異機種間プロセスマイグレーションの一手法

1P−3

長澤 育範　大胡 真明　相田 仁　齊藤 忠夫
東京大学 工学部

1　はじめに

プロセスマイグレーションは、実行中のプロセスを他のコンピュータに移動して、再配置する機能である．これは、分散システムにおいて、負荷分散、資源共有、可用性の向上などの目的でネットワーク内のCPU資源を有効に利用する技術として、研究されてきた．

同機種間のプロセスマイグレーションは、多くの分散OS上や、UNIX上で制限を設けた形などで実現されているが、これを異機種間で行なうには、同機種間の時にはない問題も解決する必要があり、さらに困難である．

筆者らは、この異機種間のプロセスマイグレーションを実現する方法について検討・考察し、制限事項は多いものの、UNIXワークステーション間でカーネルやコンパイラを変更せずに実現することに成功した．

本稿では、筆者らが考案した異機種間プロセスマイグレーションの方法について、その内容と実現した結果を制限条件と共に述べる．

2　異機種間マイグレーションの問題点

プロセスマイグレーションを行なう際には、プロセスの実行状態に関する各種の情報やプロセスのアドレス空間の内容を移動することが必要である．

同機種間のマイグレーションでは、プロセスのアドレス空間やレジスタの内容をそのまま移送先に移動して、実行を再開することで実現できる．

しかし、異機種では命令コードが異なり、レジスタ等も異なるため、移送して意味のあるものは基本的にデータの内容だけである．そのデータに関しても

- データの存在する領域の開始番地の違い
- データの割り付け規則（Alignment）の違い
- データの内部表現の違い（Endianなど）の違い
- アドレスを値として持つ変数（ポインタ）の存在

などがあるため、そのまま移送して、移送先で移送元と同じ所に読み込んでも意味をなさない可能性がある．

また、移送時の実行再開位置も、プログラムカウンタの値が意味をなさないため、別の方法で知らせなければならない．

以上のことから、異機種間でマイグレーションを行なうには、同機種間の場合には必要でない処理や変換を行なう必要がある．

"A method for heterogeneous process migartion"
Ikunori NAGASAWA, Masaaki OHGO,
Hitoshi AIDA, and Tadao SAITO
The University of Tokyo

3　異機種間でのデータ領域の転送方法

我々は、前節で述べた問題に対処する方法を検討し、以下に述べるようなアプローチを考案した．

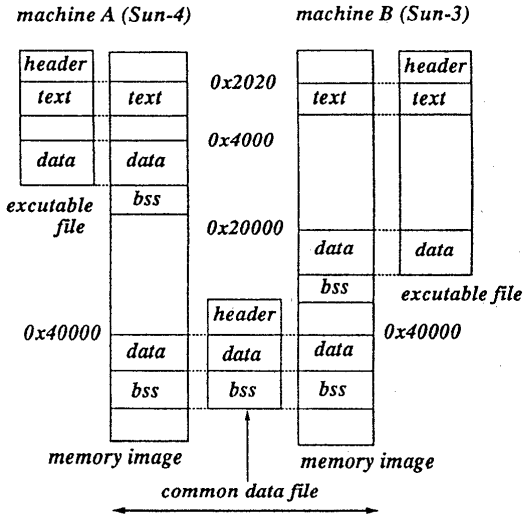

図 1: 本方式の概念

その概要は、

- ソースプログラム内で使う変数は基本的にグローバル変数に限定する．
- テキスト部と変数定義部を分離して、データ定義部から全ての機種のメモリ割り付け規則に合うようなデータ割り付けを行なうアセンブラルーチンを作成してコンパイル・リンクする．

  これによって、実行時にデータ領域が全機種で同一アドレスに割り付けられるような実行可能プログラムを作る．このコンパイルの手順は、図2のようになる．
- 移送時にはデータ領域をファイルとして書き出し、移送先で再開されたときにその内容をデータ領域に読み込む．
- 実行停止位置はあらかじめソースプログラム内ラベルを付けて限定し、実際に停止した位置に対応した値をデータ領域の特定の変数に設定する．移送先では、その変数の値によってどのラベルの位置から再開すればよいか決定し、gotoによってその位置に移動する．

というものである．

また、通常ファイルに対する処理として、

- ファイルはオープン時に“完全パス名”を、移送時に“変位”をデータ領域に保存しておき、そのファイルを移送先で再オープンし、変位を再設定する.

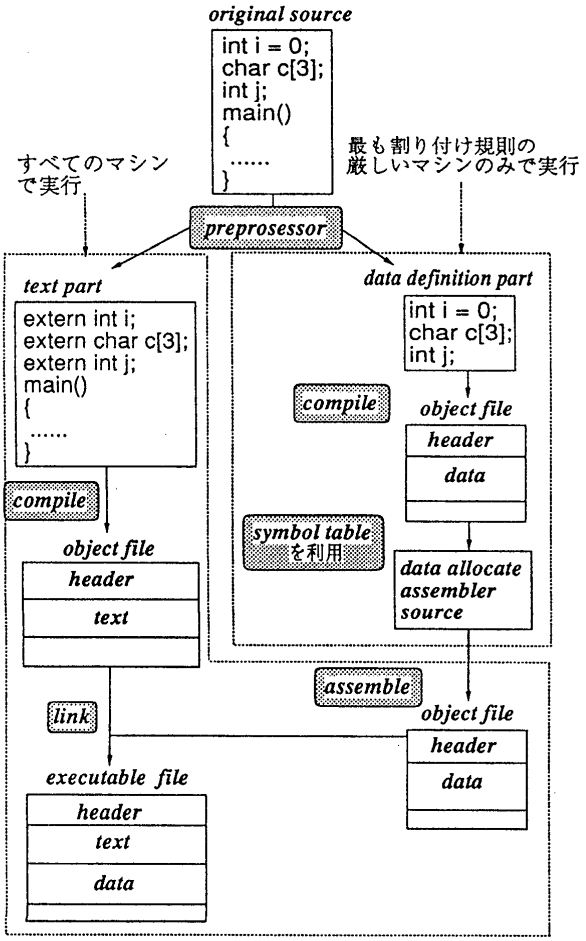

図 2: 本方式におけるコンパイルの手順

以上によって、データのメモリ割り付けの問題および実行再開位置の問題を解決でき、異機種間のマイグレーションが可能になる.

この方式では基本的にプリプロセッサやローダによって処理を行なうので、OS のカーネルやコンパイラの変更を行なわずに異機種間マイグレーションを実現できる.

4 本方式による異機種間マイグレーションの実装

上記の方式に基づき、Sun-3(Motorola MC68020) マシンと Sun-4(Sun SPARC) マシンの間でプロセスマイグレーションを実際に行なった. その際、ソースファイルを加工するプリプロセッサ、移送を行なうのに必要となるライブラリ関数、さらに移送時にプロセスを再開するサーバを作成した. 各プロセス間の通信にはソケットインタフェースを用いた. また、ファイルの受渡しについては、NFS が稼働していることを仮定した.

これを用いて、実際に数値計算等を行なうプログラムをいくつか実行してみたところ、いずれも正しい結果を得ることができた.

また、移送時のオーバヘッドに関しては、約 50KB のデータ領域を持つプロセスについて、Sun3/60 から SPARC station 2 へ移送を行なった時には、移送元でデータをファイルに書き出して移送をするまでに約 480 ms、移送先のサーバが fork/exec を行ない、データ領域を読み取って実行を再開するのに約 140ms, これにサーバ間の通信が加わって、全体で約 700ms 程度のオーバーヘッドであった. 逆方向の移送では、それぞれ約 420ms、250ms、全体で約 830ms であった.

5 現在の制限

ここで述べた方式で異機種間のマイグレーションが可能であることがわかったが、本方式にはいくつか制限が存在する.

上記の方法では、コンパイル時にデータを機種間で同一アドレスに割り付けつるようにするため、

- スタック領域に割り付けられるローカル変数
- malloc() 等で動的に得られた領域

には対処しておらず利用できない. (ただし、移送時の値がその後の実行に影響を及ぼさない場合のローカル変数は利用できる.)

また、パイプ、ソケット等のプロセス間通信やテキスト領域のアドレスを用いるシグナル処理関数の設定や関数へのポインタ等には対応していない.

6 まとめ

本稿では、プリプロセッサやローダを用いて、機種間でユーザプログラムのデータを同一アドレスに割り付けるようにすることで、OS のカーネルやコンパイラの変更等を行なうことなく、異機種間プロセスマイグレーションが実現できることを示した.

さらに一般的なプログラムにも対応できるように、malloc 等で動的に割り付けられる領域、関数内のローカル変数 (スタック領域) の移送法を現在考慮中である.

参考文献

[1] 森山茂男、多田好克：“カーネルの変更を伴わないプロセス移送の実現法”, The 18th jus UNIX Symposium Proceedings, pp.151-159 (November, 1991).