Asianux Road Show

Linux
トラブルシューティング
ソースコードの読み方

2007年10月

ミラクル・リナックス株式会社
吉岡弘隆

本日のアジェンダ

- ミラクル・リナックス (株)
- トラブルシューティングについて
- ソースコードの読み方

ミラクル・リナックス (株)

- 国産リナックス専業ベンダー
  - 国内に高度なlinux技術者が多数在籍
  - 日本人linux Kernel 投稿パッチ数 (4位)
    SuSE、東芝、富士通、ミラクル・リナックス
    http://wiki.livedoor.jp/linuxfs/d/Japanese%20Linux%20hacker
- Asianux開発元
- 本日はOSSにまつわるちょっとしたノウハウなどをお話する

ユメのチカラ (ブログ)

- http://blog.miraclelinux.com/yume/
- ブックマークで見た人気エントリー
- ソースコードの読み方(524個)
  - http://blog.miraclelinux.com/yume/2007/08/post_d6bd.html
- デバッグ方法論 (99個)
  - http://blog.miraclelinux.com/yume/2007/08/post_d3eb.html
- 多くの人に興味がある話題

トラブルシューティングとソースコードの読み方

- トラブルシューティングと問題の理解
- 実践的なコードの理解

トラブルシューティング

- 現代の魔法？
  - エキスパートのトラブルシューティングの技はまさに魔法使い

現代の魔法使い

- 例：SCTPのトラブル(パケットをダンプしたらパケットのチェックサムが不正でエラーになりコネクションが設立できなかった)
  - http://git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git;a=commit;h=28cd7752734563d5b0967b96a6bade7a1dc89c7f

```
diff -puN net/sctp/input.c~linux-2.6.9-sctp-use-linearize net/sctp/input.c
--- linux-2.6.9-42.10AX/net/sctp/input.c~linux-2.6.9-sctp-use-linearize  2007-05-14 20:36:43.000000000 +0900
+++ linux-2.6.9-42.10AX-hirofumi/net/sctp/input.c               2007-05-14 20:37:32.000000000 +0900
@@ -119,6 +119,9 @@ int sctp_rcv(struct sk_buff *skb)

                SCTP_INC_STATS_BH(SCTP_MIB_INSCTPPACKS);

+               if (skb_linearize(skb, GFP_ATOMIC))
+                       goto discard_it;
+
                sh = (struct sctphdr *) skb->h.raw;
```


トラブルシューティング

- ベンダーにとって、トラブルシューティング力は付加価値の源泉
  - 単価が高い仕事、ソースコードレベルの解析、ログ、ダンプ解析
- OSSは、情報が全て公開されているので、深堀できる
  - 商用製品はブラックボックス。差別化が難しい
  - OSSはトラブルシューティング力が付加価値

トラブルシューティング

- 技術者としての専門性
  - 技術に対する深い理解と応用力が必要
  - 付加価値になる＞簡単にできない
- 技術者としての喜び
  - 難しい問題を解決したときの達成感
  - パズルを解くような感覚

ソースコードを読むチカラ

- プログラマの基礎体力
  - ソフトウェア開発コストの大部分は保守
  - 不具合修正、改良、機能追加にはコードの理解が必須
- 技術者の付加価値
  - OSSは深追いできる
  - 陳腐化しにくい
- プロフェッショナルとしての研鑽
  - すぐれた技術者はソースコードを上手に読む

トラブルシューティングの基礎

- 問題の再現
  - 情報収集
  - 環境構築
  - 再現 (最小化、局所化に留意)
- 問題の理解 (本日のメイントピック)
- 問題の解決

問題の再現

- 情報収集
  - HW情報、SW情報、等々
  - 例：# mcinfo
- 環境構築
  - 問題が発生する環境になるべくあわせる
- 現象の局所化、最小化
  - 2分検索を試みる
    - ある環境だと問題が発生し、ある環境だと問題が発生しない。その差分を追跡する手法。

問題の局所化、最小化

- 2分検索
- 例：kernel 2.6.17 OK, 2.6.18 NG
  # git bisect start
  # git bisect good v2.6.17
  # git bisect bad v2.6.18
  ... テスト結果により
  # git bisect good/bad
  上記を繰り返して、どの変更が問題を引き起こしたかを局所化する

kernel dump (AXS3)

- kernel dumpがあれば障害時の詳細情報が入手できる
- kernel dumpの設定
  # system-config-kdump
- 設定の確認
  # echo c > /proc/sysreq-trigger
- kernel-debuginfoのインストール
- crashでkernel-dumpを解析

問題の理解

- 再現ケース
  - 問題を再現できれば、八合目に登ったも同然
  - 最小の手数で再現ケースを発見すること
- ソフトウェアの場合、ソースコードが最終的なよりどころ
- ソースコードは完璧なホワイトボックス
  - 厳密な理解が可能

コードの読み方

- なぜ、コードを読むのか
- どのように、読むのか

なぜコードを読むのか？

- 仕事だから
  - トラブルシューティング (不具合修正)
  - 機能修正、機能開発
  - 自己研鑽、勉強
- 趣味だから
  - 楽しいから
  - 自己啓発 (知的好奇心)
- 不純な動機
  - 形から入る

なぜコードを読むのか

- 人それぞれ、人生いろいろ
- 無目的でもいいじゃないか
- 熟読、濫読、積読、黙読、音読、再読、誤読、精読、速読、耽読、通読、復読、輪読、朗読、輪読、…

コードの理解について

- モットー：
  コードは読むな、
  理解しろ～

どのようにコードを理解するのか

- 個人的な方法を紹介する
  - 唯一あるいはベストな方法というわけでもない
  - 適材適所、もっと良い方法があると思う
  - 公開することによって進化したい（もっと良い方法への模索）

ソースコードを読む視点

	微視的理解	巨視的理解
静的理解		
動的理解		

理解の仕方、読み方

- 静的、動的理解
- 微視的、巨視的
- 規模の把握
- ツールの利用
- 事例

静的理解、動的理解

- 静的理解
  - 字面での理解
- 動的理解
  - 動作による理解

静的、動的構造

- 静的構造
  - 規模
  - ディレクトリ構造
  - 名前つけ規約
- 動的構造
  - 呼び出し経路
  - プロファイリング
  - 実行結果

微視的、巨視的

- 微視的：細部からの理解
  - 最終的にはコードの一行
- 巨視的：全体からの理解
  - 規模、構造、機能など。実行結果（性能？）
  - 俯瞰図、鳥瞰図。

規模の把握（巨視的理解）

- 規模重要
  - 規模（相手）を知らずして作戦を立てられない
  - 大局的な地図、俯瞰図、鳥瞰図
  - 大規模になればなるほど、システマティックな方法論が必要になってくる
  - 巨視的な理解

規模の把握

- 小規模：100K行未満程度、
  ファイル数100未満
  10人未満
- 中規模：100K行～1M行程度
  ファイル数100～1000未満
  100人未満
- 大規模：1M行以上
  ファイル数1000以上
  100人以上
  ざっくりの規模感

規模の把握 (例)

- find -type f -name "*.[ch]"¦wc
- find -type f -name "*.[ch]"¦xargs wc¦grep total

名前	ファイル数	行数
Ruby 1.8.5	257	197767
Linux 2.6.18	16522	680万
MySQL 5.0.24a	1795	988463

ディレクトリ構造

- トップディレクトリは、ソフトウェアの論理的構造を表している
  - doc ドキュメント
  - lib ライブラリ関係
  - test テスト
- ソースツリーの把握

ドキュメント

- README, INSTALL, COPYING, …
- 内部ドキュメント(Docs)
- リリースノート
- ChangeLog

変更の履歴

- ChangeLog/Release Notes
- コード管理システム
  - 例：Linux git/Subversion/CVS
- Mailing List
- Wiki
- blog
- 変更（時間軸）の微視的理解

ドキュメント

- Mailing List
  - 例：http://lists.mysql.com/
- Bug database
  - 例：http://bugs.mysql.com/
- 開発者との会話
- Googleに聞く

ソースコードを読む視点

	微視的理解	巨視的理解
静的理解	ソースコード、ChangeLog、リリースノート	ディレクトリ構造、名前付け規約、規模の把握（行数、ファイル数など）
動的理解	デバッガによる実行	実行性能、リグレッションテスト

いよいよコードを読む？

- ツール
  - エディタ：(x)emacs
  - デバッガ：gdb
  - クロスレファレンス：cscope
  - カーネルの場合、クラッシュダンプ(crash)

動的理解

- ともかく動かす
  - make; make install
  - strace
  - ltrace
  - gdb
  - oprofile
  - リグレッションテスト
  - ベンチマークテスト

make

- とりあえず、make; make install
- 実行環境の構築
  - gcc -g (デバッグシンボルを付加する)
  - cscopeのインデックスを作成
  - ビルドは(x)emacsのshellなどから行い、ビルドのログを取得しておく

strace

- システムコールのトレース

```
$ strace ruby -v
execve("/usr/local/bin/ruby", ["ruby", "-v"], [/* 39 vars */]) = 0
uname({sys="Linux", node="asianux2.miraclelinux.com", ...}) = 0
brk(0)                                  = 0x976a000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or di
open("/etc/ld.so.cache", O_RDONLY)      = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=89946, ...}) = 0
old_mmap(NULL, 89946, PROT_READ, MAP_PRIVATE, 3, 0) = 0xb7
close(3)                                = 0
open("/lib/libdl.so.2", O_RDONLY)       = 3
```


gdb

- デバッガはコードを理解するためにある
  ∴コードを読むために使う

gdbで読むための準備

- gcc -g でコンパイル
- コンパイルしたコードサイズが大きくなる以外、特に副作用はない
- printfデバッグは有害無益

gdb

- ブレークポイントを設定
- ウオッチポイント(変数の変更)
- run
- 止まった時点で
  - bt スタックフレームの表示
  - p 変数の表示
  - c 実行再開、s ステップ実行(関数に潜る)、n ステップ実行(関数に潜らない)
- 繰り返す

oprofile

- プロファイラー
  - 実行時のボトルネックを発見
  - # opcontrol --start
  - テストの実行
  - # opcontrol --stop
  - # opreport -l

oprofile

```
CPU: Core Solo / Duo, speed 2666.77 MHz (estimated)
Counted DCACHE_PEND_MISS events (Weighted cycles of L1 miss outstanding) with
a unit mask of 0x00 (Weighted cycles) count 100000
        vma       samples  %        linenr info                 app name symbol name
000000000042be50 244787   33.2383  gc.c:1661                   ruby        os_each_obj
  000000000042bfac 1        4.1e-04  gc.c:1599
  000000000042bfb9 5        0.0020   gc.c:1599
  000000000042bfbe 6        0.0025   gc.c:1599
  000000000042bfd0 4862     1.9862   gc.c:1601
  000000000042bfd3 228573   93.3763  gc.c:1601
  000000000042bfd6 2698     1.1022   gc.c:1601
  000000000042bfd8 250      0.1021   ruby.h:672
```

oprofile

- 高速道路で、ズバリ最もコストのかかっているところに連れて行ってくれる
- コードを読まないで、理解する極意

微視的理解

- ひたすらコードを読む
- 王道はない
- ↑身もふたもない
- データ構造、変数などに注目し、どこで定義され、参照され、代入(変更)されているかという観点でみる
- デバッガとエディタ、クロスリファレンスツールを駆使

微視的理解

- $ time find -type f ¦egrep \
  '\.([chp](xx¦pp)*¦cc¦hh)$'\ ¦xargs
  egrep -l hogehoge
- hogehogeを含むファイルを検索

微視的理解

- クロスリファレンスツール
  - 変数の定義(型情報)、変更(代入)、参照
- 変数(関数)が、どのように定義されていて、どのように参照、変更されているかを追う
- cscope
- lxr

ソースコードを読む視点

	微視的理解	巨視的理解
静的理解	ソースコード、ChangeLog、リリースノート、cscope	ディレクトリ構造、名前付け規約、規模の把握（行数、ファイル数など）
動的理解	デバッガ(gdb)による実行、strace	oprofile、strace、実行性能、リグレッションテスト

参考書

- Linux
  - 詳解LINUXカーネル第三版 ISBN:487311313X
  - Linuxカーネル2.6解読室 ISBN:4797338261
- コードリーディング
  - ISBN:4839912653
- Rubyソースコード完全解説
  - ISBN:4844317210(品切れ中)

- ブログ：ユメのチカラ
- http://blog.miraclelinux.com/yume/
- 未来のいつか/hyoshiokの日記
- http://d.hatena.ne.jp/hyoshiok/

人材募集中

- ソフトウェア開発マネージャ
- カーネルエンジニア
- Asianuxパッケージ開発エンジニア
- http://www.miraclelinux.com/corp/recruit/


Do the Next, Open your Window
MIRACLE