課題要旨

本課題では，データを送受信するプログラムを作成する．プログラムは，接続要求 `connect` をする所謂「クライアント」を作成する．多重化処理にするため， `select` を用いて，同期多重入力を実現する．

ソケットの説明

本節では，ネットワークプログラミングにおける諸知識について述べる．

アドレス構造体 (IPv4) アドレス構造体は，アドレスの情報を格納するための構造体である．将来的な拡張を考慮して，構造体のメンバが構造体を指す状態になっており，実際のアドレス情報は，`(sockaddr->in_addr)->s_addr` 構造体に格納される．

Listing 1 アドレス構造体 (IPv4) [1, 2]

```c
struct in_addr
{
    unsigned long s_addr; // 32-bit IPv4 address, network byte ordered
};

struct sockaddr_in
{
    unsigned short sin_family; // TCP/IP(AF_INET)
    unsigned short sin_port;   // port number
    struct in_addr sin_addr;   // IPv4 address(32-bit)
    char sin_zero[8];          // unused
};

struct sockaddr
{
    unsigned short sa_family; // address family(AF_INET, AF_INET6, AF_UNIX...)
    char sa_data[14];         // 14 bytes of protocol address
};
```

定義したアドレス構造体の各メンバに対して，プロトコルやアドレスを設定する．このとき，ホストとネットワークでのバイトオーダーの変換を行う必要がある．また，アドレス構造体は代入する前に `memset` 関数で初期化する必要がある[1)]．

ソケットの生成と破棄 ソケットの生成は， `socket` 関数を用いる． `protocol-family` には，TCP/IP の場合は `PF_INET` [2)]を指定する． `type` には，TCP の場合は `SOCK_STREAM` を指定する． `protocol` には，TCP の場合は `IPPROTO_TCP` を，UDP の場合は `IPPROTO_UDP` を指定する．

1) システムによっては不要 [2].
2) `AF_INET` を示す例もあるが，厳密にはアドレスファミリとプロトコルファミリは異なる．

PF_INET では， type と protocol の組み合わせは，1 対 1 のため， protocol には 0 を指定しても同じである [2]．ソケットの破棄は， close 関数を用いる．

```
socket(int protocol-family, int type, int protocol);
close(int socket);
```

TCP 接続 TCP 接続は， connect 関数を用いる． address には，接続先のアドレス構造体を指定するが，設定したアドレス構造体が sockaddr_in であることから， sockaddr へのキャストが必要．

```
connect(int socket, struct sockaddr *address, unsigned int address_len);
```

同時多重入力

本課題では，同時多重入力を実現するために， select 関数を用いた． select 関数は，複数のファイルディスクリプタに対して，読み込み可能かどうかを調べる関数である． select 関数は，fd_set 型の変数を用いて，読み込み可能なファイルディスクリプタを管理する． fd_set 型の変数は， FD_ZERO マクロで初期化し， FD_SET マクロでファイルディスクリプタを追加する． select 関数は，

第 1 引数 ファイルディスクリプタの最大値に 1 を足した値を指定する．
第 2 引数 読み込み可能かどうかを調べるファイルディスクリプタの集合を指定する．
第 3 引数 書き込み可能かどうかを調べるファイルディスクリプタの集合を指定する．
（今回 NULL ）
第 4 引数 例外が発生したかどうかを調べるファイルディスクリプタの集合を指定する．
（今回 NULL ）
第 5 引数 タイムアウト時間を指定する．
（今回 NULL ）

から成る． FD_ISSET より，第一引数にファイルディスクリプタ，第二引数に監視対象の集合を指定する．条件に合致したら指定の処理を実行し，最後に FD_ZERO でファイルディスクリプタの集合を初期化する．

送信手続き

送信手続きは， write(2) 関数を用いる．ここでは， write(2) が 1 度で終わらないことを想定して，本来書き出すべきバイト数まで write(2) を実行する writing 関数を作成した．引数にディスクリプタ，バッファ，バッファ長を受け取ることで，ソケットだけに限らず，標準出力についても応用できた．

ソースコードの説明

ソースコードは別途提出しているほか，付録として p.4-p.8 に貼付している．ソースコードは， **client.c** と **ErrorHandling.c** の 2 つである． **client.c** は，メインのソースコードであり，**ErrorHandling.c** は，エラー処理を行うソースコードである．本課題以外でも用いているため別ファイルにしてある．実行は **Makefile** (lst:4) を用いて， **make exec** を実行すると，C ファイルがコンパイルされ，実行する． **Makefile** により，実行ファイルの引数には， **localhost** と **12345** が与えられる．実行と実行結果を以下に示す．

```
$ make exec
gcc  -c client.c
gcc  -c ErrorHandling.c
gcc  -o client.out client.o ErrorHandling.o
./client.out localhost 1234
Host Name   : localhost
Port number : 1234
>> IP address of localhost is 127.0.0.1
>> Connected to client: 127.0.0.1:1234
... 続く ...
```

感想

今回の課題では，再利用できるように，またソースコードの可読性を上げるために複数個関数を作成した．これがどこまで処理速度に影響を及ぼすか，実験したい．同時多重入力は， **select** 関数を用いることで，簡単に実装できた．ネットワークプログラミングがカバーする部分はあくまで，ソケットの作成，破棄，接続のみであることに注意したい．

参考

[1] スティーヴンス, W. (1999). UNIX ネットワークプログラミング: ネットワーク API・ソケットと XTI. 日本: トッパン.

[2] Donahoo, M. J., Calvert, K. L. (2003). TCP/IP ソケットプログラミング C 言語編. 日本: オーム社.

付録：ソースコード

Listing 2 `client.c`

```
#include <stdio.h>      // printf(), fprintf()
#include <sys/socket.h> // socket(), connect(), send(), recv()
#include <netdb.h>      // struct addrinfo, getaddrinfo(), gai_strerror()
#include <arpa/inet.h>  // sockaddr_in, inet_addr()
#include <stdlib.h>     // atoi()
#include <string.h>     // memset()
#include <unistd.h>     // close(), read(), write()
#include <sys/select.h> // select()

#define BSIZE 32 // Size of receive buffer

void ErrorHandling(char *message);                        // Error handling
   function
char *HostName2IpAddr(char *hostName, char *port);        // Convert host name to
    IP address
void recvFromServer(char *buffer, int sock);              // Receive string from
   the server
void sendToServer(char *buffer, int sock);                // Recviced string from
    prompt
void writing(int sock, char *buffer, size_t bufferLen); // sending data to
   server
void finalProcess(int descriptor, char *message);         // final process

int main(int argc, char *argv[])
{
    int sock;                        // socket descriptor
    struct sockaddr_in server_addr; // server address structure
    char *serverHostFromArgs;        // server host from command line
    char *serverPort;                // server port
    char Buffer[BSIZE];              // Buffer for received string
    size_t sendLen, recvedLen;       // length of Buffer

    if (argc != 3) // Test for correct number of arguments
    {
        fprintf(stderr, "Usage: %s <Server Host name> <Server Port>\n", argv
           [0]); // argv[0] is in executable file name
        exit(1);
    }

    serverHostFromArgs = argv[1]; // First arg: server host name
    serverPort = argv[2];         // Third arg: server port
```

```
    printf("Host Name    : %s\n", serverHostFromArgs);
    printf("Port number  : %s\n", serverPort);

    // Create a reliable, stream socket using TCP
    if ((sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0) // Create a
       reliable, stream socket using TCP
    {
        ErrorHandling("socket() failed");
    }

    // Construct the server address structure
    memset(&server_addr, 0, sizeof(server_addr));                          //
       Zero out structure
    server_addr.sin_family = AF_INET;                                      //
       Internet address family
    char *serverIpAddr = HostName2IpAddr(serverHostFromArgs, serverPort); //
       Convert host name to IP address
    server_addr.sin_addr.s_addr = inet_addr(serverIpAddr);                 //
       Server IP address; network byte-ordered (127.0.0.1->16777343)
    server_addr.sin_port = htons(atoi(serverPort));                        //
       Server port; host to network short

    // Establish the connection to the server
    // server_addr is a pointer to struct-sockaddr_in
    if (connect(sock, (struct sockaddr *)&server_addr, sizeof(server_addr)) <
       0)
    {
        close(sock);
        ErrorHandling("connect() failed");
    }
    printf(">> Connected to client: %s:%s\n", serverIpAddr, serverPort);

    // setup select function
    fd_set fds; // Set of file descriptors
    int max_fd;
    for (;;)
    {                                                          // Maximum file
       descriptor
        FD_ZERO(&fds);                                         // Clear all bits
        FD_SET(STDIN_FILENO, &fds);                            // Set bit for
           stdin
        FD_SET(sock, &fds);                                    // Set bit for
           sock
        max_fd = (sock > STDIN_FILENO) ? sock : STDIN_FILENO; // max_fd = max(
           sock, STDIN_FILENO)
        select(max_fd + 1, &fds, NULL, NULL, NULL);            // Wait for
           activity
```

```
        if (FD_ISSET(STDIN_FILENO, &fds))
        {
            sendToServer(Buffer, sock);
        }
        if (FD_ISSET(sock, &fds))
        {
            recvFromServer(Buffer, sock);
        }
        printf("␣---␣\n");
        FD_ZERO(&fds);
    }
    return 0;
}

void sendToServer(char *buffer, int sock)
{
    ssize_t sz = read(STDIN_FILENO, buffer, BSIZE); // exclude '\0'
    if (sz == 0)
    {
        finalProcess(sock, "EOF");
    }
    else if (sz < 0)
    {
        close(sock);
        ErrorHandling("read()␣failed");
    }
    printf(">>␣sending...␣\n");
    writing(sock, buffer, sz);
    memset(buffer, 0, BSIZE); // initialize buffer
}

void writing(int descriptor, char *buffer, size_t bufferLen)
{
    size_t sentLength = 0;
    ssize_t sentSize = 0;
    while ((int)sentLength < (int)bufferLen)
    {
        sentSize = write(descriptor, buffer, bufferLen);
        if (sentSize < 0)
        {
            if (descriptor != STDOUT_FILENO)
                close(descriptor);
            ErrorHandling("send()␣sent␣a␣different␣number␣of␣bytes␣than␣
               expected");
        }
        sentLength += sentSize; // Keep tally of total bytes
    }
```

```
}

void recvFromServer(char *buffer, int sock)
{
    size_t recvLen = 0;
    recvLen = read(sock, buffer, BSIZE - 1);
    printf(">> received... \n");
    if (recvLen == 0)
    {
        finalProcess(sock, "EOF");
    }
    else if (recvLen < 0)
    {
        close(sock);
        ErrorHandling("recv() failed");
    }
    buffer[recvLen] = '\0';
    writing(STDOUT_FILENO, buffer, sizeof(buffer));
    memset(buffer, 0, BSIZE);
}

char *HostName2IpAddr(char *hostName, char *port)
{
    struct addrinfo moreInfo, *response;    // More info about host
    memset(&moreInfo, 0, sizeof(moreInfo)); // Zero out structure
    moreInfo.ai_family = AF_INET;           // IPv4 only
    moreInfo.ai_socktype = SOCK_STREAM;     // Only TCP

    if (getaddrinfo(hostName, port, &moreInfo, &response) != 0)
    {
        ErrorHandling("getaddrinfo() failed");
    }

    char *ipAddr = inet_ntoa(((struct sockaddr_in *)response->ai_addr)->
       sin_addr);
    printf(">> IP address of %s is %s\n", hostName, ipAddr);
    freeaddrinfo(response); // Free address structure
    return ipAddr;
}

void finalProcess(int descriptor, char *message)
{
    printf(">> %s\n", message);
    if (descriptor != 0)
    {
        close(descriptor);
    }
```

```
    exit(EXIT_SUCCESS);
}
```

Listing 3 ErrorHandling.c

```c
#include <stdio.h>
#include <stdlib.h>

void ErrorHandling(char *message)
{
    perror(message);
    exit(1);
}
```

Listing 4 Makefile

```
CC = gcc
CFLAGS =
OBJS = client.o ErrorHandling.o
HOST = localhost
PORT = 1234
PROGRAM = client.out

all: $(PROGRAM)

$(PROGRAM): $(OBJS)
        $(CC) $(CFLAGS) -o $@ $^
# $^ means all the files on the right side of the colon (list of dependencies).
# $@ means the file on the left side of the colon (the target name).
# $< means the first item in the dependencies list.
# $? means all the dependencies that are newer than the target.

%.o: %.c
        $(CC) $(CFLAGS) -c $<
# -c means compile only, do not link.(generate .o file)

.PHONY: clean
clean:
        rm -f *.o $(PROGRAM) a.out
exec: $(PROGRAM)
        ./$(PROGRAM) $(HOST) $(PORT)
```