マルチメディア通信と分散処理　89－19
（１９９８．６．４）

64ビットUNIXの分散制御システムへの適用

相浦 利治*、倉持 和彦*、武曽 徹**
* 三菱電機(株) 情報技術総合研究所
**三菱電機(株) 電力・産業システム事業所

64ビットUNIXでは、仮想アドレス空間の最大値が16 E Byteまで拡大され、従来固定ディスク上に置いていたデータを、メモリ上に展開することで、システムのパフォーマンスを大幅に向上させることができる。しかしながら、ソフトウェアから64ビットアドレス空間を利用するためには、プログラムをLP64データモデルに適合させなければならない。また、大容量メモリを使用した場合、ディスクキャッシュのsync、メモリ空間の確保・解放などのカーネル処理が、システムのリアルタイム性を阻害する要因となる可能性がある。本稿では、64ビットUNIXを分散制御システムに適用する上での、課題と対策について示す。

Applying 64-bit UNIX to Distributed Control System

Toshiharu Aiura*　Kazuhiko Kuramochi*　Toru Muso**
* Mitsubishi Electric Corp. Information Technology R&D Center
**Mitsubishi Electric Corp. Energy & Industrial Systems Center

In 64-bit UNIX, the maximum value of the virtual address space is expanded to 16 E Byte. System performance can remarkably be improved by placing data on memory, which traditionally had to be on the hard disk. In order to use 64-bit address space from software, however, programs should suit the LP64 data model. Moreover, when using the large memory, kernel tasks, such as disk cache sync, memory allocation and free, can be the bottleneck for real-time performance. This paper describes the problems and their solutions of applying 64-bit UNIX to Distributed Control System.

1. はじめに

CPU性能の向上、及び、主記憶、2次記憶の大容量化、低価格化により、中,小型計算機においても数百M Byteオーダのメモリ、数十G Byteオーダの固定ディスク装置を搭載する機種が発表され、今後も増加する傾向にある。しかしながら、従来の計算機は32ビットプロセッサ、32ビットオペレーティングシステムをベースとしているため、連続して利用可能なデータサイズ、仮想アドレス空間の最大値は、4 G Byteであった。

現在、主なUNIX計算機ベンダから4 G Byteの壁をやぶる64ビットUNIXが発表されている[(1)(2)]。プロセッサ、及び、オペレーティング・システムを64ビット化することで、アプリケーションに対して、64ビットアドレス空間、4 G Byteを超える連続した2次記憶領域を提供する事ができる。また、拡大したメモリ空間の有効利用方法として、固定ディスクに置いていたデータ

をメモリ上に展開する VLM（Very Large Memory）といった技術も登場している。

一方、The Open Group が1996年に統一UNIX仕様[3] (Single UNIX Specification、以下SUS)を発表した。SUS によって、90 年代前半に活発化していたUNIXの標準化動向が一本化された。さらに、SUSの第2版が1997年に発表され、64ビットUNIXではLP64データモデル(long型、pointer型の64ビット化）の採用が決定された。これによって、64 ビット UNIX をシステムに適用するための地盤固めが完了したといえる。

以上のような背景から、今回、大規模分散制御システムに対して、64 ビット UNIX の適用を検討した。64 ビット UNIX では、メモリ空間が大幅に拡大する、また、主な UNIX 計算機ベンダーが標準仕様を採用していることから、大量のデータを使用する大規模システムに適用するメリットが大きい。本稿では、64 ビット UNIX を適用した分散制御システムを構築する際の課題と適用策について示す。

2. システム適用上の課題

大規模分散制御システムに対して 64 ビット UNIX を適用する際に、以下のような課題が存在する。

(1) 64ビットソフトウェアへの移行・開発
64 ビット UNIX では、プログラムから拡大したメモリ空間を扱うため、LP64 データモデルが採用されている。よって、従来の 32 ビットソフトウェアを LP64 データモデルに適合させる必要がある。

(2) 32/64 ビット混在システム設計
分散制御システムでは、データベースなど第三者ソフトウェアを使用する場合がある。現在は64ビットUNIXの黎明期であるため、一部の第三者ソフトウェアは、64 ビットに対応していない。よって、システム内に32/64ビットのソフトウェアが混在する構成でシステムを設計する必要がある。

(3) 大容量メモリ空間に対応したシステム設計
ソフトリアルタイム性（GUI など秒オーダの応答性能）が要求される分散制御システムに、大容量メモリを搭載した 64 ビット UNIX 計算機を適用した場合、システム性能を安定化させるため、大量データの I/O や、カーネルのメモリ管理に関連する処理の影響を受けないように、システム設計を行う必要がある。

以下に、詳細について示す。

2.1. 64 ビットソフトウェアの移行・開発

32 ビット UNIX では、ILP32 データモデルが使用されている。ILP32 とは、Int, Long, Pointer データ型が 32 ビットであることを表している。64 ビット UNIX では、LP64 データモデルが使用される。LP64 とは、Long, Pointer データ型が64ビットであることを表している。

表 1：ILP32 と LP64 データモデルの差異

データモデル	データ型		
	int	long	pointer
ILP32(32 ビット)	32	32	32
LP64 (64 ビット)	32	64	64

データ型サイズの変更に伴い、32 ビット UNIX で開発したプログラムは、64 ビット UNIX 上で再コンパイルしても、コンパイル中にエラー、ワーニングメッセージが表示される。または、生成されたバイナリ・モジュールが正しく動作しない可能性がある。以下にプログラム例を示す。

(1) 異なるデータ型間での代入

```
int   a;
long  b;
 ......
a = b;
```

long 型は、64 ビットである。よって、b の値が INT_MAX よりも大きいならば、32 ビットの int 型に代入しようとした場合、上位 32 ビットが切り捨てられる。

(2) データの格上げの影響
データ格上げルールの影響により、32 ビットと 64 ビット間で、以下のような if 文の

処理結果が異なる。

```
long l_data = -1;
unsigned int  i_data = 1;
if  (l_data  >  i_data)
     printf("l_data > i_data¥n")
else
     printf("l_data < i _data¥n")
```

上記プログラムを 32 ビット環境で実行した場合は、"l_data > i_data"を出力する。64 ビット環境では、 "l_data < i_data"を出力する。

(3) ポインタの扱い

メモリ上に確保したデータのオフセット値などを算出する際に、int 型変数にアドレスを代入していた場合、アドレス情報の上位 32 ビットが切り捨てられ、正しく動作することができない。また、int 型変数の符号ビットが立っていた場合、pointer 型に代入するタイミングで符号拡張が行われる。

```
int  tmp;
void *p1, *p2;
 ......
tmp = p1;
 ......
p2 = tmp;
```

(4) 構造体フォーマットの変更

構造体内に long 型、pointer 型が定義されていた場合、64 ビット UNIX 上で、64 ビットバウンダリでアラインメントされるため、ILP32 環境でコンパイルした構造体のサイズと、LP64 環境でコンパイルした構造体のサイズ、フォーマットが異なる。

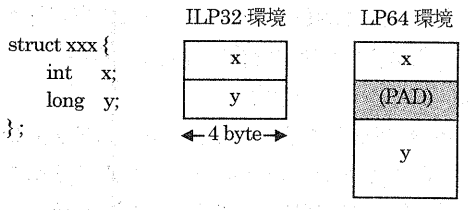

図 1：構造体フォーマットの変更

よって、long 型を含む構造体を、32/64 ビットプロセス間での通信、または、共有されるファイルのデータブロックに用いていた場合、正しくデータを共有することが出来ない。

(5) 定数値のハードコーディング

long 型が 32 ビットであることを前提に記述されているプログラムは、64 ビット環境では動作しない。以下の例は、long 型変数 b の符号ビットを設定する処理である。64 ビット UNIX 上で long 型の符号ビットを設定するためには、63 ビットシフトする必要がある。

```
long b;
 ......
b = 1 << 31;
```

2.2. 32/64 ビット混在システム設計

分散制御システムは、システムの信頼性を向上させるための 2 重系制御やファイル管理などの機能を提供するミドルウェアと、実際の制御処理を行う応用アプリケーションから構成されている。また、一部に第三者ミドルウェアを使用する場合もある。

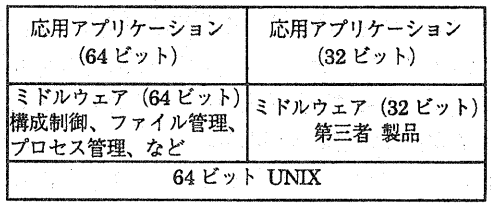

図 2：32/64 ビットの混在

64 ビット未対応ソフトウェアのために、32 ビット、64 ビットのソフトウェアをそれぞれ開発することは現実的ではない。32/64 ビット双方の環境でコンパイル可能なソフトウェア（以下、64 ビットクリーンなソフトウェア）をいかにして開発するかが課題である。

2.3. 大容量メモリ空間への対応

64 ビット UNIX では、4 G Byte 以上の主メモリを実装できるようになる。また、仮想アドレスも 4 G Byte 以上の空間を利用できるようになる。これらは性能向上には極めて有効であるが、以下に示すような問題を引き起こす可能性がある。

(1) プログラム異常終了時における問題点

プログラムが異常終了した場合には、core ファイルが生成されるが、1 G Byte のデータを使用しているプロセスの場合には 1 G Byte の core ファイルが生成されることになる。しかし、この core ファイルの生成は、①ディスクスペースの圧迫、フル状態の発生、②大容量書き込みによるディスク負荷の集中、③プロセスのメモリ空間の解放処理によるオーバヘッドによって、オンライン処理の性能に影響を与える場合がある。

(2) ディスクのボトルネック化

オペレーティングシステムの設定によっては、バッファキャッシュのサイズを、搭載する主メモリサイズに対する比率で決定する場合がある。バッファキャッシュサイズが、主メモリに対して50%の設定であり、かつ、2 G Byte のメモリを搭載した場合、バッファキャッシュのサイズは 1 G Byte となる。これは、定期的な syncer の起動などのタイミングで、1 G Byte の書込み要求が一気にディスクに対して行われることを意味する。ディスクに対して大量の I/O 要求が溜まっている状況では、同一ディスクに対する異なるプロセスのアクセス要求が長時間待たされるため、性能障害が発生する可能性がある。

(3) システムクラッシュ（パニック）時における問題点

システムクラッシュ時には、以下の2点の問題が発生する。

① ディスクキャッシュ上のデータの喪失
ディスクキャッシュ上のデータが喪失し、回復に時間がかかる。この時間はディスクキャッシュの量に比例して大きくなる。

② クラッシュデータのセーブ処理の長時間化
一般的な UNIX システムでは、システムクラッシュが発生した場合には、主メモリデータをディスクに書き込み、リブート後にそのデータを解析のためにファイルシステムに保存する。例えば1 G Byte の主メモリを搭載していた場合、計算機のダウンタイムは数十分程度の時間がかかることになる。

3. システムに対する適用策

3.1. ソフトウェア開発環境の整備

2.1に示した通り、64 ビット UNIX をシステムに適用するためには、プログラミングのレベルにまで立ち入って開発環境を整備する必要がある。そのための具体的な作業としては、以下のようなものがある。

(1) 64 ビット クリーンなソフトウェアを開発するためのルールの明確化

2.1で示したLP64 適用上の問題点

① 異なるデータ型間での代入
② データ格上げへの対応
③ ポインタの扱い
④ 構造体サイズの等価
⑤ 定数値の取り扱い

を整理し、ソフトウェア開発を行うためのルールの明確化を行った。また、SUS への準拠という観点からソフトウェアを調査し、非準拠インターフェースを使用していた箇所を、SUS 準拠インターフェースに修正した。

(2) ソフトウェア開発ツールの整備

開発したソフトウェアが64 ビットクリーン、及び、SUS に準拠しているか検証するためのツールの整備を行った。今回使用したツールは、lint コマンドの強化版であり、環境ファイルにより、データ型サイズ、出力メッセージの制御が可能である。環境ファイルによって、LP64 データモデルへの対応、及び、64 ビット UNIX 上でソフトウェアを実行させた場合の問題点を検出可能である。

(3) 開発ルールの教育

64 ビットクリーンなソフトウェアを開発するための開発ルールや、ツールの使用方法に関して、関連会社を含めたプログラマ、SE に対して教育を行った。

3.2. 大容量データの取り扱い方法

リアルタイム性が要求される処理は、64 ビット UNIX の特長である大容量メモリ空間を生か

し、メモリ上に展開したデータのみアクセスするように設計することで、パフォーマンスを向上させることができる。しかしながら、ディスクに対するI/Oや、カーネル処理に伴う遅延の影響を受けないようにするため、以下の対応を取る必要がある。

(1) I/O処理の分割

ディスクなどI/Oが伴う処理は、メモリを介してプロセス間通信によって、別プロセスに処理させる必要がある。これによって、ディスクキャッシュのsyncなどに伴うアクセス遅延の影響を回避することができる。

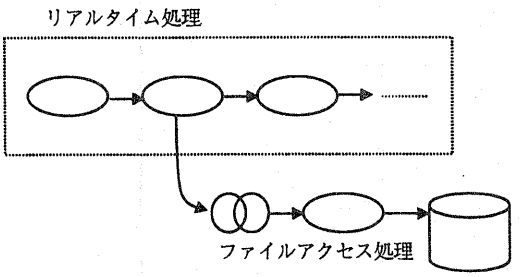

図3：I/O処理の分割

(2) 大容量メモリ空間への対応

プロセスの起動・終了時のメモリの確保・解放処理に伴うオーバヘッド、及び、異常終了時のcoreファイルの生成を防止するため、大容量データは上記の影響を受けない共有メモリ上に展開する必要がある。共有メモリ領域は、システムの初期化処理で確保した後、システムの稼働中に解放しないように設計する必要がある。

また、システムクラッシュに対しては、従来から2重系システムで対応している。しかしながら、システムの高度化、高機能化要求に伴って、搭載する主メモリは増加していくことが予想される。ダウンタイムの増加の問題に対しては、多重系システムによる対応を検討していく必要がある。

4. おわりに

本稿では、64ビットUNIXを大規模分散制御システムに適用する上での課題と解決策について示した。

(1) 64ビットUNIX上で動作するソフトウェアを開発するためには、64ビットクリーン化のためのルールを明確化させる必要がある。また、64ビットクリーン化、SUS準拠インターフェースの仕様を検証するツールを導入することで、ソフトウェアの開発効率を向上させることができる。これらの開発ルール、ツールの使用方法などは、ソフトウェア開発担当者に対して教育を行う必要がある。

(2) ソフトリアルタイム性が要求される処理は、I/Oに伴う遅延の影響を受けないように設計する必要がある。また、大容量データは、ソフトウェアの起動・終了の影響を受けない共有メモリ領域に展開する必要がある。

今後は、多重系システムを構成した場合の、計算機間のデータ等価方式や、大容量メモリ空間を利用した応用ソフトウェアの高度化について検討していく。

参考文献

(1) 清兼 編著：64ビットUNIX&CDE，共立出版，1997

(2) HP：http//www.hp.com/gsy/software/64bit/64bitwp.html

(3) TheOpenGroup:http://www.opengroup.org/regproducts/complist.htm