1. 実験の目的

事前学習でTCP/IPや構造体、ポインタ、動的メモリ、makeコマンドなどの基礎知識を学んで実践に活かす。実践ではネットワーク対戦型リバーシを作成することを通して、Socketを利用したTCP/IPプログラミングを実装する。

2. 原理

[TCP/IPについて、TCPとUDPのちがい]

TCP/IPとはネットワーク上でコンピュータ同士が通信するときに使われる手順や約束事（プロトコル）の一種である。インターネット技術特別調査委員会（IETF）によって技術仕様がまとめられたRFCに基づいて規定されている。ネットワークアーキテクチャを国際的に標準化したOSI基本参照モデルの各層と対応している。OSI基本参照モデルは下位層から順に物理層、データリンク層、ネットワーク層、トランスポート層、セッション層、プレゼンテーション層、アプリケーション層と標準化されている。TCP/IPはこの中で下位層の4つが共通していて、上位層の3つはアプリケーション層にまとめられている。

TCP（Transmission Control Protocol）は信頼性を重視していてアプリケーション層のHTTP, FTP, TELNET, SMTP, POP3などに利用される。一方UDP（User Datagram Protocol）は信頼性よりもリアルタイム性を重視していてDNS, DHCP, NTPなどに利用される。これは信頼性を担保しようとするとデータのチェックに時間がかかることによって信頼性とリアルタイム性の両立が不可能なので用途に合わせて使い分けられているということである。**[2]**

[構造体]

構造体とはプログラミング言語における変数の型の一つであり、任意の数と種類の変数を集めて管理するための変数である。以下にC言語で構造体を使う際の形式を示す。

以下の形式で構造体を宣言する。

struct 構造体の名前{

　　型1 変数名1;

　　型2 変数名2;

　　型n 変数名n;

}

構造体の変数を作成するときはmain関数の中で以下のように宣言する。

struct 構造体の名前 構造体変数名 = {値1, 値2, …, 値n}; // 初期化

変数を使うときは以下のようにする。

構造体変数名.変数名1 = 値1;

構造体変数名.変数名2 = 値2;

構造体変数名.変数名n = 値n;

[ポインタ、関数ポインタ]

ポインタとは変数のアドレスを格納する変数である。アドレスとは変数のメモリ上に割り当てられる番地のことである。C言語では変数名の先頭に「\*」をつけることで宣言した変数がポインタであることを示す。配列の場合はインデックス（[n]）を省略することで配列の先頭のポインタを表すこともできる。またポインタではない変数のアドレスを取得する際には変数名の先頭に「&」をつける。int型の整数numのアドレスをpnumに入れるときには以下のようにする。

int num; // 整数型の宣言

int \*pnum; // 整数型のポインタの宣言

pnum = &num; // 変数のアドレスを代入

char str[10]; // char型の配列の宣言

char \*pstr1; // char型のポインタの宣言

char \*pstr2;

pstr1 = &str[0]; // アドレスの代入

pstr2 = str; // 上の行と同じ意味

関数ポインタとは関数が格納されたアドレスのことである。「関数名(引数);」と同じ意味で「関数ポインタ名(引数);」と書くことができる。関数ポインタの中に異なるアドレスを入れることによって1つの関数ポインタによって複数の関数を用いることができる。関数ポインタは「返り値の型 (\*func)(引数1の型, 引数2の型);」と宣言して「func \*名前;」と初期化をして「名前 = &関数名;」と代入して使うことができる。以下に2つの関数を1つの関数ポインタで使用する例を示す。

int (\*func) (int, int); // 関数ポインタの型宣言

int add(int a, int b){ return a + b; } // 関数1

int times(int a, int b){ return a \* b; } // 関数2

int main(){

　　int num1 = 2; int num2 = 3; int ans;

　　func \*pfunc; // 関数ポインタの初期化

　　pfunc = &add; // 関数1のアドレスを代入

　　ans = pfunc(num1, num2); // 5 // 関数1を使った

　　pfunc = &times; // 関数2のアドレスを代入

　　ans = pfunc(num1, num2); // 6 // 関数2を使った

}

[malloc、free]

mallocとは構造体のメモリを動的に確保するために用いる関数である。動的にメモリを扱ったり確保したメモリ領域について関数を使って操作したりする際に用いる。確保したメモリは解放しない限りメモリを使ってしまうので使い終わった後にfree関数を用いて必ず解放する必要がある。

C言語においてはヘッダーファイル「stdlib.h」内に宣言されているのでincludeする必要がある。main関数の外に以下のように宣言する。

#include <stdlib.h>

構造体を動的メモリ確保によって使用する場合は以下の例のようにする。以下の実装ではmalloc関数を用いて構造体変数sのサイズ文だけメモリを確保してポインタにキャストしている。使った後にはfree関数を用いて使った動的メモリの解放をしている。

struct s{

　　int num; // 整数

　　char \*str; // 文字列

}

int main(){

　　s \*entity; // ポインタ型の変数

　　entity = (s\*)malloc(sizeof(s)); // 動的メモリの確保

　　entity->num = 0; // 変数の初期化

　　entity->str = (char\*)malloc(sizeof(char) \* 32); // 変数の初期化

　　// 処理

　　free(entity->num); // メモリの解放

　　free(entity); // メモリの解放

　　return 0;

}

[make、Makefile]

makeコマンドとはLinux上でコンパイルのルールを記述したMakefileを用いて、プログラムのビルド作業を自動化するツールである。コマンドを自由に作ることができるという利点がある。ビルドするプログラムと同じ階層に「Makefile」という名前のファイルを作って任意の変数や依存関係などのルールを書くことによってMakefileを作ることができる。以下に具体的な設定の仕方を示す。

**① Makefileの変数の設定**

Makefileの中で用いる変数の設定は「変数名 = 定義」とする。変数名は慣例的に大文字を使う。

**② Makefile内のルールの設定**

・任意に決められるコマンド名に:をつけて改行してタブを使ってからルールの内容を書く。

・変数を呼び出す際には「$(変数名)」とする

・makeコマンドを実行するとルール内容まで表示されるが、先頭に「@」をつけることで非表示にすることができる

・必要なファイルがあるときは「コマンド名:ファイル名」とする

・「コマンド名：コマンド1 コマンド2 … コマンドn」とすることで複数のコマンドを連続して呼び出すことができる。

**③ makeの使い方**

使う際にはMakefileを定義したパスと同じ場所で「make」とすることで自分で定義したルールを適応することができる。**[3]**

3. どのタスクまで行ったか

記載されているタスクは全て実施した。AIに関してはマス目ごとの評価点で次の手を決めるのと候補数（後ほど説明）を評価点に次の手を決める2つのプログラムを作った。以下にタスクの内容とファイル名の関係を示す。

タスク2：task2\_client.c, player.c

タスク3：task3\_server.c

タスク4 マルチスレッド：task4\_1\_server\_pthread.c

タスク4 DOS攻撃対策：task4\_2\_server\_pass.c

タスク4 マス目AI：task4\_3\_server\_AI1.c

タスク4 候補数AI：task4\_3\_server\_AI2.c

4. 実装方法（アイデア・設計）

【タスク2】

① player.cファイルの変更点

○connect\_to関数

・echo\_client.cからコピーした。サーバとの接続をするために用いる。必要なヘッダーファイルをincludeした。

○remote\_play関数

・サーバにMOVEリクエストを送り、mpに次の手を入れるための関数

・サーバには”MOVE 盤面(64個) 手番\r\n”と送るのでbufには73バイト分確保している

・サーバからは”D3\r\n”のように次の手が返ってくるのでbuf\_returnには4バイト分確保する配列を用意している

・盤面と手番の色の状態を見て、置けるところがあるならサーバにMOVEリクエストを送る。現在の手番の置けるところがないかつ次の手番の置けるところがあるときにはパスを選択する。

・write\_boardをしてbufの中に先頭から64バイト分に盤面を入れる。その後、後ろに5つずらし、先頭に”MOVE ”をつける。その後ろには手番がWHITEのときは”O”、BLACKのときは”X”をつける。最後には”\r\n”をつける。

・fwriteでサーバにbufに入れたMOVEリクエストを送る

・freadでサーバから返ってきた次の手をbuf\_returnに入れる

・read\_moveで次の手をmpに格納する

○make\_remote\_player関数

・相手プレイヤーの次の手を考える際にサーバに接続するための関数

・\_hidden\_stateにはremort\_play関数で用いるためにファイルのポインタを入れる

・\_playにはremote\_playのポインタを入れる

・\_clean\_upにはファイルを閉じてサーバにQUITリクエストを送るmyfclose\_quit関数のポインタを入れる

② task2\_clientファイルの作成（single\_play.cとの変更点）

○main関数

・変数「is\_net」を作り、相手プレイヤーをサーバに接続するかどうかのフラグとする。オプション引数の中に「-c」があるときにis\_net = 1とする。このときはmake\_remote\_player関数を呼び出す。引数はホスト名とポート番号としている。

【タスク3】

・echo\_server.cからコピーしてserve関数を変更する

〇serve関数

・73バイト分のbufを用意してクライアントからのリクエストを受け取る。受け取ったリクエストをターミナルに表示するために71バイト分のhyouziに格納する。\rと\nの分だけ2バイト少なくしている。

・MOVEリクエスト、NOPEリクエスト、QUITリクエストを受け取ったときでそれぞれ分岐させる。

・NOPEリクエストのときはクライアントに「OK\r\n」と返す

・QUITリクエストのときはクライアントには何も返さずにwhileループを抜ける

・MOVEリクエストのときは合法手を探すlegal\_movesを使うためにboard、colorを受け取ったリクエストから取り出す。合法手は最大64手あるのでその分の大きさの配列buf\_returnと何手あるかを格納するlenを定義する。合法手の中からランダムに1つ手を選び、後ろに\r\nを付けてクライアントに返す。

【タスク4】

○サーバのマルチスレッド化

・echo\_server\_pthread.cからコピーしてserve関数をタスク3で作ったserve関数に置き換えることによって実装した

○攻撃対策

・バッファオーバーフローに備えるためにecho\_serverと同じようにserve関数の最初にsetbuf関数の第2引数にNULLを指定してバッファリングオフを実装した。また、開いたファイルは関数の最後に必ず閉じるようにしている。

・DOS攻撃に備えるためにパスワード認証を設ける。クライアント側で-pを引数に指定することでパスワード認証に対応した関数を呼び出すようにした。実装を変更したところについては以下に示す。クライアントからサーバにアクセスがあった時点でサーバでランダムにパスワードを生成してサーバに表示する。クライアント側でパスワードの入力を促す。入力されたパスワードをサーバに送り正しいパスワードかどうか確認する。一致しない場合はクライアントとの接続を終了して\_playの中に-1を入れてエラー終了させて、クライアント側のmain関数の中でgame関数の返り値を-2にする。このときはGame finishedのメッセージや終了時の盤面を表示しないようにする。

○サーバの手をAIで強化する

・リバーシの置く場所それぞれに評価点をつけて評価点が最も高いところを選択するプログラムを作った。serve関数の中で以前は可能な手の中からランダムに選んでいたところを可能な手の中から評価点が最も高い手を選択するbest\_choice関数を作って変更した。best\_choice関数では現在の盤面のみ見て、その中で元々設定したマス目ごとの評価点の中で最も高いところを選択させている。評価点は以下の論文を参考にした。[4]

・次打てる場所の数を候補数と名付けると、一般的にリバーシにおいては自分の候補数が多くて相手の候補数が少なくなるようにすれば強いと言われている。現在の盤面の中で「自分の候補数 – 相手の候補数」が最も大きくなる手を選ぶようにした。具体的にはmost\_kouho関数の中で現在可能な手の中から最善手を選ぶプログラムを書いた。この関数の中で現在の盤面から次の手を1つずつ打ってそのときの自分の色の候補数から相手の色の候補数を引いた数を評価点として一番評価点が高くなるときの手を採用するようにした。

5. 工夫点・改善点

○工夫点

・1回目の実験日では仕様や書き方があまり理解できずにタスク2の序盤で終了してしまったが、2回目の実験日までに自分では解決しなくなるまで進めた。タスク2とタスク3のクライアントとサーバ間の実装をして、1往復はできるようになった。しかし、サーバからクライアントの通信の後にゲームが強制的に終了してしまうというバグが自分では解決策が見つからなかったので次回TAに聞いて解決することにした。実際に聞いて一緒にデバッグをしていたところ、クライアント側で元からある他の関数からコピーしてそのままになっていたremort\_play関数の最後でfcloseをしていることが原因だった。あんなに苦労していたのにこの1行をコメントアウトするだけで解決したので拍子抜けした。

・タスク4のDos攻撃対策としてパスワード認証をつけることを考えた。サーバの出力を見ないとパスワードがわからないという点において、実用性があるかどうかは正直微妙かもしれないが自分で試行錯誤して実装の仕方を考えた。特にパスワードが間違っていた際の挙動について、ゲームを終了させるのとゲームに関する情報を表示しないようにするのに苦労した。

・下の改善点に書いたように現在の盤面を見て置く場所の評価点のみに基づくAIでは勝率3割に満たなかったが、自分と相手の候補数に基づくAIでは50戦42勝で勝率8割まであげることができた。

○改善点

・パスワードが一致しているかどうかを判断するのに苦労した。生成したパスワードを格納するcharとクライアントから送られたパスワードを格納するcharの一致のさせかたがわからなかった。出力して目で確認して同じであるように見えるのにもかかわらず同じと判断されなかったり、charの文字列を4と指定しているのに文字列の長さが変わっていたりした。C言語の文字列の扱いがよくわかっていないので先頭から4文字のcharが1つずつ正しいか確認するという怪しい方法で判断してしまった。

・パスワード認証が失敗した際にクライアント側に盤面を表示させたくなかったので、このファイルを実行するときはクライアント側は必ず後手になるように固定してしまった。これはパスワード認証をmake\_remort\_playerの中でやっていてクライアント側が先手の場合にサーバに手を送った後にしか接続を切ることができないことが原因である。本当は手番はランダムにしたかったが方法がわからなかった。

・AIの実装については現在の状態しか見ていない上に評価点の決め方がひっくり返る石のことは考えずに置いた場所のみしか考えていないのでほんの少しだけ知識をつけた小学生程度の実力しか出せていないはずである。本来は数手先までみてその評価点の合計と相手の打てる手の評価点の合計が低くなるようにするなど色々やれることはあったが今回は一番簡単だと考えた実装にとどめた。ランダムな手と20回の対戦をしたところAIは7勝しかできず勝率は5割を超えることはないのでかなり弱いAIかもしれない。

6. 実行結果（実行例など）

テキスト

自動的に生成された説明

図 1．サーバとクライアントの通信（タスク2、タスク3）

図1はタスク2で実装したクライアント側とタスク3で実装したサーバ側を通信させているターミナルである。図1を含む以下の図では全て左半分でサーバ、右半分でクライアントを実行している。サーバの下の方（図1の左下）では実行し始めを映し、上の方（図1の左上）で実行し終わるところを映している。クライアントでは手で実際に入力している様子を映している。

グラフィカル ユーザー インターフェイス, テキスト

自動的に生成された説明

図 2．マルチスレッドの通信（タスク4）

図2はタスク4で実装したサーバをマルチスレッドに対応させたターミナルである。図2の左側でクライアントを2つ起動させて、1つのサーバと接続させている。今回は1つめのクライアントで2手進めてから2つめのクライアントとの接続を開始して、以降では交互にクライアントから手を打たせたときの様子である。

テキスト

自動的に生成された説明

図 3．DOS攻撃対策のパスワード認証（タスク4）

図3はタスク4のDOS攻撃対策としてパスワード認証を設けた際のターミナルである。クライアントからの接続があった際にサーバ側でパスワードを表示してクライアントにパスワードの入力を求める。入力が間違っている場合にはクライアントの通信を切断して、入力が正しい場合は今まで同様にリバーシを始めている。

テキスト

低い精度で自動的に生成された説明

図 4．候補数AIの複数回実行（タスク4）

図4はタスク4のサーバ側のAIの実装に関して候補数を元に評価点をつける方式を採用した際のターミナルである。クライアント側で引数に「-r」を入れることによって手入力ではなくランダムに打てるようにtask2\_client.cを変更した。図4では3回実行して3回ともAIが勝っている。体感では8割ほどAIが勝っているようだ。

**<参考文献>**

[1]実験指針C

[2]栢木厚, “令和4年 イメージ&クレバー方式でよくわかる 栢木先生の基本情報技術者教室”, 藤沢奈緒美, 株式会社技術評論社, 2022

[3][**@mizcii**](https://qiita.com/mizcii), Qiita“【初心者向け】Makefile入門”,

https://qiita.com/mizcii/items/cfbd2aa17f6b7517c37f

[4]塩田好, “リバーシの評価関数について”, 近畿大学理工学部情報学科卒業研究報告書(2012)