ユビキタスシステムアーキテクチャ
第4回

ネットワークプログラミングの基礎
Echoサーバの演習

本日の演習

- チーム編成とタームプロジェクト中間発表会
- ネットワークの基礎知識
- socket の概念
- echo サーバ，クライアント
- 課題2
  - 自分のechoサーバを作ろう！

チーム編成について

- 1チームmax4名まで
- チーム名を決める！
- チーム代表者を決める！
- チーム代表者がチーム名、すべてのメンバーの名前と学籍番号をSFC-SFSの課題ページから登録する

タームプロジェクトについて

- 5/28（木）の授業で中間発表を行います!
  - 11時10分～　@τ12
    - 1グループ5分程度でどのようなシステムを構築するか説明する
    - 5枚程度の発表資料の準備をする
    - 出席できない場合は、TA/SAに相談してください
    - 別途対応します
- 質問、相談があればTA/SAまで
  - wataru, drgnman@ht.sfc.keio.ac.jp
  - デルタ棟S213に来てもらってもOKです

分散プログラムにおける協調パターン

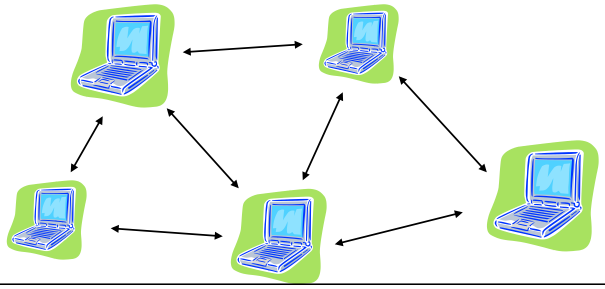
- クライアントサーバ型
- P2P 型
- アドホック型

クライアントサーバ型

- 現在最も多く利用されているネットワークの形態.
  - Web，メール，メッセンジャーなどなど
- サービスを提供するサーバーと，サービスを受けるクライアントから構成される.

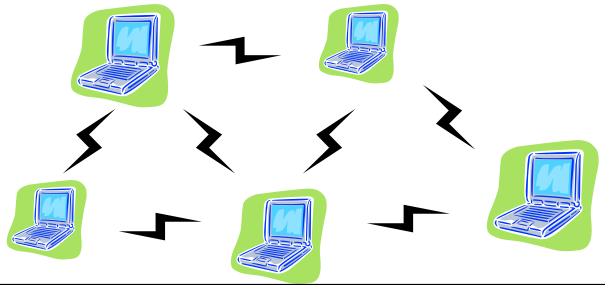
P2P型

- 定まったクライアントやサーバが存在せず，ネットワーク上の他のコンピュータに対して，サーバとしてもクライアントとしても動作するようなコンピュータの集合によって形成されるネットワークのこと.
  - ex) skype, winny, BitTorrent など



アドホック型

- 無線接続できる端末のみで構成されたネットワーク.
- 既存のネットワークインフラが必要ない！
- 狭義の P2P ネットワーク



IP アドレスとポート番号

- IPアドレス：インターネットを構築する時，個々のマシンに一意に割り当てられる識別子のこと.
  - 現在主流の IPv4 では，32 bit で表現される.
  - ex) www.sfc.keio.ac.jp -> 133.27.4.127
  - “nslookup” というコマンドで確認できます.
    - % nslookup www.sfc.keio.ac.jp
- ポート番号：一つのコンピュータ上で動作している複数のプログラムを識別するための番号.
  - ex) HTTP: 80 番， SSH: 23 番
  - “netstat -a” というコマンドで確認できます.

OSI 参照モデル

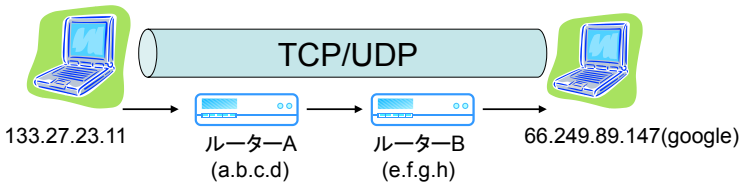
- コンピュータ同士が通信する際の，通信機能を階層構造に分割したモデルのこと.

- アプリケーション層
  - 具体的な通信サービスを提供（HTTP, SMTP）
- プレゼンテーション層
  - データの表現方法(SMTP, SNMP,FTP)
- セッション層
  - 通信プログラム間の通信の開始から終了までの手順 (NetBIOS, Named Pipe)
- トランスポート層
  - ネットワークの端から端までの通信管理(TCP,UDP,NetBEUI)
- ネットワーク層
  - ネットワークにおいて通信経路の選択. (IP, IPv6,ARP,ICMP)
- データリンク層
  - 直接的に接続されている通信機器間の信号の受け渡し (Ethernet, token ring, PPP)
- 物理層
  - 物理的な接続のこと. (RS-232, UTP,FTTH, 802.11abgn)

アプリケーション層
プレゼンテーション層
セッション層
トランスポート層
ネットワーク層
データリンク層
物理層

TCPとUDP

- ネットワーク通信を行う際，エンドホスト間でよく使われる通信プロトコル．



- TCP(Transmission Control Protocol): 信頼性のあるデータ転送を作成したいときに利用する．
  - パケットシーケンスチェック，パケット再送機能など．
  - オーバヘッドが多いため比較的低速．
- UDP(User Datagram Protocol): 信頼性が必要ない場合に利用される．
  - ビデオストリーミング，VoIP など
  - 送信するのみなので，比較的高速．

本日の演習では

- クライアントサーバ型を実現するネットワークプログラミングの基礎
- TCP/IPによるechoサーバ

本日の概要

- ネットワークの基礎知識
- socket の概念
- echo サーバ，クライアント
- 今日の宿題

socket って？

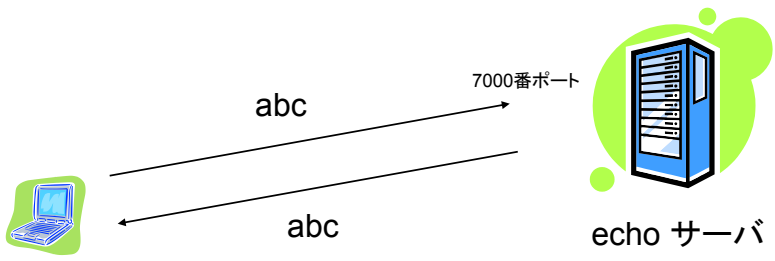
- プログラムが別のプログラムと通信を行うためのインターフェース
  - コンピュータの識別番号(IPアドレス)と，プログラムの識別番号(port番号)を指定
  - 別のプログラムは同じPCで動いていても良いし，別のPCで動いていても良い
- ファイルと同じように扱える
  - UNIXでは読み書きできるものをファイルとして扱う
    - キーボード，画面，プリンタ・・・
    - UNIX端末の/dev以下
  - ただし，ネットワークは開始の時だけ少し複雑

本日の概要

- ネットワークの基礎知識
- socket の概念
- echo サーバ，クライアント
- 今日の宿題

echo サーバを作成しよう

- echo サーバって？
  - ただ文字列を返すだけのもの

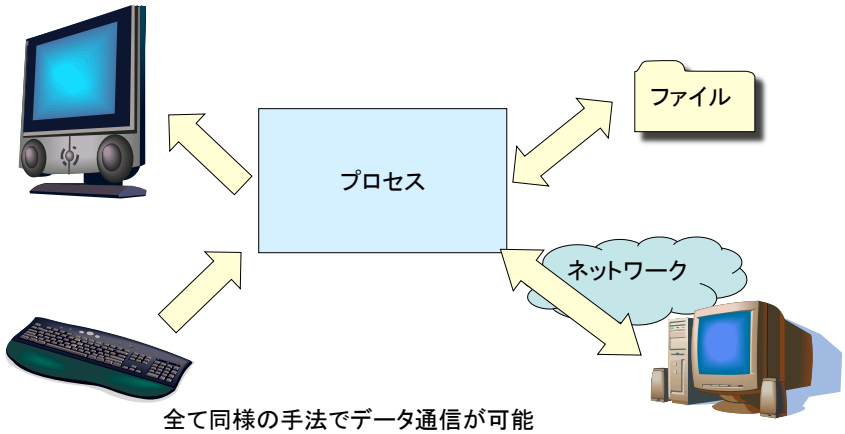


演習の準備

- SFC-SFS内の，echo_server.c を編集できる状態にしてください．
- “HERE” と書いてある場所があるので，そこを埋めていくことで，echo サーバを作成します．

ネットワークの抽象化

- ネットワークプログラミングにおいては、データ通信の端点をソケットとして抽象化している
  - ソケットはファイルと同じように扱える



全て同様の手法でデータ通信が可能

通信（サーバの場合）

- 手順
  - ソケットを作る（socket）
  - ソケットを初期化（bind）
  - 接続待ち（listen）
  - 接続確立（accept）
  - 通信する（read、write）
  - ソケット切断（close）

ソケットを作る

- socket関数を使う

  #include <sys/types.h>

  #include <sys/socket.h>

  int socket(int domain, int type, int protocol);
  - 成功するとファイルディスクリプタが返る
  - このファイルディスクリプタに対して，読み書き(read/write)を行うことで，相手と通信する.
    - ただし、読み書きできるようになるには通信先との接続が完了してから

- 使い方

  sockfd = socket(PF_INET, SOCK_STREAM, 0);
  - SOCK_STREAM: TCP
  - SOCK_DGRAM: UDP

ソケットを初期化

- bind関数を用いる
  int bind(int sockfd, const struct sockaddr *my_addr, socklen_t addrlen);
  - ソケットにポート番号等を結びつける
- sockaddr_in 構造体
  - アドレス体系，ポート番号，受け付けるIPアドレスなどを記述する.
- 使い方

```
struct sockaddr_in server;
memset((void *)&server, 0, sizeof(server));
server.sin_family = PF_INET;
server.sin_port = htons(7000);
server.sin_addr.s_addr = htonl(INADDR_ANY);
bind(sockfd, (struct sockaddr *)&server, sizeof(struct sockaddr_in));
```

接続待ち

- listen関数を用いる

int listen(int sockfd, int backlog);

  – 接続を待つ

- 使い方

```
listen(sockfd, 5);
```

接続確立

- accept関数を使う

int accept(int sockfd, struct sockaddr *addr,
socklen_t *addrlen);
  – 接続を受ける

- 使い方

len = sizeof(client);
acceptfd = accept(sockfd, (struct sockaddr*)&client,
&len);

通信

- 通信が確立したら、以降はファイルと同じように扱える
  – read、writeを使ってデータを読み書き
- read 関数

int read(int sockfd, void *buf, int size);
  – 使い方
  read(acceptfd, buf, sizeof(buf));
- write 関数

int write(int sockfd, void *buf, int size);
  – 使い方
  write(acceptfd, buf, strlen(buf));
- 通信が終了したらcloseで接続を切断しよう

close(acceptfd);
close(sockfd);

echo サーバを立ち上げてみよう

- % gcc echo_server.c
- % ./a.out

- クライアントは，telnet で代用できます．
  % telnet localhost <port 番号>
  ex) % telnet localhost 7000
  (自分のマシン上でサーバを立ち上げていて，ポート番号が 7000 番の場合)

補足: ファイル操作

- file の扱い方
- 簡単な web サーバを作ってみよう(3, 4年生向け)

UNIXにおけるファイル

- ファイルって何だろう？
  - コンピュータ上でのデータ（ビットの集合）に名前を付けたもの
  - 狭義では、ファイルシステムにより管理されたハードディスク上の一塊のデータ
- ファイルに対して行える事
  - 読み取り（read）
  - 書き込み（write）
  - etc
- UNIX系OSでは、コンピュータ上の様々なモノとのデータのやり取りをファイル操作として抽象化している
  - つまり、色々なデータ通信をreadとwriteで行う事ができる！

ファイル操作

- 手順
  - ファイルを開く（open）
  - ファイルを操作する(read、write）
  - ファイルを閉じる（close）

ファイルを開いてみよう

- ファイルを開くにはopenシステムコールを使う

int open(const char* pathname, int flags);

  - 成功すると、pathnameで指定されたファイルのファイルディスクリプタ（ファイルを識別する数字）を返す
  - 失敗（ファイルが存在しない等）すると-1が返る
  - 返ってきたファイルディスクリプタに対し、readやwrite等のファイル操作が行える

- 使い方

int fd = open(“hoge.txt”, O_RDONLY);

fdという変数にhoge.txtというファイルのファイルディスクリプタが入る

ファイルの中身を読み込んでみよう

- ファイルを読み込むにはreadシステムコールを使う

ssize_t read(int fd, void* buf, size_t count);

  - 成功すると、fdで指定されたファイルのデータをcountバイトbufに読み込み、実際に読み込んだバイト数を返す
  - 失敗すると-1を返す
  - 常にcountバイト読み込めるとは限らない

- 使い方

char buf[1024];

read(fd, buf, sizeof(buf));

bufに1024バイト分ファイルのデータを読み込む

ファイルにデータを書き込んでみよう

- ファイルに書き込むにはwriteシステムコールを使う

ssize_t write(int fd, const void *buf, size_t count);

  - 成功すると、fdで指定されたファイルに対し、countバイトbufを書き込み、実際に書き込んだバイト数を返す
  - 失敗すると-1を返す
  - 常にcountバイト書き込めるとは限らない

ファイルを閉じる

- 使い終わったファイルは必ず閉じよう
- ファイルを閉じるにはcloseシステムコールを使います

int close(int fd);

  - 成功した場合は0を返す
  - 失敗したら-1を返す

ファイルの内容を表示するプログラム

```
#include <stdio.h>
#include <unistd.h>

void main() {
    int fd; //ファイルディスクリプタ
    char buf[1024]; //ファイルのデータ

    //ファイルを読み込む
    fd = open(“hoge.txt”, O_RDONLY);
    read(fd, buf, 1024);
    close(fd);

    //ファイルを表示
    write(1, buf, strlen(buf)); //ファイルディスクリプタ1は標準出力に割
      り当てられている
}
```

C言語の補足資料

概要

- C言語の基礎
- ネットワークの基礎知識
- socket の概念
- echo サーバ, クライアント
- 課題 2 echo server
- 中間発表

C言語のmini復習

- データ型
- 関数
- if else 文
- for 文
- while 文
- 配列
- 構造体
- ポインタ

いろいろなデータを扱う（構造体）

- 複数のデータ型を組み合わせて、新しく作るデータ型

```
struct grade{
   int id;
   char name[20];
   double avg;
};
```

```
struct タグ名 {
    データ型  メンバ名;
};
```

構造体

- struct.c を実行してみる
- 構造体の宣言

struct タグ名 変数名の並び;

```
struct grade student1;
struct grade student2[20];
```

構造体 cont.

- 構造体の初期化
  - 構造体変数の初期化

    { } の間に、メンバ名をカンマで区切る

```
struct grade student1 = {4, "TANAKA", 80.5};
```

  - 構造体変数の初期化

    { } の間に、メンバ名をカンマで区切る

```
struct grade student2[20] = {{1, "SUZUKI", 68.1 },
                             {2, "SAITO", 59.2 },
                            {3, "NAKATA", 48.4 }, };
```

構造体 cont.

- **構造体の参照**

  構造体変数名. メンバ名

```
printf("%d %s %5.1f\n\n",
student1.id, student1.name,student1.avg);

printf("%d %s %5.1f\n",
student2[i].id, student2[i].name, student2[i].avg);

student1.id = 5;
sutdent1.name = “HOGE”;
```

のように使用できる

構造体演習問題

- 新たにstruct2.c というファイルを作成し、以下の処理を行なってください。

以下の表を構造体で表し、平均点を求めてください

番号	国語	数学	化学	英語	平均点
1111	69	50	70	96	71.25
2222	98	39	60	60	64.25
3333	89	78	88	90	86.25
4444	40	50	98	60	62.00
5555	63	60	89	95	76.75

構造体データ

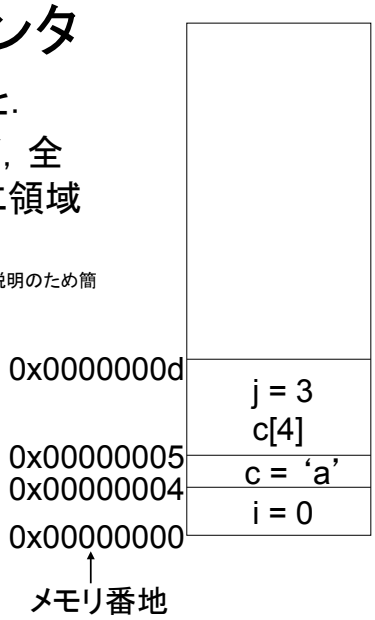
```
{{ 1111, 69, 50, 70, 96, 0.0 },
 { 2222, 98, 39, 60, 60, 0.0 },
 { 3333, 89, 78, 88, 90, 0.0 },
 { 4444, 40, 50, 98, 60, 0.0 },
 { 5555, 63, 60, 89, 95, 0.0 },};
```

ポインタ

- 「データのメモリ番地」のこと.
- int, char, short, struct など，全てのデータは，「メモリ」上に領域が確保されます.
  - 注：厳密にはこのメモリ番地は間違っています．説明のため簡単に書いています.

例）

```
int i = 0;
char c = ‘a’;
struct hoge {
   int j = 3;
   char c[4];
} hoge1;
```



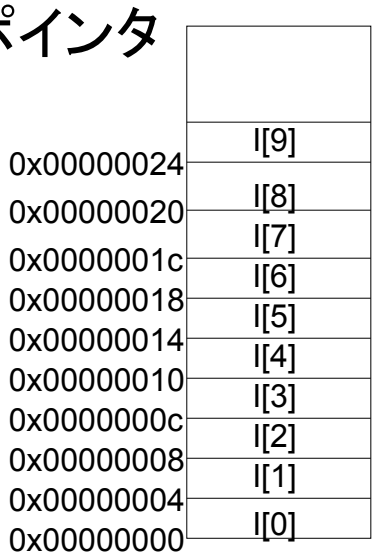
* と &

- メモリ番地を格納しておく変数のことを，「ポインタ変数」と言います.
  int *p;
- 「*」を変数の前につけることで，ポインタ変数になります.
  - int のデータが格納されたメモリ番地なのか，char 用なのかを判別するため，int *p のように，記述する.
- 変数自体のポインタ（メモリ番地）を取得するときには，変数に & をつける.
  - int i;
  - &i; ←ポインタを表す
- ポインタ変数に * をつけると，メモリ番地の中身（実体）を示します.
  - int I = 0;
  - int *p = &I;
  - *p は？　→ 0 が返ってきます.

配列とポインタ

- 配列は，メモリ番地を順番に確保している．

ex)
int i[10];
&i[0] → 0x00000000
int *p;
p = &I[0];
p++; → 0x00000004;
p++; → 0x00000008;



プログラムを眺めてみよう

- http://www.ht.sfc.keio.ac.jp/usa2007s/ から，usa05.zip をダウンロードして，中に入っている，pointer.c をコンパイルしてください．

Cygwin のターミナルでのコマンド

```
% unzip usa05.zip
% cd usa05
% gcc pointer.c
% ./a.exe
```

pointer.c から抜粋

```
int i;
int *int_p; /* int 型のポインタ変数宣言*/

i = 7;
int_p = &i;
printf("i is %d, i's pointer is %p¥n", i, int_p);
printf("*int_p is %d¥n", *int_p);
```

本日の概要

- C言語基礎の積み残し
  - 構造体
  - ポインタ
- ネットワークの基礎知識
- socket の概念
- file の扱い方
- echo サーバ，クライアント
- 今日の宿題

課題2

- 1，2年生向け（ネットワークプログラミングに不慣れな人向け）
  - echo サーバを改良して，打った文字が逆になって返ってくる，reverse サーバを作成しよう．
  - ex) “abc” と送ると，”cba” と返ってくる．
- 3，4年生向け（ネットワークプログラミング経験のある人向け）
  - echo サーバを改良して，簡単な web サーバを作成してみよう．
  - ヒント：ファイル操作ができる必要があります．
- 期限
  - 5/21, 23:59 まで．
  - SFC-SFSからソースプログラムと出力結果を提出してください．

補足

- file の扱い方
- 簡単な web サーバを作ってみよう(3, 4年生向け)

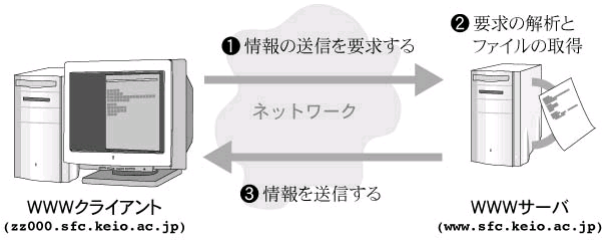
作るもの

- Webサーバ
  - Webブラウザからのリクエストを受け付け、指定されたデータを渡すソフトウェア
- 主なWebサーバ
  - Apache（UNIX系）
  - IIS（Windows系）

Web上でのデータ転送

- HTTPというプロトコルを用いてデータを転送する
  - このプロトコルに準拠していれば、自分でサーバやブラウザを作成する事も可能
- HTTPは要求-応答型プロトコル
  - ブラウザはサーバに対してリクエストを出す
  - サーバはブラウザからのリクエストに応じてデータを転送する

Web上でのデータ転送のイメージ



HTTP（Hypertext Transfer Protocol）

- Webサーバとブラウザ間のデータのやり取りを規定した通信プロトコル
  - RFC 2616で定義されている
  - 主にHTMLの転送を目的とする
    - バイナリデータの転送も可能
- HTML（HyperText Markup Language）
  - Web上のドキュメントを表現するための言語

HTTPリクエスト（バージョン1.1の場合）

GET / HTTP/1.1
Host: localhost:10080
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; ja; rv: 1.8.1.3) Gecko/20070
309 Firefox/2.0.0.3
Accept: text/xml,application/xml,application/xhtml+xml,text/ html;q=0.9,text/plai
n;q=0.8,image/png,*/*;q=0.5
Accept-Language: ja,en-us;q=0.7,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: Shift_JIS,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive

HTTPレスポンス（バージョン1．1の場合）

```
HTTP/1.1 200 OK
Date: Tue, 11 May 1999 10:27:51 GMT
Server: Apache/1.3.0 (Unix)
Vary: accept-language
Last-Modified: Thu, 08 Apr 1999 04:17:21 GMT
ETag: "8a0c3-cd6-370c2dd1“
Accept-Ranges: bytes
Content-Length: 3286
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html
Content-Language: en
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//
    EN">
<HTML>
<HEAD>
この後データが続く…
```

HTTPリクエスト（バージョン0.9の場合）

GET /

- 1行だけで済む
  - レスポンスもデータをそのまま返すだけ
- 今回は0.9で実装します

GETリクエスト

- GET <ファイル名>
  - 例：GET /
    - “GET /” は大抵”GET /index.html”と解釈される
  - 指定されたファイルのデータをクライアントに送り返す

作り方

- 授業中に完成させた，echo_server.c を改造してみましょう.
  - GET / という文字列が来たら，index.html を読み込んで，その内容を acceptfd に書く.
    - index.html は，自分で作成してください.
  - もうちょっと作ってみたいなら
    - GET /hoge.html という文字列の場合は hoge.html を返す，など，ファイル名の部分をパースしてみましょう.
    - 存在しないファイル名が要求されたら，Not Found と返すようにしてみましょう.

Webサーバの動作手順

1. Webブラウザからの接続要求を待つ
2. ブラウザと接続したら、リクエストを受け取る
3. リクエストを解析する
4. 要求されたファイルを開く
5. ファイルのデータをメモリ上に読み込む
6. データをブラウザに送信する
7. ブラウザとの接続を切断する
8. 1に戻る