メモリを高効率で活用するメモリ回収機構

2F－1　石井秀浩　高野陽介　黒岩実　横田実
NEC C&C研究所

1　はじめに

セットトップ・ボックスや携帯端末などの小型の計算機においては、多数のプログラムを同時に動作させる事や、多量の記憶領域を消費するプログラムを動作させる事は、困難である。これは、このような計算機では、小型化や価格抑制のために二次記憶装置を持たないかその容量が少ない[1]場合が多く、また主記憶の容量も少ない場合が多いからである。ネットワークを介したユーザ間の協調やマルチメディア処理等の高い機能をこのような計算機に持たせようとする場合は、記憶資源[2]の不足は深刻な問題になる。

そこで、限られた記憶資源を有効活用する仕組みを提供するのが、本論文で提案するメモリ活用機構である。この機構により、計算機全体としてより多くの仕事をしたりより効率的に仕事をしたりすることが可能になる。

2　メモリ活用のコンセプト

プロセス等は、メモリ資源を余分に独占している事が多い。例えば、C言語のmalloc()関数やC++言語のnew演算子で割当てを受けたメモリは、free()関数やdelete演算子でアプリケーションが解放しても、普通、ライブラリが管理しているメモリ・プールに蓄えられてしまい、後のmalloc()やnew等による再割当てのために保留される。つまりそのプロセスが終了するまで、それらのメモリ資源はOSには返されず、他のプロセス等が利用する事はできない。また、ライブラリに任せないで自らメモリ・プールを確保・管理し、使わなくなったメモリを独自のメモリ・プールに保留しておくアプリケーションもよくある。

本方式では、この様な余剰メモリ資源を活用するため、システムの空きメモリが少なくなると、メモリ使用者（プロセスや、カーネル内のモジュール）に余剰メモリ資源の解放を依頼する。メモリ使用者は依頼を受けると、自分が確保しているメモリ資源のうち余剰のものをシステムに解放する。

本方式によれば、システムの空きメモリに余裕がある時には、従来と同様に、ライブラリやアプリケーションのメモリ・プール内の余剰メモリ資源はそのまま保留されるので、解放処理のオーバヘッドがかからず、また将来のメモリの再獲得のオーバヘッドも最小限ですむ。そしてシステムの空きメモリが不足している時は、ライブラリやアプリケーションのメモリ・プール内の余剰メモリ資源がシステムに解放され、メモリ資源を有効活用することができる。

3　実現方式

前節に示したコンセプトを実現するにあたり、二つの仕組みを提案する。

一つは、ライブラリのメモリ割当て機構のメモリ・プール内の余剰メモリを必要に応じて回収する仕組みである。この仕組みは、アプリケーションの変更を必要としない。

もう一つは、アプリケーションが確保しているメモリを必要に応じて回収する仕組みである。この仕組みを利用するためにはアプリケーションを若干変更する必要があるが、ライブラリ内の余剰メモリだけを対象にするよりも多くのメモリを活用する事ができる。

3.1　ライブラリ内の余剰メモリの回収

C言語とC++言語の標準ライブラリのメモリ割当て機能を改造し、ライブラリのメモリ・プール内のメモリ資源を活用する機構を実現する。つまり、OSからの解放依頼をライブラリが受け、ライブラリが管理しているメモリ・プールのメモリ資源を、依頼に応じてOSに返却する。これは、ライブラリが内部で自動的に行なうものであり、アプリケーションには透過である。

3.2　アプリケーションが確保しているメモリの回収

アプリケーションが独自に確保しているメモリ資源の余剰分を活用するには、アプリケーションの協力が必要になる。こちらの仕組みでは、システムの空きメモリが少なくなった時、図1のように、ライブラリだけでなくアプリケーション本体にも余剰メモリの解放の依頼を行なう。アプリケーションはこの依頼を受けると、キャッシュ領域の縮退やガーベジコレクション等を行なって余

Efficient Utilization of Limited Amount of Memory.

Hidehiro Ishii, Yosuke Takano, Minoru Kuroiwa, and Minoru Yokota.
C&C Research Labs, NEC Corporation.

[1]二次記憶が十分あれば、ディマンド・ページングによって大きな記憶領域をプログラムが利用できる。

[2]主記憶、または主記憶とページング用の二次記憶で作られる仮想的なメモリ＝記憶資源を、次節以降では便宜上単に「メモリ」と呼ぶ事にする。

剰メモリを解放する。具体的にはアプリケーションは、malloc() や new によって獲得したメモリに余剰がある場合は、依頼を受けたらそれを free() や delete によってライブラリに返還し、ライブラリはこれを OS に返還する。またアプリケーションがこれらのライブラリを介さず独自に OS コール[3]によって OS からメモリ資源を獲得して管理している場合は、依頼を受けた時に OS コール[4]でメモリを解放する。

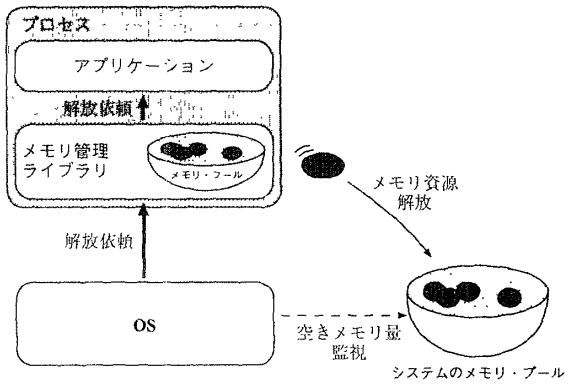

図 1: アプリケーションの協力

4 RT-Mach での実装

今回、Real Time Mach マイクロカーネルと Lites[5]の組合せの環境で、本方式を実装した。この環境では、ディマンド・ページングが行なわれるので、本論文の仕組みで管理する「メモリ資源」はページング領域である（つまり、限られたページング領域の有効活用を行なう事になる）。

ページング領域を管理するタスク「デフォルト・ページャ」の改造と、メモリ割当て機能をアプリケーションに提供する代替ライブラリの作成を行なった。本代替ライブラリは、C 言語の標準ライブラリの malloc(), realloc(), free() を置き換える。C 言語で書かれたアプリケーション、および malloc() を使って new が実装されている C++ 言語処理系（GNU の C++ など）で書かれたアプリケーションは、本ライブラリをリンクするだけで 3.1節の動作を行なうようになる。

代替ライブラリは、アプリケーションの起動時に自分のポートをデフォルト・ページャに登録する。デフォルト・ページャは、ページング領域を監視し、空き領域が少なくなると登録された代替ライブラリの各ポートに対して余剰メモリの解放を依頼する。代替ライブラリは、解放依頼を受けると、ライブラリ内のメモリ・プールの空き領域を OS に解放する。

さらに、アプリケーション内のメモリ解放処理関数を登録する API を代替ライブラリに設けた。登録しておくと代替ライブラリは、OS から解放依頼を受けた時、その登録されている関数を呼び出す（これがアプリケーションへの解放依頼である）。メモリ解放処理関数は、アプリケーション内でガーベジ・コレクション等を行ない、ライブラリにまたは直接 OS に、余剰メモリを解放するものである。

[3]UNIX の brk() や sbrk()、Mach の vm_allocate()、Windows の VirtualAlloc() 等。

[4]UNIX の brk() や sbrk()、Mach の vm_deallocate()、Windows の VirtualFree() 等。

[5]Helsinki University of Technology で開発された、4.4BSD Lite ベースの UNIX パーソナリティ・サーバ。

5 状況と効果

Lisp や Java の処理系のように、多くのメモリを動的に割当て・解放するようなプログラムが動作している環境では、その時々に使っていないメモリ資源のガーベジ・コレクションにより、本論文のメモリ活用機構の効果が期待できる。また、フォントをキャッシュするウィンドウ・システムのように、大きなキャッシュを持っているプログラムが動作している環境でも、そのキャッシュの大きさを必要に応じて増減することにより、大きな効果が期待できる。

6 おわりに

本論文では、メモリプール内のメモリ資源とアプリケーションが直接管理しているメモリ資源の余剰分の有効活用について述べた。

実際のシステムにおいて本方式がどの程度の効果を持つか定量的に評価する事が、今後の課題である。

参考文献

[1] Phillippe Bernadat and David L. Black. "Real Memory Mach", In *Proceedings of the USENIX Mach Workshop*, 1993

[2] Richard Rashid, Avadis Tevanian, Michael Young, David Golub, Robert Raron, David Black, William Bolosky, and Jonathan Chew. "Machine-Independent Virtual Memory Management for Paged Uniprocessor and Multiprocessor Architectures", In *ASPLOS II*, Oct., 1987

[3] Michael Young, Avadis Tevanian, Jr., Richard Rashid, David Golub Jeffrey Eppinger, Jonathan Chew, William Bolosky, David Black, and Robert Baron. "The Duality of Memory and Communication in the Implementation of a Multiprocessor Operating System", In *Proceedings of the 11th Symposium on Operating Systems Principles*, Nov., 1987