TLSF アロケータと mruby のマルチVM対応

TECS ジェネレータV1.5 強化機能

2017.6.11

TOPPERS プロジェクト TECS WG

TLSF アロケータとは

- メモリをアロケートする、つまり malloc, free を提供するもの
- TLSF : Two Level Segregated Fit --- 2レベルで最適な空き領域を探す
  - フラグメント化の防止．極力、連続した空き領域を温存する
  - 一定時間で最適な空き領域を探すことができる
  - 詳しくは、以下のページを参照してください (英語)

    http://www.gii.upv.es/tlsf/main/docs
- 組込みシステムで使いやすい実装
  - 固定のメモリ領域から割り付けるので、OS や特別なライブラリに依存しない
    - OS からメモリを獲得するためのシステムコール (sbrk等) を呼ばない

tTLSFMalloc セルタイプ

- TLSF アロケータを TECS コンポーネント化したもの
- 特定のライブラリや OS に依存しない実装のため、非常に汎用性が高い
- 排他制御をおこなっていない
- TLSF の本体は付属していない (前出の HP からダウンロード)
- ヒープは BSS 領域に配置される (大きな配列が取られる)

```
--- tTLSFMalloc.cdl ---
import( <sMalloc.cdl> );
celltype tTLSFMalloc {
   [inline]   entry  sMalloc  eMalloc;       // 受け口（シグニチャ説明は、次ページ)
   attr {
            size_t  memoryPoolSize;          // ヒープ容量   (Bytes)
   };
   var {
      [size_is( memoryPoolSize/ 8 )]
            uint64_t   *pool;                // uint64_t のアライメントで割り付ける
   };
};
```

sMalloc シグニチャ

- malloc, free とそのバリエーションを提供
- ヒープの初期化も含む
- 排他制御されるかどうかは、セルタイプ実装による
  - TLSF アロケータは排他制御しない

```
--- sMalloc.cdl ---
import_C( "t_stddef.h" );
[deviate]
signature sMalloc {
   int    initializeMemoryPool(void);                          // ヒープの初期化
   void   *calloc( [in]size_t nelem, [in]size_t elem_size );
   void   *malloc( [in]size_t size );
   void   *realloc( [in]const void *ptr, [in]size_t new_size );
   void   free( [in]const void *ptr );
};
```

mruby マルチ VM 対応

- TECS ジェネレータ V1.5 では、tMruby セルタイプを分解して、複合セルタイプ (composite) 化
  - TECS ジェネレータ V1.4 までの実装では、単一のセルタイプ（celltype）で、malloc, free をライブラリとしてリンクしていた
  - ユーザーコードは、無変更でバージョンアップ可能

```
--- tMruby.cdl (抜粋) ---
composite tMruby{
   entry sTaskBody eMrubyBody;                  // タスクから結合 (タスクは、再起動不可)
   [optional]  call sInitializeBridge cInit;    // ブリッジで生成された初期化コンポーネントを結合(忘れるな!)
   attr{
     [omit]char_t *mrubyFile;                   // .rb ファイルへのパス (空白区切りで複数指定可)
                                                //  factory により .mrb に変換されてリンクされる
     size_t memoryPoolSize = 1024 * 1024;       // ヒープ容量のデフォルト 1MB
   };
   /* 内部セル */
   cell tMrubyVM MrubyVM{     …     };          //  VM 本体
   cell tMrubyTaskBody MrubyTaskBody {  …  };   // タスクのメイン処理 (コンポーネント分離)
   cell tTLSFMalloc TLSFMalloc {  …  };         // TLSF アロケータ (コンポーネント分離)
};
```

mruby マルチ VM 対応版の特徴

- ヒープを VM ごとに独立化
  - ヒープの排他制御が不要
  - ヒープを使い切る事態になった場合でも、すべての VM が GC 停止することはない

- マルチ VM でのプログラムの注意点
  - グローバル変数が、グローバルではなくなる
    - VM ごとに独立したメモリ空間となるので、他の VM の変数には、一切アクセスできない
    - Windows 上でいくつもの ruby.exe を起動したのと同じ状況と考えてください
    - （ヒープを独立化させたこととは無関係）