山梨職業能力開発促進センター

クラウド活用による IoTシステム構築技術

生産現場におけるクラウドサービスを利用したアプリ開発

テキスト

＜目次＞

第1章　概要

第2章　ソケット通信

第3章　ＳＮＳを利用したＩｏＴアプリケーション開発

３．６　練習問題（搬送負荷装置における稼働状況の問い合わせ）
・・・・・・・・・・・・　６０
３．７　通知機能の実装・・・・・・・・・・・・・・・・・・・・・・　６２
３．８　練習問題（通知機能を利用した緊急停止の通知）　・・・・　６３
３．９　いろいろな機能の実装・・・・・・・・・・・・・・・・・・　６６

参考文献・・・・・・・・・・・・・・・・・・・

第１章　概要

１．１　本実習の概要

ＳＮＳ（Social Networking Service）には様々なものがあり，知人同士の連絡から情報発信ツールまで広く活用されています．

製造現場において期待できるＩｏＴ機器の機能には「製造機器の稼働情報」「エラーログ情報」「温度」「湿度」「気圧」「画像」など必要な情報の監視機能、緊急通知機能、遠方からの遠隔制御機能などが考えられます．

本実習では図１．１にあるように生産現場を想定した実習教材を用いて遠隔地からの制御および監視を目的としたＩｏＴアプリケーション開発におけるプログラミング実習を実施します．

第２章で紹介したソケット通信による通知および遠隔制御は他のアプリケーションと組み合わせることで現場やユーザに最適なインターフェース設計をしながら必要な情報を取得もしくは制御することができます．ただしアプリケーションの動作環境（プラットフォーム）に最適なプログラム言語の知識が必要になるうえ，開発にも時間を要します．

第３章では様々なプラットフォームに対応し，尚且つユーザが扱いやすいインターフェースとしてＳＮＳを使用し製造現場を想定した情報をどのように取得するか，専用 API の活用事例を紹介します．尚，本実習では国産 SNS として開発された LINE®を使用します．実習にあたり予め LINE アカウントを取得する必要があります．

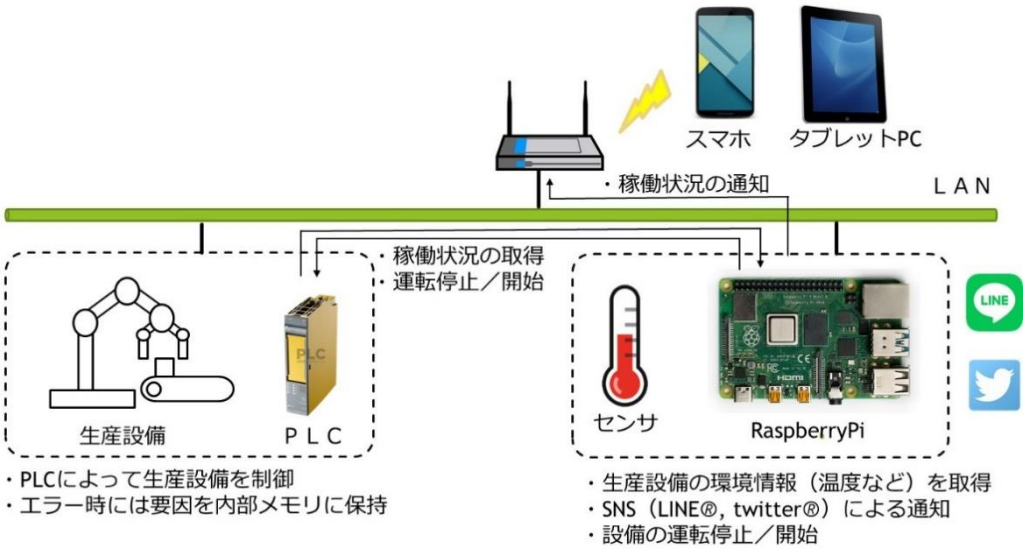


図１．１　生産現場における IoT 機器の適用事例

１．２　教材確認

表１．１　教材一覧

項目	数量	備考
RaspberryPi 3 model B+	1	
USB-TTL Serial ケーブル	1	
マイクロ SD カード　16GB	1	
マイクロ SD カードリーダ	1	
AC アダプタ	1	
ジャンパケーブル　一式	1	
ブレッドボード	1	
電子部品・センサ　一式	1	
搬送負荷装置　実習セット	1	

また、プログラム開発環境として下記の環境を用意しております。

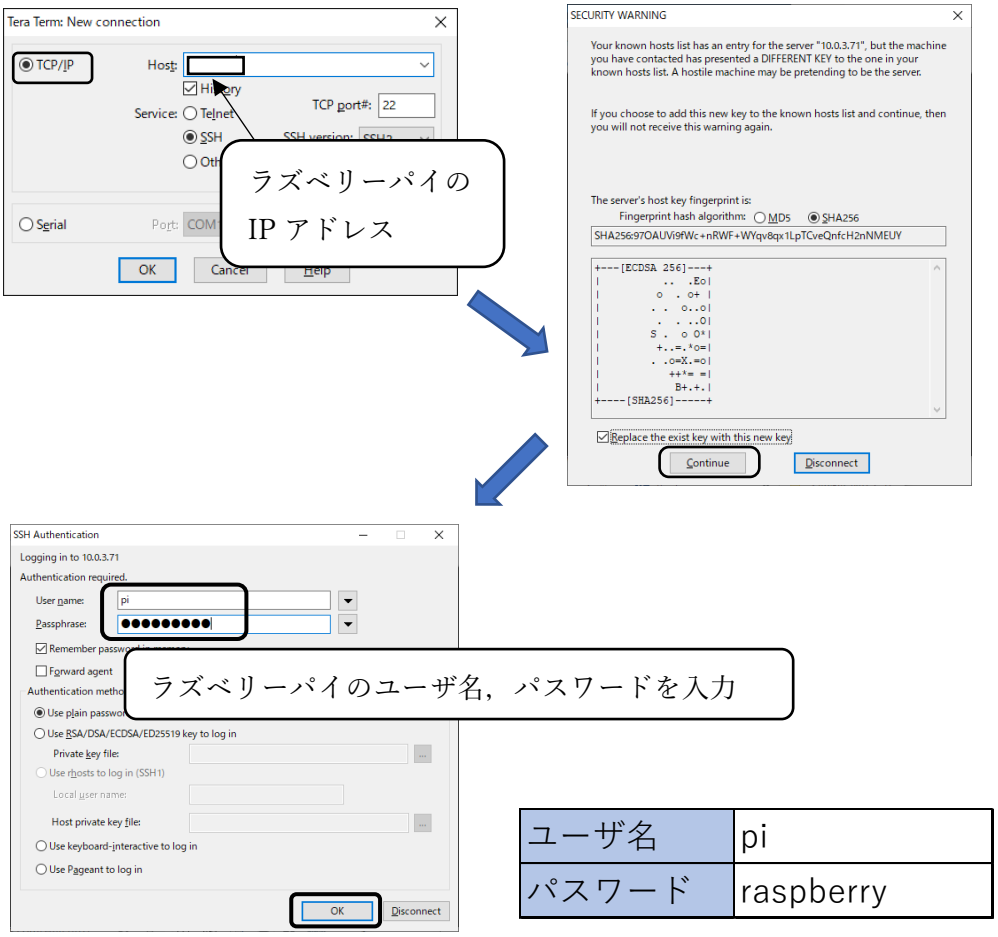
表１．２　開発環境一覧

項目	数量	備考
開発用ＰＣ	1	
Windows10 Pro 64bit		
メモリ 8GB		
SSD　512GB		

１．３　ｓｓｈ接続

(1) TeraTarm による ssh 接続

ＩＰアドレスの設定が終わったらｓｓｈ接続ができるか確認します.



ユーザ名	pi
パスワード	raspberry

図１．２　TeraTarm による ssh 接続

(2) 接続確認

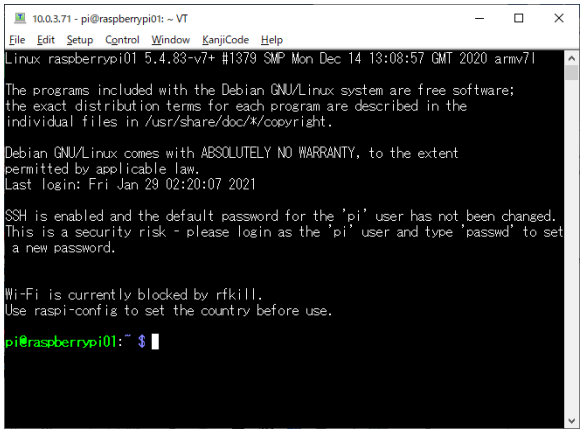


図１．３　ssh 接続した CUI 画面

１．４　開発環境

１．３までの設定で開発を進めるための準備ができました．この項目では開発をするための開発環境を整えます．本実習の開発環境は下記のとおりです．

表１．３　開発環境一覧

プログラム言語	Python 言語
エディタ	NotePad++(無償)
ファイル共有ソフト	WinSCP（無償）

プログラム言語である Python 言語はラズベリーパイ公式推奨言語のためすでにインストール済みです．また，インタプリタ型のため C/C++言語のようにコンパイラなどを別途準備したり，コンパイルのためのスクリプトを準備したりする必要はありません．

エディタである NotePad++はプログラムコード向けの軽量無償エディタです．コードを記述するための色分けやオートインデントなどの機能が搭載されており，多くのプログラム言語をサポートしています．

ファイル共有ソフトは，開発用 PC で記述したコードをラズベリーパイにコピーするために必要となります．samba などのファイルサーバを使用することもできます．今回は ssh 接続しているコンピュータにファイル転送する WinSCP を使用します．

１．５　実習用ディレクトリおよび開発環境操作方法

実習課題は①エディタでコード記述＆開発用 PC へ保存 ②ラズベリーパイへコードをコピー(WinSCP 使用)　③ラズベリーパイで実行　という流れになります．

(1)開発用 PC へコード保存用ディレクトリを作成（実習で作成したコードはここに保存する）

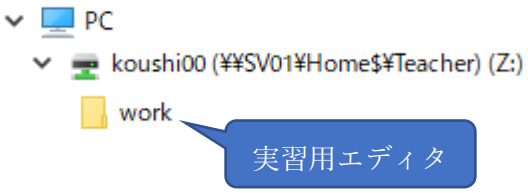


図１．４　Windows のフォルダ構成

(2)WinSCP を起動しラズベリーパイと接続する（あらかじめラズベリーパイは起動させておく）

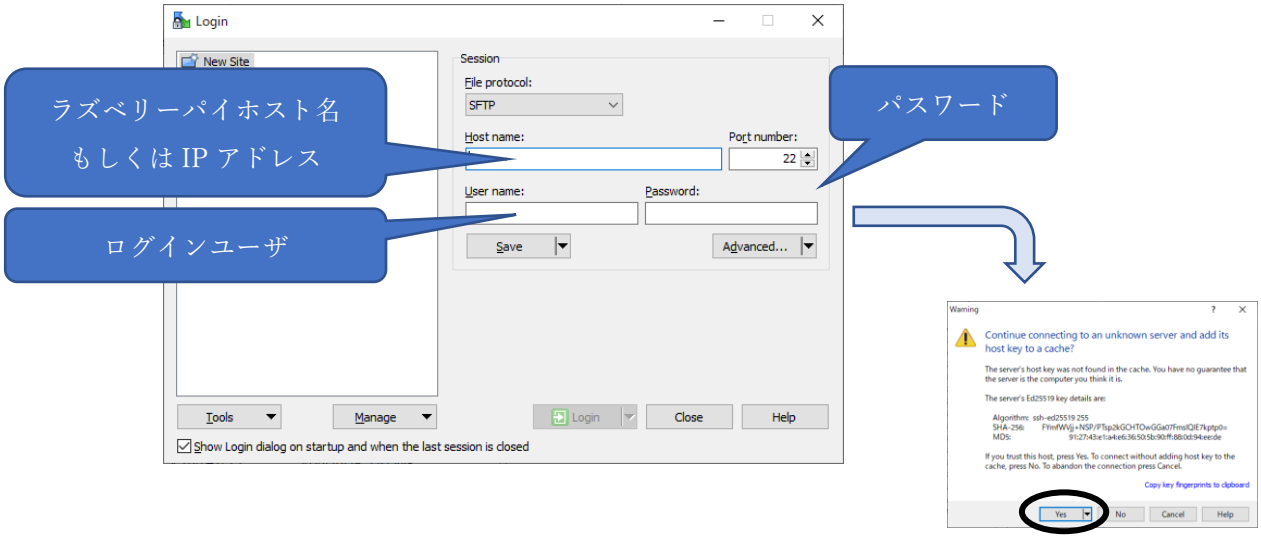


図１．５　WinSCP ログイン画面

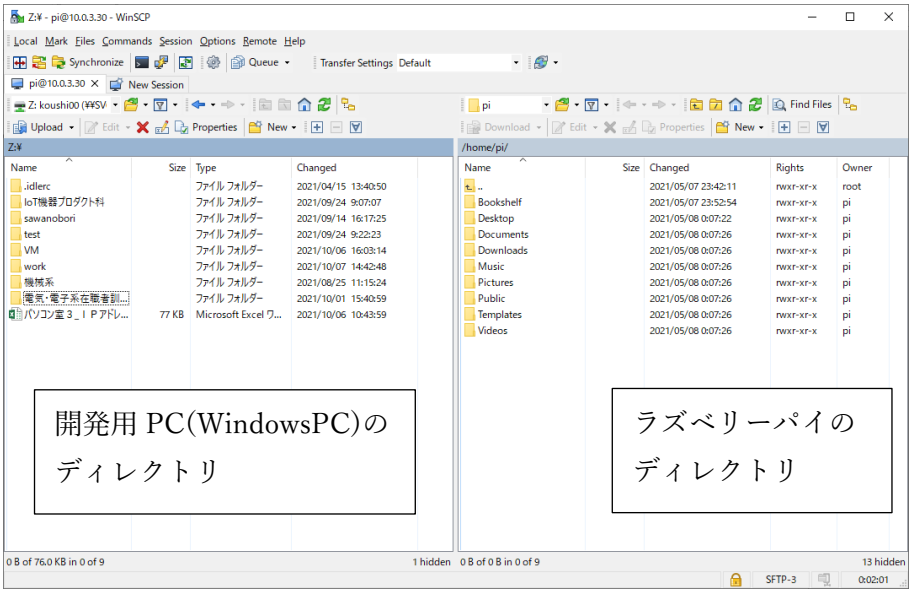


図１．６　WinSCP のメイン画面

１．６　キーボードにおける便利な操作方法

ラズベリーパイをはじめとしたターゲットボードに実装した Linux は，パソコンと違いディスプレイやキーボードを接続して動作させてプログラム開発することはあまりありません. ラズベリーパイの性能が向上したとしてもパソコンには及ばないため，ディスプレイやキーボードを接続してパソコンと同様の操作をしようとすると，どうしても動きが遅く感じてしまいます.

通常，プログラム開発や作成したアプリケーションを動作させるためにコマンド操作を行います. Linux システムにおけるコマンドは概ね下記のようなフォーマットで記述します.

```
$ [command]  [option]  [Directory/File]
```

コマンド操作はマウス操作に比べて軽快に動作するため熟練の Linux エンジニアはコマンド操作をすることが多く作業時間も短縮することができます. ただし，使い慣れていないと不便に感じます．ファイル名やディレクトリ名が長い名称だと打ち間違いが発生する可能性があり，何が原因で動作しないのか判断を間違うことがあるからです.

この項目では少しでもキーボード操作に慣れていただくことを目的として便利な操作方法を紹介します.

(1)絶対パスと相対パス

Linux システムの最上位ディレクトリのことを**／（ルート）**といいます．ルートからみた目的のファイルやディレクトリを指すパス（場所）のことを**絶対パス**といいます．一方，現在のカレントディレクトリ（作業ディレクトリ）からみた目的のファイルやディレクトリを指すパス（場所）のことを**相対パス**といいます.

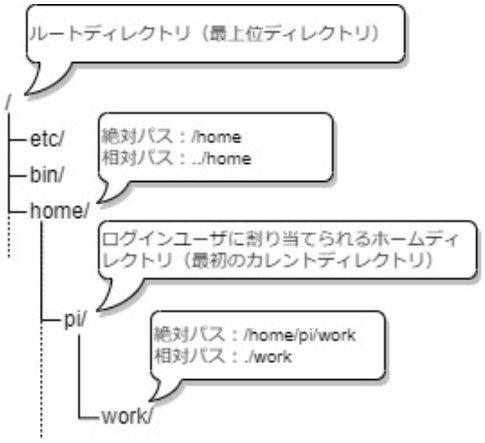


図１．７　絶対パスと相対パス

Linux システムではログインユーザごとにホームディレクトリが自動的に作成されます(Windows における Users に相当します)．ログイン直後はホームディレクトリがカレントディレクトリになっているため，ホームディレクトリを絶対パスで表すと/home/pi になります．相対パスで表す特殊な指定方法には下記のものがあります

表１．４　相対パスにおける指定方法

指定方法	意味
~/	ホームディレクトリ
./	カレントディレクトリ
../	１つ上のディレクトリ

(2)入力補助

ファイル名やディレクトリ名の中には長い名称のものがあります．１つ１つ入力していると入力間違いが起こる可能性が発生します．少しでも入力ミスを削減するために Linux システムにはファイル･ディレクトリ名の入力補助機能があります．例として下記のファイルを指定するとします．

```
$ python3  ~/work/LINE/pushMessage.py
```

上記の例では相対パスを使用して pushMessage.py を指定しています．ホームディレクトリには[w]から始まる名称のファイル・ディレクトリが work しかないためディレクトリを指定するときにキーボードから[w]を入力した後に[TAB]キーを押すと，それ以降の名称を自動入力してくれます．

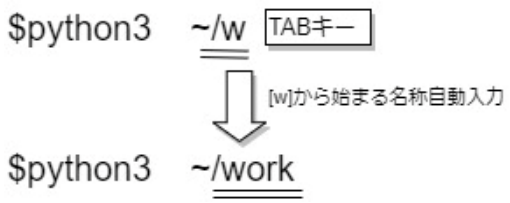


図１．８　入力補助機能を使用したディレクトリ入力

もし同じ文字から始まる名称のファイル･ディレクトリが複数ある場合は，2 文字目 3 文字目まで入力して[TAB]キーを押してください．

例えば~/work/socket/に machine_state.py と manufacture_state.py というファイルがあるとします．両方のファイルが[ma…]から始まっているため 3 文字目まで入力して[TAB]キーを使用してください．

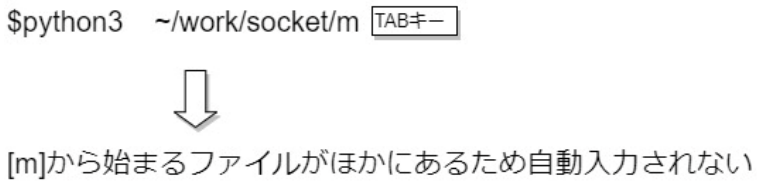


図１．９　入力補助機能の失敗例

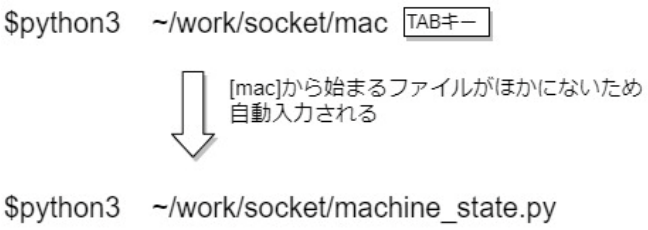


図１．１０　複数文字入力後の入力補助

(3)リダイレクト機能

プログラムコードを実行し，修正し，再度実行…．このようにプログラム作成をするウエイにおいてはトライ＆エラーを繰り返す作業となります．そのとき同じようなコマンドを何度も繰り返し入力すると，それだけで時間がかかってしまうこともあります（とくに長いコマンドや長いディレクトリ・ファイル名など）．

Linux システムには一度実行したコマンドは履歴に残っており，コマンド履歴を遡ることで何度も同じコマンドを手早く入力することができます．このような機能を**リダイレクト**といいます．

・リダイレクトをするには１度コマンドを入力する必要があります．

```
$ ls  /home/
```

・一度実行したコマンドは履歴に残りますので，履歴を辿って同様のコマンドを入力します．コマンド履歴はキーボードの[↑]を押すと辿ることができます．

```
$
```

[↑]を押すと…

```
$ ls  /home/
```

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。

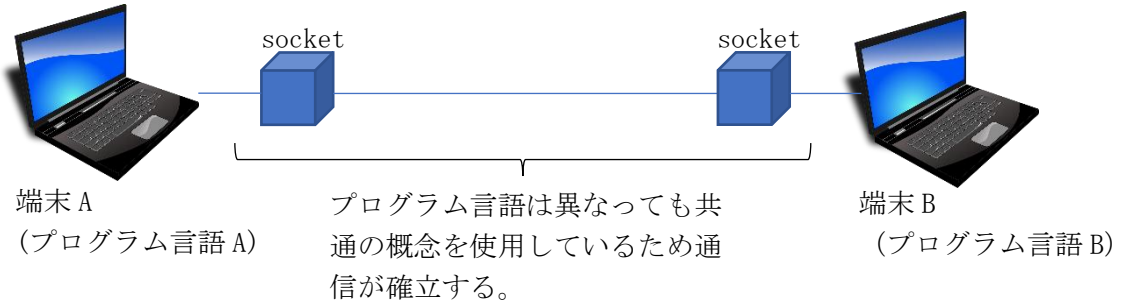


図２．１　ソケット通信イメージ

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。

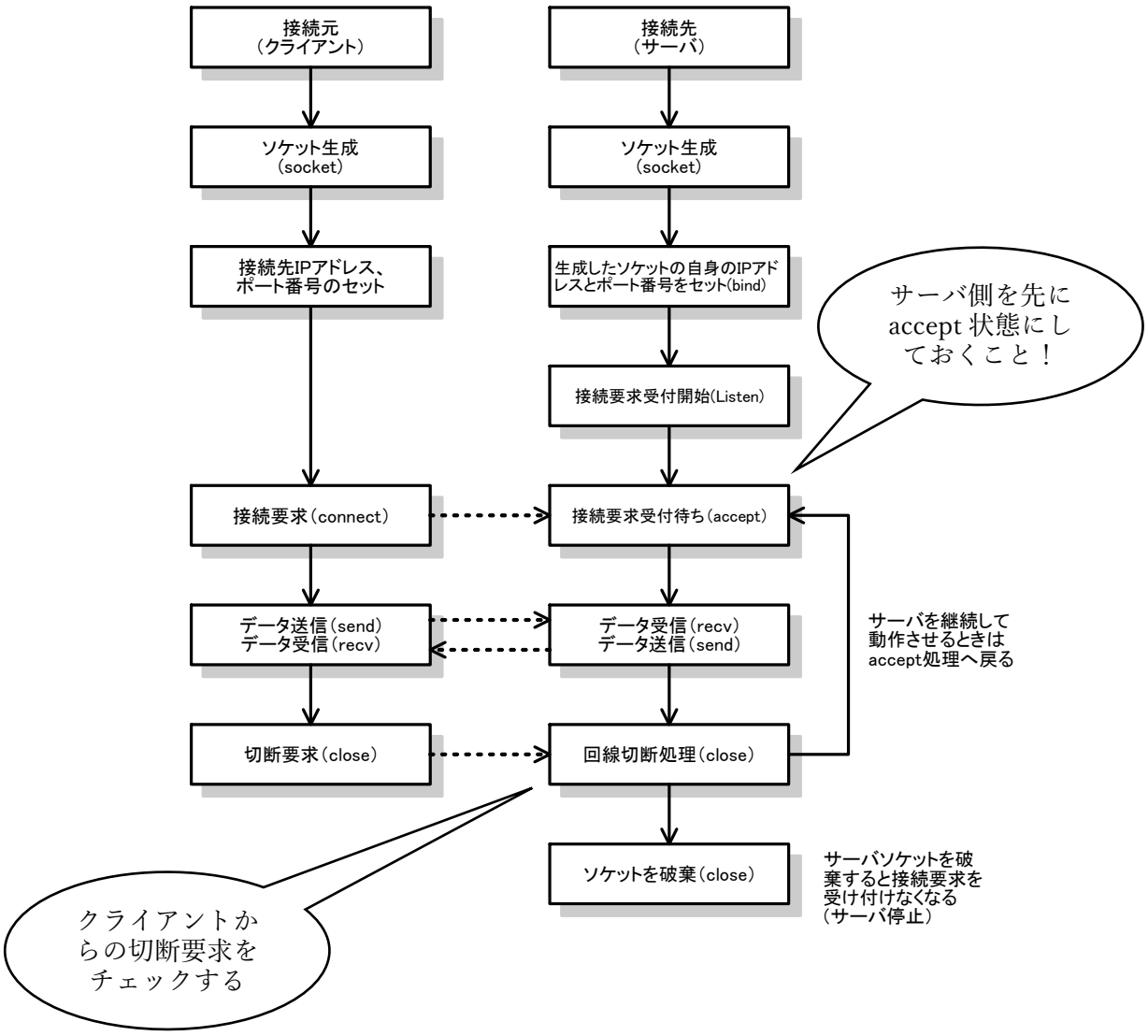


図２．２　ソケットプログラムの流れ

◆　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．

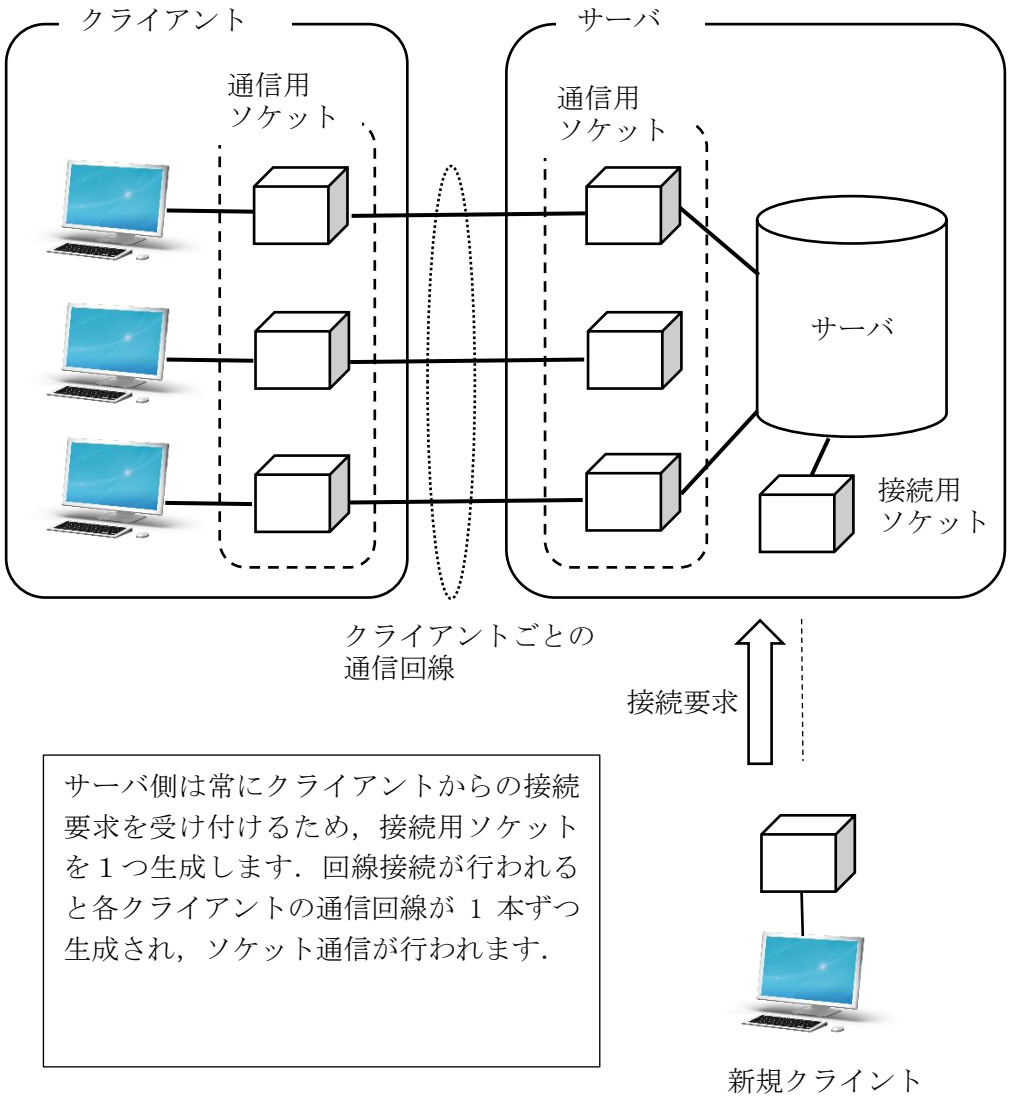


図２．３　回線接続におけるイメージ図

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．

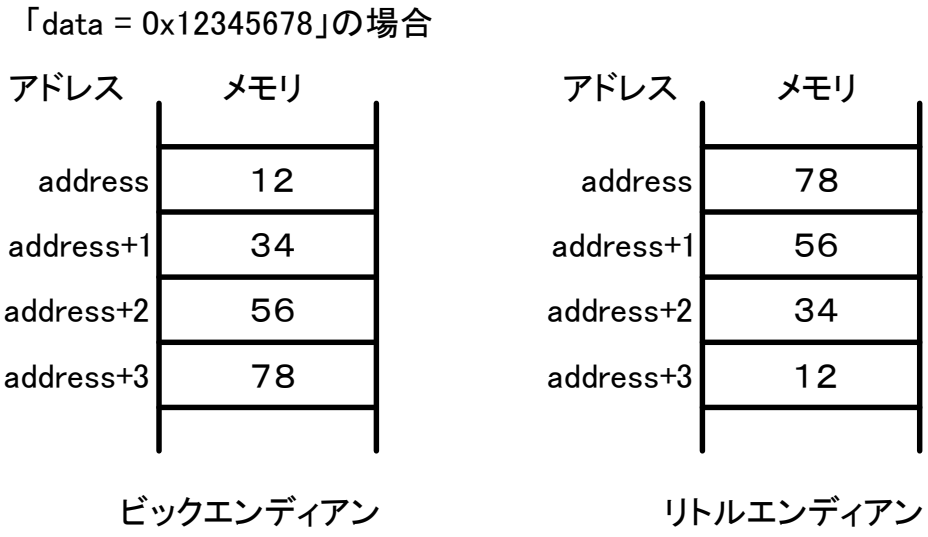


図２．６　ビックエンディアンとリトルエンディアンの違い

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）．

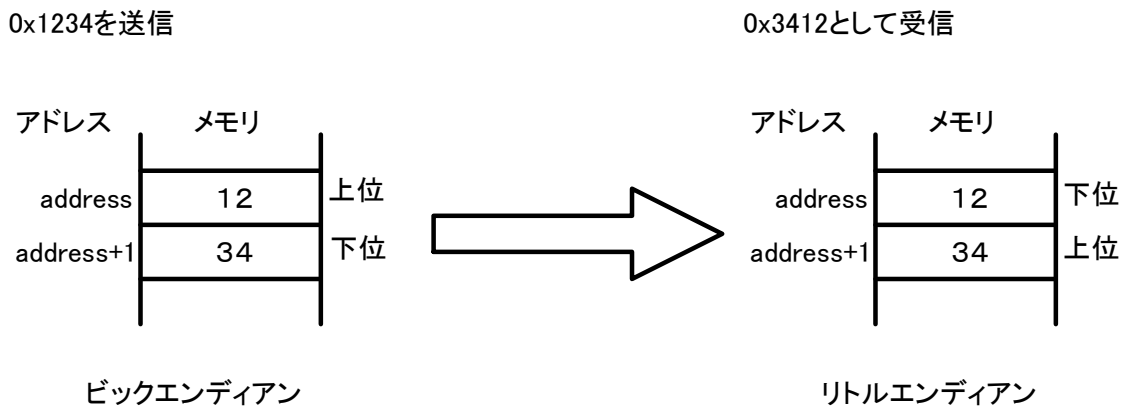


図２．７　送信と受信でエンディアンが異なる場合

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．

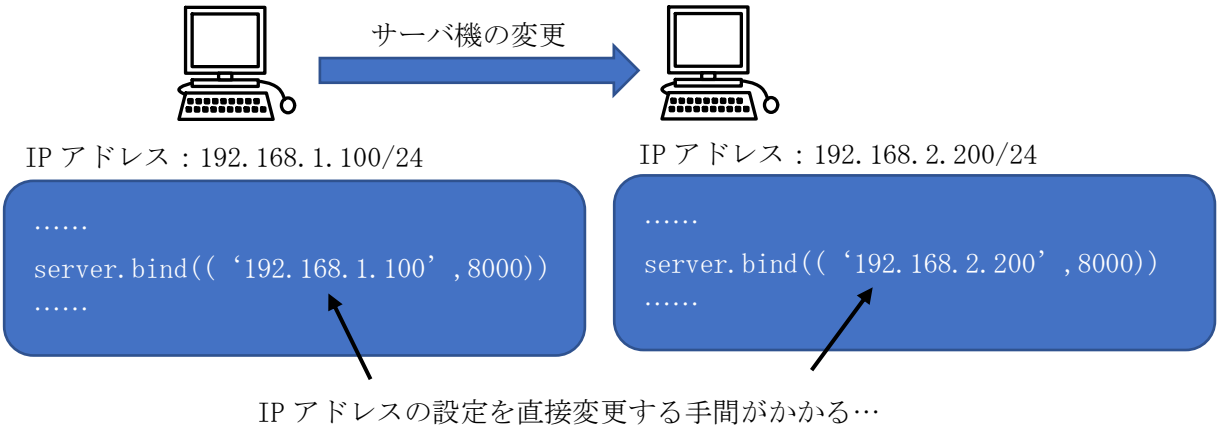


図２．９　コード内の IP アドレスを直接変更する

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

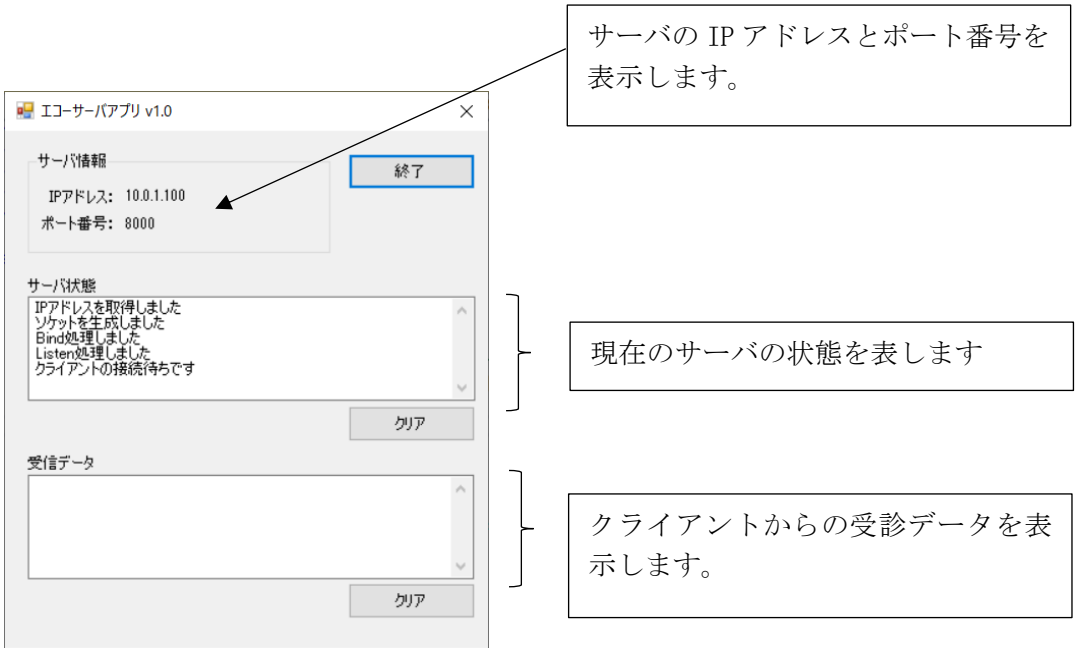


図２．１０　エコーサーバアプリ詳細

(2)エコークライアントアプリ

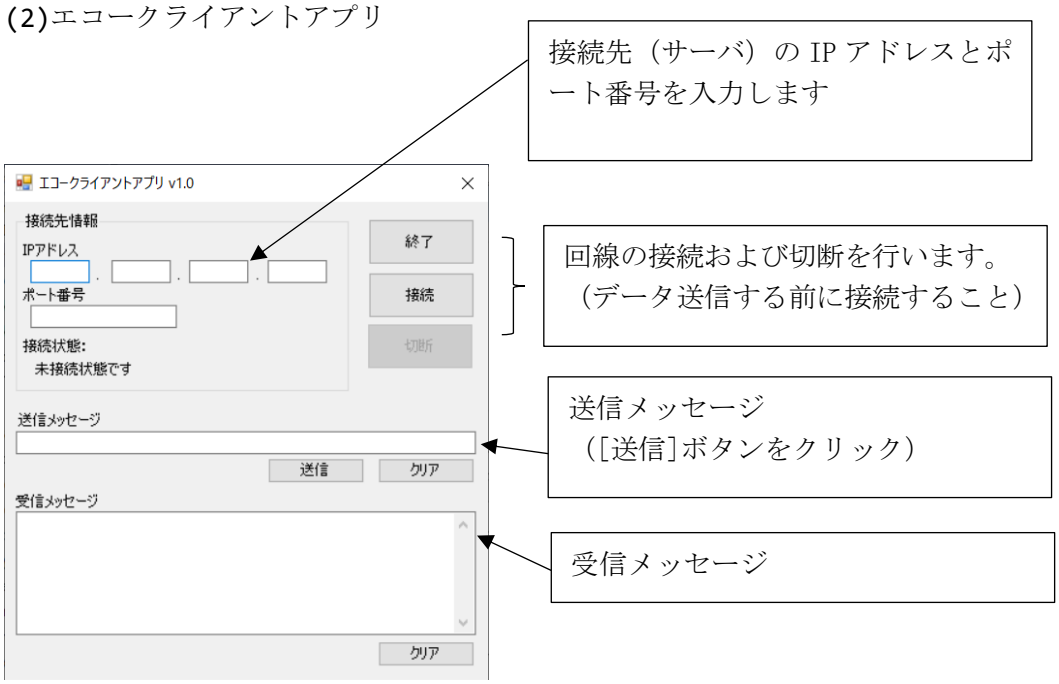


図２．１１　エコークライアントアプリ詳細

(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/

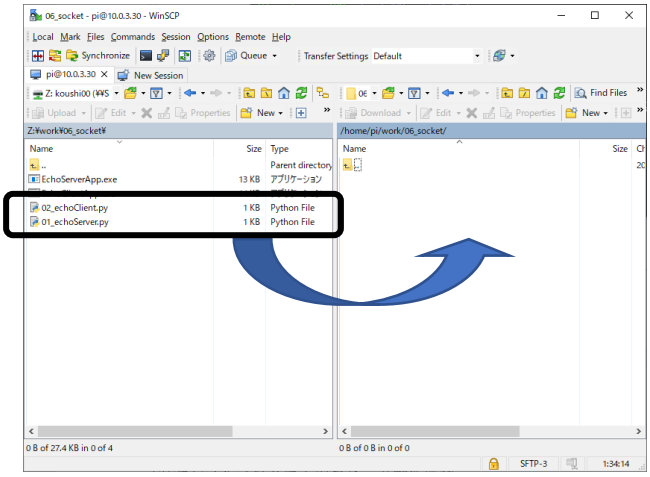


図２．１２　サンプルプログラムのコピー

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

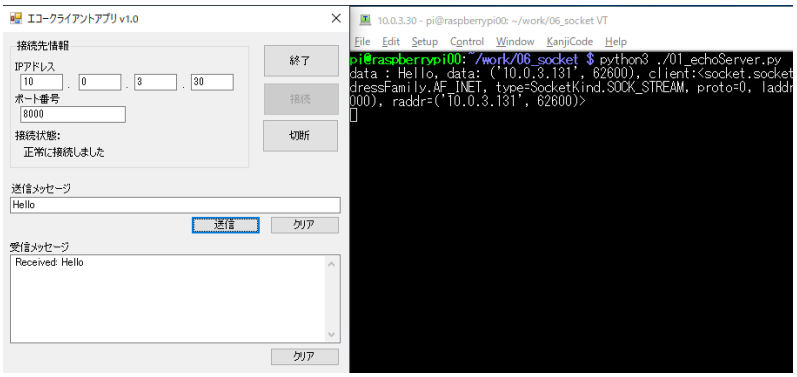


図２．１３　エコークライアントアプリの実行例

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

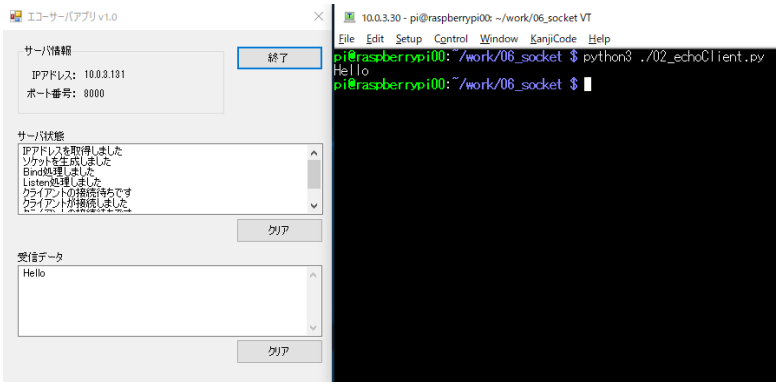


図２．１４　エコーサーバアプリの実行例

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．

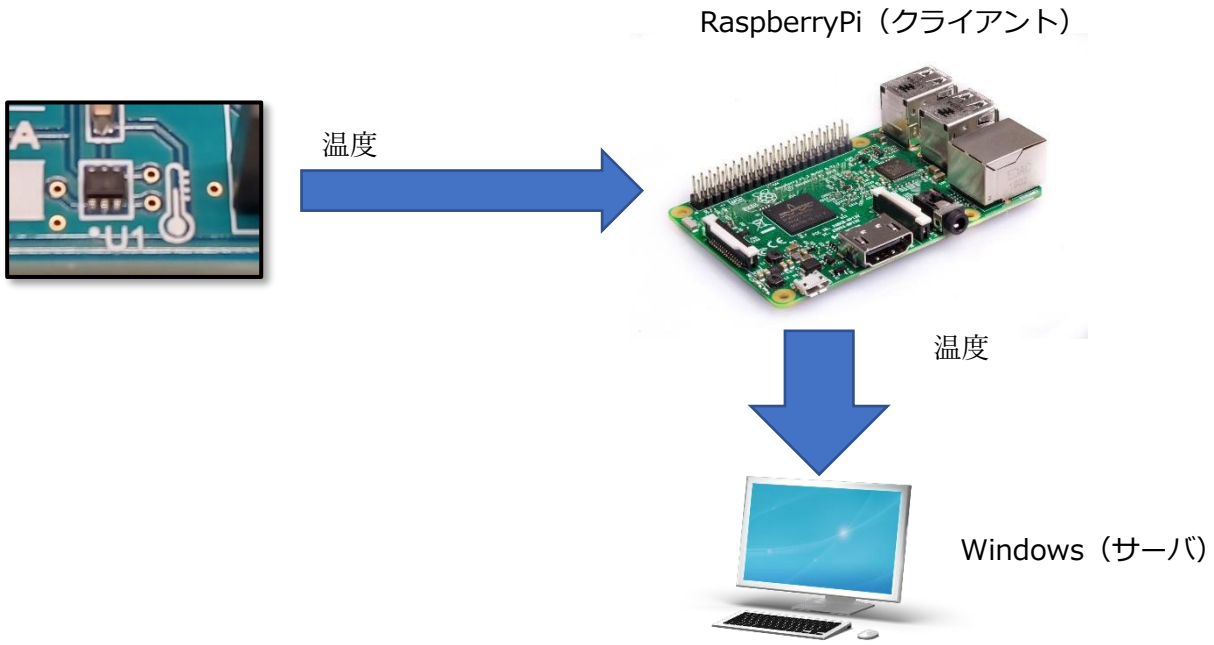


図２．１５　システムイメージ

<実行例>

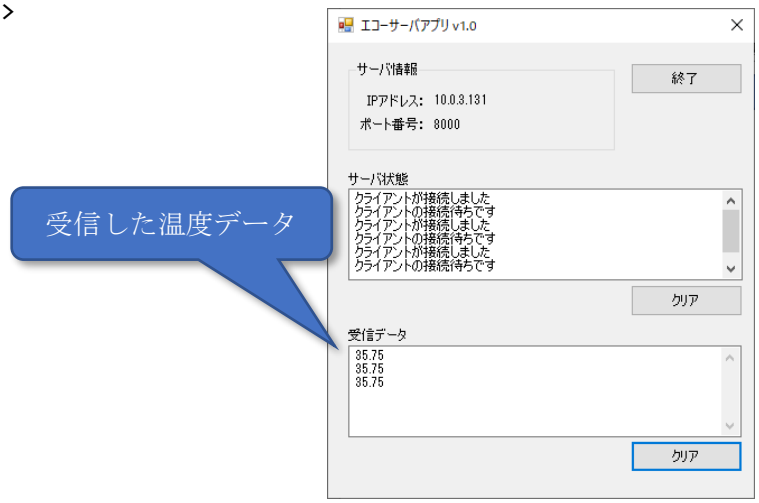


図２．１６　実行例

<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

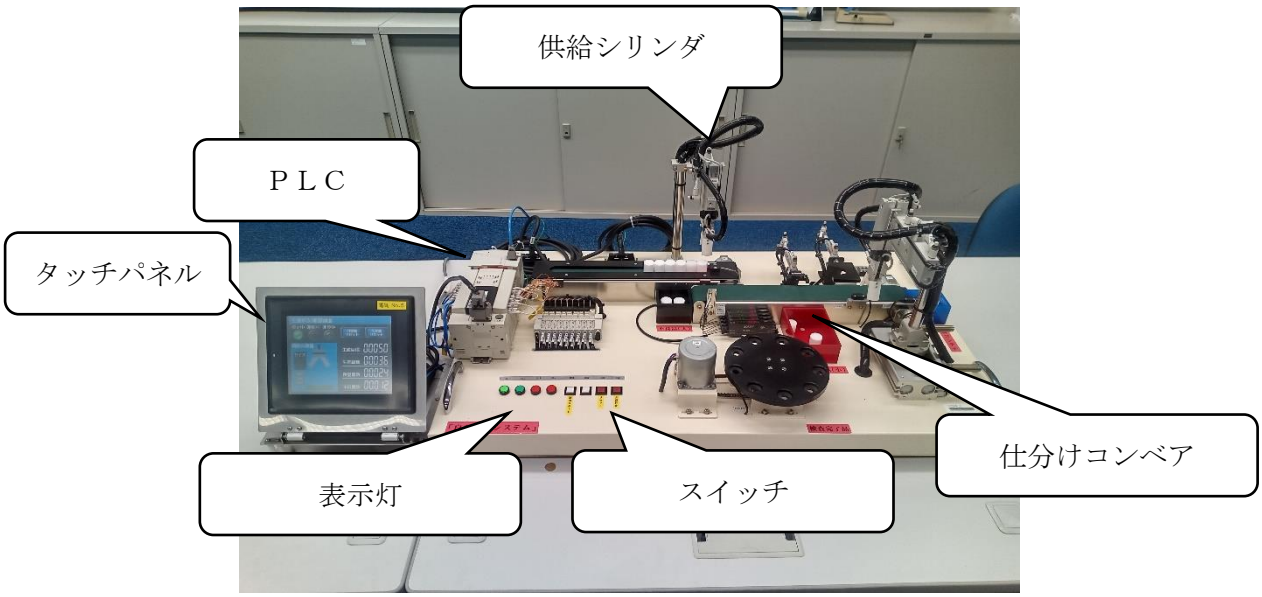


図２．１７　搬送負荷装置

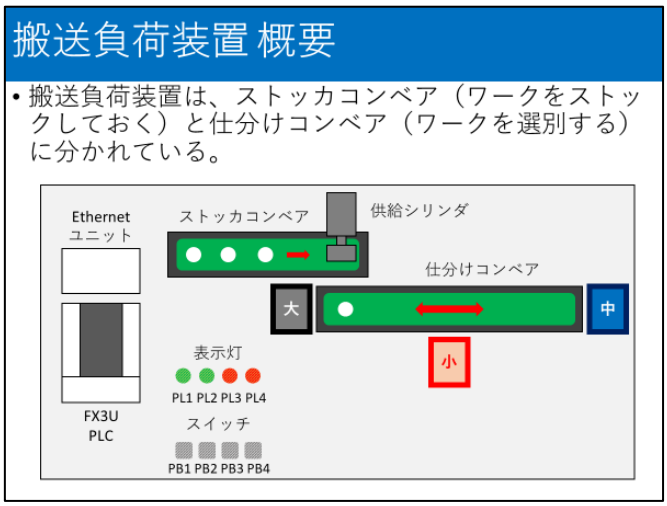


図２．１８　搬送負荷装置の各種アクチュエータ

(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．

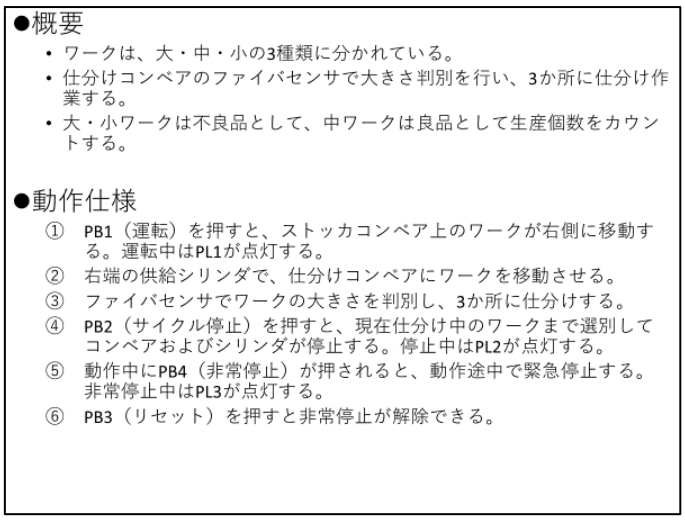


図２．１９　概要と動作仕様

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．

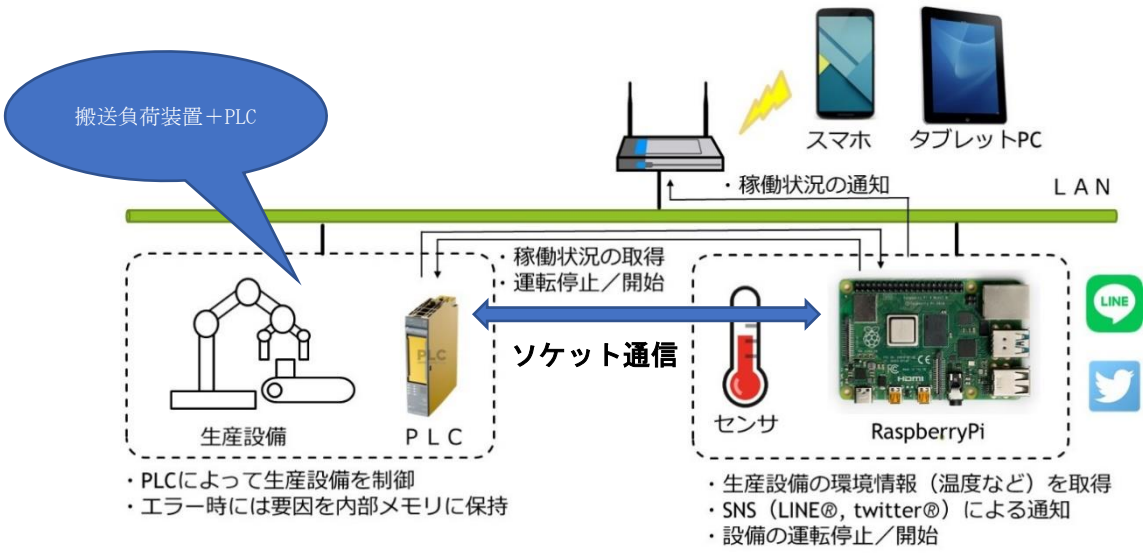


図２．２０　全体システム概要図

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．

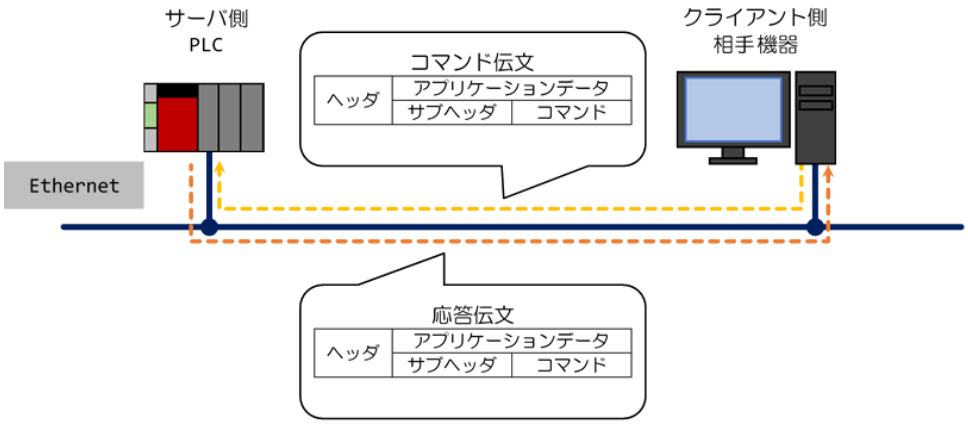


図２．２１　MCプロトコルを使用した通信

(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

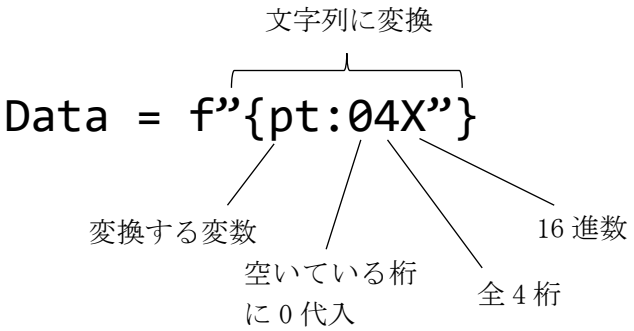
ファイルパス:/home/pi/work/socket/

(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
```

第３章　ＳＮＳを利用したＩｏＴアプリケーション開発

３．１　製造現場におけるＳＮＳ活用

ＳＮＳ（Social Networking Service）には様々なものがあり，特定の人物との会話や不特定多数の相手との連絡手段，飲食店や商業施設のＰＲなど様々な活用がされています．

ＳＮＳの中にはＡＰＩ(Application Programable Interface)が公開されているものもあり，他のアプリケーションと連携させることが容易となっているものもあります．このようにＡＰＩを活用することで生産現場におけるＳＮＳ活用の幅が広がる可能性があります．

例として製造装置の遠隔制御・監視方法を考えてみます．遠隔地から特定の制御装置には制御用コンピュータ（ＰＬＣ）にアクセスする必要があります．制御用コンピュータはネットワークからソケットを使用した特定のコマンドを受信することで内部レジスタや接点情報を取得することができます．　この場合，コマンド送受信をするためのアプリケーションがインストールされているコンピュータしかアクセスできないためセキュリティ面では強固である反面，インターフェース設計など他のプログラム言語などの知識を要します．

一方，遠隔制御・監視手段としてＳＮＳを活用すると操作方法などは普段の使用方法と変わらないため時間はかからないですし，外部ネットワークに接続するためローカルエリア内の端末以外の端末からアクセスすることも可能です．またスマートフォンの通知機能と併用すると緊急事態における端末へのサウンド、バイブレーションを活用した即時を行うことができるため迅速な対応もしやすくなります．

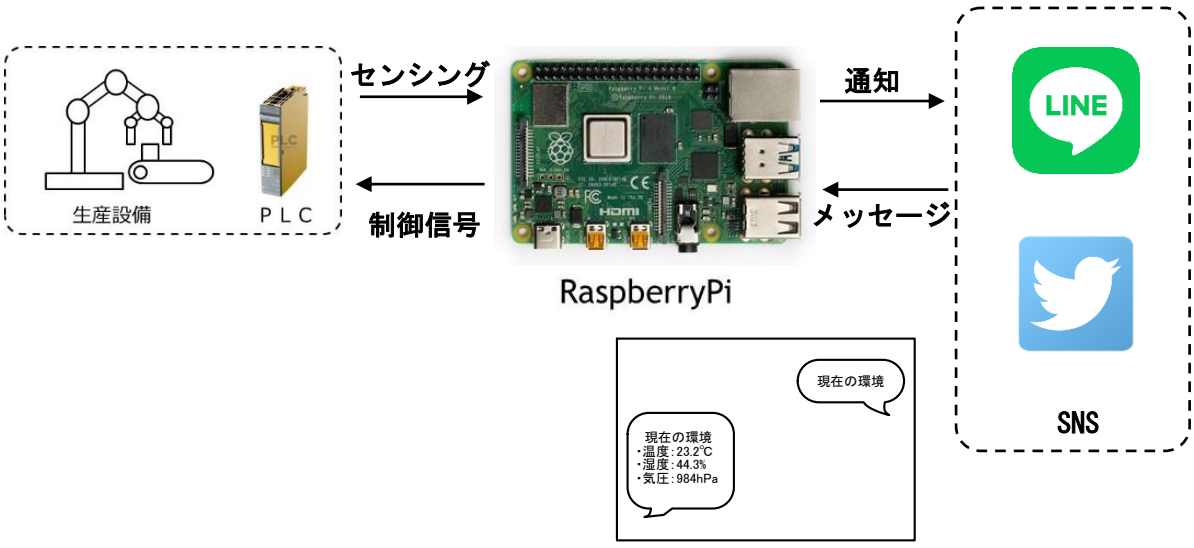


図３．１　ＳＮＳ活用のイメージ

本実習で使用する SNS は LINE®です．LINE は国内で使用されている SNS として広く知られているだけでなく，LINE bot という仕組みが公開されており API を経由して誰でも簡単に自動コミュニケーションツールを活用することができます．LINE bot を使用することでメッセージやデータ通知を利用して自動的に行うことができます．

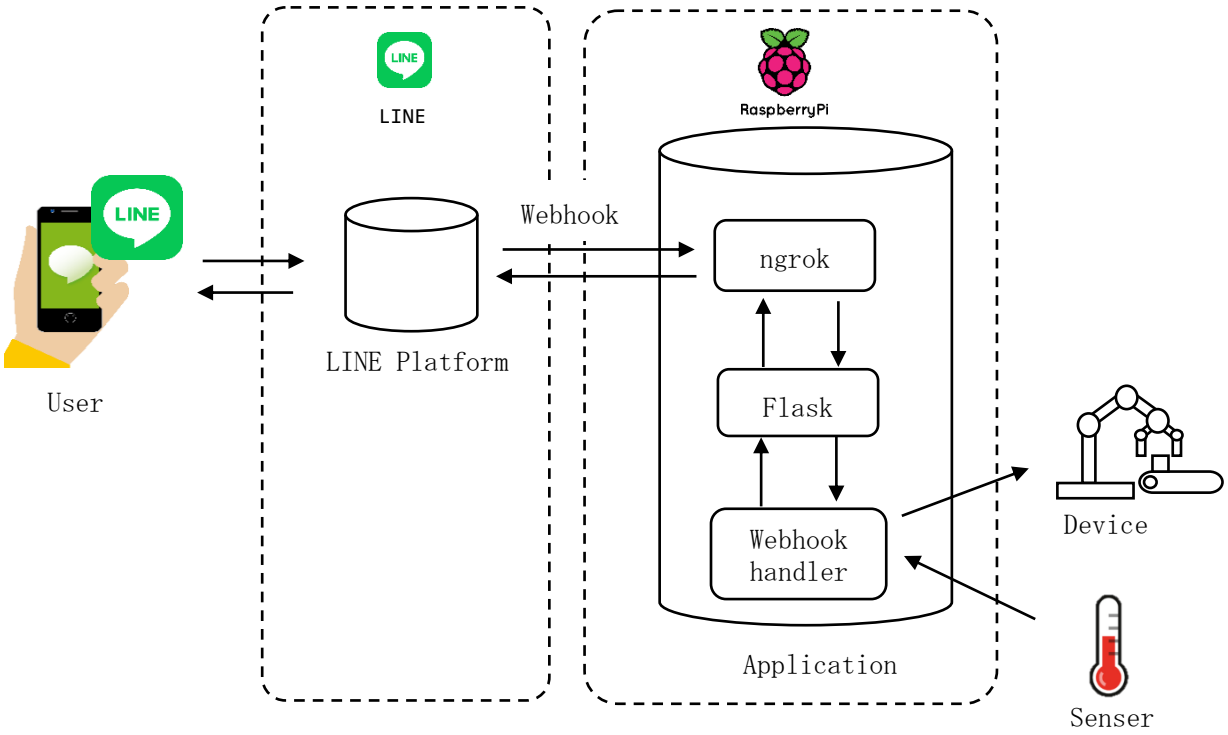


図３．２　LINE bot の仕組み

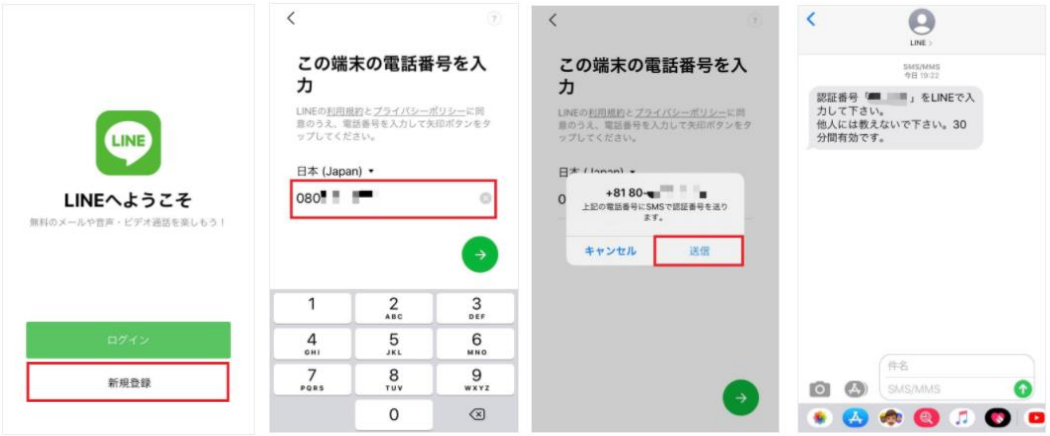
ユーザーが LINE でメッセージを送ると webhook に登録されたアドレスに HTTPS リクエストを送ります．外部公開されている URL をローカル IP に変換し(ngrok)，web フレームワーク(Flask)のコールバックルーチンを呼び出します．コールバックルーチンでは LINE が提供している MessagingAPI を利用して送信されたメッセージをアプリケーション上で使用することができます．メッセージ内容によってデバイス制御やセンサ取得ができます。

Webhook: アプリケーションの更新情報を他のアプリケーションへリアルタイム提供する仕組み
ngrok:通常はローカル環境でアクセスできる URL を外部公開するサービス
flask:Python で扱うことができる軽量 Web フレームワーク
Webhookhandler:Webhook からのイベントを受信したときに実行する処理(コールバック)

３．２　ＬＩＮＥ　ＡＰＩの環境構築

(1)LINE アカウント作成

①　②　③　④



⑤　⑥　⑦　⑧

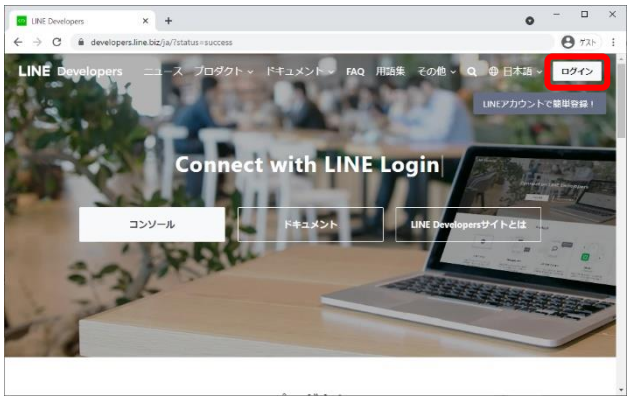
⑨　⑩　⑪　⑫

図３．３　ＬＩＮＥアカウントの取得

(2)LINE Developer のログイン

URL:https://developers.line.biz/ja/

①

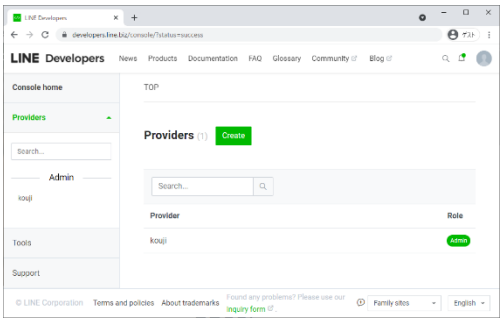


②

③

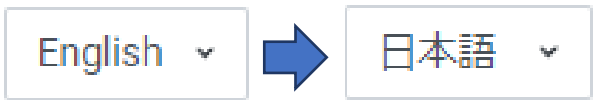
④

⑤



⑥

日本語表記に変更



⑦

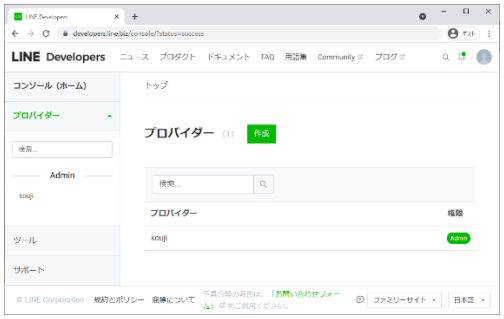


図３．４　LINE Developer へのログイン

(3)プロバイダー作成

LINE プラットフォームを通じてサービスを提供する個人や企業を登録します.

図３．５　新規プロバイダー作成

(4)MessagingAPI チャネル作成

MessagingAPI とは，LINE プラットフォームからのメッセージの通知もしくはプログラムコードで生成したメッセージを LINE プラットロームに送信する API のことです.

Messaging API で提供されるサービスには下記のものがあります.

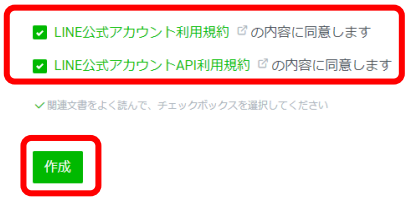
① 応答メッセージを送る
② プッシュメッセージを送る
③ その他さまざまなタイプのメッセージ（テキスト、スタンプ、画像、動画、音声、位置情報、イメージマップ、テンプレート）
④ ユーザーが送ったコンテンツを取得する
⑤ ユーザープロフィールを取得する
⑥ グループチャットに参加する
⑦ リッチメニューを使う
⑧ ビーコンを使う
⑨ アカウント連携を使う
⑩ 送信メッセージ数を取得する

MessagingAPI を使用するには下記の順に MessagingAPI チャネルを作成します．

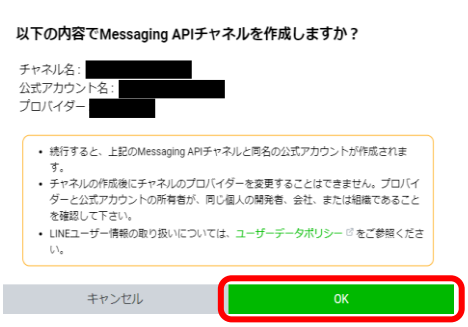
①

チャネル名（必須）大業種（必須)、小業種（必須)、利用規約への同意を記入する．

②



③



④

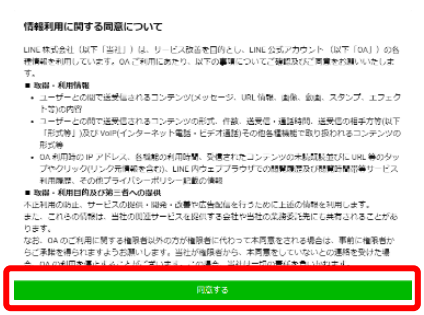


図３．６　MessagingAPI チャネル作成

(5)環境変数にチャネルアクセストークン，シークレットトークンを設定する．

LINE Developer のシークレットトークンとチャネルアクセストークンを環境変数へ代入します．各トークンは作成したチャンネルの基本設定および Messaging API 設定の画面で確認できます．

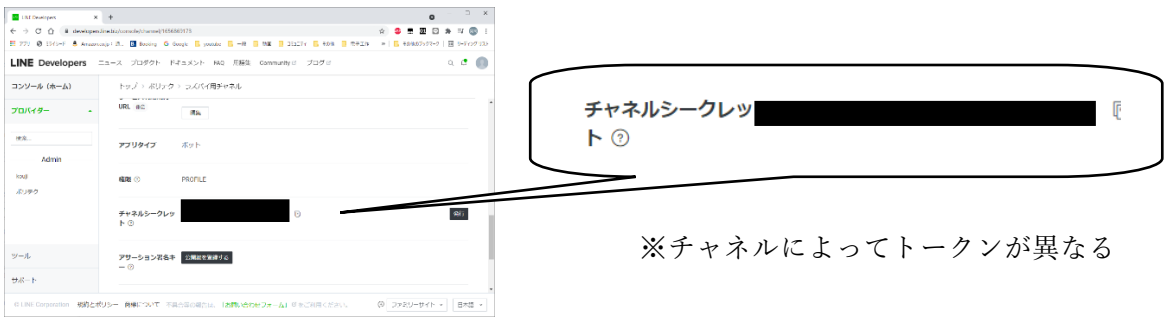


図３．７　シークレットトークンの確認

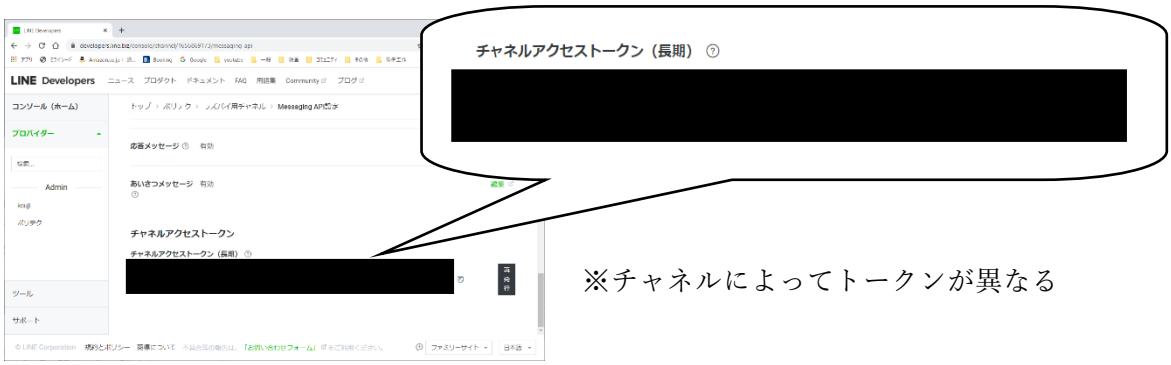


図３．８　チャネルアクセストークンの確認

①環境変数へ代入

```
$ nano ~/.bashrc
```

②.bashrc の末尾に追加する

図３．７と図３．８で確認した各トークンをコピー&ペーストします.

```
#シークレットトークン
export LINE_CHANNEL_SECRET=********************
#チャネルアクセストークン
export LINE_CHANNEL_ACCESS_TOKEN=**********************
```

③もし社内プロキシを採用しているネットワークの場合は同一の.bashrc に下記の記述を追加します.

```
export HTTP_PROXY=http://<user>:<password>@<proxyserver>:<port>
export HTTPS_PROXY=https://<user>:<password>@<proxyserver>:<port>
```

ポリテクセンター山梨ではプロキシサーバを採用しているため下記の記述のように末尾に記載します.

```
export HTTP_PROXY=http://10.0.0.2:15080
export HTTPS_PROXY=https://10.0.0.2:15080
```

(6)LINE API のインストール

```
$sudo pip3 --proxy=http://10.0.0.2:15080 install line-bot-sdk
```

(7)ngrok ダウンロードとインストール

ngrok(エングロック)とは，ローカル環境で実行している WEB アプリケーション(http,thhps,TCP)を外部公開するサービスです．ngrok には有償版と無償版があります．

有償版：プランによってひと月当たりの料金が変わります．公開される URL を自由に設定できる（つまり固定化できる）などメリットがあります．
無償版：料金がかからない代わりに，公開される URL を固定化することができません．ngrok を起動させるたびに URL がランダムに変わります．また，ダウンロードの際にユーザー登録しないと公開 URL への接続に時間制限が設けられます（２時間程度）

本実習では ngrok 無償版を使用します．必要に応じて有償版にアップグレードできます．

① 下記の URL からダウンロード・展開する

```
$wget https://bin.equinox.io/c/4VmDzA7iaHb/ngrok-stable-linux-arm.zip
$unzip ngrok-stable-linux-arm.zip
```

②ダウンロードした展開してラズベリーパイへコピー
コピー先： /usr/local/bin/

```
$sudo  mv ./ngrok  /usr/local/bin/
```

(9)ngrok 実行（本実習ではポート番号を 8080 とします）

```
$ngrok http 8080
```

ngrok によって外部公開される URL(https)を確認します．下記のように外部公開された URL を LINE Developer の Webhook URL に登録します．

TeraTarm を使用している場合，下記の網掛けになっている URL をドラックすることでクリップボードにコピーできます．

```
ngrok by @inconshreveable                                       (Ctrl+C to quit)

Session Status                online
Session Expires               1 hour, 57 minutes
Version                       2.3.40
Region                        United States (us)
Web Interface                 http://127.0.0.1:4040
Forwarding                    http:// ******************.ngrok.io -> http://localhost:8080
Forwarding                    https://******************.ngrok.io -> http://localhost:8080

Connections                   ttl     opn     rt1     rt5     p50     p90
                              0       0       0.00    0.00    0.00    0.00
```

外部公開される URL(https)をコピーする！

※この URL は ngrok（無料版）が起動するたびにランダムで与えられます.

図３．９　ngrok 起動画面

(10)Webhook URL に登録する

図３．１０　WebhookURL の設定

(11)応答メッセージを設定する

初期状態では LINEbot が受信したメッセージに対して自動返信する設定となっているため無効にします.

「応答メッセージ」－「編集」

「あいさつメッセージ」をオフ

「応答メッセージ」をオフ

図３．１１　応答メッセージ設定

(12)メッセージイベントアプリをコピー

コピー元 z:\work\LINE

コピー先 /home/pi/work/

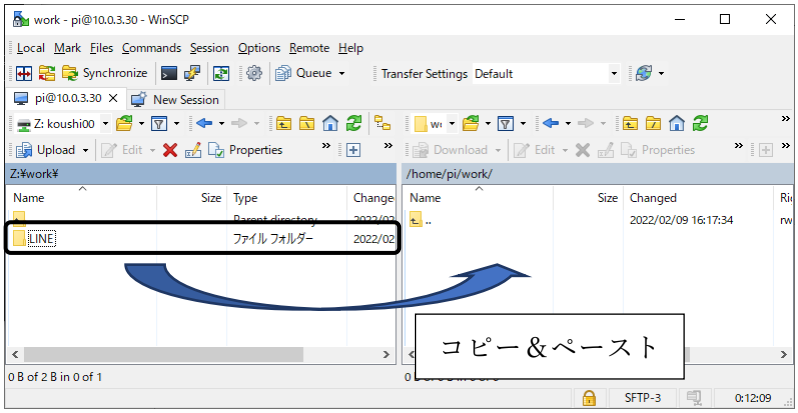


図３．１２　メッセージイベントアプリのコピー

(13)ターミナル画面を複製してメッセージイベントアプリを起動させる

①

[File]-[Duplicate session]をクリック

図３．１３　ターミナルの複製

②

```
$ cd ~/work/LINE
$ python3 ./messageEvent.py
```

(14)QR コード読み取り

スマートフォンもしくはタブレット PC の LINEbot の QR コードを読み取ります

図３．１４　チャネル登録

(15)作成したチャネルにメッセージを送る

メッセージを送ると同じメッセージが送り返されることを確認します。

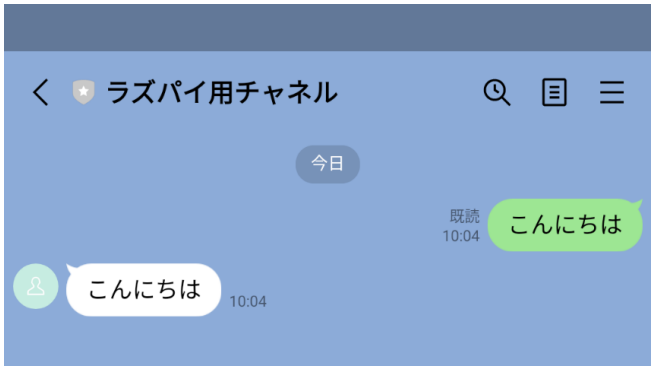


図３．１５　メッセージ送信

３．３　ＬＩＮＥ　ＡＰＩ（メッセージイベント）

サンプルコードを例にそれぞれの LINE API を解説します.

(1)LINE bot ライブラリのインポート

LINE API を使用するには下記のように LINE bot ライブラリをインポートします．非常に多くのライブラリやモジュール（クラス）が存在します．LINEbot ライブラリの中の２つのライブラリ(LineBotApi, WebhookHandler)をインポートします.

```
from linebot import (
    LineBotApi, WebhookHandler
)
```

(2)各種イベントモジュールのインポート

```
from linebot.models import (
    MessageEvent, TextMessage, TextSendMessage,
    ……
)
```

WebhookHanlder メソッドに必要なイベントモジュールは下記の通りです.

linebot.models:LINEbot の各種イベントを定義しているライブラリ

MessageEvent:メッセージを受信した際に通知されるイベント

TextMessage:メッセージオブジェクト

TextSendMessage:LINE プラットフォームにメッセージ送信する関数

(3)チャネルアクセストークンとチャネルシークレットトークンを読み込み

　チャンネルアクセストークンはあらかじめ環境変数に代入されているためコードのなかで読み込みします.

```
import os
……

#LINE チャネルシークレットトークン環境変数読み込み
channel_secret = os.getenv('LINE_CHANNEL_SECRET', None)

#LINE チャネルアクセストークン環境変数読み込み
channel_access_token = os.getenv('LINE_CHANNEL_ACCESS_TOKEN', None)
```

(4)LINE bot API のインスタンス生成

LINE bot API のインスタンスを生成するにはチャンネルアクセストークンを必要とします.

```
line_bot_api = LineBotApi(channel_access_token)
```

(5)WebhookHandler インスタンス生成

WebhookHandler のインスタンスを生成するにはチャネルシークレットトークンを必要とします.

```
handler = WebhookHandler(channel_secret)
```

(6)電子署名の検証ルーチン（コールバックルーチン）

リクエストが LINE プラットフォームから送られてきたものなのか検証するため，イベントの通知をする前に電子署名を検証します．検証するためには「X-Line-Signature」が含まれている HTTPS リクエストヘッダを取得します.

```
@app.route(“/callback”, methods=[‘POST’])
def callback():
    # HTTP リクエストのヘッダを取得
    signature = request.headers['X-Line-Signature']

    # HTTP リクエスト(POST)のボディを取得
    body = request.get_data(as_text=True)
    app.logger.info("Request body: " + body)

    # 署名を検証し、問題なければ@handler.add に定義している関数を呼び出す
    try:
        handler.handle(body, signature)
    except InvalidSignatureError:
        abort( 400 )

    return 'OK'
```

WebhookHandler メソッドの呼び出し（署名の検証も同時に行う）

(7)WebhookHandler メソッドの追加

自作関数を WebhookHandler メソッドに追加するには@handler.add を記述します.
handler.add には通知するイベント(MessageEvent)とメッセージオブジェクト(TextMessage)を引数として渡します．メッセージオブジェクトは自作関数の引数(event)に引き継がれます.

```
@handler.add(MessageEvent, message=TextMessage)
def handl_text_message(event):
    ……
```

自作関数(WebhookHandler)に登録

(8)LINE からのメッセージ取得

LINE からメッセージを取得するには HTTP リクエストボディから取得する必要があります．自作関数の引数(event)には HTTPS リクエストボディが代入されています．

```
@handler.add(MessageEvent, message=TextMessage)
def handl_text_message(event):
    #メッセージを取得
    text = event.message.text

    ……
```

LINE からのメッセージは文字列で取得できます．従って，これ以降の処理は通常の文字列処理として分岐や解析を行うことが可能です．

(9)LINE へのメッセージ送信

LINE に対してメッセージを送信するには LINEbotAPI における reply_message メソッドを使用します．reply_message メソッドには応答用トークンとメッセージ関数として TextSendMessage 関数を使用します．

```
@handler.add(MessageEvent, message=TextMessage)
def handl_text_message(event):

   ……

   #LINE にメッセージ送信
   line_bot_api.reply_message(
      event.reply_token,      #リクエスト応答用トークン
      TextSendMessage(text)  #送信メッセージ関数（メッセージをセット）
   )
```

送信メッセージは固定の文字列だけでなく，センサ情報を文字列に変換することで送信することができます．

```
@handler.add(MessageEvent, message=TextMessage)
def handl_text_message(event):
   #環境センサ(BME280)から温度,気圧,湿度データの取得(float 型)
   tmp,pre,hum = bme280.readData()

   #LINE にメッセージ送信
   line_bot_api.reply_message(
      event.reply_token,          #リクエスト応答用トークン
      TextSendMessage(f"{tmp:.1f}")   #温度データ(少数第 1 位)を文字列に変換しセットする
   )
```

(10)メッセージ送受信シーケンス

ユーザーが LINE にメッセージを送信しメッセージが送り返されるまでの流れを下記の図に示します．ユーザーがメッセージを送信すると LINE から Webhook を通じて ngrok で公開されている端末に HTTPS リクエストを送ります．Web フレームワークによってエンドポイントに指定されているコールバック関数が実行されます．コールバック関数では電子署名の検証と WebhookHandler の呼び出しを行い，”OK”を戻します．登録された WebhookHandler では受信メッセージの解析や周辺回路の制御を行い，必要に応じてメッセージを返します(line_bot_api.reply_message メソッド)．

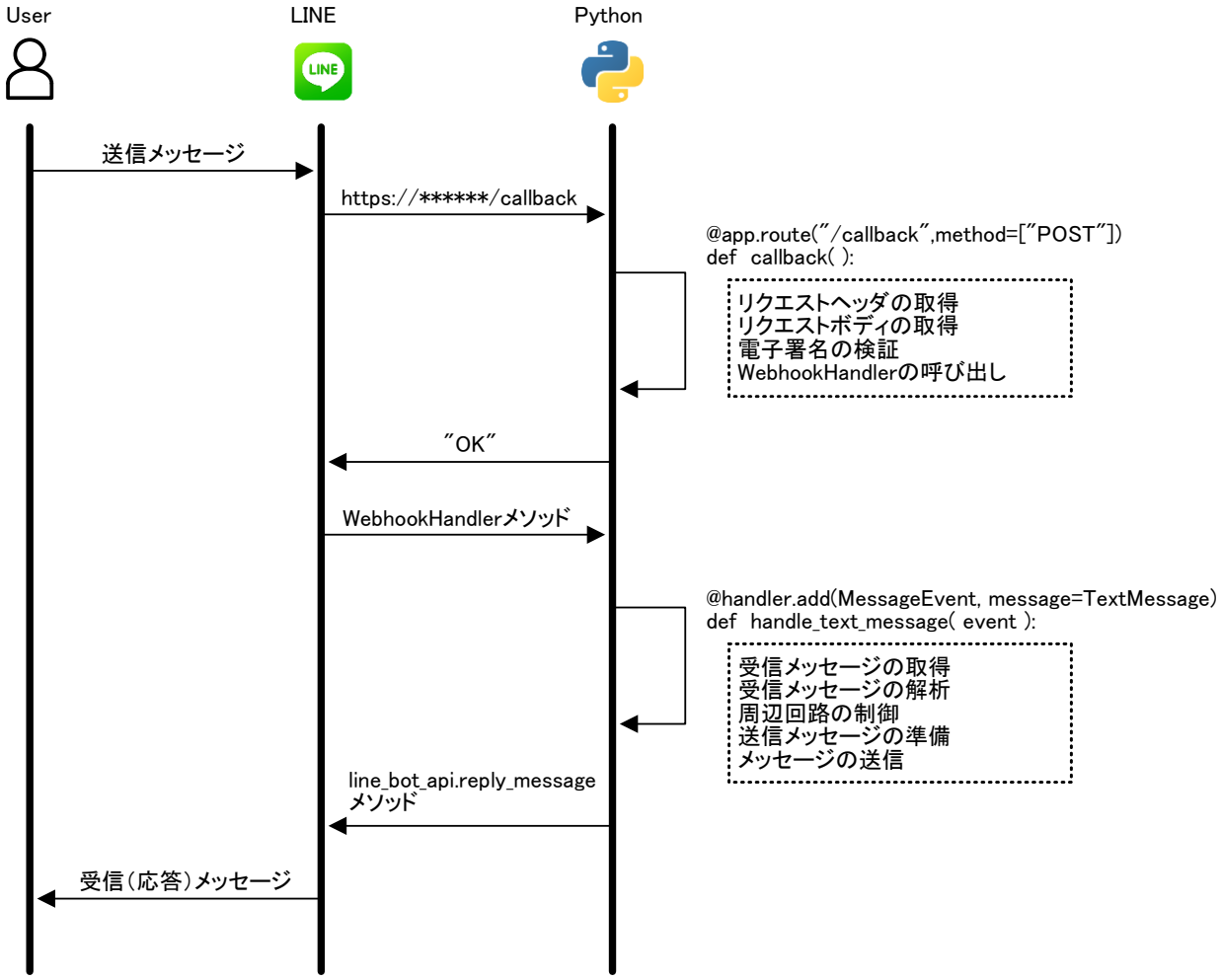


図３．１６　LINEbot におけるシーケンス図

３．４　練習問題（システム内部の温度取得）

(1)　仕様

ラズベリーパイ用 LINE チャネルに対して「温度を表示」とメッセージを表示すると「現在　○○℃です」と表示し，それ以外のメッセージを受信した場合「わかりません」と表示する LINEbot を作成しましょう．

(2)ファイル名とファイルパス

ファイル名：tempEvent.py

ファイルパス：/home/pi/work/LINE/

(3) 温度センサ

本課題ではラズベリーパイに接続されているサンハヤト製拡張 IO ボードを使用します．実装されている温度センサ TMP102 から温度を取得し，LINE メッセージに送信しましょう．

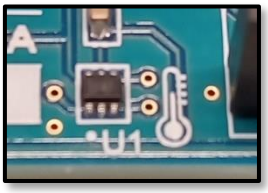


図３．１７　温度センサ(TMP102)

```
#温度センサライブラリのインポート
from tmp102 import TMP102

……

#インスタンス生成
tmp = TMP102()

#温度取得(float 型)
t=tmp.readTemperature()
```

(4)実行例

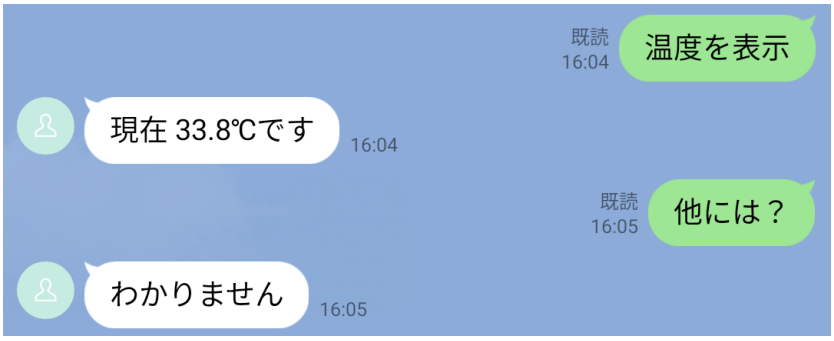


図３．１８　システム温度の問い合わせ例

３．５　搬送負荷装置ライブラリ

第１章で使用した MC プロトコルを使用し負荷装置の稼働状況を取得します．ただし MC プロトコルは要求伝文（パケット）が複雑であるため，LINEbot プログラムの中にすべて記述すると可読性が悪くなるうえ不具合発生時のデバッグに時間がかかります．

本実習ではあらかじめ搬送負荷装置の各種情報の取得と設定をするための専用ライブラリを用意しました．

(1)ライブラリインポート

```
from PLCCOM import PLCCOM
```

(2)ライブラリ解説

①インスタンス生成

関数名	PLCCOM(IPAddress, PORT)
引数	IPAddress: PLC の IP アドレス(str 型) PORT: PLC のポート番号(int 型)
戻り値	PLC と通信するインスタンス
概要	PLC と通信するインスタンスを生成する

使用例	from PLCCOM import PLCCOM
	
	#インスタンス生成
	plc = PLCCOM(“10.0.11.220”, 5001)
	…

②稼働状況の取得

関数名	read_machine_state()
引数	なし
戻り値	“STOP”　　: 停止中 “DRIVE”　　: 稼働中 “EMERGENCY” : 非常停止中 “RESET”　　: 原点復帰中
概要	搬送負荷装置の状態を取得する

使用例	from PLCCOM import PLCCOM
	
	#インスタンス生成
	plc = PLCCOM(“10.0.11.220”, 5001)
	
	#装置の状態を取得
	state = plc.read_machine_state()
	
	#もし機械が非常停止中だったら…
	if state==”EMERGENCY”:
	　……

③生産目標数の取得

関数名	read_product_target()
引数	なし
戻り値	生産目標数(int 型)
概要	生産目標数を取得する

使用例	from PLCCOM import PLCCOM
	
	#インスタンス生成
	plc = PLCCOM(“10.0.11.220”, 5001)
	
	#装置の生産目標数を取得
	ptget = plc.read_product_target()
	
	#現在の生産目標数の表示
	print(f“現在の目標数は{ptget}個です.”)

④生産数の取得

関数名	read_product()
引数	なし
戻り値	生産数(int 型)
概要	生産数を取得する．生産数とは良品の数と不良品の数を加算した値.

使用例	from PLCCOM import PLCCOM
	
	#インスタンス生成
	plc = PLCCOM(“10.0.11.220”, 5001)
	
	#装置の生産目標数を取得
	pt = plc.read_product()
	
	#現在の生産目標数の表示
	print(f“現在の生産数は{pt}個です.”)

⑤良品数の取得

関数名	read_good_product()
引数	なし
戻り値	良品の数(int 型)
概要	良品の数を取得する.

使用例	from PLCCOM import PLCCOM
	
	#インスタンス生成
	plc = PLCCOM(“10.0.11.220”, 5001)
	
	#装置の良品数を取得
	good_pt = plc.read_good_product()
	
	#現在の良品数の表示
	print(f“現在の良品数は{good_pt}個です.”)

⑥不良品数の取得

関数名	read_defective_product()
引数	なし
戻り値	良品の数(int 型)
概要	良品の数を取得する.

使用例	from PLCCOM import PLCCOM
	
	#インスタンス生成
	plc = PLCCOM(“10.0.11.220”, 5001)
	
	#装置の不良品を取得
	defective_pt = plc.read_defective_product()
	
	#現在の不良品の表示
	print(f“現在の良品数は{defective_pt}個です.”)

３．６　練習問題（搬送負荷装置における稼働状況の問い合わせ）

(1)　仕様

ラズベリーパイ用 LINE チャネルに対して「稼働状況」とメッセージを送信すると搬送負荷装置の稼働状況を「現在〇〇です」と表示し，それ以外のメッセージを受信した場合「わかりません」と表示する LINEbot を作成しましょう.

(2)ファイル名とファイルパス

ファイル名：machineStateEvent.py

ファイルパス：/home/pi/work/LINE/

(3)実行例

図３．１９　LINE による稼働状況の問い合わせ例

(4)記述例

```
@handler.add(MessageEvent, message=TextMessage)
def handle_text_message(event):
    #LINE から受信した文字を代入
    text = event.message.text

    if text=="稼働状況":

        #搬送負荷装置ライブラリのインスタンス生成
        plc=PLCCOM("???.???.???.???", ????)
        #稼働状況の取得
        state = plc.??????????????????()

        #機械の稼働状況を判断、メッセージをセット
        if state=="???????":  #もし state が”STOP”だったら…
                message = "現在停止中です"
        elif state=="???????":  #もし state が”DRIVE”だったら…
                message = "現在稼働中です"
        elif state=="???????":  #もし state が”EMERGENCY”だったら…
                message = "現在緊急停止中です"
        else:
                message = "システムエラー"

        #稼働状況を LINE で送り返す
        line_bot_api.reply_message(
                event.reply_token,              #リクエスト応答用トークン
                TextSendMessage(text=message)#メッセージ送信
        )

    else:
        #「わかりません」を LINE で送り返す
        line_bot_api.reply_message(
                event.reply_token,              #リクエスト応答用トークン
                TextSendMessage("わかりません")        #メッセージ送信
        )
```

(5)追加仕様

「稼働状況」のメッセージを送信すると搬送負荷装置の稼働状況を「現在○○です」と表示するとともに生産数，良品数，不良品数を表示するように改良しましょう．

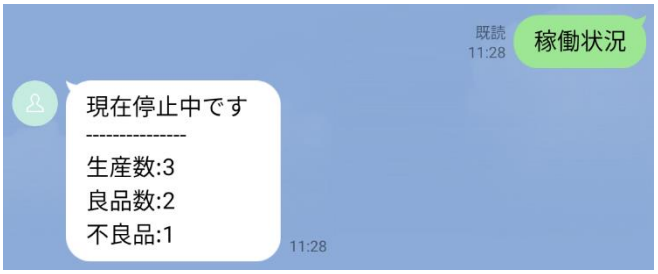


図３．２０　稼働状況と生産状況の問い合わせ例

３．７　通知機能の実装

これまでの LINE を使用した稼働状況の取得では，ユーザーが問い合わせをすることで LINEbot が応答する形式となっていました．ところがシステムエラーや何らかの要因により設備が緊急停止する状況が発生した場合は，問い合わせを待つことなく状況を通知することで手早くエラー要因を特定することができます．

LINEbotAPI にはメッセージを通知するプッシュ通知機能が実装されています．メッセージイベントのプログラムコードと比較して少ないコード量で実装できます．

(1)ユーザーＩＤの確認

プッシュ通知機能を使用するには作成したチャネルのユーザーＩＤを確認する必要があります．

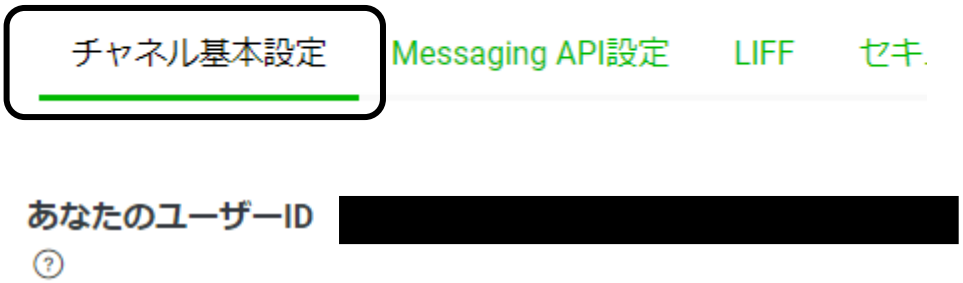


図３．２１　ユーザーＩＤの確認

(3)サンプルプログラムの編集

サンプルプログラム（pushMessage.py）にユーザーＩＤをコピー&ペーストします．

```
#LINEbotAPI インスタンス生成
line_bot_api = LineBotApi(channel_access_token)

def main():
    #ユーザーID をコピーしてください
    user_id = "****************************"

    #プッシュ通知
    line_bot_api.push_message(
        user_id,
        messages=TextSendMessage(text="こんにちは！")
    )

if  __mane__==’__main__’:
    main()
```

(4)動作例

サンプルプログラム（pushMessage.py）を実行し，チャネルにメッセージが通知されていることを確認してください.

図３．２２　メッセージ通知例

３．８　練習問題（通知機能を利用した緊急停止の通知）

(1)通知機能の処理手順

PLC に実装されている MC プロトコルは基本的に PLC 側がサーバであるため，クライアントであるラズベリーパイからコマンドによる問い合わせをすることで稼働状況を取得できます．従って LINEbot にプッシュ通知機能を実装する際にはポーリングにように一定時間ごとに稼働状況の取得処理をする必要があります.

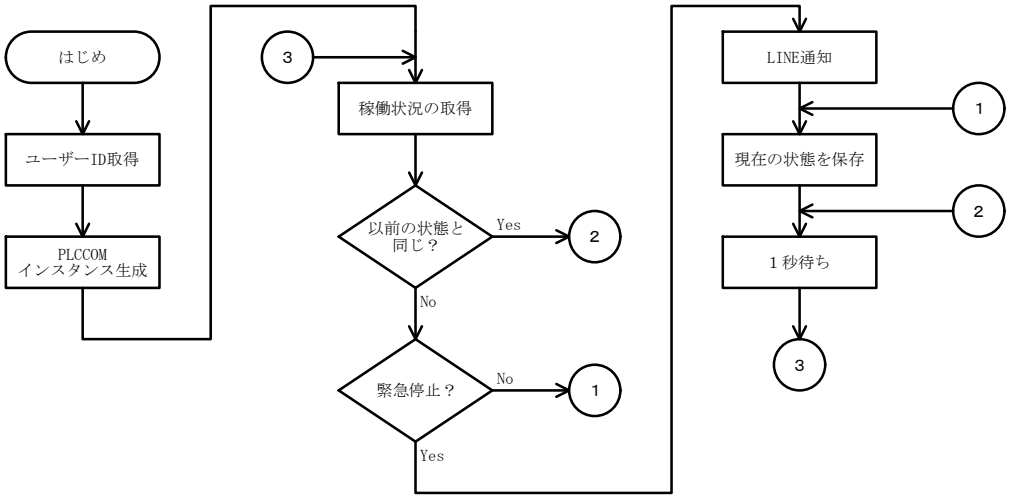


図２．２３　設備の緊急停止による通知アプリのフローチャート

(2)ファイル名，ファイルパス

ファイル名：EmergencyNotification.py

ファイルパス：/home/pi/work/LINE

(3)ヒント

```
…
#設備の保存用変数
old_state = “STOP”

while True:
   #稼働状況の取得
   new_state = ????????????????()

   if ????????????????: #もし old_state と new_state が等しくなければ…
       if ????????????: #もし new_state が”EMERGENCY”だったら…
           #LINE 通知
           ……

       old_state = new_state

   time_sleep(1.0)
```

(4)動作確認

緊急停止アプリを起動させ，設備の緊急停止ボタンを押したときに LINE に通知がされるか確認しましょう．

```
$ python3 EmergencyNotification.py
```

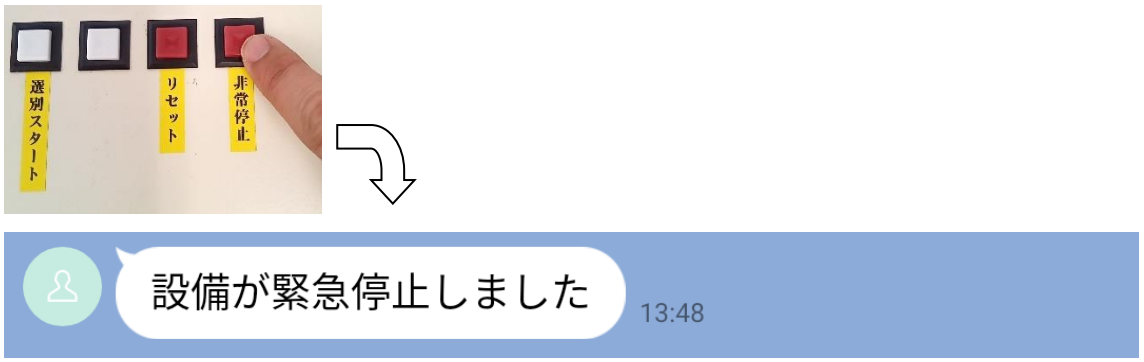


図３．２４　非常停止ボタンを押したことによる通知

(5)追加仕様

・緊急停止の通知以外に，設備が停止した日付と時刻も同時に通知してください.

<ヒント>

Python 言語でシステムの日付・時刻を取得するには datetime ライブラリをインポートします.

```
import datetime

……

#現在の日付・時刻の取得
#(〇〇〇〇/〇〇/〇〇　〇〇:〇〇)形式(文字列)として取得
days = datetime.datetime.now().strftime(“%Y/%m/%d　%H:%M”)

……
```

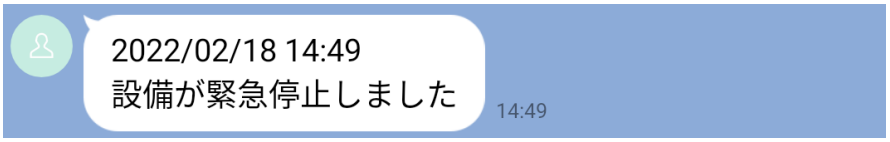


図３．２５　日付・時刻入りの通知

３．９　いろいろな機能の実装
～～ボタンテンプレートによるインターフェースの提供～～

(1)メッセージイベントの弊害

LINEbot ではユーザーからのメッセージを受信することで様々な動作をさせることができます．しかし，ユーザーは送信するメッセージが自由なため，正しいメッセージを送信しないと LINEbot が動作しないという難点があります．とくに英字の大文字と小文字(例えば”stop”と”STOP”)や同一の意味でも異なる表現(「運転状況」と「稼働状況」)があるため，ちょっとした送信間違いなどが発生しやすいといえます．

LINEbot にだれがメッセージを送信しても同一の動作をさせるために LINEbotAPI には**テンプレート**という機能が実装されています．テンプレートには様々なものがありますが今回はボタンテンプレートを紹介します．

(2)テンプレート

テンプレートとは，文字通り LINE プラットフォームにボタンや画像を表示させメッセージ送信を簡易的に行う機能です．つまり文字を１つ１つタップして打ち込む代わりに「表示されるボタンや画像をタップすることであらかじめ決まったメッセージを送信する」という解釈になります．

テンプレートには様々なものがあり，それぞれで使用するモジュールが異なります．例としてボタンテンプレートは下記のモジュールをインポートすることで使用できます．

```
from  linebot.models import (
     ……
      TemplateSendMessage, ButtonsTemplate, MessageTemplateAction,
     ……
)
```

(3)TemplateSendMessage メソッド

あらかじめ決まったメッセージを送信するために TemplateSendMessage メソッドを利用します．TemplateSendMessage メソッドには２つの引数があります．

```
message = TemplateSendMessage (
    alt_text = “****”,
    template = ……
)
```

alt_text ： 代替テキスト．テンプレートメッセージに非対応のデバイスで表示されるほか、ユーザーがメッセージを受信した際に、端末の通知やトークリストでも表示されます．
template：様々なテンプレート（今回はボタンテンプレートを使用）．ボタンテンプレー

トを使用するには TemplateSendMessage メソッドの第二引数である template に ButtonsTemplate メソッドを使用します.

(4) ButtonsTemplate メソッド

表示されるボタンは下記のようなイメージです.

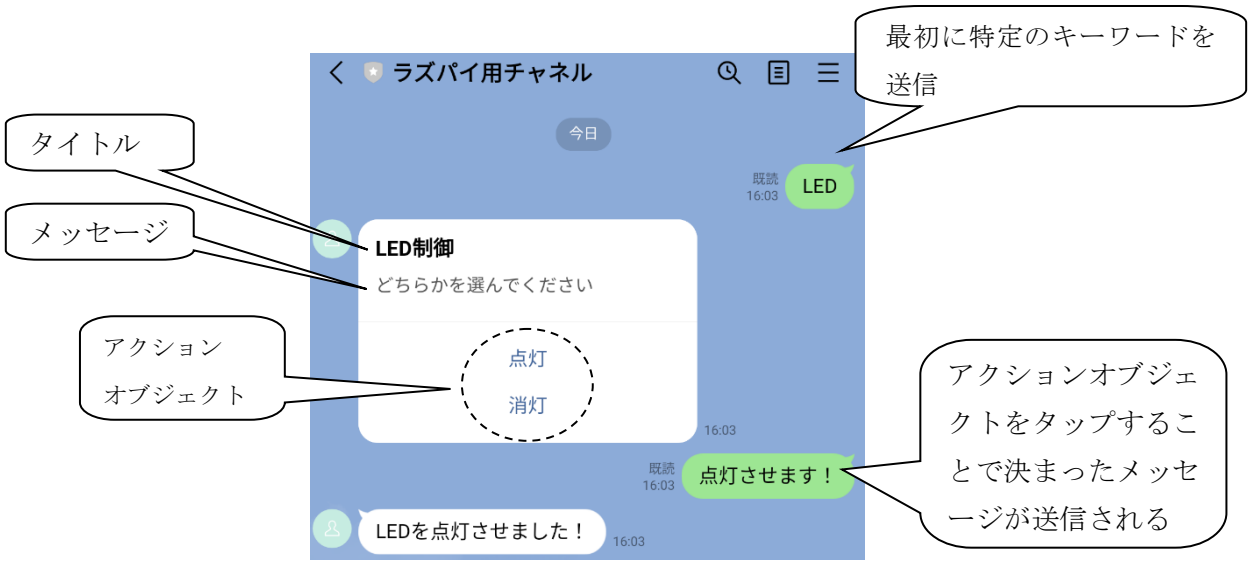


図３．２６　ボタンテンプレートによる表示

上記のようなイメージのボタンを表示させるための ButtonsTemplate メソッドは次の引数を指定します.

```
message = TemplateSendMessage (
    alt_text = “LED”,
    template = ButtonsTemplate (
        text = “どちらかを選んでください”,
        title= “LED 制御”,
        actions = [
            ……
        ]
    )
)
```

メッセージ：選択肢の意味などを説明するために用います

タイトル：端的でわかりやすいタイトルにします

アクションオブジェクト：メッセージや画像がタップした時のアクションを指定する

(5)メッセージアクション

アクションオブジェクトはリスト型で指定します．リスト内のアイテムが増えるほどタップできる選択肢（つまりボタンや画像）が増えると考えてください.

図３．２６のように特定のワードを表示し，その文字をタップすることで決まったメッ

セージを表示させるアクションのことを**メッセージアクション**といいます．メッセージアクションを使用するには MessageTemplateAction メソッドを使用します．

```
message = TemplateSendMessage (
    alt_text = “LED”,
    template = ButtonsTemplate (
        text = “どちらかを選んでください”,
        title= “LED 制御”,
        actions = [
            MessageTemplateAction (
                label=”点灯”,
                text=”点灯させます！”
            ),

            MessageTemplateAction (
                label=”消灯”,
                text=”消灯させます！”
            )
        ]
    )
)
```

タップできる表示ラベル

表示ラベルをタップすると代わりに送信されるメッセージ

MessageTemplateAction メソッド：
１つにつきメッセージ１つ

注意！：アクションオブジェクトは actions 引数にリスト形式で指定します．ただしリストのアイテム数は最大で 4 件までになります！（つまり 1 度に表示できるボタンは最大４つまで）

(6)テンプレートの送信

本来，LINEbot におけるメッセージは JSON 形式で送受信されます(それゆえに Python 以外のプログラム言語でも LINEbotAPI が存在します)．TemplateSendMessage メソッドによって JSON 形式に変更されますので，戻り値が代入された変数を送信することでテンプレートを送信できます．

```
#テンプレートセット
message = TemplateSendMessage (
    ……
)

#テンプレート送信
line_bot_api.relpy_message(
    event.reply_token,
    message
)
```

(6)生産設備管理の適用事例

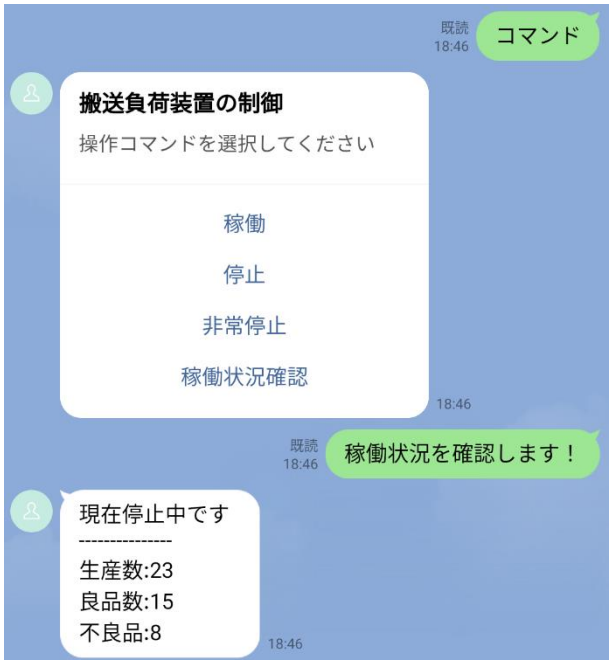


図３．２７　LINEbot を利用した生産設備管理アプリの例

――メモ――

――メモ――

参考文献

[1]みんなの RaspberryPi 入門第 4 版　石井 モルナ(著), 江崎 徳秀(著)　リックテレコム発行 ISBN-10:4865941134
[2] Raspberry Pi で遊ぼう! 改訂第 4 版　林和孝(著)　ラトルズ発行　ISBN-10:4899774354
[3] Python プログラミング逆引き大全 400 の極意 秀和システム発行 ISBN-10:4798063665
[4] Python 基礎&実践プログラミング Magnus Lie Hetland(著),松浦 健一郎(監修),司ゆき(監修), 武舎広幸(翻訳) インプレス発行　ISBN-10:4295008389

クラウド活用によるＩｏＴシステム構築技術

セミナーテキスト

作成日：２０２２年　２月２２日
更新日：２０２２年　２月２２日
作成者：(独) 高齢・障害・求職者雇用支援機構山梨支部
山梨職業能力開発促進センター
〒400-0854　山梨県甲府市中小河原 403-1
TEL 055-242-3066(代表)　TEL 055-242-3063(訓練課)
FAX 055-242-3068
URL https://www.jeed.go.jp/location/shibu/yamanashi/
https://www3.jeed.go.jp/yamanashi/poly