.dll ファイル <arch>\vc<vcversion>\bin\<lib><

ヒント:　タスク・スケジューラーは、起動時とシャットダウン時に多少負荷がかかるため、並列アルゴリズ

3.2.1

セクション 3.2 の `ApplyFoo` クラスとは異なることに注意してください。まず、`operator()` は `const`

4.1.1 コードサンプル

`examples/parallel_while/parallel_preorder` ディレクトリーには、

```
    }
```

次は、mutex の動作の概要です。

x

7 アトミック操作

アトミック操作を使用して、排他制御の問題を回避することができます。スレッドがアトミック操作

```
        oldx = globalx;
```

ロセスまたは別のプロセスのスレッドからの干渉を許可し、タスクを論理スレッドにマッピングすることにより、オーバーサブスクリプションを回避しようとします。

10.4 役立つタスク手法

このセクションでは、スケジューラーを最大限に活用するためのプログラミング手法を説明します。

10.4.1 再帰連鎖反応

スケジューラーは、ツリー構造のタスクグラフで最も良く動作します。これは、"幅優先のスチールと深さ優先の作業" 手法が最も良く適用できるためです。また、ツリー構造のタスクグラフでは、多くのタスクを迅速に生成することができます。例Ｒ骸倪寨辷_胗槁峅辷_胗欞顕ｏ焲樺譂幺ｏ煉る柴欟顕ｏ煮そ柴欟뉘ᵃᵗ 譂Ｂ

```
        } else {
            FibContinuation& c =
                *new( allocate_continuation() ) FibContinuation(sum);
            FibTask& a = *new( c.allocate_child() ) FibTask(n-2,&c.x);
            FibTask& b = *new( c.allocate_child() ) FibTask(n-1,&c.y);
            recycle_as_child_of(c);
            n -= 2;
            sum = &c.x;
            // Set ref_count to "two children".
```

10.5　タスク・スケジューラーのまとめ