SWEST22

修正第2版

Zig言語およびZen言語による
リアルタイムOSの実現

2020年8月21日

高田 広章

NPO法人 TOPPERSプロジェクト 会長
名古屋大学 未来社会創造機構 モビリティ社会研究所 教授
名古屋大学 大学院情報学研究科 教授
附属組込みシステム研究センター長
APTJ株式会社 代表取締役会長兼社長
Email: hiro@ertl.jp　URL: http://www.ertl.jp/~hiro/

AGENDA

取り組みの背景

Zig言語とZen言語の紹介

- プログラミング言語 Zig
- プログラミング言語 Zen

Zig言語およびZen言語（Zxx言語）によるRTOSの実現

- 実装の概要，ZxxによるRTOSの実装コード例
- 静的APIとコンフィギュレータの置き換え
- 実現したRTOSの性能評価
- Zig言語およびZen言語の現時点での評価

プログラミング言語の安全性

今後の取り組みとソースコードへのアクセス

［付録］Zigの言語仕様のチラ見せ：ポインタとその仲間

取り組みの背景

組込みシステム開発技術の革新

- 情報技術が進む中で，新しい技術を取り込んで，組込みシステム開発技術も革新させていくべき
  - 組込みシステム開発技術の“デジタルトランスフォーメーション”
- オープンソースソフトウェアは，新しい開発技術をトライアルするには良い題材
  - トライアルした成果をオープンに議論できる

問題意識：いつまでもC言語で良いのか？

- 多くの組込みシステム開発者が感じている問題意識
- 新しいプログラミング言語は常に登場しており，いつが乗り換え時かは難しい

プログラミング言語 Zig

☞ https://ziglang.org/

Zigとは？（Zigのウェブサイトより）

- Zig is a general-purpose programming language and toolchain for maintaining **robust**, **optimal**, and **reusable** (+ **maintainable**) software.

言語の位置付け

- C言語を現代風に改良・拡張したもの
  - OSカーネル, 組込みシステム, リアルタイムシステムは想定されている用途

なぜZigを選んだか？

- 静的APIを置き換えられる可能性があると考えたため
  - コンパイル時のコード実行の機能
- RTOSの記述に向いていると思われるため
  - すべての資源を明示的に管理できる

他の新しいプログラミング言語との位置付け

- 組込みシステムにも向いた言語として，GoやRustなどが注目されているが…
  - "Go is better Java/C#"
  - "Rust is better C++"
  - "Zig is better C"
  - ☞ https://kristoff.it/blog/why-go-and-not-rust/
- 明確な違い：ヒープ領域の扱い
  - Go：ヒープ領域の使用が前提．領域の解放は，ガベージコレクタ（GC）により行う
  - Rust：ヒープ領域の使用が前提．「所有権」の概念の導入により，解放するタイミングを静的に決定
  - Zig：ヒープ領域は必須ではない．使う場合には，明示的にアロケータを呼ぶ（複数のアロケータを持つことができ，どのアロケータを使うかも明示する）．解放も明示

Zigの言語設計における基本的な方針

- プログラムの振る舞いをすべて記述
  - 隠れた制御フローがない(プログラムは見た目の通りに動く)
  - 手動のメモリ管理(メモリの扱いを明示的に記述)
- 性能を重視しつつ，安全性も重視する
  - 性能と安全性は同時には両立できないため，ビルドオプションにより，性能を取るか安全性を取るかを選択(ReleaseSafeとReleaseFast/ReleaseSmall)
  - ただし，メモリ安全な言語ではない
- シンプルな文法(小さい言語仕様)
  - ただし，ビルトイン関数と標準ライブラリは別(ビルトイン関数は100個くらいある．標準ライブラリはそれなりの規模．これは仕方がない)
- C言語との共存を容易に

Zig言語の重要な特徴（設計方針）

- コンパイル時のコード実行（comptime）
  - 最適化と合わせて，プリプロセッサが不要に
- 安全性の向上
  - ポインタの種類分け
  - オプショナル型（NULLポインタの代替）
  - 安全なunion
- 記述性の向上
  - 新しいエラー処理のアプローチ
  - defer文，ラベル付きのbreak文とcontinue文（goto-less）
  - ジェネリックなデータ構造と関数
  - コンパイル時のリフレクション（メタプログラミング）
  - 非同期関数，並行性のサポート（☆未調査）
- 単体テスト記述の統合
- ビルドシステム（Makefileの代替）をZig自身で記述できる

開発者

- Andrew Kelley氏
  - Zig開発に対する寄附で生活している
  - 寄附は，10ドル/月のプランから（最大6,000ドル/月）
- 多数のコントリビュータ
- The Zig Software Foundationを2020年3月に設立
  - ソフトウェア技術者を雇用している

ライセンス

- 言語処理系は，MITライセンス（一条項BSDライセンスと同等）によるオープンソースソフトウェア

最新版

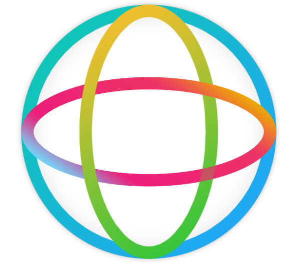
- Release 0.6.0（2020年4月にリリース）
- 毎日スナップショットがリリースされる
- Release 0.7.0は，2020年9月末？

Zigの思想

意図を正確に伝える

```
% zig zen
 * Communicate intent precisely.
 * Edge cases matter.
 * Favor reading code over writing code.
 * Only one obvious way to do things.
 * Runtime crashes are better than bugs.
 * Compile errors are better than runtime crashes.
 * Incremental improvements.
 * Avoid local maximums.
 * Reduce the amount one must remember.
 * Minimize energy spent on coding style.
 * Together we serve end users.
```

プログラミング言語 Zen



☞ https://zen-lang.org/ja-JP/

Zenとは？（Zenのウェブサイトより）

- Zen言語は，安全で善いコードを漸進的に構築可能なシステムプログラミング言語（Zenの禅：善，全，漸）

言語の位置付け

- Zig（Zig 0.3.0）から派生した言語
  - Zigの主な特徴を継承
- 独自の拡張機能
  - インタフェース機能など

開発者

- 帝都久利寿（Kristopher Tate）氏
- コネクトフリー（本社：京都）
  - Zenに対する有料サポートサービスを提供

ライセンス

- 言語処理系の利用
  - 内部利用（internal purposes，reference use）は無償
  - 商用利用には，ConnectFree Developer Networkの有料会員になることが必要（$99USD／年・チーム）
  - 公開情報では不明確なところが多い（改善を提案中）
- 言語処理系のソースコードは，（現時点では）クローズ

最新版

- Release 0.8.0LTS（2020年4月にリリース）
- 数週間に1回程度，バージョンアップされる
- Release 0.9.0の開発が進行中

Zig言語およびZen言語によるRTOSの実現

実装の概要

- TOPPERS/ASP3カーネル（ターゲットプロセッサ：ARM）のカーネル本体をZigおよびZen（以下，*Zxx*と書く）で実装
  - libkernel.aの全体を*Zxx*＋インラインアセンブラで記述することに成功（.cファイル，.Sファイルは追放した）
- コンフィギュレーション記述（静的API）とコンフィギュレータ（Rubyで記述）も*Zxx*で実装（パス3は除く）
  - *Zxx*の特性や制限から，一部の仕様を変更
- アプリケーションとシステムサービス（C言語＋TECSで記述）は修正なし
  - サンプルアプリケーションの*Zxx*版も作成
- ビルドシステム（Makefile）には修正を加えた
  - ここも*Zxx*で実現したいが，現時点ではできていない

Zxxによる RTOSの実装コード例

以下では…

- Zigで記述したASP3カーネル本体の実際のコードの一部を示す
  - C言語（点線の囲みの中）とZig（実線の囲みの中）を比較して解説
  - ZigとZenのコードの違い（わずか）も示す
  - 分かりやすさのために，少し加工してある

示すコード

- サービスコール（act_tsk）
- エラーセット型の定義
- act_tskのC言語API

サービスコール（act_tsk）

```c
ER act_tsk(ID tskid) {
   TCB   *p_tcb;
   ER    ercd;

   LOG_ACT_TSK_ENTER(tskid);
   CHECK_UNL();
   if (tskid == TSK_SELF && !sense_context()) {
      p_tcb = p_runtsk;
   } else {
      CHECK_ID(VALID_TSKID(tskid));
      p_tcb = get_tcb(tskid);
   }
   lock_cpu();
   if (TSTAT_DORMANT(p_tcb->tstat)) {
      make_active(p_tcb);
      if (p_runtsk != p_schedtsk) {
         if (!sense_context()) {
            dispatch();
         } else {
            request_dispatch_retint();
         }
      }
      ercd = E_OK;
   }
   else if ((p_tcb->p_tinib->tskatr & TA_NOACTQUE)
          != 0U || p_tcb->actque == TMAX_ACTCNT) {
      ercd = E_QOVR;
   } else {
      p_tcb->actque += 1;
      ercd = E_OK;
   }
   unlock_cpu();

  error_exit:
   LOG_ACT_TSK_LEAVE(ercd);
   return(ercd);
}
```

```zig
pub fn act_tsk(tskid: ID) ItronError!void {
   var p_tcb: *TCB = undefined;

   traceLog("actTskEnter", .{ tskid });
   errdefer |err| traceLog("actTskLeave", .{ err });
   try checkContextUnlock();
   if (tskid == TSK_SELF and !target_impl.senseContext()) {
      p_tcb = p_runtsk.?;
   }
   else {
      p_tcb = try checkAndGetTCB(tskid);
   }
   {
      target_impl.lockCpu();
      defer target_impl.unlockCpu();

      if (isDormant(p_tcb.tstat)) {
         make_active(p_tcb);
         if (p_runtsk != p_schedtsk) {
            if (!target_impl.senseContext()) {
               target_impl.dispatch();
            } else {
               target_impl.requestDispatchRetint();
            }
         }
      }
      else if ((p_tcb.p_tinib.tskatr & TA_NOACTQUE) != 0
                    or p_tcb.flags.actque == TMAX_ACTCNT) {
         return ItronError.QueueingOverflow;
      }
      else {
         p_tcb.flags.actque += 1;
      }
   }
   traceLog("actTskLeave", .{ null });
}
```

サービスコール（act_tsk）

API名はそのまま
(Zigの命名ガイドラインを逸脱)

```
ER act_tsk(ID tskid) {
   TCB    *p_tcb;
   ER     ercd;

   LOG_ACT_TSK_ENTER(tskid);
   CHECK_UNL();
   if (tskid == TSK_SELF && !se
      p_tcb = p_runtsk;
   } else {
      CHECK_ID(VALID_TSKID(tskid));
      p_tcb = get_tcb(t
   }
   lock_cpu();
   if (TSTAT_DORMANT(p_
      make_active(p_tcb
      if (p_runtsk != p_schedtsk) {
         if (!sense_context()) {
            dispatch();
         } else {
            request_dispatch_retint();
         }
      }
      ercd = E_OK;
   }
   else if ((p_tcb->p_tinib->tskatr & TA_NOACTQUE)
          != 0U || p_tcb->actque == TMAX_ACTCNT) {
      ercd = E_QOVR;
   } else {
      p_tcb->actque += 1;
      ercd = E_OK;
   }
   unlock_cpu();

  error_exit:
   LOG_ACT_TSK_LEAVE(ercd);
   return(ercd);
}
```

```
pub fn act_tsk(tskid: ID) ItronError!void {
    var p_tcb: *TCB = undefined;

    trac Log("actTsk ter", .{ tskid });
    err  |err|  ceLog("actTskLeave        err });
    try  Conte   lock();
    if (tskid == T   ELF and !target
        p_tcb =
    }
    else {
        p_tcb                          tsk

        target
        defer target_impl.unloc    u();

        if (isDormant(p_tcb.tst
                                                {
                target_impl.dispatch();
            } else {
                target_impl.requestDispatchRetint();
            }
        }
    }
    else if ((p_tcb.p_tinib.tskatr & TA_NOACTQUE) != 0
                 or p_tcb.flags.actque == TMAX_ACTCNT) {
        return ItronError.QueueingOverflow;
    }
    else {
        p_tcb.flags.actque += 1;
    }
    }
    traceLog("actTskLeave", .{ null });
}
```

ファイル外に見せる
C言語のstaticの反意

関数

「引数名: 型名」
引数はconst
関数内で代入
できない

関数の戻り値の型
ItronError型 (エラーセット型) を返すか
何も返さない

constは，定数というより，
定義した後に値を変えられない
変数と理解した方が良い

値で渡すかポインタで渡すかは，
コンパイラが決める

サービスコール（act_tsk）

```
ER act_tsk(ID tskid) {
   TCB    *p_tcb;
   ER     ercd;

   LOG_ACT_TSK_ENTER(tskid);
   CHECK_UNL();
   if (tskid == TSK_SELF && !sense_context()) {
      p_tcb = p_runtsk;
   } else {
      CHECK_ID(VALID_TSKID(tskid));
      p_tcb = get_tcb(tskid);
   }
   lock_cpu();
   if (TSTAT_DORMANT(p_tcb->tstat)) {
      make_active(p_tcb);
      if (p_runtsk != p_schedtsk) {
         if (!sense_context()) {
            dispatch();
         } else {
            request_dispatch_retint();
         }
      }
      ercd = E_OK;
   }
   else if ((p_tcb->p_tinib->tskatr & TA_NOACTQUE)
           != 0U || p_tcb->actque == TMAX_ACTCNT) {
      ercd = E_QOVR;
   } else {
      p_tcb->actque += 1;
      ercd = E_OK;
   }
   unlock_cpu();

  error_exit:
   LOG_ACT_TSK_LEAVE(ercd);
   return(ercd);
}
```

```
pub fn act_tsk(tskid: ID) ItronError!void {
   var p_tcb: *TCB = undefined;

   traceLog("ac  skEnter", .  tskid });
   errdefer |er  traceLog("a   Leave", .{ err });
   try checkCo   Unlock();
   ...F ar ... )) {
   ...?;
   }
   ...se {
      p_tcb = try checkAndG...
   ...
   ...rget_impl.lockCpu();
   ...fer target_impl.unlockCpu();
   ...(isDormant(p_tcb.tstat)) {
      make_active(p_tcb);
      if (p_runtsk != p_schedtsk) {
         if (!target_impl.senseContext()) {
            target_impl.dispatch();
         } else {
            target_impl.requestDispatchRetint();
         }
      }
   }
   else if ((p_tcb.p_tinib.tskatr & TA_NOACTQUE) != 0
                 or p_tcb.flags.actque == TMAX_ACTCNT) {
      return ItronError.QueueingOverflow;
   }
   else {
      p_tcb.flags.actque += 1;
   }
   }
   traceLog("actTskLeave", .{ null });
}
```

(ローカル）変数

TCB型へのポインタ

初期値は不定値
（動作が未定義になるので推奨されない）

初期値は必ず記述しなければならない

Zenでは，“*mut TCB” とすることが必要

サービスコール（act_tsk）

```
ER act_tsk(ID tskid) {
    TCB     *p_tcb;
    ER      ercd;

    LOG_ACT_TSK_ENTER(tskid);
    CHECK_UNL();
    if (tskid == TSK_SELF && !ser
        p_tcb = p_runtsk;
    } else {
        CHECK_ID(VALID_TSKID(tskid));
        p_tcb = get_tcb(tskid);
    }
    lock_cpu();
    if (TSTAT_DORMANT(p_tcb->tstat)) {
        make_active(p_tcb);
        if (p_runtsk != p_schedtsk) {
            if (!sense_context()) {
                dispatch();
            } else {
                reque
            }
        }
        ercd = E_OK;
    }
    else if ((p_tcb->p_tinib->tskatr &
                != 0U || p_tcb->actque == T
        ercd = E_QOVR;
    } else {
        p_tcb->actque += 1;
        ercd = E_OK;
    }
    unlock_cpu();

  error_exit:
    LOG_ACT_TSK_LEAVE(ercd);
    return(ercd);
}
```

```
pub     act_tsk(tskid: ID) ItronError!void {
    va   p_tcb: *TCB = undefined;

    traceLog("actTskEnter", .{ tskid });
    errdefer |err| traceLog("actTskLeave", .{ err });
    try checkCont xtUnlock();
    if (tskid == T  SELF and !target_impl.senseContext()) {
        p_tcb = p r
    }
    se {
        p_tcb =                      id);
    }

        target_impl.lockCpu();
        defer target_impl.unlockCpu();

        if (isDormant(p_tcb.ts
            make_active(p_tcb)
            if (p_runtsk != p_s
                if (!target impl.sense       )) {
                    tar             atch
                } else {
                    target_impl    equestDisp  chRetint();
                }
                                                        {
                p_tcb.flags.actque += 1;
            }
    }
    traceLog("actTskLeave", .{ null });
}
```

```
pub fn traceLog(comptime log_type: []const u8, args: var) void {
    if (@hasDecl(option.log, log_type)) {
        @field(option.log, log_type)(args);
    }
}
```

Zigの命名ガイドラインに従った

この場合は関数

このブロックからエラーで抜ける時に後ろの文を実行する

この変数にエラーを受け取る

任意の型のパラメータのリスト C言語のvarargに代わるもの

option.logネームスペースに，actTskLeaveの定義があるか？

文字列の型

エラーでない場合は，nullを渡す

サービスコール（act_tsk）

```
ER act_tsk(ID tskid) {
   TCB    *p_tcb;
   ER     ercd;

   LOG_ACT_TSK_ENTER(tskid);
   CHECK_UNL();
   if (tskid == TSK_SELF && !sense_context()) {
      p_tcb = p_runtsk;
   }
   ...D(tskid));
   ...id);
   lock_cpu();
   if (TSTAT_DORMANT(...>tstat)) {
      make_act...
      if (p_runtsk != p_schedtsk) {
         if (!sense_context()) {
            dispatch();
         } else {
            request_dispatch_retint();
         }
      }
      ercd = E_OK;
   }
   else if ((p_tcb->p_tinib->tskatr & TA_...
         != 0U || p_tcb->actque == TMAX_ACTCNT) {
      ercd = E_QOVR;
   } else {
      p_tcb->actque += 1;
      ercd = E_OK;
   }
   unlock_cpu();

  error_exit:
   LOG_ACT_TSK_LEAVE(ercd);
   return(ercd);
}
```

エラーの場合，goto error_exitする

隠れた制御フローがある！

```
pub fn act_tsk(tskid: ID) ItronError!void {
   var p_tcb: *TCB = undefined;

   traceLog("actTskEnter", .{ tskid });
   errdefer |err| traceLog("actTskLeave", .{ err });
   try checkContextUnlock();
   if (tskid == TSK_SELF and !target_impl.senseContext()) {
      p_tcb = p_runtsk.?;
   ...
   ...tTCB(tskid);
   ...
      defer target_impl.unlockCpu();
   ...
   ...) {
   ...get_impl.dispatch();
   ...e {
   ...imp...();
   ...
      }
      else if ((p_tcb.p_tinib.tskatr & TA_NOACTQUE) != 0
                    or p_tcb.flags.actque == TMAX_ACTCNT) {
         return ItronError.QueueingOverflow;
      }
      else {
         p_tcb.flags.actque += 1;
      }
   }
   traceLog("actTskLeave", .{ null });
}
```

右の式がエラーを返した場合，そのエラーをreturnする（大域脱出的）

下のように書くこともできる
checkContextUnlock() catch |err| return err;

左の式がエラーを返した場合，右の式を実行する

この変数にエラーを受け取る

サービスコール（act_tsk）

```
ER act_tsk(ID tskid) {
   TCB    *p_tcb;
   ER     ercd;

   LOG_ACT_TSK_ENTER(tskid);
   CHECK_UNL();
   if (tskid == TSK_SELF && !sense_context()) {
      p_tcb = p_runtsk;
   } else {
      CHECK_ID(VALID_TSKID(tskid));
                                  tat)) {
                                  sk) {
                                    {
         } else {
            request_dispatch_retint();
         }
      }
      ercd = E_OK;
   }
   else if ((p_tcb->p_tinib->tskatr & TA_NOACTQUE)
            != 0U || p_tcb->actque == TMAX_ACTCNT) {
      ercd = E_QOVR;
   } else {
      p_tcb->actque += 1;
      ercd = E_OK;
   }
   unlock_cpu();

  error_exit:
   LOG_ACT_TSK_LEAVE(ercd);
   return(ercd);
}
```

```
#define CHECK_UNL() do { ¥
    if (sense_lock()) {  ¥
        ercd = E_CTX;    ¥
        goto error_exit; ¥
    }                    ¥
} while (false)
```

```
pub fn act_tsk(tskid: ID) ItronError!void {
   var p_tcb: *TCB = undefined;

   traceLog("actTskEnter", .{ tskid })
   errdefer |err| traceLog("actTskLeav
   try checkContextUnlock();
   if (tskid == TSK_SELF and !target_i
      p_tcb = p_runtsk.?;
   }
   else {
      p_tcb = try checkAndGetTCB(tskid);
         if (p_runtsk != p_schedt
            if (!target_impl.sens
               target_impl.dispatch();
            } else {
               target_impl.requestDispatchRetint();
            }
         }
      }
      else if ((p_tcb.p_tinib.tskatr & TA_NOACTQUE) != 0
                        or p_tcb.flags.actque == TMAX_ACTCNT) {
         return ItronError.QueueingOverflow;
      }
      else {
         p_tcb.flags.actque += 1;
      }
   }
   traceLog("actTskLeave", .{ null });
}
```

```
pub fn checkContextUnlock() ItronError!void {
   if (target_impl.senseLock()) {
      return ItronError.ContextError;
   }
}
```

ItronError型（エラーセット型）を返すか何も返さない

ItronError型のContextErrorを返す

elseの場合は何も返さない

サービスコール(act_tsk)

```
ER act_tsk(ID tskid) {
   TCB   *p_tcb;
   ER    ercd;

   LOG_ACT_TSK_ENTER(tskid);
   CHECK_UNL();
   if (tskid == TSK_SELF && !sense_context()) {
      p_tcb = p_runtsk;
   } else {
      CHECK_ID(VALID_TSKID(tskid));
      p_tcb = get_tcb(tskid);
   }
   lock_cpu();
   if (TSTAT_DORMANT(p_tcb->tstat)) {
      make_active(p_tcb);
      if (p_runtsk != p_schedtsk) {
         if (!sense_context()) {
            dispatch();
         } else {
            request_dispatch_retint();
         }
      }
      ercd = E_OK;
   }
   else if ((p_tcb->p_tinib->tskatr & TA_NOA
          != 0U || p_tcb->act
      ercd = E_QOVR;
   } else {
      p_tcb->actque += 1;
      ercd = E_OK;
   }
   unlock_cpu();

  error_exit:
   LOG_ACT_TSK_LEAVE(ercd);
   return(ercd);
}
```

```
pub                                    Error!void {
   var p_tcb: *TCB =          ined;

   traceLog("actTskEnt
   errdefer |err| trac                    , .{ err });
   try checkContextUnlock();
   if (tskid == TSK_SELF and !target_impl.senseContext()) {
      p_tcb = p_runtsk.?;
   }
   else {
      p           try checkT         TCB(tskid);
                                  lo
                                  i
                                  t(
                                  ve(p_tcb);
         if (p_runtsk != p_schedtsk) {
            if (!targ
            targe
            else {
            targe
            }
         }
      }
      else if ((p_tc                                    ) != 0
                 c                                      _ACTCNT) {
         return ItronError.QueueingOverflow;
      }
      else {
         p_tcb.flags.actque += 1;
      }
   }
   traceLog("actTskLeave", .{ null });
}
```

ターゲット依存部は
ネームスペースを分けた

ネームスペース

p_runtskは ?*TCB型
?はオプショナル型を示す
TCBへのポインタ
またはnullが代入できる

nullでない場合に,
データを取り出す
(危険な記述なので
推奨されない)

ポインタの値は0になれないため,
内部的にはnullは0で表される

ポインタの値に0を
許すという宣言もある

nullであった場合,
ReleaseSafeなら検出され
panicになる
ReleaseFast/Smallでは
検出されない

サービスコール（act_tsk）

```
ER act_tsk(ID tskid) {
   TCB   *p_tcb;
   ER    ercd;

   LOG_ACT_TSK_ENTER(tskid);
   CHECK_UNL();
   if (tskid == TSK_SELF && !sense_context()) {
      p_tcb = p_runtsk;
   } else {
      CHECK_ID(VALID_TSKID(tskid));
      p_tcb = get_tcb(tskid);
   }
   lock_cpu();
   if (TSTAT_DORMANT(p_tcb->tstat)) {
      make_active(p_tcb);
      if (p_runtsk != p_schedtsk) {
         if (!sense_context()) {
            dispatch();
         } else {
            request_dispatch_re
         }
      }
      ercd = E_OK;
   }
   else if ((p_tcb-                                   UE)
         != 0U ||                                     ) {
      ercd = E_QOVR
   } else {
      p_tcb->actque += 1;
      ercd = E_OK;
   }
   unlock_cpu();

  error_exit:
   LOG_ACT_TSK_LEAVE(ercd);
   return(ercd);
}
```

```
pub fn act_tsk(tskid: ID) ItronError!void {
   var p_tcb: *TCB = undefined;

   traceLog("actTskEnter", .{ tskid });
                          eLog("actTskLeave", .{ err });
                          ock();
                          LF and !target_impl.senseContext()) {
      p_tcb                 ntsk.?;
   }
   else {
      p_tcb = try checkAndGetTCB(tskid);
   }
   {
      target_impl.lockCpu();
      defer target_impl.unlockCpu();

      if (isDormant(p_tcb.tstat)) {

                                          NOACTQUE) != 0
                   or p_tcb.flags.actque == TMAX_ACTCNT) {
         return ItronError.QueueingOverflow;

                                          });
```

```
pub fn checkAndGetTCB(tskid: ID) ItronError!*TCB {
    try checkId(TMIN_TSKID <= tskid and tskid <= cfg._kernel_tmax_tskid);
    return &cfg._kernel_tcb_table[indexTask(tskid)];
}
```

```
pub fn checkId(ok: bool) ItronError!void {
    if (!ok) {
        return ItronError.IdError;
    }
}
```

右の式がエラーなら
関数からエラーリターン

tskidが有効範囲外なら，
エラーを返す

tskidが範囲内なら，そのタスクの
TCBの先頭番地を返す

elseの場合は
何も返さない

ItronError型の
IdErrorを返す

サービスコール(act_tsk)

```
ER act_tsk(ID tskid) {
   TCB    *p_tcb;
   ER     ercd;

   LOG_ACT_TSK_ENTER(tskid);
   CHECK_UNL();
   if (tskid == TSK_SELF && !sense_context()) {
      p_tcb = p_runtsk;
   } else {
      CHECK_ID(VALID_TSKID(tskid));
      p_tcb = get_tcb(tskid);
   }
   lock_cpu();
   if (TSTAT_DORMANT(p_tcb->tstat)) {
      make_active(p_tcb);
      if (p_runtsk != p_schedtsk) {
         if (!sense_context()) {
            dispatch();
         } else {
            request_dispatch_retint();
         }
      }
      ercd = E_OK;
   }
   else if ((p_tcb->p_tinib->tskatr & TA_NOACTQU
          != 0U || p_tcb->actque ==
      ercd = E_QOVR;
   } else {
      p_tcb->actque += 1;
      ercd = E_OK;
   }
   unlock_cpu();

  error_exit:
   LOG_ACT_TSK_LEAVE(ercd);
   return(ercd);
}
```

```
pub fn act_tsk(tskid: ID) ItronError!void {
   var p_tcb: *TCB = undefined;

   traceLog("actTskEnter", .{ tskid });
   errdefer |err| traceLog("actTskLeave", .{ err });
   try checkContextUnlock();
   if (tskid == TSK_SELF and !target_impl.senseContext()) {
      p_tcb = p_runtsk.?;
   }
   else {
      p_tcb = try checkAndGetTCB(tskid);
   }
   {
      target_impl.lockCpu();
      defer target_impl.unlockCpu();
        (isDormant(p_tcb.tstat)) {
         make_active(p_tcb);
         f (p_runtsk != p_schedtsk) {
            if (!target_impl.senseContext()) {
               target_impl.dispatch();
            } else {
               target_impl.requestDispatchRetint();
            }
         }
      }
      else if ((p_tcb.p_tinib.tskatr & TA_NOACTQUE) != 0
                    or p_tcb.flags.actque == TMAX_ACTCNT) {
         return ItronError.QueueingOverflow;
      }
      else
                                                1;

   }
   traceLog("actTskLeave", .{ null });
}
```

このブロックから
抜ける時に
後ろの文を実行する

ブロック中から
returnしてもOK!

ブロック中からreturn

サービスコール（act_tsk）

```c
ER act_tsk(ID tskid) {
   TCB    *p_tcb;
   ER     ercd;

   LOG_ACT_TSK_ENTER(tskid);
   CHECK_UNL();
   if (tskid == TSK_SELF && !sense_context()) {
      p_tcb = p_runtsk;
   } else {
      CHECK_ID(VALID_TSKID(tskid));
      p_tcb = get_tcb(tskid);
   }
   lock_cpu();
   if (TSTAT_DORMANT(p_tcb->tstat)) {
      make_active(p_tcb);
      if (p_runtsk != p_schedtsk) {
         if (!sense_context()) {
            dispatch();
         } else {
            request_dispatch_retint();
         }
      }
      ercd = E_OK;
   }
   else if ((p_tcb->p_tinib->tskatr & TA_NOACTQUE)
          != 0U || p_tcb->actque == TMAX_ACTCNT) {
      ercd = E_QOVR;
   } else {
      p_tcb->actque += 1;
      ercd = E_OK;
   }
   unlock_cpu();

  error_exit:
   LOG_ACT_TSK_LEAVE(ercd);
   return(ercd);
}
```

```zig
pub fn act_tsk(tskid: ID) ItronError!void {
   var p_tcb: *TCB = undefined;

   traceLog("actTskEnter", .{ tskid });
   errdefer |err| traceLog("actTskLeave", .{ err });
   try checkContextUnlock();
   if (tskid == TSK_SELF and !target_impl.senseContext()) {
      p_tcb = p_runtsk.?;
   }
   else {
      p_tcb = try checkAndGetTCB(tskid);
   }
   {
      target_impl.lockCpu();
      defer target_impl.unlockCpu();

      if (isDormant(p_tcb.tstat)) {
         make_active(p_t  );
         if (p_runtsk      _schedtsk) {
            if (!t           enseContext()) {
                             atch();

                           estDispatchRetint();
            }
         }
      }
      else if ((p_tcb.p_tinib.tskatr & TA_NOACTQUE) != 0
                    or p_tcb.flags.actque == TMAX_ACTCNT) {
         return ItronError.QueueingOverflow;
      }
      else {
         p_tcb.flags.actque += 1;
      }
   }
   traceLog("actTskLeave", .{ null });
}
```

ポインタ型が左辺の “.” は
C言語の “->” の意味になる

エラーセット型の定義

```
///
///  サービスコールにおけるエラー
///
pub const ItronError = error {
    SystemError,
    NotSupported,
    ReservedFunction,
    ReservedAttribute,
    ParameterError,
    IdError,
    ContextError,
    MemoryAccessViolation,
    ObjectAccessViolation,
    IllegalUse,
    NoMemory,
    NoId,
    NoResource,
    ObjectStateError,
    NonExistent,
    QueueingOverflow,
    ReleasedFromWaiting,
    TimeoutError,
    ObjectDeleted,
    ConnectionClosed,
    TerminationRequestRaised,
    WouldBlock,
    BufferOverflow,
    CommunicationError,
};
```

act_tskのC言語API

```
// C言語ヘッダファイルの取り込み
pub const c = @cImport({
    @cDefine("UINT_C(val)", "val");
    @cInclude("kernel.h");
});

// C言語APIのエラーコードへの変換
fn itronErrorCode(err: ItronError) ER {
    return switch (err) {
        ItronError.SystemError => c.E_SYS,
        ItronError.NotSupported => c.E_NOSPT,
        ItronError.ReservedFunction => c.E_RSFN,
        …………
    };
}

// 返値をER型に変換
fn callService(result: ItronError!void) ER {
    if (result) {
        return c.E_OK;
    }
    else |err| {
        return itronErrorCode(err);
    }
}

// act_tskのC言語API
export fn act_tsk(tskid: ID) ER {
    return callService(task_manage.act_tsk(tskid));
}
```

エラーセット型の定義

ItronError型（エラーセット型）の定義

```
///
///  サービスコールにおけるエラー
///
pub const ItronError = error {
    SystemError,
    NotSupported,
    ReservedFunction,
    ReservedAttribute,
    ParameterError,
    IdError,
    ContextError,
    MemoryAccessViolation,
    ObjectAccessViolation,
    IllegalUse,
    NoMemory,
    NoId,
    NoResource,
    ObjectStateError,
    NonExistent,
    QueueingOverflow,
    ReleasedFromWaiting,
    TimeoutError,
    ObjectDeleted,
    ConnectionClosed,
    TerminationRequestRaised,
    WouldBlock,
    BufferOverflow,
    CommunicationError,
};
```

enum（列挙型）に似ている

act_tskのC言語API

ネームスペース

C言語ヘッダファイルのインクルード

```
// C言語ヘッダファイルの取り込み
pub const c = @cImport({
    @cDefine("UINT_C(val)", "val");
    @cInclude("kernel.h");
});

// C言語APIのエラーコードへの変換
fn itronErrorCode(err: ItronError) ER {
    return switch (err) {
        ItronError.SystemError => c.E_SYS,
        ItronError.NotSupported => c.E_NOSPT,
        ItronError.ReservedFunction => c.E_RSFN,
        …………
    };
}

// 返値をER型に変換
fn callService(result: ItronError!void) ER {
    if (result) {
        return c.E_OK;
    }
    else |err| {
        return itronErrorCode(err);
    }
}

// act_tskのC言語API
export fn act_tsk(tskid: ID) ER {
    return callService(task_manage.act_tsk(tskid));
}
```

C言語ヘッダファイルから取り込んだ定数値

ItronError型（エラーセット型）またはvoid

resultがエラーでない場合にifが成立

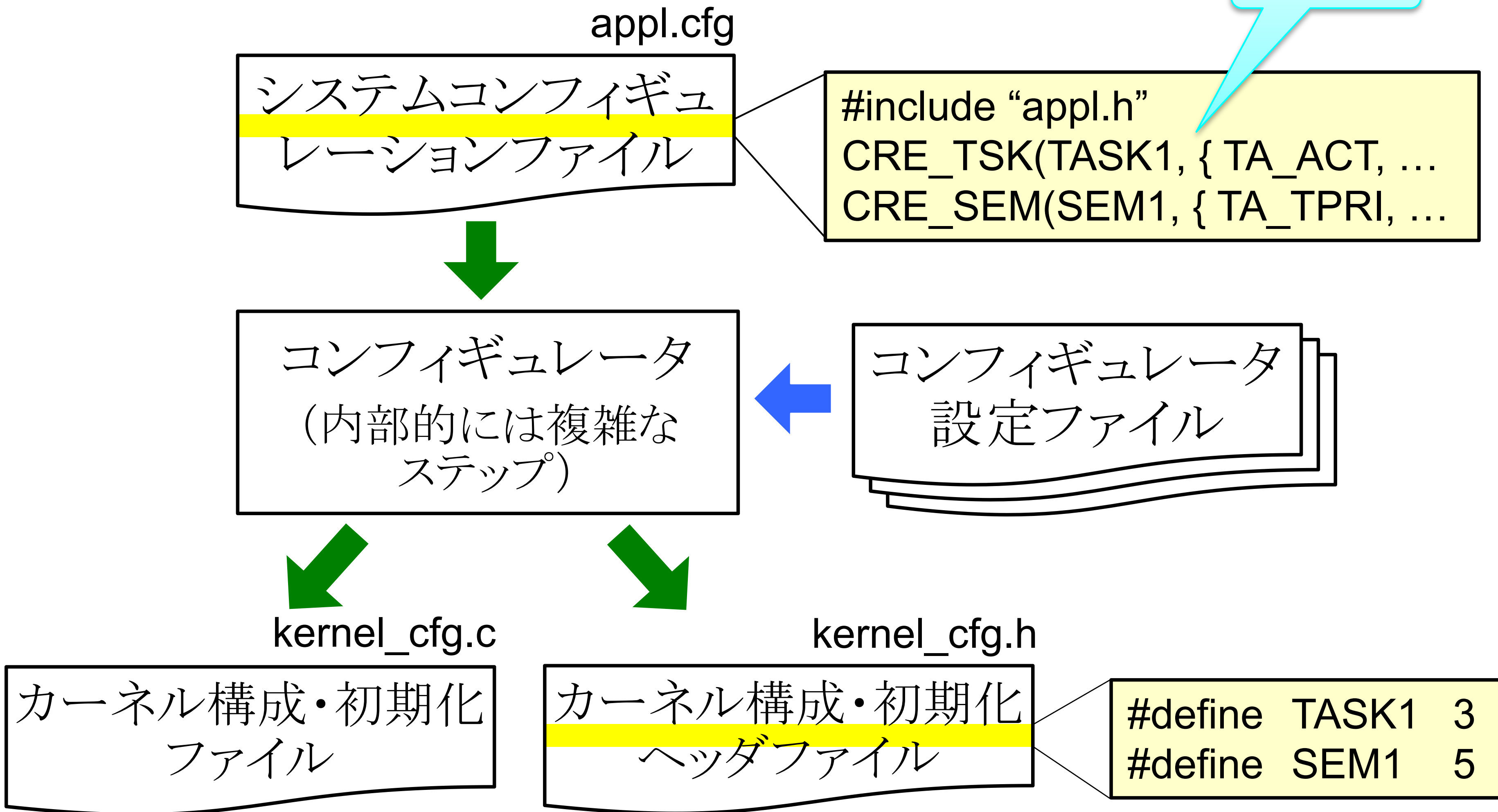
この変数にエラーを受け取る

C言語から呼べるようにする

libkernel.aの外部から呼べるようにする

ネームスペース

静的APIとコンフィギュレータの置き換え



静的APIとコンフィギュレータの入出力の例

システムコンフィギュレーションファイル

```
CRE_TSK( TASK1, { TA_ACT, 0, task_main,
                          10, STACK_SIZE, NULL } );
```

コンフィギュレータ

kernel_cfg.h

```
#define TASK1     3
```

kernel_cfg.c

```
static STK_T _kernel_stack_TASK1[COUNT_STK_T(STACK_SIZE)];
const TINIB _kernel_tinib_table[TNUM_TSKID] = {
        ……
        { (TA_ACT), (intptr_t)(0), ((TASK)(task_main)),
        INT_PRIORITY(10),   ROUND_STK_T(STACK_SIZE),
        _kernel_stack_TASK1 },
        ……
};
TCB _kernel_tcb_table[TNUM_TSKID];
```

スタック領域

タスク初期化ブロック

タスク管理ブロック

Zxxによる静的APIとコンフィギュレータの置き換えの目標

- 静的API（オブジェクトの生成記述）を，Zxxで記述する
  - 文法が少し変わってしまうのは，許容する
- コンフィギュレータ（に相当する処理）をZxxで記述する
  - カーネル構成・初期化ファイル（kernel_cfg.c）をZxxで生成するのではなく，直接データ構造を生成する
  - コンフィギュレータのパス1は不要になる
  - パス3は，まずはスコープ外とする（1つのパスに統合するのは難しいため）
- オブジェクトを動的に生成するサービスコールのコードが，そのまま静的に（コンパイル時に）実行されれば理想的だが，それは難しいと思われる
  - どこまで近いコードにできるかはトライしていない

静的API *vs.* Zigによるコンフィギュレーション記述

```
INCLUDE("tecsgen.cfg");

#include "test_sem2.h"

CRE_TSK(TASK1, { TA_ACT, 1, task1, MID_PRIORITY, STACK_SIZE, NULL });
CRE_TSK(TASK2, { TA_NULL, 2, task2, HIGH_PRIORITY, STACK_SIZE, NULL });
CRE_TSK(TASK3, { TA_NULL, 3, task3, LOW_PRIORITY, STACK_SIZE, NULL });
CRE_ALM(ALM1, { TA_NULL, { TNFY_HANDLER, 1, alarm1_handler }});
CRE_SEM(SEM1, { TA_NULL, 1, 1 });
CRE_SEM(SEM2, { TA_TPRI, 0, 1 });
```

静的APIによるコンフィギュレーション記述

```
usingnamespace @import("../kernel/kernel_cfg.zig");

const tecs = @import("../" ++ TECSGENDIR ++ "/tecsgen_cfg.zig");

usingnamespace @cImport({
    @cDefine("UINT_C(val)", "val");
    @cInclude("test_sem2.h");
});

fn configuration(comptime cfg: *CfgData) void {
    tecs.configuration(cfg);
    cfg.CRE_TSK("TASK1", CTSK(TA_ACT, 1, task1, MID_PRIORITY, STACK_SIZE, null));
    cfg.CRE_TSK("TASK2", CTSK(TA_NULL, 2, task2, HIGH_PRIORITY, STACK_SIZE, null));
    cfg.CRE_TSK("TASK3", CTSK(TA_NULL, 3, task3, LOW_PRIORITY, STACK_SIZE, null));
    cfg.CRE_ALM("ALM1", CALM(TA_NULL, NFY_TMEHDR(1, alarm1_handler)));
    cfg.CRE_SEM("SEM1", CSEM(TA_NULL, 1, 1));
    cfg.CRE_SEM("SEM2", CSEM(TA_TPRI, 0, 1));
}

//  静的APIの読み込みとコンフィギュレーションデータの生成
//  以下は変更する必要がない．
<以下省略>
```

Zigによるコンフィギュレーション記述

Zenでは，“*mut CfgData” とすることが必要

静的API *vs.* Zigによるコンフィギュレーション記述 – 拡大図

```
INCLUDE("tecsgen.cfg");
CRE_TSK(TASK1, { TA_ACT, 1, task1, MID_PRIORITY,
                                    STACK_SIZE, NULL });
```

システムコンフィギュレーション記述を行う関数

コンフィギュレーションデータを格納するデータ構造

```
fn configuration(comptime cfg: *CfgData) void {
    tecs.configuration(cfg);
    cfg.CRE_TSK("TASK1", CTSK(TA_ACT, 1, task1,
                     MID_PRIORITY, STACK_SIZE, null));
}
```

データ構造にタスクの生成情報を追加する関数

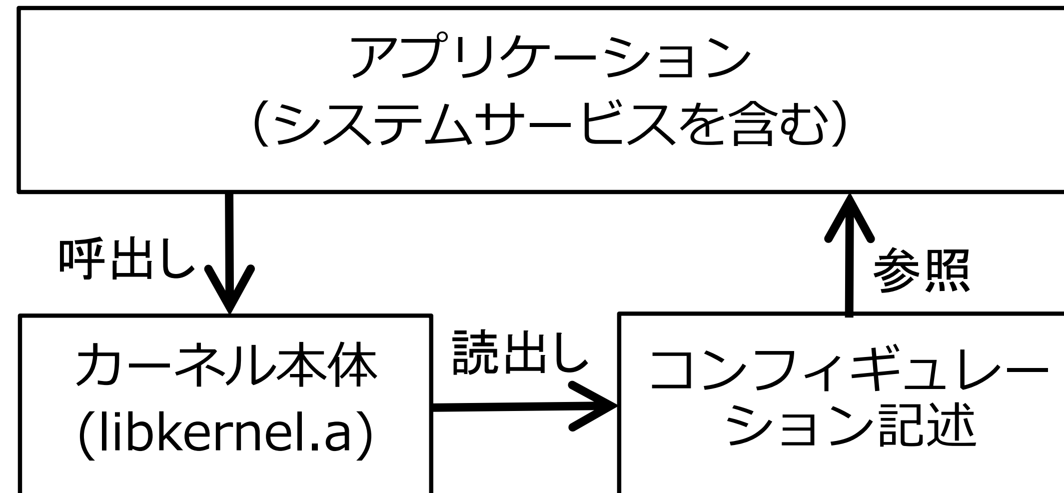
これらの記述をすべてコンパイル時に実行し，コンフィギュレーションデータを静的に生成

コンフィギュレータ実装のアプローチ

- アプローチ1
  - カーネル本体(libkernel.a)とコンフィギュレーション記述は個別にコンパイルし，リンカにより結合
  - カーネル本体のバイナリコードは，コンフィギュレーションによって変化しない
    - ✓コンフィギュレーション変更時の再コンパイルは不要
- アプローチ2
  - カーネル本体(libkernel.a)とコンフィギュレーション記述をまとめてコンパイル
  - コンフィギュレーションに最適化されたカーネルのバイナリコードが生成される
    - ✓例えば，使用するセマフォが1つだけなら，セマフォの初期化処理のループは展開されるだろう
  - テスト(品質保証)の観点からは良いかどうか疑問

コンフィギュレータ実装のアプローチ（続き）

- コンパイルオプションで，どちらのアプローチを使うかを変えられるように実装
- さらに，アプリケーション（システムサービスも含む）もZxxで記述し，カーネルやコンフィギュレーション記述とまとめてコンパイルするアプローチも考えられる（未実装）



修正スライド

実現したRTOSの性能評価

プログラムサイズの比較（1）

- ASP3カーネル（ターゲット：CT11MPCore）の標準サンプルプログラム（sample1）
- カーネル本体とコンフィギュレーション記述を個別にコンパイルした場合

† カーネルのみClangでコンパイル

ビルド条件	GCC (NDEBUG)	Zig (ReleaseSafe)	Zig (ReleaseFast)	Zig (ReleaseSmall)
プログラムサイズ	30,680B	41,497B	35,397B	34,793B

ビルド条件	Clang † (NDEBUG)	Zen (ReleaseSafe)	Zen (ReleaseFast)	Zen (ReleaseSmall)
プログラムサイズ	31,748B	41,495B	35,425B	34,821B

- ReleaseSafeは，35%程度大きい（やむをえない）
- ReleaseFast/Smallは，15%程度大きい

修正スライド

プログラムサイズの比較(2)

- ASP3カーネル(ターゲット:CT11MPCore)の標準サンプルプログラム(sample1)
- カーネル本体とコンフィギュレーション記述をまとめてコンパイルした場合

† カーネルのみClangでコンパイル

ビルド条件	GCC (NDEBUG)	Zig (ReleaseSafe)	Zig (ReleaseFast)	Zig (ReleaseSmall)
プログラムサイズ	30,680B	39,185B	34,549B	34,005B
ビルド条件	**Clang † (NDEBUG)**	**Zen (ReleaseSafe)**	**Zen (ReleaseFast)**	**Zen (ReleaseSmall)**
プログラムサイズ	31,748B	39,191B	34,569B	34,025B

- 個別にコンパイルした場合と比べて，3～5%程度小さい
- コードサイズが大きくなる主な原因は，関数を積極的にインライン展開することと思われる

実行時間の比較

- ASP3カーネル（ターゲット：GR-PEACH）
- タスクを起動するサービスコール（act_tsk）によるタスク切換え時間

ビルド条件	C (NDEBUG)	Zig (ReleaseSafe)	Zig (ReleaseFast)	Zig (ReleaseSmall)
実行時間	1.6μsec	1.4μsec	1.3μsec	1.4μsec

ビルド条件	Zen (ReleaseSafe)	Zen (ReleaseFast)	Zen (ReleaseSmall)
実行時間	1.4μsec	1.3μsec	1.4μsec

- C言語実装と比べて，15～20%程度高速
  - 実行時間が短くなる主な原因は，関数を積極的にインライン展開することと思われる
- ReleaseFastとReleaseSmallの差は小さい
  - もっと差をつけても良いように思う

Zig言語およびZen言語の現時点での評価

記述性の評価

- 記述が，小さくはならないが，きれいにはなっており，安全性や保守性は向上していると思われる
  - エラー処理がきれいに記述できる（gotoが不要に）
  - ジェネリック型とジェネリック関数は強力かつシンプル
  - バリエーションの記述方法には工夫とルール化が必要
  - 低レベルのビルトイン関数が便利
- C言語（.hファイル，.cファイル），Ruby言語，静的APIの記述を，ほぼすべてZxxに統一できた

性能（実行時間，メモリ使用量）の評価

- 実行時間は短くなっているが，コードサイズが大きくなっているのは気になる
  - ReleaseSmallは，もっとコードサイズを小さくする側に振るべきと思われる

Zxxの課題

- まだまだ未完成の言語
  - 未実装の機能，制限事項も多数残っている
  - 言語処理系の不具合も多い
  - 安全性と性能に関する概念と方針の整理 … 後で議論
  - 分割コンパイルのサポート
- ドキュメントの不足・不備（未完成部分）
  - 不具合なのか仕様なのか判断できない
  - Zenのドキュメントは読みやすい
- エラーメッセージが不親切
  - エラーメッセージの出方（特に出る順序）が違う
  - 正しく動かすまでに試行錯誤が必要
- ターゲットプロセッサが限られる
  - llvmが対応していることが必要条件

ZigとZenで書き換えが必要だった主な箇所

- ファイルの拡張子の変更(“`.zig`” → “`.zen`”)
- `mut` 指定
  - Zenでは，ポインタ はconst(指している先に書き込めない)が標準．Zigにおける`const`は削除して，そうでないポインタに `mut` を付けることが必要(そう単純ではない)
- 構造体の外部から参照されるフィールドに`pub`指定を追加
- ビルトイン関数名の違いへの対応
  - `@as`(Zig) と `@to`(Zen)
- 標準ライブラリの仕様の違い(多数あり)への対応
- Zenではグローバル変数を，`undefined`以外に初期化することが必要
- コンパイラの不具合の回避策
  - 多くの不具合は，両言語のコンパイラに共通しており，回避策も共通だったりする

議論 – プログラムの安全性と性能

- RTOSの特性と設計
  - 性能が重視される
  - 機能間の結びつきが密接
  - その結果，安全でない設計になっている
- とは言え，安全でないコードを特定・局所化し，その部分を集中的に検証したい
  - C言語ではMISRA-Cからの逸脱
  - Rust（等）ではunsafeブロック
  - *Zxx*では，現時点では，安全でない構文要素が明示されていない（考えればだいたいわかるが）
    - **!** まずは，安全性に対する概念と方針の整理が必要と思われる

プログラミング言語の安全性

「プログラミング言語の安全性」とは？

- 様々な定義がある
- ここでは，「安全なプログラミング言語で記述したプログラムは，ある種の不正動作をしない」と定義する
  - 定義の後半は，「ある種の不正動作から保護されている」と言い換えても良い

メモリ安全性

- バッファオーバフローやダングリングポインタなどの，RAMアクセス時に発生するバグやセキュリティホールなどから保護されている状態（Wikipediaより）
  - 上の定義の「不正動作」が「RAMアクセス時に発生するバグやセキュリティホールなど」ということ
- より厳密には，「不正動作」が「定義されていないメモリアクセス」（この厳密な定義が難しい）という感じになる

配列アクセスのメモリ安全性（範囲検査）を例に考える

- C，C++の場合
  - []による配列アクセスでは，範囲を超えたアクセスは検出されず，安全でない
- Java（安全な言語とされている）の場合
  - []による配列アクセスでは，範囲を超えたアクセスを行うと，実行時に検出されて例外が発生する（回復可能）
- Rust （安全な言語とされている）の場合
  - []による配列アクセスでは，範囲を超えたアクセスを行うと，実行時に検出されてpanicになる（回復不可能）
  - getによる配列アクセスはオプショナル型を返す
    - ✓ 範囲を超えたアクセスを行うと，実行時に検出されてNoneが返る
    - ✓ オプショナル型を返すので，例外処理を書くことを強制される（例外はcatchしないことができる）

配列アクセスのメモリ安全性（範囲検査）を例に考える – 続き

- “zig zen”より
  - *Runtime crashes are better than bugs.*
  - *Compile errors are better than runtime crashes.*
- 理想は，コンパイル時に範囲を超えたアクセスを行う可能性があることが検出され，コンパイルエラーになることだが，これは難しい
- *Zxx*の場合
  - []による配列アクセスでは，範囲を超えたアクセスを行うと，実行時に検出されpanicになる
  - Rustと同様のgetによる配列アクセス関数を用意することで，安全性を向上できると思われる

配列アクセスのメモリ安全性(範囲検査)を例に考える – 続き

```
// 5つの符号なし整数からなる配列
const a = [5]u32{ 1, 2, 3, 4, 5 };

fn get(array: []u32, index: usize) ?u32 {
    return if (index < array.len) array[index] else null;
}

test "" {
    var x: u32 = undefined;

    if (get(&a, 1)) |x1| {
        x = x1;
    }
    else {
        // ここに例外処理を書く
    }
    std.debug.warn("a[1] = {}¥n", .{ x });
}
```

オプショナル型
u32またはnullを返す

get(&a,1)をそのままxに代入することはできない

返り値がnullでない場合にifが成立し，x1にu32型の値が入る

x1はu32型なので，代入できる

「不正動作から保護されている」の4つのレベル

(1) 静的安全

(1-1) 不正動作をする可能性のあるプログラムは，コンパイラがエラーとする

静的安全に分類してみた

(1-2) プログラムの不正動作は実行時に（不正動作する前に）検出され，不正動作をしないようにプログラムを書くことが強制される

Zxxにはない

(2) 実行時安全

(2-1) プログラムの不正動作は，実行時に（不正動作する前に）検出されて，回復可能な例外を発生させる

(2-2) プログラムの不正動作は，実行時に（不正動作する前に）検出されて，回復不可能なpanicになる

配列アクセスのメモリ安全性（範囲検査）の例では…

- C，C++の場合
  - []による配列アクセスは，安全でない
- Java（安全な言語とされている）の場合
  - []による配列アクセスは，実行時安全 (2-1)
- Rust （安全な言語とされている）の場合
  - []による配列アクセスは，実行時安全 (2-2)
  - getによる配列アクセスは，静的安全 (1-2)
- *Zxx*の場合
  - []による配列アクセスは，ReleaseSafeでは，実行時安全 (2-2)
  - Rustと同等のgetによる配列アクセスを用意することで，静的安全 (1-2) にできる

今後の取り組みとソースコードへのアクセス

残っている開発項目

- すべてを（アプリケーションも）まとめてコンパイルする
- *Zxx*のビルドシステムの利用
- Zenの拡張機能（インタフェース）の活用
- ソースコードの整理，ドキュメントの作成

ソースコードへのアクセス

- TOPPERS/ASP3のZigによる実装は，Github上で公開中
  - ☞ https://github.com/toppers/asp3_in_zig
- Zenによる実装も，公開を検討中

その後の取り組み（検討中）

- Zig言語やZen言語の将来性を見極めて，さらに発展させることも考えたい
  - ご意見やニーズをお聞かせください

TOPPERSについてより詳しく知りたい方のために

NEP／enPiT-Proの講座「RTOSの内部構造」

☞ https://www.nces.i.nagoya-u.ac.jp/NEP/courses/2020/T01.html

- 9月12日（土）・19日（土），オンライン開催
- TOPPERS/ASP3カーネルのソースコード（C言語版です）を解説
  - Zxx版のTOPPERS/ASP3カーネルのソースコードを読んでみたい人には役立つはず

TOPPERS開発者会議

- 2020年9月27日（日）〜28日（月），オンライン開催
- TOPPERS会員以外も参加可能

謝辞

- TOPPERS/ASP3カーネルをZigおよびZenで実装するにあたり，Zigについていろいろ教えてくれた河田智明君＠高田研究室に感謝します
  - Zig処理系の不具合の切り分け
  - 不具合の回避策の提示
- Zigの開発者のAndrew Kelley氏らにも感謝します
- Zenを使用するにあたり，サポートいただいた帝都久利寿氏とコネクトフリーの皆様に感謝します
  - Zen処理系の不具合の修正
  - 言語仕様（制限事項）の説明，回避策の提示

付録

Zigの言語仕様のチラ見せ：ポインタとその仲間

C言語のポインタの問題点

- 1つの要素を指しているのか，複数の要素（配列）を指しているのかわからかない
- 複数の要素を指している場合，いくつの要素があるのかわからない
- NULL（データを指していない）の場合もある
- その結果，危険な操作ができてしまう

Zigでは…

`*T`：単一のT型のデータへのポインタ

- ポインタの指す先は `ptr.*` でアクセスする
- ポインタへの加減算は禁止
- 必ず1つのデータを指しており，データを指していない状態（null）も許されない

Zigでは… – 続き

- `*[N]T` : N個のT型の配列へのポインタ
  - N はコンパイル時に決まる値
  - ポインタの指す先は `ptr[i]` でアクセスする
  - ReleaseSafeでは，範囲を超えたアクセスが検出される
  - 実は，`[N]T` が配列で，それへのポインタ
- `[]T` : 可変個のT型の配列へのポインタ
  - 実行時に要素数も記憶
    - ✓C言語であれば，先頭番地と要素数でできた構造体で実現するようなもの
  - ポインタの指す先は `ptr[i]` でアクセスする
  - `ptr.len` で要素数を取り出すことができる
  - ReleaseSafeでは，範囲を超えたアクセスが検出される
  - 「スライス」と呼ばれており，実は，ポインタとは別物

Zigでは… – 続き

- `[*]T`：要素数がわからないT型の配列へのポインタ
  - ポインタの指す先は `ptr[i]` でアクセスする
  - ポインタへの加減算も可能
  - 範囲を超えたことが検出できないため，危険なポインタ
- `?*T` : nullが許される単一のT型のデータへのポインタ
  - ポインタの指す先は，`ptr.?.*` でアクセスする
    - ✓ `?` は `ptr` がnullでない場合にのみ許される
    - ✓ReleaseSafeでは，`ptr` がnullの場合が検出される
  - 先頭の `?` がnullが許されることを意味し，任意の型の前につけることができる（オプショナル型）
  - nullでないことが保証される（`?` を使わない）記法もある
    - ✓nullでない場合に成立するif文やwhile文

Zigでは… – 続き

- `[*c]T`：C言語と互換のポインタ
  - 1つの要素を指しているか，複数の要素（配列）を指しているか，データを指していない（null）か，わからないポインタ
  - （もちろん）危険なポインタ
  - C言語と共存させる場合に用いる
- `[*:x]T`：番兵で終端されたポインタ
  - `x` は番兵を示す
  - 典型的には，ヌル終端文字列
- その他に…
  - const（Zenでは逆の意味のmut），volatile，アラインメント，allowzero