実験2（ソフトウェア） 資料

目 次

1 概要

1.1 この実験の目標

この実験ではHTTPクライアント（=極めて簡単な「WWWブラウザ」）とHTTPサーバの作成を通して，ネットワーク・プログラミングの初歩を学ぶ．

他のプログラムとの通信を行なうためには，決められた「ルール」（プロトコル，と呼ばれる）に沿った処理を行なう必要がある．すなわち，他のプログラムと「会話（通信)」を行なうためには，自分の好き勝手に設計することはできず，予め決められた「会話のルール」に正しく従っているプログラムを作成しなければならない．このような「会話のルール」はプロトコルと呼ばれる．HTTPはプロトコルの一つである．この実験では，HTTPに準拠したプログラム作成によって,「決められた仕様を満たすプログラムの作成」を体験し理解することを目標とする．

前半にC言語でHTTPクライアントとHTTPサーバを作成し，後半に同機能のクライアント・サーバをJavaで作成する．HTTPのプロトコルに正しく従ったプログラムを作成することにより，異なる環境や言語で作成された（さらには，他の人が作成した）サーバ・クライアント間で通信が行なえることを確認する．

1.2 注意

資料に関する注意 実際に課題を始める前に，**必ずこの資料の当該部分までを通して読んでおくこと**．

この予習は，**実験の時間外に行なう**ことを想定している．実験時間になって初めて資料を読み始めていると，間違いなく時間が足りなくなるので注意すること．実験時間は「実際にプログラムを書く時間」だと考えること．

Javaの学習について重要な注意 本実験では，Javaの基礎に関しては細かい説明は行なわないので，予め指定の教科書[1]で自習しておくこと．自習の進め方については，本資料5章を参照せよ．

このJavaの自習も，**実験の時間外に行なう**ことを想定している．

成績の付け方 報告書（レポート），総合デモ，出席に基いて行う．必修の課題は5つあり，この全てを完了することが必要である．

- 報告書：以下の3つの報告書を提出する．
  1. 報告書1：課題2（課題1の内容を含む）
  2. 報告書2：課題4（課題3の内容を含む）
  3. 報告書3：課題5

[1]「すべての人のための Java プログラミング (第二版)」(立木秀樹，有賀妙子) http://www.i.h.kyoto-u.ac.jp/~tsuiki/java-everyone/

それぞれの課題の内容をレポートにまとめ，作成したプログラムとともに提出する．付録 A に従うこと．

- 総合デモ：実験で作成したプログラムの動作を実際に確認する．付録 A を参照せよ．
- 出席：本実験は出席が義務付けられている．正当な理由なく一定時間以上欠席・遅刻すると単位は与えられない．

発展課題については，加点対象とする．本資料に記載されている内容に関わらず，自由に課題を設定して実施して構わない．発展課題を実施した場合は，上記報告書の一部として提出しても，別の報告書として提出しても構わない．いずれの場合も，どのような内容の課題を行なったのかを明記すること．

1.3 サンプルのダウンロード

以下の URL にサンプルプログラムのソースがあるので予めダウンロードしておくこと．

http://www.fos.kuis.kyoto-u.ac.jp/le2soft/sample.tar.gz

展開の方法と内容は，以下のとおり．

```
% tar zxf sample.tar.gz
% ls -R sample
c/     c-cs/     java/     java-cs/

sample/c:
sample1.c       sample2.c

sample/c-cs:
client.c        server.c

sample/java:
sample1.java    sample2.java     sample3.java     sample4.java

sample/java-cs:
client.java     server.java
```

2　インターネットと TCP/IP

さらに詳しい解説は，`http://www.fos.kuis.kyoto-u.ac.jp/le2soft/basic.html` や，参考書を参照のこと．

2.1　IP と IP アドレス

IP (Internet Protocol) とはネットワーキングのためのプロトコル (通信規約、通信手順) であり，インターネットは IP に従って運用されている．インターネットに接続されたローカルネットワークの多く（企業内ネットワークやこの学生実験を行なっている実験室のネットワーク）も IP を用いて運用されており，このようなネットワークはイントラネットと呼ばれる．

IP では，ネットワークの構成要素を識別するために**IP アドレス**を用いる．IP アドレスは，ネットワークに接続されている機器に割り当てられた電話番号のようなものだと思えばよい．IP アドレス (IPv4) は 32bit の数であり、x.x.x.x のように，8bit ずつドットで区切って 10 進数で表記することが多い．例えば，130.54.23.2 は IP アドレスであり，32bit の 2 進数で表記するならば，

10000010 00110110 00010111 00000010

である．

インターネットでは，世界で一意に識別される番号が必要であり，そのような IP アドレスを**グローバル IP アドレス**と呼ぶ．その他，前述のイントラネットのためのアドレスを，**プライベート IP アドレス**と呼ぶ．プライベート IP アドレスはその範囲が，

10.0.0.0 – 10.255.255.255

172.16.0.0 – 172.31.255.255

192.168.0.0 – 192.168.255.255

と定められている．

2.2　DNS

通信相手をいちいち IP アドレスで指定していたのでは入力するのも覚えるのも大変である．そこで，より人間に分かりやすい表現として，アルファベットと数字と記号「-」「.」で記述する**ドメイン名** (domain name) という表記方法がある．

例えば，`www.kuis.kyoto-u.ac.jp` や `www.yahoo.co.jp` がドメイン名[2] である．ドメイン名も IP アドレスと同じく世界で一意に識別されており，**DNS** (Domain Name System) によって，各ドメイン名には IP アドレスが割り振られている．実際，IP ネットワーク上の

[2] より正確には，これらはドメイン名とホスト名の組合せである．

アプリケーションでは，ドメイン名が指定される度に，DNS によって対応する IP アドレスを求め，その IP アドレスを利用してネットワーク上の機器と通信を行なう．

演習 1 端末のネットワーク関係の設定を表示する `ifconfig` コマンドを利用して，自分の目の前の端末の IP アドレスを調べよ．

```
% ifconfig
```

または，

```
% /sbin/ifconfig
```

「`inet` アドレス」と表示されているのが IP アドレスである．

次に，DNS を用いてドメイン名から IP アドレスを求めるための `nslookup` コマンド（または `dig` コマンド）で，ドメイン名 `www.kuis.kyoto-u.ac.jp` などに割り当てられている IP アドレスを確認せよ．

```
% nslookup www.kyoto-u.ac.jp
% nslookup www.kuis.kyoto-u.ac.jp
% dig www.kyoto-u.ac.jp
% dig www.kuis.kyoto-u.ac.jp
```

2.3 TCP とポート番号

IP では，パケットという単位（最大長 1500 バイト程度）で転送するという単純な仕組しかなく，IP だけではより大きなサイズのファイルを転送することができない．また，IP では，ルータが処理できない場合は適宜パケットを破棄してもよいことになっているため，パケットが確実に相手に届くことも保証されていない．

そこで，インターネット上のアプリケーションの多くは，**TCP** (Transmission Control Protocol) に従って通信を行なっている．TCP は，大きなデータのパケットへの分割を行なったり，経路上のルータで破棄されたパケットを再送することによって信頼性の高い通信を可能とする．

TCP では，接続すべきアプリケーションを識別するために**ポート番号**が利用される．これによって，同一の機器が複数の TCP アプリケーション（http サーバとメールサーバなど）を提供している場合でも，それぞれのアプリケーションに対応するポート番号を指定することによって正しいサービスを受けることができる．

広く用いられるアプリケーションについては，予め利用するポート番号が決められていることが多い（必ずしもそのポートを利用しなければいけない，ということではない）．例えば，HTTP は 80 番を用いることになっているので，あるホストの HTTP サーバに要求を送る場合には，そのホストのポート番号 80 に対して TCP で接続することになる．

2.4 telnet による TCP 接続

telnet コマンドは，telnet プロトコルを利用して遠隔の PC にログインする機能だけでなく，単純に標準入出力と TCP 接続とを直結させるという機能がある．例えば，

```
% telnet www.kuis.kyoto-u.ac.jp 80
```

とすることで，サーバ www.kuis.kyoto-u.ac.jp の 80 番ポート，すなわち http サーバに TCP 接続をし，TCP 接続と標準入出力を直結させることを意味する．その後，

```
GET / HTTP/1.1[改行]
host: www.kuis.kyoto-u.ac.jp[改行]
[改行]
```

と入力することでこのサーバのトップページを入手することができる．一行目は,「HTTP/1.1 のプロトコルに従って，パス/で指定されるファイルを取得（GET）せよ」という意味である.（なお，telnet のエスケープシーケンスは CTRL キー ＋ 「]」キーである.）

演習 2 上の例と同様にして，HTTP サーバ www.fos.kuis.kyoto-u.ac.jp の 80 番ポートに telnet で接続し，各ページが入手できることを確認せよ.

```
GET / HTTP/1.1
GET /le2soft/index.html HTTP/1.1
GET /le2soft/faq.html HTTP/1.1
```

などを試してみよ（それぞれ「host: www.fos.kuis.kyoto-u.ac.jp」の行を忘れないように）．表面には登場しないが，ブラウザ等を利用している時も裏側ではこのような一定の約束（=プロトコル）にもとづいたメッセージのやり取りが行なわれている.

2.5 WWW と URL（課題 1）

World Wide Web (WWW) の直訳は「世界に張り巡らされた蜘の巣」である．インターネット上に存在するテキスト，画像，音声，映像など様々なファイルを結びつけたものを閲覧，取得する仕組みである．**Web ブラウザ**は，WWW を利用するための閲覧ソフトであり，Firefox や Google Chrome，Internet Explorer，Safari などがある．WWW においてファイル間の結び付き（リンク）が記述できるファイル記述方式として広く利用されているのが，**Hyper Text Mark-up Language** (HTML) である．**Hyper Text Transfer Protocol** (HTTP) は，クライアントが サーバから HTML などのファイルなどを取得するためのプロトコルとして開発された．現在では，それ以外にもクライアントから各種情報をサーバに送るなど，様々な機能が追加されている.

インターネット上のファイルや情報を一意に特定するために，**Uniform Resource Locator** (URL) と呼ばれる記述方式が用いられている．URL は

<スキーム>://(<ユーザ>:<パスワード>@)<ホスト>(:<ポート番号>)/<パス>

という形式であり，例えば本実験のホームページは，

http://www.fos.kuis.kyoto-u.ac.jp/le2soft/

という URL により一意に特定される．この例では，

- スキーム = `http`
- ホスト = `www.fos.kuis.kyoto-u.ac.jp`
- パス = `le2soft/`

である．また，先に述べたように，HTTP では通常 80 番ポートが利用されるため，ポート番号が省略された場合は 80 番を指定したことになる．

演習 3 次の課題では，端末から文字列として入力された URL から，ホスト名などの情報を抽出する URL 解析器を作成する．この際，文字列をトークンに区切る `strtok` 関数を利用する．

関数 `char *strtok(char *s1, char *s2)` は，分解される文字列 `s1` と区切り文字を含む文字列 `s2` を受け取り，`s1` の先頭から最初の区切り文字（の手前）までの文字列（トークン）へのポインタを返す（最後まで区切り文字がなければ文字列全体を，返すべき文字列がなければ `NULL` を返す）．同じ文字列に対して 2 番目，3 番目... のトークンを分解する場合，2 回目以降は第一引数を `NULL` にして呼び出す．`strtok` 関数は，区切り文字を終端文字（`'\0'`）に置き換えていくので，`s1` の内容は実行前後で異なる事に注意．

ダウンロードしたサンプル[3] のうち，`sample/c` ディレクトリにある `sample1.c` は，`strtok` 関数を利用した例である．入力を受け付け，入力された文字列を「空白文字とカンマと改行文字」を区切り文字と見做してトークンに分割し，最初から 3 つのトークンを表示して終了する．

```
% gcc -o sample1 sample1.c
% ./sample1
foo, bar, and hoge          （標準入力）
1st = foo; 2nd = bar; 3rd = and
```

このプログラムを実行し，動作を確認せよ．

また，付録 D 章を参考にして，デバッガ `gdb` によって `sample1.c` における `strtok` 関数の動作を確認せよ．

さらなる詳細は man ページや他の解説を参照すること．

[3] サンプルプログラムのダウンロードについては本資料 1.3 章を参照

課題 1：URL 解析器の作成

端末から URL の入力を受け付け，その URL で指定されているホスト名，ポート番号，パスを表示するプログラムを C 言語で作成せよ．ただし，この実験では（以降の課題を含めて）以下を仮定してよい．

- スキームは常に `http` が指定される
- 「<ユーザ>:<パスワード>@」の部分は常に省略される
- ポート番号は，省略された場合は 80 が指定されたものと見做す

以下に動作例を示す．

```
Input URL:
http://www.fos.kuis.kyoto-u.ac.jp:8080/le2soft/   （標準入力）

host = www.fos.kuis.kyoto-u.ac.jp
port = 8080
path = le2soft/
```

3　クライアント・サーバプログラミング

3.1　ソケットインタフェース

UNIX 系の OS では，ネットワークプログラミングを行なうために**ソケットインタフェース** (socket interface) が用意されている．アプリケーション側では，ソケットを利用することによってファイルの読み書きを行なうのと同様にネットワークでのデータ転送が行なえるようになり，実際に送受信されているデータの構造や通信手段などを意識する必要はない[4]．

今回のような**クライアントサーバ**によるアプリケーションでは，以下のような流れで処理が行なわれる（カッコ内は各々を行なうシステムコール）．

- サーバ
  1. ソケットを開く (`socket()`)
  2. 接続を受け付ける IP アドレス・ポート番号をソケットに対応づける (`bind()`)
  3. クライアントからの接続待ち状態にする (`listen()`)
  4. クライアントからの接続を受け付ける (`accept()`)

[4]OSI 参照モデルのようなネットワークの階層構造を考えるならば，ソケットは HTTP や SMTP などのアプリケーション層と TCP などのトランスポート層の仲介をするものと思える．

5. 通信を行なう (read(), write(), send(), recv())
6. ソケットを閉じる (close())

- クライアント

1. ソケットを開く (socket())
2. サーバに接続する (connect())
3. 通信を行なう (read(), write(), send(), recv())
4. ソケットを閉じる (close())

3.2 サンプルプログラム

まずは，サンプルプログラムを動かしてみよう．

演習 4 ダウンロードしたサンプル[5]のうち，sample/c-cs ディレクトリにある server.c と client.c を自分の作業ディレクトリ[6]にコピーし，コンパイル，実行して動作を確かめよ．実行の際は，端末ウィンドウを二つ開き，それぞれで server，client を実行するとよい．

```
% gcc -o server server.c
% gcc -o client client.c
% ./server 30001      （接続を受け付けるポート番号を指定して実行）
```

別の端末ウィンドウで以下を実行する．

```
% ./client localhost 30001   （接続先のホストとポート番号を指定して実行）
```

localhost は自分自身を指す特別なホスト名である．ifconfig で確認した IP アドレスを指定してもよい[7] クライアントからの接続が成功すると，通信が行なわれ，サーバ側に Request from client，クライアント側に 1 2 3 が表示されるはずである．

さらに，他のホストの IP アドレスを調べ，自分のホストと他ホストとの間の通信を試してみよ[8]．

演習 5 server.c と client.c のコンパイルを make コマンドを用いて行なえるよう，Makefile を作成せよ．make と Makefile については，付録 C 章を参照せよ．

以下では，サンプルプログラムの概要を解説しているが，説明されていない関数は man コマンドを用いて調べるなどして動作を理解しておくこと．

[5]サンプルプログラムのダウンロードについては本資料 1.3 章を参照

[6]この後の課題はこれらのサンプルコードを変更していくことになる．**課題毎のディレクトリを作成して，各課題のソースファイルを区別できるようにしておくことを強く勧める**．

[7]client.c 中の gethostbyname 関数は，引数として IP アドレス（例えば 192.168.1.1 という形の文字列）を取ることもできるので，client の引数として IP アドレスをそのまま渡しても正常に動作する．

[8]現在，演習室の Ubuntu 端末間では，30001-30100 番ポートのみ通信が可能になっているので，その範囲のポートを利用すること

3.3 sockaddr_in 構造体

接続先の IP アドレスやポート番号の情報を保持するために，sockaddr_in 構造体が用意されており，各ソケットは，bind システムコールによって sockaddr_in 構造体のデータと関連づけられる．sockaddr_in 構造体は次のように定義されている．

/usr/include/netinet/in.h:

```
struct in_addr {
   u_int32_t s_addr;
};

struct sockaddr_in {
   u_char  sin_len;      （このメンバは古い OS では存在しない）
   u_char  sin_family;     （アドレスファミリ．今回は AF_INET で固定）
   u_short sin_port;     （ポート番号）
   struct  in_addr sin_addr;     （IP アドレス）
   char    sin_zero[8];     （無視してもよい.「詰め物」のようなもの）
};
```

ポート番号や IP アドレスはネットワークバイトオーダー (big endian) になっていないといけない．このため，整数をネットワークバイトオーダーに変換する htons 関数を用いる．

3.4 サンプルプログラム server.c の概要

サンプルプログラム server.c の概要を説明する．

まず，main 関数のパラメータによって，コマンドラインからの引数の個数 argc と，引数の配列 argv（文字列配列）を参照できるようにしておく．このサンプルでは，一つ目の引数 (argv[1]) でポート番号を取る．

以下の部分で接続を受けるサーバの情報を設定し，bind システムコールを用いてソケット s に関連づけている．

server.c:

```
bzero(&sin, sizeof(sin));      （sin を 0 で初期化）
sin.sin_family = AF_INET;      （AF_INET で固定）
sin.sin_port = htons(port);     （ポート番号を設定）
sin.sin_addr.s_addr = INADDR_ANY;
```

一般に，サーバは複数の IP アドレスを持つことができるが，s_addr を INADDR_ANY とすることにより，どのアドレスに対する接続も受け付けるようにしておく．

その後，listen システムコールによりクライアントからの接続待ち状態になり，接続があれば，accept システムコールによってこれを受け付ける．

データの送受信は

- send システムコールを用いてデータを送信
- fdopen 関数を用いて FILE 構造体 fp を得，ファイルを操作するのと同様に fgets 関数などによってデータを受信

している [9].

3.5 サンプルプログラム client.c の概要

サンプルプログラム client.c の概要を説明する．

このクライアントプログラムは，一つ目の引数 (argv[1]) に接続先のホスト名，二つ目の引数 (argv[2]) にポート番号を取る．ホスト名は，gethostbyname 関数によって DNS で IP アドレスを取得し，

```
memcpy(&sin.sin_addr, hp->h_addr, hp->h_length);
```

として，sockaddr_in 構造体 hp の IP アドレス情報を設定している．

作成したソケットは，connect システムコールによって接続し，接続後はサーバと同様にしてデータの送受信を行なっている．

4 HTTP クライアントとサーバの作成

次の課題 では HTTP1.1 にもとづくクライアントとサーバを作成する．HTTP1.1 の詳しいプロトコル仕様については参考書「HTTP 詳説」を参照のこと．完全な仕様は RFC2616[10] に記述されている．ここでは全てのプロトコルを記述していないので，これらの資料を参考にすること．

本実験で実装するのは HTTP 1.1 のサブセットであり，クライアントがサーバに TCP コネクションを確立し，クライアントからリクエスト（下記文法の Request）を送り，それに対してサーバがレスポンス（下記文法の Response）を返すものとする．本実験で対象とする HTTP 1.1 の文法を以下の BNF で定義する．

```
HTTP-message    = Request | Response

;; クライアントからのリクエスト
Request         = Request-Line
                  request-header CRLF
```

[9] send(ns, str, len, flag) の len は実際に送信する文字列長であり，str の終端文字までの長さが len よりも短い場合，終端文字も含めて長さ len だけのデータを送信する．fgets(buf, len, fp) の len は，取得する文字列の「最大長」であり，もしそのサイズに達するよりも先に改行文字などの終端文字が来た場合は，その終端文字までの文字列を buf に格納する．

[10] http://www.ietf.org/rfc/rfc2616.txt（日本語訳は http://www.studyinghttp.net/cgi-bin/rfc.cgi?2616）．ただし BNF の表記は RFC2068 (http://www.ring.gr.jp/pub/doc/RFC/rfc2068.txt) に準じている．

```
Request-Line   = "GET" SP abs_path SP "HTTP/1.1" CRLF

request-header = "Host" ":" host [ ":" port ]

;; サーバからのレスポンス
Response       = Status-Line
                 *entity-header CRLF
                 [ message-body ]

Status-Line    = "HTTP/1.1" SP Status-Code SP Reason-Phrase CRLF
Status-Code    =   "200"   ; OK
                 | "404"   ; Not Found
Reason-Phrase  = <CR, LF 以外の文字列>

entity-header  = "Content-Type" ":" media-type
media-type     = type "/" subtype *( ";" parameter )
type           = "text"
subtype        = "plain" | "html"
parameter      = "charset=us-ascii" | "charset=iso-2022-jp"
```

ここで, SP は空白文字, abs_path はファイルの絶対パス, CRLF は改行コード[11], Reason-Phrase はレスポンスのステータスを説明する文字列（下記の例を参考にして適当に設定してよい）.
例えば，クライアントがサーバに送信する Request は,

```
GET /index.html HTTP/1.1[CRLF]
Host: www.hogehoge.com:80[CRLF]
```

のようになる．また，サーバがクライアントに返す Response は,

```
HTTP/1.1 200 OK[CRLF]
Content-Type: text/plain; charset=us-ascii[CRLF]
... 以下，メッセージ本体が続く ...
```

などとなる．その他の記号については，資料を参照すること.

[11] message-body 部以外では，プロトコル上，改行コードは CR LF であるので，UNIX 系の OS の場合，単に printf("...\n") ではなく，printf("...\r\n") とするのが正しい．ただし，DOS/Windows 系の OS では，アスキーモードにすると，"\n" を指定するだけで，CR LF が出力される.

4.1 HTTP クライアントの作成（課題 2）

課題 2：HTTP クライアントの作成

サンプルのクライアントプログラムを参考にして，次の仕様を満たすクライアントを作成せよ．作成したクライアントで `http://www.fos.kuis.kyoto-u.ac.jp/le2soft/`を指定し，実験ホームページのソースが取得できることを確認せよ．

クライアントの動作 クライアントの動作の概要は以下のとおりである．

1. （サンプルプログラムとは異なり,）引数をとらずに起動.
2. 標準入力から URL を受け取り，ホスト名，ポート番号，パスを取得する（課題 1 で作成した解析器を利用）.
3. 得られた情報をもとにサーバに TCP で接続し，上記プロトコルに従ってコンテンツの転送要求 (`Request`) を送信する.
4. サーバから送信された結果を画面に表示する．`message-body` のみではなく，サーバから送られる `Response` 全てを表示すればよい．このとき，一行受信するごとに表示を行なうこと.
5. 2. に戻る.

実行例 以下は実行の一例である.

```
% ./client
Input URL:
http://www.fos.kuis.kyoto-u.ac.jp/le2soft/ （←端末からの入力）

（サーバから受信した内容をそのまま表示）
HTTP/1.1 200 OK
Content-Type: text/html; charset=iso-2022-jp
<html>
...
```

プログラムの内部では，入力された URL を解析した後，`www.fos.kuis.kyoto-u.ac.jp` の 80 番ポートに TCP 接続し，以下のリクエストをサーバに向けて送信している.

```
GET /le2soft/ HTTP/1.1 (CRLF)
Host: www.fos.kuis.kyoto-u.ac.jp (CRLF)
(CRLF)
```

このリクエストは端末には表示しない

報告書 1

課題 2 で作成したプログラムの内容について報告書を作成し，提出せよ．課題 1 で作成した URL 解析部の内容についても報告書に含めること．(提出するプログラムは課題 2 のもののみで構わない．)

4.2 HTTP サーバの作成（課題 3）

課題 3：HTTP サーバの作成

サンプルのサーバプログラムを参考にして，次の仕様を満たすサーバを作成せよ．課題 2 で作成したクライアントから，本課題のサーバに接続し，テキストの送受信ができていることを確認せよ．さらに，他の人が作成した他ホスト上のクライアントとの間での通信ができることを確認せよ．

サーバの動作　1. 引数としてポート番号を受け取って起動する．

2. クライアントからの接続を待つ．
3. 接続確立後，クライアントからのコンテンツ転送要求に応じて Status-Line, entity-header とともに，コンテンツを送信する（パスによって指定されたファイルの内容をそのまま送信すればよい）．
4. 2. に戻る．

実行例 クライアントからのリクエスト（表示しない）：

```
GET /index.html HTTP/1.1 (CRLF)
Host: www.kuis.kyoto-u.ac.jp (CRLF)
(CRLF)
```

サーバの応答・成功の場合（クライアントへ送信）：

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=us-ascii
... 以下コンテンツが続く ...
```

サーバの応答・失敗の場合（クライアントへ送信）：

```
HTTP/1.1 404 Not Fonund
Content-Type: text/html; charset=us-ascii

<HTML><HEAD>Not Found</HEAD>
<BODY>
The requested URL /index.html was not found on this server.
</BODY></HTML>
```

補足

- 本実験では，サーバが返す entity-header は，

  ```
  Content-Type: text/html; charset=us-ascii
  ```

  に固定してよい．
- いわゆる Proxy サーバを作るわけではない．サーバは自身が持っているコンテンツだけ送信すればよい．

4.3 複数クライアントへの対応（課題 4）

課題 3 のプログラムでは，サーバが同時に複数のクライアントに対応することができなかった．これでは，応答時間が遅くなるばかりか，正しい動作をしないクライアントによって，サーバが他のクライアントに全くサービスを提供できない状態に陥いることもありえる．そこで，クライアントからの接続要求毎にプロセスを分けてサービスを行なえるようにサーバを変更する．

UNIX 系の OS では fork システムコールを用いてプロセスを生成することができる．fork システムコールは，プロセスが二つに分岐 (fork) することで新しいプロセスを生成する．基本的には二つのプロセスは同じ働きをするものだが，アドレス空間は別になっており，親子関係がある．親プロセスは返り値として子プロセスのプロセス ID を，子プロセスは返り値として 0 を受け取るので，この返り値を用いて役割を割り当てることができる [12]．

fork の典型的な使用例:

```
int pid;
for (;;) {
   ...
   pid = fork();   /* プロセスを親子に分岐 */
   if (pid == 0) {
      /* 子プロセスでの処理 */
      exit(0);
   } else {
      /* 親プロセスでの処理 */
   }
}
```

この例では，fork システムコールによってプロセスを親子に分岐する．それ以降の処理は親プロセス，子プロセスそれぞれで行なわれるが，pid の値は，親子で異なるため，これを用いて条件分岐して親子の役割を分けることができる．

サーバプログラムはおおよそ以下のように変更される．

[12] 子プロセスのプロセス ID は，wait システムコールを使う場合などに利用するが，この実習では使わない．

```
int pid, s, ns;

.../* ソケット s を開き，bind し，listen で接続待ち */...

for (;;) {
   ns = accept(s, ...);
   pid = fork();   /* プロセスを分岐 */
   if (pid == 0) {
      /* 子プロセスでクライアントの要求を受け付ける */
      close(listenfd);
      ...
      exit(0);
   } else {
      /* 親プロセスは次のクライアントからの接続を待つ */
      close(acceptfd);
   }
}
```

課題 4：複数クライアント対応サーバの作成

課題 3 のサーバプログラムを，接続ごとに子プロセスを起動するものに変更せよ．実際に自分で複数のクライアントから同時にサーバに接続して動作を確認せよ．

補足 複数クライアントに対応できていることを検査するためには，サーバからのコンテンツ送信時に一行ずつ待ち時間を挿入しておき，複数の端末ウィンドウから二つ以上のクライアントを同時に接続してみるとよい．このためには，C の sleep 関数が使える．例えば，

```
printf("foo\n");
sleep(1);
printf("bar\n");
```

とすると，foo を表示後，1 秒待ってから bar を表示する．

その他，telnet を使う方法もある．

```
% telnet localhost 1111
```

とすれば，localhost の 1111 番ポートへの接続を行なうので，接続した状態で別の端末からクライアントを接続してみればよい．

報告書 2

課題 4 で作成したプログラムの内容について報告書を作成し，提出せよ．課題 3 で作成した部分の内容についても報告書に含めること．(提出するプログラムは課題 4 のもののみで構わない．)

4.4 exec システムコールによる CGI の実現（発展課題）

fork するだけでは同じプログラムコードを持つ複数のプロセスができるだけであり，別のプログラムコードを持つプロセスを作ることはできない．そこで exec システムコール群を用いて，現プロセスを別のプログラムコードに置き換えることができる．

例えば次のようにして，新しいプロセスを起動できる．

```
...
pid = fork();
if (pid == 0) {
   if (execl("/bin/ls", "ls", "-lR", NULL) < 0) {
      perror("execl");
      exit(1);
   }
}
```

ここでは，fork によって分岐された子プロセスが，execl によって/bin/ls -lR を実行するプロセスに置き換えられている．execl の一つ目の引数が実際に実行するコマンドのパス，二つ目以降が引数の列である（NULL によって引数の最後であることが示されている）．

以上を踏まえて，以下のような発展課題を考えてみよ．

発展課題：CGI の実現

指定された URL の最後が「.cgi」であった場合，サーバが指定されたファイルを実行し，結果をクライアントに返すようにせよ．

なお，子プロセスが標準出力に出力するデータを別のソケット（ファイルディスクリプタ）にリダイレクトするには，dup もしくは dup2 システムコールを使えばよい．例えば以下では，ファイル ls.out に ls -lR の実行結果が出力される．

```
...
fd = open("ls.out", O_WRONLY|O_CREAT, 0644);
close(1); /* 標準出力を close */
dup(fd); /* 標準出力を fd にリダイレクト */
if (execl("/bin/ls", "ls", "-lR", NULL) < 0) {
...
```

さらに，CGI では様々なパラメータをクライアントからサーバに送ることができる．CGI がパラメータを受け取れるように変更せよ。

一般にこの実装は，サーバで，クライアントから受け取ったパラメータを環境変数に設定して CGI プログラムを起動することにより行う（詳しいパラメータの受け渡しの書式は参考書を参照すること）．

発展課題：画像ファイルの表示

HTTP 1.1 プロトコルのサブセットを拡張して，

```
type            = "text" | "image"
subtype         = "plain" | "html" | "gif" | "jpeg"
```

とする．

クライアントが `image/gif` などの `Content-Type` を受け取った場合，コマンド `display` を起動して画像を表示できるようにせよ．受け取った画像を一旦一時ファイルに保存し，それを引数として与えて `display` を起動すればよい．(この他，ファイルに保存せず `dup` を用いる方法がある．`display` は `"-"` オプションのみで起動すると標準入力から画像データを受け取る．なお，Unix-like OS では画像表示コマンドとして `display` の他に `xv` などがよく使用される．

5　オブジェクト指向プログラミング言語 Java

実験 2 の後半では，より実用的なプログラミング言語として Java を修得し，前半で作成した HTTP サーバ・クライアントを Java で作成する．

5.1　Java とは

Java は Sun Microsystems 社が開発したプログラミング言語である．当初からネットワーク上で利用されるアプリケーションの開発が意識されており，ネットワークを利用する機能がより抽象的な形で提供されているため，容易にネットワークプログラミングができる．また，プラットフォームに依存せずに実行できるプログラムが書けることも大きな利点である．

Java のもう一つの大きな特徴は，**オブジェクト指向**と呼ばれる考え方にもとづいて設計されていることである．オブジェクト指向のプログラミング言語では，データや機能など「ひとまとりの概念」の各々を**オブジェクト**であると考えることにより，プログラムをオブジェクトの集まりと捉える．このオブジェクトの設計図（またはテンプレート）が**クラス**であり，各オブジェクトはあるクラスの**インスタンス**として生成される．各オブジェクトは適当なメッセージを受け取ることにより**メソッド**と呼ばれる処理を行なう．

5.2 Java 超入門

まずは，ごく簡単なサンプルプログラムを実際に動かしてみよう．

演習 6 ダウンロードしたサンプル[13]のうち，sample/java ディレクトリ中の sample1.java をコンパイルして実行せよ．

```
% javac sample1.java
% java HelloWorld
Hello World.
```

javac によってコンパイルを行なうと，sample1.java で定義されているクラスの**クラスファイル** HelloWorld.class が生成される．Java のプログラムは，指定したクラスの main メソッドから実行が開始される（C 言語において，main 関数が実行されるのと同様である）．上の例では，java HelloWorld によって，HelloWorld クラスで定義されている main メソッドが実行される[14]．

オブジェクトの操作は，そのオブジェクトが属するクラスで定義されているメソッドを呼び出すことによって行なう．sample1.java 中の

```
System.out.println("Hello World");
```

において，System.out は標準出力を表すオブジェクトであり，println はそのオブジェクトに文字列を一行出力するメソッドである．

同様に，sample2.java，sample3.java をコンパイルして実行せよ．sample2.java の PrintFile は，標準入力からファイル名を受け取って，そのファイルの内容を表示する．

```
% javac sample2.java
% java PrintFile
sample2.java    （←適当なファイル名を入力）
...sample2.java の内容が表示される...
```

sample3.java の FindFromFile は，引数として二つの文字列 str1 と str2 をもらって実行し，標準入力からファイル名を受け取ると，ファイル中で，str1 から始まって str2 で終わる文字列を順番に探し出して表示する．

[13] サンプルプログラムのダウンロードについては本資料 1.3 章を参照

[14] gcc が生成する実行ファイルとは異なり，Java のクラスファイルを直接実行することはできず，**Java 仮想マシン**（**JVM**）上で実行される．java コマンドはクラスファイルを JVM 上で実行するためのコマンドである．

```
% javac sample3.java
% java FindFromFile a s
sample3.java    （←適当なファイル名を入力）
as
atic void main(String[] args
args
ame = new BufferedReader(new InputStreamReader(Sys
attern pat = Pattern.compile(args
args
```

コマンドライン引数は，main メソッドのパラメータとして指定されている変数 String[] args に格納される．すなわち，args は String 型の配列になっており，コマンドラインから渡した引数の 1 つ目，2 つ目，...がそれぞれ文字列として args[0]，args[1]，...に格納されている [15]．

このサンプルでは，検索を行単位で行なっているため，行をまたがる（改行を含む）文字列がマッチしても表示されない．例えば，一旦ファイルの内容を一つの文字列として，その文字列を対象としたマッチングを行なえば，複数行にまたがる文字列も検索対象となる．

もう一つの例 sample/java/sample4.java を用いて，クラス，オブジェクト，メソッドについてごく簡単に説明する．このプログラムは，2 つのクラス定義からなっている．Switch クラスは on/off の 2 状態をもつスイッチ（またはフラグ）を表しており，ClassSample クラスはこのスイッチを利用したプログラム本体を想定している．実行例は以下のとおりである．

```
% javac sample4.java
% java ClassSample
false
true
```

main メソッドでは，まず new によって Switch クラスのオブジェクト s1，s2 を作っている．このとき呼び出されるのが，Switch クラスの**コンストラクタ**

```
Switch() { flag = 0; }
```

である．flag はクラス定義の始めに宣言されている局所変数のようなものであるが，それぞれのオブジェクトごとに独自の値を持っている [16] ことに注意しなければならない．このような変数を**インスタンス変数**と呼ぶ．これによって，各オブジェクトの「内部状態」を表現することができる．この例の場合は，各スイッチの状態を整数（の偶奇）を利用して表現している．Switch のコンストラクタは，この flag の初期値を 0（この例では偶数が on なので，初期状態 on）とするオブジェクトを作成している．

[15] C の場合とは異なり，args[0] は 1 つ目の引数であってコマンド名ではない．
[16] static が指定されている場合は，オブジェクト毎の値は持たず，クラスで一つの値を持つ．

次に，Switch クラスのメソッドを見てみよう．check メソッドは引数を取らず，内部状態の整数 flag が表す on/off を真偽値として返している．すなわち，オブジェクトの flag が偶数ならば true，奇数ならば false を返すメソッドである．flip メソッドは，そのオブジェクトの on/off を切り替えるメソッドである．

これをふまえて main メソッドに戻ると，s1 は初期状態 on（true）で作成された後，flip メソッドで off に切り替わっている．一方，Switch クラスの変数 flag は，オブジェクト毎に独自の値を持つので，s2 は初期状態 on のままである．

この例の Switch クラスでは，スイッチの状態を整数の偶奇で表現して実装しているが，これを利用する ClassSample 側ではスイッチの操作はコンストラクタと 2 つのメソッドを介してのみ行なえばよく，その実装の内容を意識する必要がない．例えば，flag を（もっと素直に）boolean 型の変数として実装した別の Switch クラスに差し替えたとしても，ClassSample を変更する必要はない．

実験では Java の基本についてこれ以上詳しい解説は行なわないので，教科書「すべての人のための Java プログラミング (第二版)」（立木秀樹，有賀妙子）[17] を読んで各自勉強すること．読むべき箇所を以下に挙げる．なお，C 言語の理解が十分でない者には 4 章全てと 5.2 も読むことを勧める．

- 2 章全て (pp.8-22)
- 3 章全て (pp.23-30)
- 4 章の 4.5 まで (pp.31-40)
- 5 章の 5.1, 5.3, 5.6 (pp.43-45, pp.47-48,pp.51-52)
- 6 章の 6.5 まで (pp.53-58)
- 7 章の 7.6 まで (pp.60-74)
- 9 章の 9.2 まで (pp.94-98)
- 15 章全て (pp.200-213)
- 16 章の 7.6 まで (pp.215-225)

旧版（第一版）の使用者は，以下が該当する．C 言語の理解が十分でない者には，3.8，4 章全て，5.2, 5.3 も読むことを勧める．

- 2 章全て (pp.6-20)
- 3 章の 3.7 まで (pp.21-29)
- 5 章の 5.1, 5.4, 5.5 (pp.43-44, 48-50)

[17] http://www.i.h.kyoto-u.ac.jp/~tsuiki/java-everyone/

- 6 章の 6.5 まで (pp.51-60)
- 8 章の 8.6 (pp.83-86)
- 9 章の 9.1 と 9.2 (pp.87-90)
- 14 章の 14.2 と 14.3 (pp.191-196)
- 15 章全て (pp.202-213)

以下に参考となるウェブサイトを挙げる.

- Java SE 6 API specification (http://docs.oracle.com/javase/6/docs/api/)

5.3 Pattern クラスと Matcher クラス

sample3.java で使用しているクラス Pattern と Matcher については上記の教科書で解説されていないので，ここで少しだけ説明しておく．これらを用いることで，正規表現を利用したパターンマッチングを簡単に行なうことができる.

Pattern は,（コンパイルされた）正規表現のクラスである．Java の正規表現については, Pattern クラスの仕様 [18] を参照のこと．sample3.java では,

```
Pattern pat = Pattern.compile(args[0] + ".*?" + args[1]);
```

によってマッチングに用いる正規表現をコンパイルして Pattern オブジェクト pat を得ている．compile メソッドは正規表現を表す文字列を受けとる．上例の pat は「args[0] から始まって args[1] で終わる文字列」にマッチする正規表現の Pattern オブジェクトである．また，例えば

```
Pattern pat = Pattern.compile("aaa.+bbb");
```

とすると,「aaa で始まり，一文字以上の文字列を挟んで，bbb で終わる」文字列にマッチする正規表現の Pattern オブジェクトが得られる.

Pattern オブジェクトの matcher メソッドによって,

```
Matcher mat = pat.matcher(line);
```

のように，文字列に対するマッチング操作を行なうエンジンである Matcher オブジェクトが得られる.

Matcher オブジェクトに対する主な操作には find メソッドと group メソッドがある.

- find(): パターンとマッチする次の部分を検索する．マッチが成功した場合のみ true を返す.

[18] http://java.sun.com/j2se/1.5.0/ja/docs/ja/api/java/util/regex/Pattern.html

- group(int n): 前回の検索でマッチした部分の文字列を返す．引数は「マッチした文字列のうち，左から何番目のグループか」を指定する（詳しくは以下で説明する）．引数を 0 とすると，マッチした全体の文字列を返す．

sample3.java ではこの二つのメソッドのみを用いて，マッチした文字列を列挙している．

さらに，正規表現中で先方参照を行なうグループを，カッコ (...) を用いて指定することができる．例えば，

```
Pattern pat = Pattern.compile("b(.+)f(.+)i");
Matcher mat = pat.matcher("abcdefghijk");
mat.find();
```

とマッチングを行なった後に，mat.group(0) とすると bcdefghi が返り，mat.group(1) とすると正規表現中の「左から一番目のグループ」である，一つ目の.+にマッチした部分である cde が返り，同様に mat.group(2) とすると gh が返る．

5.4 クライアント・サーバプログラム

Java でのネットワークプログラミングは，C 言語に比べて，より抽象化された機能として提供されており，プログラムを書くことも理解することも容易である．

演習 7 ダウンロードしたサンプル[19] のうち，sample/java-cs ディレクトリ中の client.java と server.java をコンパイルして実行せよ．

Server は，マルチクライアント対応のサーバである．第一引数に接続を受けるポート番号，第二引数に送信間隔（ミリ秒）を取って起動する．Client は，第一引数にサーバのホスト名，第二引数にポート番号を取って起動する．接続が確立すると，サーバは指定された送信間隔で乱数を 10 回送信し，クライアントは受信した乱数を順次表示する．

サーバ起動例:

```
% javac server.java
% java Server 30001 500
（接続待ちに入る．接続が確立すると，New connection. と表示される.）
（クライアントからのリクエストメッセージが来れば，それを表示する.）
```

クライアント起動例:

[19] サンプルプログラムのダウンロードについては本資料 1.3 章を参照

```
% javac client.java
% java Client localhost 30001
0.2425523772474496
0.932094495806451
...
0.2663380226295512
%
```

一つのクライアントを接続させたまま，他の端末ウィンドウから別のクライアントで同じサーバに接続可能であることを確認せよ．

これらのサーバ，クライアントは，C のサーバ，クライアントのサンプルとの間でも正常に通信が可能である．これを確認せよ．

client.java の概要 コマンドに対する引数は main への引数 args で文字列の配列として受け取っている．配列の内容は C の場合と異なり，一つ目の引数が args[0] に入ることに注意．次に，

```
Socket s = new Socket(args[0], Integer.parseInt(args[1]));
```

で，Socket クラスのインスタンスを生成している．コンストラクタに渡している一つ目の引数 args[0] が接続先のホスト名，二つ目の引数がポート番号である（文字列として受け取った引数を整数化して渡している）．このインスタンス生成は，C の時の socket システムコールのようにソケットを開くだけではなく，指定したホストとポートに接続するところまで行なう．接続に失敗した場合は，原因に応じて UnknownHostException などの例外を発生するのでこれを try···catch で処理している．

次に，ソケット s を通じてデータ送受信を行なうために，BufferedReader のインスタンス in（受信用）と PrintWriter のインスタンス out（送信用）を用意し，out を通じてリクエストを送信した後，in を通じて受信したデータを表示している．readLine はストリームから 1 行を読み込むメソッドである [20]．

server.java の概要 サーバプログラム側では，サーバ用のソケットを

```
ServerSocket serverS = new ServerSocket(Integer.parseInt(args[0]));
```

によって，ServerSocket クラスのインスタンスとして生成している．コンストラクタの引数はポート番号である．この段階で C のときの listen システムコールを行なったのと同様，接続待ち状態になり，クライアントからの接続が行なわれれば serverS.accept() によって確立される．このとき，

[20] C の fgets と似ているが，readLine の返値は終端の改行文字などを**含まない**ことに注意．

```
new ServerThread(serverS.accept(),
                 Integer.parseInt(args[1])).start();
```

によって ServerThread クラスのインスタンスを生成することにより，新たなスレッド上で一つのサーバを実行する（C の時に fork によってプロセスを分岐させたのと似ている）．生成された ServerThread インスタンスの start() メソッドを実行することにより，新スレッドのサーバを動かし，もとのスレッドでは New connection のメッセージを表示した後，次のクライアントからの接続を待つ状態に戻る．

Thread クラスを継承した ServerThread クラスは，インスタンス変数 clientS と sleepTime，コンストラクタ，および，run メソッドからなる．run メソッドには，start メソッドによって開始される処理を記述する．ここでは，接続が確立したソケット clientS を通じて送受信を行なう，PrintStream と BufferedReader のインスタンスを用意して，クライアントと同様にデータのやりとりを行なっている．

5.5 Java による HTTP サーバ・クライアントプログラミング（課題 5）

課題 5：Java による HTTP サーバ・クライアントプログラミング

課題 4 において C で作成したものと同機能の HTTP サーバと HTTP クライアントを Java で実装せよ．この時，クライアントにおいては，以下の仕様を満たす Browse クラスを作成・利用して，サーバからダウンロードした html ファイル内のリンクを自由に辿れるようにせよ．クライアントの実行例は以下の通りである（プロンプトやリストの表示などの UI は自由に工夫すること）．

```
% java Client
> http://www.fos.kuis.kyoto-u.ac.jp/le2soft/index.html （URL を入力）

...（実験 2 のページの index.html の内容が表示される）...

[list of links]
1. http://www.fos.kuis.kyoto-u.ac.jp/le2soft/faq.html
2. http://www.fos.kuis.kyoto-u.ac.jp/le2soft/basic.html
...（index.html 内のリンク先 URL 一覧）...
> 1     （リンクの番号を入力）

...（faq.html のソースとリンク一覧が表示される）...
>
```

外部仕様

- コンテンツ内のリンクの一覧を表示し，番号入力などによって選択したリンク先のページを表示できるような UI を実装すること（上記実行例を参考にせよ）．

- リンクとして抽出するのは，a（アンカー）タグの href 属性として指定されているもののみでよい.（html のコメントとなっている部分のリンクは無視すべきであるが，この課題ではコメント内のリンクを拾ってもよいものとする.）
- リンク先がファイル名（や，../index.html などの相対パス）のみで指定されている場合は，現在見ているページのパスをもとに URL を補完する．例えば，http://www.hoge.com/foo/bar/index.html 中に... というアンカーがある場合，このリンクは http://www.hoge.com/foo/bar/top.html という URL を指している.（この課題では，先頭が http://でないものは全てファイル名として処理すればよい.）

Browse **クラスの内部仕様** 現在閲覧しているページの URL やコンテンツの情報を保持するインスタンス変数群と，その内容を更新するメソッド群を実装すること.

インスタンス変数の例：

- 現在閲覧しているページの URL
- そのページのコンテンツ
- そのページに含まれるリンクの一覧

メソッドの例：

- ページのコンテンツからリンク一覧を更新するメソッド
- リンクの一覧を表示するメソッド
- 入力された番号のリンクを返すメソッド

実装例 以下は Browse クラスの実装の一例である．実装の方法はこの他にも色々考えられる．例えば，以下では links は String の配列として実装しているが，リストを利用して List<String>型のインスタンス変数として実装することもできる．また，相対パスに対応するために，基準となるパスをインスタンス変数として保持することも考えられる．この例では，コンストラクタは初期値を固定しているが，引数として初期値を受け取るように実装することもできる．どのようなインスタンス変数とメソッドを用意しどのように実装するのがよいか，自分なりに工夫すること.

```
class Browse {
  // instance variables
  public String url, contents;
  public String[] links;
  ....

  // constructor
  public Browse() {
```

```
    url = "";
    links = new String[0];
    contents = "";
  }

  // methods
  public void collectLinks() {
    // contents 中のリンクを抽出し，links を更新
  }

  public void printLinks() {
    // links の内容をナンバリングしながら表示
  }

  public String getLink(int n) {
    // links 中の n 番目の URL を返す
  }
  ...
}
```

Client クラスでは Browse クラスのオブジェクトを生成（例えば browse とする）しておき，入力された URL や，選択されたリンク番号に応じて，browse の内容を更新したり，browse の情報を取得したりしながら，サーバに接続したり，コンテンツを表示したりすればよい．以下は一例である．

```
class Client {
  public static void main(String[] args) {
    Browse browse = new Browse("", new String[0], "");
    ...

    // 最初の URL 入力を受けつけ，browse を更新

    while(true) {
      // browse の「現在閲覧中の URL」を解析
      // サーバに接続し，リクエストを送信
      // 受信したコンテンツを表示し，browse を更新
      // コンテンツ中のリンクを収集し，browse を更新
      // リンク一覧を表示
      // 入力された番号に応じて browse を更新
```

```
    }
  }

  // main 中で利用されるメソッド群
}
```

クラス毎の役割についても，URL の解析やリンクの番号選択の処理をどこで行なうか？など，設計の仕方は様々である．正しく動作しさえすれば「必ずこのように実装しなければいけない」という絶対的な正解はないので，各自でどのように実装すればよいかを考えること．

報告書 3

課題 5 で作成したプログラムの内容について報告書を作成し，提出せよ．報告書 3 についてはとくに以下を採点基準とする．

外部仕様 起動方法，URL の入力方法，リンクのたどり方等，操作方法が容易に理解できるかどうか．

内部仕様 クラス・変数・メソッドの説明，呼び出し関係，処理の流れ等，プログラムコードが容易に理解できるかどうか．

実行例，評価 エラー処理，例外処理等，緻密で完成度の高い開発を行ったことが明確に示されているかどうか．

A　報告書とデモに関して

本演習では報告書 1-3 の三つの報告書の提出と総合デモの実施が必須である.

A.1　報告書の提出方法

必要なファイル全てを含むアーカイブを添付したメールを

le2soft@fos.kuis.kyoto-u.ac.jp

まで提出する．この際，以下に注意すること.

- タイトル (Subject) は,「**le2-[報告書番号]-[氏名]**」とする.（例）Subject: le2-1-Koji Nakazawa
- メール本文には，**氏名・入学年度・学籍番号・課題番号**を記載する.

また,「必要なファイル」とは、最低限以下のものを含んでいること.

- **提出ファイルの説明**（ファイル名を README とし，ディレクトリ内のどれが何のファイルなのかを説明する.）
- **報告書**（ファイル名を REPORT1.txt などとして，報告書であることが分かるようにすること.）
- **ソースプログラム**（複数ファイルに分かれているときは，その全て）
- **その他コンパイルの際に必要となるファイル**（Makefile など）

なお，README と報告書はプレーンテキスト形式で記述すること.

課題提出の具体例　1. 課題提出に必要なファイル一式を一つのディレクトリ（例えばkadai1/）に入れる.

2. アーカイブを作成する.

```
% tar zcvf kadai1.tar.gz kadai1/
```

3. できあがった kadai1.tar.gz をメールに添付して上記アドレスに送信.

課題提出に関する注意事項 万が一時間が足りなくて課題の提出期限までに全部出来なかった場合，全部できていないことを明記し，できた所までの内容で提出すること (当然，評価は下がる)．その上で，期限に関わらず，完成させたものを提出すること.

A.2 報告書の内容について

報告書に期待される構成と内容は概ね以下の通りである．全ての内容が含まれていることが期待されるが，章立てがこの通りである必要はない．その他必要な項目があれば適宜追加すること．

- プログラム仕様
  - 外部仕様
    - 動作の概要
    - ファイルの説明・コンパイル方法および実行方法
    - コマンドライン引数の説明
    - 操作方法
    - 入力データの形式
    - 出力データの形式，等々
  - 内部仕様
    - モジュール構成
    - (自分で定義した) データ構造 (構造体・共用体) の説明
    - 大域変数の名前、意味、参照関係の説明
    - 各モジュールの仕様
      - 関数名
      - 関数の機能
      - 引数の意味，返り値の意味，等々
- 実行例
  - 実行の手順
  - 出力結果
- 評価
  - 採用した設計について
    - 正当性
    - 効率
    - その他の評価
  - 完成度，改善の余地
- 感想
- 参考文献

A.3 総合デモについて

以下の動作が正しく行なえることをデモで確認する．

- C で実装したクライアントが，C で実装したサーバ，Java で実装したサーバそれぞれから，指定したファイルを受信できる．
- Java で実装したクライアントが，C で実装したサーバ，Java で実装したサーバそれぞれから，指定したファイルを受信できる．
- 各サーバは複数のクライアントに並行してファイルを送信できる．（課題 3 の注意を参照すること．）
- 実験のサイト（`http://www.fos.kuis.kyoto-u.ac.jp/le2soft/`）内でリンクを自由にたどることができる．

B シェルスクリプト

UNIX ではシェルスクリプトと呼ぶバッチ処理方法がある．UNIX で本来直接実行可能なファイルは，ELF 形式や a.out 形式と呼ばれるバイナリ形式だが，実行権限があるにもかかわらずそのような形式でないものもある．この場合，ファイルの先頭が「`#!`」で始まっているかどうかを調べ，もしそうならば，その後に続く文字列をコマンドと見なして，さらにそのファイルを引数に加えて実行を試みる．

例えば，

```
#!/bin/sh

ls -aF
```

という内容のファイルを，`ls.sh` という名前で保存し，

```
% chmod +x ls.sh
```

として実行権限を与えると `ls.sh` を実行することができる．

また例えば，`server.c` と `client.c` というプログラムからコンパイルして実行ファイルを作りたければ，

```
#!/bin/sh

for i in server client
do
   gcc -Wall -g $i.c -o $i
done
```

というスクリプト (ファイル名を compile とする) を用意しておくことが考えられる．

シェルスクリプトはパイプや sed，awk などのプログラムを組み合わせてこそ，その真価を発揮する．ここではこれ以上詳しく解説しないので，各自学習しておくこと．

C　make と Makefile

make とは，複数のファイルの依存関係を記述することによって，プログラムをコンパイルして生成するためなどに使われる．make は，ファイルの修正時刻をチェックし，修正されたファイルに依存した部分に関するコンパイル等だけを実行する依存関係は，通常 Makefile という名前のファイルに記述される．

Makefile には，

目的となるもの：目的のため必要なファイル（複数記述可）
[タブ] 実行すること

という形式で記述される.「実行すること」の手前には**必ず**タブが必要である．例えば，

```
all: server client

server: server.o
   gcc -o server server.o

client: client.o
   gcc -o client client.o

server.o: server.c
   gcc -Wall -g -c server.c

client.o: client.c
   gcc -Wall -g -c client.c
```

このような Makefile を用意しておけば，

```
% make
```

とするだけで適切なコンパイルが行われる．また，client.c のみを修正した場合は client のみコンパイルを行ない，common.h を修正した場合は client，server ともにコンパイルを行なう，といった依存関係のチェックを自動的に行なってくれる．もし，client.c と server.c で，同一のヘッダファイル common.h を利用している場合は，client.o と server.o の両方がこのヘッダファイルに依存することになる．この場合は，例えば，

```
server.o: server.c common.h
   gcc -Wall -g -c server.c

client.o: client.c common.h
   gcc -Wall -g -c client.c
```

などとすれば，common.h を編集した際に両方を再度コンパイルしてくれる．

さらに，このように書くこともできる．

```
CC = gcc

all: server client

server: server.o
   $(CC) -o $@ $^

client: client.o
   $(CC) -o $@ $^

server.o: server.c common.h
client.o: client.c common.h

.c.o:
   $(CC) -Wall -g -c $^

clean:
   rm -f server client server.o client.o
```

このとき，make コマンドに引数 clean を渡して，

```
% make clean
```

とすると，最後の clean の部分が実行され，コンパイル時に生成されるファイルを消去することができる．

D　デバッガ (gdb)

C プログラム中のバグを発見するためのツール（一般に**デバッガ**と呼ばれる）として **GNU Project Debugger** (gdb) を利用することができる．gcc と組み合わせて利用することにより，プログラムを特定の位置で一時停止させたり（**ブレーク・ポイント**の利用），停止中の位置での変数や式の値を確認することができるなど，バグの原因特定に非常に有用である．

D.1 gdb の利用例（その 1）

gdb を使って，サンプル[21] sample/c/sample1.c における strtok の動作を確認してみよう．次のように-g オプションをつけて gcc でコンパイルすれば，gdb 上でこのプログラムをデバッグすることができる．

```
% gcc -g -o sample1 sample1.c
% gdb sample1
... gdb のメッセージ ...
(gdb)
```

最後の (gdb) は**コマンド**を待つプロンプトである．ここで，run コマンドを実行すると，通常通りプログラムの実行が開始される．プログラムにコマンドライン引数を渡す場合は，この run に引数を与えればよい．

```
(gdb) run
Starting program: .../sample1
foo, bar, hoge        （foo, bar, hoge を入力）

1 = foo； 2nd = bar； 3 = hoge
[Inferior 1 (process XXXXX) exited normally]
```

次に，プログラム実行途中での変数の値を表示させて，strtok の動作を確認してみよう．プログラムを最初の strtok の手前で一時停止させるために，**ブレーク・ポイント**を利用することができる．

```
(gdb) break 9
```

と，break コマンドによってブレーク・ポイントを 9 行目（a = strtok(...) の行）に設定する．この状態で run コマンドを実行し，入力文字列として，上と同様に foo，bar，hoge を与えると，strtok は実行されずに

```
Breakpoint 1, main () at sample1.c:9
9   a = strtok(in, " ,\n");
```

と表示されて，実行が一時停止される．このメッセージは，sample1.c の 9 行目（の先頭）で停止していることを示している．ここで，print コマンドを利用して，式 in[0]（つまり，文字列 in の 1 文字目）を表示させると，

[21] サンプルプログラムのダウンロードについては本資料 1.3 章を参照

```
(gdb) print in[0]
$1 = 102 'f'
```

in[0] の内容が正しく 102（文字としては f）になっていることが確認できる．単に print in とすると，

```
(gdb) print in
$2 = "foo, bar, hoge\n\000\005..."
```

と，終端文字より先も含めて，配列としての in の値が全て表示される．これによって，改行文字 \n の次が終端文字'\000' になっていることがわかる．また，printf を利用してフォーマットを指定して表示することもできるし，what を利用して式の型を確認することもできる．

```
(gdb) printf "%s\n",in
foo, bar, hoge
(gdb) what in
type = char [40]
```

ここで step コマンドを実行すると,「次の命令のみ」を実行することができる（ステップ・イン実行）.

```
(gdb) step
10   b = strtok(NULL, " ,\n");
```

この時点での a の内容と，in の内容を調べてみよう．

```
(gdb) printf "%s\n",a
foo
(gdb) print in
$3 = "foo\000 bar, hoge\n\000\005..."
```

a の内容が確かに最初のトークン foo になっていることがわかる．また，もとの文字列 in 中の最初の区切り文字だった 4 文字目のカンマが終端文字'\000' に置き換えられていることもわかる．同様に，

```
(gdb) step
11    c = strtok(NULL, " ,\n");
(gdb) printf "%s\n",b
bar
(gdb) print in
$4 = "foo\000 bar\000 hoge\n\000\005..."
```

と，2 回目の strtok によって，2 つめのトークン bar が b に格納され，bar の後の区切り文字だったカンマが終端文字'\000' に置き換えられていることがわかる．以上のように，strtok 関数は，区切り文字を終端文字に置き換えながら，返り値としてトークン文字列を返していることがわかる．

ブレーク・ポイント以降の実行を続けるには，continue コマンドを実行する．

```
(gdb) continue
Continuing.
1st = foo; 2nd = bar; 3rd = hoge
[Inferior 1 (process XXXXX) exited normally]
```

ブレーク・ポイントは複数設定することが可能であり，この場合 continue コマンドによって「次のブレーク・ポイント」までが実行される．現在設定中のブレーク・ポイントは

```
(gdb) info breakpoints
```

で一覧表示できる．また，設定したブレーク・ポイントは，

```
(gdb) delete 1
```

などで削除できる．引数は行番号ではなく，ブレーク･ポイントの通し番号（info breakpoints で確認できる）であることに注意．

gdb を終了する場合は，quit コマンドを実行する．

D.2 gdb の利用例（その 2）

もう少し複雑なプログラムとして，sample/gdb/sample2.c を考える．このプログラムは，実行すると Segmentation fault で停止してしまう．この原因を gdb を利用して発見しよう．

```
% gcc -g -o sample2 sample2.c
% ./sample2
a = 456, b = 123
Segmentation fault
```

先程と同様，gdb を起動して run すると，例えば以下のようになる．

```
% gdb sample2
...
(gdb) run
Starting program: .../sample2
a = 456, b = 123

Program received signal SIGSEGV, Segmentation fault.
0x0000000100000e89 in swap (i=0x7fff5fbff96c, j=0x0) at sample2.c:6
6     *i = *j;
(gdb)
```

このメッセージから，swap 関数の 6 行目で停止していることがわかる．また，このとき swap 関数に与えられている引数が (i=0x7fff5fbff96c, j=0x0) であることもわかる．そこで，停止地点での変数 j の値を確認すると，

```
(gdb) print j
$1 = (int *) 0x0
```

0x0 すなわち NULL であり，この NULL ポインタを*j で参照しようとしたことが直接の原因であることがわかる．

問題は，swap 関数の第二引数に j=0x0 と NULL が渡されているところにある．では，この関数呼出がどこで起こったものかを探そう．停止地点までの関数呼出しの履歴は，backtrace コマンドで表示できる．

```
(gdb) backtrace
#0  0x0000000100000e89 in swap (i=0x7fff5fbff96c, j=0x0) at sample2.c:6
#1  0x0000000100000f08 in f () at sample2.c:20
#2  0x0000000100000f44 in main () at sample2.c:25
```

これによって，いま呼び出されている swap は，関数 f () の 20 行目（swap(&a, c);）で呼出されており，さらにこの f () は，関数 main () の 25 行目（f();）で呼出されていることがわかる．

では，swap が呼び出された時点での変数の値などを調べてみよう．しかし，停止している

地点は swap 関数が呼び出された後なので，このままでは f 中の局所変数 a や c を調べることができない．この場合，frame または up コマンドで，フレーム [22] を移動することによって，swap が呼び出された直前の状態を確認することができる．

```
(gdb) frame 1      （または up）
#1  0x0000000100000f08 in f () at sample2.c:20
20     swap(&a, c);
```

frame コマンドの引数は，backtrace コマンドで一番左に表示される番号である．今回の場合，現在呼び出されている関数を呼び出した地点に移動する（一つ「上がる」）だけなので，up コマンドを利用しても同じである．ここで，swap の第二引数 c を調べると，

```
(gdb) print c
$2 = (int *) 0x0
```

確かに NULL が渡されていることがわかる．

D.3 Emacs 上での gdb の利用

Emacs 上でソースファイルを表示させたまま，同時に gdb を実行させることができる．このようにすれば，ソースファイルが表示されているバッファにおいて，ブレーク・ポイントを設定した場所や実行が停止している箇所に印が表示され，非常に便利である．

Emacs 上で gdb を起動するときは，M-x gdb とすればよい [23]．ミニバッファに，

```
Run gdb (like this): gdb --annotate=3 sample1
```

などと表示されるので（実行ファイル名 sample1 が表示されていない場合は入力して）改行すると，gdb が起動され gdb のバッファが作られる．

D.4 gdb のコマンド一覧

上の例で紹介したコマンドのほとんどは省略形を持つ．例えば run コマンドは r だけで実行できる．

[22]（関数）フレームとは，各関数呼び出し時の情報を確保しておくメモリ（スタック）上の領域のことで，呼び出された関数本体の実行中に参照される実引数や局所変数の情報が格納されている．この例では，最初に swap が呼び出されている停止地点でのフレームから，その swap の呼び出し元である f のフレームに移動している．フレームの状態は，swap を呼び出す直前の状態である．詳細は「コンパイラ」の講義で解説される（はず）．

[23]Option キーを押しながら x（または，Esc を押してから x）を押すと，ミニバッファに M-x と表示され，入力待ちになる．ここで，gdb と入力してリターンキーを押す．

コマンド名	省略形	動作
run [args]	r	プログラムを（引数 args で）実行する
break n	b n	ソースの n 行目にブレーク・ポイントを設定
delete n	d n	n 番目のブレーク・ポイントを削除
info breakpoints	i b	ブレーク・ポイント一覧を表示
continue	c	プログラムの実行を再開
step	s	ステップ・イン実行
quit	q	gdb を終了
backtrace	bt	バックトレース（関数呼出の履歴）を表示
frame n	f n	n 番目のフレームを選択
up	up	一つ上のフレームを選択
down	down	一つ下のフレームを選択
info locals	i locals	選択中のフレームの局所変数一覧
info args	i args	選択中のフレームの実引数一覧
print e	p e	式 e を評価して表示
printf f,e	printf f,e	式 e を評価してフォーマット f で表示
what e	wha e	式 e の型を表示