【2F】Delphi/C++チュートリアル セッション

「Delphiでキカイを制御する」

アプリケーションの設計とテクニック

株式会社イマジオム 代表取締役
高木 太郎

EMBARCADERO
TECHNOLOGIES.

はじめに

この講演の内容

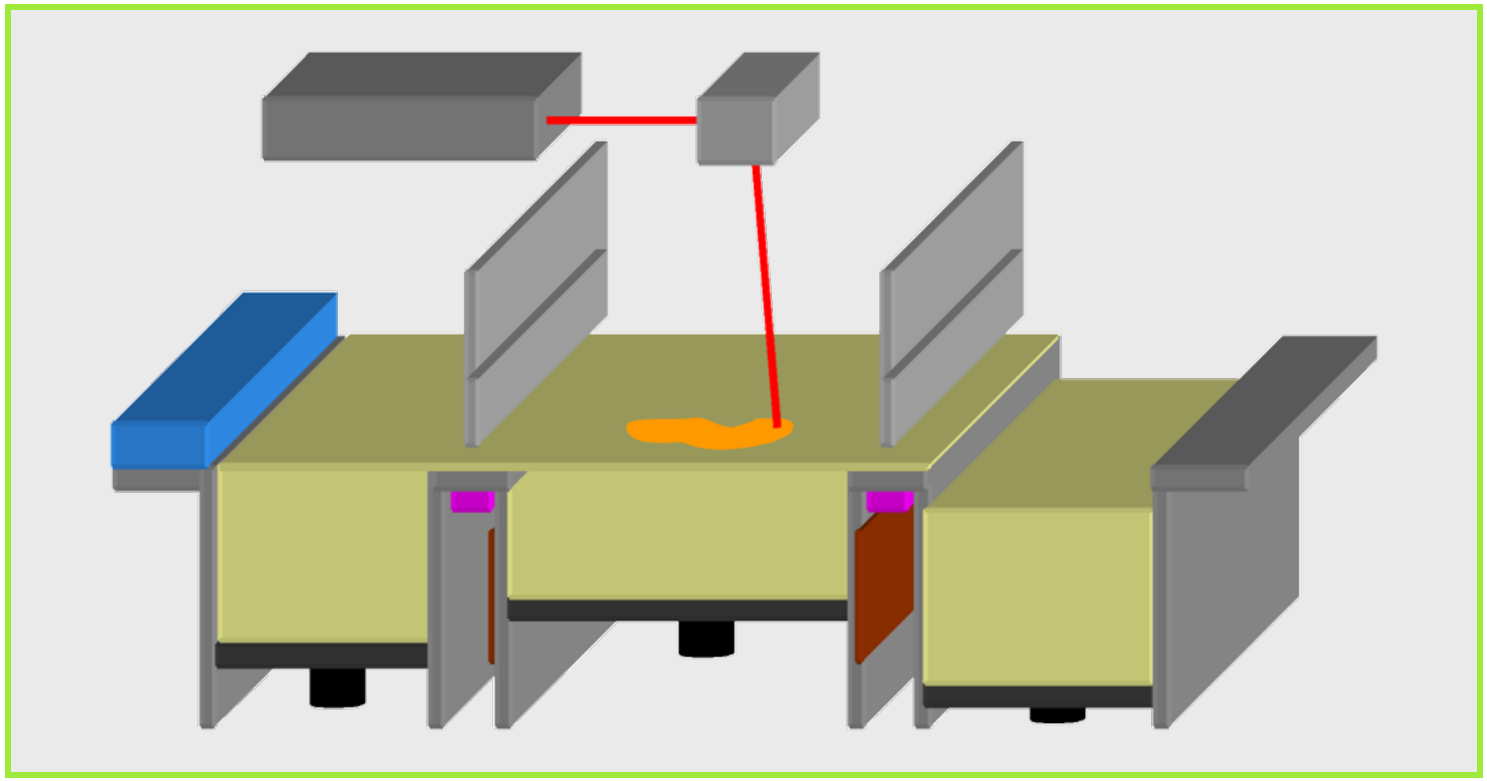
- 制御プログラムというもの
- 制御プログラム 設計のポイント
- 制御プログラム 実装のテクニック

どんなものを考えているのか？

- 例：3次元プリンタ

ここに入っているPCが
システム全体を制御

3次元プリンタ 原理



- レーザで材料粉末を層ごとに焼き固め、積み重ねて「実物」を作る

3次元プリンタ 造形物

- データさえ用意すれば、3次元形状を自由に作ることができる

制御プログラムの「データ」画面

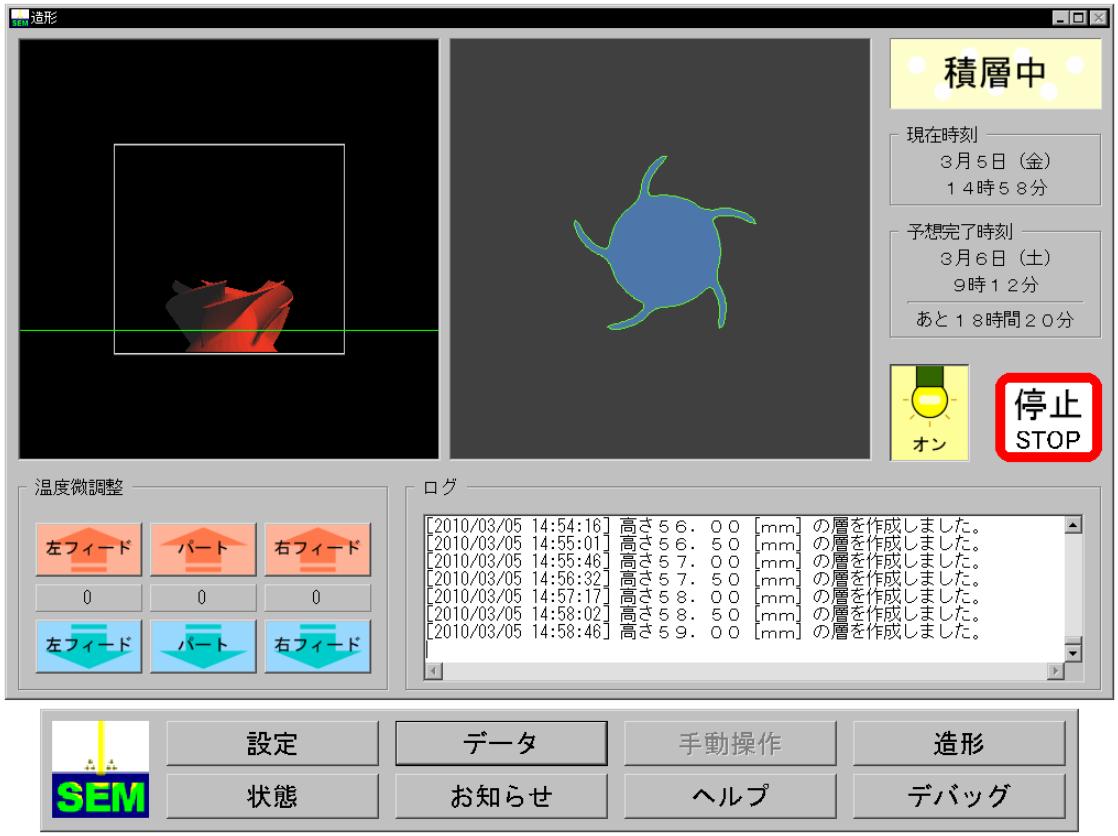
- ３次元データをロードして配置

制御プログラムの「手動操作」画面

- 装置を準備して3次元プリントを開始

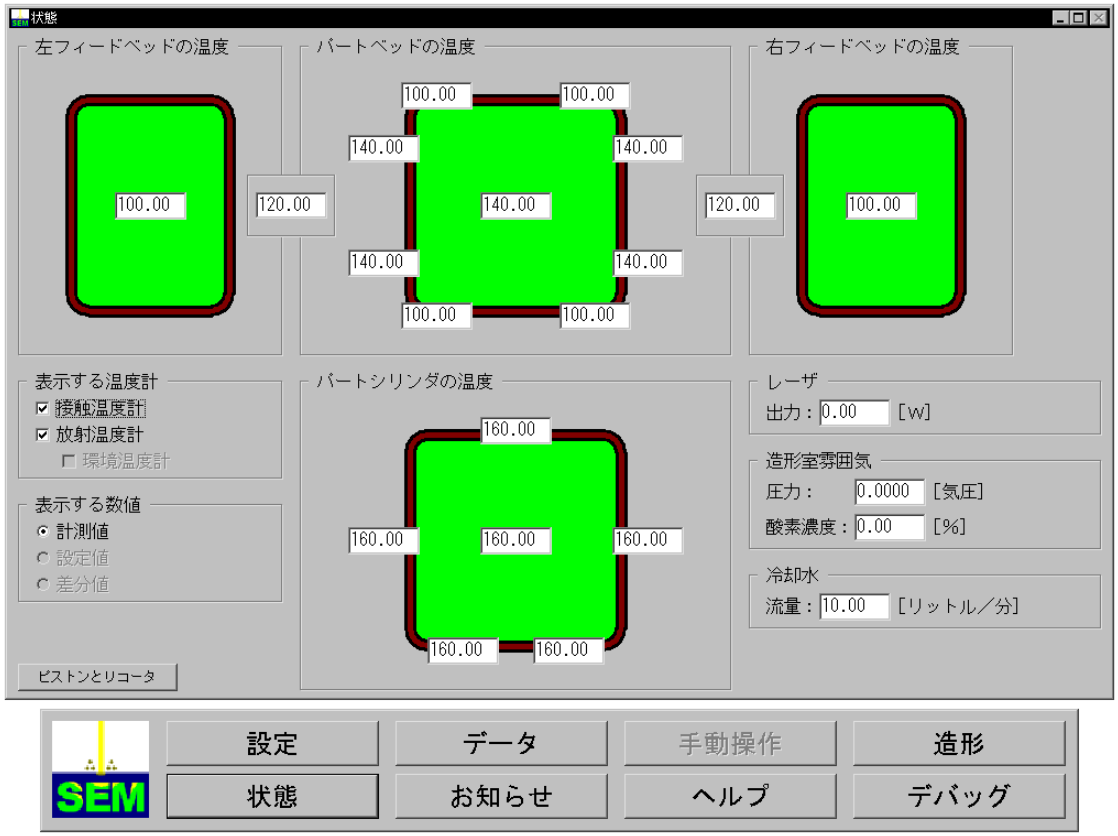
制御プログラムの「造形」画面

- ３次元プリントの進捗状況を表示



制御プログラムの「状態」画面

- 装置の状態をリアルタイムに監視



メインテーマ

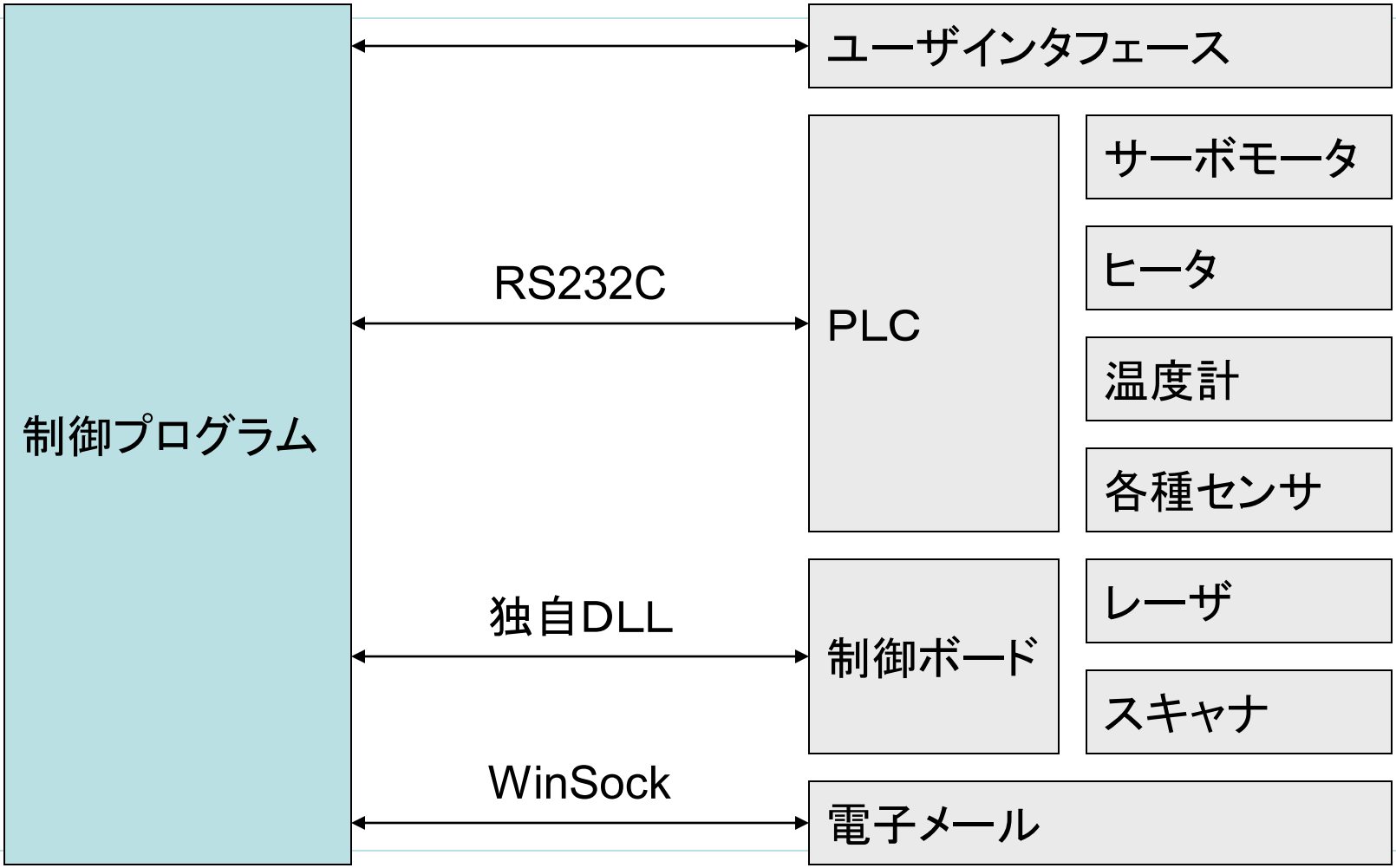
- このような制御プログラムを、どのように設計・実装すればいいのか？

制御プログラムというもの

「普通のプログラム」と何が違う？

- サードパーティ製品を使わないといけない
- 多くの動作を同時に行わないといけない
- 頻繁な仕様変更に対応しないといけない
- ユーザにはいつでも応答しないといけない

サードパーティ製品を使わないといけない



多くの動作を同時に行わないといけない

- 3次元データ配置動作
- 断面形状計算動作
- 光学系制御動作
- 機械系制御動作
- 環境監視・制御動作
- 電子メール送信動作

- しかもリソースの取り合いが起きる
  - 3次元データ
  - 断面形状
  - 画面表示
  - PLC
  - 光学系
  - 電子メール

頻繁な仕様変更に対応しないといけない

- 下層側（ハードウェア側）が変わることが多く、修正が大変

使用部品の変更

寸法の変更

画面の変更

データ形式の変更

制御方法の変更

ユーザにはいつでも応答しないといけない

- どんな時でも、非常停止だけはさせたい

制御プログラム設計のポイント

最初に結論を書きます

- マルチスレッドプログラムにする
- ユーザインタフェース以外の「すべて」を
  ワーカスレッドにやらせる

本当か？

- マルチスレッドが必要なのはわかる
- しかし「すべて」をワーカスレッドにやらせなくてもいいのでは？

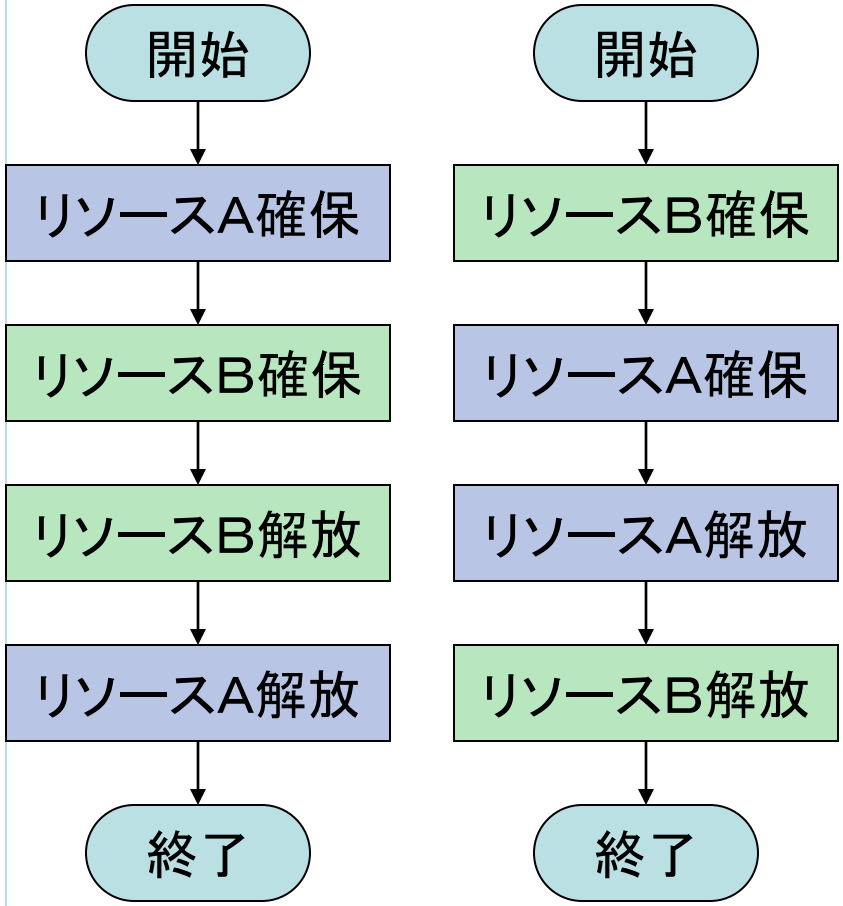
やってみると、ハングアップが頻発することがわかる

マルチスレッドと排他制御

- マルチスレッド＝複数の実行単位（スレッド）を同時に実行させる方法
- 排他的リソース＝複数のスレッドが同時に使ってはいけないリソース
  - リソース＝コンピュータが扱うあらゆる「もの」。コードやデータ、ハードウェアなど
- 排他処理＝排他的リソースが、同時に複数のスレッドから使われないようにする方法。TCriticalSectionを使うのが一般的

```
try
  CriticalSection.Enter;
  UseExclusiveResource(...);
finally
  CriticalSection.Leave;
end;
```

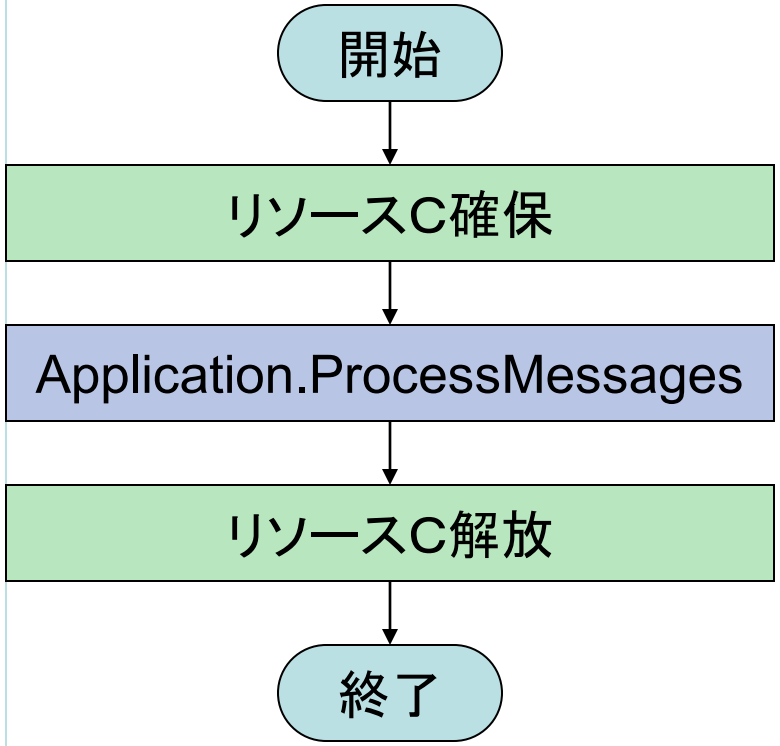
デッドロック



- 異なる順番でリソースを確保する複数のスレッドは、デッドロックを起こすことがある
- つまりデッドロックを避けるには、リソース順位を決め、いつも同じ順番で確保すればよい

これは、よく言われること

リソースが一つでもデッドロック？

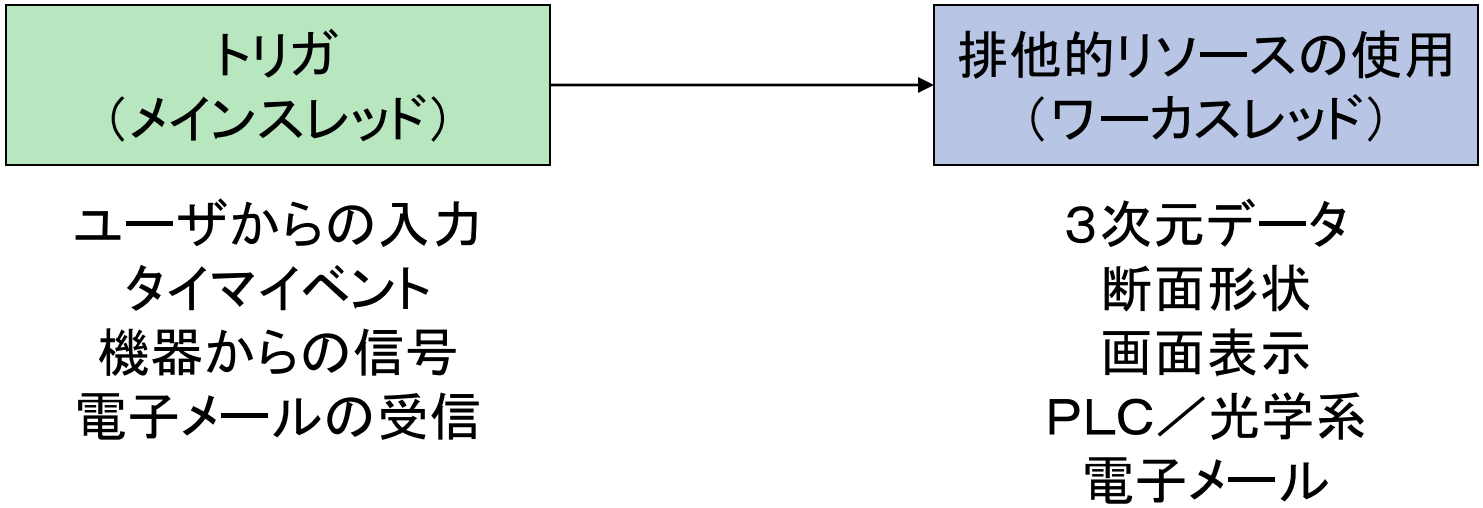


- Application.ProcessMessagesの処理するメッセージが、リソースCを使おうとするとどうなるか？
- リソースCを使わないメッセージを選んで処理するのは困難。どんなメッセージが来るかわからない

メインスレッドでは、排他的なリソースを使ってはいけないことがわかる

それではどうするか？

- 排他的リソースは、すべてワーカスレッドで使用する。
- メインスレッドでのトリガによって排他的リソースを使う処理を実行する場合、ワーカスレッドに処理をやらせる
- ワーカスレッドでは、通常の「リソース順位」を使った排他制御を行う



制御プログラム実装のテクニック

今回紹介するテクニック

- 別のスレッドに実行させる
  - 実行待ち行列クラスTCommandQueue
  - 引数を渡すことのできるSynchronize
  - ワーカスレッドで実行させるAsynchronize
- コンテキストスイッチの高速化

別のスレッドに実行させる

	メインスレッドに実行させる	ワーカスレッドに実行させる
実行終了を待たない（ノンブロッキング）	TTimerCommand-Queue	TThreadCommand-Queue
実行終了を待つ（ブロッキング）	Synchronize	Asynchronize

実行待ち行列クラスTCommandQueue

```
uses CommandQueues;

type TSomeForm=
        class(TForm)
          private
            CommandQueue:TTimerCommandQueue;
            procedure DoSomething(CommandParams:TCommandParams);
          public
            procedure Execute;
        end;
```

- メソッドを非同期で順番に実行させる

```
procedure TSomeForm.Execute;
begin
  SomeParams:=TSomeParams.Create;
  SomeParams.Name:='高木太郎';
  SomeParams.Age:=43;
  CommandQueue.Add(DoSomething,SomeParams);
end;
```

TCommandParamsで引数を渡す

- 引数をTCommandParamsオブジェクトでメソッドに渡す

```
type TSomeParams=
        class(TCommandParams)
          public
            Name:string;
            Age:LongInt;
        end;

procedure TSomeForm.DoSomething;
begin
  WriteLn('名前は ',TSomeParams(CommandParams).Name);
  WriteLn('年齢は ',TSomeParams(CommandParams).Age);
end;
```

タイマキューとスレッドキュー

- 実行方法の異なる(使い方は同じ)二つの実行待ち行列を用意
- TTimerCommandQueue:メインスレッドのメッセージループで実行
- TThreadCommandQueue:一つのワーカスレッドで順番に実行

```
type TCommandQueue=
        class(TObject)
          public
            function CommandCount:LongInt;
            procedure Add(Method:TCommandMethod;  Params:TCommandParams);
            procedure Delete(CommandIndex:LongInt);
            property OnError:TCommandErrorEvent;
        end;

type TTimerCommandQueue=class(TCommandQueue)

type TThreadCommandQueue=class(TCommandQueue)
```

引数を渡すことのできるSynchronize

- TThread.Synchronizeは引数を持っていない。引数を渡そうとすると、クラスのメンバ変数やグローバル変数を使わなければならない
- 引数を渡すことのできるSynchronizeを作っておく

```
  uses Synchronizers;

begin
  SomeParams:=TSomeParams.Create;
  SomeParams.Name:='高木太郎';
  SomeParams.Age:=43;
  Synchronize(DoSomething,SomeParams);
end;
```

ワーカスレッドで実行させるAsynchronize

- ワーカスレッドからメインスレッドに処理をさせる場合、Synchronizeが使える。しかしメインスレッドからワーカスレッドに処理をさせる簡便な方法は用意されていない

```
  uses Synchronizers;

begin
  SomeParams:=TSomeParams.Create;
  SomeParams.Name:='高木太郎';
  SomeParams.Age:=43;
  Asynchronize(DoSomething,SomeParams);
end;
```

コンテキストスイッチの高速化

- コンテキストスイッチ＝実行スレッドを切り替えること
- 通常のコンテキストスイッチの間隔は20ミリ秒ほど。これを短縮することでシステムの応答が向上する
- グローバル設定なので、高速応答が不要になったら必ずもとに戻す

```
  uses MMSystem;

procedure FormCreate;
begin
  TimeBeginPeriod(1);
end;

procedure FormDestroy;
begin
  TimeEndPeriod(1);
end;
```

まとめ

制御プログラムを設計するには（1）

- プログラムが同時に行わなければならない処理をリストアップ
- それぞれの処理が使うリソースをリストアップ
- 複数の処理に使われる「排他的」リソースをリストアップ
- 排他的リソースを使う処理は、すべてワーカスレッドに実行させる（Asynchronize）
  - VCLを使う処理は、メインスレッドに実行させる（Synchronize）

制御プログラムを設計するには(2)

- 排他的リソースの依存関係をできるだけ排除
  - 「リソースAを使っていなければ、リソースBが使えない」状況を避けるよう、実行タイミングをずらす(TCommandQueue)
- それでも依存関係が残る場合、排他的リソースに順位を決める
- すべての排他的リソースをクリティカルセクションで保護
- メッセージ応答を高める必要がある場合、コンテキストスイッチの間隔を短くする(TimeBeginPeriod)

まとめ（1）

- 制御プログラムは、ユーザ操作が主体のプログラムとは異なる性格を持っている。開発の際には、その性格を意識した設計が求められる
- 特に処理実行のトリガが複数ある場合、デッドロックや再入には特に注意する必要がある。それらの防止に、排他的リソースを使う処理をすべてワーカスレッドに実行させる方法が効果的

まとめ（2）

- 多くのスレッドを使用する制御プログラムの実装に役立つテクニックを紹介した。Delphiユニット（ソースコード付き）も、近く弊社ホームページ（http://www.imageom.co.jp/）で公開するので、併せて活用されたい

Q & A