

# ネットワークプログラミング

## 第3回

### ソケットプログラミング(3)

#### ーWindowsでソケットプログラミングー

# Windowsでのプログラミング

- Win95より前はネットワーク関係のプログラム作成は難しかった。
- POSIX準拠になってからUNIXと同様に扱える部分が増えた。
  - ソケットプログラムもできます。

## WinSock (Windows Sockets)

- Windowsでソケットを使ってプログラムをつくるためのAPI (Application Program Interface)

# でも、

- 全く同じプログラムで動作するわけではないのが実状...
  - UNIX系の場合と、Windowsの場合では若干の違いがある。
  - その違いを押さえれば、それほど難しくはない

# WinSockの使い方

sys/types.h  
sys/socket.h  
arpa/inet.h  
unistd.h  
のかわりに

1. ヘッダファイル winsock2.h を読み込む

```
#include <winsock2.h>
```

2. WinSockを初期化する

```
WSADATA wsaData;
```

```
WSAStartup(MAKEWORD(2, 0), &wsaData);
```

WinSockの機能を使う場合は、  
(socket())を使わなくても)  
初期化が必要である。

MAKEWORD: 引数の2つの数値を  
2バイトデータにパックする

使用するWinSockのバージョンを指定する  
ここでは、2.0

# WinSockの使い方(つづき)

## 3. ソケットディスクリプタの型

UNIXでは `int` 型、WinSock では `SOCKET` 型  
(実体は符号なし `int` 型)

→ `socket` のエラー時の戻り値が違ふ

UNIXは負値、WinSockでは `INVALID_SOCKET`

## 4. `close()` のかわりに、`closesocket()` を用いる

`closesocket(s);`

## 5. 終了時に WinSock の終了処理をする

`WSACleanup();`

あとは、だいたい  
UNIXの場合と同じ。

# Windows版 プログラム例

- wserver.c
  - 12345番ポートで待ち受け、  
接続してきたクライアントに「HELLO」という文字列を送るプログラム
- wclient.c
  - ローカルホスト(127.0.0.1)の12345番ポートにアクセスし、サーバから送られた文字列を表示するプログラム

# Windows版プログラムの コンパイル/実行

- BCCを使う場合：  
    **bcc32c wserver.c**  
    **bcc32c wclient.c**
- コマンドプロンプトを2つ開き、  
    一方は、**wserver.exe** を実行  
    もう一方で、**wclient.exe** を実行する

Visual Studio コマンドプロンプトの場合：

```
cl wserver.c ws2_32.lib  
cl wclient.c ws2_32.lib
```

# もっと複雑なプログラム作成に向けて...

- サーバプログラム
  - 複数のクライアントを相手にすることになるため、WinSockでは非同期モード、UNIXではforkによる子プロセス生成やスレッド化で対応することになる。  
(これは少し難しそう。)
- クライアントプログラム
  - 基本的に、使いたいネットワークサービスのプロトコルに従って、必要なデータをsendし、サーバからデータをrecvすることになる。
  - 比較的大きなプログラムでも作りやすい(かもしれない)。
  - サーバからの応答待ち時間が長くなる場合、サーバプログラムと同様に非同期モードなどによって、プロセスがスタックしないようにする配慮が必要になる。

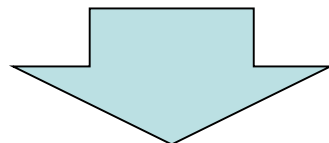


# ソースの統合

Windows用とLinux用  
別々にソースファイルを  
用意しなければならないのか??

# 条件付コンパイルの利用

- 異なる処理系で同じプログラムを動作させたい。  
(例えば、Linux と Windows)
  - 通常、プログラムに若干の修正が必要。
    - 別のソースファイルを用意しなければダメ??



条件付コンパイルを利用すれば、  
1つのソースプログラムで対応可能！

# プログラム中で、 WindowsかLinuxか見分ける

通常、条件付コンパイルは、コンパイル時に  
識別子を定義することで条件を変更する。

処理系を見分けてコンパイルする場合は、  
予め定義されている識別子を利用する。

Windows の場合、「\_WIN32」という  
識別子が定義されている。

# 処理系別のプログラムをまとめる

```
#ifdef _WIN32
#include <winsock2.h>
#else
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#endif
```

Windowsの場合に  
読み込ませる  
ヘッダファイル

Windows以外の場合  
に読み込ませる  
ヘッダファイル

Windowsの場合、socket を使う前に  
WSAStartupで初期化する必要がある。

```
#ifdef _WIN32
    WSAStartup(MAKEWORD(2, 0), &wsaData);
#endif

    s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
#ifdef _WIN32
    if (s == INVALID_SOCKET)
#else
    if (s < 0)
#endif
    {
        perror("socket");
        exit(EXIT_FAILURE);
    }
```

ソケットを確保できなかった  
ときのエラーの値がWindowsと  
Linuxで異なる

# マクロを使ったソース統合

- #ifdef ~ #endif を使う方法は、何度も同じような記述をすることになると煩雑。(socketを複数回使用する場合など。)
- マクロ定義の機能を併用すると、すっきりと記述できる場合が多い。

プログラムの冒頭や  
ヘッダファイル内に  
書くのが普通

```
#ifdef _WIN32
#define IS_INVALID_SOCKET(x) ((x) == INVALID_SOCKET)
#else
#define IS_INVALID_SOCKET(x) ((x) < 0)
#endif

s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
if (IS_INVALID_SOCKET(s)) {
    perror("socket");
    exit(EXIT_FAILURE);
}
```

# マクロ定義（おさらい？）

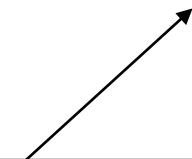
#define	マクロ名 (仮引数名の並び)	置換の並び
	プログラム中	コンパイル時

#define F(x) sin(x)

$b = F(A) \rightarrow b = \sin(A)$

#define X(x, y) ((y) / (x))

$c = X(1+a, b) \rightarrow c = ((b) / (1+a))$



式などの場合は、引数にどのような記述が入っても式の意味が変わらないように置換記述中は、仮引数名を( )で囲うとよい。

# マクロ使用時の注意

- マクロの名前に注意
  - 変数名などと重ならないようにする。
  - 大文字や下線\_を多用する。
  - プログラム中で、マクロだな(関数ではないな)と分かるような名前にしよう。



# ソースを統合したプログラム例

- wgetip.c
  - 既出の getip.c をWindows, Linux のどちらでも実行できるようにしてある。
  - 具体的には、Windowsの場合とそれ以外 (UNIX系) の場合で読み込むヘッダファイルを切り替えている。

コンパイルして、実行してみよう。

wgetip hostname

hostname: IPアドレスを調べたいホスト名

# 課題

- 前回課題で作成した、Linux用のサーバプログラム、クライアントプログラムを Windows でも動作するように修正せよ。
  - ただし、Linuxでコンパイルしても動作するように、条件付コンパイルの手法で対応すること。
  - Teamsで課題を割り当てるので、プログラムソースファイルをアップロードして、提出すること。