ETロボコン向けTOPPERS活用セミナー

EV3RTを使った
アプリケーション開発の始め方

2019年6月15日

松原豊（名古屋大学）

EV3RTのインストール

- 開発環境をホストPCにインストール
  - http://dev.toppers.jp/trac_user/ev3pf/wiki/DevEnv の「開発環境(クロスコンパイラ，ツール)のインストール」を参考に
  - **動作確認済のバージョンのコンパイラを使うことを推奨**
- パッケージをダウンロード
  - http://dev.toppers.jp/trac_user/ev3pf/wiki/Download から β7-2(ev3rt-beta7-2-release.zip)を取得
- パッケージを解凍

```
$ unzip ev3rt-beta7-2-release.zip
```

- カーネルソースコードを解凍

```
$ cd ev3rt-beta7-2-release
$ tar xvf hrp2.tar.xz
```

EV3RTパッケージのフォルダ構成

ファイル／フォルダ名	内容
Changelog.txt	リリース情報．変更履歴.
EV3RT_C_API_Reference/	EV3RTのC APIリファレンス（index.htmlをブラウザで開くと閲覧可能）
EV3RT_CPP_API_Reference/	EV3RTのC++ APIリファレンス
ngki_spec-171.pdf	TOPPERSカーネルの仕様書（Ver.1.7.1）
sdcard/	• EV3RTのカーネルやアプリケーションのイメージファイル（SDカードに保存し，EV3本体に挿入することで使用） • EV3RTの動作設定ファイル（次項で説明） • サンプルプログラムが使用するファイル
hrp2/	EV3RTのソースコード本体（後ろで説明） • HRP2カーネル • デバイスドライバやミドルウェア • アプリケーションのワークスペース

SDカードに置くファイルのサンプル

sdcardフォルダの中身

ファイル／フォルダ名		内容
uImage		EV3RTのカーネルと動的ローダのイメージファイル（EV3に電源を入れると，このファイルを使って起動）
ev3rt/		EV3RTが使用するディレクトリ
	apps/	アプリケーションのロードイメージを置くフォルダ
	etc/rc.conf.ini	各種設定ファイル
	res/	サンプルアプリケーション（ファイルI/O）で使用しているフォルダ．プログラムでは「/ev3rt/res」というパスでアクセスできる．

設定ファイルでできること

動作ログの出力先の変更

rc.conf.ini

```
[Debug]
# DefaultPort = UART ← UARTを使用する場合
DefaultPort = BT ← Bluetoothを使用する場合
# DefaultPort = LCD ← LCDを使用する場合
```

無効化

※何も指定しない場合の出力先はLCD

Bluetoothの接続情報の変更

rc.conf.ini

```
[Bluetooth]
LocalName=Mindstorms EV3
PinCode=0000
```

ホストPCから見える
EV3本体のデバイス名

ペアリング時の
ピンコード

設定ファイルでできること

その他の設定項目

rc.conf.ini

```
[Bluetooth]
DisablePAN=0
IPAddress=10.0.10.1
[Sensors]
DisablePort1=1
[USB]
AutoTerminateApp=1
[Debug]
LowBatteryWarning=0
```

Bluetooth PANの無効化
EV3のIPアドレス

センサポート１の無効化
（シリアルとして使用）

USB接続した時、アプリを終了

バッテリ低下のアラーム機能

EV3RTのカーネル（hrp2フォルダ）の内容

アプリケーション開発に関係しそうなもののみ

フォルダ名	内容
arch, include, extension, kernel, library, pdic, syssvc, target	HRP2カーネルのソースコード. デバイスドライバなどは target/ev3_gcc/ の下
base-workspace	アプリケーションローダ
sdk/workspace	アプリケーション開発用ワークスペース サンプルアプリケーション
cfg	EV3RTでのアプリケーション開発に必要なツール（静的API→Cコード生成） Windows以外の環境では，cfgのバイナリを入れ替える必要がある．http://www.toppers.jp/cfg-download.html から環境にあったバイナリをダウンロードし，cfg/cfg/ に置く．
configure	アプリケーションのMakefileをテンプレートから生成するユーティリティ

EV3RTの基本的な起動シーケンス

1. EV3の電源を入れる

2. EV3のメモリに書き込まれているブートローダ（uboot）が起動する

3. SDカードにあるuImageファイルを，EV3のSDRAMに展開して実行する

アプリケーションの実行形式

実行形式	実行モジュール	アプリ実行方法	アプリ更新方法
スタンドアローン	EV3RT（カーネル）と１つのアプリをまとめた実行モジュール（uImage）	EV3RT起動後に，１つのアプリを実行	SDカードにあるuImageを更新。アプリを変更する度に，uImageを再ビルド＆SDカードにコピーする必要あり
動的ローディング	EV3RT＋アプリローダ*（uImage）と，アプリ（デフォルト名app，変更可）が別々	EV3RT起動後にアプリローダが起動。アプリローダ上で，アプリを選択して実行	アプリの実行モジュールだけ更新すれば良い

一般的な用途（例：EV3用制御アプリを開発）では，アプリ更新が高速かつ容易な動的ローディングがオススメ。高度な用途（例：大規模なアプリ開発，RTOS機能をフル活用）ではスタンドアローン（詳細は，次セッションで解説）。

*スタンドアローン形式で提供されるアプリの一種（カーネルソースコード内のbase-workspaceに存在）

アプリケーションのビルド

- EV3RTのworkspaceフォルダに移動

```
$ cd ev3rt-beta7-2-release/hrp2/sdk/workspace/
```

- アプリケーションをビルド

a) スタンドアローン形式のモジュールをビルドする場合は make img=<フォルダ名>

```
$ make img=helloev3
```

uImage というファイルが生成される

b) 動的ローディング形式のモジュールをビルドする場合は make app=<フォルダ名>

```
$ make app=helloev3
```

app というファイルが生成される

アプリケーションのビルド

- スタンドアローン形式が成功した場合

```
~/ev3rt-beta7-1-release/hrp2/sdk/workspace
  CC      ../common/ev3api/src/ev3api_speaker.c
  CC      ../common/ev3api/src/ev3api_lcd.c
  CC      ../common/ev3api/src/ev3api_motor.c
  CC      ../common/ev3api/src/ev3api_newlib.c
  CC      ../common/ev3api/src/ev3api_sensor.c
  CC      ../workspace/helloev3/cli_main.c
  CC      ../workspace/helloev3/cli_sensor.c
  CC      ../workspace/helloev3/cli_motor.c
  CC      kernel_cfg.c
  CC      kernel_mem2.c
  CC      kernel_mem3.c
cfg: warning: 17% of the allocated page table area is used
cfg: warning: 4/130 of the allocated nonglobal page entries area is used
check complete
  CC      kernel_mem.c
Image Name:   TOPPERS/hrp2 Kernel (EV3)
Created:      Fri Jun 15 19:39:01 2018
Image Type:   ARM Linux Kernel Image (uncompressed)
Data Size:    629740 Bytes = 614.98 kB = 0.60 MB
Load Address: c0008000
Entry Point:  c0008000

liyixiao@NCES-VAIO-LYX ~/ev3rt-beta7-1-release/hrp2/sdk/workspace
$
```

アプリケーションのビルド

- 動的ローディング形式が成功した場合

~/ev3rt-beta7-1-release/hrp2/sdk/workspace

```
  CC      ../../library/t_perror.c
  CC      ../../library/strerror.c
  CC      ../../library/vasyslog.c
  CC      ../../target/ev3_gcc/TLSF-2.4.6/src/tlsf.c
  CC      app.c
  CC      ../common/ev3api/src/ev3api.c
  CC      ev3api_cfg.c
  CC      ../common/ev3api/src/ev3api_battery.c
  CC      ../common/ev3api/src/ev3api_brick.c
  CC      ../common/ev3api/src/ev3api_fs.c
  CC      ../common/ev3api/src/ev3api_speaker.c
  CC      ../common/ev3api/src/ev3api_lcd.c
  CC      ../common/ev3api/src/ev3api_motor.c
  CC      ../common/ev3api/src/ev3api_newlib.c
  CC      ../common/ev3api/src/ev3api_sensor.c
  CC      ../workspace/helloev3/cli_main.c
  CC      ../workspace/helloev3/cli_sensor.c
  CC      ../workspace/helloev3/cli_motor.c
  LD      app
make[1]: Leaving directory '/home/liyixiao/ev3rt-beta7-1-release/hrp2/sd
k/OBJ'

liyixiao@NCES-VAIO-LYX ~/ev3rt-beta7-1-release/hrp2/sdk/workspace
$
```

アプリケーション実行方法：スタンドアローン形式

1. 実行モジュール（uImage）を make img で生成
2. uImage をSDカードのトップに置く
3. EV3を起動（中央ボタンを押す）
4. 「Run App」と画面下部に表示されているときに，　中央ボタンを押すとアプリケーションが起動

アプリケーションの実行方法：動的ローディング形式

1. アプリケーションのモジュールを make app で生成
2. app をSDカードの/ev3rt/apps/に置く
   - EV3RT起動中に，Bluetooth/USB/シリアルケーブル経由でアプリケーションモジュールを転送可能
   - http://dev.toppers.jp/trac_user/ev3pf/wiki/SampleProgram#PCからEV3へのアプリケーションのロード方法の選択
3. EV3を起動（中央ボタンを押す）
4. 「Load App」と画面下部に表示されているときに中央ボタンを押すと，アプリケーションローダが起動するので「SD card」を選択し，アプリケーションを選択して起動
   - 上下ボタンでカーソル移動，中央ボタンで決定
   - アプリケーション実行時にバックボタンを長押しすると，コンソールに戻る
   - コンソールで右ボタンを押して「Shutdown」と画面下部に表示されているときに中央ボタンを押すと電源を切る（左＋右＋バックボタンを同時に長押しする方法もある）

新規アプリ開発の始め方

- 簡単な方法は既存のプロジェクトをコピーする

1. cp –a ev3way-cpp new_proj

   C言語の場合は，gyroboyをベースにすると良い

2. 不要なソースコードファイルを削除し，必要なファイルを追加

   注意：app.h, app.cpp[c], app.cfg は必要なファイルなので削除しないこと！ファイル名を変える場合は，workspace/Makefileの“-A app”の部分を変更するか，sdk/common/Makefile.img[app]のAPPLNAMEを直書きすればよい

3. Makefile.incでアプリケーションの設定

   app.c以外のソースコードを追加する場合は，以下のようにビルド対象に追加する．app.cfg以外のcfgを追加する場合はapp.cfgからINCLUDEすればよい

   - APPL_COBJS += xxx.o # gccでビルドするファイル群
   - APPL_CXXOBJS += xxx.o # g++でビルドするファイル群
   - SRCLANG := c | c++ # CのみかC++ありか

タスクを追加する(1/2)

- cfgファイルにタスクを生成するための静的APIを追加
  - CRE_TSK(タスクID, { タスク属性, タスクに渡す引数, タスクの関数名, 優先度, スタックサイズ, スタックの先頭番地 })
    - タスクIDはシステムサービスのパラメータとなるマクロ識別子
    - タスク属性は初期状態（TA_ACT：起動，TA_NULL：休止）
    - 優先度は0に近いほど高い優先度となる
    - スタックの先頭番地はNULLを指定するとカーネルが自動的にスタック領域を確保する

タスクを非特権モードで動かすための記述

app.cfg

```
DOMAIN(TDOM_APP) {
    CRE_TSK(TEST_TASK, { TA_ACT , 0, test_task,
                8, 1024, NULL });
}
```

タスクをカーネルに登録するための記述

タスク生成に関係するマクロ

target/ev3_gcc/ev3.h

```
/* タスクの優先度 */
#define TPRI_INIT_TASK      (TMIN_TPRI)
#define TPRI_USBMSC         (TMIN_TPRI + 1)
#define TPRI_BLUETOOTH_QOS  (TMIN_TPRI + 1)
#define TPRI_BLUETOOTH_HIGH (TMIN_TPRI + 2)
#define TPRI_APP_TERM_TASK  (TMIN_TPRI + 3)
#define TPRI_EV3_LCD_TASK   (TMIN_TPRI + 3)
#define TPRI_EV3_MONITOR    (TMIN_TPRI + 4)
#define TPRI_PLATFORM_BUSY  (TMIN_TPRI + 5)
#define TPRI_APP_INIT_TASK  (TMIN_TPRI + 6)
#define TPRI_EV3_CYC        (TMIN_TPRI + 7)
#define TMIN_APP_TPRI       (TMIN_TPRI + 8)
#define TPRI_BLUETOOTH_LOW  (TMAX_TPRI)

/* タスクのスタックサイズ */
#define STACK_SIZE  4096
```

EV3RTのアプリで指定できる最高優先度

デフォルトのスタックサイズ

タスクを追加する(2/2)

- Cファイルにタスクとして動作する関数を追加

app.c

```
void
test_task(intptr_t exinf)
{
    …
    ext_tsk();
}
```

cfgファイルに記述したタスクの関数名と同じ名前

cfgファイルに記述したタスクに渡す引数がexinfに渡される

タスクを終了するためのAPI呼出し
※そのままリターンしてもext_tskにジャンプする仕組みになっているが，明示的にext_tskを呼び出すほうが正式

app.h

```
#ifndef TOPPERS_MACRO_ONLY
…
extern void     test_task(intptr_t exinf);
…
#endif /* TOPPERS_MACRO_ONLY */
```

プロトタイプ宣言の追加

EV3RTの提供するEV3用API

- APIを提供するモジュール一覧
  - サーボモータ
  - 各種センサ
    - 超音波，ジャイロ，タッチ，カラー
  - LCD
  - ファイルシステム
    - シリアル送受信機能を含む
  - EV3本体機能
    - バッテリ，ボタン，LED，スピーカ
  - 拡張RTOS機能
    - ユーザ空間での周期ハンドラ

APIリファレンス

- EV3用C言語APIリファレンス
  - パッケージのEV3RT_C_API_Referenceフォルダ
    - または http://www.toppers.jp/ev3pf/EV3RT_C_API_Reference/index.html
  - こちらのリファレンスがC++APIに対するベース
- EV3用C++言語APIリファレンス
  - パッケージのEV3RT_CPP_API_Referenceフォルダ
    - または http://www.toppers.jp/ev3pf/EV3RT_CXX_API_Reference/index.html
  - モータやセンサをクラス化

モータ制御APIの例(1/3)

- 関数の引数で使用する列挙型

sdk/common/ev3api/src/ev3api_motor.h

```
typedef enum {
   EV3_PORT_A = 0,  // ポートA
   EV3_PORT_B = 1,  // ポートB
   EV3_PORT_C = 2,  // ポートC
   EV3_PORT_D = 3,  // ポートD
   TNUM_MOTOR_PORT = 4  // モータポートの数
} motor_port_t;  // モータポートを表す番号
typedef enum {
   NONE_MOTOR = 0,       // モータ未接続
   MEDIUM_MOTOR,         // サーボモータM
   LARGE_MOTOR,          // サーボモータL
   UNREGULATED_MOTOR,    // 未調整モータ
   TNUM_MOTOR_TYPE       // モータタイプの数
} motor_type_t;  // サポートするモータタイプ
```

モータ制御APIの例(2/3)

- クラスMotor (namespace ev3api)
- Motorの公開関数
  - Motor (コンストラクタ)
  - ~Motor (デストラクタ)
  - reset
  - getCount
  - setCount
  - setPWM
  - setBrake
  - stop

APIリファレンスを確認してみましょう！
http://www.toppers.jp/ev3pf/EV3RT_CXX_API_Reference/classev3api_1_1_motor.html

モータ制御APIの例(3/3)

- 使用例

sdk/workspace/ev3way-cpp/app.cpp

```
/* モータオブジェクトの生成 */
tailMotor = new Motor(PORT_A);
/* モータ停止，回転角度を0初期化 */
tailMotor->reset();
/* モータの回転角度を取得 */
float pwm = (float)(angle - tailMotor->getCount()) * P_GAIN;
/* モータ回転速度(PWM)を設定 */
tailMotor->setPWM(pwm);
```

- コンストラクタの引数は，モータを繋いでいるポートに対応するmotor_port_t，ブレーキモードのtrue/false（デフォルトはtrue），モータに対応するmotor_type_t（デフォルトはLARGE_MOTOR）
- getCountはreset直後はreset時からの回転角度を返すが，setCountを呼び出すことで，回転角度のオフセットを設定可能

センサ取得APIの例(1/3)

- 関数の引数で使用する列挙型

sdk/common/ev3api/arc/ev3api_sensor.h

```
typedef enum {
    EV3_PORT_1 = 0,          // ポート1
    EV3_PORT_2 = 1,          // ポート2
    EV3_PORT_3 = 2,          // ポート3
    EV3_PORT_4 = 3,          // ポート4
    TNUM_SENSOR_PORT = 4     // センサポートの数
} sensor_port_t; //センサポートを表す番号
typedef enum {
    NONE_SENSOR = 0,         // センサ未接続
    ULTRASONIC_SENSOR,       // 超音波センサ
    GYRO_SENSOR,             // ジャイロセンサ
    TOUCH_SENSOR,            // タッチセンサ
    COLOR_SENSOR,            // カラーセンサ
    TNUM_SENSOR_TYPE         // センサタイプの数
} sensor_type_t;  // サポートするセンサタイプ
```

センサ取得APIの例(2/3)

- クラスGyroSensor (namespace ev3api)
  - クラスSensorの子クラス
- GyroSensorの公開関数
  - GyroSensor (コンストラクタ)
  - setOffset
  - reset
  - getAnglerVelocity
  - getAngle

センサ取得APIの例(3/3)

- 使用例

sdk/workspace/ev3way-cpp/app.cpp

```
/* ジャイロセンサオブジェクトの生成 */
gyroSensor  = new GyroSensor(PORT_4);
/* ジャイロセンサ初期化 */
gyroSensor->reset();
/* ジャイロセンサのオフセット値を設定 */
gyroSensor->setOffset(OFFSET);
/* ジャイロセンサ値の取得 */
gyro = gyroSensor->getAnglerVelocity();
```

- getAnglerVelocityは，ジャイロセンサの現在の値（角速度）と，setOffsetで指定したオフセット値との差分を返す

ファイルシステムAPIの例(1/2)

- シリアル入出力用の列挙型

sdk/common/ev3api/src/ev3api_fs.h

```
typedef enum {
  EV3_SERIAL_DEFAULT   // EV3RTコンソール用ポート
  EV3_SERIAL_UART      // UARTポート（センサポート1）
  EV3_SERIAL_BT        // Bluetooth仮想シリアルポート
} serial_port_t;  //シリアルポートを表す番号
```

- シリアル入出力用の関数
  - ev3_bluetooth_is_connected
  - ev3_serial_open_file
  - 入出力自体はnewlib APIを使用できる
    - fprintf, fputc, ...
- 他にファイルストリーム入力用のAPIもある

ファイルシステムAPIの例(2/2)

- 使用例

app.c

```
/* Bluetooth(SPP)経由でのシリアルI/Oのopen */
bt = ev3_serial_open_file(EV3_SERIAL_BT)
/* ホストPCとBluetooth接続が確立されるまで待つ */
while (!ev3_bluetooth_is_connected())
tslp_tsk(100);
/* Bluetooth経由でのシリアル受信(1文字) */
uint8_t c = fgetc(bt);
/* Bluetooth経由でのシリアル出力 */
fprintf(bt, "main task started.¥n");
```

拡張RTOS機能APIの例(1/2)

- ユーザ空間の周期ハンドラ生成用静的API
  - EV3_CRE_CYC(周期ハンドラID, { 属性, 引数, 周期ハンドラの関数名, 周期[ms], 初期位相[ms] });
  - 各パラメータの意味はCRE_CYCと同じ
- ユーザ空間の周期ハンドラ制御関数
  - ev3_sta_cyc
    - 周期ハンドラの動作開始
  - ev3_stp_cyc
    - 周期ハンドラの動作停止

上記，周期制御関数の実行コンテキストは，**タスクコンテキスト**になることに注意（RTOSの本来の仕様と異なるので，**拡張RTOS機能**という位置付け）。興味のある方は，ngki-spec-171.pdfのp.242 周期ハンドラの仕様，p.249 sta_cyc の仕様等を確認してみましょう。

拡張RTOS機能APIの例(2/2)

- 使用例

app.cfg

```
DOMAIN(TDOM_APP) {
    EV3_CRE_CYC(TEST_EV3_CYC,    { TA_NULL, 0,
test_ev3_cychdr, 250, 0 });
}
```

app.c

```
void main_task(intptr_t exinf){
    /* 周期ハンドラの開始 */
    ev3_sta_cyc(TEST_EV3_CYC);
}
void test_ev3_cychdr(intptr_t exinf) {
        ...
}
```

250ms周期で
起動するようになる

FAQ & Tips

FAQ

- Q：アプリケーションのソースコードを複数のサブディレクトリに分けて管理するにはどうすれば良いでしょうか？
- A：ev3rt/hrp2/sdk/common/Makefile.prj.common を以下のように修正します。

a) すべてのサブディレクトリを探索対象に入れる場合

```
APPL_DIR += $(foreach dir,$(shell find $(APPLDIR) -type d),$(dir))
```

b) 対象を個別に指定する場合（ここではtestを追加）

```
APPL_DIR += $(foreach dir,$(shell find $(APPLDIR) -type d -name src),$(dir))
APPL_DIR += $(foreach dir,$(shell find $(APPLDIR) -type d -name test),$(dir)) ←これをディレクトリごとに追加
…
```

Tips：コマンドによる１発アプリ更新

- Bluetooth PAN(Personal Area Network)による，遠隔ソフトウェア更新（uImage, アプリ）
  - EV3とPCをBluetoothで接続
  - アプリローダのメニューで「Bluetooth PAN」を選択
- アプリの更新

```
$ make upload [ip=<ev3のIPアドレス>] [from=<手元のファイル名>] [to=<アップロード後のファイル名>]
```

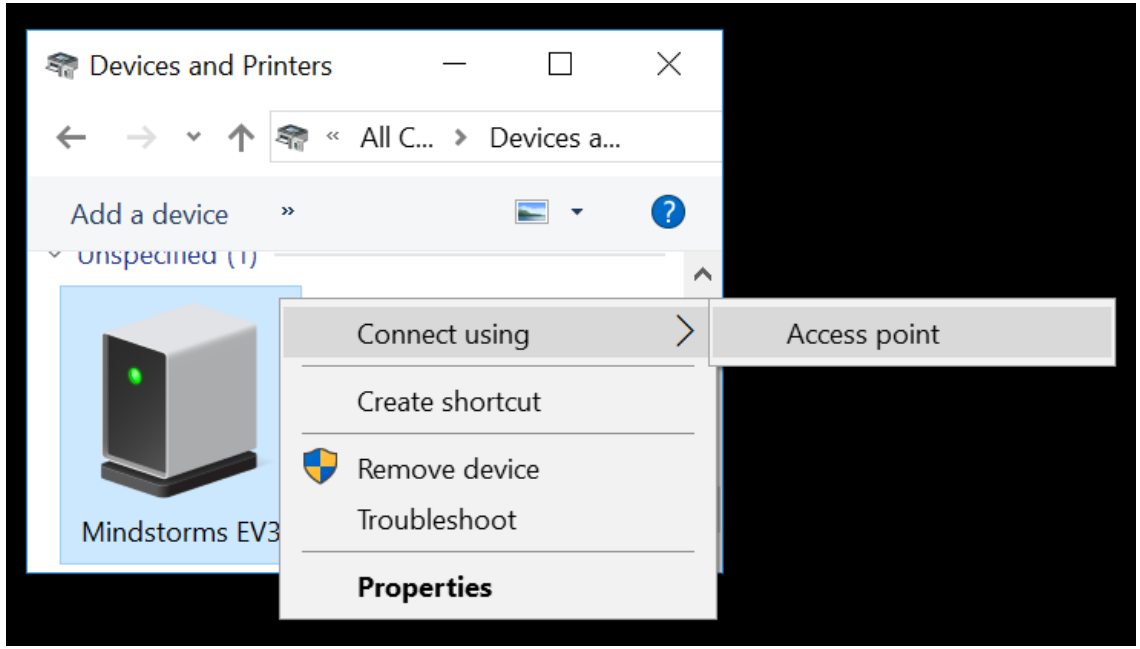
  - デフォルトの設定
    - BT PAN IP: 10.0.10.1
    - 手元のファイル名：app
    - アップロード後のファイル名：app
- イメージファイル（uImage）の更新

```
$ make uploadimg [ip=<ev3のIPアドレス>] [from=<手元のイメージファイル>
```

Ex. hrp2/sdk/workspace で実行する場合
$ make uploadimg from=../../base-workspace/uImage

Tips ：コマンドによる１発アプリ更新

- Bluetoothで接続した後、PANサービスの有効化が必要
  - 例：Windows 10の場合
    - Control Panel -> Devices and Printers -> EV3 -> Connect using



Tips ：コマンドによる１発アプリ更新

- Uploadが成功した場合

~/ev3rt-beta7-1-release/hrp2/sdk/workspace

```
liyixiao@NCES-VAIO-LYX ~/ev3rt-beta7-1-release/hrp2/sdk/workspace
$ make upload
= Upload user application file
  BT PAN IP: 10.0.10.1
  FILE NAME: (Local) "app" -> (Remote) "app"
= Execute cURL:
= Application has been successfully uploaded.

liyixiao@NCES-VAIO-LYX ~/ev3rt-beta7-1-release/hrp2/sdk/workspace
$
```

Tips：RTOSの活用，タスク分割のタイミング

- １ループがめちゃくちゃ長くなってきた…
  - 見やすく，管理し易くしましょう
    →複数のタスク／関数に分割しましょう
- １ループに色々詰め込んだら，動作が遅い／間に合わない処理が出てきた …
  - 重要さの違う処理が混在している場合は，タスクを分け，重要さの低い処理を後回しにしましょう
    →重要度に応じたスケジューリングをOSに任せましょう
  - 動作して欲しいタイミングが異なる処理（100ms秒周期，1秒周期など）が混在している場合は，タスクor周期ハンドラに分け，**タイミング制御をOSに任せましょう**
  - 理論的に間に合わないという場合もあるので，うまくいかない場合はタスク設計を見直しましょう

参考：ビルドの仕組み（make での処理内容）

1. sdk/workspace/Makefileを使ってmake img=test or app=test を実行
2. sdk/workspace/test/Makefile.incをinclude
   - configureのパラメータを設定
3. sdk/OBJをmkdirしてOBJ/に移動
4. OBJ/でconfigureを実行して，Makefileを生成
   - sdk/workspace/testをパスに含めて，2の設定を使用
   - Makefileのテンプレートはsdk/common/Makefile.img, Makefile.app
5. OBJ/で4.で生成したMakefileを使ってmake
6. スタンドアローン版の場合は，objcopyでELF形式のモジュールからバイナリ形式のhrp2を生成したあと，mkimageコマンドでhrp2からuImageを生成
   - ubootがロードするためのファイル形式に変換
7. sdk/workspace/にapp or uImageをコピー
   - appは動的ローディング形式のアプリケーションモジュール