http://x68000.q-e-d.net/~68user/net/func.html
この項では、ネットワークプログラムで使用する perl の関数を解説します。関数には、perl の組み込み関数 (socket・bind など) と、Socket モジュールによって設定される関数 (inet_aton・sockaddr_in など) があります (use Socket と書いてある関数)。

C の関数を知りたいときはオンラインマニュアルを見て下さい。 perl の関数とは引数の数が違うこともありますが、考え方はほぼ同じです。

また、定数は Socket モジュールによって設定されていますが、実際は /usr/include/sys/socket.h で設定されている値がそのまま使われています。

おさらいしておくと、

クライアント

・getservbyname でサービス名からポート番号を取得
・inet_aton で接続先のホスト名を IP アドレスに変換
・sockaddr_in で IP アドレスとポート番号をひとまとめにした構造体を作成
・socket で、データ入出力の出入口となるソケットを生成
・connect で相手先に接続

サーバ

・socket で、データ入出力の出入口となるソケットを生成
・bind で、ソケットにポート番号を割り当て
・listen で、OS に接続を受け入れるよう命令
・accept で、接続してきたクライアントと通信を開始

という流れになります。

関数

■ホスト名と IP アドレスの変換

```
inet_aton
inet_ntoa
gethostbyaddr
gethostbyname
```

■サービス名とポート番号の変換

```
getservbyname
```

■ IP アドレスとポート番号

```
sockaddr_in
```

```
pack_sockaddr_in
unpack_sockaddr_in
```

■ソケット

```
socket
setsockopt
connect
getsockname
getpeername
```

■サーバプログラムのコネクション受け付け

```
bind
listen
accept
```

詳細

■ accept … クライアントとの接続を確立

```
SOCKADDR = accept(NEW_SOCKET, SOCKET)
```

クライアントとの接続を確立し、新しいソケット NEW_SOCKET を生成します。 SOCKET は、socket で作成し、bind でポートに結び付けたソケットです。ソケット NEW_SOCKET を通じて、クライアントとデータのやりとりをすることができます。 SOCKET は、その後もポート監視用ソケットとして使われます。
より詳しく言うと、listen により OS が陰で作成していた待ち行列から、1 つだけクライアントを取り出し、そのクライアントとの接続を確立するのが accept です。
accept は、戻り値としてクライアント側の情報を返します。 IP アドレスとポート番号を一体化した SOCKADDR 形式になっていますが、 unpack_sockaddr_in で分解することができます。

■ bind … ソケットにポート番号を割り付ける

```
bind(SOCKET, SOCK_ADDR)
```

ソケット SOCKET に対して、ポート番号 SOCK_ADDR を割り当てます。 SOCKET は socket を使って事前に用意しておきます。
例えばソケット S にポート 1234 番を割り当てたいなら

```
bind(S, pack_sockaddr_in(1234, INADDR_ANY))
```

とします。特定のポートを使用できるのは一つのプロセスだけなので、他のプロセスがたまたまポート 1234 を使っていると、bind に失敗します。
何番のポートでもよいなら、OS にポートを選ばせることもできます。

```
bind(S, pack_sockaddr_in(0, INADDR_ANY))
```

と、ポート番号に 0 番を指定すると、OS が勝手にポートを選んでくれます。 FTP クライアントのデータコネクション の例を参照して下さい。

なぜ bind に pack_sockaddr_in の戻り値 (IP アドレスとポート番号をひとまとめにしたもの) を渡すかというと、同じマシンに IP アドレスが複数割り当てられている場合に、どちらの IP アドレスのポートを使うのかを選択できるようになっているのです。しかし通常は IP アドレスとして INADDR_ANY を指定しておけば大丈夫です。

■ connect … サーバに接続

```
connect(SOCKET, SOCK_ADDR)
```

SOCK_ADDR で指定したホストとポートに接続します。データのやりとりはソケット SOCKET を通じて、

```
print SOCKET "....";
$in = <SOCKET>
```

などとします。
SOCKET は socket 関数で事前に作成しておきます。 SOCK_ADDR は接続先の IP アドレスとポート番号をひとまとめにした構造体で、 pack_sockaddr_in を使って事前に作成しておきます。

■ gethostbyaddr … IP アドレスをホスト名に変換

```
HOSTNAME = gethostbyaddr(IADDR, ADDR_TYPE)
```

IP アドレスの構造体と、アドレスファミリを渡すと、IP アドレスに対応するホスト名が返されます。 IADDR は、IP アドレスを表す構造体で、アドレスファミリには、AF_INET を指定します (AF_INET は Socket モジュールによりセットされる定数です)。

```
use Socket;
$ip_address = '133.8.13.11';
$iaddr = pack('C*', split(/\./, $ip));
$hostname = gethostbyaddr($iaddr, AF_INET);
```

■ gethostbyname … ホスト名・IP アドレスの情報を取得

```
gethostbyname(NAME)
```

引数には X68000.startshop.co.jp などというホスト名を渡します。

```
$iaddr = gethostbyname('X68000.startshop.co.jp');
$ip_address = join('.', unpack("C*", $iaddr));
```

とすることで $ip_address に 192.168.1.22 のような IP アドレス (表示可能な文字列) が代入されます。 Perl5 を使っていて、Socket モジュールが使えるなら、

```
use Socket;
$ip_address = inet_ntoa(inet_aton("X68000.startshop.co.jp"));
```

とした方がよいでしょう。

■ getpeername … 相手先のホスト名・IP アドレスの情報を取得

```
getpeername
```

■ getservbyname … サービス名からポート番号を取得

```
PORT = getservbyname(SERVICE, PROTO)
```

getservbyname は http・smtp・ftp などのサービス名を、対応するポート番号に変換します。

```
$port = getservbyname('http', 'tcp');   # $port には 80 が代入される
$port = getservbyname('smtp', 'tcp');   # $port には 25 が代入される
$port = getservbyname('ftp',  'tcp');   # $port には 21 が代入される
```

HTTP のポート番号は 80 であり、 SMTP は 25 であると決まっています。ですから HTTP クライアントを作るなら 直接ポート番号を 80 と決め打っても動作するのですが、 プログラム中に 80 という値を埋め込むと、ソースを読んでいる人にとっては、どうして突然 80 という数字が出てきたのか わかりません。 そこで `HTTP' を 80 に変換してくれる getservbyname を使って「HTTP というプロトコルを使う」 ということをわかりやすくしているのです。
第 2 引数の 'tcp' は、常にそう書くものだと思っておいてください。 UDP/IP を利用する DNS などは 'udp' を指定しますが、 HTTP・FTP・POP3・SMTP などは全て TCP/IP を使用するので、 'tcp' を指定しておけば問題ありません。
プロトコル名とポート番号の対応表は、UNIX なら /etc/services に、 Windows なら C:\windows\services に、Windows NT なら C:\WinNT\system32\drivers\etc\services に置いてあります。なお、getservbyname の serv というのは service の略です。

■ getsockname … ソケットの情報を取得

```
(SOCK_ADDR) = getsockname(SOCKET)
```

ソケットの IP アドレスとポート番号を調べます。相手先の IP アドレス・ポート番号を調べるのではありません。それは getpeername で調べることができます。
ここで得られる IP アドレスは、自分自身のホストに付けられた IP アドレスです。

```
$sockaddr = getsockname(SOCK);
($port, $iaddr) = unpack_sockaddr_in($sockaddr);
$hostname = gethostbyaddr($iaddr, AF_INET);
print "ホスト $hostname、ポート $port\n";
```

■ inet_aton … ホスト名を IP アドレスに変換 (Socket モジュール関数)

```
use Socket;
IADDR = inet_aton(HOSTNAME);
```

ホスト名を表す文字列を受け取り、IP アドレスに変換します。例えば

```
use Socket;
$iaddr = inet_aton("X68000.startshop.co.jp");
```

とすると、X68000.startshop.co.jp に対応する IP アドレスが $iaddr に代入されます。 $iaddr には '127.0.0.1' などという文字列が代入されるのではなく、生のデータが代入されます。127.0.0.1 なら `0x7f 0x00 0x00 0x01' という 4 バイトの構造体が代入されるわけです。そのため、$iaddr を print 文などで表示しようとしても、化けた表示になります。文字列として表示したい場合は inet_ntoa を使います。
inet_aton を実行すると、/etc/resolv.conf に書かれた DNS サーバに対して IP アドレスの問い合わせが行われます。これらの変換作業は、OS が全て面倒を見てくれます。
もし指定したホスト名に対応する IP アドレスが見付からない (正引きできない) と、エラーになります。 TCP/IP では IP アドレスがわからないと絶対に相手先と通信することはできないからです。
また、

```
use Socket;
$iaddr = inet_aton("X68000.startshop.co.jp");
```

は

```
$iaddr = gethostbyname("X68000.startshop.co.jp");
```

と等価です。perl4 しかなくて Socket モジュールを利用できないならこの書き方で同じ結果が得られます。しかし perl5 が使えるなら、素直に inet_aton を使いましょう。

■ inet_ntoa … IP アドレスを表示可能な形式に変換 (Socket モジュール関数)

```
use Socket;
IP_STRING = inet_ntoa(IP_ADDRESS);
```

inet_aton、unpack_sockaddr_in の項で説明したように、

```
$iaddr = inet_aton($hostname);
```

や

```
($port, $iaddr) = unpack_sockaddr_in($sockaddr);
```

とすると、$iaddr にはそのままでは表示できない形で IP アドレスが代入されます。 inet_ntoa は、その IP アドレスを表示可能な文字列に変換します。これはただの文字列なので、print 文などで表示することができます。
また、

```
$ip_address = inet_ntoa(inet_aton("X68000.startshop.co.jp"));
```

とすると、$ip_addr には X68000.startshop.co.jp を IP アドレスに変換した 133.8.3.1' などという文字列が代入されます。
なお、上記の書き方は

```
$iaddr = gethostbyname("X68000.startshop.co.jp");
```

```
$ip_address = join('.', unpack("C*", $iaddr));
```

と等価です。 perl4 しかなくて Socket モジュールを利用できないならこの書き方で同じ結果が得られます。しかし perl5 が使えるなら、素直に inet_ntoa を使いましょう。

■ listen … クライアントからの接続を受け入れるよう OS に指示を出す

```
listen(SOCKET, BACKLOG)
```

クライアントからの接続を受け入れるよう、OS に命令します。 listen が実行されると、OS は接続してきたクライアントとの接続を確立し、待ち行列に登録します。
この待ち行列への登録作業は、プロセスからは見ることができません。全て OS が陰で行います。この後、accept を実行することでクライアントと実際に接続することができます。
BACKLOG には、まだ accept されていないコネクション (OS が待たせているコネクション) の最大数を指定するものです。もし BACKLOG 以上のクライアントが同時に connect してきた場合は、サーバは接続を拒否するか、何もレスポンスを返しません。
BACKLOG には直接数字を指定してもいいのですが、実際に何個のクライアントを受け付けるかは OS により異なります。 BACKLOG の解釈は OS によって異なり、

- BSD 系 OS では、backlog に 1.5 を乗じた値が使われる
- Solaris 2.6 や HP-UX では、適当な係数を乗じた値が使われる
- Linux では backlog に指定した値がそのまま使われる
- Solaris2.5.1 では +1 した値が使われる

などと解釈されます。
(use Socket として) Socket モジュールを使っているなら、 BACKLOG として SOMAXCONN を指定することができます。これは OS が許す最大値を意味します。FreeBSD-2.2.7 では /usr/include/sys/socket.h で

```
#define   SOMAXCONN   128
```

と定義されているので、

```
listen(CLIENT_WAITING, SOMAXCONN)
```

と

```
listen(CLIENT_WAITING, 128)
```

は同じ意味です。
特に理由がなければ SOMAXCONN を指定しておくといいでしょう。

■ pack_sockaddr_in … IP アドレスとポート番号を、SOCKADDR 構造体に変換 (Socket モジュール関数)

```
use Socket;
SOCK_ADDR = pack_sockaddr_in(PORT, IP_ADDRESS);
```

ポート番号 PORT と、IP アドレスを表す構造体 IP_ADDRESS を受け取り、2 つをひとまとめにした構造体 SOCK_ADDR を返します。
IP_ADDRESS は inet_aton で作成します。 PORT は getservbyname を使ってサービス名から変換するか、あるいはそのまま数値を渡します。
ここで得られた SOCK_ADDR は connect で使用します。
pack_sockaddr_in の逆の操作を行うには unpack_sockaddr_in 関数を使います。

■ setsockopt … ソケットオプションを設定

```
setsockopt(SOCKET, LEVEL, OPTION_NAME, OPTVAL)
```

ソケット SOCKET に対して、ソケットオプションを設定します。 echo サーバでの使用例 を参照して下さい。

■ sockaddr_in … IP アドレスとポート番号をひとまとめにした構造体を作成 (Socket モジュール関数)

```
use Socket;
SOCK_ADDR = sockaddr_in(PORT, IP_ADDRESS)
(PORT, IP_ADDRESS) = sockaddr_in(SOCK_ADDR)
```

この関数は、2 通りの使い方があります。スカラーコンテキストの場合、つまり

```
$sock_addr = sockaddr_in($port, $iaddr)
```

のように1つの戻り値を受け取るような形で呼び出したときは、$port と $iaddr をひとまとめにした構造体を返します。一方、リストコンテキストの場合、つまり

```
($port, $iaddr) = sockaddr_in($sock_addr)
```

配列を受け取る形で記述したときは、先の例とは逆に $sock_addr を受け取り、$port と $iaddr に分解します。
sockaddr_in は、スカラーコンテキストなら pack_sockaddr_in を呼び出し、リストコンテキストなら unpack_sockaddr_in を呼び出すだけの wrapper 関数です。この関数は呼び出し方によって、全く逆の動作をするわけで、なるべく pack_sockaddr_in、unpack_sockaddr_in を使用した方がよいでしょう。その方がわかりやすいと思います。

→ 関数説明 : pack_sockaddr_in
→ 関数説明 : unpack_sockaddr_in

■ socket … ソケットを生成

```
socket(SOCKET, DOMAIN, TYPE, PROTOCOL);
```

ソケットを生成します。ソケットとは、データ入出力の出入口のようなもので、クライアントもサーバもソケットを通じてデータをやりとりします。

・ TCP/IP を使う場合は socket(SOCKET, PF_INET, SOCK_STREAM, 0)
・ UDP/IP を使う場合は socket(SOCKET, PF_INET, SOCK_DGRAM, 0)
・ UNIX ドメインソケットを使う場合は socket(SOCKET, PF_LOCAL, SOCK_DGRAM, 0)

とします。HTTP・FTP・POP3・SMTP などは全て TCP/IP を使用するので、最初の書式だけを覚えておけばいいでしょう。4 番目の引数は常に 0 にして下さい。
SOCKET の部分には 好きなファイルハンドル名を指定して下さい。例えば

```
socket(S, PF_INET, SOCK_STREAM, 0)
```

とすれば、ファイルハンドル S が生成されます。ソケットから読み込むには

```
$in = <S>
```

とし、ソケットに送信するには

```
print S "....";
```

とします。

■ unpack_sockaddr_in … SOCKADDR 構造体をポート番号と IP アドレスに変換 (Socket モジュール関数)

```
use Socket;
(PORT, IP_ADDRESS) = unpack_sockaddr_in(SOCK_ADDR);
```

SOCKADDR 構造体を、ポート番号と IP アドレスとに分解します。
用途としては､例えば accept で生成されたソケットの IP アドレスとポート番号を調べる場合に使います。まず、getpeername でソケットの情報 (SOCKADDR 形式) を得て、unpack_sockaddr_in で IP アドレスとポート番号を取得できます。
例 1: accept の戻り値を利用

```
$sockaddr = accept(CLIENT, CLIENT_WAITING);
($port, $iaddr) = unpack_sockaddr_in($sockaddr);
$hostname = gethostbyaddr($iaddr, AF_INET);
print "ホスト $hostname、ポート $port からの接続です。\n";
```

例 2: ソケットから getsockname で SOCKADDR を受け取ることもできる

```
accept(CLIENT, CLIENT_WAITING);
$sockaddr = getpeername(CLIENT);
($port, $iaddr) = unpack_sockaddr_in($sockaddr);
$hostname = gethostbyaddr($iaddr, AF_INET);
print "ホスト $hostname、ポート $port からの接続です。\n";
```

他にも、getsockname の戻り値に対して使用することがあります。