64ビットアドレス空間上の協調作業を指向したオペレーティングシステムLucas

5B－1

猪原 茂和　上原 敬太郎　宮澤 元　益田 隆司

東京大学 大学院 理学系研究科 情報科学専攻

1　はじめに

複数のユーザーが分散計算環境上で一つの作業を行なう協調作業は、グループによるソフトウェア開発の支援やグループCADなど、多くの応用分野をもつ技術である。我々は、多くの言語やアプリケーションから共有可能な、協調作業の効率的な実現のための枠組みをみいだすことを目標として、Lucasオペレーティングシステム [2] の研究と開発を行なっている。本稿では、協調作業を実現する上でもっとも基本的な機能である、ユーザー間のデータ共有に対するサポートについて述べる。Lucasでは、近い将来主流になると思われる64ビットマイクロプロセッサのもつ広いアドレス空間を仮定することにより、従来大きなオーバーヘッドの原因となっていた主記憶上と二次記憶上とデータ表現の変換や、プロセス間のデータ表現の変換を最小限に抑え、協調作業のためのデータ共有を効率的に実現することを可能にしている。

2　Lucasにおける64ビット空間の管理

協調作業のためのデータ共有には、その特徴的な性質として、二次記憶上におかれた複雑なデータ構造、特にポインタを含む位置依存のデータが共有の対象になることと、データ構造に対する複数のユーザーのアクセスが時間的に細かい粒度で行なわれることがある。これに対し現在の多くのシステムでは、データ構造を主記憶と二次記憶との間で移動する際や、2つのプロセスの間で移動する際に、データの表現を変換する処理が必要であった。これは現在の多くのマイクロプロセッサの仮想空間の大きさが、二次記憶全体をアドレスするのに十分でないため、ひとつのデータ構造の主記憶上の表現と二次記憶上の表現とを違うものにせざるを得ないことに起因する。データ表現の変換は、複数のユーザーが時間的に細かい粒度でデータ構造を共有する場合には、協調作業全体の効率を低下させる大きなオーバーヘッドの原因となっていた。

Lucasでは、64ビットプロセッサの広大なアドレス空間を前提として、従来のデータ表現の変換を最低限に抑えるための仮想記憶管理と二次記憶管理を導入した。二次記憶の操作単位であったファイルに位置情報を付加し、仮想空間上の特定の位置を占めるリージョンを導入することによって、ポインタを含むデータ構造を直接二次記憶に格納することを可能にした。そして、2^{64}バイトのバージョン化された空間内で、リージョン間の位置の衝突を最小限にするよう全リージョンの位置を管理している。この部分は他の64ビットシステムと大きく異なる [1, 3]。この結果協調作業を行なう複数のプロセスは、ポインタを含む位置依存のデータ構造をデータ表現の変換なしで共有することができる。唯一の例外はひとつのデータ構造の異なるバージョンを同時に参照する場合であり、この場合Lucasはユーザーレベルのサーバーを用いてリージョンの再配置を行なう。

3　空間のバージョン化による再配置の回避

リージョンは、非永続的データ (UNIXにおける仮想空間上のセグメント) と永続的データ (UNIXにおけるファイル) とを統一的に扱うための概念であり、協調作業におけるデータ共有で、主記憶上のデータと二次記憶上のデータとを、データ表現を含めて透明に扱うために導入したものである。リージョンは資源空間と呼ぶ空間に格納される。

プロセスのリージョンへのアクセスは、資源空間からプロセスの仮想空間へのメモリーマッピングによって行なう。リージョンをプロセスの仮想空間上のどこへ張り付けるかは、メモリーマッピングの要求を行なうプロセスではなく、Lucasが決定する。各リージョンには、それが初めてプロセスの仮想空間にマップされる時点で、現在システム内にある他のリージョンと位置の衝突を起こさない位置が、システムによって割り当てられる。各リージョンの生成時のサイズはゼロであり、UNIXのファイルと同様動的に2^{32}バイトまで伸び縮みできる。このため各リージョンは2^{32}バイト境界に置かれるが、2^{64}バイトの空間ではこのような使用法でも、10,000台

Shigekazu Inohara, Keitaro Uehara, Hajime Miyazawa, and Takashi Masuda, "Support for Cooperation on 64-bit Address Spaces in the Lucas Operating System," Dept. of Info. Sci., Graduate School of Sci., Univ. of Tokyo.

規模のネットワーク上の全二次記憶を衝突なしでアドレスすることができる。

64 ビットの仮想空間、またはより広い仮想空間を用いても、リージョン間の位置の衝突をすべて回避することはできない。例えば、リージョンのコピー操作は現在のシステム上のファイルのコピーに当たるごく基本的な操作であるが、コピーの対象が位置依存のデータ構造であった場合、そのデータ構造のコピーはオリジナルと同じ位置に置かれないとデータ構造の意味、特にポインタの意味が破壊される。あるプロセスから別のプロセスへメッセージ通信によってリージョンを送信する場合も、メッセージが call-by-value 的なセマンティクスを持っているために、同様の状況が生ずる。この問題を解決するため、Lucas では、複製操作を行なう場合にのみ2つ以上のリージョンを同一の位置に置く。すなわち、ひとつのリージョンの複数のバージョンは資源空間上で単一の位置が割り当てられ、別々のリージョンには衝突しない位置が割り当てられる。このバージョン化された資源空間からプロセスへのリージョンのメモリーマッピングは、複数のバージョンを持つリージョンの特定のバージョンを選択する操作であると考えることができる。バージョン化した資源空間は、リージョンのコピーにおいてもメッセージ通信においても、単一のプロセスが同時にひとつのリージョンの複数のバージョンに対してアクセスすることが稀であることを利用したものである。

4 再配置処理のサポート

ひとつのリージョンの2つ以上のバージョンをひとつのプロセスが同時に使用する場合でも、そのリージョンが位置依存なデータ構造を格納していない場合には、そのリージョンの内容に一切手を加えずに資源空間の別の場所へ再配置することができる。リージョンが位置依存であるかどうかをシステムが把握するために、Lucas ではリージョン間参照を提供している。ユーザーは、あるリージョン A から別のリージョン B へのリージョン間参照を宣言することにより、この2つのリージョン間のポインタ参照の存在をシステムに教える。B を参照しているリージョンが存在しなければ、B は位置独立なリージョンとして、システムが自由に再配置できる。

位置依存のリージョンの2つ以上のバージョンをひとつのプロセスが同時に使用する場合、またその場合にのみ、それらのリージョンのどちらかを再配置する必要が生ずる。あるリージョンの再配置には、そのリージョンを指しているすべてのポインタを書き換えることが必要なるが、Lucas ではユーザーがリージョン間参照をシステムに宣言するため、再配置すべきリージョンを参照しているすべてのリージョンをシステムが決定できる。

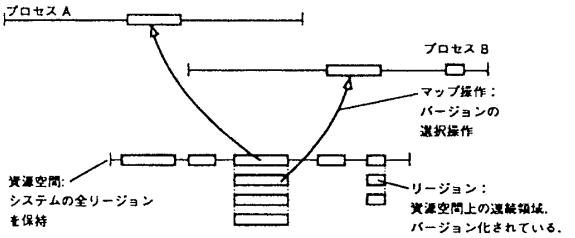

図 1: リージョンとバージョン化された資源空間

再配置のためのポインタ書き換え作業は、ユーザーが指定した再配置用のサーバープロセスへの upcall により実現される。このサーバーは、言語レベルまたはアプリケーションレベルの知識を用いて、指定されたリージョンを書き換える。一方、オペレーティングシステムレベルで単一のポインタ管理方式を導入し、すべてのアプリケーションをその方式に従わせることにより、システム側が再配置処理を直接行なう方法も考えられるが、アプリケーション毎の再配置処理の効率化を可能にするため、Lucas ではこの方法はとらなかった。すなわち、Lucas 自身はリージョン間参照を用いて、どのリージョンの再配置をいつ行なうか (policy) のみを決定し、実際の再配置処理 (mechanism) は各言語、各アプリケーションが提供するというものである。

5 現在の状況

現在 DECstation5000 上のプロトタイプで本稿で述べたすべての機能が動作している。このプロトタイプは Mach 3.0 マイクロカーネル上のユーザーレベルサーバー群であり、リージョンおよびリージョン間参照の操作部分は約 24,000 行の C 言語である。詳細な性能測定は現在進行中であるが、例えば、非永続リージョンのマップ＋アンマップは 3.2msec であり、これは同一マシン上の UNIX 4.3 BSD サーバー上のファイルのメモリーマップ＋アンマップ 4.3msec と比較しても遜色ない。

参考文献

[1] J. Chase, H. Levy, M. Baker Harvey, and E. Lazowska, "Opal: A Single Address Space System for 64-bit Architecture," IEEE WWOS-III, April 1992.

[2] 猪原 茂和、上原 敬太郎、宮澤 元、益田 隆司, "オペレーティングシステム Lucas における 64 ビットアドレス空間の管理," 6th SWOPP, August 1993.

[3] T. Okamoto, H. Segawa, S. H. Shin, H. Nozue, K. Maeda, and M. Saito, "A Micro Kernel Architecture for Next Generation Processors," USENIX Micro-Kernels and Other Kernel Architectures Workshop, pp. 83–94, April 1992.