マルチプロセッサ対応のリモート・シンボリック・カーネルデバッガ

2G−9

湯原雅信[1]，井森一樹[2]，岸本光弘[1]

(1：富士通研究所　2：富士通AEL)

1．はじめに

オペレーティング・システムのカーネルを開発する際，そのデバッグは頭の痛い問題である．実際，我々がMach-OSを移植した時にも，カーネル内部の動作を簡単に把握することができず，苦労した経験がある．そこで，カーネルのデバッグを一般のアプリケーションのデバッグと同じくらい簡単に行えるようにカーネルデバッガを開発した．

2．KGDBの概要

本カーネルデバッガは，次のような特徴を持っている．

- マルチプロセッサもターゲットとしている．
- 快適なシンボリックデバッグが行える．
- ターゲットシステムをできる限り変更しないし使用しない．デバッガ本体が大量のメモリを消費しても問題ない．
- リモートデバッガなのでカーネルが不安定な時でも使える．
- 特殊なハードウェアを必要としない．

図1に示したように今回のターゲットマシンは68030の密結合4マルチプロセッサ・システム[1]（OSはMach2.5/3.0）であり，ここで通信用の小さなモニタを動作させる．デバッガ本体が動作するホストマシンはSparcStation2である．2つのマシンの間をRS232Cで結んでいる．イーサネット関連のソフトもデバッグ対象であるため，デバッガ用通信にイーサネットを使用することはできない．デバッガ本体には，GNU Debugger[2]を拡張して使用した．このため，本デバッガをKGDBと呼んでいる．

KGDBでデバッグするには，C言語のソースプログラムをコンパイルする時に，通常のシンボリックデバッグの場合と同じように，-gオプションを指定する．その結果をリンクしてできあがるカーネルは，例えば，ACE/MACH-2.5の場合，約5MBにもなる(mach.dbg)．このファイルは，KGDBが参照するのでホスト上に必要である．ターゲットシステム上の実行バイナリには，詳細デバッグ情報を除いて小さくしたファイル(mach)を使用する．

3．KGDBのユーザビュー

ユーザから見たKGDBの具体的な特徴について述べる．

●シンボリック・デバッグ

カーネルのCで書かれた部分のデバッグは，アプリケーションプログラムをシンボリック・デバッグするのと同じ感覚で行なえる．Cの式をそのまま指定できるので，構造体の中の各要素をプリントしたり，ポインタをたどっていくなど自由自在に行なえる．アセンブラで書かれた部分でも，命令アドレスからソースプログラムでの位置を簡単に調べられるし，逆アセンブルでは参照しているアドレスをラベル名で表示できる．

●リモート・デバッグ

KGDB本体は，安定したホストマシンで動作しているため，ホストマシンの快適な環境を享受できる．今回は特別なGUIのサポートは行なっていないが，emacsの中からKGDBを使えるだけでも格段に使いやすくなっている．ターゲットマシン組み込みのデバッガでは望めなかった点である．また，ターゲット側の小さなKGDBモニタさえ壊れていなければ，たとえカーネルが暴走したりデッドロックに陥っていても，カーネルを停止して調査することができる．

●中断・切り離し

ホストマシンでCtrl-C (SIGINT)を入力することにより，ターゲットマシンでカーネルが走行中でもカーネルを停止させることができる．そこから，実行を再開することもできるし，一旦KGDBをターゲットから切り離すこともできる．

●マルチプロセッサの取り扱い

デバッガモードにいる時には，全てのCPUが停止している．ユーザは，選択CPUを切替えながらデバッグを進めていく．ほとんどのKGDBコマンドは，選択CPUに対するものになる．実行を再開する"continue"コマンドでは，全CPUの実行再開，別コマンドで指定したCPUグループの実行再開，および，選択CPUのみの実行再開を行える．今回対象にしたのが共有メモリ型であるため，ブレークポイントはすべてのCPUに対して設定することになる．

●低レベル機能の追加

カーネルのデバッグに必要な次のような低レベル追加機能を利用できる．これらのコマンドの引数には，Cの式を指定できるため，従来の低レベルデバッガより使いやすくなっている．

- 論理および物理アドレスによるメモリダンプや変更
- アドレス変換テーブルのシンボリックな表示
- アドレス変換テーブルに従ったアドレス変換
- 特殊レジスタのシンボリックな表示

●OSコマンドの取り込み

OSにはカーネルの内部状態を表示するためのコマンドが用意されている．これらのコマンドをKGDB内部から利用できると便利である．今回は，Machの"ps"コマンドと同じ仕様の"ps"コマンドをKGDBから使用できるようなっている．

●ターゲットメモリのキャッシュ

高速化のため，ターゲットのメモリをKGDBの中でキャッシュするようにした．キャッシュしては困るI／Oレジスタなどの対策として，コンフィギュレーション・ファイルで指定したアドレス範囲をキャッシュ不可とすることや，KGDBのコマンドで全領域をキャッシュ不可にすることができる．KGDBが

A Remote Symbolic Kernel Debugger for Multi-processor Systems

Masanobu Yuhara[1], Kazuki Imori[2], Mitsuhiro Kishimoto[1]

1: Fujitsu Laboratories Ltd. 2: Fujitsu Aichi Eng. Ltd.

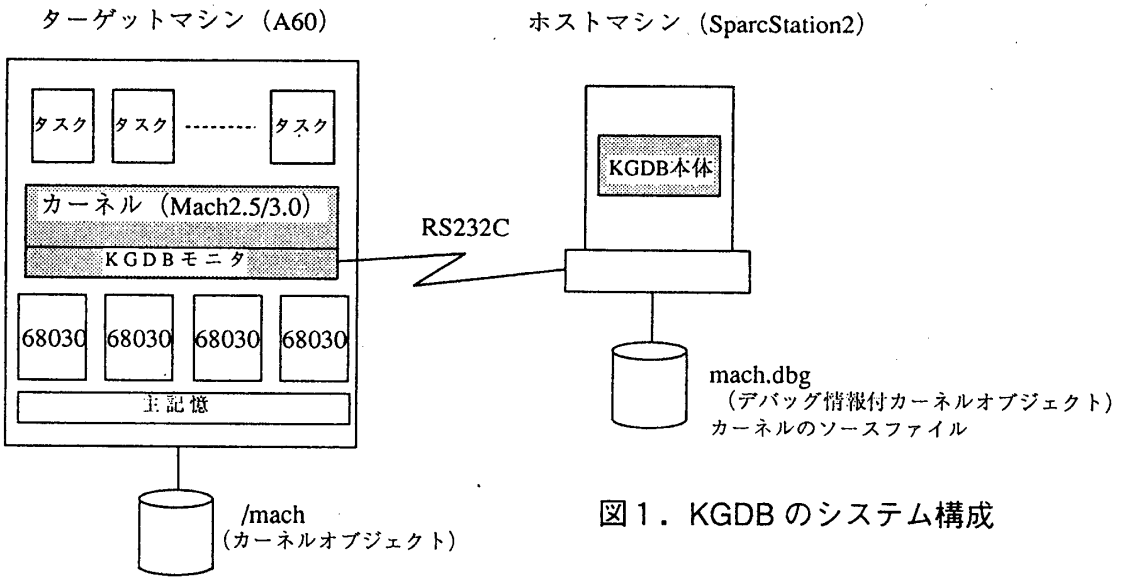

図１．KGDBのシステム構成

ターゲットシステムに制御を渡す時，このキャッシュは自動的に無効化される．

4．KGDBの実装

KGDBの実装に関していくつかの点に触れる．

デバッグ中に割り込みによる副作用が出ないように，KGDBモニタは割り込み禁止モードで動作させる．

●RS232C割り込み

カーネルの実行を中断できるように，カーネル実行中にKGDB本体で受けたCtrl-C (SIGINT)は，RS232Cへの出力に変換される．これでターゲットマシンのカーネルに割り込み，KGDBに制御を移す．どんな場合でも割り込めるように，NMIを使用している．ターゲットの特定のCPUがKGDB本体との通信を行う．他のCPUがKGDB本体と通信する時は，メモリを介してそのCPUに依頼して行う．

●CPU間割り込み

いずれかのCPUでブレークポイントに達したり，RS232Cからの割り込みがあった場合には，そのCPUのKGDBモニタは，選択CPUになるためにロックをとる．ロックがとれた場合には，選択CPUになり，急いで他の動作中のCPUに割り込みをかけ，KGDBモニタの制御に移行させる．このCPU間割り込みもNMIである．他のCPUを止めないと，デバッグ中にメモリの内容が書き換えられてしまうからである．

●多重例外

ブレークポイントヒットの例外，RS232Cの割り込み，CPU間割り込みが，多重に発生することに対処する必要があった．例えば，複数のCPUでほぼ同時にブレークポイントにヒットすることがある．この場合，最初にロックを獲得したCPUが選択CPUとなる．ユーザにより選択CPUが，ブレークポイントヒットしていた他のマシンに切り替えられると，その時点でブレークポイントヒットを通知する．ユーザが実行を再開しようとした時，実行再開しようとしたCPUの中にまだ通知していないブレークポイントヒットがあれば，実際には実行を再開しないで，そのCPUを選択CPUにしてヒットを通知することにした．

●psコマンド

ＯＳコマンドの例として，"ps" コマンドを "ps" のソースプログラムをなるべく変更せずに，KGDBに組み込んでみた．このため，"ps" が使用しているシステムコール（tableシステムコールなど）を，KGDBでターゲット・システムにアクセスしながらシミュレートしている．"ps" のソースの中で，ターゲットのカーネルの構造体を参照しているところは，機械的にKGDBのシンボル参照関数に変換した．以上により，非常に簡単に "ps" コマンドを組み込むことができた．他のＯＳコマンドも同様の手法によりKGDBに組み込むことができる．

5．将来の拡張

共有メモリ型マルチプロセッサの１つのプロセッサをKGDB専用にして，常にKGDBコマンドを実行可能したり，ユニプロセッサでも定期的にKGDBモードにしてメモリの内容を監視するなどの拡張が考えられる．また，今回のKGDBは共有メモリ型マルチプロセッサを対象としたが，共有メモリを持たないマルチプロセッサにも適用可能である．

6．おわりに

本稿ではリモート・シンボリック・カーネルデバッガKGDBについて報告した．KGDBモニタの開発でも，開発が進むにつれて，KGDBで自分自身のデバッグが行えるようになり，効率よくデバッグできた．コンソールでXwindowサーバが動作している時や，他マシンからrloginされている最中でも，カーネルのデバッグを快適に行えるようになった．もはやKGDB抜きのカーネルデバッグは考えられない程である．

参考文献

[1] 住元他，"Limited Symmetry方式を用いた並列ＯＳの試作"，CPSY89-80, 電子情報通信学会, 1989

[2] "Using GDB: A Guide to the GNU Source-Level Debugger (for GDB version 4.3)", Free Software Foundation, 1991.