

情報工学実験 2

～ ソケットプログラミング ～

2019.07.22

山田 浩史

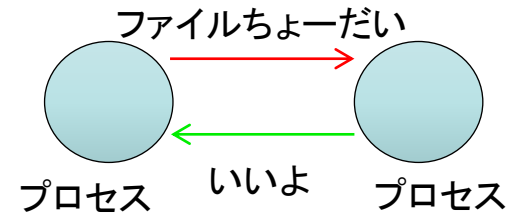
hiroshiy @ cc.tuat.ac.jp

本日のスケジュール

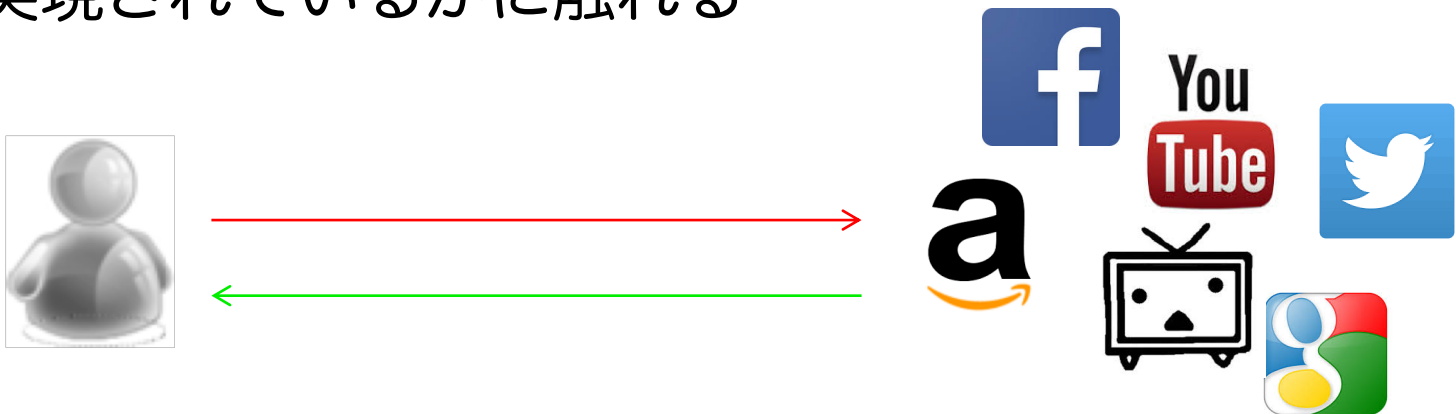
- 2 限: 課題の説明(1 部)
- 3 限: 課題の説明(2 部)
 - 説明を聞きたい学生さんは聞いて下さい
 - 説明不要の学生さんは粛々と課題に取り組んで下さい
 - 説明中は声のボリュームにお気遣いください
- 説明終了後: 課題に取り組む
 - 本日 3, 4 限中は TA さんが EDEN をうろうろしています
 - 質問がありましたら遠慮なくしてください
 - 他の時間でも質問は受けつけています
 - メールでアポをとるのが確実です

内容

- ソケットプログラミングをやってみる
 - ソケットプログラミング: プログラム同士がネットワーク通信を行うプログラミング
 - プロセス間でネットワーク越しにデータ(メッセージ) をやりとりして処理を進める

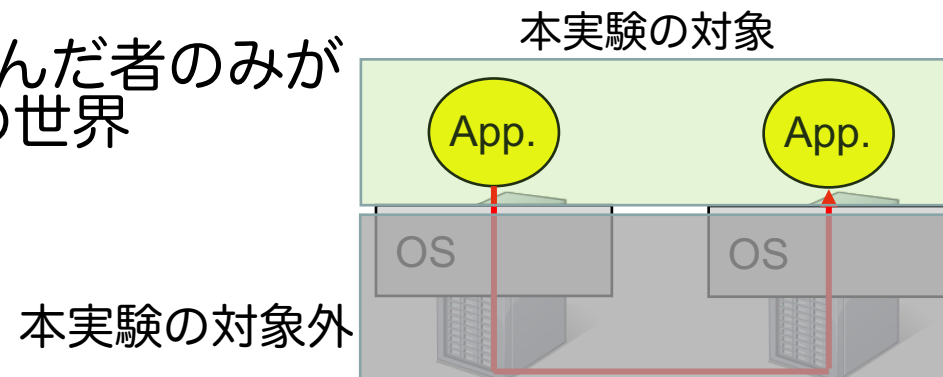


- 簡単なファイル転送システムを作成
 - 今日のネットワークサービスがどのように実現されているかに触れる



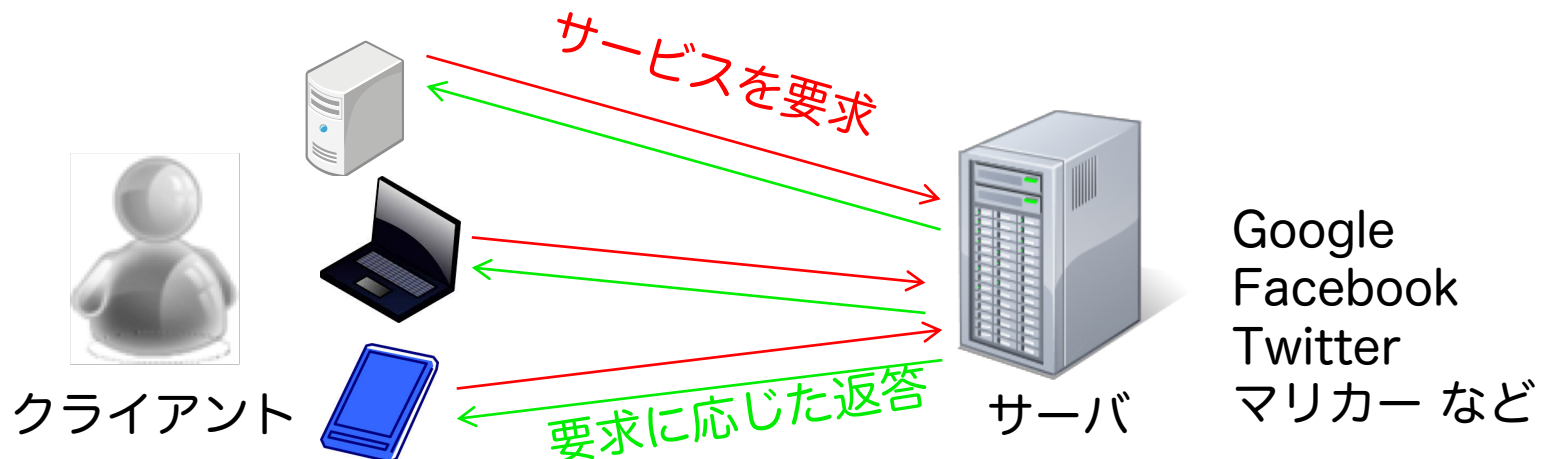
Why socket programming ?

- インターネットサービスを実現するプログラムを理解する
 - 普段使っているインターネットサービスがどのようにプログラミングされているかを知る
- ネットワークの**使い方**を理解する
 - OS によるネットワーク通信のための機能を知る
 - 通信の**仕組み**は 3 年後期:計算機ネットワークにて
- システムプログラミングとして骨のある課題
 - システムプログラミング: OS の機能を使いまくるプログラミング
 - Computer Science を学んだ者のみが許されるプログラミングの世界



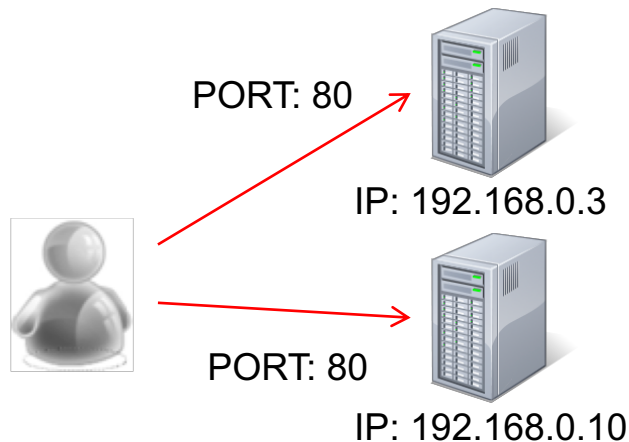
サーバ/クライアントモデル

- 役割を分けたモデル（対義語: P2P モデル）
 - サーバ: サービスを提供する側（接続される側）
E.g.) Web サーバ、メールサーバ など
 - クライアント: サービスを受ける側（接続する側）
 - E.g.) Web ブラウザ、スマホにインストールされている SNS クライアント、など
- 大半のサービスがこのモデルで実現されている

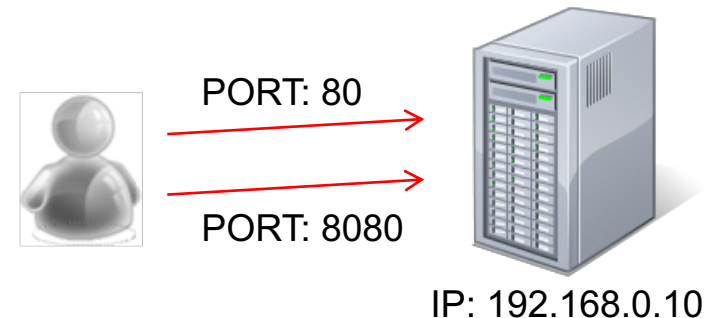


通信先を特定するには・・・

- IP アドレスとポート番号を指定してやりとりするマシン・プロセスを指定
 - IP アドレス: ネットワーク上のマシンを識別する番号
 - ポート番号: 通信するプロセスを特定する番号
 - サーバプロセスはポート番号と結び付けられている
- っことは・・・
 - サーバ: ポート番号と自身を結びつけ、リクエストを待つ
 - クライアント: IP アドレスとポート番号を指定して、リクエストを送信



異なるマシンであれば同じポートにリクエスト送信可能



同じマシン上の異なるポートに対してリクエスト送信可能

ポート番号とプロトコル

- サービスごとにポート番号は決まっている
 - E.g.) ホームページ取得(80), メール送信(25), メール受信(110)など

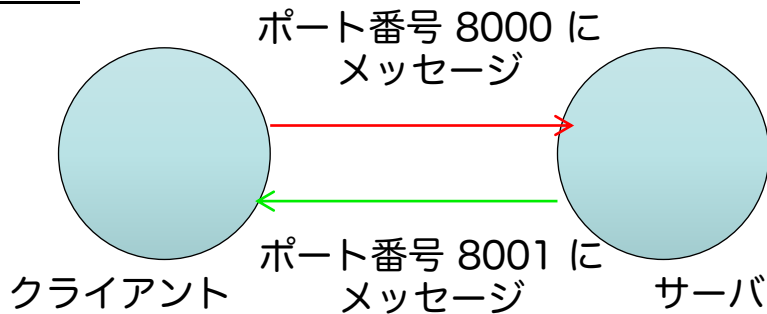
⇒ ポート番号を指定すれば目的のサービスを受けられる
- サービスごとにメッセージのやりとりは決まっている
 - やりとりの規約をプロトコル(Protocol)と呼ぶ
 - E.g.) ホームページ取得(HTTP), メール送信(SMTP), メール受信(POP3)など
 - “GET /index.html HTTP/1.1¥r¥n” というメッセージが投げられたら、“HTTP/1.1 200 OK¥r¥n . . . index.html のファイルの中身”を返すなど

⇒ 誰でもサーバ/クライアントプログラムを作成できる

通信の種類

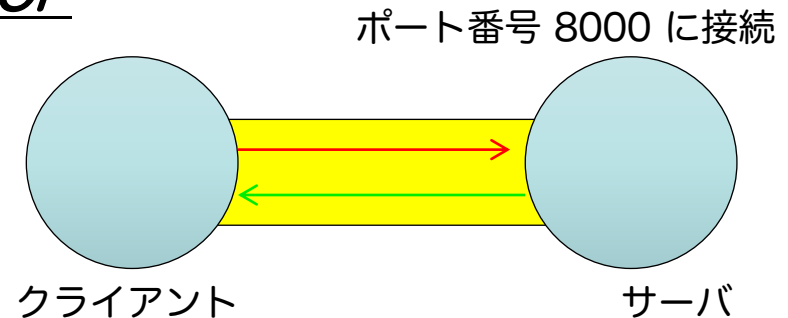
- UDP(User Datagram Protocol)
 - 接続を維持しない通信 (コネクションレス型)
 - 通信路を確立しないイメージ
 - 信頼性確保の仕組みなし
- TCP(Transfer Control Protocol)
 - 接続を維持する通信 (コネクション型)
 - 通信路を確立するイメージ
 - 信頼性を確保する仕組みあり
 - こちらを使うことが多数

UDP



通信路が確立されないため、互いにポートを指定してメッセージをやりとりする

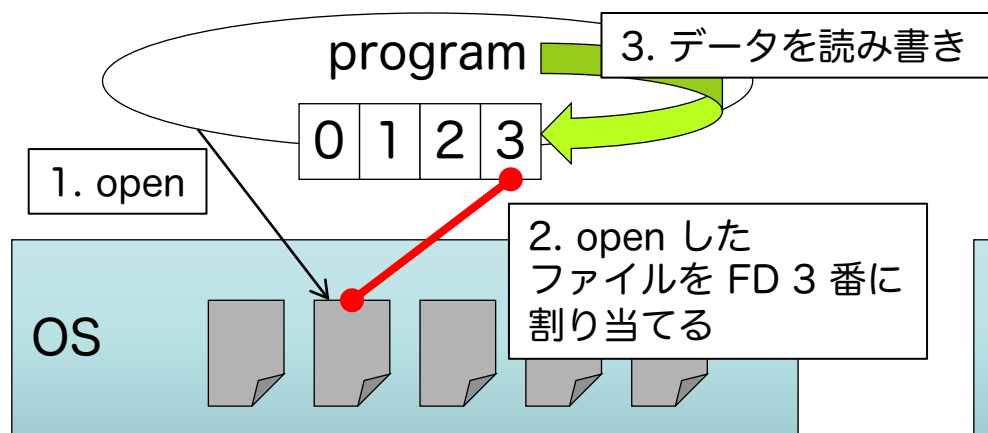
TCP



クライアントが接続すると、通信路が確立されて、そこでメッセージをやりとりする

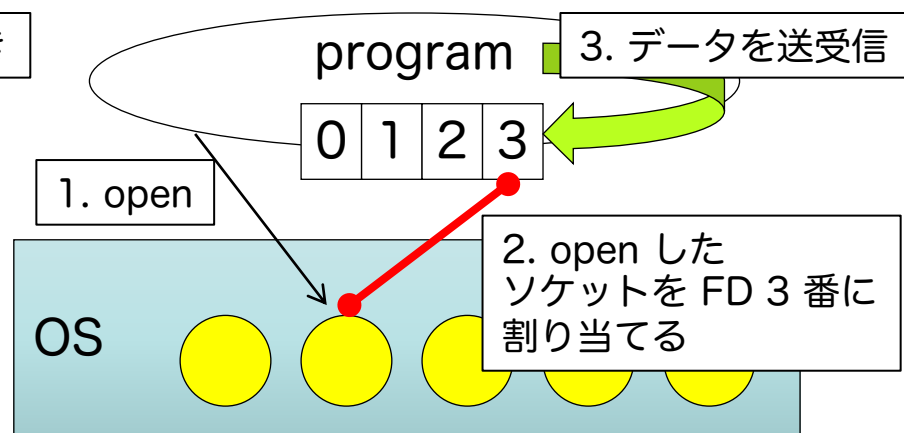
プログラム同士が通信するには

- ネットワークの抽象化である“ソケット”を使う
 - OS はネットワークをソケットとしてプログラムに見せる
 - 参考: OS はディスクをファイルとして見せる
- ⇒ プログラムはソケットに対して送受信する
 - ファイルディスクリプタを経由して行う



ディスクを
ファイルとして
抽象化

ディスク



NIC を
ソケットとして
抽象化

ネットワークカード

通信プログラムの処理の流れ

1. ソケットをオープンする
 - ソケット: OS が提供するネットワークの抽象化
 - 通信路するための窓口を作るイメージ
 - `socket()`
2. オープンしたソケットの下準備をする
 - ポート番号と結びつける, 接続したい IP アドレスを指定, 接続されたリクエストを受け付ける. など
 - `bind()`, `listen()`, `connect()`, `accept()`
3. メッセージを送受信する
 - 送信したいデータ、受け取るためのバッファを指定する
 - `send()`, `recv()`, `recvfrom()`, `sendto()`

通信にまつわるシステムコール

- *socket()* : 通信の種類を指定してソケットを開く
- *bind()* : 呼び出したプロセスをポート番号や各種設定と結びつける
- *listen()* : 指定したソケット上で接続を待つ(TCP)
- *connect()* : 指定したソケット上で、指定したマシン・ポート番号に通信路確立を試みる(TCP)
- *accept()* : *listen()* していたソケットに *connect()* されたら、それを受け入れる(TCP)
- *recv()*, *recvfrom()* : メッセージを受け取る
- *send()*, *sendto()* : メッセージを送る

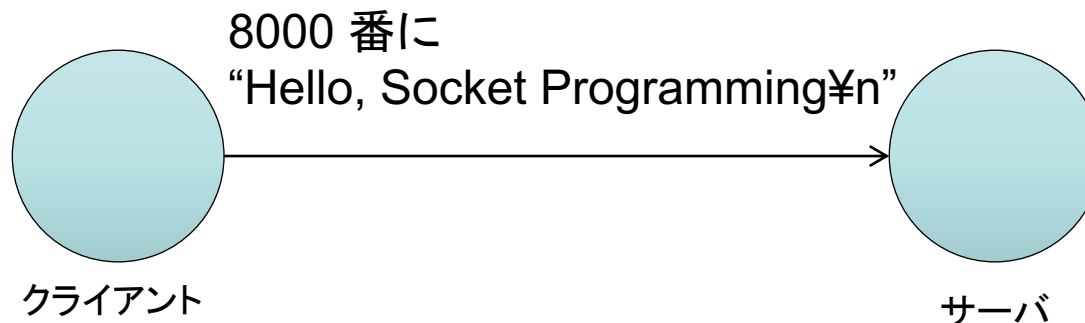
UDP を用いたネットワーク プログラミング(サンプルコード)

クライアント

- UDP でソケットをオープン(socket())
- マシンの IP・ポート番号を指定してメッセージを送信(sendto())

サーバ

- UDP でソケットをオープン(socket())
- ポート番号を結びつける(bind())
- メッセージを待つ(recvfrom())
 - メッセージが来たらそれを出力



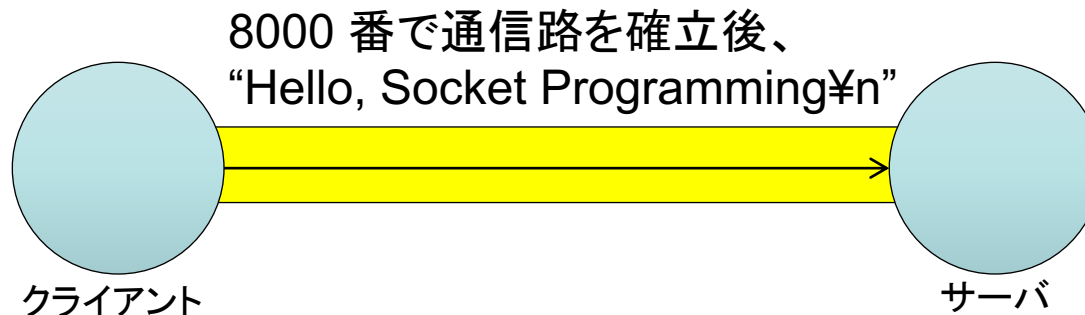
TCP を用いたネットワーク プログラミング(サンプルコード)

クライアント

- TCP でソケットをオープン(socket())
- マシンの IP・ポート番号を指定して通信路確立を試みる(connect())
- メッセージを送信(send())

サーバ

- TCP でソケットをオープン(socket())
- ポート番号を結びつける(bind())
- 接続を待つ(listen())
- 接続が来たら受け付ける(accept())
- メッセージを受信(recv())
 - メッセージがきたら出力



課題 1-3 (ヒント)

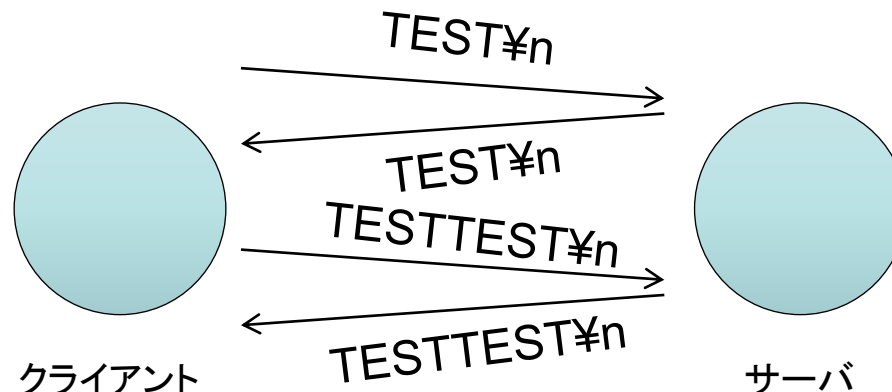
- 以下を無限に繰り返せばよい

クライアント

- 標準入力を受け取る
- 受け取ったデータをサーバに送信
- サーバからデータを受け取り、それを出力

サーバ

- クライアントからデータを受け取り、それを出力
- 受け取ったデータをそのままクライアントに送信



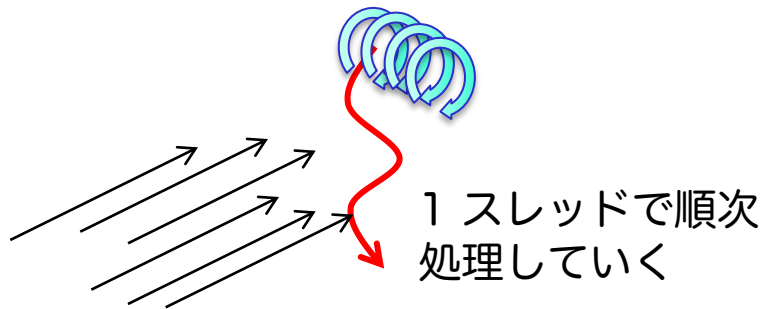
リクエストを並行処理するには

- 今までのコードは 1 クライアントを想定
 - 1 つのリクエストを受ける ⇒ 処理する ⇒ 終了
 - 実際のサービスは複数ユーザからのリクエストを裁かないと話にならない
- 並行処理の実現方法:
 - select() による I/O の多重化
 - ファイルディスクリプタに I/O があったかを監視する
 - 監視するディスクリプタ番号を受け取り, 変化が生じたら返る
 - メッセージ到着などを検知し, 1 つずつリクエストを処理
 - スレッドによる I/O 多重化
 - リクエストがきたらスレッドを生成し, その後の処理を任せる

両者のイメージ

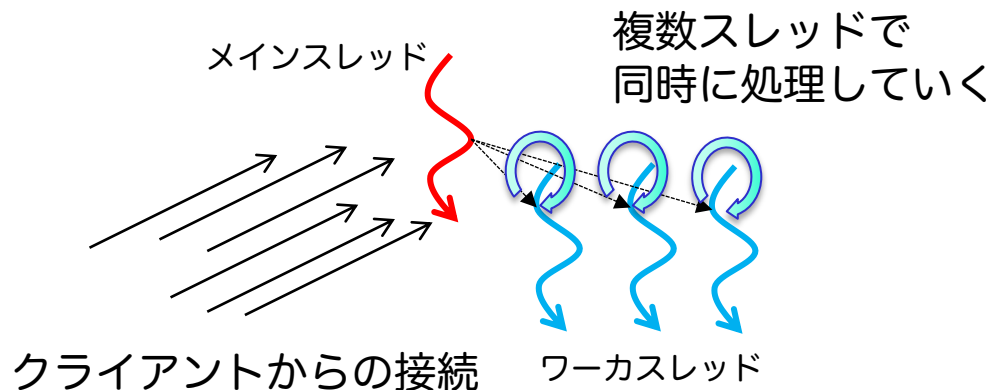
- `select()`: 単一スレッドが順にリクエストを捌く
 - リクエストをひとつずつ処理していく
- 複数スレッド: 文字通り複数スレッドが同時にリクエストを捌く
 - メインスレッドがリクエストを受け取ったら、ワーカスレッドを生成して処理をまかす

select() による並行処理



クライアントからの接続

スレッドによる並行処理



クライアントからの接続

select() を用いたサーバ (サンプルコード)

- ソケットをオープンする(socket())
- ポート番号を自身に結びつける(bind())
- ポート番号上で接続を待つ(listen())
- select() 用のデータ構造を初期化
 - FD_ZERO(&prev_fds) : prev_fds を 0 で初期化
 - FD_SET(fd1, &prev_fds) : fd1 を監視するように設定する
 - fd1: 接続を待つためのソケット. connect() されたら気づきたい
- select() を呼んで, fd1 に I/O があったかを確認
 - FD_ISSET(fd1, &fds): fd1 に I/O があれば 1 を返す
 - accept()して, その新しくできたディスクリプタを監視対象にする
- 新たに監視対象としたディスクリプタに I/O 処理があったらそれを表示, その後 close()
 - FD_CLR(fd2, &prev_fds): fd2 を監視対象からはずす

スレッドを用いたサーバ

- ソケットをオープンする(socket())
- ポート番号を自身に結びつける(bind())
- ポート番号上で接続を待つ(listen())
- 接続を受け付ける(accept())
- スレッドを生成する(pthread_create())
 - 生成されたスレッドはメッセージを受け取る(recv_and_resp())
 - 接続してきたクライアントとやりとりする
ディスクリプタ番号を渡す
 - 生成したスレッドは accept() で待つ

課題 2-2 (ヒント)

- サーバに以下の識別機能を加えればよさそう
 - `bind()`したファイルディスクリプタに I/O があった
⇒ 接続を受理する
 - 接続を受理したファイルディスクリプタに I/O があった
⇒ echo サーバとしての処理を施す
- サンプルコードは `select()`を使っているものの、1 クライアントにしか対応できていない
 - `FD_ISSET()` が `fd1`, `fd2` の条件しかないので、複数クライアント (= 複数ディスクリプタ) に対応できるよう工夫する必要がある

課題 3-1 について（ヒント）

- サーバに以下の機能があればできそう
 - クライアントからメッセージを受け取る
 - 文字列を解析する
 - GET という文字列があって、そのあとのファイル名を抽出
 - ファイルをオープンする
 - ファイルがなければ “NOT FOUND¥n” を返す
 - あれば、サイズを確認して、テキストの記述に従ってメッセージをクライアントに返す
 - 複数クライアントのリクエスト処理は `select()` を使えばできそう

レポートについて

- 問の解答だけを書いてください
 - 問題文は書かなくてよいです
- プログラムは各解答部分に書いてください
 - 付録としてまとめないこと
 - プログラムの設計方針・拡張方針を明記した上でプログラム・実行結果を掲載ください
- 原理は不要です
- 無駄な考察は不要です
- PDF 形式で提出してください