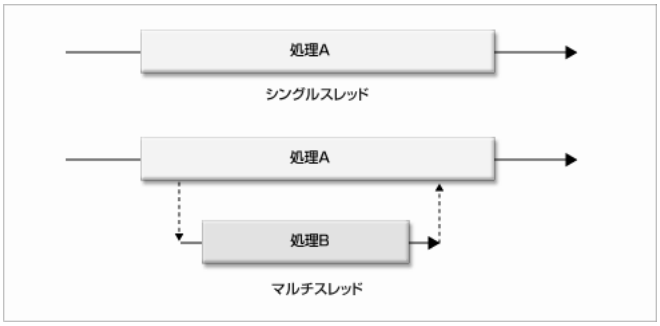
Builder におけるマルチスレッドプログラミング

マルチスレッドとは、1 つのアプリケーションソフトがスレッドと呼ばれる処理単位を複数生成し、並行して複数の処理を行なうことを言います。



シングルスレッドとマルチスレッドの処理イメージ

マルチスレッドでは、1 つのアプリケーションにおいて同時に複数の処理が実行可能です。

重い処理を行うときにマルチスレッドが大変重要になります。

プログラムは 1 つの処理が実行されるとその処理が終わるまで次の処理に移ることができません。なので、重い処理を行った場合そこでプログラムが固まってしまったようになります。これを防ぐには新しいスレッドを生成し、そのスレッドで処理を行えば元スレッドが固まることを防ぐことが出来ます。

C 言語でのマルチタスクでは Pthread や mutex、CreateThread、beginthreadex など様々な方法がありますが、全てそう簡単にマスターできるものではありません、しかし C++Builder にはもっと簡単にマルチスレッドプログラミングをすることができるようになっています。

- TThread

1. C++Builder のメニューから「ファイル」→「新規作成」を選択
2. 「スレッドオブジェクト」を選択し「OK」
3. クラス名を入力し「OK」

これでスレッドクラスのひな形が作成されます。

では例として NewThread というスレッドクラスを作ります。クラス名に「NewThread」を入れて OK を押します。すると「Unit2.cpp」と「Unit2.h」が作られます。これで TThread を継承した NewThread クラスが作成されました。

作成した NewThread には元から①「__fastcall NewThread(bool CreateSuspended);」と②「void __fastcall Execute();」という関数（メソッド）が宣言されています。

①はこのクラスのコンストラクタです。ここで初期化などを行います。次に②はこのスレッドで実行される関数です。ここに処理を書きましょう。

コンストラクタの引数「bool CreateSuspended」が false の場合、Execute メソッドがすぐに呼び出されます。true の場合、Resume メソッドが呼ばれるまで Execute メソッドは呼び出されません。ここで

false にした場合、コンストラクタの処理中に Execute メソッドが呼び

出されてしまうので、true にして自分で Resume メソッドを呼び出した方が良いと言われています。

では、参考として新しく作ったスレッド内で無限ループを行い、その中の変数の値を調べるプログラムを作ってみます。

まず NewThread にメンバ変数を宣言します。

```
public:
    int num;
```

そしてコンストラクタで初期化します。

```
__fastcall NewThread::NewThread(bool CreateSuspended)
    : TThread(CreateSuspended){
    num = 0;
    FreeOnTerminate = true;
}
```

「FreeOnTerminate」とは、これが true の時スレッド終了時にスレッドオブジェクトを自動的に破棄します。false の時スレッドオブジェクトはアプリケーションコードによって明示的に破棄されなければなりません。

次に Execute メソッドでメンバ変数 num を無限に足していきます。

```
void __fastcall NewThread::Execute()
{   // TODO : スレッドとして実行したいコードをここに記述 */
    while(1) num++;
}
```

もしこの「while(1) num++;」と言う処理を元スレッドで行った場合、無限ループによりプログラムが固まってしまいます。しかし、ここではマルチスレッドによりプログラムが固まることはありません。

では元スレッド（TForm1）内でスレッドのクラスを宣言します

```
public:         // ユーザー宣言
    NewThread *thread;
```

そして、TForm1 のコンストラクタでクラスを new し、Resume メソッド呼び出します。

```
__fastcall TForm1::TForm1(TComponent* Owner): TForm(Owner){
    thread = new NewThread(true);
    thread->Resume();
}
```

これにより新しいスレッドが作成されResumeメソッドにより処理が実行されました。ここでプログラムがちゃんと動いているかを確認するためにボタンでNewThreadクラスのnumの動きを見てみましょう。

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    Form1->Caption = thread->num;
}
```

これによりButton1をクリックするとNewThreadのスレッドで処理されているnumが確かに処理されていることが確認できます。

このようにマルチスレッドプログラミングによりシングルスレッドでは処理が終わらず固まってしまうプログラムを別のスレッドで行うことでプログラム全体が固まってしまうことを防ぐことができます。

これをゲームなどで使う場合は画像、音声、動画の読み込み、ファイルの書き出しなど時間がかかる処理を別のスレッドで行うことで快適なゲームを作ることができます。

別スレッドで画像を読み込む処理場合、そのスレッドに変数を渡さなければ行けません。その場合、渡すための関数を作ったりコンストラクタの引数を自分で加えたりすることで出来ます。

【サンプルプログラム（NewThreadクラス)】

```
public:
    __fastcall NewThread(bool CreateSuspended,Graphics::TBitmap *_
Image,String _FileName);
    Graphics::TBitmap *Image;
    String FileName;
```

```
__fastcall NewThread::NewThread(bool CreateSuspended,Graphics::TBi
tmap *_Image,String _FileName)
    : TThread(CreateSuspended)
{
    FreeOnTerminate = true;
    Image = _Image;
    FileName = _FileName;
}
//---------------------------------------------------------------
void __fastcall NewThread::Execute()
{   // TODO : スレッドとして実行したいコードをここに記述 */
    Image->LoadFromFile(FileName);
}
```