マルチタスクプログラミング

本田　晋也
名古屋大学 大学院情報科学研究科
honda@ertl.jp

最終更新
2016年6月20日

概要

シングルタスクプログラミングの問題を解決する
マルチタスクプログラミングについて学ぶ

- アジェンダ
  - マルチタスクプログラミング環境R2CAのインストール
  - マルチタスクプログラミング

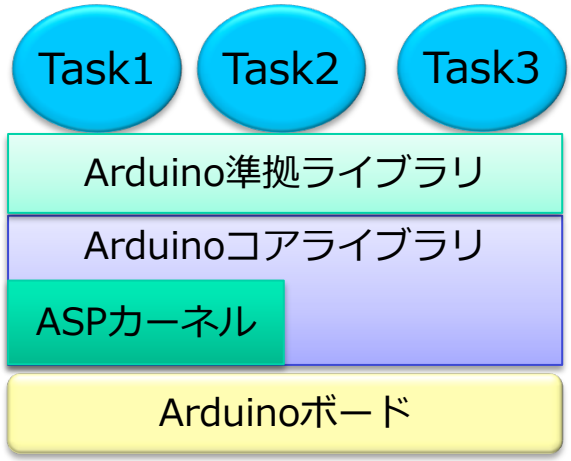
マルチタスクプログラミング環境
R2CAのインストール

TOPPERS/R2CA (RTE/RTOS compatible with Arduino libraries)

ASPカーネルとArduinoライブラリを組み合わせた環境

- マルチタスク環境でArduinoライブラリを使用可能
- GUIベースのデバッガを使用可能

- TOPPERSの問題点の解決
  - 開発環境の導入や使用の敷居が高い
    - Arduino IDEをインストールするだけでビルド可能
    - バッチファイルによるビルドが可能
    - 安価で入手性の良いArduinoボードで実行可能
    - マクロの定義によるタスクの生成
  - ライブラリ・ミドルウェアが少ない
    - 多くのArduinoライブラリが使用可能



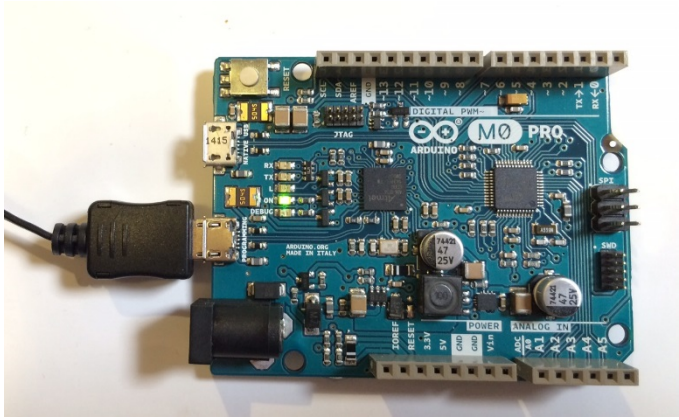
R2CAの使用方法

インストール・ビルド・実行・デバッグ・プログラミングモデルについて説明

- Qiitaにもチュートリアル記事がある(R2CAで検索すると出てくる)
  - 基本的な使い方
    - インストールとサンプルの実行，マルチタスク，優先度・スケジューリング，デバッグ
  - 通信
    - Wifi通信，Wifi通信(マルチタスク)，CAN通信
  - IoT
    - Milkcocoaへの接続，ThingSpeakへの接続
  - Shield
    - Zumo，NCES IoT Base Shield
  - MacOSXでの使用方法
    - TOPPERS/R2CA を MacOSXで動かす

必要な機材

- ホストPC
  - Windows or Mac OS
  - 本チュートリアルではWindowsで説明
  - Linuxでも動作するはず

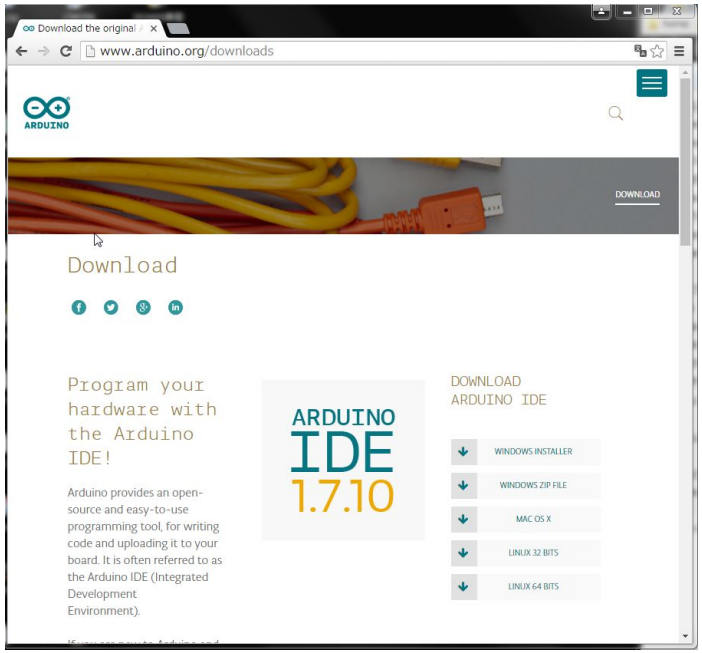


- Arduino M0 Pro
  - 6000円程度
  - 秋月，Amazon, Switch Science，マルツパーツ等で購入可能
  - Cortex-M0+ 48MHz/ROM 256KB/RAM 32KB
  - デバッガ機能あり（EDBG(Atmel’s Embedded Debugger)）
    - デバッガ機能なしのArduino M0 もあるが推奨しない
      - ✓Arduino UNOとの互換性が低いためプログラムの書き換えが必要
  - 一般的なArduinoボード(Arduino UNO)との違い
    - ARM(Cortex-M0)を搭載（UNO等はAVR）
    - IOが3.3V (UNO等は5V)

ツールのインストール

- Arduino IDE
  - Arduino.org(http://www.arduino.org/downloads）からダウンロード
    - 動作確認済みバージョン：1.7.10
  - !!Arduino.ccではないため注意!!
  - インストーラによりインストール
    - GCC/Make/GDB/OpenOCDがインストールされる
- ターミナルエミュレータ
  - Teraterm等をインストール

- 以下のツールはオプション
  - Atmal Studio
    - GUIによるデバッグ
    - 7.0で確認済み



R2CAパッケージのダウンロード

- TOPPERSのContributed Softwareから公開
  - trac
    - http://dev.toppers.jp/trac_user/contrib/wiki/rtos_arduino
    - http://dev.toppers.jp/trac_user/contrib/browser/rtos_arduino/trunk
  - svn
    - http://dev.toppers.jp/svn_user/contrib/rtos_arduino/trunk

- ダウンロード方法
  - ZIPファイルをダウンロード
    - http://dev.toppers.jp/trac_user/contrib/browser/rtos_arduino/trunk?rev=245&format=zip
  - SVNクライアントをインストールしてチェックアウト
    - TortoiseSVN, CygwinのSVNクライアント
    - パッケージの更新頻度が高いのでこの方法を推奨

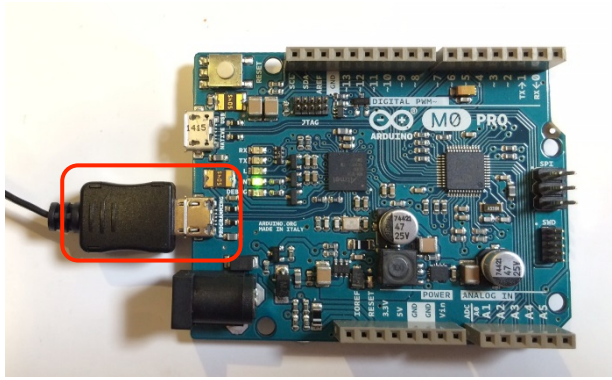
フォルダ構成

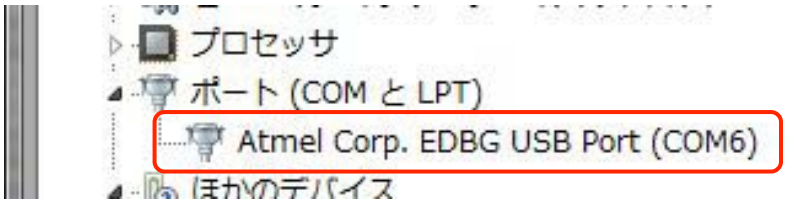
- パッケージのフォルダ一覧
  - ./arduino_lib：Arduinoライブラリ
    - hardware ：コアライブラリ
    - libraries ：Arduino準拠ライブラリ
  - ./asp_1.9.2 ：Arduino M0 依存部を含む ASP 1.9.2ソースコード
  - ./examples ：サンプルプロジェクト
  - ./lib ：R2CAライブラリ

- Arduinoライブラリについて
  - Arduino IDEに付属のライブラリがベース
  - バグフィックスやマルチタスク対応のための排他制御を入れている
  - ライセンスはGPLやMIT
  - librariesにはArduino IDE付属以外のライブラリも含まれている
    - ESP8266_Arudino_AT, Milkcocoa_Arduino_SDK, ZumoShield, NcesCan

セットアップ

- Arduino IDE のインストールパスの設定
  - 以下のファイルの”C:¥Program Files (x86)¥Arduino”の箇所を書き換える
    - example/do_path.bat
      - ✓SET ARDUINO_DIR=C:¥Program Files (x86)¥Arduino
    - asp_1.9.1/target/arduino_m0_gcc/Makefile.target
      - ✓ARDUINO_BASE_DIR_WIN = C:¥Program Files (x86)¥Arduino
- ボードの接続
  - M0の**PROGRAMMINGポート**とPCをUSBケーブルで接続
  - ドライバがインストールされ，COMポートとEDBGが認識される
  - COMポートの番号を確認して，Teraterm等で115200bpsで接続





プログラムの一覧

- フォルダ：¥examples¥MultiTaskText

- シングルタスクプログラム(SingleTask)
  - プログラム1,2,3,4をマクロで選択して実行可能

- プログラム5R（LEDBlink_CLEDBlink）
  - プログラム5をマルチタスクで実現

- プログラム8R（Interrupt）
  - プログラム8をマルチタスクで実現（タスク起動を使用）

- プログラム9（Exclusion）
  - プログラム8にカウンタをクリアする処理を追加

プログラムの実行

シングルタスクプログラムをビルド&実行する

- フォルダ：¥examples¥MultiTaskText¥SingleTask
- ユーザープログラム等
  - Makefile　　：ライブラリやファイルの指定
  - rca_app.h　：ユーザープログラムヘッダーファイル
  - rca_app.cpp：ユーザープログラムプログラムファイル
  - rca_app.cfg ：コンフィギュレーションファイル(静的APIを記述)
  - pitches.h　　：プログラムで使用するヘッダファイル
- バッチファイル
  - do_make.bat ：ビルド
  - do_run.bat　　：ビルド&書き込み&実行
  - do_clean.bat: ファイルのクリーン
  - do_debug.bat：ビルド&書き込み&デバッグ
- AtmelStudio用ファイル
  - rca.atsln，rca.componentinfo.xml，rca.cproj

サンプルの実行：プログラムの選択

- ¥examples¥MultiTaskText¥SingleTask¥rca_app.cpp には，シングルタスクプログラミングのプログラム1,2,3,4のいずれかを選択可能
- 有効にするプログラムのマクロのコメントアウトを外す
  - 同時に有効可能なプログラムは1個まで

- #define LED_BLINK
  - プログラム1：1000m周期でLEDを点滅
- #define CLED_BLINK
  - プログラム2：250ms周期でChange LEDの色を変更
- #define TOUCH_SENSE
  - プログラム3：Touchセンサの検知によるLEDのON/OFF切り替え
- #define LUX_SENSE
  - プログラム4：光センサの値のOLEDへの出力

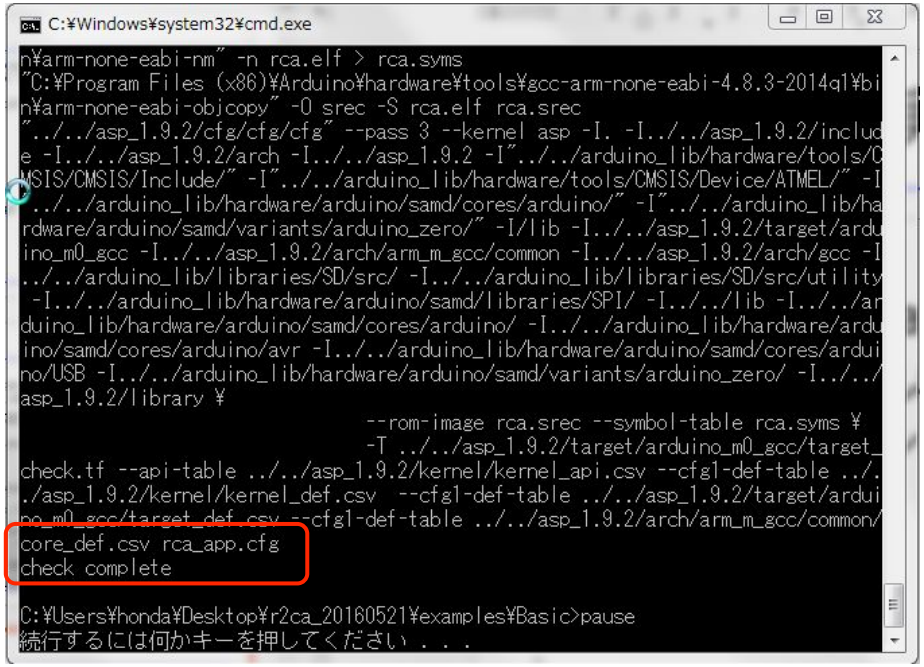
サンプルの実行：プログラムの内容

- R2CA用ヘッダ（rca.h）のインクルード以外はArduinoと同等のプログラムを実行可能
- setup()：起動時に一度だけ実行される関数
- loop()　：繰り返し実行される関数

```
#include "rca.h“

#define LED_PIN 4

void setup() {
  pinMode(LED_PIN, OUTPUT);
}

void loop() {
  digitalWrite(LED_PIN, HIGH);
  delay(1000);
  digitalWrite(LED_PIN, LOW);
  delay(1000);
}
```

サンプルの実行：ビルド

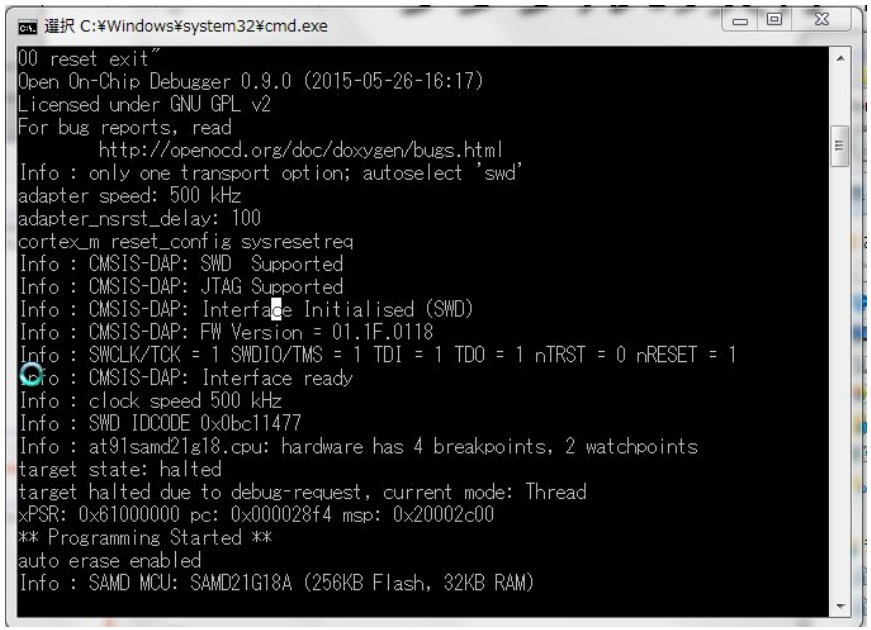
- do_make.bat をダブルクリックするとビルドが開始される
- ASPカーネル&Arduinoライブラリ&ユーザープログラムがビルドされてダウンロードファイルが生成される
- check completeが出ればビルドは成功
  - rca.elfが出来ている



サンプルの実行：実行

- do_run.bat をダブルクリックすると書き込みが行われる
  - OpenOCDによる書き込み
- 書き込み後に実行される

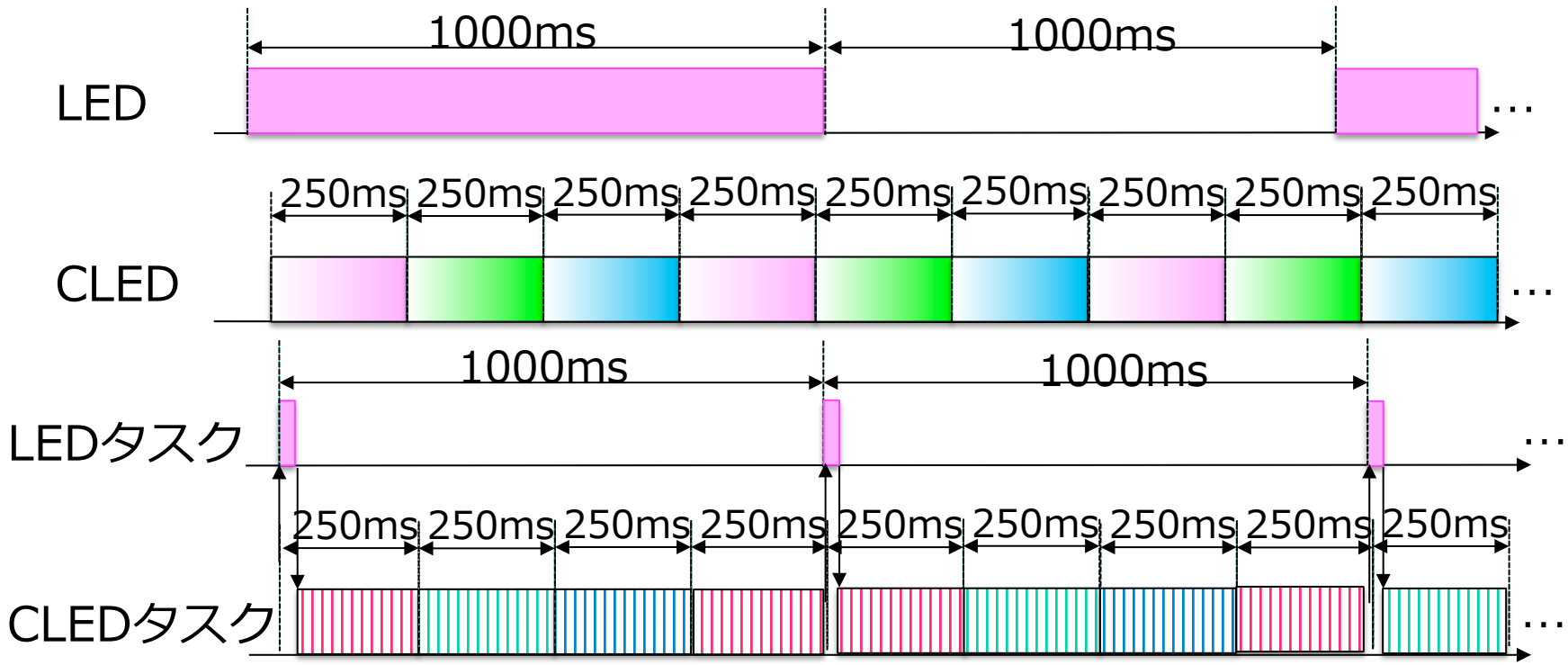
<u>書き込み</u>



マルチタスクプログラミング

プログラム5R（LEDBlink_CLEDBlink）

- プログラム5をマルチタスクで実現
  - 1000m周期でLEDを点滅させる
  - 250m周期でChange LEDの色を変更



プログラム5R：プログラミング

- タスク構成
  - setup()/loop()を実行するタスクをメインタスクと呼ぶ
  - task1_setup()/task1_loop()はタスク1と呼ばれるタスクにより実行される
  - 起動時は両方のタスク共に実行可能状態
  - 優先度は等しい
  - 起動時の優先順位はメインタスクが高い
  - 両タスク共にシングルタスクと同じコードを実行

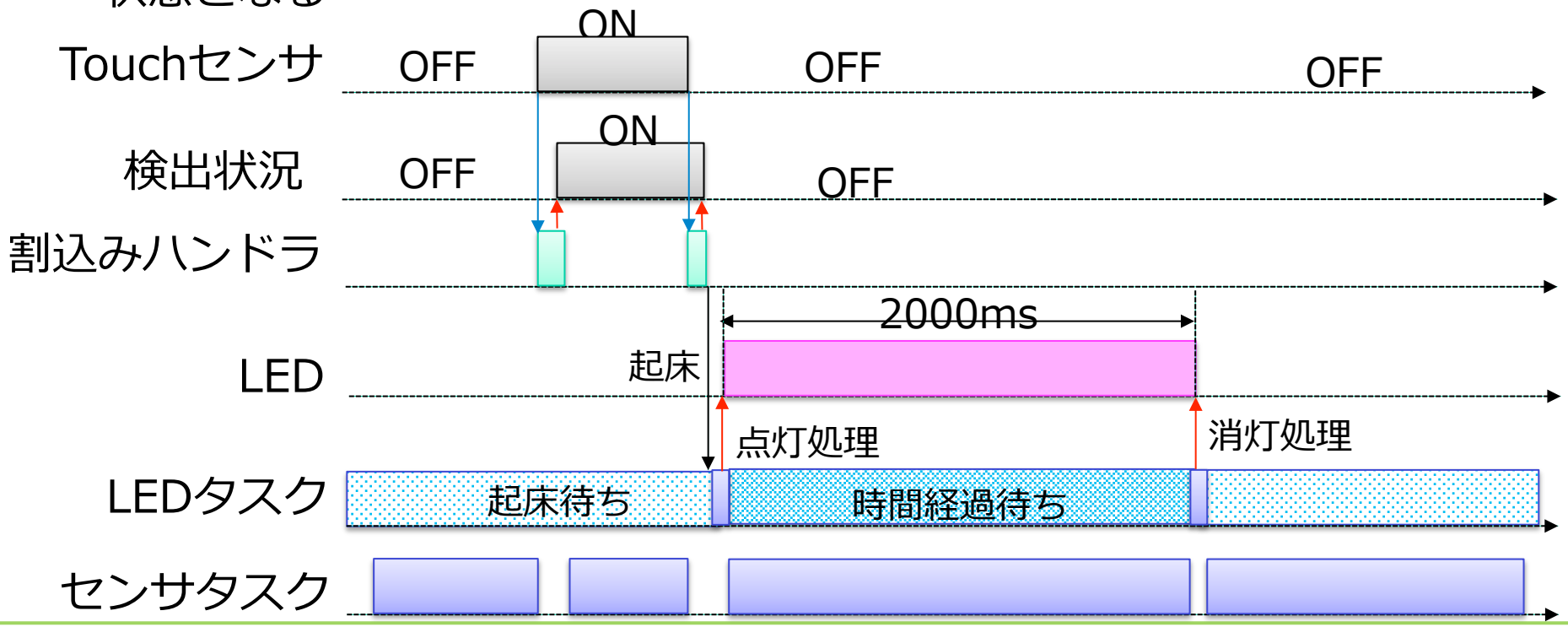
```
#define LED_PIN 4

void setup() {
  pinMode(LED_PIN, OUTPUT);
}

void loop() {
  digitalWrite(LED_PIN, HIGH);
  delay(1000);
  digitalWrite(LED_PIN, LOW);
  delay(1000);
}
```

- タスクの実行
  - メインタスク実行状態になりdelay()により待ち状態となると，タスク1を実行状態として再びdelay()により待ち状態となる
  - その後は待ち状態がタイムアウトしたタスクを起床させ，実行可能状態として処理を継続する

```
#define NUM_LEDS 1
ChainableLED leds(8, 9, NUM_LEDS);
void task1_setup() {
  leds.init();
}
void task1_loop() {
  int i;
  for(i = 0; i < 25; i++){
    leds.setColorRGB(0, i*10, 0, 0);
    delay(10);
  }
   ….
}
```

プログラム8R（Interrupt）

- プログラム8をマルチタスクで実現（タスク起動を使用）
  - LEDタスクは起動後，起床待ちAPIを呼び出して待ち状態へ
  - TouchセンサがOFF-ON-OFFすると割込みハンドラから起床APIを呼び出して起床させ，実行可能状態へ
  - LEDタスクはLEDを点灯させた後，時間経過待ちAPIにより，2秒間の待ち状態となる



プログラム8R（Interrupt）：プログラム

- タスク構成
  - メインタスク：センサタスク
  - タスク1：LEDタスク
- タスク起床待ちAPI：slp_tsk();
- タスク起動API：iwup_tsk()
  - タスク1のID(RCA_TASK1)を指定

```
void setup() {
  Wire.begin();
  …
  attachInterrupt(TOUCH_PIN, CheckTouch, CHANGE);
}
void loop() {
  int lux;
  lux = TSL2561.readVisibleLux();
  SeeedOled.setTextXY(0, 0);
  …
  SeeedOled.putNumber(OLEDCount++);
  SeeedOled.putString("        ");
}
```

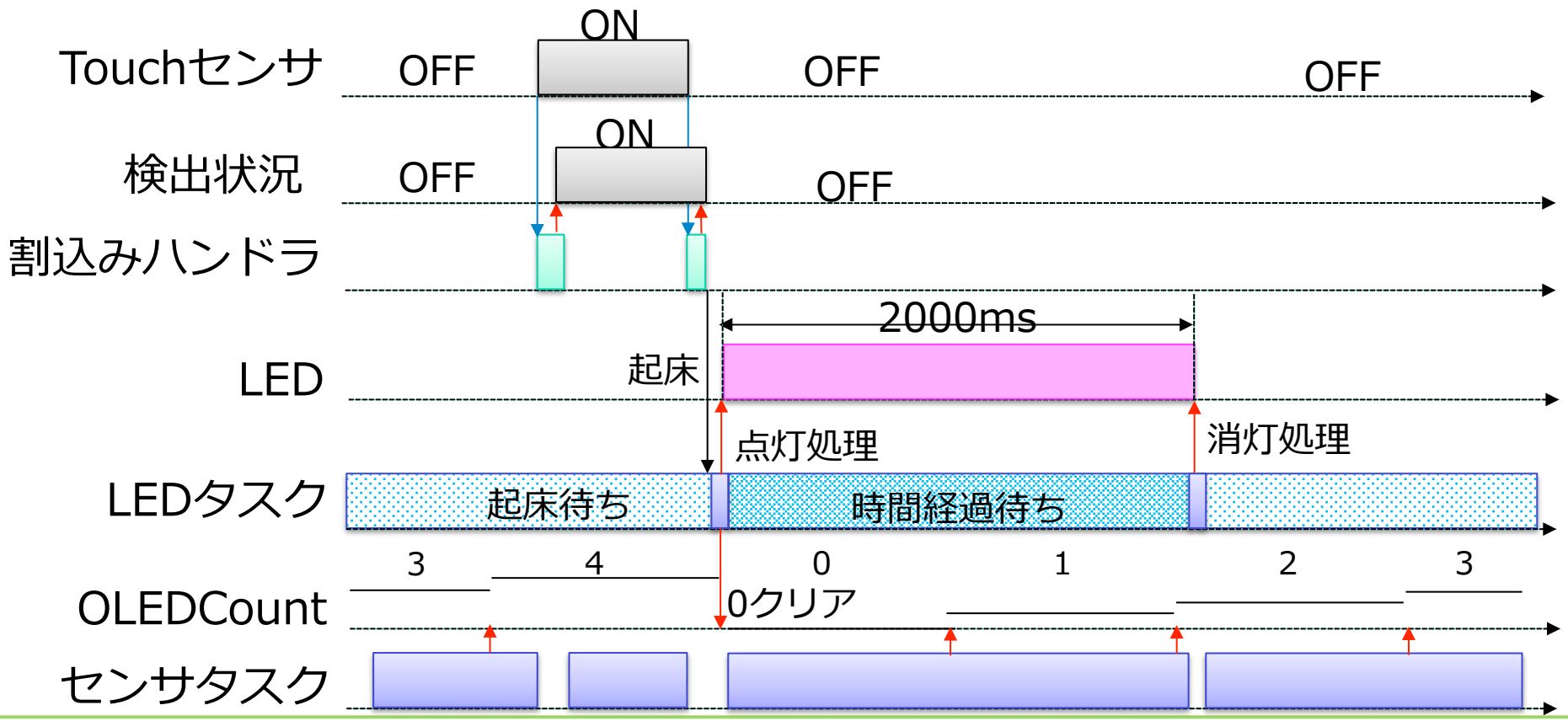
```
void CheckTouch(){
  int TouchValue = digitalRead(TOUCH_PIN);
  if ((PreTouchValue == 0) && (TouchValue == 1)
      && (TouchState = 0)) {
    TouchState = 1;
  }
  if ((PreTouchValue == 1) && (TouchValue == 0)
      && (TouchState = 1)) {
    TouchState = 0;
    iwup_tsk(RCA_TASK1);
  }
  PreTouchValue = TouchValue;
}
```

起床

```
void task1_setup() {
}

void task1_loop() {
  slp_tsk();
  digitalWrite(LED_PIN, HIGH);
  dly_tsk(2000);
  digitalWrite(LED_PIN, LOW);
}
```

プログラム9

- プログラム8に光センサ表示カウンタ(OLEDCount)をクリアする処理を追加
  - センサタスク：OLEDに表示する毎にインクリメント
  - LEDタスク　：割込みハンドラから起動されたら0クリアする



プログラム9：プログラム

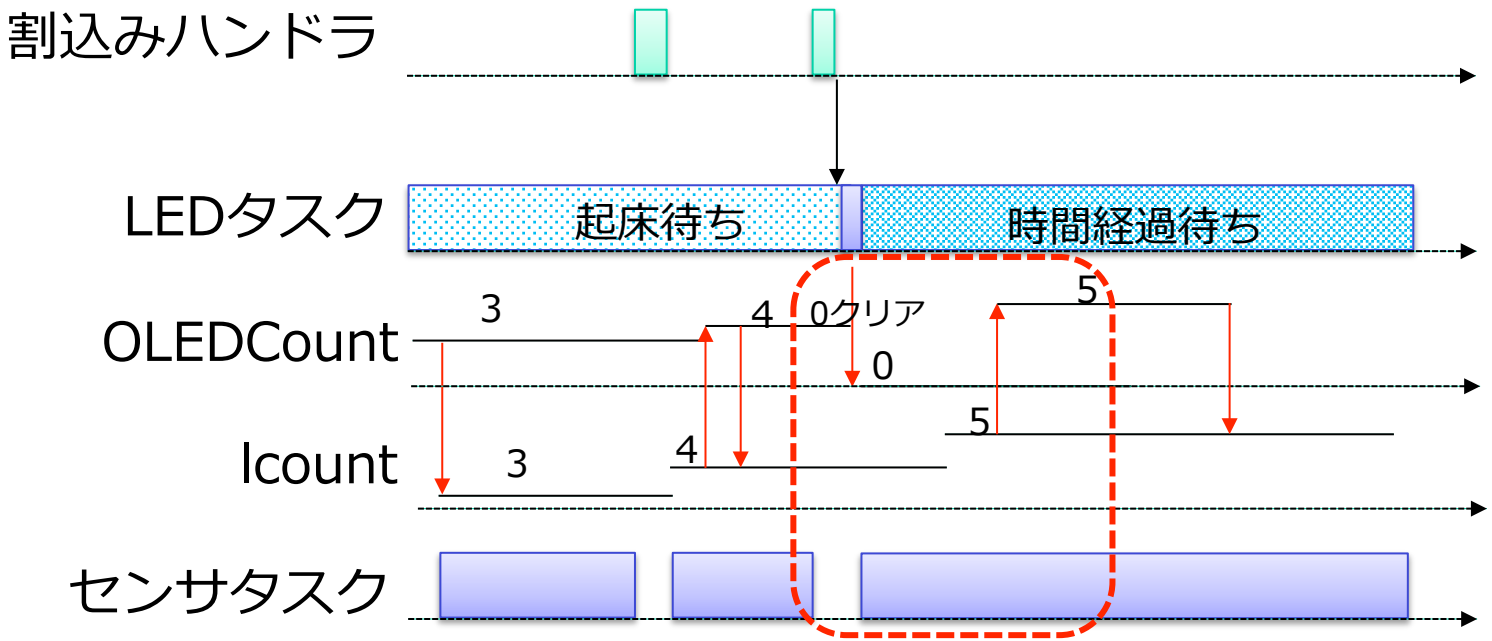
- センサタスク
  - OLCDへの表示前にローカル変数に読み込み，表示後にインクリメントして書き戻す
- LEDタスク
  - 起床した後(slp_tsk()からリターン)OLEDCountを0にする

```
void setup() {
  Wire.begin();
  …
  attachInterrupt(TOUCH_PIN, CheckTouch, CHANGE);
}
void loop() {
  int lux;
  int lcount;
  lcount = OLEDCount;
  lux = TSL2561.readVisibleLux();
  SeeedOled.setTextXY(0, 0);
   …
  SeeedOled.putNumber(lcount++);
  SeeedOled.putString("        ");
  OLEDCount = lcount;
}
```

```
void task1_setup() {
}

void task1_loop() {
   slp_tsk();
   OLEDCount = 0;
   digitalWrite(LED_PIN, HIGH);
   dly_tsk(2000);
   digitalWrite(LED_PIN, LOW);
}
```

プログラム9：問題

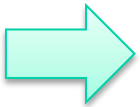
- カウンタがクリアされない場合がある
- センサタスクが値をローカル変数に保持している間にLEDタスクがOLEDCountを更新した場合，その結果がセンサタスクによって上書きされる



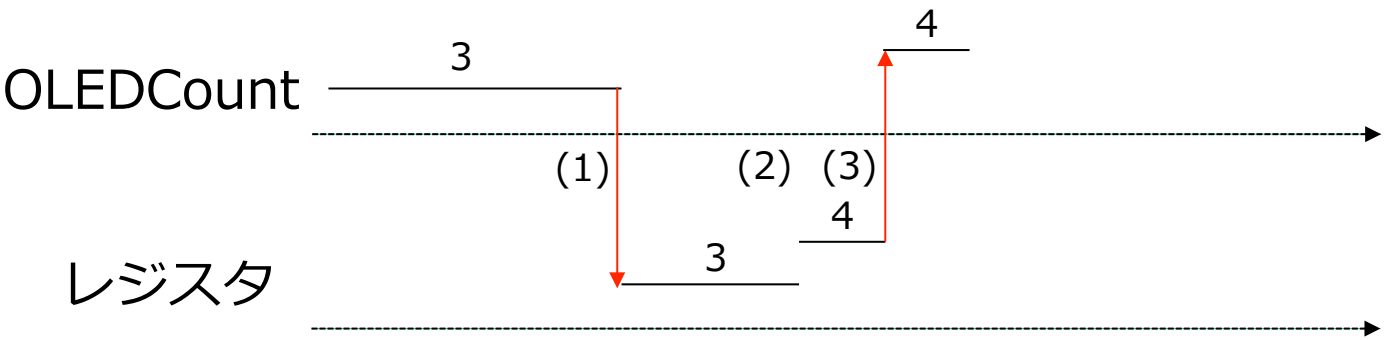
プログラム9：問題の解決

- ローカル変数を使わずC言語1行で記述する
  - 発生頻度は下がるが本質的な解決にならない
  - プロセッサレベルでは複数の処理となるためその間で割り込まれる可能性がある
    - (1)メモリのレジスタへの読みこみ, (2)計算,
      (3)レジスタのメモリへの書き戻し

```
void loop() {
  int lux;
  lux = TSL2561.readVisibleLux();
  SeeedOled.setTextXY(0, 0);
   …
  SeeedOled.putNumber(OLEDCount++);
  SeeedOled.putString("        ");
}
```

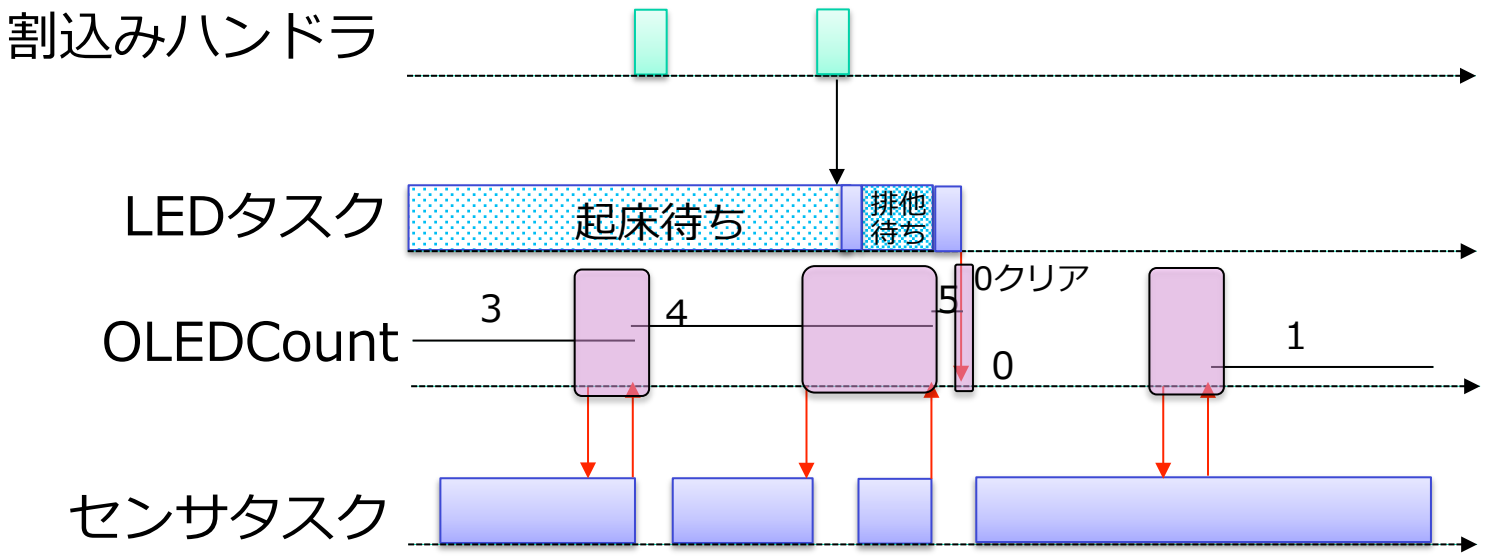


```
(1) ldr r2, [r, #0]
(2) adds r2, #1
(3) str r2, [r3,#0]
```



プログラム9：問題の解決

- 排他制御機構を用いる
  - OLEDCountを更新する場合には排他制御を行い，他の処理がアクセスしないようにする

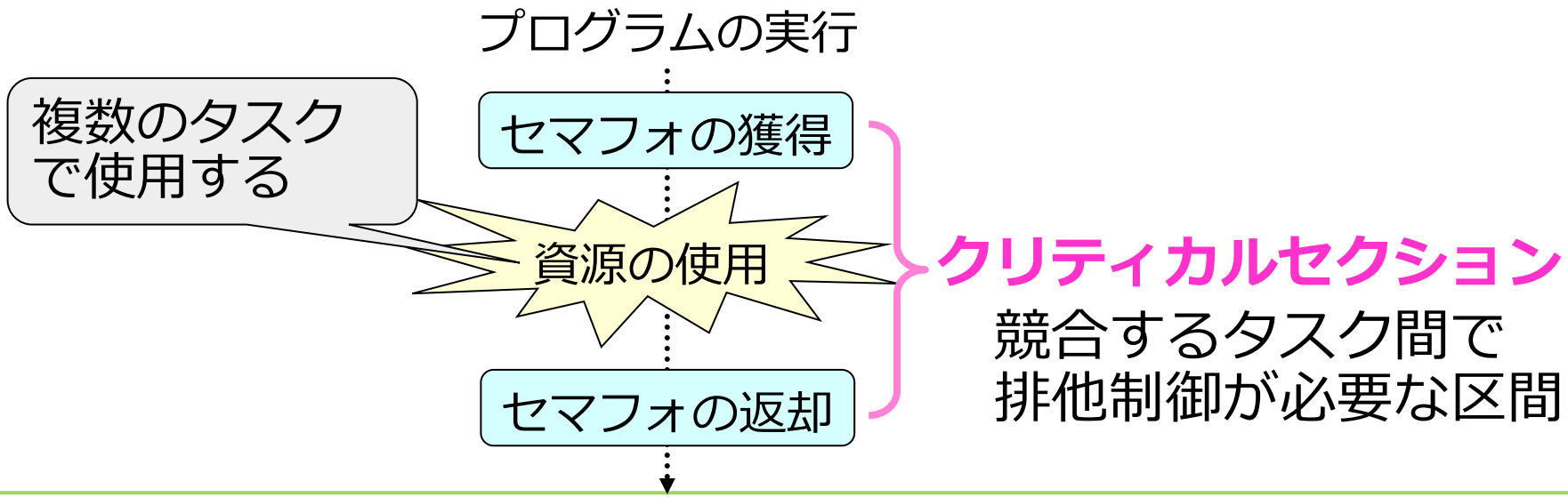


プログラム9：問題の解決

- 使用する排他制御機構を選択する
  - 割込み禁止
    - I2C通信が割込みを使用するため今回は使用出来ない
  - ディスパッチ禁止
    - I2C通信が時間待ちを伴うため今回は使用出来ない
  - 優先度を上げる
    - 使用可能だが動的な優先度変更は可能な限り使わない方がよい
  - セマフォ
    - 今回はこれを使用する

セマフォ（Semaphore）

- 使用されていない資源の有無や数量をセマフォの資源数として管理することにより排他制御を実現
- 資源を使用する前にセマフォ資源を獲得し，資源を使用した後にセマフォ資源を返却する.
- 資源が全て使用されていれば，セマフォ資源獲得の時点で待ち状態になる



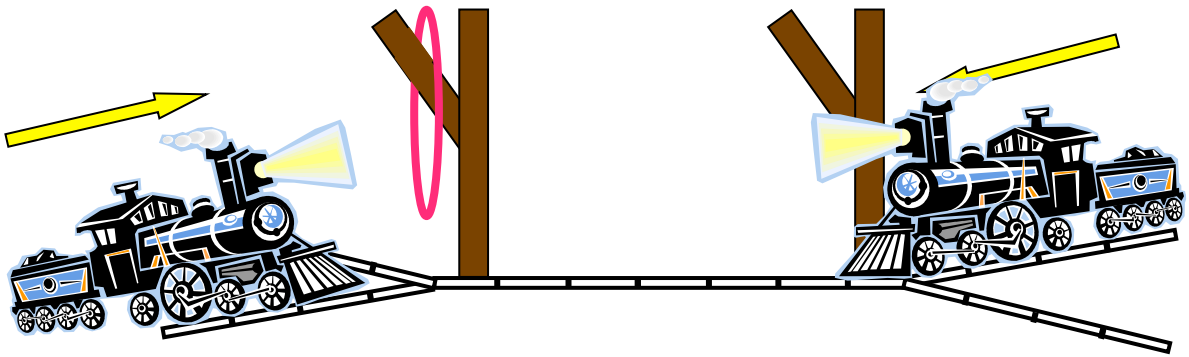
セマフォ（Semaphore）：サービスコール

- サービスコール

CRE_SEM	セマフォの生成
sig_sem, isig_sem	セマフォ資源の返却
wai_sem, pol_sem, twai_sem	セマフォ資源の獲得

- 語源

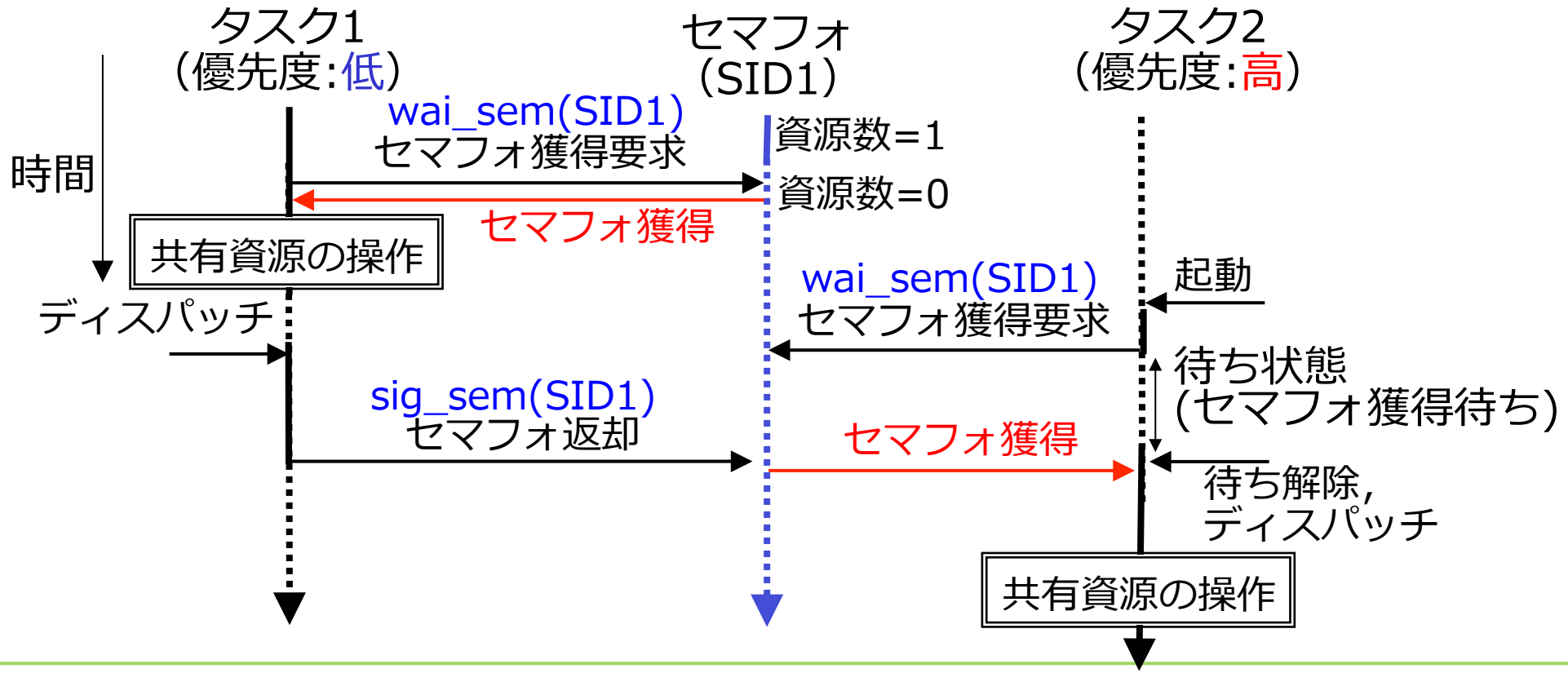
鉄道の単線区間で進入制御に用いていた「輪」.
単線区間に入る場合は，この輪（セマフォ）を取る



輪　　：セマフォ
列車　　：タスク
単線区間：資源

セマフォ（Semaphore）：セマフォ操作

- セマフォ獲得 ： セマフォ資源があれば獲得して実行を継続.
  なければ，資源が解放されるまで待つ
- セマフォ返却 ：セマフォ資源を返却し，その時点で待っているタスク
  があれば，そのタスクに資源を渡し待ちを解除する



プログラム9：セマフォの使用

- OLEDCountのアクセスの前にセマフォを取得
- OLEDCountの更新後セマフォを返却

センサタスク
(ローカル変数版)

```
void loop() {
  int lux;
  int lcount;
  wai_setm(COUNT_SEM);
  lcount = OLEDCount;
  lux = TSL2561.readVisibleLux();
  SeeedOled.setTextXY(0, 0);
   …
  SeeedOled.putNumber(lcount++);
  SeeedOled.putString("        ");
  OLEDCount = lcount;
  sig_setm(COUNT_SEM);
}
```

センサタスク
(グローバル変数版)

```
void loop() {
  int lux;
  lux = TSL2561.readVisibleLux();
  SeeedOled.setTextXY(0, 0);
   …
  wai_setm(COUNT_SEM);
  SeeedOled.putNumber(OLEDCount++);
  sig_setm(COUNT_SEM);
  SeeedOled.putString("        ");
}
```

LEDタスク)

```
void task1_loop() {
    slp_tsk();
    wai_setm(COUNT_SEM);
    OLEDCount = 0;
    sig_setm(COUNT_SEM);
    digitalWrite(LED_PIN, HIGH);
    dly_tsk(2000);
    digitalWrite(LED_PIN, LOW);
}
```

- セマフォの宣言(rca_app.cfg)

```
CRE_SEM(COUNT_SEM, { TA_TPRI, 1, 1 });
```

まとめ

まとめ

- シングルタスクプログラミングの問題を解決する
  マルチタスクプログラミングについて学ぶ

- 独立した処理の並列実行
  - 時間的な処理順序はOSが決定して実行
  - 処理内容を独立させることが出来る

- 割込みからのタスクの起動
  - 割込み発生後に待ち状態を伴う処理を実行可能

- 排他制御
  - 共有する変数の更新で不整合の発生を抑制
  - 排他の状況により使用する機構を選択する