計算機アーキテクチャ 83−27
（1990．7．18）

マルチプロセッサ UNIX MUSTARD における記憶域管理方式

二瓶 勝敏*, 広屋 修一*, 川口 浩美**

* 日本電気(株)C&C 共通ソフトウェア開発本部　** 日本電気技術情報システム開発(株)

MUSTARD は筆者らが開発した密結合マルチプロセッサ用の OS である. マルチプロセッサでは, それぞれのプロセッサの TLB の一致制御が必要であり, このための方式は既に提案されている. 本報告では, MUSTARD での, 運用中のプロセッサの追加 / 切り離しや, 系の切り替え等の特徴と, TLB の一致制御が同時に起こった時の問題点と解決法について述べる. また,PTE の書き換えをバスをロックせずに行なうプロセッサにおいては, 複数のプロセッサが同時に PTE の書き換えを行なったときに変更ビットが失われる可能性がある. この問題への MUSTARD での対処法を示す.

Memory Management of Multiprocessor UNIX "MUSTARD"

Katsutoshi NIHEI,*, Shuichi HIROYA*, Hiromi KAWAGUCHI**

*C&C Common Software Development Laboratory, NEC Corporation,
**NEC Scientific Information System Development Ltd.
Igarashi Bldg.,2-11-5, Shibaura, Minato-Ku, Tokyo 108, Japan

In multiprocessor systems, TLB (translation look-aside buffer) must be maintained consistently. Some methods for this problem have already been proposed. However, because our OS, MUSTARD, supports dynamic processor mounting/unmounting and configuration change, it has potential deadlock with those methods.

In addition, when using processors that update PTE (page table entry) without locking bus, there is another problem that dirty bit is lost when updating PTE.

We show the software method for each problem that is used in MUSTARD.

1 はじめに

高性能で, 小型のマイクロプロセッサの出現によって, 比較的簡単に, マルチプロセッサ構成のシステムを構成できるようになった. 特に, 密結合型のマルチプロッサ構成においては, 過去のソフトウェア資産を継承しつつ, マルチプロセッサ構成による高いスループットを達成できる等のメリットを持っている.

そこで, 筆者らは, 交換機システム等で利用実績を持つ V60/V70 用リアルタイム UNIX **RX-UX 832**[1, 2] をベースに, 密結合マルチプロセッサシステム上で動作するリアルタイム UNIX **MUSTARD**[3] を開発した.

MUSTARD は, 交換機 / 通信機等でサポートが望まれている無停止化機能を備えており, システム運用中に, プロセッサをシステムから切り離したり, 追加したりすることができる. また実行中のコンテクストを保存し, 系を切り替え, 保存されたコンテクストから実行を開始することができる. これらの性質と関連してメモリ管理についても工夫が必要となっている. また, ページテーブルエントリを複数のプロセッサで同時にアクセスする場合に起こる問題とその解決法についても述べる.

以下, MUSTARD の特徴について簡単に述べたあと, MUSTARD での記憶域管理の問題と実現について述べる.

2 MUSTARD の特徴

MUSTARD は次のような特徴を持つ.

・システム運用中にプロセッサの追加 / 切り離しができる.

システム運用中の, プロセッサ・パワーの増強, 故障が多発するプロセッサの切り離し, 保守のためのプロセッサボードの交換等を可能とするためにプロセッサの追加 / 切り離しをサポートする, mnt_prc() / umt_prc() システムコールを提供している [4].

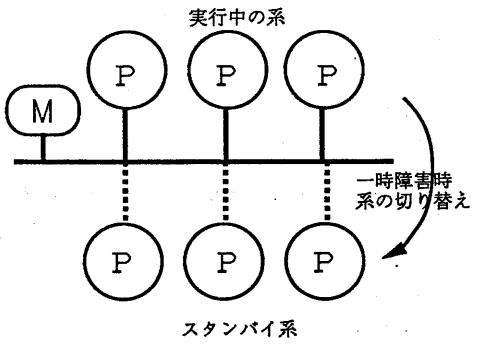

図 1: 二重系構成

・コンテクストを保存した系の切り替えができる.

システムを論理的に図 1のような二重系構成にしておき, 故障発生時や, メンテナンス時などに, 実行中のタスクのコンテクストを保存して, もう一方の系に切り替え, セーブされたコンテクストから実行を継続することができる.

・2 階層 OS の構成となっている.

MUSTARD は, リアルタイム性の高いリアルタイムカーネル部 **MUSTARD-RK** と, その上位層に位置する UNIX カーネル **MUSTARD-UX** から構成されている [5]. UNIX タスクは, リアルタイムタスクよりも優先度が低く, リアルタイムタスクにプリエンプトされる. リアルタイム性を必要とする処理は, オーバヘッドの小さいリアルタイムカーネルで実行させることにより, リアルタイム性を保証することができる.

また, タスクのディスパッチ機構, 記憶域管理等に必要なプロセッサ間通信機構等は, MUSTARD-UX で直接行なうのではなく, MUSTARD-RK の機構を利用するなど, MUSTARD-UX は, MUSTARD-RK で提供されている機能を使用して実現されている.

3 MUSTARD における記憶域管理上の問題点

MUSTARD の開発に当たって, 筆者らが遭遇したメモリ管理上の問題点は次の 2 点である.

- TLB の一致制御
- PTE 中の変更ビットの喪失

以下の節でそれぞれの問題点と筆者らがとった実現法を述べる.

4 TLB の一致制御

密結合マルチプロセッサシステムでは, それぞれのプロセッサで, 同一ページに対する TLB(Translation Lookaside Buffer) を一致させる必要がある.

UNIX においてこれが問題になるのは, プロセスのページアウト / スワップアウト時である. あるプロセッサで実行中のプロセスのページを, システムがページアウトしたとする. ページアウトされた後は, そのページの物理メモリは回収されて, 別のプロセスが使用する可能性がある. ページアウトされたプロセスのページテーブルエントリ (PTE) が, プロセッサの TLB に残っていた場合, これを使用してプロセスはこのページにアクセスできてしまう.

このような事態を避けるために, システムがページの状態を変更する時には (ページの状態の変更は PTE の書き換えという形で行なわれる), この変更を他のプロセッサにも認識させる必要がある.

この問題に対処するための方式が MACH オペレーティングシステムにおいても提案されている (この方式を Shootdown と呼ぶことにする)[6].

MUSTARD でも TLB の一致制御の方式としてこの方式を採用している. この方式はプロセッサ間での同期を必要とし, また一般的にプロセッサ間の同期は TLB の一致制御以外にも必要であり不注意に使用するとシステムのデッドロックを引き起こす.

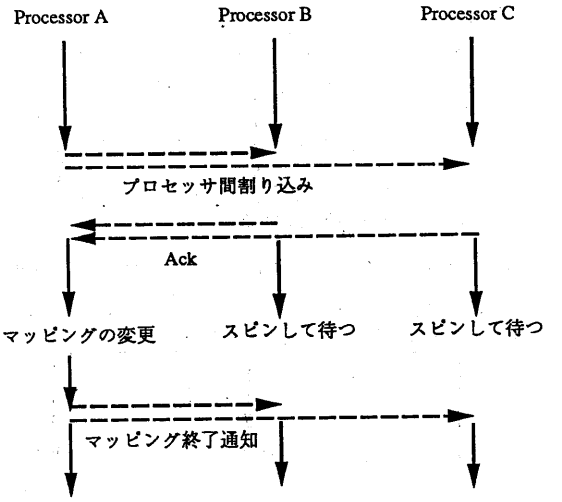

図 2: Shootdown

MUSTARD では, プロセッサ間での同期を系の切り替え時にも使用していること, プロセッサの動的な追加 / 切り離しによって, プロセッサの数が変化する可能性があることなどと関連してデッドロックを引き起こす可能性があった. 次節では, Shootdown と系の切り替えの処理について簡単に説明し, その後 MUSTARD での対処法について述べる.

4.1 Shootdown 方式

各プロセッサの TLB に矛盾が起きそうな変更を PTE に行なう時には, 全てのプロセッサで完全に同期をとる方法である.

PTE に変更を加えようとするプロセッサ (A とする) は, 他のプロセッサに, プロセッサ間割り込みをかける. 割り込みを受けたプロセッサは, プロセッサ A にアクノレッジを返し, その後プロセッサ A が, PTE の変更を完了するのを待つ. 全てのプロセッサからアクノレッジを受けたら, A は, PTE に必要な変更を加え, その後待ちに入っているプロセッサに変更が終了したことを知らせる. 待ちに入っていた他のプロセッサは, 必要な TLB をフラッシュして, 処理を続行する (図 2).

このようにして, 矛盾が生じそうな PTE

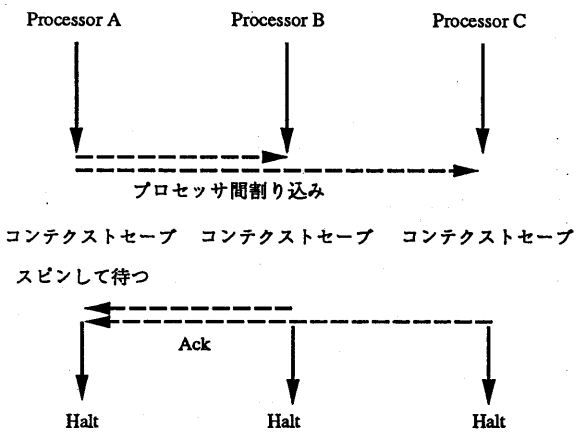

図 3: 系の切り替え

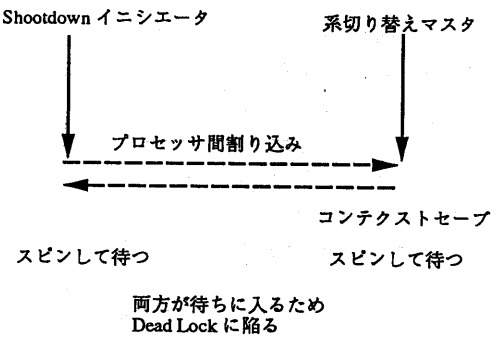

図 4: Shootdown と系の切り替え

の変更に対しては, 必ず同期をとりながら TLB をフラッシュする.

4.2 系の切り替え処理の実現

系に故障が発生した場合や, メンテナンスを行なう場合など, システムコールを使用して系の切り替えを行なうことができる. これは次のように行なわれる (図 3参照). まず系の切り替え用のシステムコールを発行したプロセッサ (マスタ) は, 全てのプロセッサ (スレーブ) に対して, 現在実行中のプロセッサのコンテクストをセーブするよう要求を出す. 次にマスタは, 全てのスレーブがコンテクストのセーブを行なうのを待つ. スレーブはコンテクストをセーブした後, マスタにコンテクストセーブ終了を通知する. マスタは全てのスレーブから通知を受けた後, スタンバイ系に切り替えを行なう. スタンバイ系は, セーブされたコンテクストをリカバして実行を継続する.

4.3 MUSTARD における Shootdown の問題と実現

MUSTARD における TLB の一致制御の方法は Shootdown と基本的に同じである. しかし MUSTARD では, プロセッサの動的な追加 / 切り離しを可能としたため, システムに存在するプロセッサの数は動的に変わることになる. このため, Shootdown 等のような全てのプロセッサに対して割り込みをかけて, それらからの返答を待つというような時には, デッドロックが起こる可能性がある. Shootdown イニシエータは Shootdown 開始時に全てのプロセッサにプロセッサ間割り込みをかける. この途中で, あるプロセッサがシステムから切り離されたとすると, このプロセッサは Shootdown のアクノレッジを返すことができず, Shootdown のイニシエータは, 永遠に待ち続けることになり, デッドロックを引き起こしてしまう.

また系の切り替えの処理と TLB の一致制御処理が同時に起こった時にもデッドロックが生じる (図 4参照). Shootdown イニシエータが, プロセッサ間割り込みをかけると同時に, 系の切り替えのマスタが他のプロセッサにプロセッサ間割り込みをかける. この時両者とも互いに相手のプロセッサで処理が終るのを待つためデッドロックが起きる.

このため, MUSTARD では, プロセッサの追加 / 切り離しの処理と, Shootdown の処理, および系の切り替え処理を排他処理として同時に起こらないようにしている. この場合, これらが衝突した時に処理に遅延が起こるが, プロセッサの追加 / 切り離しの処理や, 系の切り

替えは頻繁に行なわれるものではないので, 簡単な方式を採用した.

5 PTE 中の変更ビットの喪失

V60/V70 マイクロプロセッサでは, ページテーブルをもとに, 仮想－物理アドレスのマッピングを管理している. ページテーブルのエントリには, 物理メモリのページ番号の他に, そのページの保護情報, そのページが参照されたことを示す情報 (参照ビット), そのページに書き込みが起きたことを示す情報 (変更ビット), 等が含まれている.

マルチプロセッサの構成にした場合には, ページテーブル自体が複数のプロセッサから同時にアクセスされる可能性がある. 参照だけならば同時にアクセスされても問題ないが, ページの参照情報, 書き込み情報を, プロセッサはページテーブルエントリに書き込む. このため, ページテーブルエントリの情報が破壊される可能性がある.

プロセッサは, 参照ビット, 変更ビットを自動的にセットすることはあるが, 自動的にクリアすることはない. また, 変更ビットをセットする時には必ず参照ビットもセットする. このことから実際に問題となるのは, 参照ビット, 変更ビットがともに立っていないページに対して, 二つのプロセッサがほぼ同時に, 一方が参照をし, 他方が変更を行なった時である. 変更を行なったプロセッサでの,PTE の読み込み, 参照ビットと変更ビットのセット,PTE の書き出しという一連の操作が, 参照を行なったプロセッサでの PTE の読み込み, 参照ビットのセット, PTE の書き出しという一連の操作に完全にはさまれてしまった時に, セットすべき変更ビットがセットされなくなってしまう.

MUSTARD では, 次節で述べる解決策によって PTE を読み込んでから書き込むまでをバスをロックせずに行なうプロセッサにも対応している.

5.1 解決策

PTE の情報が破壊される可能性が生じるのは UNIX の場合であると, プロセス間で UNIX の共有メモリ関連システムコールを使用して, メモリを共有している時である. UNIX では, プロセスのページをページアウトする時に, そのページに変更が加えられているかどうかを調べ, 変更が加えられていた場合にのみ, そのページを二次記憶に書き出す. 万一変更ビットが落ちてしまうと, そのページをページアウトしようとした時に二次記憶上にページが書き出されないという問題がある.

これに対応する方法として次のようなことが考えられる.

1. PTE をプロセス間で共有しない.
2. 変更ビット喪失の可能性のあるページ (プロセス間での共有メモリ) は, 常に変更されたものとしてページアウトの時に二次記憶上に書き出す.
3. ページを read-only にして, 書き込みが起こった時にプロテクションフォールトを起こさせる. プロテクションフォールトのハンドラで, PTE の変更ビットをアトミックオペレーションを使用して書き換える.

1 の方法は, PTE を共有せずに, PTE 中の物理アドレスを同じにして物理メモリだけを共有するという方法である. 2 は, 共有メモリとして使用されているページのページアウトの時には PTE の変更ビットの内容に関わらずに, そのページは変更されたものとして二次記憶上に書き出してしまうものである. 3 は, Copy-On-Write の手法と同様の方法であるが, PTE 変更ビットの書き換えを OS のソフトウェアの制御の元で行なうことができる.

オーバヘッドの最も少ないのは, 1 の方法であるが, カーネルに対する改造は最も大きくなる. 3 の方法は 1 ページにつき, 1 回の割合

で例外処理のオーバヘッドを伴う．カーネルの改造量としては，方法 1,3 の中間と思われる．2 の方法は，カーネルに対する改造量は小さいが，これにより二次記憶上の領域を大量に消費する可能性があること，本質的には不要な二次記憶への書き出しを必要とすること等から効率は悪い．

筆者らは UNIX における共有メモリ関連システムコールの使用頻度とカーネルに対する改造量とを考え，2 の方式を採用した．

6 まとめ

本報告書では MUSTARD での記憶域管理について述べた．特に MUSTARD では，運用中にプロセッサの追加 / 切り離しや，一時故障多発時などに系の切り替えを行なうことができ，このことと TLB の一致制御のための処理とが同時に起きた場合の，デッドロックの可能性を示し，MUSTARD での対処法を述べた．

プロセッサをマルチプロセッサ構成で使用する場合に必要な機構として，バスをロックして排他的に行なう Test-And-Set, Compare-And-Swap 等の命令がある．これら以外にサポートが望まれる特性として，ページの参照，変更時にプロセッサが行なう PTE の書き換えをアトミックに行なうということが挙げられるが，これを行なわないプロセッサでのソフトウェアによる解決法を示した．

参考文献

[1] 下島, 世良, 水橋, 古城, 寺本,“V60/V70 リアルタイム UNIX – 概要 –”, 情処第 35 回全国大会 3D-3, 1987.

[2] Mizuhashi,and Teramoto, “Real-Time UNIX Operating System: RX-UX832”, Microprocessing and Microprogramming, Vol.27,1989.

[3] 広屋, 渡辺, 川口, 宮地,“種々のハードウェアに適用可能な密結合マルチプロセッサ用 OS MUSTARD”, 情処第 39 回全国大会 4P-6, 1989.

[4] 川口, 広屋, 渡辺, 宮地, “密結合マルチプロセッサ用 OS MUSTARD の耐故障性の強化と障害処理機構”, 情処第 39 回全国大会 4P-7,1989.

[5] 桃井, 関, 広屋, “密結合マルチプロセッサ用階層構造 OS MUSTARD における UNIX 核の実現”, 情処第 40 回全国大会 6G-3,1990.

[6] David L. Black, Richard F.Rashid, David B. Golub, Charles R. Hill, Robert V. Baron, “Translation Lookaside Buffer Consistency: A Software Approach”, Proceedings of the Third International Conference on Architectural Support for Programming Languages and Operating Systems, ACM,Boston,MA,April 1989.