01. WinSock でのプログラミング準備

WinSock は WindowsSockets の略称で Windows 環境でのソケットによる通信プログラムを作成するための API である。

WinSock でのプログラムを作成するためには、まず下記のヘッダーの Include とライブラリの追加をおこなう必要がある。

```
#include<winsock2.h>
#pragma comment(lib,"ws2_32.lib")
```

WinSock を利用したプログラムを作成する場合は、最初に WinSock の初期化を記述する。WinSock の初期化は WSAStartup 関数を使用する。

```
WSAData wsaData;
int ret = WSAStartup(MAKEWORD(2,2),&wsaData);
if(ret != 0)
{
	printf("WinSockの初期化に失敗\n");
}
```

WSAStartup 関数の一つ目の引数は利用する WinSock のバージョンを指定する。指定できるバージョンとして、1.0・1.1・2.0・2.1・2.2の 5 種類がある。二つ目の引数には WSAStartup の初期化結果情報を格納するための WSAData 構造体を指定する。結果情報には初期化に成功したバージョンや利用できる最高のバージョンなどが格納される。

WinSock の利用を終了する場合は WSACleanup 関数を使用する。

```
WSACleanup();
```

WSAStartup と WSACleanup は最初と最後に 1 度だけ実行すればいいため、WinSock を利用したプログラムを作成する場合はプログラムの最初と最後に加えると良い。

02. TCP と UDP

ネットワークでデータをやり取りするためには、通信をするためのルールを決めなくてはならない。この通信するための手順や規約の集合を**通信プロトコル**という。プロトコルというと TCP/IP という言葉を聞いた事はあると思うが、これは TCP と IP という二つのプロトコルから名づけられたプロトコル群である。

OSI 参照モデルやネットワークの仕組みはここでは解説をおこなわないが、一般的な通信のプログラムを作成する際に IP より下位の層（IP は第 3 層ネットワーク層）をそのまま使うことはなく、プログラムで利用するプロトコルは IP の上位層（第4層トランスポート層）に位置する **TCP** と **UDP** である。

IP より下位の層で相手のコンピュータまでの通信を行い、データ（パケット）を送信する。ここまでの通信ではただ相手のコンピュータにデータを送るだけで、通信の順番も正確性も保障しない。

TCP と UDP はここから届いたデータをプログラム（アプリケーション）へ渡す役割を担う。

TCP は信頼性の高い通信プロトコルで、パケットの喪失や重複、順序の問題を検出し、喪失したパケットの再送をおこない、順序を正しく並べ替え、プログラムへ渡す。対して UDP では、パケットの喪失や重複、順序の問題を保障しない。それらに対応が必要な場合はプログラム側でおこなう必要がある。

TCP ではパケットの喪失時の再送要求などが必要になると、データの受信に遅延が生じるが、UDP ではそういったオーバーヘッドがないため非常に高速である。

ゲームプログラミングでは TCP と UDP の使い分けとして、ボードゲームなどのようにターン制で進行するゲームでは TCP、アクションゲームや格闘ゲームなどのリアルタイムでのレスポンスが重要なゲームでは UDP が主に使われる。また両方利用する場合もあり、その場合は例えば確実性の必要な通信（ユーザーのログインメッセージなど）を TCP で、

喪失が起きても問題がなくリアルタイムな更新の必要な通信（毎フレームごとの更新情報など）をUDPでおこなう。

03. ソケット

ソケットはプログラムからネットワークへデータの入出力をおこなうためのインターフェースであり、プログラムではソケットを作成し、作成したソケットに対してデータを書き込んだり、読み込んだりすることで通信をおこなう。

プログラムでソケットを作成するにはsocket関数を使用する。

```
SOCKET tsock = socket(AF_INET, SOCK_STREAM, 0);
if(tsock == INVALID_SOCKET)
{
	printf("Error SockCreate\n");
	return;
}
```

socket関数は戻り値として作成したソケットをSOCKET型の変数として返す。一つ目の引数は通信をおこなうアドレスの種類を指定する。ネットワークゲームで使うのはインターネットアドレスを指定するAF_INETとAF_INET6の2種類が主で、それぞれIPv4のアドレスとIPv6のアドレスを使用することを指す。二つ目の引数はソケットのタイプを指定できる、TCPならSOCK_STREAM、UDPならSOCK_DGRAMを指定する。三つ目の引数はプロトコルを指定する。特に指定をおこなわない場合は0を指定する。

ソケットが不要になればclosesocket関数で作成したソケットを閉じる。

```
closesocket(tsock);
```

04. TCPでのソケット通信（サーバー（ホスト）側）

TCPによる通信をおこなうプログラムはホストとクライアントで手順が異なる。ここではホスト側の通信プログラムを紹介する。

TCPのホスト側のプログラムは下記の手順になる。

1.socketでソケットを作成

2.bindでソケットにポート番号やアドレスを割り当てる

3.listenで接続待ちソケットとして設定する

4.acceptでクライアントからの接続を待機する

5.sendで送信、recvで受信をおこなう

6.closesocketで接続したソケットを終了する

7.closesocketで接続待ちソケットを終了する

なお、7の接続待ちソケットの終了は、クライアントの接続確認後にすぐに終了してもかまわない。

TCPでの通信をおこなうため、socketでのソケットの作成はタイプにSOCK_STERAMを指定した作成をする。

次のbind関数は作成したソケットに**ポート番号**などを割り当てる関数である。ポート番号はデータをプログラムに届けるための識別番号で0～65535までの数字を指定する。ただし独自のネットワークプログラムを作成する場合は、ウェルノウンポートと呼ばれる主要なプロトコルで使用されるポート番号はさける必要がある。

bindが終了するとポート番号が割り当てられソケットの使用準備が整った状態になる。次にクライアントからの接続待機状態にするための準備としてlisten関数を実行する。これで接続待機のための準備が完了しacceptを実行して実際にクライアントからの接続を待機する。accept関数はクライアントからの接続があるまで処理をブロックする。そして接続があれば接続されたソケットを戻り値として返す。

accept に成功すると、戻り値として返ったソケットを使用してクライアントと通信をおこなうことができる。

```
#include<stdio.h>

//ソケット通信用
#include<winsock2.h>
#pragma comment(lib,"ws2_32.lib")

void main(void)
{
	//WSAStartUp
	WSAData wsaData;
	int ret = WSAStartup(MAKEWORD(2,2),&wsaData);
	if(ret != 0)
	{
		printf("WinSockの初期化に失敗\n");
	}

	//ソケットの作成
	SOCKET tsock = socket(AF_INET, SOCK_STREAM, 0);
	if(tsock == INVALID_SOCKET)
	{
		printf("Error SockCreate\n");
		return;
	}
	//アドレス構造体
	struct sockaddr_in tmp_addr;
	memset(&tmp_addr,0,sizeof(struct sockaddr_in));
	//ネットワークのデータを設定
	tmp_addr.sin_family = AF_INET;
	tmp_addr.sin_port = htons(18900);
	tmp_addr.sin_addr.s_addr = ADDR_ANY;
	//アドレスとソケットをバインド
	if((bind(tsock,(struct sockaddr*)&tmp_addr,sizeof(struct sockaddr_in))) == SOCKET_ERROR)
	{
		printf("Error SockBind\n");
		closesocket(tsock);
		return;
	}
	//接続待機状態にする
	if((listen(tsock,SOMAXCONN)) == SOCKET_ERROR)
	{
		printf("Error SockListen\n");
		closesocket(tsock);
		return;
	}
	//接続してきたアドレス情報
	SOCKADDR_IN ta;
	//アドレス構造体のサイズ
	int addrin_size = sizeof(SOCKADDR_IN);
	//接続待ち
	printf("Wait Accept\n");
	SOCKET ts = accept(tsock,(struct sockaddr*)&ta,&addrin_size);
	printf("Client IP %s\n",inet_ntoa(ta.sin_addr));

	//受信
	while(TRUE)
	{
		int Data;
```

```
                int s = recv(ts,(char*)&Data,sizeof(int),0);
                printf("%d Bytes Receive\n",s);
                printf("%d Receive\n",Data);
                if(Data == -1)
                {
                        printf("-1 Receive End\n");
                        break;
                }
        }
        //後始末
        closesocket(ts);
        closesocket(tsock);
        WSACleanup();
        return;
}
```

05. TCP でのソケット通信（クライアント側）

クライアント側の通信プログラムを紹介する。

TCP のクライアント側のプログラムは下記の手順になる。

1.socket でソケットを作成

2.connect でホストに接続をする

3.send で送信、recv で受信をおこなう

4.closesocket で接続したソケットを終了する

TCP でのクライアントの処理はソケットの作成までは同じだが、bind 関数をおこなわずに connect 関数でサーバーへの接続処理をおこなう。connect が成功すれば指定したソケットは相手のソケットと接続状態になり通信をおこなうことができる。

```
#include<stdio.h>

//ソケット通信用
#include<winsock2.h>
#pragma comment(lib,"ws2_32.lib")

void main(void)
{
        //WSAStartUp
        WSAData wsaData;
        int ret = WSAStartup(MAKEWORD(2,2),&wsaData);
        if(ret != 0)
        {
                printf("WinSockの初期化に失敗\n");
        }

        //ソケットの作成
        SOCKET tsock = socket(AF_INET, SOCK_STREAM, 0);
        if(tsock == INVALID_SOCKET)
        {
                printf("Error SockCreate\n");
                return;
        }
        //接続IPを入力
        char ip[256];
        printf("Press Connection IP : ");
        scanf("%s",ip);
```

```
	//アドレス構造体
	struct sockaddr_in tmp_addr;
	memset(&tmp_addr,0,sizeof(struct sockaddr_in));
	//ネットワークのデータを設定
	tmp_addr.sin_family = AF_INET;
	tmp_addr.sin_port = htons(18900);
	tmp_addr.sin_addr.s_addr = inet_addr(ip);
	if(connect(tsock,(struct sockaddr*)&tmp_addr,sizeof(struct sockaddr_in)) == SOCKET_ERROR)
	{
		closesocket(tsock);
		printf("Error Connect\n");
		return;
	}
	//送信
	while(TRUE)
	{
		int Data;
		printf("Press Send INT Data : ");
		scanf("%d",&Data);
		int s = send(tsock,(char*)&Data,sizeof(int),0);
		printf("%d Bytes Send\n",s);
		printf("%d Send\n",Data);
		if(Data == -1)
		{
			printf("-1 Send End\n");
			break;
		}
	}
	//後始末
	closesocket(tsock);
	WSACleanup();
	return;
}
```

06. UDP での受信

UDP による通信をおこなうプログラムでは、TCP による通信のように接続の確立が不要になる。そのため基本的な通信プログラムではホスト・クライアントの変更はない。ここでは UDP でデータの受信をおこなうプログラムを紹介する。

UDP の受信プログラムは下記の手順になる。

1.socket でソケットを作成

2.bind でソケットにポート番号やアドレスを割り当てる

3.recvfrom で受信をおこなう

4.closesocket で接続したソケットを終了する

UDP では特定の相手と接続を行い通信をするわけではない。そのため実際に受信をおこなったデータが誰から届いたのかをアプリケーションで知る必要がある。そのため受信には recvfrom 関数を使用する。recvfrom 関数は recv 関数に加えて送られてきた相手のアドレス情報を格納するため、その情報から相手を判断する。

```
#include<stdio.h>

//ソケット通信用
#include<winsock2.h>
#pragma comment(lib,"ws2_32.lib")
```

```
void main(void)
{
	//WSAStartUp
	WSAData wsaData;
	int ret = WSAStartup(MAKEWORD(2,2),&wsaData);
	if(ret != 0)
	{
		printf("WinSockの初期化に失敗\n");
	}

	//ソケットの作成
	SOCKET tsock = socket(AF_INET, SOCK_DGRAM, 0);
	if(tsock == INVALID_SOCKET)
	{
		printf("Error SockCreate\n");
		return;
	}
	//アドレス構造体
	struct sockaddr_in tmp_addr;
	memset(&tmp_addr,0,sizeof(struct sockaddr_in));
	//ネットワークのデータを設定
	tmp_addr.sin_family = AF_INET;
	tmp_addr.sin_port = htons(18900);
	tmp_addr.sin_addr.s_addr = ADDR_ANY;
	//アドレスとソケットをバインド
	if((bind(tsock,(struct sockaddr*)&tmp_addr,sizeof(struct sockaddr_in))) == SOCKET_ERROR)
	{
		printf("Error SockBind\n");
		closesocket(tsock);
		return;
	}
	//受信時のアドレス構造体
	SOCKADDR_IN ta;
	//アドレス構造体のサイズ
	int addrin_size = sizeof(SOCKADDR_IN);
	//受信
	printf("Wait Receive\n");
	while(TRUE)
	{
		int Data;
		int s = recvfrom(tsock,(char*)&Data,sizeof(int),0,(struct sockaddr*)&ta,&addrin_size);
		printf("%d Bytes Receive\n",s);
		printf("%d Receive\n",Data);
		printf("%d : %d\n",ntohs(ta.sin_port),ntohs(tmp_addr.sin_port));
		printf("%s : %s IP\n",inet_ntoa(ta.sin_addr),inet_ntoa(tmp_addr.sin_addr));
		if(Data == -1)
		{
			printf("-1 Receive End\n");
			break;
		}
	}
	//後始末
	closesocket(tsock);
	WSACleanup();
	return;
}
```

07. UDP での送信

UDP でデータの送信をおこなうプログラムを紹介する。

UDP の送信プログラムは下記の手順になる。

1.socket でソケットを作成

2.sendto で送信をおこなう

3.closesocket で接続したソケットを終了する

UDP では特定の相手と接続を行い通信をするわけではない。そのため送信処理では sendto 関数を使用して毎回送信先のアドレス情報を指定して送信をおこなう。

```
#include<stdio.h>

//ソケット通信用
#include<winsock2.h>
#pragma comment(lib,"ws2_32.lib")

void main(void)
{
	//WSAStartUp
	WSAData wsaData;
	int ret = WSAStartup(MAKEWORD(2,2),&wsaData);
	if(ret != 0)
	{
		printf("WinSockの初期化に失敗\n");
	}

	//ソケットの作成
	SOCKET tsock = socket(AF_INET, SOCK_DGRAM, 0);
	if(tsock == INVALID_SOCKET)
	{
		printf("Error SockCreate\n");
		return;
	}
	//接続IPを入力
	char ip[256];
	printf("Press Send IP : ");
	scanf("%s",ip);
	//アドレス構造体
	struct sockaddr_in tmp_addr;
	memset(&tmp_addr,0,sizeof(struct sockaddr_in));
	//ネットワークのデータを設定
	tmp_addr.sin_family = AF_INET;
	tmp_addr.sin_port = htons(18900);
	tmp_addr.sin_addr.s_addr = inet_addr(ip);
	//送信
	while(TRUE)
	{
		int Data;
		printf("Press Send INT Data : ");
		scanf("%d",&Data);
		int s = sendto(tsock,(char*)&Data,sizeof(int),0,(struct sockaddr *)&tmp_addr,sizeof(tmp_addr));
		printf("%d Bytes Send\n",s);
		printf("%d Send\n",Data);
		if(Data == -1)
		{
			printf("-1 Send End\n");
```

```
                        break;
                }
        }
        //後始末
        closesocket(tsock);
        WSACleanup();
        return;
}
```