portable dumper: アーキテクチャに依存しない Emacsの起動時間短縮手法

永野 圭一郎, 林 芳樹

Linux Conference 2002, 20 Sep 2002

portable dumper とは何か

可搬性の高い Emacs 起動高速化手法

- Emacs 起動高速化: 必須 Lisp プログラムを解釈済みの状態で保存し、次回起動時からは Lisp プログラムを高速に復元する
- 可搬性の高い: プラットホーム依存の知識を用いない

背景 (1/2)

Emacs = Lisp インタプリタ + Lisp プログラム

- 起動までに大量の必須 Lisp プログラムを解釈する必要がある (loadup)
  - 時間的に高価

背景 (2/2)

高速化手法 (dump)

- Lisp プログラム解釈後の状態を保存
- 次回以降の起動時には、解釈済みの Lisp プログラムを用いて高速に起動する
- 手法: unexec (既存手法), portable dumper (我々の提案)

dump 既存手法: unexec

プロセスのメモリ内容から直接実行ファイルを作成する

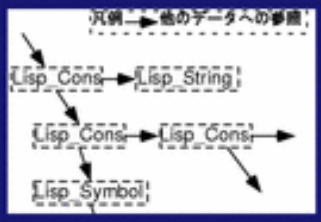
- Lisp インタプリタ実行ファイルにプロセスのメモリ内容を追加した新たな実行ファイルを作成する

portable dumper: 概要

- Lisp プログラムおよび Lisp インタプリタの状態をファイルに保存し、インタプリタがそこから loadup 直後の状態を復元できるようにする
- 保存・復元の際にプラットホーム依存の知識を用いない

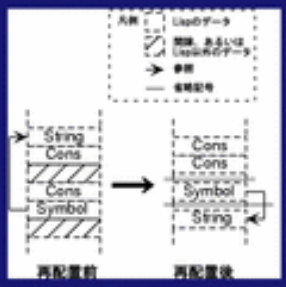
portable dumper: 実装(1/4)
Lisp プログラム: 保存対象の同定

- ガベージコレクションのマーク段階と同様に、Lisp のデータから別のデータへの参照を次々とたどることによって、十分なデータに到達することができる



portable dumper: 実装(2/4)
Lisp プログラム: 保存

保存サイズを最小とし、復元時の高速化をはかるため、再配置を行う



portable dumper: 実装(3/4)
Lisp プログラム: 復元

- 保存された Lisp プログラムを読み込む
- 期待した読み込みメモリアドレスを外れた場合、データ中の参照を更新する

portable dumper: 実装(4/4)
Lisp インタプリタの状態

- Lisp インタプリタの状態は C のグローバル変数に相当する
- 現在はひとつひとつ人手で同定し、保存・復元のコードを書き下している

実験(1/2): 起動時間の評価

高速化 (dump) 手法	起動に要した時間
高速化なし	1.43sec
unexec	0.0639sec
portable dumper (1)	0.0950sec
portable dumper (2)	0.113sec

- 高速化手法は依然、必要である
- unexec と portable dumper との速度差は無視できる程度である

実験(2/2): 可搬性の評価

以下のプラットホームで同一の portable dumper コードが動作することを確認した

- GNU/Linux (kernel 2.2/2.4)
- FreeBSD (RELEASE-4.5)
- Solaris (2.6/2.7/2.8 on SPARC, 2.8 on Intel)
- Microsoft Windows (Windows 98, Windows 2000)

まとめ

- Emacs 起動時の処理(loadup,dump)および dump 既存手法 unexec の紹介
- portable dumper の解説
  - Lisp プログラム部分の保存に関しては、GC のマーク段階と同様の手法で十分なデータに到達することができる

今後の課題

- Lisp インタプリタの状態の保存・復元に関して人手によらない方法の開発
- GNU Emacs への正式採用
- Meadow 2 のリリース

portable dumper: ユーザメリット

- 移植が容易になる → 移植が早くなる
- unexec に起因する不可解なバグが減少する
- (Meadow) Windows 9x 系列と Windows NT 系列が一本化される。バイナリをダウンロードしてきたら、loadup+dump なしでそのまま使えるようになる
- (Meadow) MSVC, Cygwin(MinGW) 両方でコンパイルできるようになる(なった)
