TOPPERS 活用アイデア・アプリケーション開発コンテスト

部門　：　がじぇるね IoT 部門

作品のタイトル　：　mROS ~組込みマイコン向け ROS ノード軽量実行環境~

作成者　：　森　智也（京都大学）

共同作業者　：　高瀬　英希（京都大学）

対象者　：　ロボットシステム開発者・組込みシステム開発者

使用する開発成果物　：　TOPPERS/ASP カーネル
https://github.com/ncesnagoya/asp-gr_peach_gcc-mbed.git

目的・狙い

ロボットソフトウェア開発支援フレームワークである ROS（Robot Operating System）をリアルタイム OS である TOPPERS/ASP カーネル上で実行可能にすることを目的として開発を行った．これにより組込みデバイスを ROS システムで活用できるようになる．

アイデア/アプリケーションの概要

ロボットソフトウェア開発において，開発支援フレームワークである ROS（Robot Operating System）が注目されている． しかし，ROS は Linux 上での動作を想定しているため，Linux が搭載できないような組込みデバイスを活用することができない． そこで，ホスト PC 上で動作する ROS システムに対する通信をサポートするライブラリを TOPPERS/ASP カーネル環境で実装することで GR-PEACH 上で ROS プログラムを実行可能にした．

1　開発環境・開発対象・開発成果物

開発環境

- Atollic TrueSTUDIO(`https://atollic.com/truestudio/`)

開発対象

- GR-PEACH
- TOPPERS/ASP カーネル
- ASP カーネル搭載 GR-PEACH 用 mbed ライブラリ (`https://github.com/ncesnagoya/asp-gr_peach_gcc-mbed.git`)

開発成果物はすべて `https://github.com/tlk-emb/mROS` で公開している．

2　アプリケーション実行

本アプリケーションでは，GR-PEACH 上で実行されるプログラムとホスト PC 上で実行される ROS システム間で通信を行う．そのため，ROS が実行可能な環境が必要となる．

2.1　アプリケーション概要

今回は，ホスト PC 上で実行されるパブリッシャノードとサブスクライバノードに対して，mROS 上では対応するサブスクライバ，パブリッシャを実行する．mROS サブスクライバは，送信される文字列に応じて GR-PEACH の LED を点灯・消灯させる．また，mROS パブリッシャは超音波距離センサによって取得されるデータを出版する．図 1 のようなシステムが実現される．

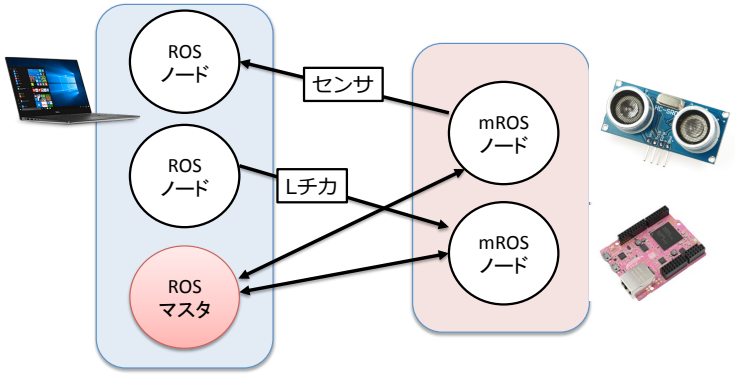

図 1: 実現されるシステム

2.2　使用する機材・環境

- 超音波距離センサ (HC-SR04)
- ホストとなる PC（今回は NEC LaVie Direct HZ を使用）
- Ubuntu 14.04 LTS

- ROS indigo

Ubuntu 14.04LTS がインストールされた PC に ROS indigo ディストリビューションを導入しておく.（参考：`http://wiki.ros.org/indigo/Installation/Ubuntu`）

2.3 アプリケーション実行手順

開発した mROS は github(`https://github.com/tlk-emb/mROS`) にて公開している.

2.3.1 mROS 側の作業

GR-PEACH 側では，上で git からダウンロードした中にある/ros_emb プロジェクトを TrueSTUDIO でビルドして asp.bin を GR-PEACH に書き込む．その際，ホスト PC の IP アドレスを ros_emb.cpp:19 で指定しておく必要がある．また，以下のように HC-SR04 をピン接続する．

HC-SR04 ピン	GR-PEACH ピン
Vcc	+5v
Trig	P2_14
Echo	P2_15
Gnd	GND

表 1: PIN アサイン

2.3.2 ホスト PC 側の作業

UbuntuPC 側では ROS の Catkin ワークスペースとなるディレクトリを作成し，/src ディレクトリに mROS の中にある/mros_test ディレクトリをコピーする.

ワークスペースディレクトリで catkin_make を行い，/mros_test をコンパイルする.

`$rosrun mros_test mros_talker` および，`$rosrun mros\_test mros_listener` が実行できることを確認する.

2.3.3 組合わせる作業

用意したホスト PC と GR-PEACH とを同一ネットワーク上に接続する.

ホスト PC では`$roscore`,`$rosrun mros_test mros_talker` および，`$rosrun mros_test mros_listener` を実行し，ROS ノードとマスタを起動する.

ROS ノードとマスタが起動した状態で，GR-PEACH 側のスイッチを押し，プログラムを実行することで ROS システムとの通信が開始され，アプリケーションが実行される.

2.4 アプリケーション実行の様子

サブスクライバが動作しているの様子を図 2(a) と図 2(b) に示す．図 2(a) では，ホスト側から red と出版されたデータを購読し，赤く LED が光り，図 2(b) では green を購読し，緑に光っている.

パブリッシャとしての動作は図 3 に示す．超音波距離センサから取得したデータを出版している.

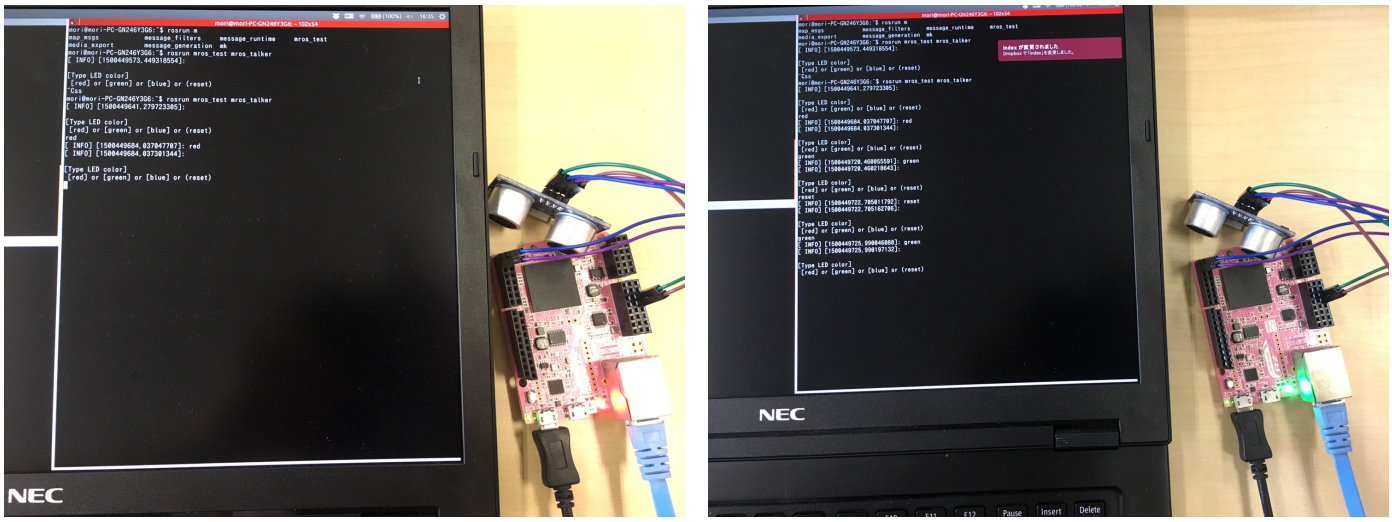

図 2: サブスクライバの様子

3　実装

mROS では，ROS の通信を実現するために複数のタスクで機能を実現した．

3.1　ROS 通信プロトコル

ROS におけるノード間通信は XML-RPC と TCPROS のプロトコルを使用して行われる．

XML-RPC による通信では，ノード情報の登録などを行い通信相手のノードを特定する．その後，TCPROS による通信でデータ転送を行う．図 4 にノード間での pub/sub 通信が開始されるまでの通信フローを示す．

サブスクライバノードが初期化されると，まず ROS マスタに自身の情報を登録する (1).

そのレスポンスとして，対応するトピックをパブリッシュするノードの IP と XML-RPC ポートが返ってくる (2).

サブスクライバノードはそのポートに対して XML-RPC でトピックのリクエストを送信する (3).

パブリッシャノードはトピックのパブリッシュを行う TCP ポートを返す (4).

サブスクライバノードは TCP ポートに対して TCPROS コネクションヘッダを送信する (5).

パブリッシャノードもコネクションヘッダを返信する (6).

その後，パブリッシャから TCPROS によってデータ転送が開始される．

mROS ではこのような一連の通信フローを mROS 通信ライブラリとして実装し，サポートすることで ROS システムとの通信を可能とし，ROS ノードとして振る舞うことを実現している．

3.2　タスク構成

今回実装したアプリケーションのタスク構成を図 5 に示す．

ユーザタスクにかかわらず共通な mROS 通信ライブラリタスク

- XML_MAS_TASK ユーザタスク中で ROS ノードの初期化が行われたとき，および SUB_TASK がトピックのリクエストを行うときに実行される．

  ホスト PC 上にある ROS マスタとパブリッシャノードに対して XML-RPC のエンコードしたデータを HTTP 転送する．

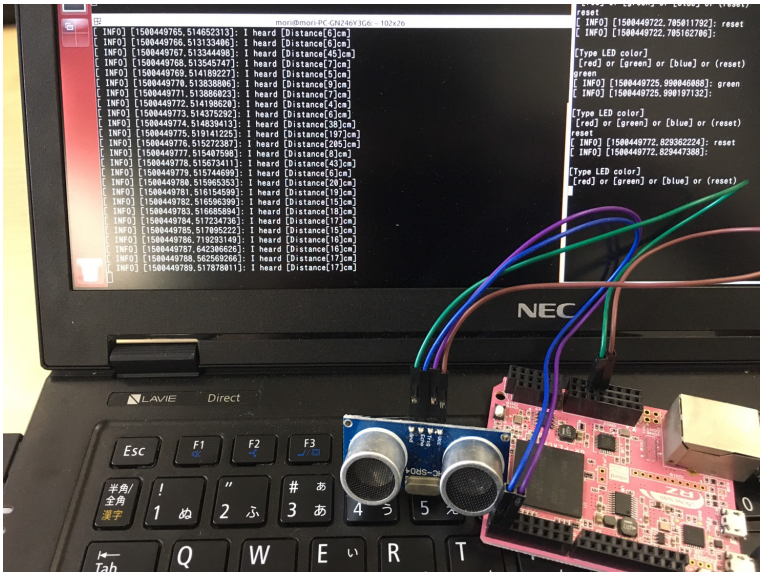

図 3: パブリッシャの様子

- XML_SLV_TASK 外部サブスクライバからのトピックのリクエストを受け付けるタスク.

  周期ハンドラ実行され，要求があればパブリッシャノードのポートを教える．

- PUB_TASK データをパブリッシュするタスク.

  ユーザタスクからパブリッシュするデータが送信されると待ち状態から解除され，実行される．

- SUB_TAKS データをサブスクライブするタスク.

  ユーザタスク中で ROS ノード初期化が行われたときにコールバック関数のポインタを受け取り，サブスクライブしたデータに対して実行する．周期ハンドラで周期的にサブスクライブ処理が行われる．

アプリケーションを記述するユーザタスク

- usr_task1 本アプリケーションではサブスクライバノードの記述を行っている．ROS の関数をマッピングしているため，ROS のノードを記述するために使用される関数がそのまま使用できる．コールバック関数として L チカを行う関数を指定している．

- usr_task2 本アプリケーションではパブリッシャノードの記述を行っている．本アプリケーションでは，超音波距離センサの値を取得し文字列として ROS サブスクライバにパブリッシュを行う．

また，mROS におけるタスク間通信にはデータキューと共有メモリを使用している．データキューで扱うデータは 32bit 使用し，8 ビットでノードを識別する ID，8 ビットで共有メモリのアドレスを指定するオフセット，16 ビットでデータ長を示している．

タスク優先度では，mROS 通信ライブラリタスクがユーザタスクより高位に設定しているが，PUB_TASK,XML_MAS_TASK ではデータキューの受信でブロックされる．また，XML_SLV_TASK，SUB_TASK では周期的に動作させることでユーザタスクを含めたタスクすべてが実行されるように設計している．

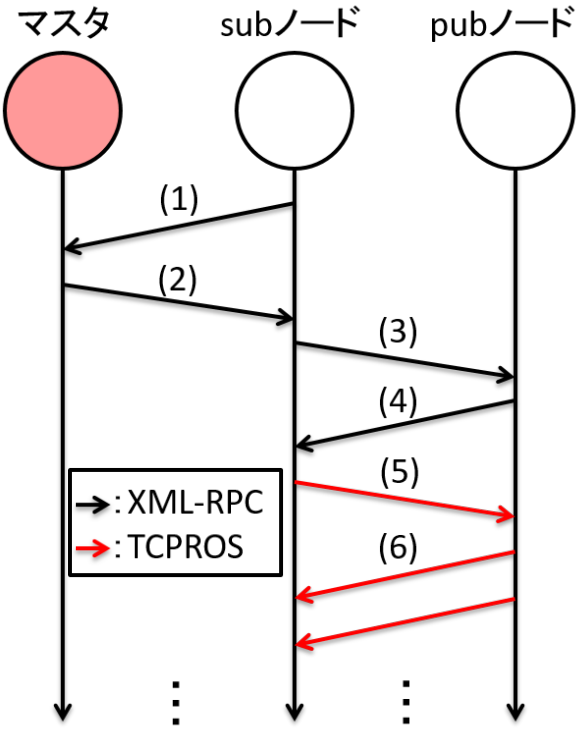

図 4: ROS データ通信

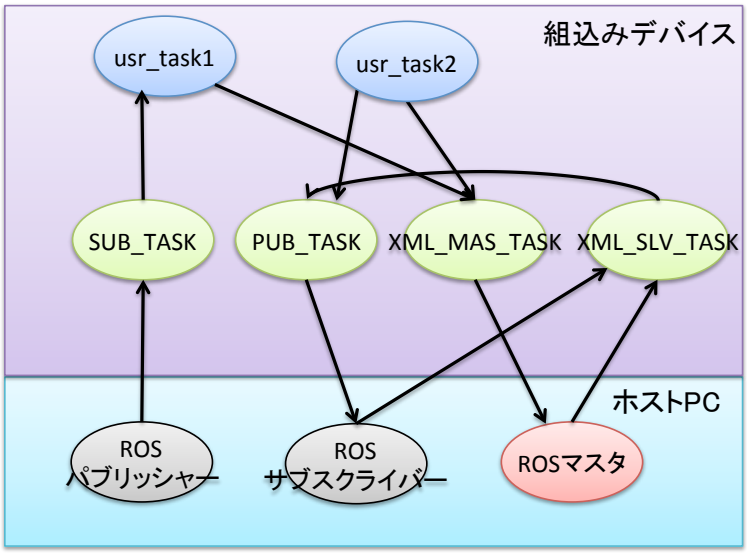

図 5: アプリケーションのタスク構成