

Windowsのスレッドプーリング

スレッドプーリングの実装例

C++ Windows

WEB用を表示

ツイート

0

0

0

G+

Joshua Emele[著] / japan.internet.com[訳]

2005/04/01 00:00

ダウンロード ↓ [VS.NET2003のデモプロジェクト \(19.0 KB\)](#)

ダウンロード ↓ [ソースファイル \(5.1 KB\)](#)

複数の実行スレッドを管理し、処理を各スレッドに分散させるためのテクニックである「スレッドプーリング」を解説する。

はじめに

スレッドプーリングとは、複数の実行スレッドを管理し、処理を各スレッドに分散させるためのテクニックである。場合によっては、同時実行制御などの機能も、スレッドプーリングの一部と考えられる。スレッドプーリングは、以下の処理を作業を行うのに適した方法である。

複雑な処理を管理する

スレッドプーリングはステートベースの処理に適している。システムをいくつかのステートマシンに分解できる場合には、スレッドプーリングを利用してそのシステムを効果的に実現することができる。これには、たいていの場合にマルチスレッドアプリケーションのデバッグが単純化されるという副次的なメリットもある。

アプリケーションの拡張性を高める

正しく実装すれば、スレッドプールによって同時実行の制限を設け、アプリケーションの拡張性を高めることができる。

リスクを最小限に抑えながら新規コードを追加する

スレッドプーリングを利用すると、「サンドボックス」と呼ばれる、システムの実行をいくつかの小さな単位に分割するセキュリティモデルを実現できる。「サンドボックス」モデルでは、スレッドプーリングによってプロセス間の結び付きが緩やかになり、プロセスからデータが独立するため安全である。この方式では、プロセス同士が複雑にからみ合うのではなく、点で接することになる。これにより、保守の手間が大幅に減少する。特に、大規模なマルチスレッドアプリケーションでは効果が大きい。

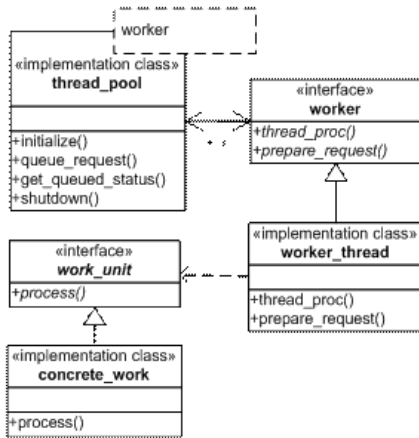
設計

スレッドプールの概念モデルは単純で、

1. スレッドプールがスレッドを開始する
2. 処理がスレッドプールのキューに入れられる
3. キュー内の処理が使用可能スレッドで実行される

という形である。テンプレートを使用すると、プールをスレッド/処理の実装から独立させることができる(このテクニックは静的ポリモーフィズムと呼ばれる)。

図1 スレッドプールのコラボレーション図



スレッドプールはスレッドの作成を担当する。スレッドは`worker::thread_proc`によって実行を開始する。リクエストはスレッドプールのキューに入れられる。このとき、ワーカーがリクエストの準備をし、リクエストをキューに入れる。スレッドが処理を実行できる状態になったときは、スレッドが `thread_pool::get_queued_status`を使用して、スレッドプールに待機中の処理をリクエストする。待機中の処理がない場合は、処理が送信されてくるまで、スレッドは休止状態になる。

チェインング

このワーカー実装によって処理をキュー化できるが、さらに一歩進んでみよう。このスレッドプールは問題を個別のステップに分解するのに役に立ち、それによってシステムの能力を最大限に活用しつつ、複雑さとリスクを最小限に抑えることができる。しかし、現在の実装では一度に1つの処理しかキュー化できないので、一連の処理を論理的にまとめるのが難しい。さらに、後続の処理をキュー化するためには、処理がいつ実行されたかを知る手段が必要になる。

具体的な例

システムをダウンさせ、システムデータを再構築し、システムを再びオンラインにする、という一連の処理がある。これは3つのステップ、3つの処理である。これを次のような形で実装したいと思う。

```
thread_pool::instance().queue_request(
    (core::chain(), new system_down, new rebuild_data,
     new system_up));
```

ここで出てきたチェイン(chain)とは、一種の処理単位のことである。正確には、このチェインは、実際の処理単位(実は処理単位のコンテナ)のコンテナとしてのみ機能する。

```
struct chain {
    struct data : work_unit, std::list<smart_pointer<work_unit> > {
        void process();
    };
};

chain() : m_work(new data) {}
chain& operator,(work_unit* p_work);
operator work_unit*() { return m_work; }
smart_pointer<data> m_work;
}; // struct chain
```

`chain::operator`は、次の処理だけを行う。

```
m_work->push_back(p_work);
return *this;
```

`chain::data::process()`は、次の処理だけを行う。

```
front()->process();
pop_front();

// if not empty, requeue
if (true == empty()) return;
thread_pool<worker_thread>::instance().queue_request(this);
```

コードの使用方法

まず、使用するスレッドプールを初期化する。スレッドプールはパラメータ付きのシングルトンなので、使用するワーカーの種類ごとに1つのスレッドプールインスタンスが存在する。global::thread_poolクラスは、core::thread_pool<core::worker_thread>を表す便利なtypedefである。

```
global::thread_pool::instance().initialize();
```

core::worker_threadをワーカー実装として使用する場合は、すべての処理がcore::work_unitから派生することになり、このプロセスが呼び出されたときに処理が実行される。

```
struct mywork : core::work_unit
{
    void process() throw()
    {
        // work is processed here
    }
}
```

処理をキュー化するには、クラスのインスタンスを作成し、必要に応じて初期化する。この処理を、thread_pool::queue_requestを使用してキュー化する。

```
// demonstrate chaining
global::thread_pool::instance().queue_request(
    (core::chain(), new work_1, new work_2, new work_3));
```

スレッドプールをシャットダウンするには、thread_pool::shutdownを使用する。

```
global::thread_pool::instance().shutdown();
```

他のスレッドプールを作成するには、ワーカースレッドとその構成部品を定義すればよい。ワーカースレッドでは、request_type、prepare_request、thread_procを**必ず**定義する必要がある。thread_procに渡されるLPVOIDパラメータは必ず0で、使用してはならない。このパラメータを通じてコンテキストを提供するようスレッドプールを拡張することができる。次のコードでは、io::threadとio::thread_poolを定義している。

```
// sample io worker thread
namespace io {
    struct thread
    {
        typedef io::session request_type;
        static void prepare_request(request_type*) throw();
        static void thread_proc(LPVOID) throw();
    };    // struct thread

    struct thread_pool : core::thread_pool<io::thread> {};
}    // namespace io
```

ユーザーはio::thread_pool::instance();を使用してio::thread_poolにアクセスできる。

デモプログラムについて

デモプログラムは次のことを行う。

- ・ global::thread_poolインスタンスの初期化
- ・ 3種類の処理のインスタンス化
- ・ 処理をまとめるチェーンのインスタンス化
- ・ 処理のキュー化と実行
- ・ スレッドプールのシャットダウン

チームで開発を行っている場合は、大きな処理を小さな処理単位に簡単に分割し、チーム内で分配することができる。それぞれの処理単位は独立してテストでき、かつ、最終的な製品として統合もできる。それぞれの開発者は、担当範囲内で自分の仕事をまっとうすればよいのだ。

スレッドプールは、拡張性とパフォーマンスに優れた大規模システムを短期間で安全に開発するための素晴らしいツールである。

ぜひ試していただきたい。

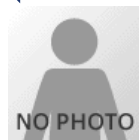
バックナンバー

WEB用を表示

ツイート000G+

- PR『マンガで分かるプログラミング用語辞典』プログラミング入門書の副教材としてぜひ
- PR『C#で始めるテスト駆動開発入門』C#でのTDD実践方法をステップバイステップで紹介
- PR『Scott Guthrie氏 Blog翻訳』マイクロソフトの最新技術動向はここでチェック


著者プロフィール



NO PHOTO

Joshua Emele (Joshua Emele)

サンフランシスコ在住。Plugware Solutions, Ltd. に勤務。C++ でのネットワーク、データベース、ワークフロー アプリケーション開発を専門とする。人生とパートナーをこよなく愛し、クラシック ギター、ハイキング、デジタル機器を好む。Plugware Solutions,...



NO PHOTO

japan.internet.com (ジャパンインターネットコム)

japan.internet.com は、1999年9月にオープンした、日本初のネットビジネス専門ニュースサイト。月間2億以上のページビューを誇る米国 Jupitermedia Corporation (Nasdaq: JUPM) のニュースサイト internet.com や EarthWeb.c...

※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
Article copyright © 2005 Joshua Emele, Jupitermedia Corporation, Shoeisha Co., Ltd.

[ページトップへ](#)



CodeZineについて

各種RSSを配信中

プログラミングに役立つソースコードと解説記事が満載な開発者のための実装系Webマガジンです。
掲載記事、写真、イラストの無断転載を禁じます。
記載されているロゴ、システム名、製品名は各社及び商標権者の登録商標あるいは商標です。



ヘルプ	スタッフ募集!	IT人材	プロジェクトマネジメント
広告掲載のご案内	メンバー情報管理	教育ICT	書籍・ソフトを買う
著作権・リンク	メールバックナンバー	マネー・投資	電験3種対策講座
免責事項	マーケティング	ネット通販	電験3種ネット
会社概要	エンタープライズ	イノベーション	第二種電気工事士
		ホワイトペーパー	

 [メンバーメニュー](#) |  [ログアウト](#)