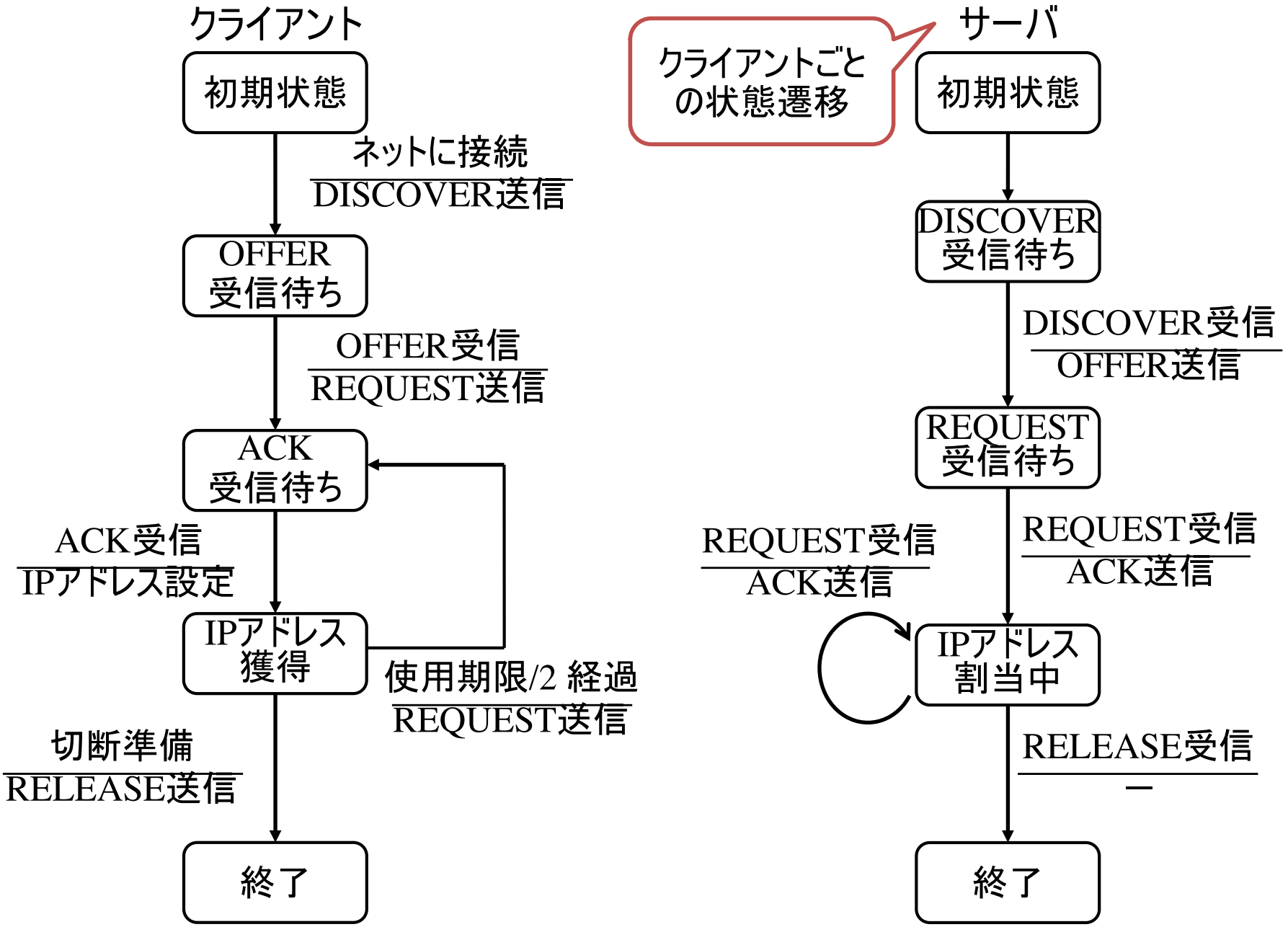


DHCPの動作



- DHCPDISCOVER (C → S)
  - サーバの発見
- DHCPOFFER (S → C)
  - 利用可能なIPアドレスの通知
- DHCPREQUEST (C → S)
  - IPアドレスの使用要求
- DHCPACK (S → C)
  - 確認応答
- 利用期限が切れそうになると再度要求
- 明示的にIPアドレスを解放してもよいし解放しなくてもよい

DHCPの状態遷移図 (正常な動作のみ)



ノンブロッキング受信

- recvfrom( )やrecv( )はパケットを受信するまでリターンしない ＝ ブロッキング受信
- パケットを受信していなくてもすぐに関数やシステムコールからリターンする ＝ ノンブロッキング受信
  - パケット受信を待つ間，別の処理を行える
- recvfrom( )やrecv( )の場合，flagsにMSG_DONTWAIT を指定すればよい

- やってはいけない例

```
while (recv(s, buf, sizeof buf, MSG_DONTWAIT) == 0) {
    // パケット未受信のときの処理
    I/O待ちなどのない処理;
}
```

select( ): 複数の受信待ち

```
#include <sys/select.h>

int
select(int nfds, fd_set *readfds, fd_set *writefds,
       fd_set *exceptfds, struct timeval *timeout);

FD_SET(fd, &fdset);
FD_CLR(fd, &fdset);
FD_ISSET(fd, &fdset);
FD_ZERO(&fdset);
```

- 引数
  - nfds: 記述子集合の中で調べる記述子の数
  - readfds: 入力できる状態かを調べる記述子集合
  - writefds: 出力できる状態かを調べる記述子集合
  - exceptfds: 例外処理がペンディング状態かを調べる記述子集合
  - timeout: タイムアウト

select( ): 複数の受信待ち

- nfds の設定の仕方
  - 記述子集合のなかで0番目から nfds – 1 番目までの記述子が調べられる
- 入力
  - 記述子集合の調べたい記述子をセットする (FD_SET( )を使用する)
- 出力結果
  - 記述子集合の中で、処理が可能な記述子がセットされている
  - select( )自体は処理が可能な記述子の総数を返す
- 記述子集合を扱うためのマクロが用意されている
  - FD_SET(fd, &fdset);　　fdsetにfdをセットする
  - FD_CLR(fd, &fdset);　　fdsetでfdをリセットする
  - FD_ISSET(fd, &fdset);　　fdsetにfdがセットされているかの確認
  - FD_ZERO(&fdset);　　fdsetをゼロクリアする

select( ): タイムアウトの指定

- timeout にはタイムアウトになる時間を設定
  - struct timeval は以下を参照
- null pointer を指定すると処理可能な記述子が現れるまで待つ

```
struct timeval {
    long tv_sec;          // 秒
    long tv_usec;         // マイクロ秒
};
```

select( )の使用例

```
int s;
struct sockaddr_in myskt;
fd_set rdfds;

if ((s = socket(...)) < 0) {
    // エラー処理
}
// myskt の設定
if (bind(s, ...) < 0) {
    // エラー処理
}

FD_ZERO(&rdfds);
FD_SET(0, &rdfds);
FD_SET(s, &rdfds);
```

```
if (select(s+1, &rdfds, NULL,
      NULL, NULL) < 0) {
    // エラー処理
}
if (FD_ISSET(0, &rdfds)) {
    // 標準入力から入力
}
if (FD_ISSET(s, &rdfds)) {
    // パケット受信
}
```

select( )におけるタイムアウトの指定

- 第5引数の struct timeval *timeout でタイムアウトを指定
  - 入力，出力，例外処理で処理可能なものが無い場合は，指定した時間が経過すると select( )は0を返す.

gettimeofday( ): 現在時刻の取得

```
#include <sys/time.h>

int
gettimeofday(struct timeval *tp, struct timezone *tzp);

struct timeval {
        long tv_sec;        // 1970年1月1日からの秒
        long tv_usec;       // マイクロ秒
};
```

- 引数
  - struct timeval *tp: 現在時刻が設定される
  - struct timezone *tzp: 現在は使用されていない. NULLを指定すればよい.

ctime( ): 時刻表示の文字列への変換

```
#include <time.h>

char *ctime(time_t *clock);

struct tm *localtime(time_t *clock);
```

- 引数
  - time_t *clock: gettimeofday( )の第一引数 struct timeval のメンバーである tv_secへのポインタとする.
    ((time_t *) へのキャストを忘れないように)
- 返り値
  - “Fri Dec 10 08:09:35 2010¥n¥0” のような26文字固定長の文字列が格納されている領域へのポインタ

time( ): epochからの経過時間

```
#include <time.h>

time_t time(time_t *tloc);
```

- 機能
  - 1970年1月1日0時0分 (UTC)からの経過時間(秒)を得る
- 引数
  - non nullの場合，上記の結果が格納される
- 返り値
  - 上記の値

課題4: 擬似DHCPクライアント/サーバ

- テキストの7.5節に示した擬似DHCPクライアント (mydhcpc) と擬似DHCPサーバ (mydhcps)を作成しなさい.

- 両方のプログラムとも状態遷移図を描きなさい.
  - エラー処理なども含めなさい.

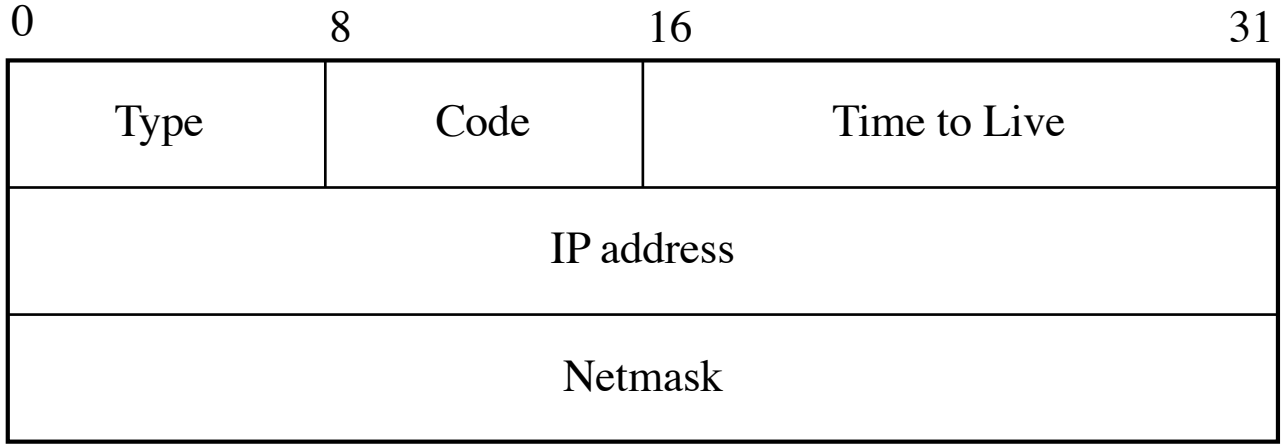
- 状態遷移図はPDFファイルに変換してプログラムのソースファイルと一緒に提出しなさい.

課題4: 擬似DHCPクライアント/サーバ

- 提出方法
  - 締切 : 2020年1月9日(木) 20:00JST
  - 課題用のディレクトリを作成 (e.g., “mydhcp”)
  - “mydhcp” ディレクトリに必要なすべてのファイルを保存
    - ソースファイル，Makefile，状態遷移図(PDF)，コンフィグファイル
    - 余分なファイルは消すように．(*.o やバックアップファイルなど)
  - memo.txt というファイルに学籍番号，氏名，どの程度できているか，などを記入すること．
  - “mydhcp” ディレクトリの1つ上のディレクトリで tar を実行
    - 例 : tar czf mydhcp.tgz mydhcp
  - keio.jp に *.tgz をアップロード

ヒント: パケットフォーマットの定義

- 疑似DCHPヘッダ




```
struct dhcph {
    uint8_t    type;
    uint8_t    code;
    uint16_t   ttl;
    in_addr_t  address;
    in_addr_t   netmask;
};
```

練習問題

- テキスト7章の練習問題1
  - キーボードとネットワークの両方からの入力待ち
  - キーボードからの入力を画面に表示
  - 受信したデータを表示
  - キーボードからの入力が “FIN” だったら実行終了
  - 受信データが “FIN” だったら実行終了