4V-10

ＯＳ／omicronにおけるデバッギングツール －言語Ｃインタプリティブデバッガ－

田中泰夫、中川正樹、高橋延匡
(東京農工大学　工学部　数理情報工学科)

１．はじめに

我々の研究室では我々自身の研究をサポートする計算機システム（System/omicronおよびOS/omicron）の開発を進めている。このシステムでは基底言語として言語Ｃを選択し、言語Ｃコンパイラcatをはじめほとんどの基本ソフトウェアを言語Ｃによって記述している。一方、その開発過程を支えるデバッガとしては開発システムとして用いているCP/M-68Kのユーティリティであるddt-68K、および本研究室で作成されたDEBU[1]などを使用している。しかし、どちらもアセンブリ言語レベルのデバッガであるため言語Ｃのソースプログラムとの対応が取りにくいということが指摘されていた。そのため、言語Ｃと対応したデバッガを開発した。

本デバッガは、ソースプログラムとの対応、およびポインタ型に対する厳重なチェックという特徴を持つ。

２．開発方針

２．１ ソースプログラムレベル

高水準言語で記述したプログラムとコンパイラの出力したコードとの対応を取るのは困難なことである。また対応をとることが比較的容易にできても、アセンブリ言語レベルでデバッグするのは一般に能率が悪い。つまり、高水準言語で記述してもアセンブリ言語レベルでデバッグするのでは、生産性の向上は望めない。そこで本デバッガでは次のことを方針とした。

(1) 関数、データに対しての操作･参照は、ソースプログラムで記述される識別名を用いる。
(2) 制御構造を持つ文レベルまでソースプログラムとの対応を取る。
(3) 構造体メンバ名までのシンボル化を行なう。

２．２ シンボリックなポインタ値

ポインタ型によるアドレス操作はオブジェクト効率がよくなるといった利点がある一方、誤った操作を引き起こす場合がある。ポインタ型がプログラム全体を、場合によっては全メモリ空間を一つの配列のように扱えるためである。また、アセンブリ言語レベルにおいてポインタ値(アドレス)は処理系依存のアドレスを使用しているが、ユーザはポインタ値として識別名を思い浮べる。

そこで、本デバッガでは図１のように識別名とその中のオフセットでポインタ値を表わすことにした。この表現によりデータのレンジチェックを容易に行なえるため、ポインタ型による誤った操作を発見することができる。

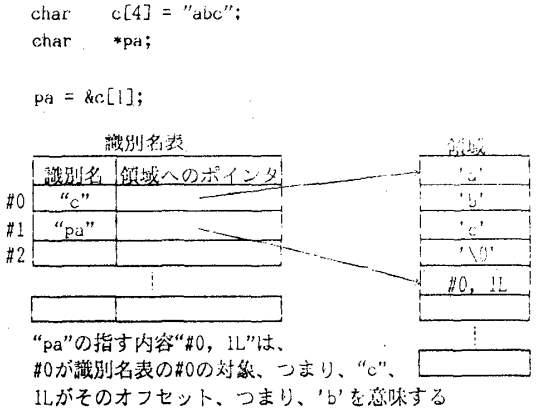

図１　シンボリックなポインタ

２．３ 単体デバッグ機能

言語Ｃではエントリポイントを与えるために、関数 main から実行を開始するようになっている。このため必ず関数 main を付加してデバッグしなければならないが、デバッグ時にはデバッグのための多種多様なデータを設定しなければならない。そのため関数 main を付加するだけといっても多大な手間を要する。

そこで、関数 main がなくても実行できる単体デバッグ機能の実現を目指した。

２．４ 動的フロー解析

プログラム中にブレークポイントをかけてデバッグする方法は大抵のデバッガに備わっている。ところがどんな制御でそのブレークポイントを通過したかはわからない場合が多く、プログラムの実行状態が不明でありデバッグ効率を低下させる。

そこで、プログラムの動的フローのための情報をデバッガ内の表やファイルに取っておいてユーザが解析できるようにする。

Debugging Tool for OS/omicron -The Interpretive Debugger for Language C-
Yasuo TANAKA, Masaki NAKAGAWA and Nobumasa TAKAHASHI
Tokyo University of Agriculture and Technology

3．実現

3．1 言語Cコンパイラcatとの関係

本デバッガは図2に示す言語Cコンパイラcatのパーザを共有する形で構成されている。これは、文書化ツールへの応用を含めて、 cat第一版をソフトウェア工学の見地から検討した結果の構成である。

3．2 式の投入

本デバッガではデータに対する操作・参照を含め、次のことに対しても式を投入してもらうことで実現した。

(1) データに対する操作･参照
(2) 単体デバッグ時の関数コール
(3) ブレークポイントの設定

これは、言語Cの文法において関数コールが一次式に分類されること、および式の結果としてポインタ値(アドレス)が許されることから式の形式に統一した。ところで、投入された式は一旦、中間コードに変換して実行している。これは、本デバッガ自体が中間コードをインタプリットしていることから比較的実現が容易であったこともあげられるが、投入された式にエラーがあった場合に実行をキャンセルすることができるためである。

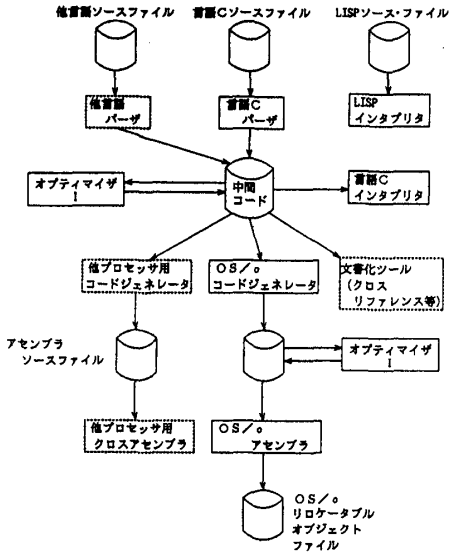

図2　OS/o言語処理系の概要

3．3 未定義シンボル

単体デバッグにおいて参照される全ての関数やデータが定義されているとは限らない。つまり未定義シンボルが現れることになる。その解決方法を関数、データの場合に分けて述べる。

(1) 未定義関数

関数が未定義の場合にはその関数がライブラリ関数である場合と、ユーザ自身が定義するつもりのものとに分かれる。前者についてはデバッガが持つライブラリ関数とリンクすることで解決した。後者については関数コールが起きた時点でデバッガに制御を移し、リターン値を返せるようにした。

(2) 未定義データ

データが未定義の場合には、デバッガ内の表にデータ領域を定義してもらいその領域とリンクすることで解決した。この表をワークデータ表と呼ぶ。定義方法はソースプログラムで記述するようにして行なう。 この時、デバッグ対象となるプログラム中に現れたtypedef名、構造体タグ名を使用することができる。

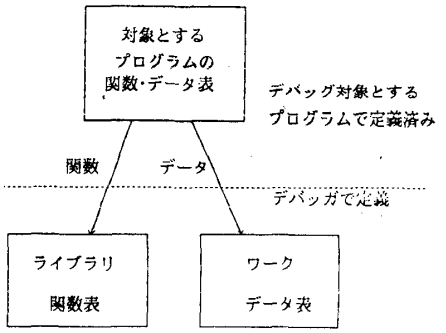

図3　デバッガにおける関数、データの関係

よって、本デバッガでの関数・データの関係は図3のようになる。

3．4 コマンド

本デバッガを立ちあげるとメッセージが出力された後プロンプト “$” が出力され、ここからコマンドを入力する。表1にコマンドを示す。

表1　コマンド表

cng	ファイル名	コマンドの対象とするファイルの設定
exp	式	データの参照、操作、及び関数コール
run	コマンド引数	関数mainより実行
bpp	オプション　関数名	関数に対するブレークの設定
fbpp	数	関数に対するブレークの解除
prbpp		関数に対するブレークの表示
bps	オプション　関数名　文種類	文に対するブレークの設定
fbps	数	文に対するブレークの解除
prbps		文に対するブレークの表示
bpd	オプション　式	データに対するブレークの設定
fbpd	数	データに対するブレークの解除
prbpd		データに対するブレークの表示
stat	オプション	実行状態の表示
strun	オプション	インタプリタ実行中の状態表示を行う
fstrun		インタプリタ実行中の状態表示を行うことの禁止
stfl	オプション　ファイル名	実行状態の履歴をとる
fstfl		実行状態の履歴をとることの禁止
ccl	宣言	データを定義
ret	式	リターン値の設定
exit		デバッガ部から抜ける
stop		デバッガを止める

4．おわりに

本デバッガは、CP/M-68Kの言語Cコンパイラを用いて開発された。総ステップ数は、言語Cで約10000行である。　将来的には、cat のオブジェクトに移行を行なう予定である。

最後に、 catの基礎を築いてくれた藤森英明(現:日本電気)、 篠田佳博(現:日立製作所)、並木美太郎(現:日立基礎研究所)の各位に深甚なる感謝の意を表する。

5．参考文献

[1] 並木美太郎、他：“ＯＳ／ｏのツールセット（５）－デバッガＤＥＢＵ－”、情報処理学会第30回全国大会

[2] 屋代寛、他：“ＯＳ／omicron用言語Cコンパイラｃａｔの開発”、情報処理学会ソフトウェア工学研究会48-1(1986)

[3] 田中泰夫、他：“言語Ｃインタプリティブデバッガ”、情報処理学会ソフトウェア工学研究会47-2(1986)