# スケジューリング・シミュレータ schesim チュートリアル

名古屋大学 大学院情報科学研究科
松原豊（yutaka@ertl.jp）
佐野泰正（yasu@ertl.jp）
URL: http://www.schesim.org
最終更新日：2013年11月29日

# はじめに

## 免責

- 本ソフトウェアは，無保証で提供されているものです．
- 著作権者は，本ソフトウェアに関して，特定の使用目的に対する適合性も含めて，いかなる保証も行いません．
- また，本ソフトウェアの利用により直接的または間接的に生じたいかなる損害に関しても，その責任を負いません．

## お問い合わせ

- 本シミュレータをより良いものにするためのご意見，ご要望等を歓迎します．
- 本シミュレータに関する質問やバグレポート，ご意見，ご要望等は，開発者用メールアドレス（schesim@ertl.jp）にお送りください．

# 目次

# アプリケーションの作成方法

# アプリケーション作成の流れ

1. アプリケーション情報ファイルを作成する
タスク処理記述ファイルを以下の機能を使って作成する
   - タスク処理記述ライブラリ
     - exc
     - act_tsk
     - wai_sem/sig_sem
     - GetResource/ReleaseResource
     - SetEvent/WaitEvent/ClearEvent
   - 動作モード
2. シナリオ記述ファイルを作成する（オプション）
3. フック関数（PreTaskHook/PostTaskHook）を使用する（オプション）

# タスク処理の記述方法

アプリケーションを構成するタスクの処理内容を，rubyの関数として記述したもの

- シミュレータのファイルの一部として実行される

```
class TASK
 # タスクIDが10のタスク処理記述
    def task10
        wai_sem(@@sem1)
        exc(1)
        sig_sem(@@sem1)
        exc(1)
    end

 # タスクIDが20のタスク処理記述
    def task20
        exc(1)
        wai_sem(@@sem1)
        exc(2)
        sig_sem(@@sem1)
        exc(1)
    end
end
```

# タスク処理ライブラリ（プロセッサ時間の消費）

## exc：指定した単位時間分だけ実行時間を消費

- 実行時間を予測／測定した環境とは異なる動作プロセッサでの動作をシミュレーションする場合には，プロセッサの処理性能の差を考慮した値を指定する
  - 例えば，２倍の速度のプロセッサでは処理時間を半分にした値を指定する

```
def task1
    exc(10)
end
```
← 10単位時間消費する

```
@@cpu_ratio = 2.0
def task1
    exc(10 / @@cpu_ratio)
end
```
← 2倍の速度のプロセッサでの動作を想定

←実行時間が半分（5単位時間）になる

# タスク処理ライブラリ（タスク起動）

## act_tsk：指定したIDをもつタスクを起動

- 起動するタスクのID番号と相対起動時刻を指定する
  - 指定したIDを持つタスクが存在しない場合は，実行時にエラーが発生する
  - 相対起動時刻を指定すると，act_tskを呼び出した時刻から相対時刻経過後に，指定したタスクが起動される.
  - 相対起動時刻の指定を省略すると，即座にタスクを起動する.

指定したタスクを即座に起動する例

```
def task1
  act_tsk(5)
  …
end
```

← task5を起動する

相対時刻を指定してタスクを起動する例

```
def task1
  act_tsk(5, 10)
  …
end
```

← 10単位時間後に task5を起動する

# タスク処理ライブラリ（セマフォ，リソース）

## wai_sem/sig_sem：セマフォの確保／解放

```
@@sem1 = SEMAPHORE.new(1)

def task1
        wai_sem(@@sem1)
        exc(1)
        sig_sem(@@sem1)
end
```

← 資源数１のセマフォ@@sem1を生成

← @@sem1を確保する

← @@sem1を解放する

## GetResource/ReleaseResource:リソース獲得／解放

- OSEK OS仕様のリソース機能と同等

```
@@res1 = RESOURCE.new(1)

def task1
        GetResource(@@res1)
        exc(1)
        ReleaseResource(@@res1)
end
```

← 上限優先度1のリソース@@res1を生成する

← @@res1を獲得する

← @@res1を解放する

# タスク処理ライブラリ（イベント）

## WaitEvent/SetEvent/ClearEvent

- OSEK OS仕様で規定されているイベント機能を実現する
  - 現状は，簡易の実装で，一つのフラグで１ビットのフラグを管理している
  - 今後，必要性があれば，複数ビットを管理できるよう拡張する

```
@@evt1 = EVENT_FLAG.new()  ← イベントを生成する
def task1
        while(1)
            WaitEvent(@@evt1)   ← @@evt1がセットされるのを待つ
            ClearEvent(@@evt1)  ← @@evt1をクリア
            exc(1)
        end
end
```

# 動作モード

動作モードを共有変数で管理し，共有変数の値に応じて，各タスクの処理内容を変更することで実現できる

```
@@mode = MODE1 ← 動作モード用変数
void task () {
        case $mode
        when MODE1
            # 動作モード1の処理
            exc(5);  ← 5単位時間実行する
        when MODE2
            # 動作モード2の処理
            exc(5);
            wai_sem(SEM1);  ← SEM1の獲得
            exc(5);
            sig_sem(SEM1);   ← SEM1の返却
}
```

# シナリオ入力機能

## シナリオ

- イベントと，イベントの発生時刻を記述したもの
- フォーマット
  - 時刻：コアID：イベント：引数

## 現在サポートしているイベント

- タスクの起動（act_tsk）
- 動作モード（変数）の変更（chg_mod）

## シナリオ記述ファイルの例

```
1:2:act_tsk:4 ← 時刻1でコア2のタスク4を起動する
10:2:act_tsk:4
# mode2を2に変更する ← #で始まる行はコメント
12:2:chg_mod:@@mode2:2 ← 時刻12でコア2のモード（変数）
                          @@mode2に2を代入する
15:2:act_tsk:4
20:2:chg_mod:@@mode2:1
26:2:act_tsk:4
```

# フック関数（PreTaskHook/PostTaskHook）を使用する

- OSEK OS仕様で規定されている PreTaskHookとPostTaskHookを使用できる
- デフォルトのフック関数の処理内容は, include/hook.rbで定義されている

```
class TASK
    def pretask_hook
        print_log_pretaskhook_start    ←PreTaskHookフックの動作開始を示すログ出力
        exc(0.5)                       ← タスク切り替えのオーバヘッドを指定
        print_log_pretaskhook_finished ←PreTaskHookフックの動作開始を示すログ出力
    end

    def posttask_hook
        print_log_posttaskhook_start    ←PostTaskHookフックの動作開始を示すログ出力
        exc(0.5)                        ← タスク切り替えのオーバヘッドを指定
        print_log_posttaskhook_finished ←PostTaskHookフックの動作開始を示すログ出力
    end
end
```

# サンプルプログラムを動かしてみる

# タスク起動とリソースのサンプルアプリケーション

| アプリケーションID | タスクID | 優先度 | 周期 | WCET | 起動属性 |
|---|---|---|---|---|---|
| 1 | 1 | 5 | 4 | 2 | 周期起動 |
| 1 | 2 | 6 | 10 | 4 | 周期起動 |
| 1 | 3 | 7 | 20 | 1 | 周期起動 |
| 2 | 4 | 5 | 4 | 2 | 周期起動 |
| 2 | 5 | 6 | 10 | 2 | - |
| 2 | 6 | 7 | 20 | 1 | 周期起動 |

注意点

- タスクは固定優先度ベーススケジューリングでスケジューリングされる
- この例では，デッドラインは周期と同じにしているが，シミュレータでは別々に指定可能である

# タスク処理記述

## アプリケーション１

```
@@res1 = RESOURCE.new(1)
# タスクIDが1のタスク処理記述
def task1
	GetResource(@@res1)
	exc(1)
	ReleaseResource(@@res1)
	exc(1)
end
# タスクIDが2のタスク処理記述
def task2
	exc(1)
	GetResource(@@res1)
	exc(2)
	ReleaseResource(@@res1)
	exc(1)
end
# タスクIDが3のタスク処理記述
def task3
	exc(1)
end
```

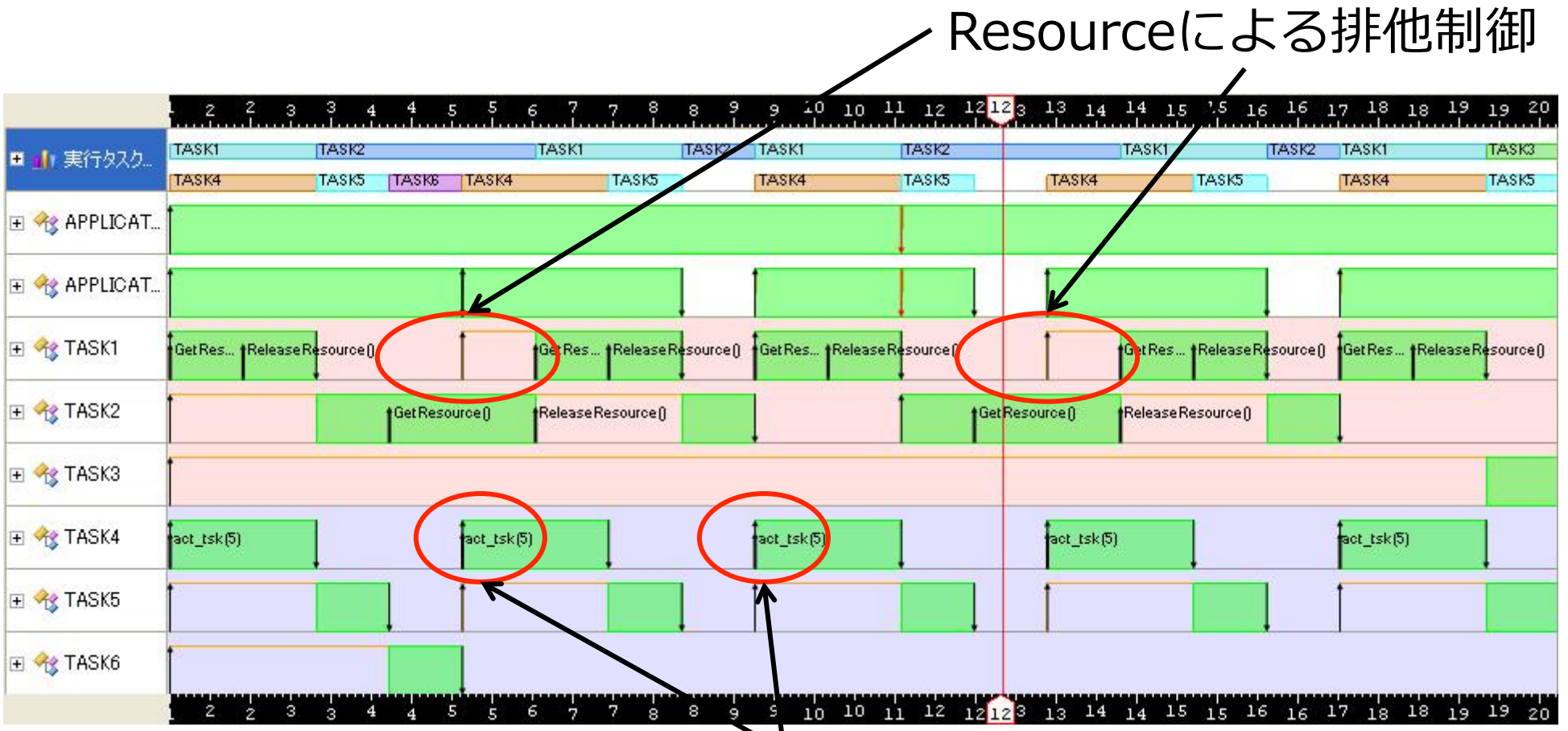
## アプリケーション2

```
# タスクIDが4のタスク処理記述
def task4
	act_tsk(5)
	exc(2)
end

# タスクIDが5のタスク処理記述
def task5
	exc(1)
end

# タスクIDが6のタスク処理記述
def task6
	exc(1)
end
```

# サンプルアプリケーションを動かす

1. タスク起動とリソース機能のサンプルアプリケーションresouceをシミュレータで動かす
   - シミュレーション結果を格納するディレクトリの作成
     - mkdir obj
   - シミュレーションの実行
     - ./schesim.rb -t ./sample/resource.json -r ./obj/resource.res -d sample/resource.rb -e 20 > obj/resource.log
2. TLVでの表示
   - objディレクトリ以下に生成されたresouce.logとresource.resをTLVに入力してログを可視化する
     - 詳細はTLVのドキュメントを参照のこと

# 可視化結果の例


Resourceによる排他制御
実行タスク...
APPLICAT...
APPLICAT...
TASK1
TASK2
TASK3
TASK4
TASK5
TASK6
GetRes...
ReleaseResource()
GetResource()
act_tsk(5)
act_tsk(5)
act_tsk(5)
act_tsk(5)
act_tsk(5)
act_tskによる明示的なタスク起動

# シナリオ機能のサンプルアプリケーション

## シナリオ記述ファイル

```
1:2:act_tsk:4   ← 時刻1でコア2のタスク4を起動する
10:2:act_tsk:4
# mode2を2に変更する    ← #で始まる行はコメント
12:2:chg_mod:@@mode2:2    ← 時刻12でコア2の変数@@mode2に2を代入する
15:2:act_tsk:4
20:2:chg_mod:@@mode2:1
26:2:act_tsk:4
```

# シナリオ機能サンプルアプリケーションのタスク記述

## アプリケーション2のタスク処理記述の変更部分

```
 @@mode2 = 1    ← 動作モードの定義
# タスクIDが4のタスク処理記述
def task4
      case @@mode2
      when 1
          # モード1のときの動作
           act_tsk(5)      ← 動作モードが1のときだけタスク5を起動する
           exc(2)
      when 2
           # モード2のときの動作
           exc(2)
      end
end
```

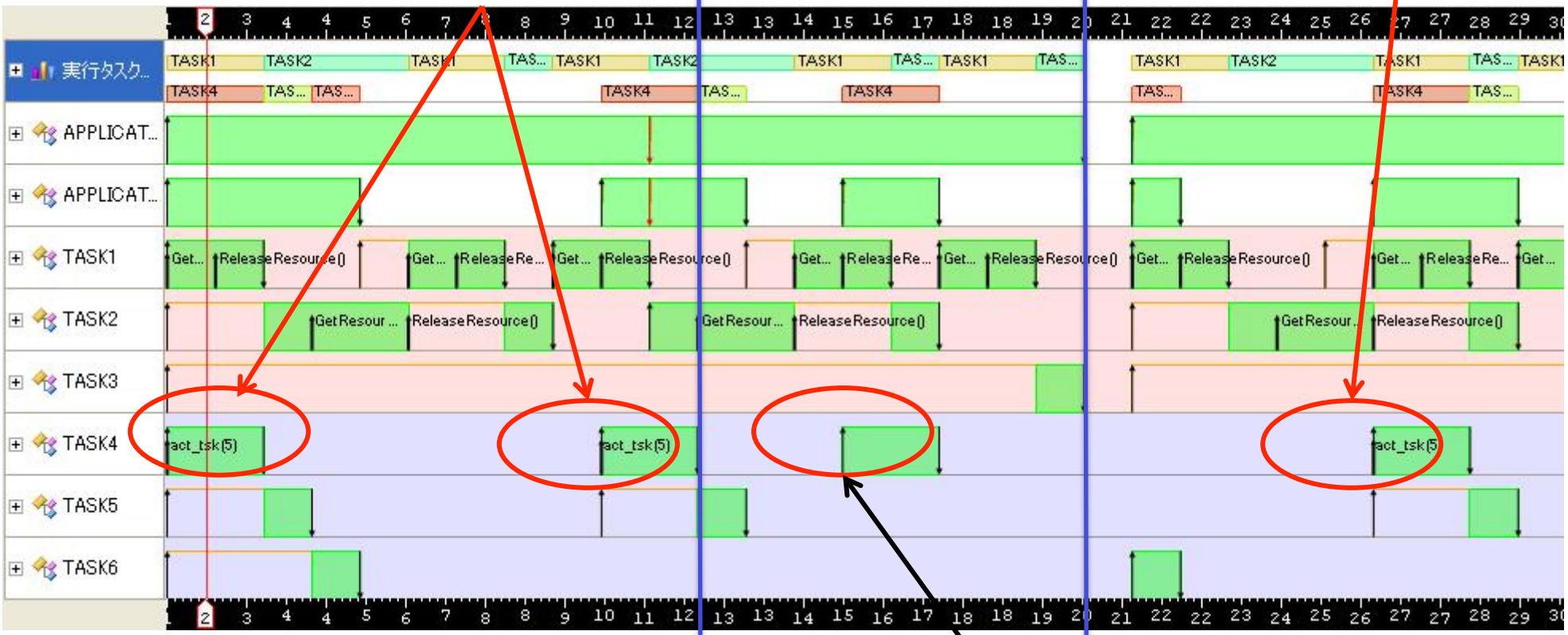
# サンプルアプリケーションを動かす

1. シナリオ機能サンプルアプリケーション scenarioを動かす
   - シミュレーション結果を格納するディレクトリの作成
     - mkdir obj
   - シミュレーションの実行
     - ./schesim.rb -t ./sample/scenario.json -r ./obj/scenario.res -d sample/scenario.rb -c sample/scenario.scn -e 40 > obj/scenario.log
2. TLVでの表示
   - objディレクトリ以下に生成されたscenario.logとscenario.resをTLVに入力してログを可視化する
     - 詳細はTLVのドキュメントを参照のこと

# テストシナリオの動作例

@@mode2 = 1 の区間
TASK4はTASK5を起動する

@@mode2 = 1 の区間
TASK4はTASK5を起動する




@@mode2 = 2 の区間
TASK4はTASK5を起動しない

# イベント機能のサンプルタスクセット

| アプリケーションID | タスクID | 優先度 | 周期 | WCET | 起動属性 |
|---|---|---|---|---|---|
| 1 | 1 | 5 | 4 | 2 | 周期起動 |
| 1 | 2 | 6 | 10 | 4 | 周期起動 |
| 1 | 3 | 7 | 20 | 1 | 周期起動 |
| 2 | 4 | 5 | 10 | 2 | 非周期 |
| 2 | 5 | 6 | 10 | 1 | 周期起動 |
| 2 | 6 | 7 | 20 | 1 | 周期起動 |

シナリオ記述で，システム動作開始時に１度だけ起動する

停止しない（待ち→待ち解除を繰り返す）タスクの場合は，待ち解除から待ちになるまでの処理を１回の実行と考えて，相対デッドラインとWCETを設定する．

# イベント機能のサンプルプログラム

## シナリオ記述ファイル

```
1:2:act_tsk:4 ← タスク4を初期起動
10:2:chg_mod:@@mode2:2
20:2:chg_mod:@@mode2:1
30:2:chg_mod:@@mode2:2
```

## アプリケーション2のタスク処理記述の変更部分

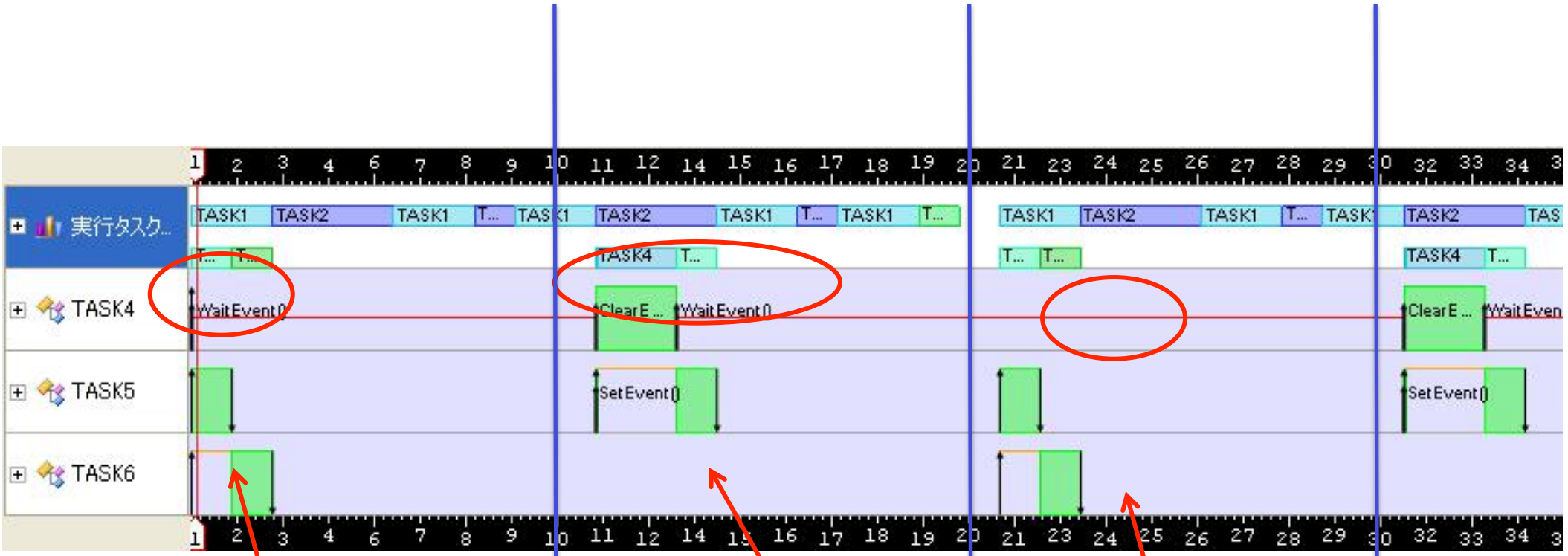
```
@@evt1 = VENT_FLAG.new()← イベントの生成
@@mode2 = 2
# タスクIDが4のタスク処理記述
def task4
    while (1)
        WaitEvent(@@evt1) ← @@evt1を待つ
        ClearEvent(@@evt1)← @@evt1をクリア
        exc(2)
    end
end
```

```
def task5
        case @@mode2
        when 1
            exc(1)
        when 2                ↓ @@evt1をセット
            SetEvent(@@evt1)
            exc(1)
        end
end
```

# サンプルアプリケーションを動かす

1. イベント機能サンプルアプリケーション eventを動かす
   - シミュレーション結果を格納するディレクトリの作成
     - mkdir obj
   - シミュレーションの実行
     - ./schesim.rb -t ./sample/event.json -r ./obj/event.res -d sample/event.rb -c sample/event.scn -e 40 > obj/event.log
2. TLVでの表示
   - objディレクトリ以下に生成されたevent.logとevent.resをTLVに入力してログを可視化する
     - 詳細はTLVのドキュメントを参照のこと

# イベントの動作例




TASK4がWaitEventする
@@mode2 = 1
TASK5はSetEventしない

@@mode2 = 2
TASK5はSetEventする

@@mode2 = 1
TASK5はSetEventしない

# 初期起動オフセット機能用サンプルタスクセット

シミュレーション
開始時に起動

| タスク ID | 優先度 | 周期 | WCET | オフセット | 起動属性 |
|---|---|---|---|---|---|
| 1 | 1 | 10 | 5 | 0 | 周期起動 |
| 2 | 1 | 10 | 5 | — | 周期起動 |
| 3 | 1 | 10 | 5 | 5 | 周期起動 |

システム時刻5で起動

オフセットを設定しない場合
シミュレーション中に起動要
求がない限り起動しない

# 初期起動オフセット機能用サンプルプログラム

## アプリケーション情報ファイル

```
"task":[
        {
            "id": 1,
            "offset": 0,
            "period": 10,
            "priority": 1,
            "deadline": 10,
            "wcet": 7
        }
        {
            "id": 2,
            "offset": 0,
            ・・・・・
        }
        {
            "id": 3,
            "offset": 5,
            ・・・・・
        }
]
```

↑ タスク2のオフセットは記述しない

## タスク処理記述

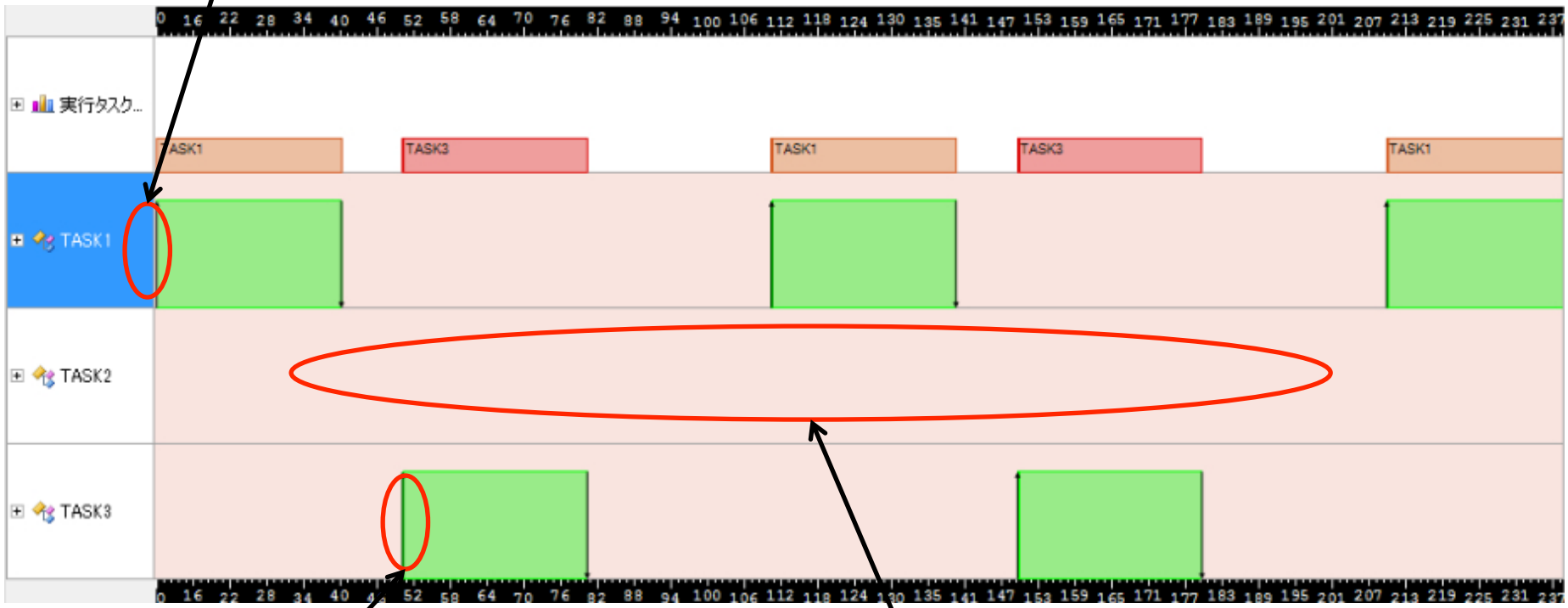
```
def task1
    exc(5)
end

def task2
    exc(5)
end

def task3
    exc(5)
end
```

# サンプルアプリケーションを動かす

1. 初期起動オフセット機能サンプルアプリケーションoffsetを動かす
   - シミュレーション結果を格納するディレクトリの作成
     - mkdir obj
   - シミュレーションの実行
     - ./schesim.rb -t ./sample/offset.json -r ./obj/offset.res -d sample/offset.rb > obj/offset.log
2. TLVでの表示
   - objディレクトリ以下に生成されたoffset.logとoffset.resをTLVに入力してログを可視化する
     - 詳細はTLVのドキュメントを参照のこと

# 初期起動オフセット機能の動作例

最初の時刻に自動起動




オフセット5を指定した
ので時刻5で起動

起動命令がない限り
起動しない

# 区間を指定した統計情報取得機能のサンプルタスクセット

| タスクID | 優先度 | 周期 | WCET | オフセット | 起動属性 |
|---|---|---|---|---|---|
| 1 | 1 | 10 | 3 | 0 | 周期起動 |
| 2 | 2 | 15 | 7 | 0 | 周期起動 |

タスク処理記述

```
def task1
    begin_measure(1)
    exc(3)
    end_measure(1)
end

def task2
    begin_measure(3)
    exc(4)
    begin_measure(2)
    exc(3)
    end_measure(2)
    end_measure(3)
end
```

# サンプルアプリケーションを動かす

1. 区間計測サンプルアプリケーションmeasureを動かす
   - シミュレーション結果を格納するディレクトリの作成
     - mkdir obj
   - シミュレーションの実行
     - ./schesim.rb -t ./sample/measure.json -d sample/measure.rb > obj/measure.log
2. 統計情報取得ツールで確認
   - objディレクトリ以下に生成されたmeasure.logを統計情報取得ツールに入力して計測区間を確認する
     - ./utils/stats.rb -i ./obj/measure.log -o ./obj/measure.xls
     - ./obj/measure.xlsを開いて確認

# 区間計測機能の出力例

| | A | B | C | D | E | F | G | H | I | J | K |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | タスクID | 起動回数 | 平均応答時間 | 最大応答時間 | 最小応答時間 | 総実行時間 | 平均実行時間 | 最大実行時間 | 最小実行時間 | CPU利用率 | |
| 2 | 1 | 4 | 30 | 30 | 30 | 90 | 22 | 30 | 30 | 30 | |
| 3 | 2 | 2 | 100 | 100 | 100 | 140 | 70 | 70 | 70 | 46.6666667 | |
| 4 | | | | | | | | | | | |
| 5 | | | | | | | | | | | |
| 6 | | | | | | | | | | | |
| 7 | | | | | | | | | | | |
| 8 | | | | | | | | | | | |
| 9 | | | | | | | | | | | |

タスク情報 | システム情報 | measure情報 | +

標準表示 | コマンド | 合計=0

区間計測による統計情報のシートへ

| | A | B | C | D | E | F | G | H | I | J | K | L |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | ID | 計測回数 | 平均応答時間 | 最大応答時間 | 最小応答時間 | 平均実行時間 | 最大実行時間 | 最小実行時間 | | | | |
| 2 | 1 | 3 | 30 | 30 | 30 | 30 | 30 | 30 | | | | |
| 3 | 2 | 2 | 45 | 60 | 30 | 30 | 30 | 30 | | | | |
| 4 | 3 | 2 | 85 | 100 | 70 | 70 | 70 | 70 | | | | |
| 5 | | | | | | | | | | | | |
| 6 | | | | | | | | | | | | |
| 7 | | | | | | | | | | | | |
| 8 | | | | | | | | | | | | |
| 9 | | | | | | | | | | | | |

タスク情報 | システム情報 | measure情報 | +

標準表示 | コマンド | 合計=0

各統計項目の定義はユーザズマニュアルを参照のこと