

# システムプログラミング ソケットプログラミング 1 (client)

芝浦工業大学 情報工学科  
菅谷みどり

1

## 今回の内容



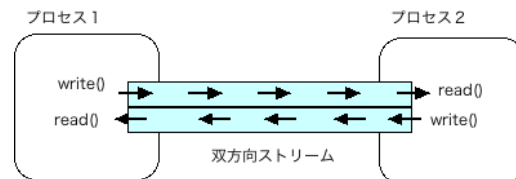
- TCP/IP のプログラムからみた時の考え方
  - プロトコルスタック
  - IPアドレス (IPv4) とポート番号
  - ストリーム
  - 通信路の開設
  - クライアント、サーバモデル
  - ホスト名からIPアドレスの変換
- telnet コマンドによる接続
  - クライアント側のプログラム、tcp\_connect()の考え方

2

## TCP/IPの基本的な考え方



- TCP/IP 通信プロトコル
  - 信頼性のある(reliable) 双方向のストリーム転送サービスを提供



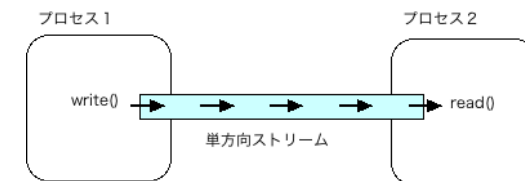
- 特徴
  - 通信路の作成
    - データの送り手、受け手の間に作られる
  - データの順番
    - 複数回にわけて送る。これらのデータの順番は入れ替わることはないが、データの区切りは保存されない
  - 高信頼
    - 送り出したデータが相手に届くようにする（途中で失われたときは再送する）

3

## (参考) Unix のパイプ



- 単方向ストリーム
  - 双方向ストリームに対して、単方向であるが、同じストリームに分類される転送サービスを提供するものである

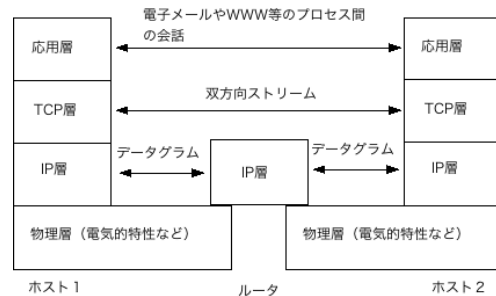


- ストリームを扱う関数
  - C言語のライブラリ関数 fopen(), fgets(), fputs() もストリームと呼ばれることがある
  - ファイル
    - もともとランダムアクセス可能で、メモリ中の配列と同じようにアクセスすることもできる
    - まるでプロセス間通信のストリームと同じように扱うことができることにことにも関係している

4

## 層 (プロトコルスタック)

- プロトコル (規約、約束事)
  - TCP/IP による通信では、TCP層 (2つ)、IP層 (2つ) の二つのプロトコルに分解される
- プロトコルスタック
  - 様々なプロトコルが決められ、全体として層をなしている様子

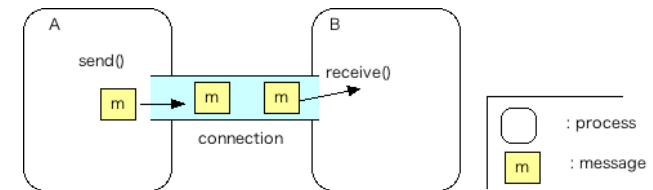


TCP/IP によるプロトコルスタック

5

## IPのデータグラム

- IP (アイピー)
  - (信頼性がない) データグラム転送サービスを提供する通信プロトコル
  - TCPは、IPという通信プロトコルを利用して実現されている。
- データグラムの特徴
  - データの送り手と受け手の間に結合 (通信路, connection) が作られない
  - 複数回に分けて送り出したデータの順番が入れ替わる可能性がある
  - 送り出したデータが途中で失われることもある

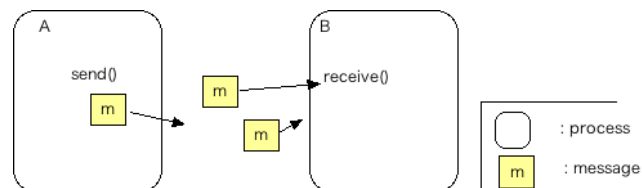


結合が作られる通信プリミティブでのメッセージの配送

6

## (補足) データグラム

- データグラム
  - パケットと呼ばれることもある
  - 結合が作られない通信で使われる、一塊のデータ
  - データ + 電報 (telegram) の類推から名前がつけられた



結合が作られない通信プリミティブでのメッセージの配送

7

## TCPでの通信

- TCPで通信するとき
  - 通信相手を識別するには、IPアドレスとポート番号が必要
- IPアドレス
  - IPのデータグラムが配送される時に使われる番号
  - IPv4(32bit) e.g. 192.168.0.1, IPv6(128bit)
- ポート番号
  - 16bit 整数, 同じホストの中で提供されている様々なサービスを区別するために使用される
    - E.g. /etc/services
  - Well-known port
    - よく使うアプリケーションでは、あらかじめどの番号を使うかが決められている
  - 特権ポート番号 (privileged port number)
    - Unix では、1024番より小さい値は特権ユーザ権限が必要

8

## 応用(アプリケーション) 層



### • TCP層の上

- この層では、ftp, ssh, sendmail などのTCP/IP を利用するプログラムの間の会話の方法が定義される

### • アプリケーション層のプロトコルの例

ポート番号	プロトコル	目的
21	FTP (File Transfer Protocol)	ファイル転送
22	SSH (Secure Shell)	暗号通信路によるログイン
23	Telnet	遠隔ログイン
25	SMTP (Simple Mail Transfer Protocol)	電子メールの転送
80	HTTP (Hyper Text Transfer Protocol)	www データ送信
110	POP (Post Office Protocol)	電子メールのアクセス
143	IMAP(Internet Message Access Protocol)	電子メールのアクセス

9

## 通信路の開設



### • 電話

TCP/IP

- 電話番号を指定 .... 呼び出し ← クライアントプロセス
- (話相手) .. 電話を取る ← サーバプロセス

### • TCP/IP の通信路

- プロセス間に形成されたストリーム通信路
  - コンピュータ間に張られた物理的な回線に似ていることから仮想的回線 (virtual circuit) といわれた
- 仮想回線の開設
  - IP アドレスとポート番号
    - ポート番号は、同じ IP アドレスを持つホスト上で動作するプロセスを区別する
- 一度仮想回線が接続された後
  - 両方のプロセスは、TCP/IP のレベルでは対称

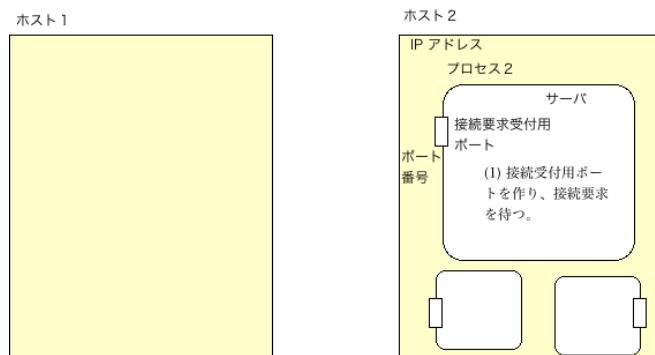
11

## TCP/IP 通信路の開設手順 (1)



### • 第一段階

- サーバ・プロセスがポート番号を指定して、接続要求受付用ポートを作る。サーバ・プロセスは、クライアント・プロセスからの接続要求を待つ



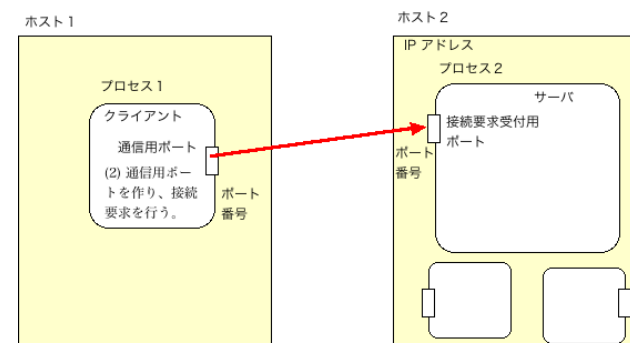
12

## TCP/IP 通信路の開設手順 (2)



### • 第二段階

- クライアント・プロセスが通信用ポートを作る。このポートを、サーバ・プロセスが動いているホストの IP アドレスと、サーバ・プロセスが作った接続要求受付用ポートのポート番号を使って、接続要求を行う



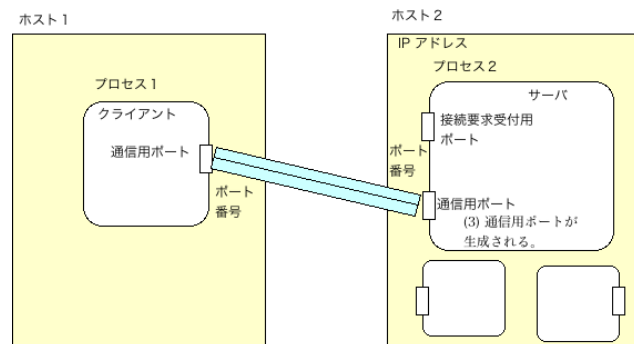
13

## TCP/IP 通信路の開設手順 (3)



### 第三段階

- 接続要求が受け付けられると、サーバ・プロセスには、新たに通信用ポートが作られる。これは、特定のクライアントとの通信のために使われる。

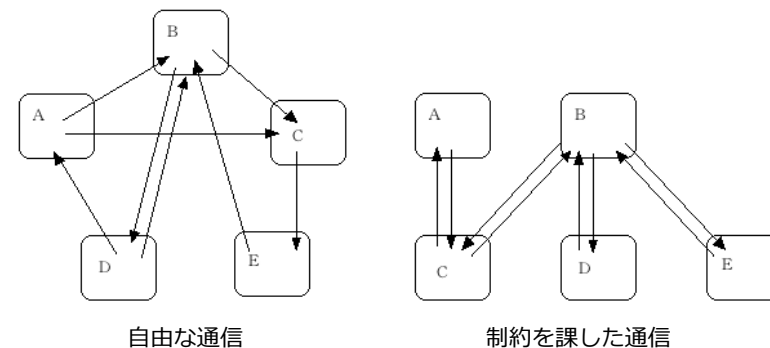


14

## 通信を行うプロセス



- どちらの通信の方が見通しが良いか？



15

## (補足) 構造化/非構造化



- (a), (b) プロセスの数とメッセージの数は一緒だが、(b) の方が見通しが良い
- プロセス
  - 何時でも、自由にメッセージを送/受信できる
    - しかし、いつでも自由に送信、受信を行うと、プログラムが複雑になる → 見通しが悪い → バグが混入する
- 通信プログラムの構造化
  - 制約を設定し、プログラムを単純化して、見通しのよいものにして、という考え方が生まれた

16

## クライアント・サーバモデル (1)



- プロセス間通信を構造化したもの
  - 構造化とは？
- プロセス間通信の構造化
  - プロセスをクライアントとサーバの二種類に分ける



通信の定義からみた、クライアントとサーバの定義

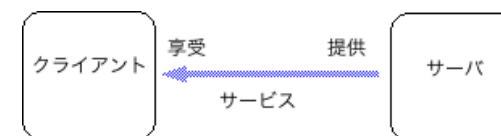
17

## クライアント・サーバモデル (2)

- ・ **クライアント、サーバプロセスの動作**
  - ・ クライアント：先に要求を送る、後で結果を受け取る
  - ・ サーバ：先に要求を受け取る、後で結果を返す
- ・ **クライアント、サーバモデルに基づくプログラムの制約**
  - ・ 次の動作を行うプロセスは存在しない
    - ・ 送信しかない
    - ・ 受信しかない
    - ・ 送信を2回して、受信を1回だけやる
    - ・ 受信したら、処理の内容によって送信したり、しなかったりする

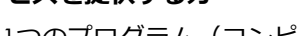
## サービス観点からの定義

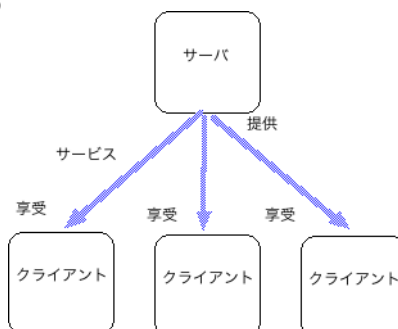
- ・ **クライアント (client)**
  - ・ サービスを受ける側、顧客
- ・ **サーバ (server)**
  - ・ サービス (service) を提供する側



## サービス授受によるクライアントとサーバ定義

## 複数のクライアントへのサービス

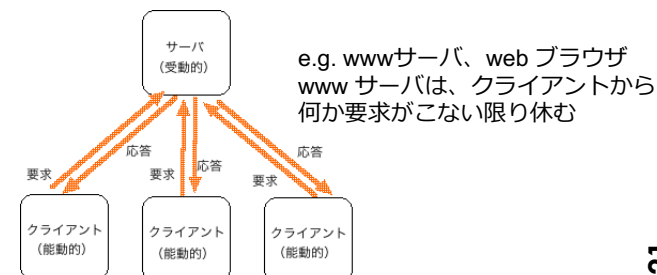
- ・ **サービスを提供する方**
    - ・ 1つのプログラム（コンピュータ）で複数の利用者 の面倒をみる。その結果 1台のサーバに複数のクライアントがつながる。
  - ・ **クライアント**
    - ・ 一人のユーザが利用するもの
  - ・ **サーバ**
    - ・ 複数人で共有するもの
- 
- ```
graph LR; Client[クライアント  
ユーザ] --- Network[ネットワーク]; Network --- Server[サーバ];
```



## 複数のクライアントによるサーバの共有

## 能動、受動からみたクライアントサーバ

- ・ 能動的(**active**)
  - ・ ほっといても自分でメッセージを発信し始める
- ・ 受動的(**passive**)、受け身
  - ・ 何か言われると答えるが、自分ではメッセージを発信し始めることはない
- ・ クライアントとサーバから作られたシステム
  - ・ クライアントが能動的でサーバは、受動的になることが多い。



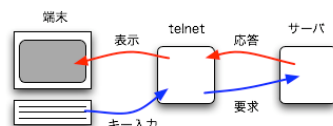
## TCP/IP 汎用クライアントプログラム



- **telnet コマンドとは**
  - 本来は、遠隔ログインのコマンドであり、通常下記のように使われる
- **遠隔ログインしてみよう**
  - `$ telnet yli.sic.shibaura-it.ac.jp`

実行例：

```
$ telnet yli.sic.shibaura-it.ac.jp
Trying 172.16.24.15...
Connected to yli.sic.shibaura-it.ac.jp.
Escape character is '^['.
```



```
Vine Linux 5.2 (Palmer)
Kernel 2.6.27-52v15 on a 4-processor x86_64
login: doly
Password:
Last login: Thu Jul 2 11:07:55 from yvpn026080.sras.sic.shibaura-it.ac.jp
```

- 送受信されるデータは、テキストのみ。  
- キーボードから打ち込んだ文字列は、サーバへ送られる(要求)。  
- サーバから送られてきた文字列(応答)は、画面へ表示される。

23

## TCP/IP 汎用クライアントプログラム



- **telnet コマンド**
  - `$ telnet yli.sic.shibaura-it.ac.jp`
    - これは、`$ telnet host1 telnet_port` の省略形
  - 第2引数の telnet は、ポート番号 (`/etc/services`)

```
telnet 23/tcp
```

- **動作手順**
  - コマンド実行 → `/etc/services` ファイルの検索
  - 与えられた記号から、ポート番号を特定
  - `host1` のアドレスを `/etc/hosts` や DNS から特定
  - IPアドレス、ポート番号を用いてTCP/IP 通信路を開設

24

## HTTPサーバにアクセスしてみよう



- **HTTP サーバのポート番号80 番にアクセスしてみよう**

```
$ telnet www.shibaura-it.ac.jp 80
Trying 54.65.4.197...
Connected to www.shibaura-it.ac.jp.
Escape character is '^['.
GET /index.html HTTP/1.0
HTTP/1.1 200 OK
```

←入力して少し待つと、サーバからHTMLファイルが転送されてくる

```
Accept-Ranges: bytes
Content-Type: text/html; charset=UTF-8
Date: Thu, 02 Jul 2015 03:33:57 GMT
ETag: "817c9-1bfaf-519cc4071497f"
Last-Modified: Wed, 01 Jul 2015 08:38:48 GMT
Server: Apache/2.2.27 (Amazon)
Content-Length: 114607
Connection: Close
<!DOCTYPE HTML>
<html lang="ja" xmlns:fb="http://ogp.me/ns/fb#">
<head>
<meta charset="UTF-8" />
```

25

## Echo サーバへのアクセスしてみよう



- **telnet クライアントプログラムから、エコーサーバにアクセスしてみよう**
- **ポート番号の確認**
  - `[doly@yli002 ~]$ egrep '^echo[ ]*/tcp' /etc/services`  
echo 7/tcp
- **telnet クライアントを利用したサーバへのアクセス**

```
[doly@yli002 ~]$ telnet 172.21.39.16 7
Trying 172.21.39.16...
Connected to 172.21.39.16.
Escape character is '^['.
hello
Hello
exit
exit
^]
telnet> quit
Connection closed.
```

← echo サービス(ポート番号7番)を提供しているサーバに telnet コマンドをクライアントとして接続

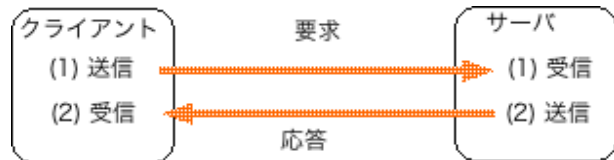
← echo サービスは、送られて来た文字列(最後に改行)をそのまま送り返す。

26

## Echo クライアントを作ろう

### ・ プロセス間通信の構造化

- ・ プロセスをクライアントとサーバの二種類に分ける



### ・ データ通信の方法

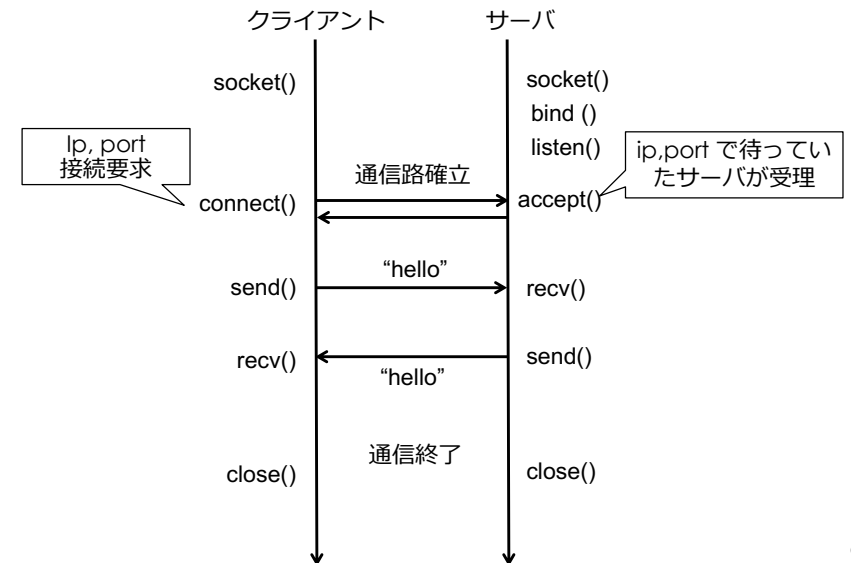
- ・ TCP/IP 通信路の開設手順(1), (2), (3) の手順

### ・ プログラミングのインターフェイス

- ・ ファイル → ソケット

28

## TCP/IP通信路: C/S システムコール



29

## クライアント/サーバの基本プログラミングパターン

connect(s); // 接続要求、accept() と対応

```

send(s,message); // 要求
receive(s,message); // 応答
send(s,message); // 要求
receive(s,message); // 応答 ... // 必要回数繰り返す

```

close(s); // 接続の切断

Client

make\_port(a); // 受付端の登録

```

while( 1 ) { s=accept(a); // 実際の受付 connect() と対応
    while( !eof(s) ) {
        receive(s,message); // 要求の受信
        send(s,message); // 応答の送信
    } close(s); // 接続の切断。
}

```

Server

30

## ソケット

### ・ ソケットインターフェイス

- ・ UNIX 上で動作するプログラムで TCP/IP機能を利用する場合に使用するインターフェイス
- ・ TCP/IP のみではなく、様々な通信プロトコルを扱うために設計された (TCP/IP に特化していない)

### ・ ソケットは、ドメイン(Protocol Family) と型で区別する

- ・ 例) socket() システムコールに与えるドメインと型

ドメイン	型	Option	プロトコル
PF_INET	SOCK_STREAM	0	TCP/IP
PF_INET	SOCK_DGRAM	0	UDP/IP
PF_INET	SOCK_RAW	?	IP, ICMP等
PF_UNIX	SOCK_STREAM	0	同一ホスト(Unix ドメイン) のストリーム
PF_UNIX	SOCK_DGRAM	0	同一ホスト(Unix ドメイン) のデータグラム

31

## ソケット API の主要なシステムコール

名前	説明
socket()	通信プロトコルに対応したソケット・オブジェクトを作成する
connect()	結合(connection)を確立させる。サーバのアドレスを固定する
listen()	サーバ側で接続要求の待ち受けを開始する
accept()	サーバ側で接続されたソケットを得る
bind()	ソケットにアドレス(名前)を付ける
getpeername()	通信相手のアドレス(名前)を得る
getsockname()	自分のアドレス(名前)を得る
send(), sendto(), sendmsg()	メッセージを送信する
recv(), recvfrom(), recvmsg()	メッセージを受信する。
shutdown()	双方向の結合を部分的に切断する
getsockopt()	オプションの現在の値を取得する
setsockopt()	オプションを設定する
select(), poll()	複数の入出力(通信を含む)を多重化する
write()	メッセージを送信する。ファイルと共通
read()	メッセージを受信する。ファイルと共通。
close()	ファイル記述子(ファイルディスクリプタ)を閉じる。他に参照しているファイル記述子がなければ、ソケット・オブジェクトを削除する。ファイルと共通。

32

## Echo クライアント/サーバ

### • Echo クライアントプログラムの動作

- コマンドラインから2つの引数(IP, port)
- 指定されたIP/ポートで動作しているサーバに接続
- キーボードから1行単位で入力 > サーバに送付
  - サーバは同じ文字列を送る
- 送られてきた文字列を受け取り、結果を画面に表示

### • Echo サーバ

- コマンドラインから2つの引数(IP, port)
- Socket を作成して、指定されたポートに bind
- Port でクライアントからの要求待ち(listen)
- 接続要求(connect) があった場合受理(accept) する
  - 通信路の確立
- データを受信し、画面に表示

33

## 課題1

### • Echo クライアントプログラムを作成しよう

- TCP/IP通信を参考にする

34

## 解説

### • 通信路の開設

- 19: socket() システムコールで、クライアント側のソケットを作成、PF\_INETと SOCK\_STREAM なので、TCPを利用することを意味する

### • バイトオーダーの変換

- 複数バイトの整数をメモリに置く場合、連続した複数番地のメモリを利用する。以下の二つの方法がある
  - ビックエンディアン：番地が小さい方に上位バイトを置く
  - リトルエンディアン：番地が小さい方に下位バイトを置く

レジスタ		0x12	34	56	78
番地		ビッグエンディアン		リトルエンディアン	
メモリ	a+0	0x12			0x78
	a+1	0x34			0x56
	a+2	0x56			0x34
	a+3	0x78			0x12

37



## 解説



### • バイトオーダーの変換

- 変換のためのライブラリ関数

```
#include <netinet/in.h>
unsigned long int htonl(unsigned long int hostlong );
unsigned short int htons(unsigned short int hostshort);
unsigned long int ntohl(unsigned long int netlong );
unsigned short int ntohs(unsigned short int netshort );
```

- n (network), h (host), s (short), l (long) を意味する
  - ファイルやネットワークに出力する際には、htonl() や htons() で標準系（ビッグエンディアン）に変換する。
  - 入力の際は、ntohl(), ntohs() でもとに戻す
- ポート番号
    - 16bit なので、htos() (host to network, short) を用いて変換
  - IP アドレス
    - inet\_aton() で、ネットワークバイトオーダーに戻す

38

## 実行例



```
$ ./client 172.21.39.16 7
==> hello
<== hello

==> aaa
<== aaa

==> exit
<== exit
```

39

## 課題2：ファイルを送る



- プログラムを書き換えて、ファイルを送信できるようにする
- `./client-file 172.21.39.16 7 src.txt`
- ポイント
  - ストリームデータをどのように送るのか？
- 提出
  - 授業中の指示に従ってください。

40

## 課題2：ファイルを送る



- ホームページからサンプルコード(echo-client.c) をダウンロード
  - ファイルのデータを送ることができるように変更する。
  - 下記のように、ファイル名を引数に取れるようにする。
  - ヒント: ファイルコピー、引数の使い方、を参考にする
- `./echo-client-file 172.21.39.16 7 src.txt`

```
==> sending: [aaaaaaaa
]
received: [aaaaaaaa
]
==> sending: [bbbbbbbb
]
received: [bbbbbbbb
]
==> sending: [ccccccc]
received: [ccccccc]
```

41