ネットワークプログラミング

Jacques Garrigue, 2004 年 12 月 7 日

この講義は，ネットワークを介してプログラム間通信を行う技術と，それを利用して，プログラムの共同作業あるいは分散アルゴリズムの実装を教えようとしている．実際に OS のシステムコールを使って，物理的なネットワークにアクセスするというとても具体的な話があれば，分散アルゴリズムの正しさを証明するための理論的なモデルなどの抽象的な話もある．実習もあるので，理論から実装ではなく，逆に具体的な実装からもっと複雑なものを実現するための抽象化へ発展していくのである．

1 TCP/IP とソケット

第一回目として，ネットワークを介したプログラム間通信の基本的な方法を見る．

ほとんどの OS が提供する二つの概念が中心になる．

1.1 IP アドレスとポート番号

まず，通信を行おうと思ったら，相手を指定する方法が提供されなければならない．人間の世界では住所 (または電話番号) と氏名が使われる．氏名だけだと，同姓同名があるかも知れないし，その人にどうやって連絡を取ればいいか分からない．住所だけだと，そこにいるどの人が受け取るべきか分からない．(携帯の電話番号は例外かも知れない)

IP アドレス インターネットなどのネットワークでは，住所としては IP アドレスを使う．原則として，ネットワークにつながっている全てのコンピュータが異る IP アドレスを持っている．現在，主に使われている IPv4 では，IP アドレスは 32bit に整数で表現される．(moose.math.nagoya-u.ac.jp などの “.”

で区切られた文字列は IP アドレスではない．ホスト名と言い，別の仕組みで IP アドレスに変換される．)

IP アドレスを書くとき，その 32bit の整数をそのまま書くのではなく，8bit ごとに値を区切る．例えば，前の moose というマシンの IP アドレスは “172.16.254.3” である．書くマシンの IP アドレスを調べるには，Unix/Linux では host か nslookup を使う．

```
$ host moose.math.nagoya-u.ac.jp
moose.math.nagoya-u.ac.jp has address 172.16.254.3
```

ちなみに，外のネットワークから上記のコマンドを使うと IP アドレスは 133.6.130.3 になる．同じマシンがいくつかの IP アドレスを持つことがある．少なくとも，全てのマシンは自分にしか分からないもう一つの IP アドレスを持っている．ループバック・アドレスと言われている 127.0.0.1 という特別の IP アドレスである．自分を指すアドレスなので，同じマシン内にあるものを指すことしかできない．

ポート番号 物理的な通信はコンピュータ間で行われるものだが，実際に通信しているのはプログラム同士 (あるいは，そのプログラムを利用している人間) である．IP アドレスだけではプログラムが特定できない．プログラムはどう指定すればいい．プログラムが実行されるときには，プロセス番号という一意的なものは割り振られるが，それは実行される度に変わるものなので使えない．そのプログラムのファイル名も考えられるが，プログラムは人間より合理的なので，通信するときに知りたいのは相手の名前ではなく，役割である．コンピュータによって，同じ役割を果すプログラムは違う名前になっているかも知れない．あるいは，同じプログラムは複数の役割を担っているかも知れない．そのために，あらかじめ決められてポート番号を使う．逆に，自分に役割があるのではなく，相手から呼び返して欲しい場合は，適当にまだこのマシンの誰もが使っていないポート番号を使えばいい．

役割を表しているポート番号をいくつか挙げる．Unix では，安全性のために 1023 以下のポート番号はユーザーが直接に束縛できないので，以下のポートで呼ばれるプログラムは管理者が決める．

ポート番号	サービス名	役割
21	ftp	データ転送の制御
22	ssh	セキュアシェルによる遠隔ログイン
23	telnet	遠隔ログイン
25	smtp	メールの送信先
53	domain	ホスト名変換サーバ
80	http	ウェブサーバ

1.2 TCP とソケット

通信プロトコルは大きく二つに分けられる．電話のように，通信が継続的にできるものと，手紙のように単発的に行われるもの．後者でも継続性のある動きを構築することができるが，多くの場合では前者のように，最初から継続性が与えられた方が便利である．Internet では前者の代表者は TCP と言い，双方向の継続した通信を提供している．しかも，データの全てが正しく配達されることを保証している．後者は UDP という，メッセージ単位で行われる通信方法を使う．UDP は正しく配達されることを保証しないが，仕事が単純になっている分，通信が早い．

TCP を使うプログラムを書くときに，OS が提供する通信チャンネルをプログラム上に表す必要がある．そこで使われる概念はソケットである．新しく作られたソケットは何もできないが，そのソケットを通信チャンネルに束縛すれば，そのソケットに書くときはチャンネルへの送信，読むときはチャンネルからの受信という意味になる．

ソケットに関するいくつかのシステムコールを見よう．

- *int* `socket`*(int domain, int type, int protocol);*
  `socket()` は新しいソケットを作る．ファイルと同様に，正の整数のハンドルが返される．引数は，接続性を表す `domain`，TCP か UDP の別を表す `type` があるが，インターネット用の TCP ソケットなら，`socket(PF_INET, SOCK_STREAM, 0)` で作る．

- *int* `bind`*(int s, struct sockaddr *addr, socklen_t addrlen);*

bind() はあるソケットをあるポート番号に束縛する．今後のこのポート番号への接続依頼はこのソケットに届けられる．通常の返り値は 0 である．第 1 引数はソケットハンドル，第 2 引数は束縛するアドレスとポート番号．接続の種類によって，具体的な sockaddr が変わるが，インターネットの場合では sockaddr_in になる．最後の引数は第 2 引数の長さを与える．

```
struct sockaddr_in {
        uint8_t         sin_len;
        sa_family_t     sin_family;
        in_port_t       sin_port;
        struct in_addr  sin_addr;
        char            sin_zero[8];
};
```

sin_family の値は socket() の domain と同じ値を使う．インターネットだと，AF_INET である．sin_port はポート番号だが，ネットワーク順を使うので要注意 (例では htons() 関数を使っている)．sin_addr は束縛されるアドレス．通常 INADDR_ANY で，このコンピュータが持っている全てのアドレスを使うが，それ以外の IP アドレスを与えると，そのアドレスに来た接続依頼だけが対象となる．例えば，ループバックアドレス 127.0.0.1 にすることで，他のコンピュータからの接続依頼を不可にできる．

- *int* listen*(int s, int backlog);*
  既に bind されたソケットに対して，TCP の接続を受け付けるようにする．第 2 引数は接続待ちの依頼を何個まで許すかを決める．

- *int* accept*(int s, struct sockaddr *addr, socklen_t *addrlen);*
  accept() は接続待ちの依頼を一個開く．返り値はその接続を扱うための新しいソケットハンドルである．addr が NULL 以外のポインターであれば，相手の IP アドレスとポート番号をそこに書き込む．そのときは書かれた情報の大きさが addrlen に書かれる．

- *int* connect*(int s, struct sockaddr *name, socklen_t namelen);*
  connect() はあるアドレスに接続を依頼する．accept されたら，渡さ

れたソケット s はこの接続へのハンドルになる．拒否されたら，エラーを返す．

- `read()`, `write()`, `close()`
  ソケットに対して，通常のファイル関連のシステムコールも使える．双方が `close()` を呼んだら，接続がなくなる．

以上のシステムコールを呼ぶときに，データ構造の中の値がネットワーク順になっていることに留意しなければならない．ネットワーク順の必要性は，マシンによって整数の表現方法が違うことから来ている．

バイト順	式	CPU 例
リトルエンディアン 下位バイトが先	$x = a_0 + 2^8 a_1 + 2^{16} a_2 + 2^{24} a_3$	Intel x86 など
ビッグエンディアン 上位バイトが先	$x = 2^{24} a_0 + 2^{16} a_1 + a^8 a_2 + a_3$	PowerPC など

ネットワーク順とはビッグエンディアンのことなので，PowerPC の場合では特に気にする必要はないが，Intel と逆の順番なので，整数を逐一ネットワーク順に変換する必要がある．整数の長さによって `htonl`(32 ビット) と `htons`(16 ビット) を使う．逆変換は `ntohl` と `ntohs`.

プログラム例

とても単純なプログラム例である．サーバはあるポート番号で待ち，接続が来たらそこから一行だけ読み，それを端末に出力する．クライアントは一行を端末から読み，サーバに送る．

実習　まずサーバを書いて，`gcc server.c -o server` でコンパイルする．(Solaris の場合，`-lsocket -lnsl` も付けなければならない．) 移すときは，細かい間違いに気を付けなければならない．

`server` ができたら，それを実行する．別の端末で

```
$ telnet localhost 6333
```

でテストができる．exit を送るとサーバが終了する．

クライアントもコンパイルして，そちらでも使って見る．

時間があれば，サーバに今までの全メッセージを記録するようにし，クライアントから log という文字列を送ったら，その全記録を送り返すようにする．

サーバ (server.c)

```
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define PORT 6333

int main(int argc, char **argv)
{
    struct sockaddr_in sin, from;
    int sd, asd, nbyte;

    /* set address */
    sin.sin_family = AF_INET;
    sin.sin_port = htons(PORT);
    sin.sin_addr.s_addr = INADDR_ANY;
    /* allocate socket */
    sd = socket(PF_INET, SOCK_STREAM, 0);
    if (sd < 0) {
        perror("socket error");
        exit(2);
    }
    /* bind socket */
    if (bind(sd, (struct sockaddr*)&sin, sizeof(sin)) < 0) {
        perror("bind error");
        exit(3);
    }
    /* set max connection number */
    listen(sd, 5);
    /* infinite loop waiting for connections */
```

```
    for (;;) {
        char line[128];
        /* accept connection */
        nbyte = sizeof(from);
        asd = accept(sd, (struct sockaddr*)&from, &nbyte);
        if (asd < 0) {
            perror("accept error");
            continue;
        }
        nbyte = read(asd, line, 128);
        line[nbyte] = 0;
        printf("%s:%d: %s\n", inet_ntoa(from.sin_addr),
               ntohs(from.sin_port), line);
        close(asd);
        if (!strcmp("exit", line)) exit(0);
    }
}
```

クライアント (client.c)

```
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define PORT 6333

int main(int argc, char **argv)
{
    struct sockaddr_in sin;
    int sd;
    char line[128];

    /* set address */
    sin.sin_family = AF_INET;
    sin.sin_port = htons(PORT);
    sin.sin_addr.s_addr = inet_addr("127.0.0.1");
    /* allocate socket */
    sd = socket(PF_INET, SOCK_STREAM, 0);
    if (sd < 0) {
        perror("socket error");
        exit(2);
    }
    /* connect to server */
    if (connect(sd, (struct sockaddr*)&sin, sizeof(sin)) < 0) {
        perror("connect error");
        exit(3);
    }
    printf("Send message: ");
    fgets(line, 128, stdin);
    write(sd, line, strlen(line) - 1);
    exit(0);
}
```