単一仮想記憶の特徴を考慮したスケジューラ構成について

5M－5

寺本 圭一　　岡本 利夫

(株) 東芝 研究開発センター　情報・通信システム研究所

1 はじめに

我々が現在研究，開発中の次世代64ビットOS (Cubix[CUBe of unIX]) では，単一仮想記憶空間上で走行する複数のスレッドに対して，参照可能な領域範囲 (保護領域) を個別に設定できるような柔軟なモデルを採用している．こうしたスレッド毎に設定可能なアクセス保護機構を既存のページ・テーブルを用いる多くのアーキテクチャ上で実現する場合，スレッド単位に個別のページ・テーブルを保持させるという方式が考えられる．

本稿では，このアクセス保護に関連して必要となる仮想記憶管理用資源の削減 / 管理効率の向上を実現する制御機構をユーザ側からの最適化ヒントとして指定可能にしたスレッド・インタフェース，および，各スレッド間の保護領域に関する親和性を考慮したスケジューリング手法について述べる．

2 thread viewについて

Cubixにおけるスレッドは，単一仮想記憶空間内に複数配置されうる領域間を跨って直接参照することが可能である．ただし，スレッドには，アクセス可能な領域の範囲 (保護領域) が各々設定され，アクセスが許可されない領域に対する操作から各領域は保護される．

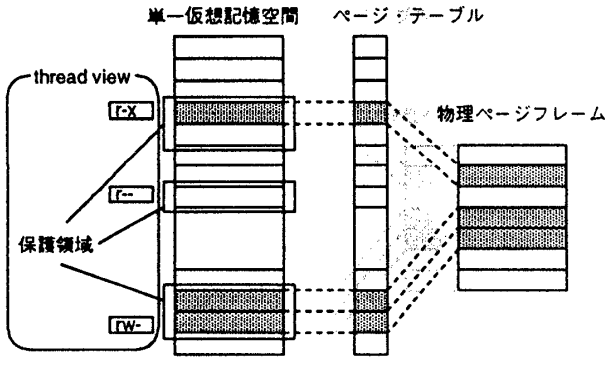

図1：thread view

このような，あるスレッドに関する保護領域およびその領域の保護属性や状態をここでは “thread view” と呼ぶことにする (図1)．

thread viewは，あるスレッドに着目した場合にそのスレッドがアクセス (read / write / execute) 許可されうる領域の集合体を指し，具体的な下位レベルに展開されると，あるスレッドに関して有効化されうるページ・テーブル・エントリ (PTE) 集合に対応付けられる．また，thread viewは，スレッド実行中に動的に変更されうる性質を有する．

こうして個別のページ・テーブルをスレッドに対応させることにより，各々の領域に対する保護は実現できるが，これでは従来の多重仮想記憶空間の場合と同様，スレッド (プロセス) 切替え時のページ・テーブル切り替え操作や，スレッド (プロセス) 単位でのページ・テーブル資源保持による主記憶の占有率の増加などが生じてしまう．また，単一仮想記憶方式では仮想記憶空間は全スレッドから共有されるため，仮想 / 物理アドレス変換情報やページ状態変更時などにおけるページ共有スレッドへの通知コストなどが，同一ページの参照スレッド数の増加に応じて大きくなることが危惧される．

そこで，こうした問題の改善のために，あるスレッド集合内では単一のページ・テーブルが共有可能な環境を提供することを考える．

3 thread view対応スレッド・インタフェース

ある複数スレッド内では，互いに関連 / 協調性が強く，その操作する対象となる領域の範囲や保護属性 (thread view) が類似しているような場合がいくつか考えられる．例えば，あるプログラム・テキスト上のサービスを単純に複数スレッドにより並列実行したり，サーバ / クライアント間のみで閉じた実行を行う場合などが挙げられる．

そこで，あらかじめ複数スレッド間でthread viewの関連性を考慮し，必要に応じてスレッド間でページ・テーブルを共有できるような制御をユーザ側から指定できるようなスレッド・インタフェース機構を提供する．

```
int thread_share_view(thread_t src,
                      thread_t dst, long flags);
```

このスレッド・インタフェースは，`src` スレッドのthread viewを `dst` スレッド側で共有させるというものである．共有時の動作は `flags` によって指定され，ペー

A Scheduler Model for a Single Address Space Operating System
by Keiichi TERAMOTO, Toshio OKAMOTO,
Communication and Information Systems Research Labs.,
R&D Center, TOSHIBA CORPORATION
1, Komukai-Toshiba-cho, Saiwai-ku, Kawasaki 210, Japan

ジテーブルの共有を伴う / 伴わない, thread view の共有可能性を確認する / しない, などが指定できる.

このインタフェースは, 新規スレッドを生成した時点で即座に, あるスレッドに付随の thread view との共有を行いたい場合などにも利用される.

次に, 以下のスレッド・インタフェースでは, thread view 共有 (可能) スレッドに対して, 実行待ち行列への登録時には後述する thread view 切替えの発生頻度を抑制するスケジューリング制御機構 ("thread view affinity scheduling") の適用を要望するスレッドであることをスケジューラに通知するための機能を提供する. これはユーザ側からスケジューリング時の振る舞いに関するヒントをカーネルに与えるものであり, スケジューラはこれに基づいた動作を行う. なお, 本インタフェースが実際に行う操作は, スレッド属性として, thread view affinity scheduling 対象スレッドであることを示すフラグをセットすることである.

```
int thread_set_view_affsched(thread_t thr);
```

4 thread view affinity scheduling

ここでは, スレッド間で thread view の親和性 (affinity) を考慮したスケジューリングを行う thread view affinity scheduling 方式について説明する.

前述したように, thread view が共有される両スレッド間では, ページ・テーブル切替えのための操作は不要である. そこでまず, スレッド・ディスパッチ部へ, スケジューラにより選択された次期実行スレッドにおける thread view が現在のスレッドと等しい場合にはページ・テーブルを変更しない (TLB を有する場合は TLB フラッシュも行なわない) という判定コードを挿入する.

次に, ユーザによって本スケジューリング方式の対象と指定された各スレッドに対して, 実行待ち行列内で thread view(ページ・テーブル) を共有するスレッドをなるべく連続して配置するよう制御する方針を適用する (図 2).

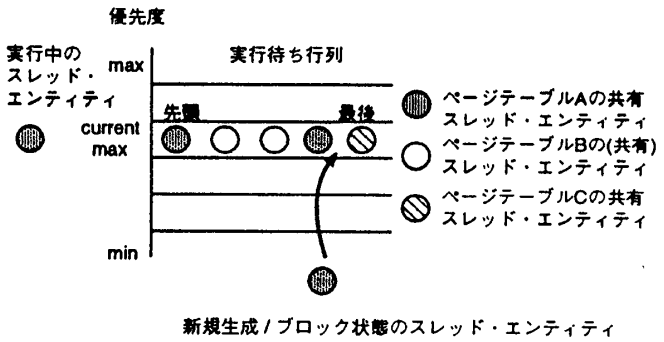

図 2 : スレッド・エンティティの実行待ち行列への登録

具体的には, あるスレッドがブロック状態から再開される際や新規にスレッドが生成される際など, スレッド・エンティティを実行待ち行列内に登録する際に, そのスレッド属性内に thread view affinity scheduling を指示するフラグがセットされていれば, 行列の最後から thread view を共有するスレッド・エンティティを探索し, これが見つかれば更に前方をスキャンし, 今度は thread view を共有しないスレッド・エンティティの有無を調査する. これが見つかった場合, 最初に見つかった thread view 共有スレッド・エンティティの直後に再開スレッド・エンティティを登録し, 見つからない場合には, 行列の最後尾に登録する.

これにより, 実行待ち行列内で thread view 共有スレッドが連続して配置される可能性が高くなり, これらの間でスレッドを切り替えている限り, ページ・テーブルの切り替えが不要なディスパッチを行える機会が増加する.

また, この応用として, 疎結合型マルチプロセッサシステムにおいて, スレッドのプロセッサ割り付け時に, thread view が同等となりうるスレッド群を同じプロセッサに対して割り付け, ページ・テーブルを共有化させる制御方針を適用することも考えられる.

5 さいごに

本稿では, 単一仮想記憶空間内でスレッド個別に設定されうる保護領域 (thread view) を既存のページ・テーブルを使用して実現する際に, thread view の共有化に関する制御能力をユーザに与えるスレッド・インタフェース, および, これを利用した "thread view affinity scheduling" と呼ぶスケジューリング方針を導入することにより, ページ・テーブルの資源 / 管理コストの削減, および, スレッド・ディスパッチ時のページ・テーブル切り替え頻度の低減など, スレッド・スケジューリング時の軽量化を図った.

なお, 今回は, ページ・テーブルの共有化はユーザ指定により即座に行われるものとしたが, 本来各スレッドが保持する thread view はページ・テーブル共有時に同一であっても後に動的に変更されうるので, 時間の経過に伴い共有ページ・テーブルの内容に不整合が生じる可能性がある. こうした thread view 変更時のスレッド間 PTE の一貫性保証については, ページ・フォルト時にこれを検出し, 必要に応じて共有化を解除して対処するなど, 現在検討中である.

参考文献

[1] Okamoto et al., "A Micro Kernel Architecture for Next Generation Processor", pp.83-94, Microkernels and Other Kernel Architectures Symposium, USENIX, 1992/4.

[2] 岡本 俊夫, 津田 悦幸, 福本 淳, 寺本 圭一, 「次世代アーキテクチャ向けオペレーティングシステムの開発」, 情報処理学会システムソフトウェアとオペレーティングシステム研究会報告, 95-OS-66, pp.17-23, 1995/9.

[3] Eric J.Kokdinger, Jeffrey S. Chase, and Susan J. Eggers, "Architectural support for single address space operating systems", In Proc. of 5th International Conference on Architectural Support for Programming Language and Operating Systems, pp. 175-186, Vol. 26 of ACM SIGPLAN Notices, 1992/10.