システムプログラミング実験 (OS演習) 第7回

石川研究室　助教　美添 一樹
yoshizoe@is.s.u-tokyo.ac.jp
TA　澤田 武男、藤原 祐二
os-enshu-ta@il.is.s.u-tokyo.ac.jp

予定

- 5月23日
  - マルチスレッド
  - 課題6
- 5月30日
  - ネットワーク・ソケット
  - 課題7
- 6月6日・13日
  - ネットワーク・HTTPサーバー
  - 課題8, 課題9
- 6月20日
  - 質問受け付け
  - 生物情報科学科はここまで
- 6月27日～最後
  - 詳細未定

多少変更される可能性有り。これ以後の予定は、詳細未定。

本日の内容

- ネットワークプログラミングの基礎
  - ネットワーク関連の基礎知識
  - TCP/IP, UDP/IP
  - ソケットを使った通信
  - 複数クライアントへの対応
- 課題7

ネットワークとは

- コンピュータ間をつないで通信する仕組み
- ここでは特にIPネットワークについて
  - IPネットワークではないネットワーク
    - 電話網 (音声電話やi-mode)
    - 銀行のATMネットワーク (大部分はIPではないはず)
    - 人間同士のネットワーク (???)
  - IPネットワークで普段お世話になっているもの
    - インターネット
    - IP電話

プロトコル

- IP = Internet Protocol
- そもそもプロトコルとは？
  - 通信を行うためのルール
  - と言ってもたくさんある
    - 有線？無線？
    - 電圧は？周波数は？
    - 通信相手をどうやって特定する？
    - どうやって遠くの相手にデータを届ける？
    - どうやって通信を開始する？
    - どうやって終了する？
    - 認証は？
  - 通信が正常に行われるためにはこういう（気の遠くなるくらい）細かいことが全部正しく動かなくてはならない
  - こんな細かいルールを一個のプロトコルに定義するのはよくない（実装の難易度、拡張性などいろいろな問題が生じる）

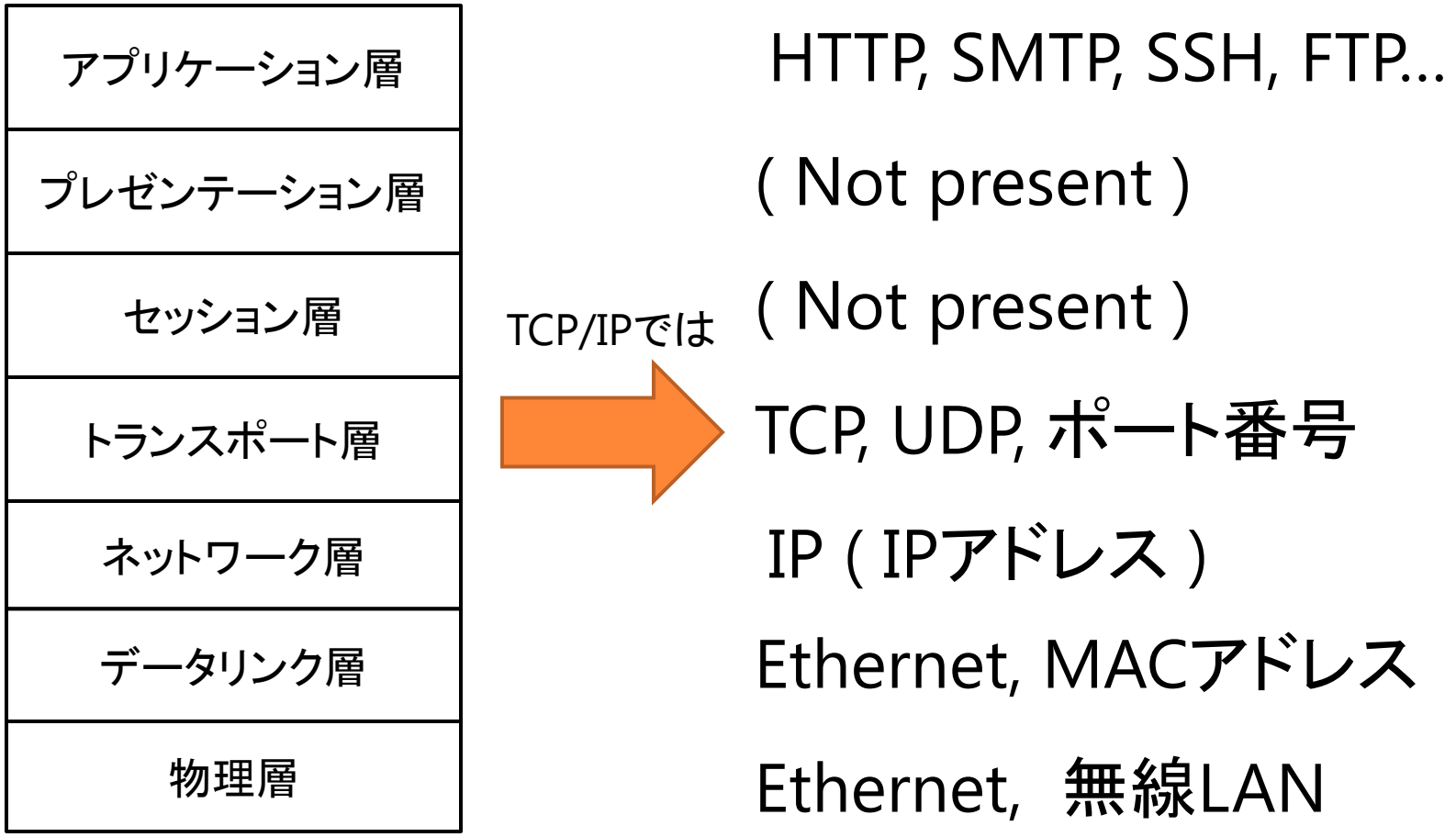
OSI 階層モデル

- OSI階層モデル
  - 責任を分割する
    - 各層は決められた機能を提供するならどうやって作ってもいい
    - 例えば、データが通るなら光ファイバでも銅線でも無線でも物理層としてOK
  - 各層が上位層に対して一定の機能を提供すれば全体が正しく動く
  - ある計算機のN層と他の計算機のN層との規約をプロトコルという
- 階層構造の利点
  - 理解しやすい
  - 開発、保守が容易
    - 新しい技術にも対応できる

OSI 階層モデル

TCP / IPでの階層構造

- インターネットなどで最もよく使われている



OSI 階層モデル

IP

- ネットワーク層のプロトコル
  - データリンク層は何でもよい
    - 身近でよく使うのはEthernet
    - 長距離通信向けのデータリンク層プロトコルにも知らないところでお世話になっている
  - 物理層も何でもよい
    - 大抵、銅線と光ファイバを通ってくる
- 下位層のことは普段は気にしないはず
  - 階層化されているおかげ

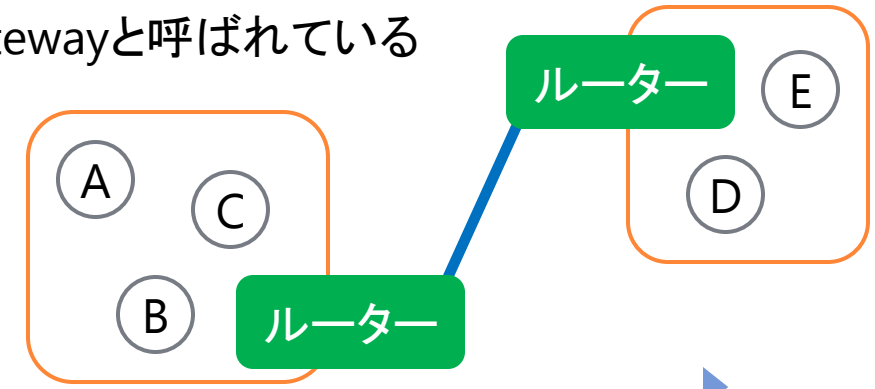
OSI 階層モデル

IPの基礎

- IPアドレス（IPv4の話）
  - 32bitの整数、8bitずつ区切って読む
    - 133.11.233.21
- ネットマスク
  - 32bit = 43億個のアドレスが世界中に散らばっていても困るので細かいネットワーク(Subnet)に分ける
  - IPアドレスの上位何ビットかはsubnetのアドレス、残りがsubnet内でのアドレス
  - 例えば、東大は133.11.*.*, 130.69,*,*, 157.82.*,* の3つのサブネットを持つ
    - この場合、上位16bitがネットワークアドレスなのでネットマスクは255.255.0.0 (/16と書くこともある）
    - つまり、IPアドレスとネットマスクをANDするとネットワークアドレスになる
  - Subnet内のアドレスにはさらに細かいSubnetがあるかもしれない
    - 石川研のsubnetは133.11.233. *
    - ネットマスクは？

ルーティング

- 例えば自分のマシンが133.11.233.130/24だったとする
  - ネットワークアドレスは133.11.233.0
  - これは133.11.233.* のマシンとは直接通信できることを意味する
    - 自分の属するネットワーク全体に届くデータ(ブロードキャスト)を使って「133.11.233.150はどこにいますか？」と聞く。返事をした人にデータを送る
  - では、ネットワーク外のマシンと通信したければ？
    - ルーターに配送を任せる
    - 普通、末端のマシンが属するネットワークにはルーターは一個しかなく、それがすべての中継をする
      - Default gatewayと呼ばれている



ネットワークアドレス
133.11.24.0

ネットワークアドレス
133.11.233.0

ルーティング

- では、Default Gatewayになっているルーターは世界中すべてのマシンの居場所を知っているのか？
  - 普通そんなことはない
  - 例えば7号館のgatewayは
    - 各研究室のネットワークアドレスは知っているので、それらが宛先になっているパケットは当該研究室に送る
    - あとは知らないので東大のネットワーク (UT-Net)のルーターに任せる (つまりルーターにもDefault Gatewayがある)
  - これを繰り返していくと、やがて世界中のネットワークアドレスを知っているルーターにたどりつく
    - このように完全なアドレスのリストを持つルータは実は結構たくさん存在する
    - つまり中心部は木構造になっているわけではない
      - 末端付近は木構造にしておくのが普通だろう

TCP / IP & UDP / IP

- IPだけではまだ問題がある
  - マシンは決まったが、その中のどのプロセスと通信するのかわからない
  - ネットワークの経路中でパケットが落ちたらどうする？
- そこで IP の機能の上にプロセス間（エンドポイント間）で通信を行うプロトコルが規定されている
  - TCP と UDP
    - マシンの中のプロセスを特定するためにポート番号を使う
  - TCP はコネクション型の Reliable 通信
  - UDP はコネクションがなく Unreliable 通信
- 詳しくは後期の授業で

TCP / IP

- コネクション型通信
  - データ転送前に送受信双方で準備を行う
- Reliable 通信
  - すべてのデータが相手に届くことを保障
    - 到着確認のパケットをやり取りし、到着しない場合は再送する
  - 送った側に届くことを保障
    - パケットに番号を付け、受信側でその番号順に並べ替える、といった処理が OS 内部で行われている
- Stream通信
  - データの境界は保存されない
  - 10 bytesずつ10回sendしても、受信側は100bytesの塊を1つとして受信するかもしれない

UDP / IP

- コネクションレス通信
  - 準備なしで突然送受信を開始する
- Unreliable 通信
  - 受信側は送信側が送った順にパケットを受け取るとは限らない
  - 到着すらしないこともある
    - 途中で通過する経路上で何らかの問題が発生したとき
    - 受信するコンピュータやスイッチ・ルータ等の性能が悪く処理が追いつかないとき
- Datagram通信
  - データ境界が保存される
  - 10bytesずつ10回sendしたら、受信側は10bytesの塊を10個受け取る

ポート番号

- IPアドレスではホストの指定のみができる
- 1つのホスト上では複数のサービスが動いていることが多い
  - 例：
    - メールサーバの上に smtp サーバと pop サーバ
- TCPやUDPでは、ホスト上のサービスを識別するためにポート番号を使う
  - ポート番号は16bit整数
  - 代表的な用途のポートの番号は決まっている
    - 例えば http は80番、smtp は 25番など
    - well-known port と言う。リストは以下などに
      - /etc/services (UNIX系)
      - C:¥WINDOWS¥System32¥Drivers¥etc¥services (Windows)
  - 1023以下のポートを使うにはroot権限が必要

ドメイン名とIPアドレス - DNS

- 相手のIPアドレスとポート番号が分かれば、原理的には通信可能
  - 例えば、http://133.11.233.23/ にアクセスするとレポート提出サーバが見える
    - しかし面倒くさい
  - 大昔は手動でIPアドレスとドメイン名の対応表を管理していた
    - 今でもその気になればできる
- ドメイン名とIPアドレスを対応づけるサービス (DNS = Domain Name System)を使い名前解決を行う
  - 例
    - https://report.il.is.s.u-tokyo.ac.jp/とブラウザに入力すると、ブラウザがreport.il.is.s.u-tokyo.ac.jpというドメイン名をDNSサーバに問い合わせ、133.11.233.23というIPアドレスを得る

ドメイン名とIPアドレス – 使用する関数

- 名前とIPアドレスの変換に使う関数
  - getaddrinfo (3) ライブラリコール
    - IPv6等、多様なプロトコル/アドレス体系に同時に対応可能
  - gethostbyname (3) ライブラリコール
    - 古いAPIだが扱いやすい
- 特殊なアドレス
  - localhost (127.0.0.1)は「自分自身を指すアドレス」

ソケット – ソケットとは

- プロセス間通信の仕組み
  - ネットワークを通じた通信が可能
  - 多くのOSが同名の仕組みを用意している
- 通信路を使うためのインターフェース
  - ユーザプログラムとネットワークのインターフェース
- ネットワークに対してもread/writeを可能にする
  - ユーザはソケットに対してファイルと同じように読み書きができる
  - 通常のファイルとの違いはOSが吸収する
- 通信にかかわる様々なオプションを持つ
- ネットワーク (TCPなど) を使うとは限らない
  - UNIX ドメインソケットなど

ソケット – ソケットの作成方法

- ソケットの作成
  - `int socket ( int domain, int type,`
    `int protocol )`
  - 第1引数 : domain
    - プロトコル・ファミリを指定
    - 通常のTCP / IP (IPv4)ではPF_INETを指定
  - 第2引数 : type
    - UDP / IPのときはSOCK_DGRAM
    - TCP / IPのときはSOCK_STREAM
  - 第3引数 : protocol
    - UDP / IPのときはIPPROTO_UDP
    - TCP / IPのときはIPPROTO_TCP
  - これでまずソケットを用意する
    - 次に bind でIPアドレスとポート番号を指定する

サーバ側の通信準備 - bind

- bind (2) システムコール
  - `int bind ( int s, const struct sockaddr *addr, socklen_t addrlen )`
  - 普通サーバ側で使われる
    - TCPのときもUDPの時も使用する
  - 「自分のIPアドレス」、「使用するポート」の組み合わせを第1引数のソケット s に結びつける
  - 第2引数addrには実際に使用するアドレス体系に合わせた構造体のポインタを渡す
    - IPv4なら、struct sockaddr_in
    - 普通は、addr の中身を用意してから bind を呼ぶことになる
  - addr の内容によって IP アドレスとポート番号を指定する

IPv4で用いる構造体

- man 7 ip

```
struct sockaddr_in {
    sa_family_t    sin_family; /* address family: AF_INET */
    u_int16_t      sin_port;     /* port in network byte order */
    struct in_addr sin_addr;   /* internet address */
};

/* Internet address. */
struct in_addr {
    u_int32_t      s_addr;     /* address in network byte order */
};
```

- 例

```
struct sockaddr_in addr;

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

サーバ側の通信準備 - listen

- listen (2) システムコール
  - `int listen ( int s, int backlog )`
  - 接続を待つ (TCPでのみ使う)
    - サーバ側で使われる
    - 第2引数backlogで処理されていない接続要求が貯められるキューの長さを指定する

サーバ側の通信準備 - accept

- accept (2) システムコール
  - `int accept ( int s, struct sockaddr *addr, socklen_t *addrlen )`
  - ソケットに接続要求がくるまで待つ
    - TCPで使う
    - 第1引数sはsocket (2)システムコールによって生成され、bind (2) システムコールによりローカルアドレスに結び付けられ、listen (2) システムコールを経て接続を待っているソケットである
    - クライアントから接続されるとそのクライアントの通信に使う専用のソケットが作成され、そのファイルディスクリプタが返される
    - addrにはクライアントの情報が入る
      - bind (2) システムコールと同様、実際に使用するアドレス体系にあわせた構造体を使う

クライアント側の通信準備 - connect

- connect (2) システムコール
  - `int connect ( int s, struct sockaddr *addr, socklen_t addrlen )`
  - 主にコネクション型のクライアントで使われる
  - 第2引数addrで指定されたサーバに接続する
  - struct sockaddrはsocket (2) システムコール時に指定した第1引数のdomainによって異なる
    - IPv4での通信時は struct sockaddr_in を使う

通信 (1 / 2)

- connect, acceptにより通信相手が明らかとなったソケットを使用しての通信
  - read/writeシステムコールが使用できる
    - read/writeは要求したバイトすべてを処理するとは限らない (返り値で確認する)
    - これまでの問題は無視しても動くプログラムにはなったがネットワークでは注意が必要
  - 特にオプションを指定する必要があるときはsend/recvシステムコールを使用する
    - オプションの詳細はマニュアル参照

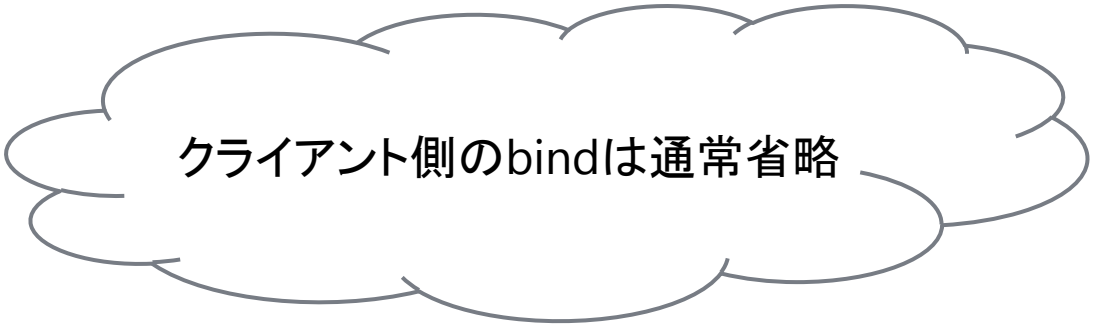
通信 (2 / 2)

- 通信相手が明らかでないソケットを使用しての通信 (UDP 使用時)
  - sendto/recvfrom システムコールを使用
  - `int sendto ( int s, const void* msg, size_t len, int flags, const struct sockaddr *to, socklen_t tolen )`
  - `int recvfrom ( int s, void* buf, size_t len, int flags, struct sockaddr *from, socklen_t *fromlen )`
- udp(7)も参照

通信の流れ - TCP

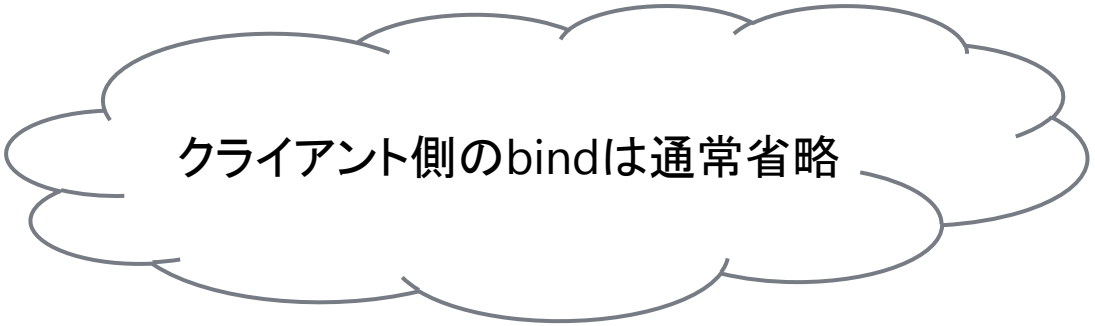
- サーバ側
  1. socket
  2. bind
  3. listen
  4. accept
  5. read/write
  6. close

- クライアント側
  1. socket
  2. (bind)
  3. connect
  4. read/write
  5. close



通信の流れ - UDP

- サーバ側
  1. socket
  2. bind
  3. recvfrom / sendto
  4. close
- クライアント側
  1. socket
  2. (bind)
  3. sendto / recvfrom
  4. close

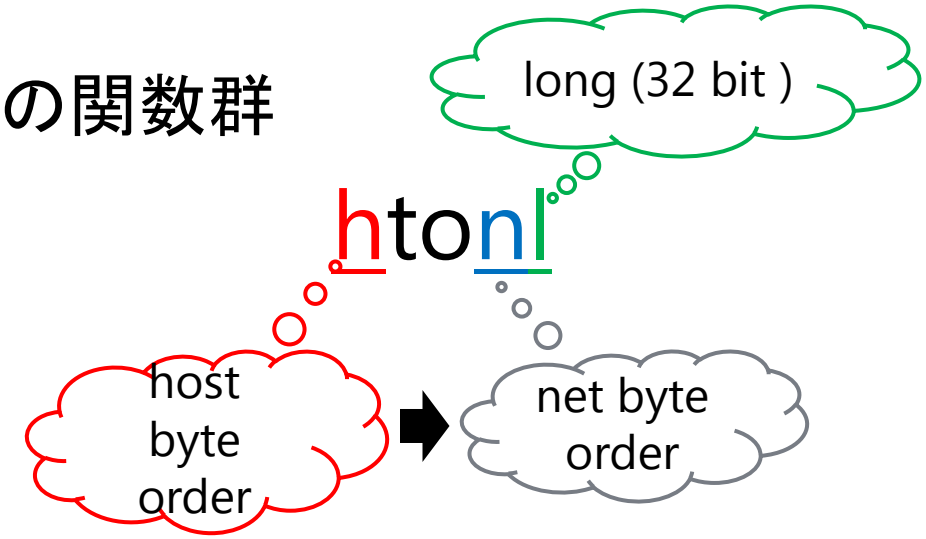


バイトオーダー – バイトオーダーとは

- ネットワーク上には様々なCPU-OSのホストが存在
  - リトルエンディアン (e.g. x86)のマシンとビッグエンディアンのマシン (e.g. PowerPC)との間でも正しく通信できなければならない
- ネットワークを流れるデータはビッグエンディアンに統一すると約束されている
  - 「ネットワークバイトオーダー」と呼ぶ
  - これに対してホスト固有のバイトオーダーを「ホストバイトオーダー」と呼ぶ

バイトオーダー – 使用する関数

- バイトオーダー変換のための関数群
  - htons (3) ライブラリコール
  - htonl (3) ライブラリコール
  - ntohs (3) ライブラリコール
  - ntohl (3) ライブラリコール



  - n : network byte order, h : host byte order
  - l : long (32bit), s: short (16bit)
  - ホストのバイトオーダーがネットワークバイトオーダーと等しい場合、これらの関数は何もしない
  - #include <netinet/in.h> が必要

複数クライアントへの対応

- 複数のクライアントからの接続に対応するためには工夫が必要
  - あるクライアントからの入力を待ってread(2)システムコールで止まっていると他のクライアントからきたデータが読めない
- 解決策
  - 各クライアントごとにスレッドまたはプロセスを割り当てる
    - プロセス版
  - poll(2)システムコール、またはselect (2)システムコールを使って、複数のソケットを同時に監視する
    - poll/select版

複数クライアントへの対応 – プロセス版

- あらかじめ適当な数のプロセスを作っておき、それらがサーバソケットをacceptする

```
int server () {
    int srv_sock;
    srv_sock = socket( ..... );
    ret = bind( srv_sock, ..... );
    ret = listen( srv_sock, ..... );
    for ( i = 0;  i < NPROCESS; ++i ) {
        if ( ( pid = fork() ) == 0 ) {
            while ( 1 ) {
                connectedsock = accept ( srv_sock, ......);
                サーバとしての処理;
            }
            exit ( 0 );
        }
    }
    wait (......);
}
```

NPROCESS分プロセスをあらかじめ準備し、複数クライアントに備える

複数ソケットに対する多重I/O

- select (2) システムコール
  - ```
    int select ( int n, fd_set *readfds,
                 fd_set *writefds,
                 fd_set *exceptfds,
                 struct timeval *timeout )
    ```
    - シグナルの集合に似ている
    - 操作はselect (2)のマニュアル参照
  - 第2引数readfdsには読み出し可能を通知してほしいソケット
  - 第3引数writefdsには書き込み可能を通知してほしいソケット
  - 第4引数exceptfdsには例外を通知してほしいソケット
- 1つのスレッドの中で複数の入出力を並行して行うことができる。似たようなシステムコールとしてpoll, epollもある
    ```

複数クライアントへの対応 – poll/select版

- poll/selectはその後のread, writeが「最低1回は」ブロックせずに返ってくることを保障する
  - poll/select は別にソケット専用では無く、例えばキー入力があったときに何かするプログラムの実装にも使われる

```
int selecttest() {
    if ( select( ... ) < 0 ) error();
    for ( i = 0; i < num; ++i ){
        if ( pfd[ i ].revent & POLLIN ) {
            m = read( pfd[ i ].fd, buf, len );
            (この後、もう一度readするとブロックするかもしれない)
        }
    }
}
```

ネットワーク性能の測定

- ネットワーク性能とは？主に二つの尺度
  - スループット (throughput) あるいは 帯域幅 (bandwidth)
    - 単位時間あたりの転送ビット数 (単位 : Mbits / sec)
    - 本来「帯域幅」という用語は別の意味だが、ネットワークの世界ではスループットを指して使われるようになってしまっている
  - レイテンシ (latency)
    - ネットワークにデータを送信してから受信するまでの遅延時間 (単位 : μsec)
    - 普通は往復で図る (RTT : round trip time)

第7回課題A (required) *

- UDP echoサーバ&クライアントを作成せよ
  - サーバ
    - 受信した文字列をそのまま送り返すだけのプログラム
  - クライアント
    - キーボードから入力された文字列を送り返答をディスプレイに表示するプログラム

課題A：仕様

- 仕様
  - プログラム名
    - それぞれudpechoserver，udpechoclient
  - コマンドラインオプション
    - udpechoserver (port number)
    - udpechoclient (host) (port number)
      - hostはIPv4アドレスで与える
  - 入出力仕様
    - サーバーはUDPの(port number)ポートを受信し、そのまま送り返す、を繰り返す
      - SIGINTで終了 (ハンドラをちゃんと書いても書かなくてもよい)
    - クライアントは、標準入力を待ち、あればUDPで (host), (port number)に送り、サーバーからの返答を標準出力に出す
      - EOFで終了

第7回課題B (required) *

- 複数クライアントに対応したTCP echoサーバを作成せよ
  - 複数プロセス/スレッド とpoll/selectの両方を作成すること

  - 複数プロセス or スレッドによるものをtcpechoserver1
  - poll or selectによるものをtcpechoserver2

課題B：仕様

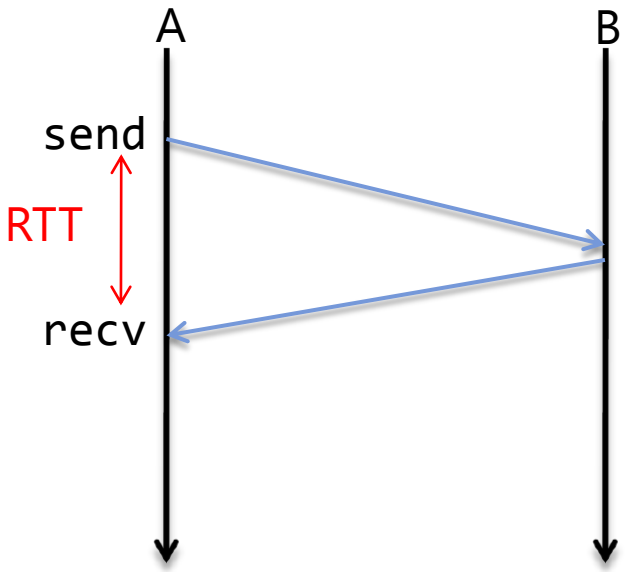
- 仕様
  - プログラム名
    - tcpechoserver1, tcpechoserver2
  - コマンドラインオプション
    - tcpechoserver1 (port number)
    - tcpechoserver2 (port number)
  - 入出力仕様
    - サーバーはTCPの(port number)ポートでコネクションを待ち、接続したクライアントから受信したデータをそのまま送り返す、を繰り返す
      - SIGINTで終了 (ハンドラをちゃんと書いても書かなくてもよい)
    - クライアントは、TCPで (host), (port number)に接続し、標準入力からのデータを待ち、それを送り、受信したデータを標準出力に出す
      - EOFで終了
  - 複数のクライアントからの接続を同時に処理できること

第7回課題C (required) *

- ネットワーク帯域幅、レイテンシを測定するツールを作れ
  - 当たり前だがサーバとクライアントが必要
  - TCPを推奨
  - 単方向通信でよい
  - 時間計測にはgettimeofday (2)システムコールを推奨
- (Optional) 実際の身の回りのマシン同士の間でネットワークの帯域幅とレイテンシを計測してみよ
  - ただし、他人に迷惑をかけない範囲で
  - 帯域幅の測定結果について考察せよ

課題Cの注意 (1)

- レイテンシ
  - 一般的には、サイズが0に限りなく近い場合に、相手に届くまでの時間をいう
    - 大きなデータの往復にかかる時間を指すこともあるにはある
  - Round Trip Timeで通常は計測する
    - 片道だけを測るのは困難
    - RTT / 2 を latency とする

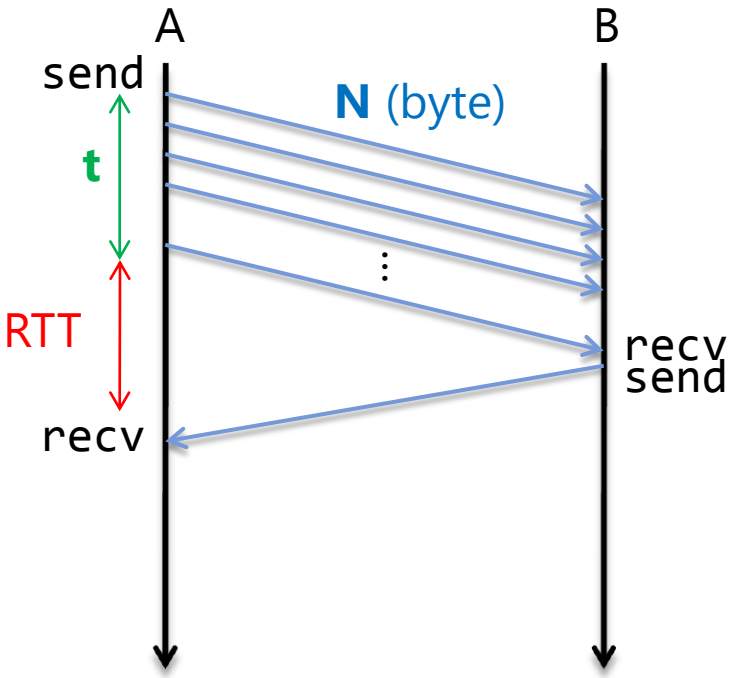


課題Cの注意 (2)

- 帯域幅
  - 計測の開始点が(ある程度)確実な点で、計測を始めること
    - (特にデータが流れてない場合の)sendの開始点
    - recvの終了点
  - **N** * 8 / **t** (bps)
  - 以下のようなコードは避けること

```
s = accept(…);
t1 = get_time();
while ( sum < SIZE ){
    sum += recv(data);
}
t2 = get_time();

printf("%f Mbps¥n",
        sum * 8 / (t2 – t1));
```



課題Cの注意 (3)

- 帯域幅を測定する際には十分な量の通信を行うこと
  - 10秒程度かかる量にすることを推奨
- 異なるマシン間で測定すること
- write (2)システムコールから戻った時点で通信が終了したと思わないこと
  - 実際はカーネル内のバッファにコピーされただけ
  - 受信側でデータをすべて受け取るまでの時間を測定すること

課題C : 仕様

- バンド幅測定のみ自動チェック
  - プログラム名
    - サーバーは、bwserver
    - クライアントは、bwclient
  - コマンドライン
    - bwserver (port number)
    - bwclient (host) (port number)
  - 入出力仕様
    - 標準入力 : なし
    - bwclientは、(host):(port number)に接続しバンド幅測定を行う
    - bwclientは測定終了後、結果を表示する
      - 次の3項目を、数値で、この順でスペースorタブ区切りで表示すること
      - 送受信したデータサイズ (byte) 、かかった時間 (s)、バンド幅 (Mbps)
    - 測定時間は一回につき20s以内とすること

第7回課題D (optional)

- 課題Cのプログラムを改良し、双方向通信の帯域幅を測定できるようにせよ
  - サーバーとクライアント、両方が同時にデータを送りあう
- さらに自由に拡張せよ
  - たとえば、UDP版を作ってみる
    - MTU, sendmsg(2) の使用などを考慮にいれる必要があるかもしれない

課題Dのヒント

- 双方向受信のためには送受信を多重化する仕組みが必要
  - read待ちでブロックしていると、その間データ受信ができなくなってしまう
- 多重化の方法
  - 複数スレッド
  - 複数プロセス
  - epoll / poll / select + nonblocking I / O

参考 - iperf

- 既存のバンド幅測定ツール
  - http://iperf.sourceforge.net/
    - Original : http://dast.nlnr.net/Projects/Iperf/
- サーバ側
  - iperf –s
- クライアント側
  - iperf –c *server_name*
- 詳細
  - iperf --help

参考書籍

- コンピュータネットワーク 第4版
  - A･S･タネンバウム 著
  - 日経BP, 2003
  - ISBN: 4822221067
- UNIXネットワークプログラミング 第2版
  - W. Richard Stevens 著, 篠田 陽一 訳
  - ピアソンエデュケーション, 2000
  - ISBN: 4894712059

第7回課題の締切

- 課題 A,B,C が必須
- 締切 : 2011年6月12日(日) 23:59 (JST)

- Webで提出
  - https://report.il.is.s.u-tokyo.ac.jp/os2011/

- レジュメ
  - https://report.il.is.s.u-tokyo.ac.jp/os2011/resume/

- 途中の場合でも**必ず**締め切りまでに提出すること
  - 提出は単位の**必要条件**

質問など

- 提出システム上の掲示板
- メール
  - os-enshu-ta@il.is.s.u-tokyo.ac.jp
- 石川研究室で直接
  - 理学部7号館5階の507号室