スレッドローカルデータ

Maple 17 では、変数にアクセスする各スレッドごとに異なる値を格納できる変数を宣言できるようになりました。これにより、別のスレッドによって状態が変更される心配をせずに、状態を保持する必要があるアルゴリズムを実装できます。このようなアルゴリズムのスレッドセーフバージョンを簡単に実装できるようになりました。

一般に、内部の状態を保持したいモジュールは、並列処理では適切に動作しません。以下の例を考えてみましょう。Mapper モジュールは変数 func および data を使用して内部状態を保持します。これらの変数は setMapper ルーチンを使用して設定します。 ModuleApply が呼び出されると、設定したデータで func が マップ されます。

```
> Mapper := module ()
   local func, data;
   export setMapper, ModuleApply;
    setMapper := proc (f::procedure, d::anything)
   func := f; data := d;
   NULL
  end proc;
  ModuleApply := proc (d)
   map(func, d, data)
   end proc
  end module:
```

```
> adder := proc ()
   Mapper:-setMapper(proc (x, y) options operator, arrow; x+y end
  proc, 1);
   Mapper([seq(i, i = 1 .. 10)])
  end proc:
```

```
> adder();
```

$$[2, 3, 4, 5, 6, 7, 8, 9, 10, 11] \quad (1)$$

```
> multer := proc ()
   Mapper:-setMapper(proc (x, y) options operator, arrow; x*y end
  proc, 3);
   Mapper([seq(i, i = 1 .. 10)])
   end proc:
```

```
> multer();
```

$$[3, 6, 9, 12, 15, 18, 21, 24, 27, 30] \quad (2)$$

adder と multer のコマンドを並列に実行した場合、間違った結果が返される可能性があります。以下の処理を複数回実行すると、異なった結果が返され、そのうちのいくつかは間違った結果を示

します

```
> Threads:-Task:-Start(passed, Task = [adder], Task = [multer]);
```

[2, 3, 4, 5, 6, 7, 8, 9, 10, 11], [2, 3, 4, 5, 6, 7, 8, 9, 10, 11] (3)

```
> Threads:-Task:-Start(passed, Task = [adder], Task = [multer]);
```

[2, 3, 4, 5, 6, 7, 8, 9, 10, 11], [3, 6, 9, 12, 15, 18, 21, 24, 27, 30] (4)

```
> Threads:-Task:-Start(passed, Task = [adder], Task = [multer]);
```

[2, 3, 4, 5, 6, 7, 8, 9, 10, 11], [3, 6, 9, 12, 15, 18, 21, 24, 27, 30] (5)

```
> Threads:-Task:-Start(passed, Task = [adder], Task = [multer]);
```

[2, 3, 4, 5, 6, 7, 8, 9, 10, 11], [3, 6, 9, 12, 15, 18, 21, 24, 27, 30] (6)

```
> Threads:-Task:-Start(passed, Task = [adder], Task = [multer]);
```

[2, 3, 4, 5, 6, 7, 8, 9, 10, 11], [3, 6, 9, 12, 15, 18, 21, 24, 27, 30] (7)

func と data の状態変数がスレッド間で共有されているため、不正確な結果が計算されます。1 つのスレッドで func の値が変更されると、別のスレッドにも影響が生じます。Maple 17 では、状態変数を thread_local として宣言することで、 この問題が修正されました。つまり、各スレッドに状態変数の値が独自に保持されます。これにより、1 つのスレッドで値を変更しても、別のスレッドに影響することがなくなりました。

```
> MapperTL := module ()
    local func::thread_local, data::thread_local;
    export setMapper, ModuleApply;
      setMapper := proc (f::procedure, d::anything)
    func := f; data := d;
    NULL
    end proc;
    ModuleApply := proc (d) map(func, d, data)
    end proc
    end module;
```

MapperTL := **module**( ) (8)

local *func*::*thread_local*, *data*::*thread_local*;

export *setMapper*, *ModuleApply*;

end module

```
> adderTL := proc ()
    MapperTL:-setMapper(proc (x, y) options operator, arrow; x+y end
   proc, 1);
    MapperTL([seq(i, i = 1 .. 10)])
    end proc;
```

adderTL := **proc**( ) (9)

MapperTL:-*setMapper*((*x*, *y*) → *x* + *y*, 1); *MapperTL*([*seq*(*i*, *i* = 1 ..10)])

end proc

```
> adderTL();
```

[2, 3, 4, 5, 6, 7, 8, 9, 10, 11] (10)

```
> multerTL := proc ()
   MapperTL:-setMapper(proc (x, y) options operator, arrow; x*y end
  proc, 3);
   MapperTL([seq(i, i = 1 .. 10)])
   end proc;
```

multerTL := **proc**( ) (11)
    MapperTL:-setMapper((*x*, *y*) → *x* * *y*, 3); *MapperTL*([*seq*(*i*, *i* = 1 ..10)])
end proc

```
> multerTL();
```

[3, 6, 9, 12, 15, 18, 21, 24, 27, 30] (12)

スレッドローカルデータを使用したものを実行すると、期待した結果が返されます。

```
> Threads:-Task:-Start(passed, Task = [adderTL], Task = [multerTL]);
```

[2, 3, 4, 5, 6, 7, 8, 9, 10, 11], [3, 6, 9, 12, 15, 18, 21, 24, 27, 30] (13)

```
> Threads:-Task:-Start(passed, Task = [adderTL], Task = [multerTL]);
```

[2, 3, 4, 5, 6, 7, 8, 9, 10, 11], [3, 6, 9, 12, 15, 18, 21, 24, 27, 30] (14)

```
> Threads:-Task:-Start(passed, Task = [adderTL], Task = [multerTL]);
```

[2, 3, 4, 5, 6, 7, 8, 9, 10, 11], [3, 6, 9, 12, 15, 18, 21, 24, 27, 30] (15)

```
> Threads:-Task:-Start(passed, Task = [adderTL], Task = [multerTL]);
```

[2, 3, 4, 5, 6, 7, 8, 9, 10, 11], [3, 6, 9, 12, 15, 18, 21, 24, 27, 30] (16)

```
> Threads:-Task:-Start(passed, Task = [adderTL], Task = [multerTL]);
```

[2, 3, 4, 5, 6, 7, 8, 9, 10, 11], [3, 6, 9, 12, 15, 18, 21, 24, 27, 30] (17)

参照

thread_local