ネットワーク実験

ソケットを用いた
ネットワークプログラミング実習

シラバス

- [授業の概要]
  - 授業科目「ネットワーク実験」の１課題として，ソケットを用いたクライアント・サーバプログラミングの実習を行い，ネットワークアプリケーションプログラミングの基礎を学習する．

- [授業の内容]
  - 1.TCP/IPプロトコルとソケットの基礎
  - 2.ソケットを用いたコネクションレス型クライアント/サーバネットワークプログラミング
  - 3. ソケットを用いたコネクション型クライアント/サーバネットワークプログラミング

  前半2週

  後半2週

- [到達目標]
  - ソケットを用いたネットワークプログラミングの基礎技術を習得する．

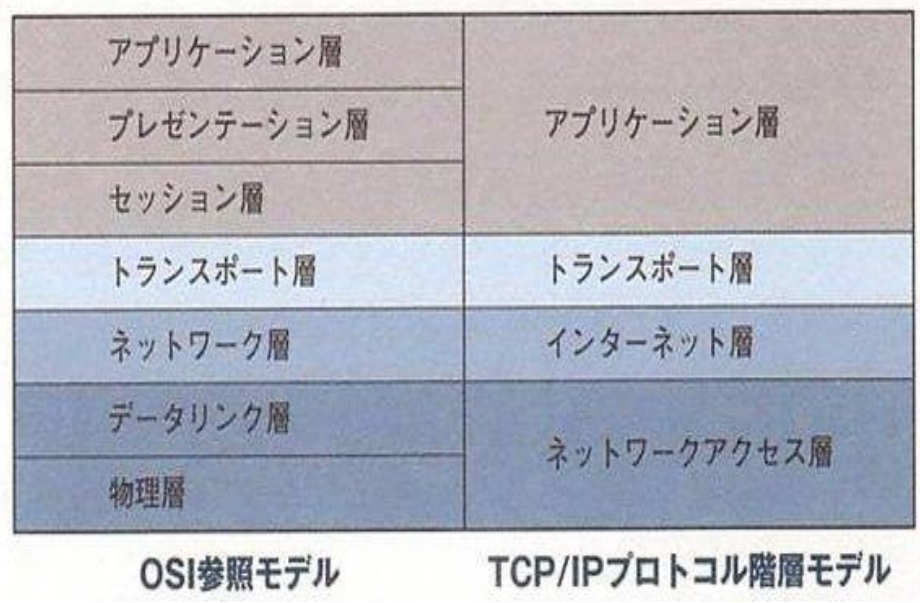
- [参考書]
  - 基礎からわかるTCP/IP　ネットワークコンピューティング入門，村山公保著，オーム社，¥2,200．
  - TCP/IPソケットプログラミング C言語編，M.J. Donahoo, Kenneth L. Calvert 共著，小高知宏監訳，オーム社．¥1,800．

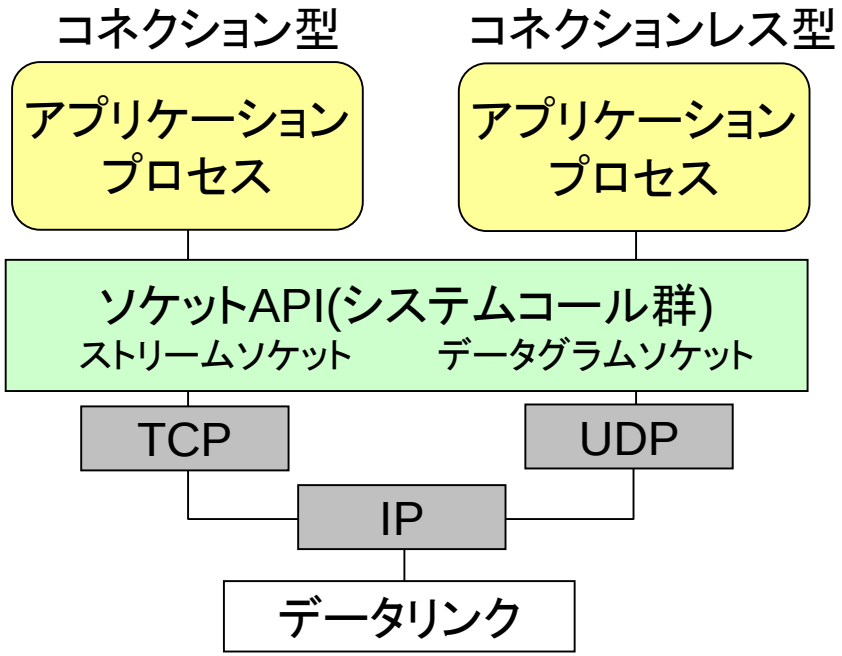
第1&2回

コネクションレス型
クライアント/サーバモデル

ソケットAPI(Socket Application Programming Interface)

- アプリケーションに対して，異なるホスト間の通信をサポートするためのトランスポート層以下の機能を提供するプログラミングインターフェースで，クライアント/サーバモデルを実現するために用いられる．

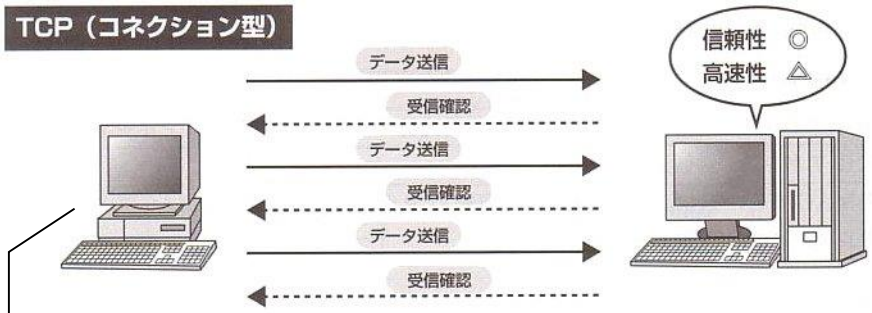




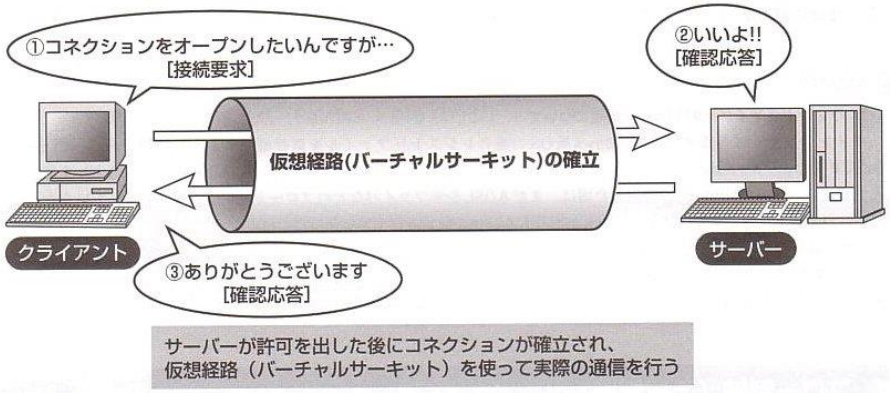
TCPとUDPの違い

- TCP
  - コネクション型
  - 信頼性高い：データが相手に到着することを随時確認
  - 送信効率低い：コネクションの確立・切断の制御，送達確認等のパケットが使われる

- UDP
  - コネクションレス型
  - 信頼性低い：データが相手に到着することを確認しない
  - 送信効率高い：ヘッダ情報を付与する以外はほとんど素通りで上下の層にデータを渡す



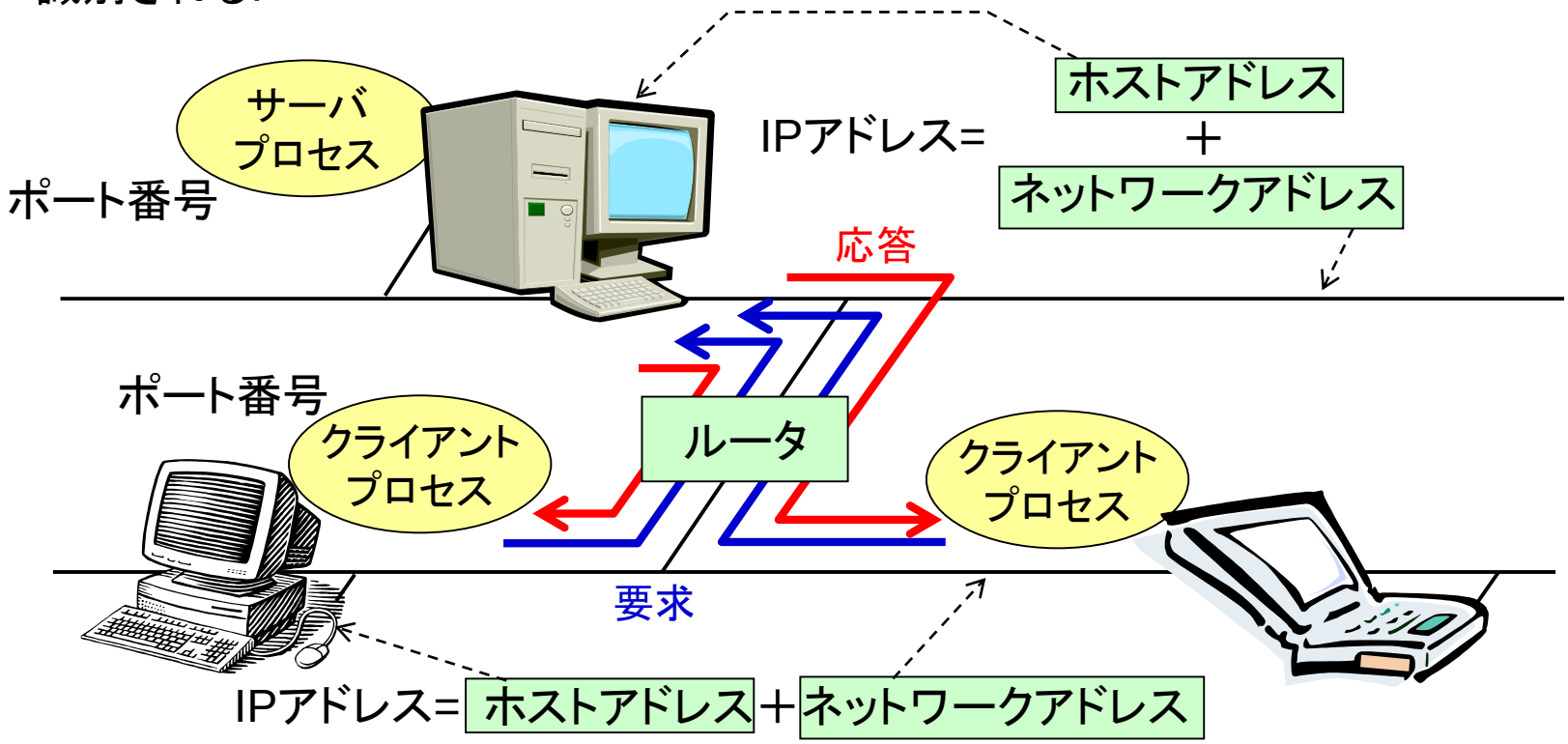
事前に仮想経路を確立



コネクション：クライアントとサーバ間でデータを流すために使用する仮想的な通信路

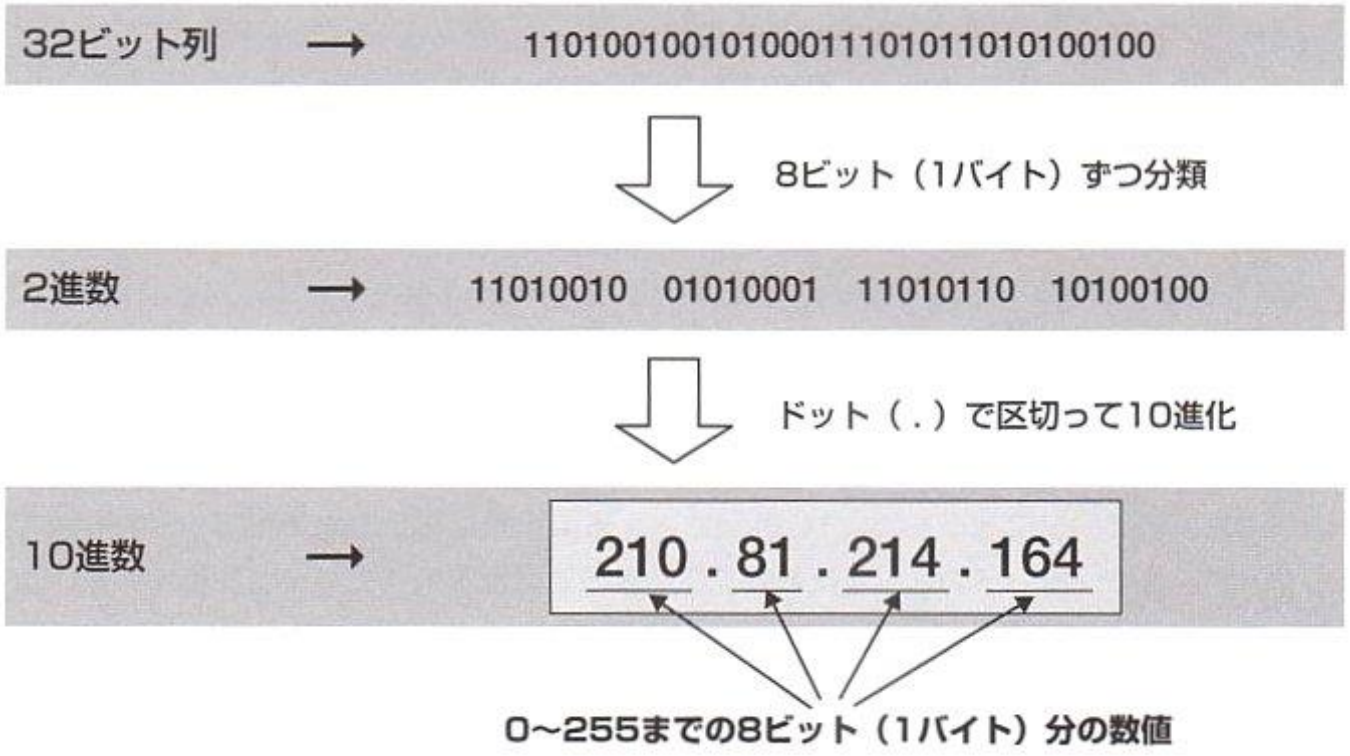
クライアント/サーバモデル

- クライアント/サーバモデルは，ネットワークアプリケーションにおけるプログラム(プロセス)間の相互作用の標準的モデルである．
  - サーバ：クライアントからの要求を受け入れ，それに対するサービスを実行し，結果を返すプロセス．
  - クライアント：ある処理を，サーバに要求を送り結果を受け取ることにより実行するプロセス．
- 通信は，クライアントのIPアドレスとポート番号，サーバのIPアドレスとポート番号により識別される．



IPアドレスとその表記方法

- IPアドレス(IPv4アドレス)
  - 個々のホストを識別するための「32ビットのビットパターン」
  - インターネット上のホストが一意に決定されるようにインターネット全体で管理.
- IPアドレスの表記方法



(註)ただし,「0.1.2.3」のように先頭が0から始まるアドレスは利用できない.

ポート番号

- 16ビット(0～65535の範囲)の番号で各種通信サービス(アプリケーションプロトコル)を識別するために用いられる.
- ポート番号の分類
  - The Well Known Ports(0~1023)：ウェルノウンポート番号．特に良く利用されるアプリケーションを識別するために予約されている番号で，どのサーバもウェルノウンポート番号に従ってサービス提供する.
  - The Registered Ports(1024~49151)：特定のアプリケーションに割り当てられていることもあるが，他のアプリケーションに割り当ててもよいことになっている番号．(最近のサービス種類の増加により，この範囲の番号でもサービスとして正式に登録されているのもがある．）
  - The Dynamic and/or Private Ports(49152~65535)：クライアントのポート番号として動的に割り当てたり，独自のアプリケーションのポート番号として自由に使ってよい番号.
- 慣例的には，Windows系OSでは1024番以降，UNIX系OSでは8000番以降の，その時点でクライアントホストが使用していないランダムな番号がクライアントのポート番号として動的に割り当てられる.

クライアント/サーバモデルにおけるサーバとクライアントの動作

- サーバの動作
  - (1)通信チャネルをオープンし，自らのアドレス(IPアドレスとポート番号)をOSに登録する．即ち，登録したアドレスに届いたメッセージをオープンした通信チャネルに渡すようにOSに依頼する．
  - (2)そのアドレスにおいて，クライアントからの要求が到着するのを待つ．
  - (3)受け入れた要求を処理して結果をクライアントに返す．
  - (4)別のクライアントからの要求を待つ((2)にジャンプ)

- クライアントの動作
  - (1)通信チャネルをオープンし，あるホストのサーバ(アドレス)に要求を送る準備をする．(例えば，自らのアドレスを登録したり，接続を確立したりする．）
  - (2)サーバに要求を送り，処理結果を受け取る．
  - (3)処理が終わったならば，通信チャネルをクローズして終了する．

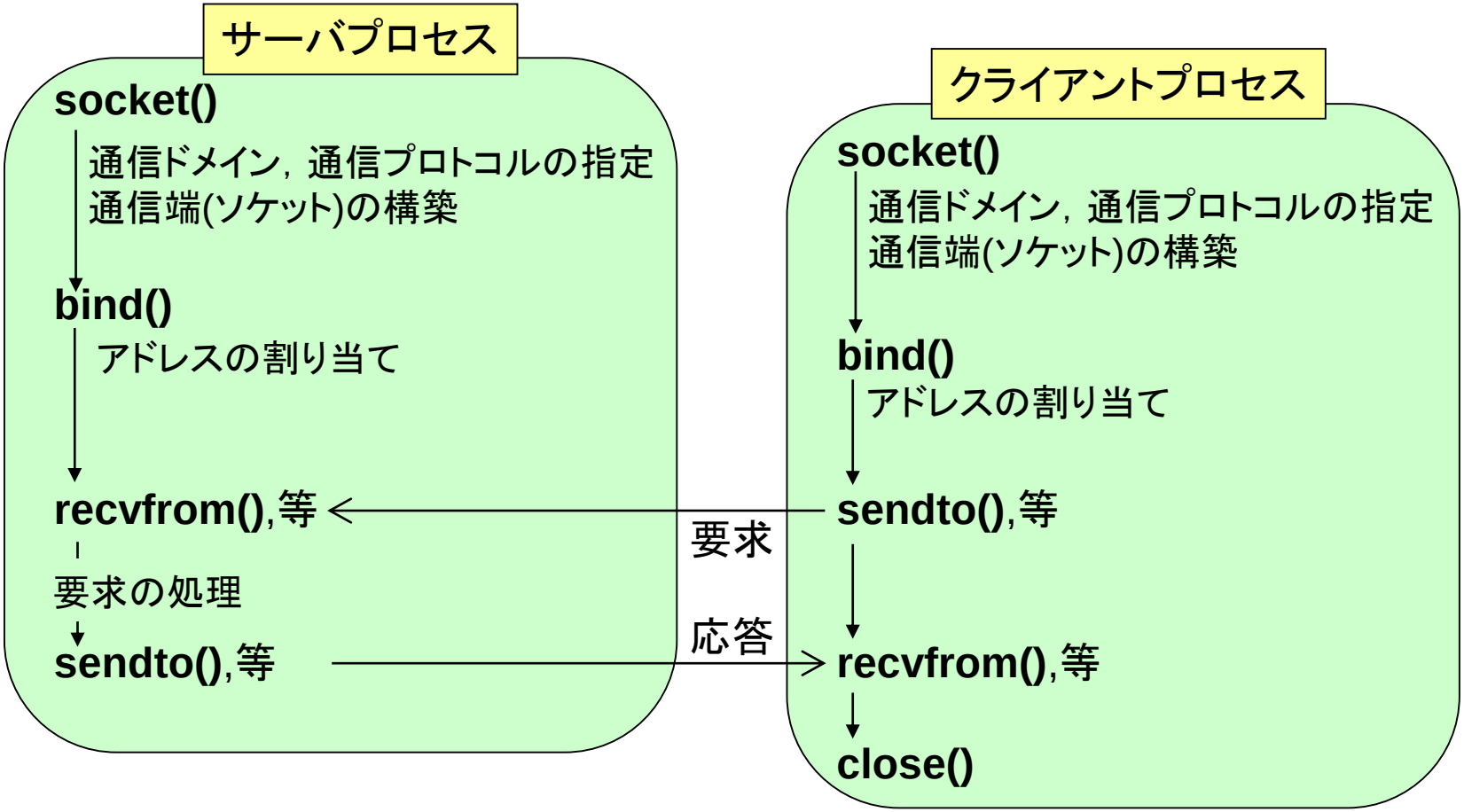
サーバの種類

- 並行サーバ
  - クライアントからの要求に対して，forkシステムコールにより他のプロセスを生成して，そのプロセスに要求の処理を任せる．
  - 生成された子プロセスは，要求を処理し，クライアントとの通信チャネルをクローズして終了する．
  - 親プロセスであるサーバは，別のクライアントからの要求を待つ．
  - 要求の処理に要する時間が不明で，要求によって処理時間が変わりうる処理を並行的に行う必要があるサーバに用いられる．
- 反復サーバ
  - クライアントからの要求に対して，自らがその処理を行い結果をクライアントに返す．
  - あらかじめわかっている短い時間で処理可能な要求を逐次的に処理するサーバに用いられる．

ソケットを用いたプロセス間通信

- ソケットAPI(Socket Application Programming Interface)：アプリケーションに対して，異なるホスト間の通信をサポートするためのトランスポート層以下の機能を提供するプログラミングインターフェースで，クライアント/サーバモデルを実現するために用いられる．
- ソケットの種類：ストリームソケットとデータグラムソケット
  - ストリームソケット
    - TCPプロトコルに従うコネクション型の接続形態，即ちクライアントプロセスとサーバプロセスの間に通信に先立って論理的な接続を確立する信頼性の高い通信を提供．（コネクション型の接続形態はバーチャルサーキットとも呼ばれる．）
  - データグラムソケット
    - UDP プロトコルに従うコネクションレス型の接続形態，即ちデータグラムと呼ばれるメッセージを単位に行われる送信効率の高い通信を提供．（コネクションレス型の接続形態はデータグラムとも呼ばれる．）
- 並行サーバに対してはコネクション型の接続形態，反復サーバに対してはコネクションレス型の接続形態が用いられることが多い，

コネクションレス型クライアント/サーバモデルの実装の枠組



例題1:コネクションレス型クライアント/サーバシステム(1)

- 問題:簡単なデータ検索クライアント/サーバシステム
  - サーバ上の電話番号に関するデータベースをクライアントの要求に応じて検索し，検索結果をクライアントに返す.
- プログラムの構成
  - サーバ:dg_server.c
  - クライアント:dg_client.c
- 実行例

サーバ(例えば，cpu1)

```
% ./dg_server
Received key> ishida-jiro
Sent data>0426-91-9872
```

クライアント(例えば，e408d201)

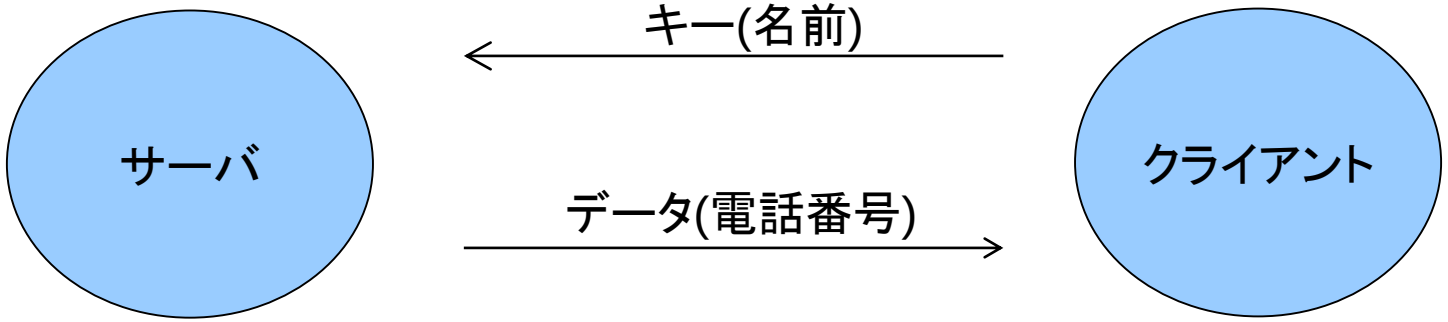
```
% ./dg_client
server host name?: cpu1
Key?:  Ishida-jiro
Data: 0426-91-9872
```

例題1:コネクションレス型クライアント/サーバシステム(2)

- データベースの形式
  - キー(名前)とデータ(電話番号)が交互に並んだ文字列(文字ポインタ)の配列(最後にNULLが入っている).

```
static char *db[] = {“amano-taro”,”0426-91-9418”,
                     “ishida-jiro”,”0426-91-9872”,
                     “ueda-saburo”,”0426-91-9265”,
                     “ema-shiro”,”0426-91-7254”,
                     “ooishi-goro”,”0426-91-9618”,
                     NULL};
```

- 通信プロトコル



例題1:コネクションレス型クライアント/サーバシステム(3)

- コンパイルの方法
  - e408dxxx(pclinux)の場合
    gcc -o dg_server dg_server.c
    gcc -o dg_client dg_client.c
  - cpu1(sun)の場合：telnet cpu1.soka.ac.jpでcpu1にログイン後,
    gcc -o dg_server dg_server.c -lsocket -lnsl
    gcc -o dg_client dg_client.c -lsocket -lnsl
- 実行の方法
  - サーバーをe408dxxxで実行する場合：
    ./dg_server
  - クライアントをe408dxxxで実行する場合：
    ./dg_client
  - サーバーをcpu1で実行する場合：telnet cpu1.soka.ac.jpでログイン後,
    ./dg_server

問題1

- (1)このクライアントプログラムは1回の実行で1つの電話番号の検索しかできないが，CTRL-Dが入力されるまで繰り返し検索ができるようにプログラムを改良せよ.
- (2)一度に複数の名前を指定してそれらの電話番号の検索ができるようにプログラムを改良せよ．即ち，
  - クライアントからの問い合わせメッセージ”name1,name2,…”に対して，サーバは電話番号”tel-number1,tel-number-2,…”を返す.
- (3)データベースの電話番号の検索だけでなく，更新もできるようにプログラムを改良せよ．即ち，
  - クライアントからの問い合わせメッセージ”GET:name1,name2,…”に対して，サーバは電話番号”tel-number1,tel-number-2, …”を返す.
  - クライアントからの更新メッセージ”PUT:name,tel-number”に対して，サーバは”name”の電話番号を”tel-number”に更新して，古い電話番号を返す.

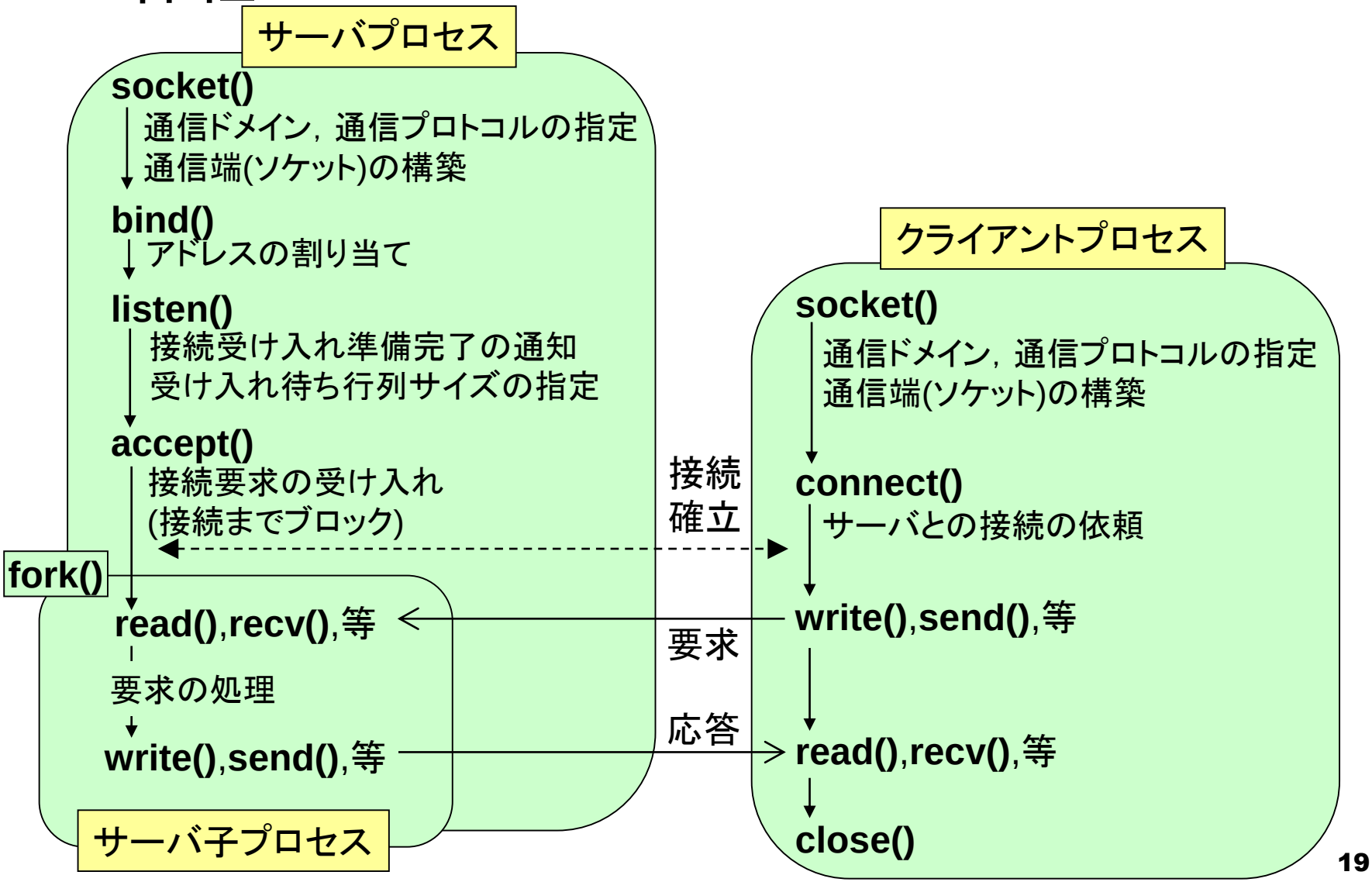
課題レポート1

- 問題
  - 例題1のプログラムを問題1の要求に従って拡張しなさい.
- 提出物
  - プログラムの拡張点の説明を記述したWordファイル
    - 拡張の方針
    - 拡張コードの説明
    - 実行例を用いての拡張箇所のテスト結果
  - プログラムファイル
- 提出期限
  - 第3週の実験日の朝9時
- 提出先：ポータル

第3,4回

コネクション型
クライアント/サーバモデル

コネクション型クライアント/サーバモデルの実装の枠組



例題2：コネクション型クライアント/サーバシステム(1)

- 問題：簡単なリモートファイル転送サーバ/表示クライアントシステム
  - クライアントの要求するサーバ上のファイルをサーバがクライアントに転送し，それをクライアントが表示する
- プログラムの構成
  - サーバ：vc_server.c
  - クライアント：vc_client.c
- 実行例

サーバ(例えば，cpu1)

```
% ./vc_server
ファイル /etc/services を送信
```

クライアント(例えば，e408d201)

```
% ./vc_client
server host name?: cpu1
remote file name?:  /etc/services
ファイル /etc/services を受信
#ident  "@(#)services   1.27    00/11/06 SMI"   /* SVr4.0
#
#
# Copyright (c) 1999-2000 by Sun Microsystems, Inc.
# All rights reserved.
#
# Network services, Internet style
#
tcpmux          1/tcp
echo            7/tcp
echo            7/udp
```

例題2:コネクション型クライアント/サーバシステム(2)

- 通信プロトコル

例題2:コネクション型クライアント/サーバシステム(3)

- コンパイルの方法
  - e408dxxx(pclinux)の場合
    gcc -o vc_server vc_server.c
    gcc -o vc_client vc_client.c
  - cpu1(sun)の場合:telnet cpu1.soka.ac.jpでcpu1にログイン後,
    gcc -o vc_server vc_server.c -lsocket -lnsl
    gcc -o vc_client vc_client.c -lsocket -lnsl
- 実行の方法
  - サーバーをe408dxxxで実行する場合:
    ./vc_server
  - クライアントをe408dxxxで実行する場合:
    ./vc_client
  - サーバーをcpu1で実行する場合:telnet cpu1.soka.ac.jpでログイン後,
    ./vc_server

問題2

- (1)このクライアントプログラムは1回の実行で1つのファイル転送しかできないが，CTRL-Dが入力されるまで繰り返しファイル転送ができるようにプログラムを改良せよ.
- (2)サーバ上の複数のファイルの転送を同時に指定することできるようにプログラムを改良せよ．即ち，
  - クライアントからの複数のファイル転送要求に対して，サーバは各ファイルが存在するかどうかの応答，及び存在する場合はそのファイルの内容の転送を行う.
- (3)サーバからのファイルのダウンロードだけでなく，サーバにファイルをアップロードできるようにプログラムを改良せよ．即ち，
  - クライアントからの複数のファイルダウンロード要求に対して，サーバは各ファイルが存在するかどうかの応答，及び存在する場合はそのファイルの内容の転送を行う.
  - クライアントからのファイルアップロード要求”PUT:ファイル名”に対して，サーバは送信される内容を標準出力に出力する，もしくはそのファイル名で保存する.

課題レポート2

- 問題
  - 例題2のプログラムを問題2の要求に従って拡張しなさい.
- 提出物
  - プログラムの拡張点の説明を記述したWordファイル
    - 拡張の方針
    - 拡張コードの説明
    - 実行例を用いての拡張箇所のテスト結果
  - プログラムファイル
- 提出期限
  - 次の実験テーマの最初の実験日の朝9時
- 提出先:ポータル

資料

ソケットAPI，他

ソケット基本システムコール(1)

- socketシステムコール
  - int socket(int address_family, int socket_type, int protocol);
  - 通信の端点(socketデータ構造体)を生成し，そのソケット記述子を得る.
  - 引数
    - int address_family：通信ドメインを指定する．インターネットドメインの場合は定数AF_INETを指定する.
    - int socket_type：ソケットの型-バーチャルサーキットかデータグラムか-を指定する．バーチャルサーキットの場合は定数SOCK_STREAMを，データグラムの場合は定数SOCK_DGRAMを指定する.
    - int protocol：通常は0をセットする．0をセットすると，OSが適当なプロトコルを選んでくれる.
  - 戻り値：成功した場合，ソケット記述子(非負整数)を，失敗した場合，-1を返す.

通信ドメイン

- 通信ドメイン：異なるホスト上のプロセスを識別するためには，各ホストの各プロセスにそれを識別する名前（アドレス）をつけることが必要となる．あるアドレス体系に従って名前を付けられるホスト/プロセスの領域を通信ドメインという．通信ドメインはアドレスの形式，及びその解釈の仕方を規定する．
- ソケットがサポートする通信ドメイン
  - インターネットドメイン(定数AF_INETで識別)：異機種ホスト上のプロセス間の通信をTCP/IPプロトコル群に従って行うための通信ドメイン
  - XNSドメイン(定数AF_NSで識別)：Xeroxネットワークシステムのための通信ドメイン
  - Unixドメイン(定数AF_UNIXで識別)：同一ホスト上でソケットを用いてプロセス間通信を行うための通信ドメイン

ソケット基本システムコール(2)

- bindシステムコール
  - int bind(int socd, struct sockaddr *address, int addrlen);
  - ソケットにアドレスを割り当てる．2つのソケット間で通信を行うためには各ソケットがアドレスを持っていなくてはいけない．
  - 引数
    - int socd：ソケット記述子(socketシステムコールの戻り値)
    - struct sockaddr *address：socketシステムコールで指定した通信ドメイン特有のアドレス構造体へのポインタ．(すべてのドメインを対象とした汎用アドレス構造体struct sockaddrへのポインタにキャストして渡される．）

```
struct sockaddr_in address;
…sin_family, sin_port, sin_addrメンバに値を設定…
bind(sd, (struct sockaddr *)&address, sizeof(address));
```

    - int addrlen：アドレス構造体の大きさ．(アドレス構造体の大きさが通信ドメインによって異なるため，大きさを渡さないといけない．)
  - 戻り値：成功した場合0，失敗した場合-1を返す．

sockaddr_in構造体

- インターネットドメインのアドレス構造体(16バイト). アドレス構造体の形式は通信ドメインごとに異なる.

```
struct sockaddr_in {
    short           sin_family;   /* アドレスファミリAF_INET */
    u_short         sin_port;     /* ネットワークバイト順序のポート番号(16ビット) */
    struct in_addr  sin_addr;     /* ネットワークバイト順序のIPアドレス(32ビット) */
    char            sin_zero[8];  /* 未使用 */
};
```

  - struct in_addr型は実質的にはu_long型.

```
struct in_addr {
    u_long          s_addr;
}
```

  - すべての通信ドメインを対象とした汎用のアドレス構造体の型としてsockaddr構造体が用意されている.

```
struct sockaddr {
    u_short         sa_family;
    char            sa_data[14];
};
```

ネットワークバイト順序

- コンピュータは，そのベンダーの種類により複数のバイトで構成されるデータの格納方法が異なる．例えば，整数を格納するのに，あるコンピュータ(例えば，インテル80x86)では開始アドレスに下位のバイトを格納し，一方，別のコンピュータ(例えば，モトローラ680x0)は開始アドレスに上位のバイトを格納する．

- 通信プロトコルでは，アドレスを指定するのに複数バイト整数を用いるため，通信ドメインで統一的にバイト格納順序を決めて，通信に先立ってプロトコルが用いるデータをその順序に変換することが必要となる．

- この通信ドメインで決められたバイト格納順序をネットワークバイト順序という．一方，ホストごとに決められたバイト格納順序をホストバイト順序という．

hostent構造体

- ホスト名とそのアドレスを対応付ける構造体で<netdb.h>で定義される.

```
struct hostent {
    char   *h_name;          /* ホストの正式名 */
    char   **h_aliases;      /* ホストの別名のリスト */
    int    h_addrtype;       /* ホストアドレスのタイプ(=AF_INET) */
    int    h_length;         /* ホストアドレスの長さ(=4(IPアドレス長)) */
    char   **h_addr_list;    /* ホストアドレスのリスト(リストの終わりはNULL) */
#define h_addr    h_addr_list[0]
};
```

  - hostent構造体中のデータはネットワークバイト順序で格納されている.
  - ポインタ配列h_addr_list[0], h_addr_list[1], …は文字列のポインタではなくin_addr構造体(実質的にはu_long)へのポインタが格納される.

```
struct in_addr {
    u_long          s_addr;
}
```

gethostnameシステムコール

- int gethostname(char *name, int namelen);
  - 自らのプロセスが存在するホストの名前を求めるために用いられる. 第1引数にホスト名が求まる.
  - 引数
    - char *name：ホスト名の文字列配列へのポインタ
    - Int namelen：ホスト名を入れる文字列配列の長さ.
  - 戻り値：成功した場合0を，失敗した場合-1を返す.

gethostbynameライブラリ関数

- struct hostent *gethostbyname(char *name);
  - ホスト名からそのアドレスをメンバに持つhostent構造体を求めるために用いられる.
  - 引数
    - char *name：ホスト名の文字列配列へのポインタ
  - 戻り値：指定したホストが見つかった場合，そのホスト名とアドレスをメンバに持つhostent構造体が返される．指定したホストが見つからない場合にはNULLが返される.

参考:ホスト名とホストアドレスを求めて出力

- 次のプログラム(host.c)をサーバマシンcpu1とクライアントマシンe408dxxxで動かしてみよう.

```
/* ホスト名とホストアドレスを求め出力する */
#include <stdio.h>
#include <netdb.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <errno.h>
#define  NAMELEN        64  /* ホスト名の長さの限界値 */

struct hostent  *h;
char   hostname[NAMELEN];

main() {
        if(gethostname(hostname,NAMELEN) < 0) { perror("gethostname");exit(1); }
        if((h = gethostbyname(hostname)) == NULL) { fprintf(stderr,"hostname error¥n");exit(1); }
        printf("%s %s¥n",h->h_name, inet_ntoa((struct in_addr *)(h->h_addr)));
        /* char   *inet_ntoa(const struct in_addr inaddr);                    */
        /* は、アドレスのin_addr構造体形式をdotted-decimal形式に変換するライブラリ関数 */
}
```

- 実行結果(cpu1の場合)

```
% ./host
cpu1 150.37.112.133
```

bzero,bcopyライブラリ関数

- ネットワークバイト順序を保ったままデータを操作するために，それらデータをバイト列として操作するライブラリ関数.

- void bzero(char *mem, int length);
  - Memから始まる領域に0(ヌルバイト)をkengthバイト分セット.

- void bcopy(char *src, char *dest, int length);
  - バイト列srcをdestにlengthバイト分コピー.
  - 指定された長さのバイト列中にヌル文字が現れる場合もそれをコピーする.

ホストバイト順序をネットワークバイト順序に変換するライブラリ関数

- u_long htonl(u_long hostlong);
  - ホストバイト順序の長整数longをネットワークバイト順序の長整数longに変換.
- u_short htons(u_short hostshort);
  - ホストバイト順序の短整数shortをネットワークバイト順序の短整数shortに変換.

ポート番号とIPアドレスの自動割り当て

- ポート番号の自動割り当て
  - c_address.sin_port = htons(0)
  - クライアントが自らのポート番号を動的に割り当てる場合に利用
- IPアドレスの自動割り当て
  - c_address.sin_addr.s_addr = htonl(INADDR_ANY)
  - クライアントが自らのIPアドレスを自動的に割り当てる場合に利用
  - サーバが自らのIPアドレスを自動的に割り当てる場合に利用

ソケット基本システムコール(3)

- sendtoシステムコール
  - int sendto(int socd, char *msg, int len, int flags, struct sockaddr *to, int tolen);
  - 送信先を指定してデータを送信する.
    (主に，コネクションレス型の通信に用いられる．)
  - 引数
    - int socd：ソケット記述子
    - char *msg：送信データへのポインタ
    - int len：送信データ長
    - int flags：通常は0
    - struct sockaddr *to：送信先のアドレス(構造体へのポインタ)
    - int tolen：送信先アドレス構造体の大きさ
  - 戻り値：成功した場合，送信したバイト数を，失敗した場合，-1を返す.

ソケット基本システムコール(4)

- recvfromシステムコール
  - int recvfrom(int socd, char *buf, int len, int flags, struct sockaddr *from, int *fromlen);
  - データを受信し，同時に送信元のアドレスを受け取る．
    (主に，コネクションレス型の通信に用いられる．)
  - 引数
    - int socd：ソケット記述子
    - char *buf：受信データへのポインタ
    - int len：受信データ長(予定)
    - int flags：通常は0
    - struct sockaddr *from：送信元のアドレス(構造体)が返される．送信元のアドレスを必要としないときはNULLとすればよい．
    - int *fromlen：送信元のアドレス構造体の大きさが返される．(呼び出し前には，第5引数fromの大きさにセットしておく．呼び出しから戻ったときに実際の大きさがセットされる．)
  - 戻り値：成功した場合，受信したバイト数を，失敗した場合，-1を返す．

ソケット基本システムコール(5)

- closeシステムコール
  - int close(int socd);
  - ソケットをクローズする.
  - 引数
    - int soocd：ソケット記述子.
  - 戻り値：成功した場合0，失敗した場合-1を返す.

ソケット基本システムコール(6)

- listenシステムコール
  - int listen(int socd, int backlog);
  - コネクション型のサーバが接続受け入れ準備完了をOSに通知し，接続要求待ち行列の長さを指定する.
  - 引数
    - int socd：ソケット記述子(socketシステムコールの戻り値)
    - int backlog：acceptシステムコールによる接続の受け入れを待つことができる接続要求の最大数. (最大数の上限は5とシステムにより決められている.)
  - 戻り値：成功した場合0，失敗した場合-1を返す.

ソケット基本システムコール(7)

- connectシステムコール
  - int connect(int socd, struct sockaddr *s_address, int s_addrlen);
  - クライアントがサーバとの接続を確立する.
  - 引数
    - int socd：ソケット記述子(socketシステムコールの戻り値)
    - struct sockaddr *s_address：接続先のサーバのアドレス(構造体へのポインタ)
    - int s_addrlen：接続先のサーバのアドレス構造体の大きさ
  - 戻り値：成功した場合0，失敗した場合-1を返す.

- Connectシステムコールの呼び出しによりクライアントのアドレス設定がなされるので，コネクション型の通信ではconnectシステムコールの呼び出し前にbindシステムコールを呼び出す必要がない.

ソケット基本システムコール(8)

- acceptシステムコール
  - int accept(int socd, struct sockaddr *c_address, int *c_addrlen);
  - コネクション型のサーバが接続要求を受け入れる．Acceptシステムコールは待ち行列の先頭の接続要求を取り出し，第1引数のsocdに対応するソケットと同じ性質を持つ別のソケットを生成しそのソケット記述子を返す．通信は，この生成されたソケットを通して行われる．もとのソケットは，クライアントからの接続要求を受け入れるためだけに用いられる．接続要求がない場合は，新たに接続要求が到着するまでブロックする．
  - 引数
    - int socd：ソケット記述子(socketシステムコールの戻り値)
    - struct sockaddr *c_address：接続したクライアントのアドレス(構造体)が返される．サーバがクライアントのアドレスを必要としないときはNULLとすればよい．
    - int *c_addrlen：接続したクライアントのアドレス構造体の大きさが返される．(呼び出し前には第2引数c_addressの大きさにセットしておく．呼び出しから戻ったときに実際の大きさがセットされる．）
  - 戻り値：成功した場合，新しく生成されたソケット記述子，失敗した場合-1を返す．

ソケット基本システムコール(9)

- sendシステムコール
  - int send(int socd, char *msg, int len, int flags);
  - 接続されたソケットを使ってデータを送信する.
  - 引数
    - int socd：ソケット記述子
    - char *msg：送信データへのポインタ
    - int len：送信データ長
    - int flags：通常は0
  - 戻り値：成功した場合，送信したバイト数を，失敗した場合，-1を返す.
- recvシステムコール
  - int recv(int socd, char *buf, int len, int flags);
  - 接続されたソケットを使ってデータを受信する.
  - 引数
    - int socd：ソケット記述子
    - char *buf：受信データへのポインタ
    - int len：受信データ長(予定)
    - int flags：通常は0
  - 戻り値：成功した場合，受信したバイト数を，失敗した場合，-1を返す.