

ネットワークプログラミング
2012/01/11

コンピュータネットワーク

+

2

前回の復習

- サーバの仕組み
- P2P
- パケット解析
- ネットワークプログラミング

本日の流れ

- UDPプログラミング
- ブロッキングとノンブロッキング
- TCPプログラミング

次週: 1月18日 休講

2012/01/11

+

UDPプログラミング

2012/01/11

3

+

4

IPv4 UDP 送信プログラム

```
int main()
{
    int sock;
    struct sockaddr_in addr;

    sock = socket(PF_INET, SOCK_DGRAM, 0);

    addr.sin_family = AF_INET;
    addr.sin_port = htons(12345);
    addr.sin_addr.s_addr = inet_addr("130.69.251.130");

    sendto(sock, "HELLO", 5, 0, (struct sockaddr *)&addr, sizeof(addr));

    close(sock);
}
```

2012/01/11

+ IPv4 UDP 受信プログラム

```
int main()
{
    int sock;
    struct sockaddr_in addr;
    char buf[2048];

    memset(buf, 0, sizeof(buf));

    sock = socket(PF_INET, SOCK_DGRAM, 0);

    addr.sin_family = AF_INET;
    addr.sin_port = htons(12345);
    addr.sin_addr.s_addr = INADDR_ANY;

    bind(sock, (struct sockaddr *)&addr, sizeof(addr));
    recv(sock, buf, sizeof(buf), 0);

    printf("%s\n", buf);
    close(sock);
}
```

2012/01/11

+ sockaddr_in / sockaddr_in6 構造体

```
struct sockaddr_in
{
    __SOCKADDR_COMMON (sin_);
    in_port_t sin_port; /* Port number. */
    struct in_addr sin_addr; /* Internet address. */
    /* Pad to size of `struct sockaddr'. */
    unsigned char sin_zero[sizeof (struct sockaddr) -
        __SOCKADDR_COMMON_SIZE -
        sizeof (in_port_t) -
        sizeof (struct in_addr)];
};

#define __SOCKADDR_COMMON(sa_prefix) \
sa_family_t sa_prefix##family

struct sockaddr_in6
{
    __SOCKADDR_COMMON (sin6_);
    in_port_t sin6_port; /* Transport layer port # */
    uint32_t sin6_flowinfo; /* IPv6 flow information */
    struct in6_addr sin6_addr; /* IPv6 address */
    uint32_t sin6_scope_id; /* IPv6 scope-id */
};
```

2012/01/11

+ 使われた主な API (関数)

- socket
 - ネットワーク通信の出入り口を作成
 - ファイルディスクリプタ
- sockaddr_in 構造体
 - 通信相手や自分の情報を入れる
- sendto
 - データを宛先に対して投げつける
- bind
 - データを待ち受けるための準備
- recv
 - データを待ち受ける

2012/01/11

+ socket() 関数

- int socket(int domain, int type, int protocol)
- ネットワーク通信のための出入口を作成
 - domain にはプロトコルファミリを指定
 - AF_INET IPv4 プロトコル
 - AF_INET6 IPv6 プロトコル
 - AF_LOCAL UNIX Domain Socket
 - Type にはソケットのタイプ (以下のどれか)
 - SOCK_STREAM ストリームソケット (TCP)
 - SOCK_DGRAM データグラムソケット (UDP)
 - SOCK_RAW rawソケット
 - Proto には raw ソケット以外、通常 0

2012/01/11

+ socket() 関数

- 返り値
 - 成功: ソケットディスクリプタが返る
 - 失敗: -1が返る
 - ソケットディスクリプタはファイルディスクリプタの友達
- 実際のコードでは...
 - `fd = socket(AF_INET, SOCK_STREAM, 0)`

2012/01/11

+ sendto

- `sendto(int s, const void *buf, size_t len, int flags, const struct sockaddr *to, socklen_t tolen);`
- メッセージを送信する
- 主に UDP の投げっぱなしアプリケーションで利用される

2012/01/11

+ bind / recv

- `bind`: データを待ち受けるための準備
 - 手元のどのインタフェースの
 - どのポート番号で

待ち受けるかを指定
- `recv`: データを受信
 - 受信したデータをメモリバッファに格納
 - ブロッキング受信

2012/01/11

+ bind 関数

- `int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);`
- 通信元の情報を与える
- 引数
 - `sockfd`: ファイルディスクリプタ
 - `addr`: 通信元の情報が入った `sockaddr_in / sockaddr_in6`
 - `addrlen`: `sockaddr` の長さ
- 返り値
 - 成功 0
 - 失敗 -1

2012/01/11

+ Ruby で書くと。。。 13

- udpseend のサンプルプログラムを Ruby で書くと

```
require 'socket'
s = UDPSocket.open()
sockaddr = Socket.pack_sockaddr_in(12345, "130.69.251.116")
s.send("HELLO", 0, sockaddr)
s.close
```

- ただし。。。どうやって動作しているか、まったく理解が深まりません。。。
 - 仕組みを理解した後に、使いましょう

2012/01/11

+ コマンドライン引数 : argc,argv 14

- コマンドの引数を用いるには?

- udpseend -d 130.69.251.116
 - -d というオプションに IP address が指定されている
- udpseend file1 file2 file3
 - 3つのファイルが指定されている

2012/01/11

+ コマンドライン引数の利用法 15

- main(int argc, char **argv)
- main(int argc, char *argv[])

- argc には引数の数
- argv[0] にはコマンド名
- argv[1] には1番目の引数
- argv[2] には2番目の引数

2012/01/11

+ コマンドライン引数 (サンプル) 16

```
#include<stdio.h>

int main(int argc, char *argv[]){
    int i;
    for(i = 0; i < argc; i++){
        printf("arg[%d]: %s\n", i, argv[i]);
    }
}
```

```
sekiya@LECTURE-CLIENT:~/EXAMPLE$ gcc -o arg_example arg_example.c
sekiya@LECTURE-CLIENT:~/EXAMPLE$ ./arg_example ABC DEF 123
arg[0]: ./arg_example
arg[1]: ABC
arg[2]: DEF
arg[3]: 123
```

2012/01/11

+ エラー処理

■ perror() 関数

- システムコールやライブラリ関数の呼び出しにおいて、最後に発生したエラーに関する説明メッセージを生成し、標準エラー出力に出力する

■ 使用例

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <errno.h>
int main() {
    int sock;
    sock = socket(1000, 1000, 1000);
    if (sock < 0) {
        perror("socket");
        return 0;
    }
    return 1;
}

sekiya@LECTURE-CLIENT:~/EXAMPLE$ gcc -o sock_error
sock_error.c
sekiya@LECTURE-CLIENT:~/EXAMPLE$ ./sock_error
socket: Invalid argument
```

2012/01/11

+ ブロッキングと ノンブロッキング

2012/01/11

18

+ Blocking / Non-Blocking

■ Blocking

- データを受信する際、データが到着するまで処理を中断して待機する

■ Non Blocking

- 処理を中断することなくデータを待機し、受信したデータを後ほど取得する方式
- ポーリング方式とも呼ばれる
- select という API を用いて実現

2012/01/11

+ TCPクライアントサンプル (抜粋)

```
deststr = "133.11.205.161";
destport = 13;

sock = socket(AF_INET, SOCK_STREAM, 0);
server.sin_family = AF_INET;
server.sin_port = htons(destport);
s = inet_pton(AF_INET, deststr, &server.sin_addr);

connect(sock, (struct sockaddr *)&server,
        sizeof(server));

memset(buf, 0, sizeof(buf));
n = read(sock, buf, sizeof(buf));
printf("%s\n", buf);
```

2012/01/11

+ サンプルプログラム実行結果

21

- daytime プロトコル
 - 現在の時刻をテキスト形式で返すサービス
 - TCP port 13番

■ 実行例

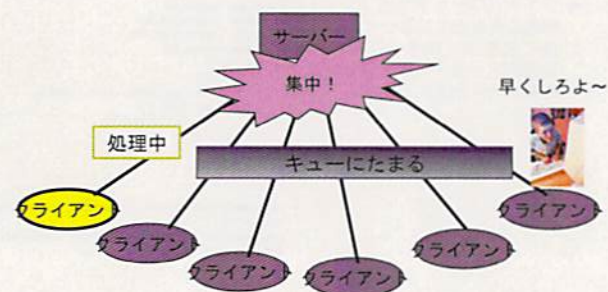
```
sekiya@LECTURE-CLIENT:~/EXAMPLE$ gcc -o tcp_client tcp_client.c
sekiya@LECTURE-CLIENT:~/EXAMPLE$ ./tcp_client
Thu Jan 18 08:47:47 2011
```

2012/01/11

+ ブロッキングするサーバだと。。。

22

先着順の処理

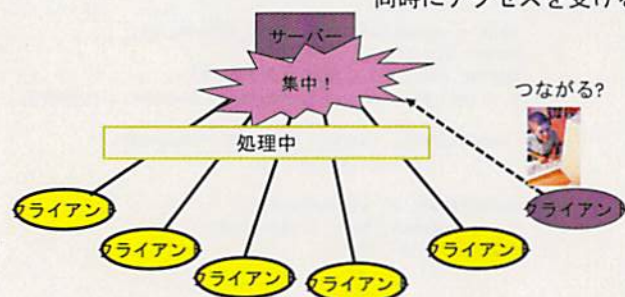


2012/01/11

+ ノンブロッキングならば

23

複数のクライアントから同時にアクセスを受ける



2012/01/11

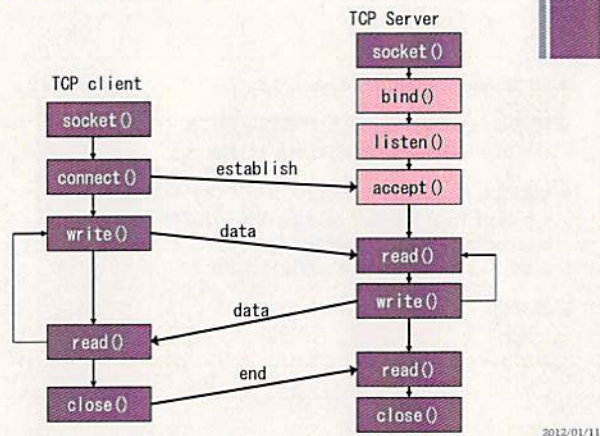
+

TCPプログラミング

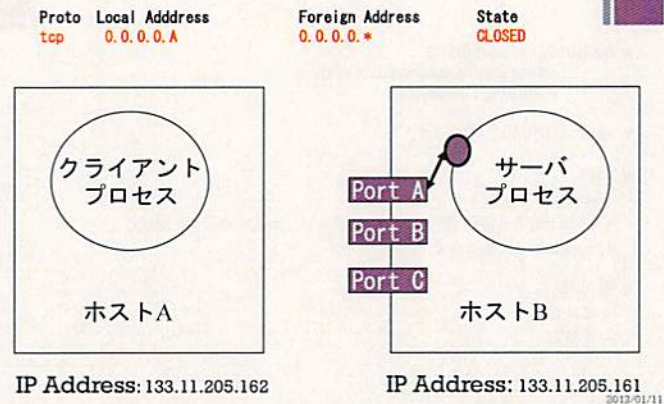
2012/01/11

24

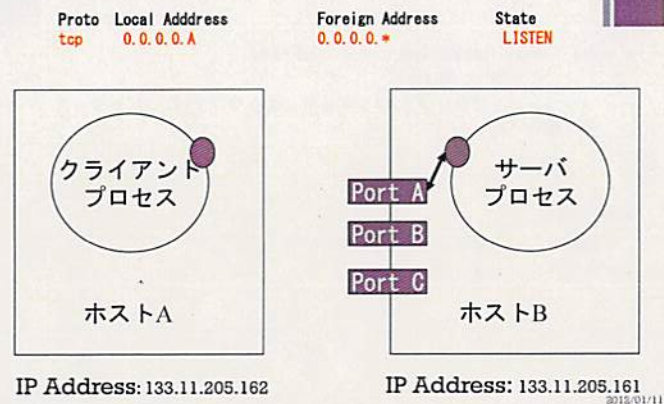
+ TCP 通信の流れ



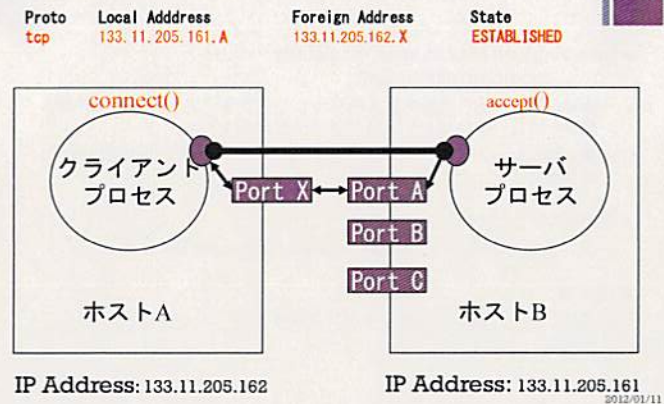
+ bind()



+ Listen()



+ connect() -> accept()



+ bind 関数

29

- `int bind(int sockfd,
const struct sockaddr *addr,
socklen_t addrlen);`
- 通信元の情報を与える
- 引数
 - `sockfd`: ソケットディスクリプタ
 - `addr`: 通信元の情報が入った `sockaddr_in / sockaddr_in6`
 - `addrlen`: `sockaddr` の長さ
- 返り値
 - 成功 0
 - 失敗 -1

2012/01/11

+ listen() 関数

30

- `int listen(int sockfd, int backlog)`
- 用意した socket を待ち受け準備状態にする
 - ソケットの状態を CLOSED から LISTEN へ
- `backlog` は connect に来たクライアントの待機数
 - `accept()` されるまで backlog キューに保持
 - `accept()` した順に、順次処理
 - キューがあふれると、その要求は無視
- 返り値
 - 成功 0
 - エラー -1

2012/01/11

+ accept() 関数

31

- `int accept(int sockfd, struct sockaddr *addr,
socklen_t *addrlen)`
- listen キューから接続要求を取り出して、クライアントと通信を開始 => 新しいソケットディスクリプタを作成
- 新しいソケットディスクリプタを利用してサーバ側での通信処理が始まる
- 実際のコードでは...
 - `cli = accept(sockfd, (struct sockaddr *)&cliaddr, &clilent)`
- 返り値
 - 成功 新しいソケットディスクリプタ
 - エラー -1

2012/01/11

+ write() 関数

32

- `ssize_t write(int sockfd, const void *buf,
size_t nbyte)`
- ソケットディスクリプタ(ファイルディスクリプタ)に対してデータ列を書き込む
- 返り値
 - 成功 書き込んだデータサイズ
 - 失敗 -1

2012/01/11

+ サーバサンプルプログラム

(抜粋 - エラー処理なし)

```
sock = socket(AF_INET, SOCK_STREAM, 0);
addr.sin_family = AF_INET;
addr.sin_port = htons(12345);
addr.sin_addr.s_addr = INADDR_ANY;

bind(sock, (struct sockaddr *)&addr, sizeof(addr));
listen(sock, 5);

while (1) {
    len = sizeof(client);
    cli = accept(sock, (struct sockaddr *)&client, &len);
    write(cli, "HELLO\n", 7);
    close(cli);
}
```

2012/01/11

+ 接続してきたクライアントの確認

- `cli = accept(sock, (struct sockaddr *)&client, &len);`
- ここで新しいソケットディスクリプタ (cli) が返される
- 同時に、sockaddr_in client に情報が入る
- sockaddr_in client 構造体に、接続してきたクライアント(相手)に関する情報が格納される
 - サーバは、つないできたクライアントに関する情報を得ることができる

2012/01/11

+ 実行例

■ クライアント

```
sekiya% telnet -4 lecture-server.nc.u-tokyo.ac.jp 12345
Trying 133.11.205.161...
Connected to lecture-server.nc.u-tokyo.ac.jp.
Escape character is '^]'.
HELLO
Connection closed by foreign host.
```

■ サーバ

```
sekiya@LECTURE-CLIENT:~/EXAMPLE$ ./tcp_server
accepted connection from 130.69.251.130, port=49381
```

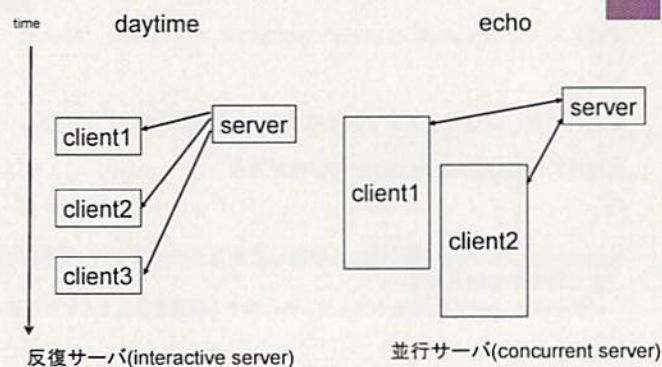
2012/01/11

+ ブロッキングと ノンブロッキング

2012/01/11

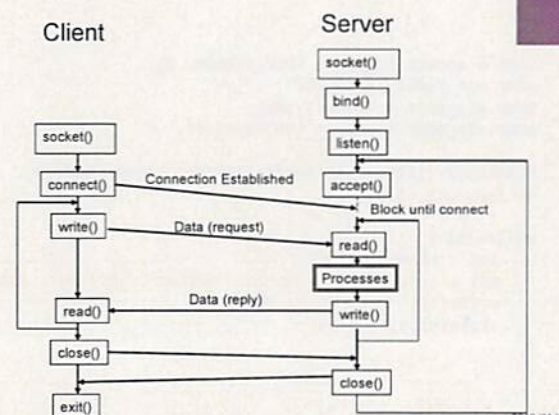
36

+ daytime server / echo server



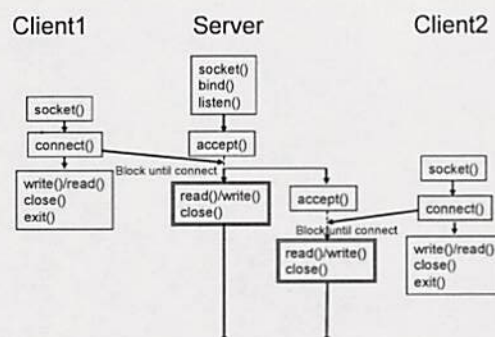
2012/01/11

+ ブロッキングの場合



2012/01/11

+ ノンブロッキングの場合



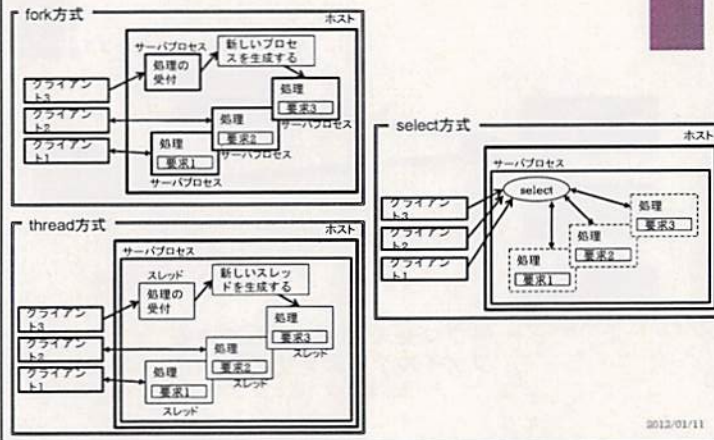
2012/01/11

+ サーバプログラミング

- 処理の多重化
 - 同時に複数のクライアントに対する処理を行う
- 多重化の方法
 - fork()
 - select()
 - thread
 - epoll
 - kqueue

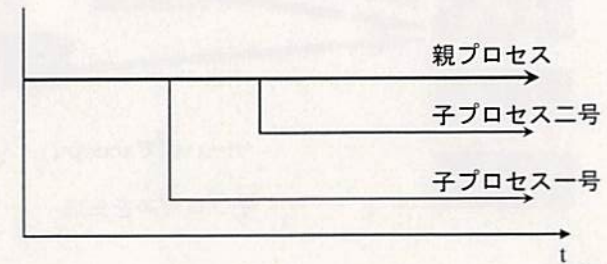
2012/01/11

+ ノンブロッキングサーバの構成方法



+ fork - 自分の分身を作る

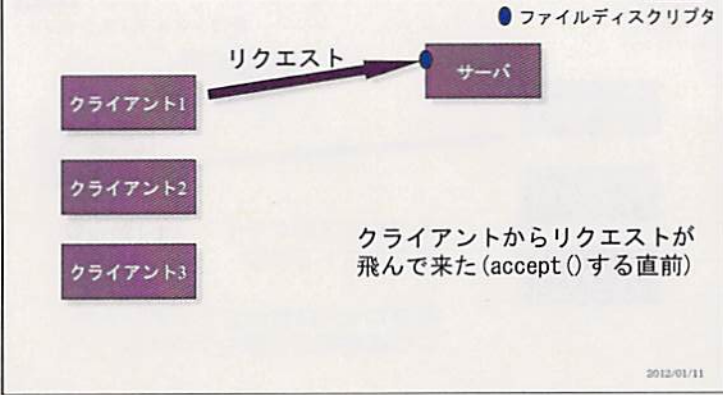
- システムコールで自分の分身を作成する
- 親の仕事は、コネクションがきたときに分身を作ることだけ



+ fork() を用いたサーバの仕組み

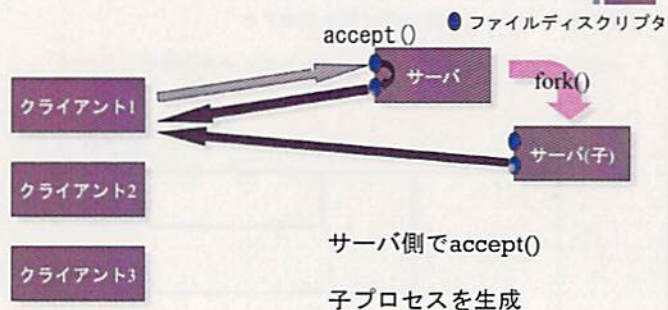


+ fork() を用いたサーバの仕組み



fork()を用いたサーバの仕組み

45



2012/01/11

fork()を用いたサーバの仕組み

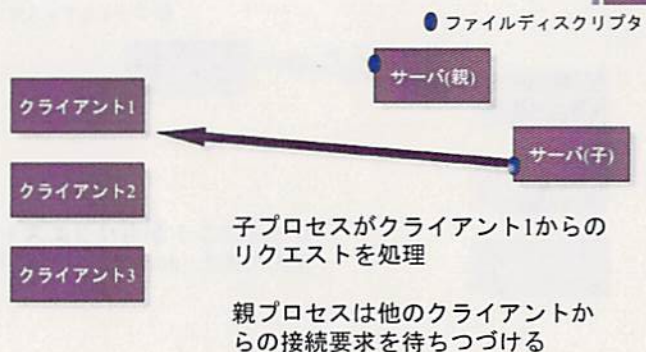
46



2012/01/11

fork()を用いたサーバの仕組み

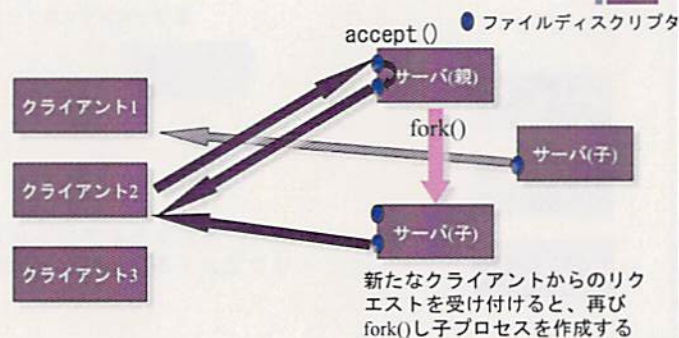
47



2012/01/11

fork()を用いたサーバの仕組み

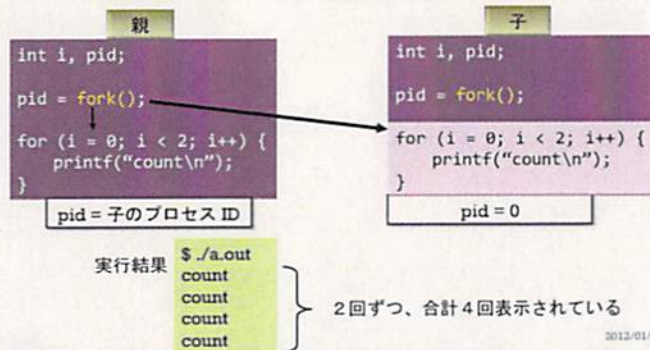
48



2012/01/11

+ fork()

- 自分とまったく同じ複製プロセスを作るシステムコール



+ 多重化サーバの作り方 (fork編)

■ サンプルコード

```
pid_t pid;
int listenfd, connfd;

listenfd = socket(...);
bind(listenfd, ...);
listen(listenfd, LISTENQ);

for (;;) {
    connfd = accept(listenfd, ...);

    if ((pid = fork()) == 0) {
        close(listenfd); doit(connfd);
        close(connfd); exit(0);
    }
    close(connfd);
}
```

■ fork() について

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

```
pid_t fork(void)
```

戻り値: 子プロセスなら 0
親プロセスなら子プロセスの PID
エラーなら -1

2012/01/11

+ 多重化サーバの作り方(thread編)

■ サンプルコード

```
static void *doinit(void *connfd)
{
    pthread_detach(pthread_self());
    do_task((int) connfd);
    close((int) connfd);
    return(NULL);
}

int main()
{
    int i connfd;
    pthread_t tid;

    socket(...); bind(...); listen(...);

    for (;;) {
        connfd = accept(...);
        pthread_create(&tid, NULL, &doinit,
            (void*)connfd);
    }
}
```

■ pthread_create()について

```
#include <pthread.h>
```

```
int pthread_create(pthread_t *restrict thread,
    const pthread_attr_t *restrict attr,
    void *(*start_routine)(void *),
    void *restrict arg)
```

戻り値: thread が生成できたら 0
thread が生成できない時はエラー値

2012/01/11

+ 多重化サーバの作り方(select編)

■ サンプルコード

```
for (i=0; i < FD_SETSIZE; i++)
    client[i] = -1;
FD_ZERO(&allset);
FD_SET(listenfd);

for (;;) {
    rset = allset;
    ready = select(maxfd+1, &rset, NULL, NULL, NULL);
    if (FD_ISSET(listenfd, &rset)) {
        connfd = accept(...);
        client[i] = connfd;
        FD_SET(connfd, &allset);
    }
    for (i=0; i <= maxfd; i++) {
        if (FD_ISSET(client[i], &rset)) {
            n = read(client[i], buff, sizeof(buff));
        } else {
            write(client[i], buff, n);
        }
    }
}
```

■ select() について

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

```
int select(int nfds,
    fd_set *restrict readfds,
    fd_set *restrict writefds,
    fd_set *restrict errorfds,
    struct timeval *restrict timeout);
```

戻り値: ready descriptor の総数
エラーの時は、-1

2012/01/11

+ 利点・欠点

| 方式 | 利点 | 欠点 |
|--------|-------------------------|--|
| fork | 変数空間が独立 プロセス独立 | メモリ利用効率 利用できるOSに制限 |
| select | 資源の有効利用 プロセス内部処理 | 処理が複雑になりがち ソケットにバケットが届いているかを逐一確認しなければならないため、処理が重くなりがち |
| thread | メモリ有効利用 ほとんどのOSで利用可能 | デバッグが難しい |

2012/01/11

+ ホスト名とIPアドレス (1)

- IPアドレスは数字の並びなので、人間が使うには適さない
 - 61.121.201.83
 - www.u-tokyo.ac.jp
 - 覚えやすいのはどっち？
- インターネットのアプリケーションはIPアドレスの代わりにホスト名を使うことができる

2012/01/11

+ ホスト名とIPアドレス (2)

- インターネットでは、Domain Name System (DNS) を使ってホスト名とIPアドレスの相互変換をしている

www.u-tokyo.ac.jp
↔
133.11.114.194
 ホスト名 IPアドレス

```
% nslookup www.u-tokyo.ac.jp
Non-authoritative answer:
Name: www.u-tokyo.ac.jp
Address: 133.11.114.194
```

2012/01/11

+ ネットワークプログラムでホスト名を扱う

- 名前⇄アドレス変換を行うライブラリ関数を使って操作する
 - gethostbyname ()
 - 基本的な名前→アドレス変換関数
 - getaddrinfo()
 - gethostbyname を汎用的に拡張した関数
 - マルチプロトコル対応

2012/01/11

+ getaddrinfo()

```
getaddrinfo(    const char *nodename,
                const char *servname,
                const struct addrinfo *hints,
                struct addrinfo **res);
```

■ 引数

- nodename: ホスト名,
- servname: ポート番号
- hints: 取得する情報のタイプに関するヒント,
- res: 取得した結果の addrinfo リストが格納される

■ 返り値:

- 成功すると 0 を返
- 失敗すると 0 以外のエラーコード

2012/01/11

+ addrinfo 構造体

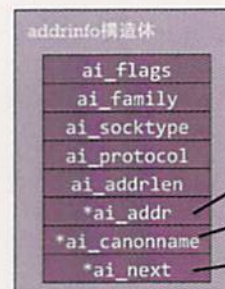
- ホスト名、サービス名、addrinfo 構造体を関数に渡すと、addrinfo 構造体を代入する

addrinfo 構造体

```
struct addrinfo {
    int ai_flags;           /* input flags */
    int ai_family;         /* protocol family */
    int ai_socktype;        /* socket type */
    int ai_protocol;       /* protocol for socket */
    socklen_t ai_addrlen;   /* ソケットアドレスの長さ */
    struct sockaddr *ai_addr; /* 対象のソケットアドレス */
    char *ai_canonname;     /* 正式名称 */
    struct addrinfo *ai_next; /* リスト構造で使うポインタ */
};
```

2012/01/11

+ addrinfo 構造体



addrinfo 構造体はリスト構造になっている

sockaddr 構造体

www.yahoo.com

次の addrinfo 構造体

ai_flags

...

2012/01/11

+ getaddrinfo() を使ったサンプルコード

```
char *hostname = "www.yahoo.com";
struct addrinfo hints, *res;
struct in_addr addr;
int err;

memset(&hints, 0, sizeof(hints));
hints.ai_socktype = SOCK_STREAM;
hints.ai_family = AF_INET;

if ((err = getaddrinfo(hostname, NULL, &hints, &res)) != 0) {
    printf("error %d\n", err);
    return 1;
}

while (res != NULL) {
    printf("ip address : %s\n", inet_ntop(res->ai_family, &res->ai_addr,
        buf, sizeof(buf)));
    res = res->ai_next;
}

freeaddrinfo(res);
```

2012/01/11

+ getaddrinfo() 実行例

```
sekiya@LECTURE-CLIENT:~/EXAMPLE$ ./getaddr
```

```
ip address : 80.244.171.8
```

```
ip address : 136.244.171.8
```

```
ip address : 192.244.171.8
```

```
ip address : 248.244.171.8
```

2012/01/11