Linuxにおける再利用可能なスレッドライブラリの実装*

1H−1

五十嵐 浩隆†　井田 昌之‡

青山学院大学大学院 国際政治学研究科国際コミュニケーション専攻§

1 はじめに

マルチスレッドを用いるアプリケーションの動作を高速化することを目的として研究を行った。スレッドの生成／消滅を繰り返すのではなく、一度生成したスレッドを使い回して再利用することに着目して、スレッド管理機構に関する設計と試作を行った。実装はLinuxに対して行い、GNU C Library[1] と LinuxThreads Library を用いた。LinuxThreads は POSIX 1003.1c スレッドパッケージの Linux への実装である [2]。実装言語には C++言語を用いて、オブジェクト指向による設計を行った。

2 設計の方針

設計の方針として、以下の五点を定めた。

- スレッドを使い回して再利用する
- 通常の関数を別スレッドとして実行させる
- 非同期に子スレッドからの返り値を取得することを可能にする
- 使い回しを意識させないスレッド管理機構を提供する
- スレッドのガベジコレクション機構を提供する

これらについて順に説明する。

2.1 スレッドを使い回して再利用する

POSIX スレッドインタフェースでは、別スレッドとして関数を実行するとき、新たなスレッドを生成して関数を実行させ、関数が実行を終えるとスレッドを終了するという仕様になっている [3]。そこで任意の関数を実行できるようなブローカを作成し、このブローカを別スレッドとして起動する。ブローカは関数の実行を終えると、条件変数を待つことで休眠状態にはいる。親となるスレッドからの指示によって、ブローカは休眠状態から抜け、新たな関数の実行にとりかかる。ブローカの休眠や合図のために必要な情報を管理するオブジェクトを用意し、各々のスレッドを表現する。

2.2 通常の関数を別スレッドとして実行させる

POSIX スレッドインタフェースでは、別スレッドとして関数を実行する場合、子スレッドは pthread_exit インタフェースを用いて親スレッドに返り値を渡すと同時に、スレッドを終了するという仕様になっている。これに対して、通常の return 文によって返り値を返すことを可能にする。

2.3 非同期に子スレッドからの返り値を取得することを可能にする

POSIX スレッドインタフェースでは、pthread_join インタフェースを用いて目的のスレッドの終了まで親スレッドの実行が止められ、返り値を得る仕様になっている。これに加えて、子スレッドと同期を取らずに目的のスレッドの終了を検査するインタフェースを提供する。

2.4 使い回しスレッドを意識させないスレッド管理機構を提供する

スレッドを使い回す場合、開発者は実行中のスレッドだけでなく休眠中にあるスレッドをも管理する必要がある。この負担を軽減するため、使い回しスレッドを管理するスレッドプールオブジェクトを提供する。図1にスレッドプールの動作の様子を示す。

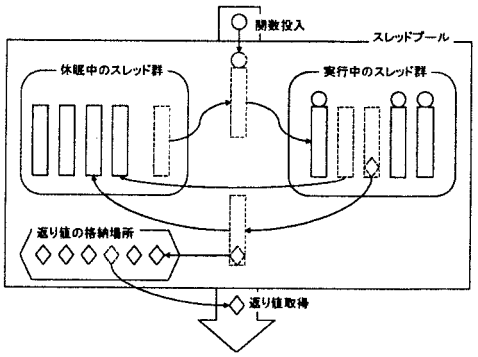

図 1: スレッドプールの動作の様子

スレッドプールは初期化時に定数個のスレッドオブジェクトを生成し、スレッドを休眠状態にする。ユーザが別スレッドとして関数を実行させるときは、特定のスレッドオブジェクトを指定するのではなく、スレッドプールに関数の実行を依頼することで行う。スレッドプールは、休眠中のスレッドオブジェクトと実行中のスレッドオブジェクトを分類して管理する。関数の実行が依頼されたときには、管理下にある休眠中のスレッドを一つ起こす。休眠中のスレッドが存在しない場合、スレッドを新たに起動する。

関数が実行を終えると、スレッドは返り値をスレッドプール内に格納してから、休眠状態にはいる。この仕様により、返り値の返却が要求されていないスレッ

---

*A Design and Implementation of Re-usable Thread Library on Linux

†Hirotaka Igarashi <hiro@sipeb.aoyama.ac.jp>

‡Masayuki Ida <ida@sipeb.aoyama.ac.jp>

§Graduated School of Int'l Communication, SIPEB, Aoyama Gakuin University, Shibuya, Tokyo, JAPAN, 150-8366

ドでも、依頼された関数の実行を終えたなら、次の関数の投入に備えることが可能になる。

呼び出し元は、スレッドプールに返り値の要求をすることで、返り値を得る。

2.5 スレッドのガベジコレクション機構を提供する

OSのリソースを節約することを目的に、スレッドの使用状況をプロファイルし、再利用されないと推測されるスレッドを終了させる。

多数のスレッドが長時間使用されないと認められるとき、使用されないスレッドの一つをガベジコレクションの回収対象とする。タイマ等を用いた非同期のガベジコレクションは行わず、スレッドが休止状態に入るタイミングでプロファイリングとガベジコレクションを行う。またこのタイミングをガベジコレクタの時間の単位とする。

同時に起動されるスレッドの数には、ゆらぎがあることを想定する。ここでは、同時に存在しているスレッド数が実行中のスレッド数と動的に変化するあそびとを加算した数になる状態を目的の状態とし、余剰のスレッドをガベジコクションの対象とする。極端に休眠しているスレッド数が多い場合は、それらは迅速に回収されるようにする。また、ゆらぎの幅の変化に備え、休眠中のスレッド数があそび近傍にあるときには、回収の速度を低速にする。

3 実装

実装のポイントとして、スレッドの休眠と、ガベジコレクションについて述べる。

まずスレッドの休眠は条件変数を用いることで実現した。Solarisなどの持っているスレッドライブラリには、スレッドをサスペンドするインタフェースが存在しているが、POSIXスレッドインタフェースには該当するインタフェースが存在しない（例えば文献[4]）ためである。

次にスレッドのガベジコレクションのアルゴリズムについて以下に示す。あそびは全スレッド数の1割程度であると、経験的に定めた。また、プロファイリングに必要な演算を高速に行うため、あそびを全スレッド数の12.5%($0.125 = 1/2^3$)と定めた。

生成するスレッド数の上限を定数max、現在存在しているスレッドの総数を total、正の整数定数 bias に対して、total×0.125+bias をあそびとし、各々のスレッドが休眠状態に入る度に total×0.125+bias < sleeping であるか調べ、max-total 回連続して上記の条件を満たしているとき、スレッドを一つ消去する。

成果は共有ライブラリとして利用できる。ライブラリのファイルサイズは238,478 bytesである。

4 評価

Linux(Redhat Linux 5.2J, kernel 2.0.36, glibc-2.0.7, gcc-2.9.5.1, PentiumPro/200MHz, main memory 64Mbytes)上で性能測定を行った。性能測定はPOSIXスレッドインタフェースを直接用いてスレッドの生成／消滅をn回繰り返すのに要する時間と、使い回しスレッドを用いて同じ回数だけスレッドの依頼を繰り返したときにかかる時間とを比較した。使い回しスレッドの計測時間はスレッドプール初期化処理及び終了処理にかかる時間も含んでいる。図2に、結果について最小二乗法を用いて直線近似を行ったグラフを示す。

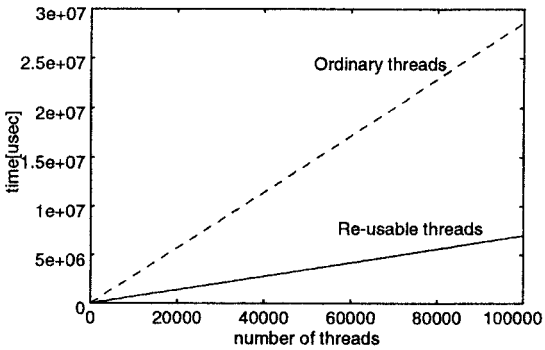

図2: 使い回しスレッドの性能測定

スレッドが15回以上利用される場合には本方式が有効であることが分かる。マルチクライアントサーバのように多くのスレッドを使うときにパフォーマンスの向上が見られる。

ガベジコレクションについての厳密な評価は今後の課題である。

5 おわりに

スレッドを使い回してパフォーマンスを向上させると同時に、スレッドの管理を行う仕組みを設計し実装した。またこのライブラリを利用するためのAPIを設計した。

なお、本方式を用いてGCP(Global Computation Platform：大域計算プラットフォーム)[5]のLinuxへの移植を行い、本方式の有効性を確認した[6]。

本発表の内容は http://noa.sipeb.aoyama.ac.jp/Rthread/にある。

謝辞：(株)デジタル・ビジョン・ラボラトリーズの前川博俊氏には予稿執筆にあたってアドバイスを受けた。同じく千葉哲央氏には実装技術に関して多くの示唆を受けた。

参考文献

[1] Free Software Foundation, http://www.gnu.org/software/libc/libc.html
[2] Xavier Leroy,http://cristal.inria.fr/~xleroy/linuxthreads/
[3] David R. Butenhof/著 油井尊 宗方あゆむ/訳,「POSIXスレッドプログラミング」, アジソン・ウェスレイ・パブリッシャーズ・ジャパン, Oct. 1998
[4] Xavier Leroy, http://cristal.inria.fr/~xleroy/linuxthreads/faq.html
[5] 前川博俊 斎藤隆之 千葉哲央,「大域計算アーキテクチャ—広域環境での平行計算とマルチメディア処理の統合的実現」, 情報処理学会論文誌, Vol.39, No.2, Feb. 1998, pp.267-282
[6] 五十嵐浩隆,「GCP Linux 移植レポート」, Mar. 2000 (予定)