第２章　ソケット通信

２．１　ソケット通信

(1) ソケットとは

コンピュータネットワークの世界では、異なるアーキテクチャ（内部構造)、異なるオペレーティングシステムの端末が同時に送受信できるようにルール（通信規格）が定められています。このルールのことを通信プロトコルといい、世界で一番使用されている通信プロトコルが TCP/IP です。TCP は TransmissionControlProtocol の略称であり相手端末との通信手段の確立におけるルールが定められています。IP とは InternetProtocol の略称であり、TCP/IP 通信におけるアドレス（IP アドレス）にかかるルールが定められています。TCP/IP 通信におけるデータ送受信を行うためにデータの出入り口としてイメージしたものが**ソケット**であり、ソケットを使用した通信手段（プログラム）のことを**ソケット通信**といいます。

ソケット通信ではプログラム言語の種類を問わず、共通して使用される概念です。つまりプログラム言語がどうあれ、TCP/IP というルールで通信をするためにはソケット経由でデータ送受信をする必要があります。

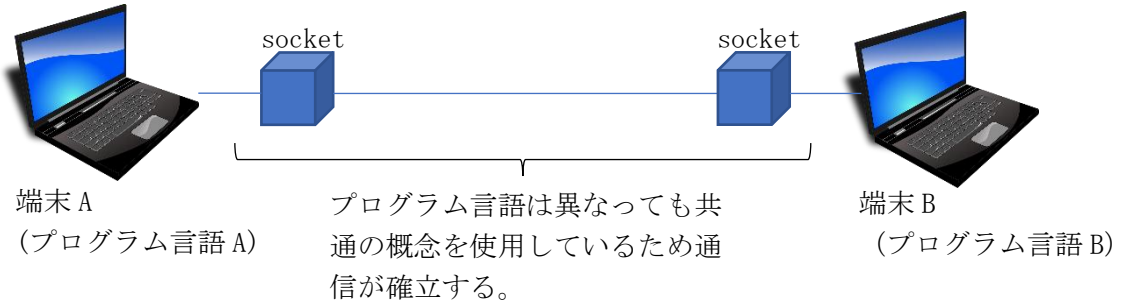


図２．１　ソケット通信イメージ

２．２　ソケット通信における通信回線の確立

ソケット通信では通信回線を受け付ける側（サーバ側）と通信回線の確立を要求する側（クライアント側）によってプログラムの流れが下記のようになっています。

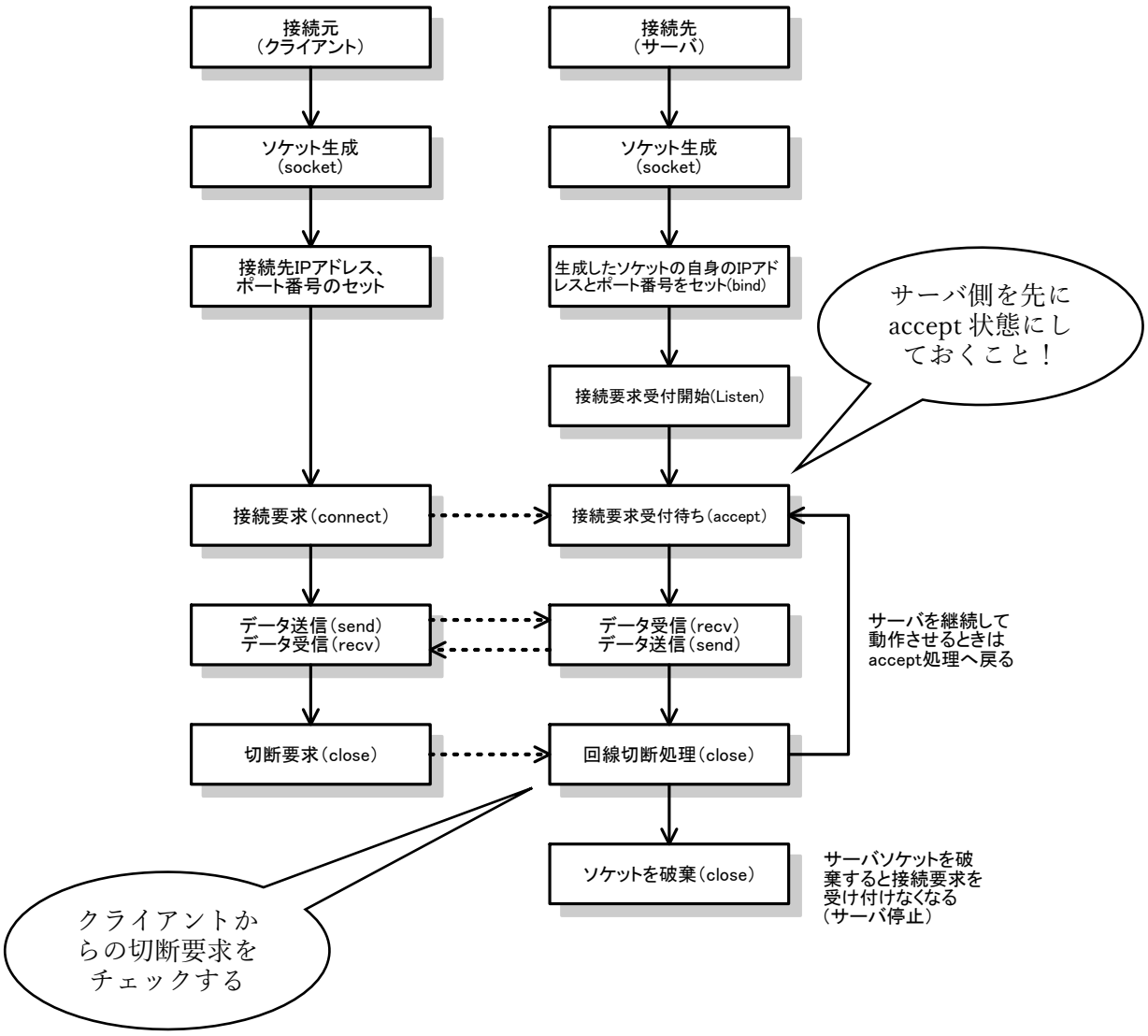


図２．２　ソケットプログラムの流れ

◆　ソケットが複数？？

サーバは accept 処理を利用することで接続した端末ごとの通信用ソケットが生成されます．　１度に複数のサーバが接続することが予想されるときは複数の通信用ソケットの生成が必要となります．ソケット通信を利用してオリジナルサーバを作成する場合は複数のクライアントと非同期に通信をしなくてはならないため、マルチスレッドプログラミングなど何らかの並列処理が必要となるケースがあります．

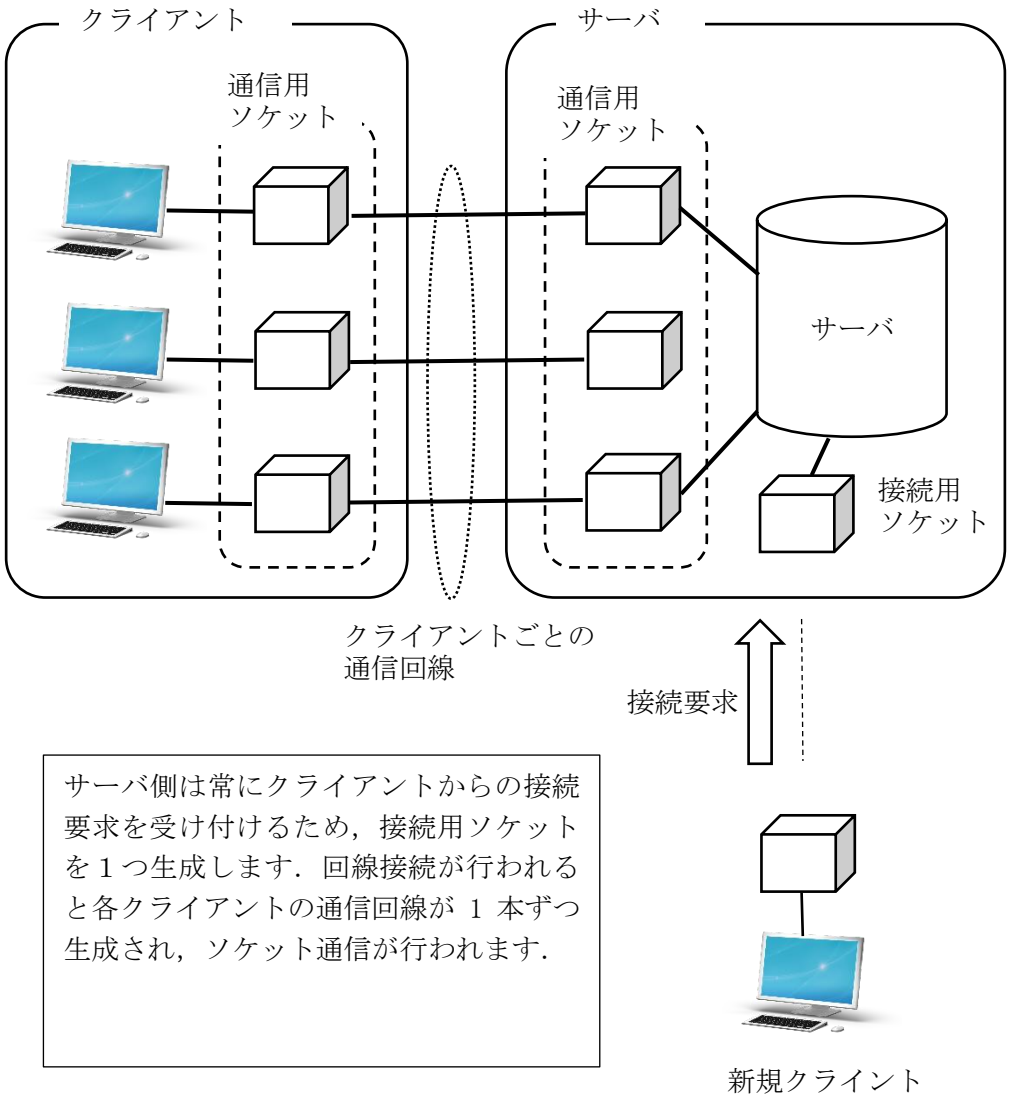


図２．３　回線接続におけるイメージ図

２．３　ソケット通信ライブラリ

ソケット通信で必要なライブラリは Python 言語標準のため特にインストールする必要はありません．下記のようにライブラリをインポートさせます．

(1)ライブラリインポート

```
import  socket
```

(2)ライブラリ解説

①ソケット生成

関数名	socket.socket(family, type)
引数	family: 　socket.AF_INET…IPv4 を使用する 　socket.AF_INET6…IPv6 を使用する type: 　socket.SOCK_STREAM…TCP 通信を使用する 　socket.SOCK_DGRAM…UDP 通信を使用する
戻り値	ソケット通信で利用するインスタンス
概要	ソケットを生成する

使用例
(server)
	import socket
	
	#ソケット生成(socket)
	server = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
	
	…

②bind

関数名	***socket***.bind((IPAddress, port))
引数	IPAddress: 　ソケットに設定する IP アドレスを**文字列で指定する** port: 　ソケット通信で使用するポート番号を指定する
戻り値	なし
概要	生成したソケットに対して IP アドレスとポート番号を設定する（bind 処理）

使用例
(server)
	import socket
	
	#ソケット生成(socket)
	server = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
	
	#bind 処理
	server.bind((’10.0.1.10’, 8000))
	…

③listen

関数名	***socket***.listen(max_client)
引数	max_client: 　同時に接続受付するクライアントの最大数(必ず１以上にすること)
戻り値	なし
概要	同時接続できるクライアント数を設定し，ソケットを接続受付できる状態にする(Listen 処理)

使用例
(server)
	import socket
	
	#ソケット生成(socket)
	server = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
	
	#bind 処理
	server.bind((’10.0.1.10’, 8000))
	
	#listen 処理
	server.listen(1)
	
	…

④accept

関数名	client , port = ***socket***.accept( )
引数	なし
戻り値	client : 接続したクライアントオブジェクト port : クライアントが提供したポート番号
概要	クライアントの接続待ち状態になる(accept 処理).

使用例
(server)
	import socket
	
	#ソケット生成(socket)
	server=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
	
	#bind 処理
	server.bind((’10.0.1.10’, 8000))
	
	#listen 処理
	server.listen(1)
	
	while True:
	　　client , port = server.accept()
	　　…

⑤connect

関数名	***socket***.connect((IPAddress, port))
引数	IPAddress: 接続したいサーバの IP アドレス port : サーバのポート番号
戻り値	なし
概要	接続先（サーバ）に接続要求を出す（connect 処理）

使用例
(client)
	import socket
	
	#ソケット生成(socket)
	client = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
	
	#connect 処理
	client.connect((’10.0.1.10’, 8000))
	
	…

⑥send

関数名	***socket***.send(data)
引数	data:送信データ（バイナリデータ（bytes 型)）
戻り値	送信したバイト数
概要	接続先（サーバもしくはクライアント）にバイナリデータを送信する

使用例
(client)
	import socket
	
	#ソケット生成(socket)
	client = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
	
	#connect 処理
	client.connect((’10.0.1.10’, 8000))
	
	#send（バイナリデータ送信）
	clinet.send(b’Hello’)

備考	ソケット経由で送信するデータはバイナリデータになります．python 上でバイナリコードを送るには一度文字列型へ変換し，decode メソッドを使用する必要があります.

データの表現方法

コンピュータが扱うデータ（数値）は 2 進数ですが，2 進数の数値が組み合わされたデータ形式は大きく次の３つに分けられます.

１．バイナリデータ

バイナリデータとは単純な 2 進数としての形式のことです．例えば 10 進数「100」は 2 進数（バイナリデータ）では「0110 0100」となります．10 進数から 2 進数の計算は下記のように計算します.

```
2) 100
2)  50 …  0
2)  25 …  0
2)  12 …  1
2)   6 …  0
2)   3 …  0
     1 …  1
```

図２．４　10 進数→2 進数変換

２．BCD コード

BCD コードとは「Binary Code Decimal」の略称であり２進化１０進数とも呼ばれる数値の表現方法です．通常の２進数変換方法と異なり各桁ごとに２進数に変換します．１０進数で「100」は BCD コードで「0001 0000 0000」となります.

３．ASCII コード

ASCII コードとは文字やアルファベット、数値、記号を収録した文字コードのひとつ．１文字を 8bit で表現し「ASCII 制御コード」として世界的に普及しています．10 進数で「100」は ASCII コードでは「0x313030」となります.

⑦sendall

関数名	***socket***.sendall(data)
引数	data:送信データ（バイナリデータ（bytes 型)）
戻り値	正常終了の場合：None 異常発生の場合：例外が発生（例外要因：InterruptedError）
概要	接続先（サーバもしくはクライアント）にバイナリデータを送信する. すべてのデータが送信し終えるまで待ち状態となる.

使用例
(client)
	import socket
	
	#ソケット生成(socket)
	client = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
	
	#connect 処理
	client.connect((’10.0.1.10’, 8000))
	
	#送信したい文字をセット
	msg = ‘Hello’
	
	#send（送信）
	clinet.sendall(msg.encode())
	
	……

備考	ソケット経由で送信するデータはバイナリデータになります．python 上でバイナリコードを送るには一度文字列型へ変換し，encode メソッドを使用する必要があります.

【Python 言語におけるバイナリコードへの変換方法】

Python 言語では数値は整数型(int)、実数型(float)、文字列型(str)に分けられます．整数型、実数型では例え 2 進数で代入しても int 型として判断されソケットによる送信がされません.

```
val = 0b0110 0100  #2 進数で代入．しかし int 型として判断される
……
client.sendall( val )  #NG!!  int 型を直接送信することはできない
```

整数型データをバイナリ変換するには，まず文字列型へ変換し encode メソッドを使用することで変換ができます.

```
val = 0b0110 0100  #2 進数で代入．しかし int 型として判断される
……
client.sendall( str(val).encode )  #OK!!  str 型へ変換し，さらに encode メソッドを使用する
```

⑧recv

関数名	***socket***.recv(data)
引数	data:受信するバイト数
戻り値	受信したデータ（bytes 型）
概要	接続先（サーバもしくはクライアント）からバイナリデータを受信する.

使用例
(client)
	import socket
	
	#ソケット生成(socket)
	client = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
	
	#サーバへ接続（connect 処理）
	client.connect((’10.0.1.10’, 8000))
	
	#送信したい文字をセット
	msg = ‘Hello’
	
	#send（バイナリデータ送信（bytes 型））
	clinet.sendall(msg.encode())
	
	#recv（受信）
	recvData = client.recv(1024)
	
	#受信したデータを表示（バイナリデータから文字列型へ変換）
	print(recvData.decode())

使用例
(server)
	import socket
	
	#ソケット生成
	server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
	
	#ソケットに IP アドレス、ポート番号を設定（bind 処理）
	server.bind(('192.168.1.8',8000))
	
	#同時接続できる端末数設定（listen 処理）
	server.listen(1)
	
	while True:
	　　#接続待ち状態へ（accept 処理）
	　　client,addr=server.accept()
	
	　　with client: #close 処理をするために with 文を使用
	　　　　while True:
	　　　　　　data = client.recv(1024)
	
	　　　　　　if not data: #もし受信できなければ
	　　　　　　　　break　　#　ループを抜ける
	　　　　　　……

備考	ソケット経由で送信するデータはバイナリデータ(bypes 型)になります. python 上でバイナリコードを文字列型へ変換するには decode メソッドを使用する必要があります.

⑨close

関数名	***socket***.close( )
引数	なし
戻り値	なし
概要	生成したソケットを閉じる. クライアントの場合、サーバに対して回線切断要求を出す.

使用例
(client)
	import socket
	#ソケット生成(socket)
	client = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
	
	#サーバへ接続（connect 処理）
	client.connect((’10.0.1.10’, 8000))
	
	#送信したい文字をセット
	msg = ‘Hello’
	
	#send（バイナリデータ送信（bytes 型））
	clinet.sendall(msg.encode())
	
	#recv（受信）
	recvData = client.recv(1024)
	
	#受信したデータを表示（バイナリデータから文字列型へ変換）
	print(recvData.decode())
	
	#回線切断要求（close 処理）
	client.close()

備考	ソケット生成の際に with 文を使用すると close 処理の記述を省略することができます. （例） with socket.socket(socket.AF_INET,socket.SOCK_STREAM) as client:

(3)　ソケットの再使用でエラーが発生するとき
連続してソケット通信をしようとするとき，下記のようにエラーが発生することがあります。

```
pi@raspberrypi:~/work/socket $ python3 ./001_socket_Led.py
Traceback (most recent call last):
  File "./001_socket_Led.py", line 14, in <module>
    soc.bind(('10.0.199.25',8000))
OSError: [Errno 98] Address already in use
```

図２．５　ソケット通信におけるエラー

上記のエラーは，python 言語において close もしくは with を利用してソケットを閉じても，しばらくは同一のソケットが残ることにより，同一のアドレスを付与することができない旨のエラーです．通常はしばらく時間を経過すればソケットの再利用ができますが，短時間でソケットの再使用をする場合は *bind 処理をするまえに* 下記のソケットオプション処理を記述してください．

```
server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
```

生成したソケット　　ソケットオプション　　同一アドレスの再利用を許可

(4)　ネットワークバイトオーダとは
　コンピュータのメモリ内にデータをメモリに保存するとき，複数バイトのデータであれば上位バイトから保存する「**ビックエンディアン**」と下位バイトから保存する「**リトルエンディアン**」に分かれます．ビックエンディアンかリトルエンディアンかはＣＰＵによって変わります．

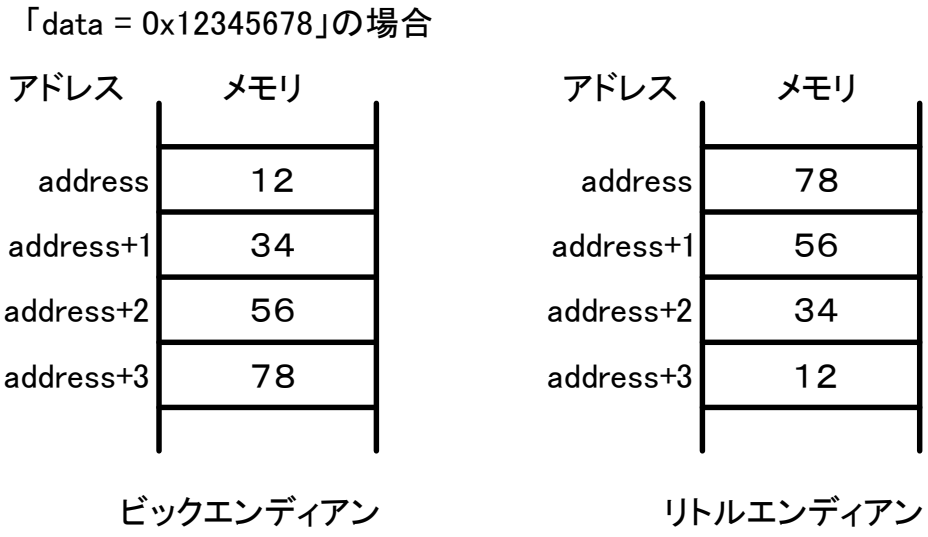


図２．６　ビックエンディアンとリトルエンディアンの違い

TCP/IP 通信は送受信するコンピュータにかかわらず共通のプロトコルとして決められているため，送受信するコンピュータのエンディアンが異なると，上手く通信ができても正しいデータ形式として読み取りができないこともあります（下位バイトと上位バイトが入れ替わるため）．

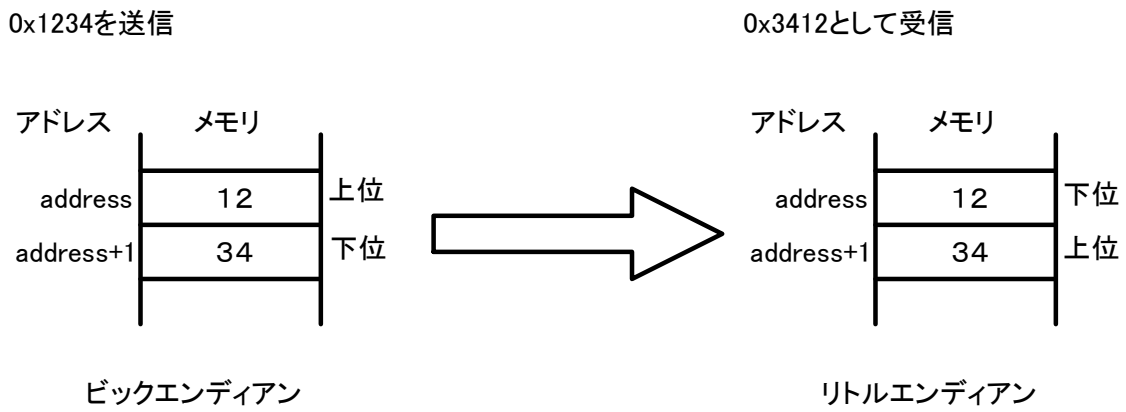


図２．７　送信と受信でエンディアンが異なる場合

このような通信上のエラーを防ぐため TCP/IP では送受信するコンピュータにかかわらず「ビックエンディアン」として送受信し，送受信するコンピュータに応じてデータ形式に変換する方式がとられています．このようにコンピュータに関係なくプロトコルによって決まるデータ形式を「**バイトオーダ**」といい，特に TCP/IP 通信のバイトオーダを「**ネットワークバイトオーダ**」といいます．

Python 言語ではこれらの変換を行うために encode メソッドと decode メソッドを使用します．

図２．８　ネットワークバイトオーダへの変換

(5)　int 型、float 型データのバイナリ変換

カウント値や時刻データ、センサデータなどは int 型もしくは float 型がよく用いられます。しかしソケット経由で送受信するデータはバイナリデータでなくてはなりません。そこで、int 型もしくは float 型を一度文字列へ変換し、さらにバイナリデータへ変換することで int 型や float 型のデータをソケット経由で送受信させてみます。

<int 型(整数)→str 型(文字列)、float 型(実数)→str 型(文字列)>

```
data = 100   #変数 data は int 型になる
msgData = str(data)   # 文字列「100」に変換

data = 3.14  #変数 data は float 型になる
msgData = str(data)   # 文字列「3.14」に変換
```

< str 型(文字列) →　バイナリデータ>

```
BinaryData = msgData.encode( )   #バイナリデータへ変換
```

<バイナリデータ →　str 型(文字列)>

```
StringData = BinaryData.decode( )   #文字列に変換
```

< str 型(文字列)→int 型(整数)、str 型(文字列)→float 型(実数)>

```
data = int ( StringData )    #int 型へ変換
data = float ( StringData )  #float 型へ変換
```

(6) ローカル IP アドレスの取得

サーバ側のプログラムを作成する場合，ソケットにサーバ機の IP アドレスをセットなくてはなりません．しかし IP アドレスはサーバ機やネットワーク形態によって変わるためコードの中で IP アドレスを直接記述するとアプリとしての互換性が保てず，サーバ機が変わるたびにコードを直接書き直す必要があるため手間がかかります．

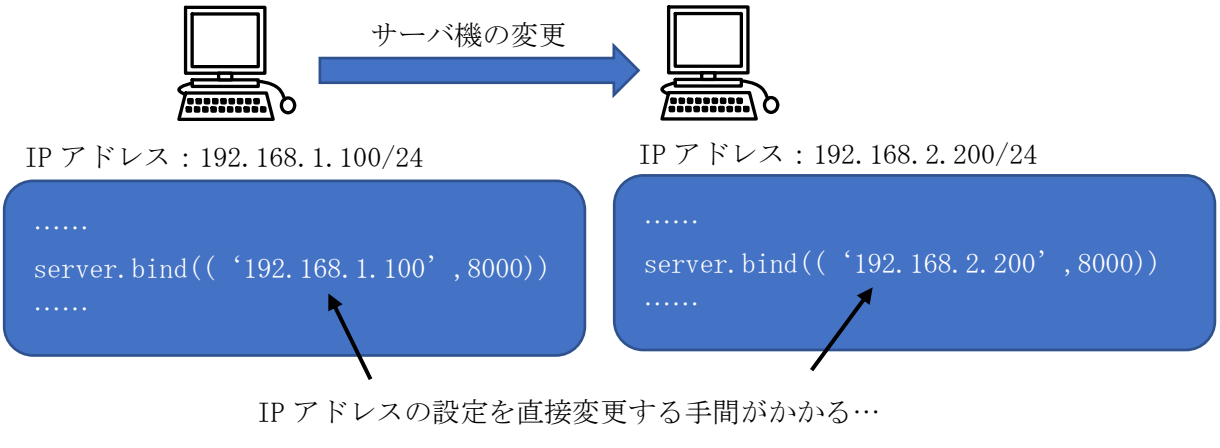


図２．９　コード内の IP アドレスを直接変更する

サーバ側のプログラム互換性を向上させるために，一般的にはサーバ機にあらかじめセットされている IP アドレスを Python プログラムで取得し，ソケットに設定すればアプリの互換性を向上させることができるうえ，入力間違いなどのミスを軽減できます.

Python 言語において自身の IP アドレスを取得する方法はいくつかありますが，ここでは下記のライブラリを使用します.

①ipget ライブラリのダウンロード・インストール

```
$ sudo  pip3  --proxy=http://10.0.0.2:15080  install  ipget
```

上記における「--proxy=http://10.0.0.2:15080」はプロキシサーバの IP アドレスとポート番号です．一般家庭などプロキシサーバを利用しない場合は「$ sudo pip3 install ipget」を実行してください.

②サーバ側プログラムを記述

```
import  ipget

#インスタンス生成
ip = ipget.ipget()

#IP アドレスの取得(str クラス)
ipAddr = ip.ipaddr("eth0").split('/')

#ソケット生成
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server:
      #取得した IP アドレスを使用して bind 処理をする
       server.bind((ipAddr[0],8000))


       ……
```

このライブラリによって IP アドレスを取得するにはインターフェース(上記の例では eth0)を指定しなくてはなりません．もし wifi を利用している場合は「wlan0」を指定してください.

(例１)有線 LAN の場合

```
ipAddr = ip.ipaddr("eth0").split('/')
```

(例２)無線 LAN の場合

```
ipAddr = ip.ipaddr("wlan0").split('/')
```

```
"""
エコーサーバ　サンプルプログラム


"""
import socket
import ipget
import sys

#ipgetインスタンス生成
ip = ipget.ipget()
try:
    #IPアドレス取得
    ipAddr = ip.ipaddr("eth0").split('/')
    #取得したIPアドレスを表示
    print(ipAddr[0])
    print(type(ipAddr[0]))
except Exception as e:
    print(e)
    sys.exit()

#ソケット生成
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server:
    #ソケットオプション指定（ソケットの再利用可能）
    server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

    #ソケットにIPアドレス、ポート番号セット（bind処理）
    #server.bind(('10.0.3.70',8000))
    server.bind((ipAddr[0],8000))

    #ソケットの接続待ち用に設定（listen処理）
    server.listen(1)

    while True:
        #ソケット接続待ち（accept処理）
        client,addr=server.accept()

        # 接続したクライアントへの処理
        with client:
            while True:
                #ソケットからデータ受信
                data = client.recv(256)

                #接続がなければ終了
                if not data:
                    break

                #接続したクライアント情報を表示（省略可能）
                print('data : {}, data: {},
                client:{}'.format(data.decode(),addr,client))

                #全て送信（ネットワークバイナリ）
                client.sendall(b'Received: '+data)
```

```python
"""
エコークライアント　サンプルプログラム

"""
import socket

#ソケット生成
with socket.socket(socket.AF_INET,socket.SOCK_STREAM) as client:

    #サーバ接続（connect処理）
    client.connect(('10.0.1.100',8000))

    #送信するデータを準備
    msg = 'Hello'

    #データ送信（ネットワークバイトオーダで送信）
    client.sendall(msg.encode())

    #データ受信
    recvData=client.recv(1024)

    #受信したデータを表示（ネットワークバイトオーダから変換）
    print(recvData.decode())

    ###
    #注意！　with文を使用してソケット生成しているので
    #　　close処理を省略しています！
```

２．４　Windows 版エコーサーバ、エコークライアントアプリ

本実習では Windows 版エコーサーバおよびエコークライアントアプリを用意しました.

(1)エコーサーバ

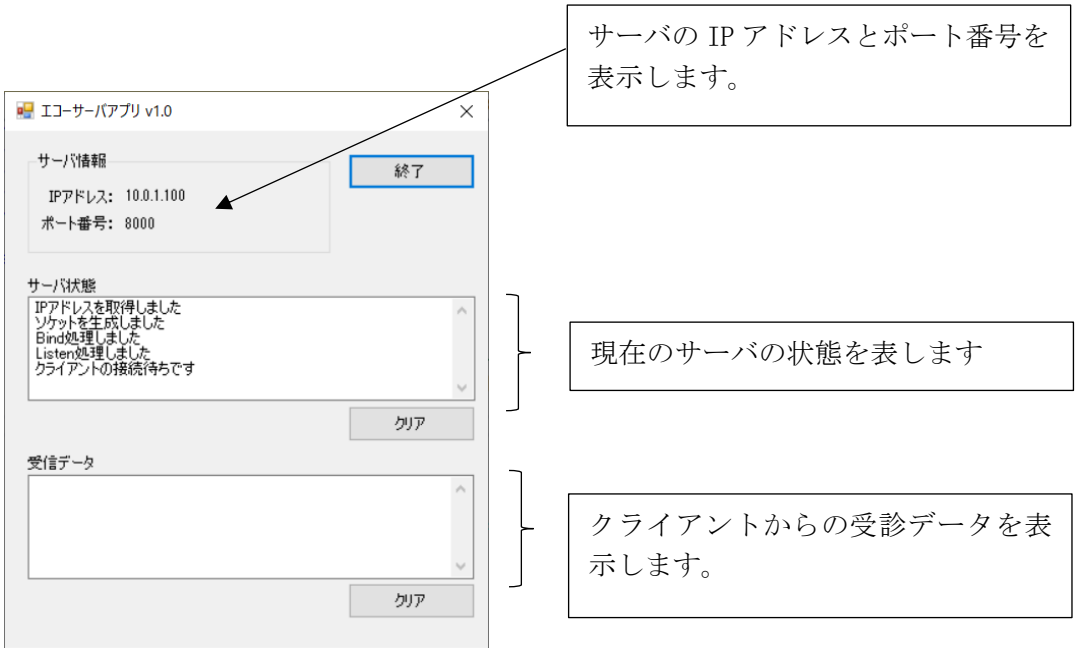


図２．１０　エコーサーバアプリ詳細

(2)エコークライアントアプリ

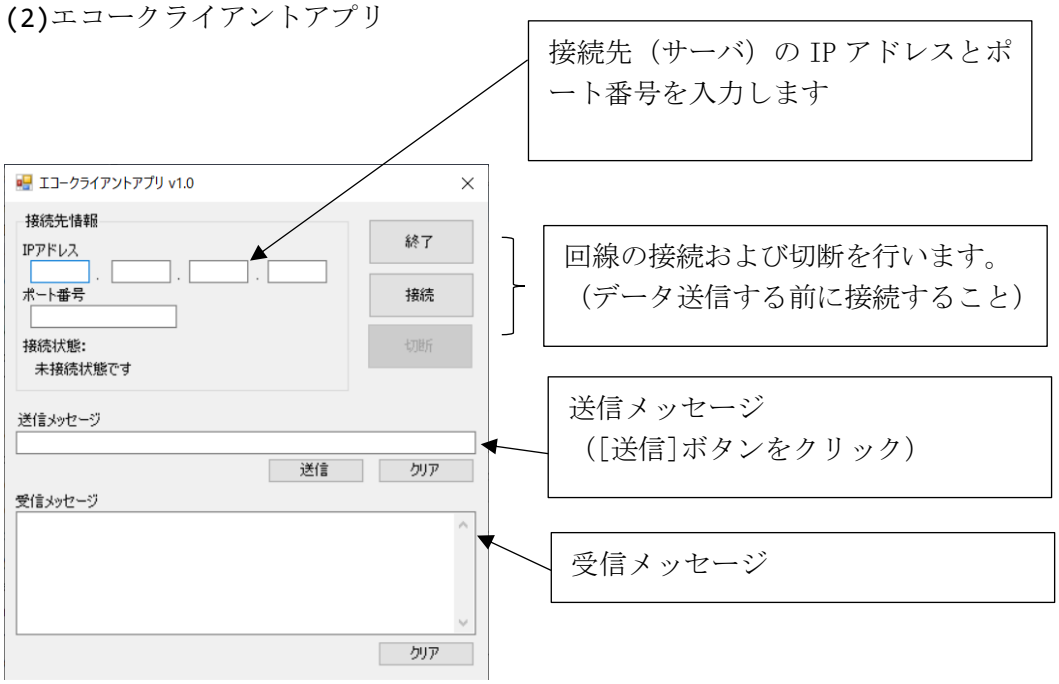


図２．１１　エコークライアントアプリ詳細

(3)サンプルプログラムのコピーと動作確認
コピー元：z:¥work¥socket
コピー先：/home/pi/work/socket/

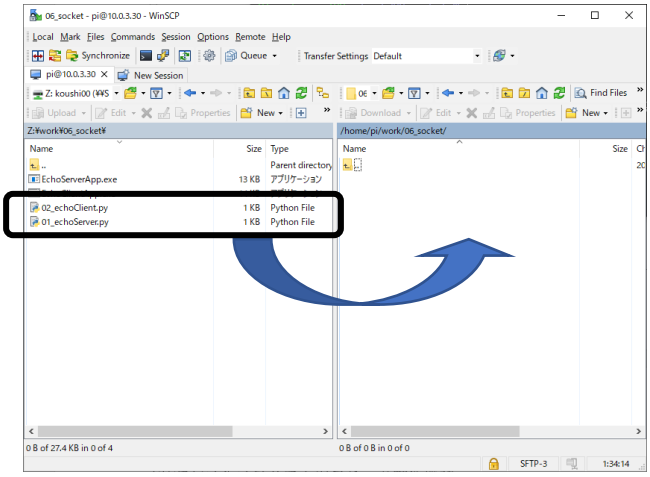


図２．１２　サンプルプログラムのコピー

動作確認①
ＰＣ：エコークライアント
ラズベリーパイ：エコークサーバ

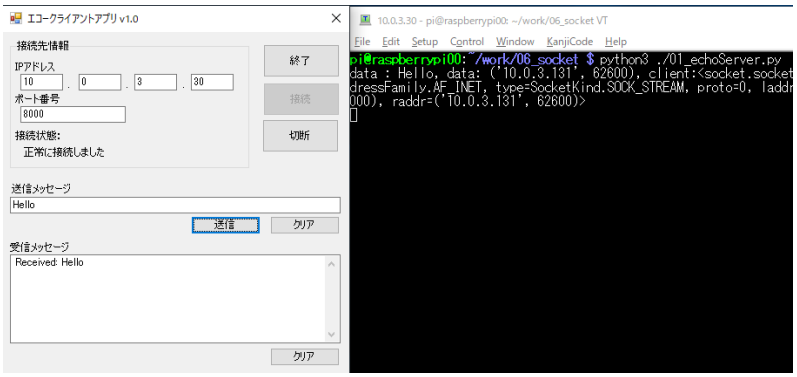


図２．１３　エコークライアントアプリの実行例

動作確認②
ＰＣ：エコーサーバ
ラズベリーパイ：エコーククライアント

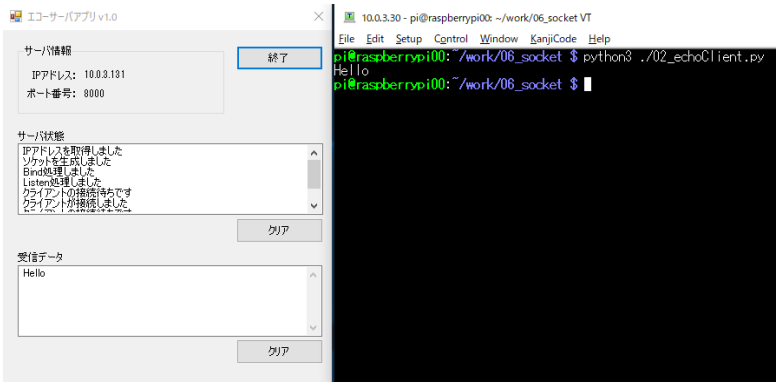


図２．１４　エコーサーバアプリの実行例

２．５　練習問題（サーバへのセンサデータ送信）

(1) 下記の仕様のようなプログラムを作成しましょう．

＜準備＞

```
$ cd ~/work/socket/
```

ファイル名：tmpAppli.py

<仕様>
クライアント（RaspberryPi）に接続されている温度センサ（TMP102）から測定データ（温度）をサーバ（Windows）に１秒間隔で送信する．

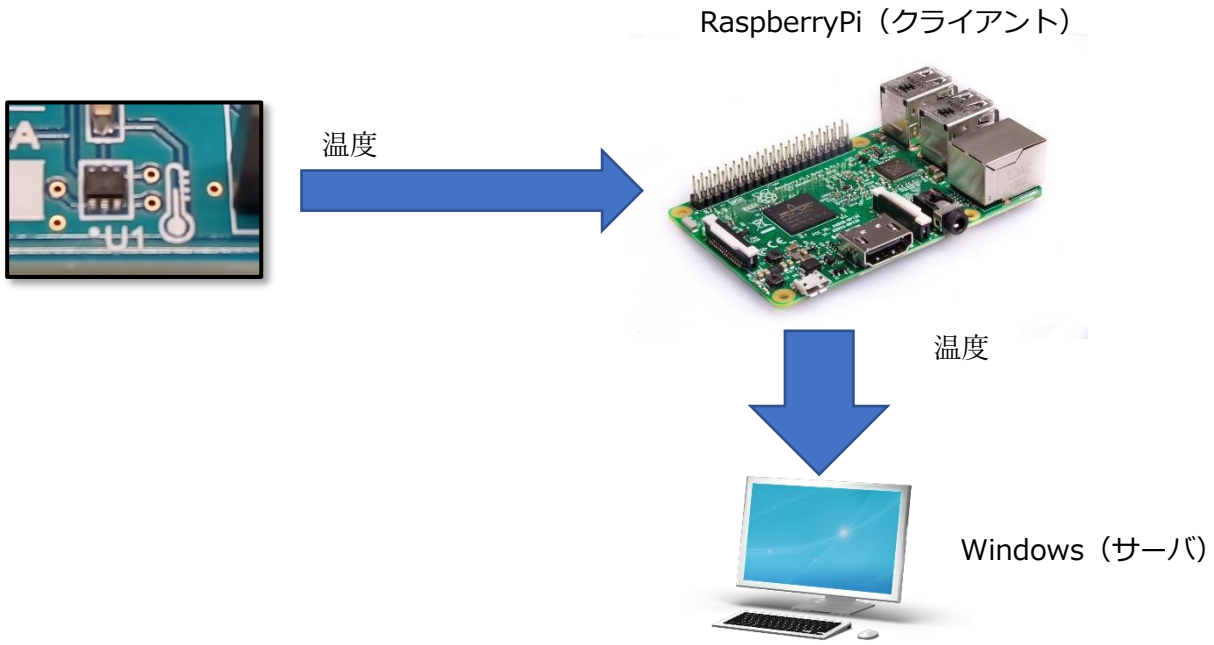


図２．１５　システムイメージ

<実行例>

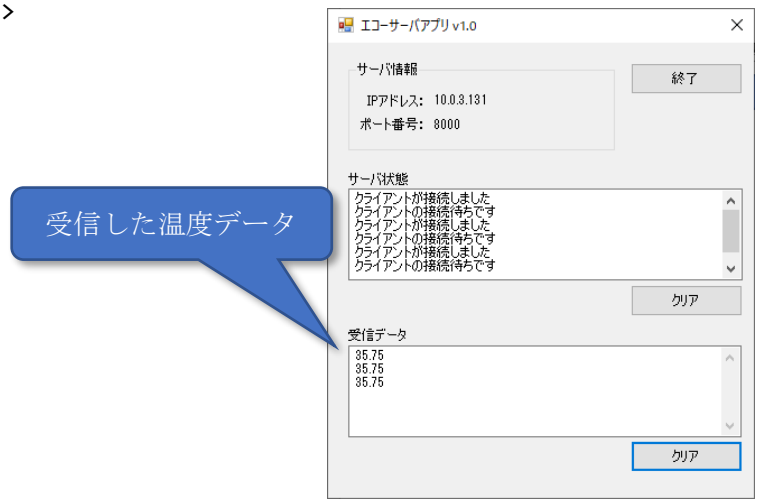


図２．１６　実行例

<ヒント>
オンボード温度センサ TMP102 のライブラリは下記のように使用します

```
#ライブラリインポート
form tmp102 import TMP102

#インスタンス生成
tmp = TMP102()

#温度データ取得(float 型)
tmp.readTemperature()
```

２．６　搬送負荷装置

(1)搬送負荷装置の外観

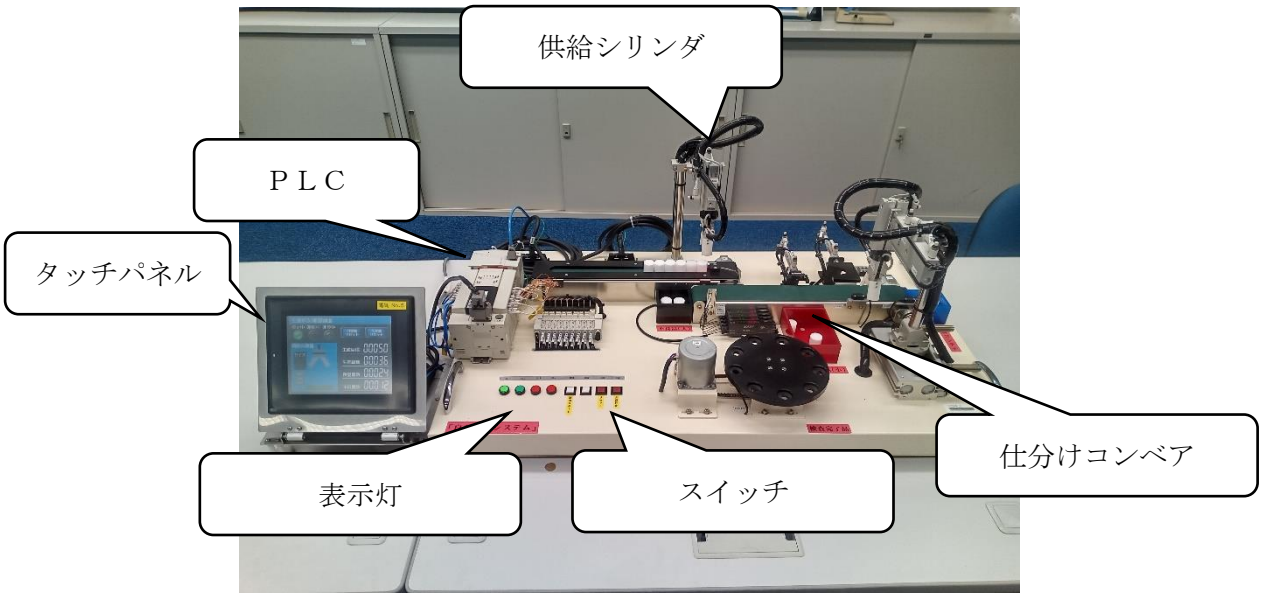


図２．１７　搬送負荷装置

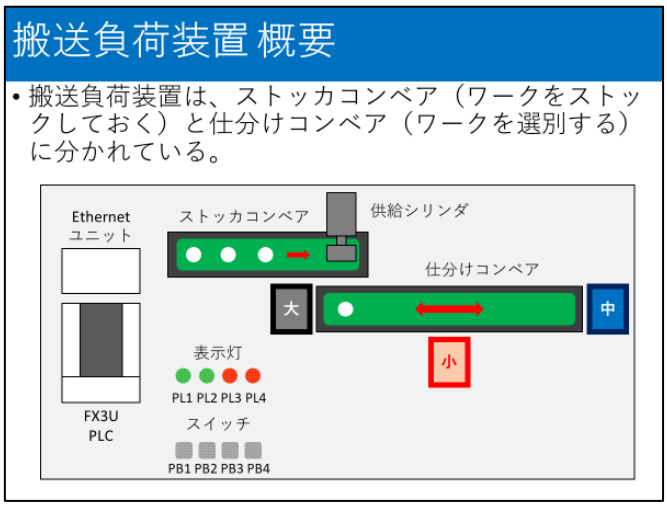


図２．１８　搬送負荷装置の各種アクチュエータ

(2)動作概要

本実習で使用する搬送負荷装置は，コントローラである PLC(Programable Logic Controller) によってコンベア，ロボットアーム，シリンダ，ＦＡセンサを制御し製品に見立てた搬送ワークの仕分けをする実習装置です．また，本実習装置には標準インターフェースとして押しボタンスイッチ（４個）とパイロットランプ（４個）が搭載されています．そのほかにタッチパネルディスプレイを接続しておりＦＡ装置の生産目標数，生産数，良品・不良品数の表示を行っております．

装置の概要と動作仕様を図２．１９に示します．

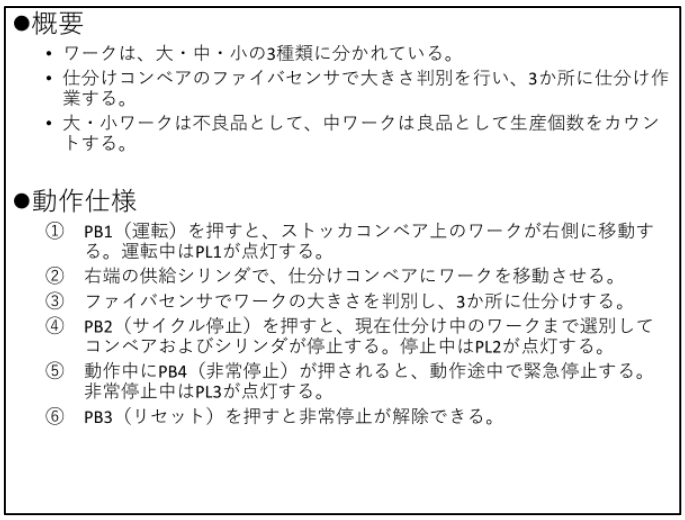


図２．１９　概要と動作仕様

２．７　ラズベリーパイと連携したＦＡ制御

(1)全体システム概要

PLC だけでは装置の制御はできても，遠隔地からの監視や制御を行うことは困難です．一般的な製造現場を想定し，PLC に Ethernet 機能を追加し，遠隔監視・制御できるようにシステム構築を行います．

ラズベリーパイからはソケット通信を使用して搬送負荷装置の稼働状況や生産数の取得および運転停止／開始の遠隔制御を行います．

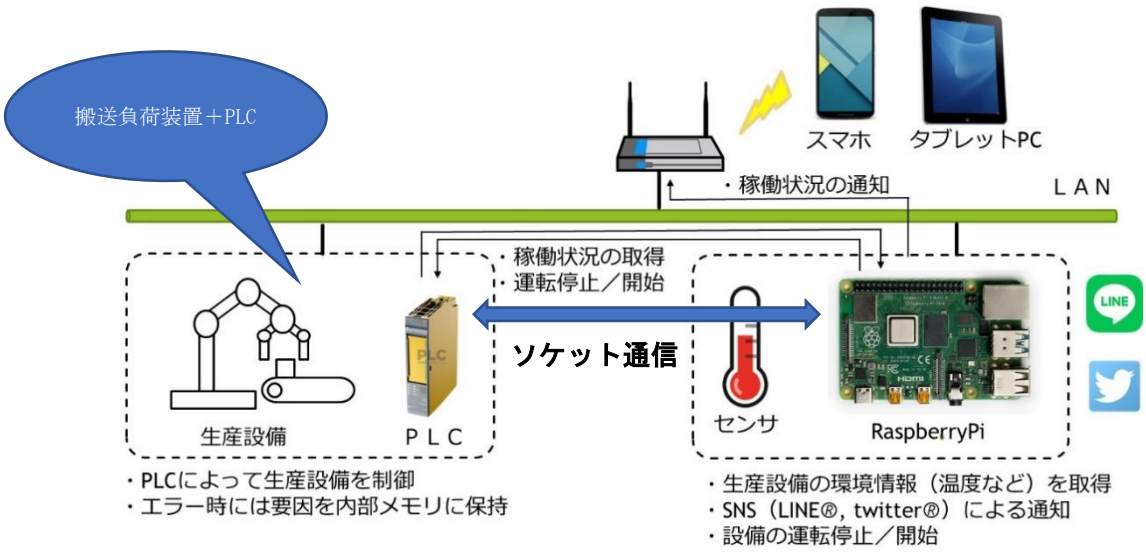


図２．２０　全体システム概要図

表２．１　ＰＬＣのＩＰアドレスとポート番号

PLC側	設定値
IPアドレス	10.0.11.220
ポート番号	5001

(2)PLC との通信プロトコル

Ethernet 経由で PLC におけるレジスタのリード／ライトおよび内部接点の制御を行うためにはソケットを利用して決められた通信パケットを送信する必要があります．この通信パケットは PLC メーカーや型番，シリーズによって変わるため詳細は PLC ハードウェアマニュアル等を確認する必要があります．

本実習では**MC プロトコル**を使用します．MC プロトコルとは MELSEC コミュニケーションプロトコルのことであり，Ethernet ポートを介してデバイスデータリード／ライトをおこなうための三菱電機㈱製 PLC 専用の通信方式です。MC プロトコルには Q シリーズの,F シリーズによって詳細なパケット情報は異なります．

また，SLMP(Seamless Message Protocol)を採用している機器があります．SLMP とは，汎用 Ethernet 機器と CC-LinkIE 対応機器間において，ネットワークの階層・境界を意識しないアプリケーション間通信を可能にする共通プロトコルです．シンプルなクライアント・サーバ型プロトコルであるため，各種機器への実装も簡単にできます．SLMP にて、MC プロトコルで対応可能な伝文フォーマットとコマンドを使用することで MC プロトコル搭載機器と通信ができます．

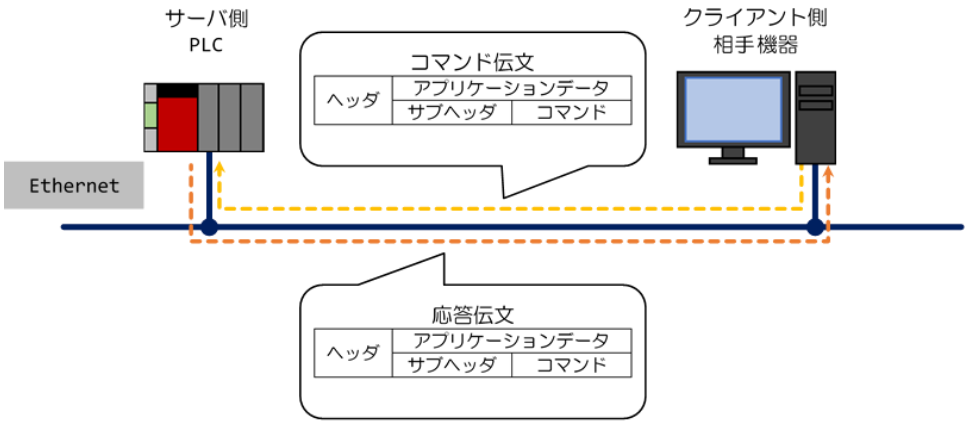


図２．２１　MCプロトコルを使用した通信

(3)通信パケットフォーマット

MC プロトコルの通信パケットには，クライアントからの制御内容が伝えられる**コマンド伝文**（要求伝文）と要求結果を伝えられる**応答伝文**があります．コマンド伝文の機能には様々な機能が定義されています．要求するコマンド（制御内容）によって通信パケットフォーマットが異なります．要求するコマンドと応答は，内容によって固有の値（**サブヘッダ**）が割り当てられております．またリード／ライトの対象となる PLC 内臓デバイスにも固有の値（**デバイスコード**）が割り当てられています．**ヘッダ，PC 番号，監視タイマ，終了コード**を含めることで PLC に対して様々な情報をリード／ライトできます．

下記に MC プロトコルにおける通信パケットのフォーマットを示します．

①　デバイス読込み

要求伝文　相手機器 → PLC【ASCII コード】

ヘッダ	サブヘッダ	PC 番号	監視タイマ	要求データ			00
				デバイスコード	デバイス先頭番号	デバイス数	
(0)	2	2	4	4	8	2	2

応答伝文　PLC → 相手機器【ASCII コード】

ヘッダ	サブヘッダ	終了コード	応答データ
(0)	2	2	デバイス数

② デバイス書込み

要求伝文　相手機器 → PLC【ASCII コード】

ヘッダ	サブヘッダ	PC 番号	監視タイマ	要求データ			00	書込データ
				デバイスコード	デバイス先頭番号	デバイス数		
(0)	2	2	4	4	8	2	2	デバイス数

応答伝文　PLC → 相手機器【ASCII コード】

ヘッダ	サブヘッダ	終了コード
(0)	2	2

③ 異常終了

応答伝文　PLC → 相手機器【ASCII コード】

ヘッダ	サブヘッダ	終了コード	異常コード
(0)	2	2	ユニット依存

(4)通信パケット詳細

通信パケットの詳細を下記に示します.

① ヘッダ

TCP/IP、UDP/IP 用の Ethernet ヘッダです. 通常は要求伝文へ自動的に付加されます.

② サブヘッダ

サブヘッダは表１．２にあるように制御内容によって固有の値(2byte)を割り当てられております.

表２．２　サブヘッダ設定値一覧(一部抜粋)

機能	サブヘッダ		参照
	要求伝文	応答伝文	
デバイスメモリ読出し	00H	80H	ビットデーター括読出し
	01H	81H	ワードデーター括読出し
デバイスメモリ書込み	02H	82H	ビットデーター括書込み
	03H	83H	ワードデーター括書込み

③ PC 番号

アクセス先の局番を指定します.

■　接続局（自局）アクセス

FF を指定してください.

■　ネットワーク経由の他局アクセス

アクセス先のネットワークユニット局番 01H～40H（1～64）を指定してください.

④　監視タイマ

読出し/書込みの処理を完了するまでの待機時間を設定します.

■　0000H(0)　　：無限待機(処理完了まで待機する)

■　0001H～FFFFH(1～65535)　：待機時間(単位：250ms)

※正常なデータ交信を行うため、交信先により以下の設定範囲で使用することが推奨されています.

自局：1H～28H(0.25～10 秒)、他局：2H～F0H(0.5 秒～60 秒)

⑤　要求データ

制御対象となるデバイスデータを設定します.要求データはデバイスコード,デバイス先頭番号,デバイス数を指定します.

■　デバイスコード

CPU ユニットで使用されるデバイスとデバイス毎の固有番号（デバイスコード）を下記に示します.

表２．３　デバイスコード設定値一覧(一部抜粋)

デバイス名		記号	種別	デバイスコード
入力		X	ビット	5820 H
出力		Y	ビット	5920 H
内部リレー		M	ビット	4D20 H
タイマ	現在値	T	ワード	544E H
	接点		ビット	5453 H
カウンタ	現在値	C	ワード	434E H
	接点		ビット	4353 H
データレジスタ		D	ワード	4420 H

■　デバイス先頭番号

アクセス先の CPU ユニットで使用できるデバイス範囲で指定します.

00000000H～FFFFFFFFH(0～上限値)で指定すること。

※入出力 X/Y は、シリーズにより 8 進/16 進が、内部リレーM やデータレジスタ D などは 10 進の表現が使用されるため注意すること

■　デバイス数

読出し/書込みを行うデバイスの点数を指定します.

1 コマンド内のデバイス点数が､1 回の交信で行える処理点数以内になるように指定すること。

⑥　終了コード・異常コード

コマンド処理結果が格納される。

■　正常終了時は 0 が格納される。

■　異常終了時はアクセス先のエラーコードが格納される。

■　エラーコードは、発生したエラー内容を示す。

２．８　ＰＬＣからのデータ取得

(1)CPU ユニットの M200～M203 までのデバイス（4 ビット）を読み出す.

■　要求伝文　相手機器 → PLC

サブヘッダ	PC 番号	監視タイマ	要求データ			終了
			デバイス	デバイス先頭	デバイス数	
H　L	H　L	H　L	H　L	H　　L	H　L	
00	FF	0000	4D20	000000C8	04	00

M -- デバイスコード(4D20 H)
200 -- デバイス先頭番号(00C8 H)

先頭(M200)から 4 つ分のデバイス
(M200, M201, M202, M203)

■　応答伝文　PLC → 相手機器【ASCII コード】

サブヘッダ	終了コード	応答データ			
		M200	M201	M202	M203
H　L	H　L	H			L
80	00	0	1	0	1

M200～M203 までの接点情報が戻る
（この部分を分析すると，どの接点が ON か OFF か判断できる）

■　ソケット通信による記述例

```
#要求伝文を送信（M200 から 4 つ分のビットデータ読み込み）
client.sendall(b”00FF00004D20000000C80400”)
…
#応答伝文を受信
Data = client.recv(128).decode()

#接点(M200)情報の取得(str 型から int 型へ変換する)
M200_state = int(Data[4])

#接点(M200)情報の解析と制御
if M200_state == 1:  #もし M200 が 1 だったら…
   ……
```

(2)CPU ユニットの D200～D203 までのデバイス（4 ワード）を読み出す.

■　要求伝文　相手機器 → PLC

サブヘッダ	PC 番号	監視タイマ	要求データ			終了
			デバイス	デバイス先頭	デバイス数	
H　L	H　L	H　L	H　L	H　　L	H　L	
01	FF	0000	4420	000000C8	04	00

D -- デバイスコード(4420 H)
200 -- デバイス先頭番号(00C8 H)

先頭(D200)から 4 つ分のデバイス
(D200, D201, D202, D203)

■　応答伝文　PLC → 相手機器【ASCII コード】

サブヘッダ H L	終了コード H L	応答データ D200 H L	D201 H L	D202 H L	D203 H L
81	00	00FF	03E8	0000	000A

D200～D203 までの接点情報が戻る
（この部分を分析すると，PLC 内部の数値情報が取得できる．ただし 16 進数 4 桁なので 10 進数に変換すること）

■　ソケット通信による記述例

```
#要求伝文を送信（D200 から 4 つ分のビットデータ読み込み）
client.sendall(b”01FF00004420000000C80400”)
…
#応答伝文を受信
Data = client.recv(128).decode()

#接点(D200)情報の取得(ASCII コード 4 桁で受信するので[4]~[7]までの値を取得)
D200_value = int(Data[4:8], 16)

#接点(D200)情報の解析と制御
if D200_value < 100:  #もし D200 の値が 100 未満だったら…
    ……
```

２．９　練習問題　（搬送負荷装置における稼働状況の遠隔監視）

(1)システム概要

搬送負荷装置の生産管理システムを構築します．本実習では搬送負荷装置の稼働状況（稼働中，停止中，非常停止中）を取得しコンソールに表示するアプリケーションを作成しましょう．

(2)共有デバイス

搬送負荷装置の稼働状況は下記のデバイスに格納されています．

表１．２　システム状況が格納されているデバイス

デバイス		番号	格納されている情報
ビット	M	200	システム停止中(1:停止状態)
	M	201	システム稼働中(1:稼働状態)
	M	202	システム非常停止中(1:非常停止状態)
	M	203	原点復帰中（1:原点復帰状態）

PLC に実装されているプログラムにおいて，装置の状況に合わせて各デバイスに 1 を格納しています．ラズベリーパイからソケット通信を使用して M200～M202 を読み込むことで現在の稼働状況を判断することができます．

(3)搬送負荷装置の IP アドレスおよびポート番号
　搬送負荷装置に配線されている PLC の IP アドレスおよびポート番号は下記のとおりです.

表１．３　ＰＬＣのＩＰアドレスとポート番号

PLC側	設定値
IPアドレス	10.0.11.220
ポート番号	5001

(4)ファイル名，ファイルパス
ファイル名：machine_state.py
ファイルパス:/home/pi/work/socket/

(5)記述例

```
……
cmd = “00FF00004D20000000C80400”  #コマンドセット
client.sendall( cmd.encode() )    #コマンド送信

#応答伝文を受信
Data = client.recv(128).decode()

#接点(M200)情報の取得(str 型から int 型へ変換する)
M200_state = int(Data[4])

#接点(M200)情報の解析と制御
if M200_state == 1:  #もし M200 が 1 だったら…
  print(“システム停止中です”)
```

(6)動作例

```
$ python3 machine_state.py
システム停止中です。
```

――メモ――

２．１０　練習問題　（搬送負荷装置における生産管理の遠隔監視）

(1)システム概要

搬送負荷装置の生産管理システムを構築します．搬送負荷装置は製品に見立てた搬送ワークを仕分けするシステムがすでに組まれています．搬送ワークは大，中，小の大きさがあり大と小の搬送ワークは不良品とし中のみ良品として判定されます．システム動作中に良品･不良品の数をカウントし，特定のデータレジスタに保存されます．

本課題ではラズベリーパイからソケット通信を利用して搬送負荷装置の生産目標数，生産数，良品の数，不良品の数を取得しコンソールに表示する生産管理システムを構築します．

(2)共有デバイス

搬送負荷装置の製造状況（生産目標数，生産数，良品の数，不良品の数）は下記のデータレジスタに格納されています．

表２．４　製造状況が格納されているデバイス

デバイス		番号	格納されている情報
ワード	D	200	生産目標数
	D	201	生産個数（良品と不良品の合計数）
	D	202	良品の数（中）
	D	203	不良品の数（大，小の合計数）

(3)搬送負荷装置の IP アドレスおよびポート番号

搬送負荷装置に配線されている PLC の IP アドレスおよびポート番号は下記のとおりです．

表２．５　ＰＬＣのＩＰアドレスとポート番号

PLC側	設定値
IPアドレス	10.0.11.220
ポート番号	5001

(4)ファイル名，ファイルパス

ファイル名：manufacture_state.py

ファイルパス:/home/pi/work/socket/

(5)記述例

```
#要求伝文を送信（D200 から 4 つ分のビットデータ読み込み）
client.sendall(b”01FF00004420000000C80400”)
…
#応答伝文を受信
Data = client.recv(128).decode()

#接点(D200)情報の取得(ASCII コード 4 桁で受信するので[4]~[7]までの値を取得)
D200_value = int(Data[4:8], 16)
```

```
#生産目標数の表示
print(f”生産目標数:{D200_value}”)
```

(6)動作例

```
$ python3 manufacture_state.py
生産目標数:50
生産数:23
良品数:19
不良品数:4
```

――メモ――

２．１１　ＰＬＣへのデータ書き込み

(1)CPU ユニットの M400～M403 までのデバイス(４ビット)へ書き込む.

■　要求伝文　相手機器 → PLC

サブヘッダ H L	PC 番号 H L	監視タイマ H L	要求データ デバイス H L	要求データ デバイス先頭 H L	要求データ デバイス数 H L	終了	指定デバイスデータ M400 H	M401	M402	M403 L
02	FF	0000	4D20	00000190	04	00	1	0	0	1

M -- デバイスコード(4D20 H)
400 -- デバイス先頭番号(0190 H)
先頭(M400)から 4 つ分のデバイス
(M400, M401, M402, M403)

■　応答伝文　PLC → 相手機器【ASCII コード】

サブヘッダ H L	終了コード H L
82	00

■　ソケット通信による記述例

```
#要求伝文を送信（M400 から 4 つ分のビットデータ書き込み[1,0,0,1]）
client.sendall(b”02FF00004D200000019004001001”)
…
#応答伝文を受信
res = client.recv(128).decode()
```

(2)CPU ユニットの D300～D303 までのデバイス（4 ワード）へ書き込む.

■　要求伝文　相手機器 → PLC

サブヘッダ	PC 番号 H L	監視タイマ H L	要求データ デバイス H L	要求データ デバイス先頭 H L	要求データ デバイス数 H L	終了	指定デバイス　書込データ D300 H L	D301 H L	D302 H L	D303 H L
03	FF	0000	4420	0000012C	04	00	000A	00C8	0000	0064

D -- デバイスコード(4420 H)
300 -- デバイス先頭番号(012C H)
先頭(D300)から 4 つ分のデバイス
(D300, D301, D302, D303)
それぞれに格納する値を 16 進数 4 桁で指定する

■　応答伝文　PLC → 相手機器【ASCII コード】

サブヘッダ	終了コード H L
83	00

■　ソケット通信による記述例

```
#要求伝文を送信（D300 から 4 つ分のビットデータ書き込み[10,200,0,100]）
client.sendall(b”03FF000044200000012C0400000A00C800000064”)
…
#応答伝文を受信
res = client.recv(128).decode()
```

２．１２　練習問題　（搬送負荷装置における生産目標数の設定）

(1)システム概要

搬送負荷装置の生産管理システムを構築します．搬送負荷装置は製品に見立てた搬送ワークをいくつ仕分けるか目標値が設定できます．

本課題では生産目標数の設定を遠隔で行えるようにプログラムを作成します．

(2)共有デバイス

搬送負荷装置の製造状況（生産目標数，生産数，良品の数，不良品の数）は下記のデータレジスタによって設定できます．

表２．６　製造状況が格納されているデバイス

デバイス		番号	格納されている情報
ビット	D	300	生産目標数
	D	302	良品の数（中）
	D	303	不良品の数（大，小の合計数）

(3)搬送負荷装置の IP アドレスおよびポート番号

搬送負荷装置に配線されている PLC の IP アドレスおよびポート番号は下記のとおりです．

表２．７　ＰＬＣのＩＰアドレスとポート番号

PLC側	設定値
IPアドレス	10.0.11.220
ポート番号	5001

(4)ファイル名，ファイルパス

ファイル名：production_target.py

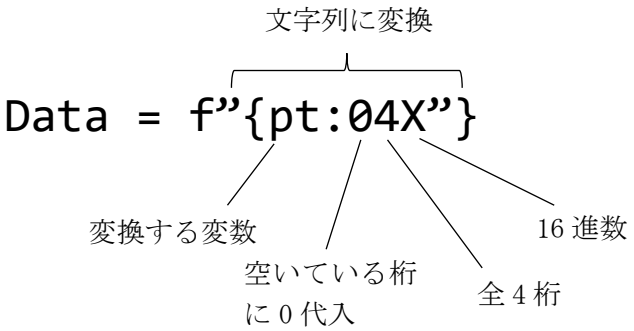
ファイルパス:/home/pi/work/socket/

(5)記述例

```
#要求伝文を送信（D300 に 100 を書き込み）
pt = 100 #数値をセット(10 進数)
senddata = f”03FF000044200000012C0100{pt:04X}”  #16 進数に変換し文字列にセット
client.sendall(senddata.encode())  #ネットワークバイトオーダに変換して送信
…
```

(6)10 進数から 16 進数(ASCII コード)への変換

MC プロトコルで PLC のデータレジスタに値を設定するには，16 進数 4 桁に変換し ASCII コードに変換しなくてはなりません. さらに 16 進数の 4 桁なので空いている桁には 0 を代入します. Python 言語では f 文字列を使用すると便利です.



上記のように f 文字列を使用することで，桁数の指定や空いている桁に 0 代入など比較的簡単に数値を送信パケットにセットすることができます.

(7)実行例(下線部はキーボードからの入力)

```
$ python3  production_target.py
生産目標数を入力> 100
生産目標をセットしました
現在の生産目標数:100
```