C++11:スレッド・ライブラリひとめぐり【補足編:2】

C++

επιστημη[著]

2018/02/15 14:00

ダウンロード↓ サンプルファイル (53.3 KB)

マルチスレッド・アプリケーションでは複数のスレッドが互いに干渉することなくフルスピードでぶん回ってくれるのが最もパフォーマンス高くて理想的。ところが実際にはそうも言ってられません、他のスレッドから邪魔されないよう一時的に他スレッドに待ってもらわなければならないことがしばしば起こります。前回に引き続きC++スレッドサポートライブラリによるスレッドの排他のおはなし。

目次

mutex:スレッド同士が邪魔しない/させないからくり

とっても簡単な関数:add()を用意しました。

list01

```
void add(long* address, long value) {
  *address += value;
}
```

どうということはない、addressが指すlong値にvalueを加えるだけの簡単なお仕事です。4つのスレッドがこの関数を呼んで1つのlong値にそれぞれ何度も+1、-1、+2、-2します。

list02

```
#include <iostream>
#include <thread>
#include <functional>
#include <chrono>
#include <mutex>

// data-race が発生するハダカの足し算
```

```
void add(long* address, long value) {
  *address += value;
}

void run(std::function<void(long*, long)> fun, long* address) {
  using namespace std;
  using namespace std::chrono;

  auto task = [=](int n, long v) { while ( n-- ) fun(address, v); };
  const int N = 100000;
  thread threads[4];
  auto start = high_resolution_clock::now();

  // スレッドを4本起こして
  threads[0] = thread(task, N,  1L);
  threads[1] = thread(task, N, -1L);
  threads[2] = thread(task, N,  2L);
  threads[3] = thread(task, N, -2L);

  // 全スレッド終了を待つ
  for (thread& thr : threads) thr.join();
  auto stop = high_resolution_clock::now();

  cout << duration_cast<milliseconds>(stop - start).count() << "[ms]" << flush;

}

int main() {
  using namespace std;

  long count = 0L;

  cout << "data-race:            ";
  count = 0;
  run(add, &count);
  cout << ", count = " << count << endl;
}
```

4つのスレッドが何度も行う加算と減算は相殺されて最終的には初期値である0に戻るかというとさにあらず、

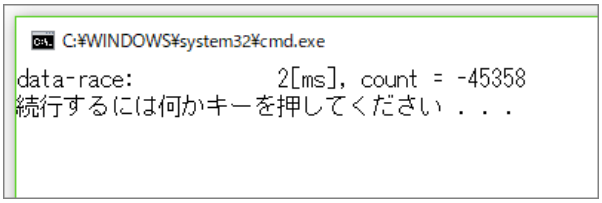

fig01

add()内の処理:*address += valueは、「*addressを読み/valueを加えて/*addressに書く」の3ステップを行います。この3ステップ中に他のスレッドが割り込むことで辻褄が合わなくなるんです。例えば*addressが100のとき、+1するスレッドAと-1するスレッドBがほとんど同時にadd()したとしましょう。両者は*addressを読み、Aは100+1→101、Bは100-1→99を計算します。続いて双方が*addressに書き込むと、どちらが後になるかによって*addressは101か99のいずれかとなり、いずれにせよ期待する100にはなりません。

4 デプロイ自動化をマルチクラウドで! CDツール「Spinnaker」をAWS上で検証してみた【デブサミ2018】
5 超簡単! WPFなどの.NETのアプリからUWPのAPIを使う ~日本語の読み仮名を取得するAPIを題材に
6 デブサミ2018、講演関連資料まとめ
7 オブジェクト指向がわからない! そんなあなたの脳味噌をオブジェクト脳にする準備体操
8 サーバレスアーキテクチャとは何か?~AWS LambdaとAPI Gatewayによる簡単なAPIの実装を試す
9 Pythonは今のうちに習得すべき?『スラスラわかるPython』著者・岩崎さんとPyCon JP理事の寺田さんが語る
10 650円で買えるマイコンボード「Raspberry Pi Zero」でIoTをはじめよう! ~環境構築とレチカのレシピ

新着
記事 ニュース
業務システムでも最新技術を使いたい! ITILの壁を乗り越えるため、機能の一部をマイクロサービス化してクラウドに移行【デブサミ2018】
大規模解析サービスを支える監視サービスと監視構成のポイント
デプロイ自動化をマルチクラウドで! CDツール「Spinnaker」をAWS上で検証してみた【デブサミ2018】
超簡単! WPFなどの.NETのアプリからUWPのAPIを使う ~日本語の読み仮名を取得するAPIを題材に
情シスの仕事こそ、クリエイティブでおもしろい! 12000人以上が利用するヤフーの社内システムづくり【デブサミ2018】
新着記事一覧を見る

add()内で行われる「読んで/計算して/書く」一連の処理はその途中で割り込まれてはならないatomic(不可分)な処理なのです。atomic性が保証されないためにデータの辻褄が合わなくなる現象はdata race(データ競合)と呼ばれています。

atomicでなくてはならない一連の処理の実行を複数のスレッドに行わせないカラクリがmutex。

ヘッダ<mutex>に定義されたstd::mutexの主要メンバ関数は、

- lock():ロックを取得する
- try_lock():ロックの取得を試みる
- unlock():ロックを解放する(手放す)

の3つ。ロックは実行権/使用権を手に入れる鍵であり、この鍵を取得できるスレッドはただ1つです。lock()によって使用中となったmutexを他のスレッドがlock()すると、そのスレッドはunlock()によって解放されるまで待ち状態となりlock()から戻ってきません。複数のスレッドがロックを待っている状態でロックが解放されると、いずれか1つのスレッドのみがロックを取得しlock()から抜けてきます。なので、

list03

```
std::mutex mtx;

void mutex_add(long* address, long value) {
  mtx.lock();
  *address += value;
  mtx.unlock();
}
```

としておけば複数のスレッドがmutex_add()してもmtx.lock()とmtx.unlock()に挟まれた処理を行えるスレッドは1つだけ(ほかのスレッドは待たされる)となりatomic性が保証されるってスンポーです。

mutexでガードしていないハダカのadd()と比べてみました。

list04

```
// std::mutexでガードした足し算
std::mutex mtx;

void mutex_add(long* address, long value) {
  mtx.lock();
  *address += value;
  mtx.unlock();
}

……

int main() {
  using namespace std;

  long count = 0L;

  cout << "data-race:            ";
  count = 0;
  run(add, &count);
  cout << ", count = " << count << endl;

  cout << "std::mutex:           ";
  count = 0;
```

```
  run(mutex_add, &count);
  cout << ", count = " << count << endl;

}
```

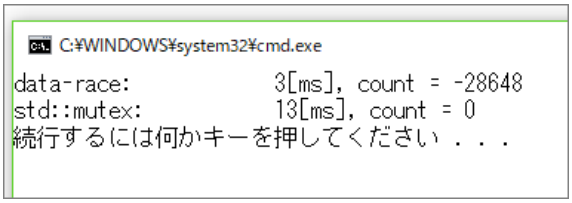

fig02

countはめでたく初期値0に戻っていますが、そのかわりちょっと遅くなっています。ロックの取得/解放には、そこそこの時間がかかるんですよ。

残るメンバ関数:try_lock()は、ロックが取得できるなら取得してtrue、取得できないならあきらめてfalseを返します。スレッドが待ち状態にはなりません。

次ページ≫ スレッドの3状態

1 2 3 次へ≫

バックナンバー 印刷用を表示

著者プロフィール

επιστημη(エピステーメー)

C++に首まで浸かったプログラマ。Microsoft MVP, Visual C++ (2004.01〜) だったり わんくま同盟でたまにセッションスピーカやったり 中国茶淹れてにわか茶人を気取ってたり、あと Facebook とか。著書: - STL標準講座 (監修) -...

※プロフィールは、執筆時点、または直近の記事の寄稿時点での内容です

バックナンバー

連載:C++11:スレッド・ライブラリひとめぐり

- C++11:スレッド・ライブラリひとめぐり【補足編:3】
- C++11:スレッド・ライブラリひとめぐり【補足編:2】
- C++11:スレッド・ライブラリひとめぐり【補足編:1】

ページトップへ