CGL3.0の実現に向けた Live Patch機能の実装

NTTコムウェア
オープンソースソフトウェア推進部
チーフエンジニア
中野　宏朗

2004.10.15

概要

現在、OSDLにおいて、CGL仕様3.0が検討されている。その仕様に書かれている要求の1つに、Live-Patch機能（プロセスを再起動することなく修正や機能追加を行う機能）がある。この機能は、通信業界では一般的に必要とされるものである。

今回のプレゼンテーションでは、Live-Patch機能の概要、動作原理について説明する。

アジェンダ

- CGL仕様3.0版と Live-Patch機能
  - OSDLとCGL
  - CGLの歴史
  - CGL仕様3.0
  - CGL仕様3.0の策定過程
  - CGL仕様3.0とLive-Patch機能
  - OSSプロジェクト「Pannus」
- Pannusプロジェクトによる Live-Patch機能の実装
  - Pannus動作原理
  - ロード機能
  - リゾルバ機能
  - パッチャ機能
  - コントローラ機能
  - Pannusシステムコール一覧
  - 機能構成
  - Pannus動作環境
  - インストール
  - サンプルの実行
  - サンプルコード例
  - サンプルの実行比較
  - まとめ

CGL仕様3.0版とLive Patch機能

OSDL とCGL

- OSDL（Open Source Developers Labs）
  - 2000年に設立されたNPO（非営利団体）
  - Linuxの基幹系システムへの適用推進を目的(Linus Torvalds も参加)
  - CGL（Carrier Grade Linux）、DCL（Data Center Linux）、DTL（Desk Top Linux）ワークグループが活動
  - ボード会議への参加、CGLジャパンのチェアマン等の貢献

- CGL（Carrier Grade Linux）
  - 高可用性で堅牢な、通信事業者の基幹系システムの基盤となりうるLinux
  - OSDLのCGLワークグループ（2002年設立）により仕様策定

CGL の歴史

- CGL 仕様 1.0 / 1.1　　<2002.08 / 2002.10>
  - ディストリビューター各社より製品版としてリリース済
  - Registration（*1）について、特にルールはなかった

- CGL 仕様 2.0　　<2003.9>
  - Registration（*1）について明確化
    - 仕様書中の、「優先度 1の項目」を盛り込む事が「準拠」の必須条件
    - 「部分準拠」も認める（盛り込んでいない項目を開示する必要あり）
  - 現時点で、CGL仕様2.0に「準拠」している製品はない
    （「準拠」と称するための条件を満たし、その手続きを行った製品はない）

- CGL 仕様 3.0 / 3.1　　<2005.2 / 2005.6 公開予定>
  - 現在、OSDL CGLワークグループにおいて検討・編集作業を実施中

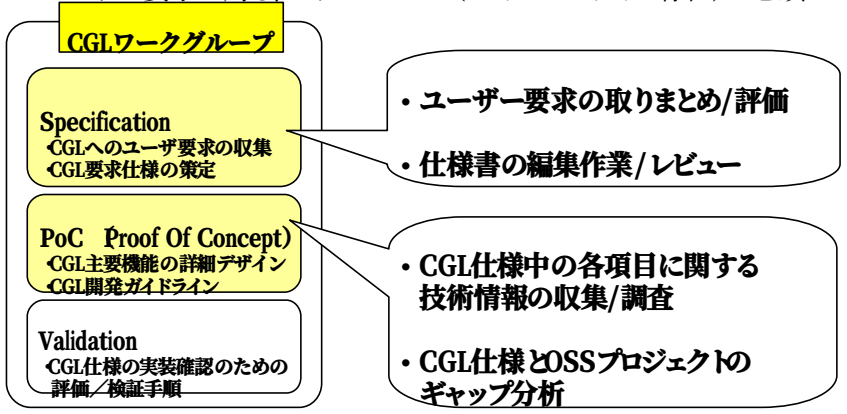
（*1）Registration：
仕様書に記述されている内容とディストリビューションの差を無くすため、仕様書中の優先度 1の項目の実装を行うことが「CGLx.x準拠」と称するための条件とルール付けした

CGL仕様3.0

- CGL仕様3.0は、7つのカテゴリにより構成される
  - Standard / API
  - Cluster
  - Availability
  - Serviceability
  - Performance
  - Security
  - Hardware
- 通信事業者の基幹系システムに必要な機能・性能要求を提出
  ⇒仕様に盛り込まれる見込み
- 履歴/スケジュール
  - 2003年10月　仕様検討開始（ユーザー要求の収集開始）
  - 2004年9月　ドラフト版公開済
  - 2005年2月　CGL仕様3.0版公開予定
  - 2005年6月　CGL仕様3.1版公開予定

CGL仕様3.0の策定過程

- CGL仕様策定：
  - ユーザー要求を取りまとめ
  - 技術的観点から評価
  - ユーザー要求に対し、ソリューション（OSSプロジェクトの存在）が必須



CGL仕様3.0 とLive-Patch機能

■ Live-Patch機能の概要

□ CGL仕様3.0検討に、通信事業者から提出された要求項目の1つ

- 通信基盤システム（交換機等）では、99.999%級の信頼性が求められ、サービスを停止できない
- 頻繁にサービス追加・変更等のシステム変更を行う必要がある

□ プロセス停止することなく、アプリケーションにパッチを適用する機能
⇒サービスプロセスの再起動をする必要がない

通常のパッチのやり方
・ソースを修正・再コンパイル
・バイナリを直接修正
⇒ 再読み込み（プロセス再起動）

⇒ プロセスの再起動によるサービス停止時間が発生

CGL仕様3.0 とLive-Patch機能

■ CGL仕様3.0 とLive-Patch機能

□ 現在検討中のCGL3.0仕様書中では、Availability章に掲載されている

□ Live-Patch機能に対するOSSプロジェクト情報として、「Pannus」プロジェクトが紹介されている

CGL仕様3.0 ドラフト版（6月版）より

ID	Name
AVL14.0 V2.0 --	Live Patching
Description：	OSDL CGL specifies that carrier grade linux shall provide the mechanism for a running application's functions to be replaced dynamically (without restarting).

キャリアグレードリナックスは、（プロセス）を停止・再起動することなく、アプリケーションの機能を置き換える機構を提供しなければならない

OSSプロジェクト「Pannus」

- 2004年5月に、NTTグループにより設立されたOSSプロジェクト
- Live-Patch機能を実装し、GPLの下ソース公開中
- NTTコムウェア は、Pannusプロジェクト設立当初より参加
  - 仕様検討・ソース作成等の活動を実施・継続中
- URL：
  http://developer.osdl.jp/projects/pannus/
- ML：
  pannus@developer.osdl.jp

Pannusプロジェクトによる Live-Patch機能の実装

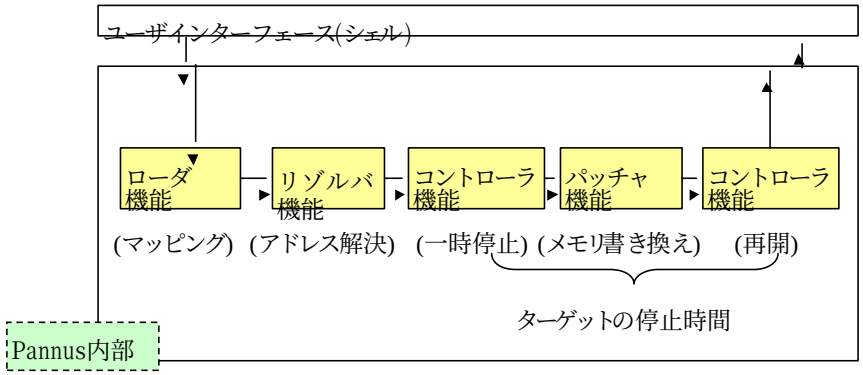
Pannusの動作原理

- Pannusはpannusコマンドを実行するだけでターゲットを終了させることなく関数単位で処理を入れ替えることができる。
- Pannus内部では以下の機能を実装している。

1. ロード機能 :パッチファイルをディスクから読み込み、ターゲットのアドレス空間に貼り付ける。
2. リゾルバ機能 :パッチ内のアドレスが解決していない関数や変数について、アドレスを計算して解決する。
3. パッチャ機能 :ターゲットのメモリを読み書きする。
4. コントローラ機能 :ターゲットを停止させたり動かしたりする。

Pannusの動作原理

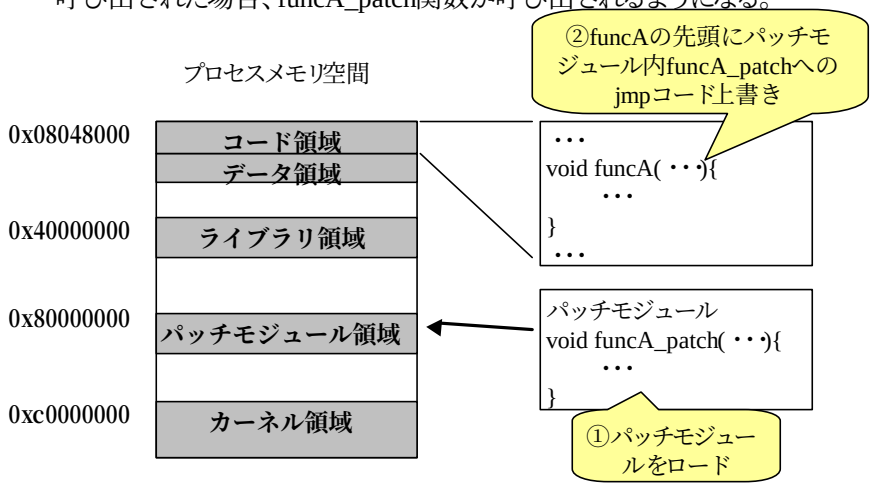
- Pannus内部での処理の流れは下図となる。



Pannusの動作原理

■ 動作概念図

□ パッチモジュールのロード、jmpコードの上書きにより、次回funcA関数が呼び出された場合、funcA_patch関数が呼び出されるようになる。



ロード機能

■ パッチファイルをディスクから読み込み、ターゲットのアドレス空間に貼り付ける機能。

■ 既存の似たような機能として、共有ライブラリのロードがある。

□ 共有ライブラリのロードはglibcのdlopen()関数などで行われる。この場合、共有ライブラリがロードされるのはdlopen関数を実行したプロセス自身のアドレス空間である。

□ pannusコマンドでdlopenを用いると**pannusコマンド自身のアドレスにロードされてしまう**。

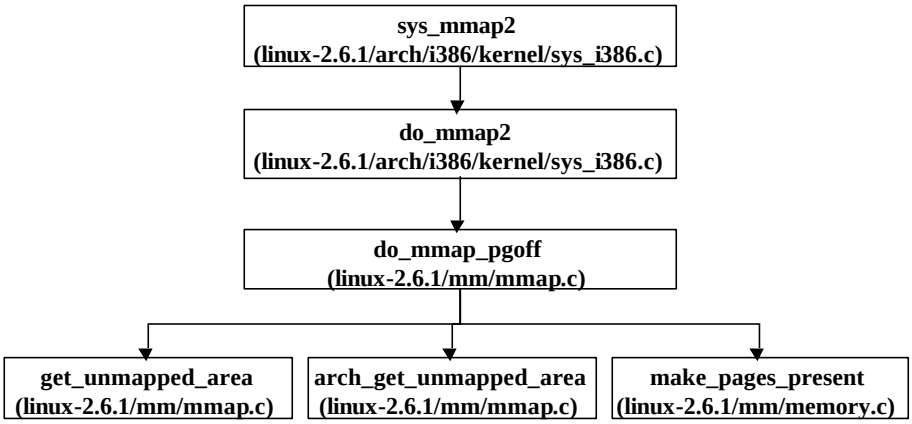
□ **ターゲットプロセス、つまりpannusコマンド以外のプロセスのアドレスにファイルをロードする機能が必要**→カーネルにシステムコールとしてその機能を作り、pannusコマンドから呼び出す。

■ dlopenでは自身のアドレス空間へのロードのためにmmap2という既存のシステムコールを用いている。

■ mmap2を改造し、mmap3という任意のプロセスのアドレス空間にロードが出来るシステムコールを追加。

ロード機能 – mmap2の構造 -

- 既存のmmap2システムコールはどうなっているか？
  - currentタスクのmm_struct構造体から空き領域を検索し、空いていればアドレスを確保してユーザ空間にリターンする。

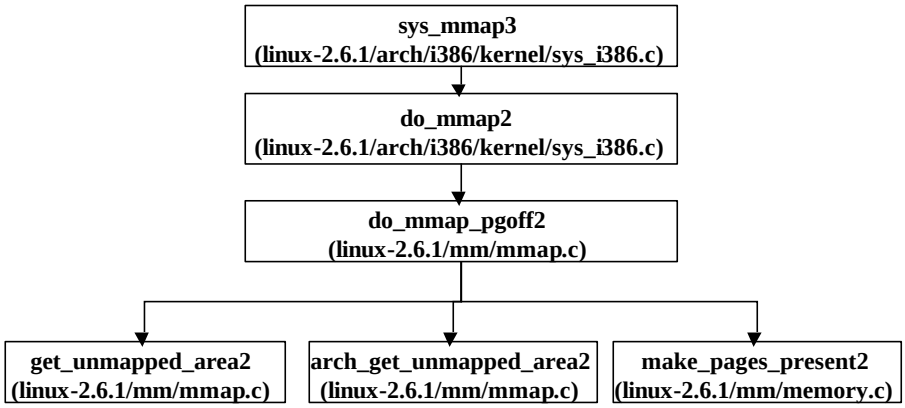


ロード機能 – mmap2の構造 -

- mmap2では扱うプロセスをcurrentに指定しているために、自プロセスのアドレス空間にしかロードできない。
- ユーザ空間でpidを指定することで、current以外にも対応させれば・・・

ロード機能 – mmap3の構造 -

- mmap3システムコールの構造
  - currentタスクではなく、指定されたプロセスのmm_struct構造体から空き領域を検索し、空いていればアドレスを確保してユーザ空間にリターンする。



ロード機能 – mmap3の構造 -

- mmap3システムコールの引数はmmap2システムコールの引数にpidを追加したものとなる。

システムコール

```
mmap3(unsigned long start, size_t length,
      int prot, int flags, int fd, int pid)
```

リゾルバ機能

- 読み込んだパッチでアドレスが解決していない関数や変数について、ロードしたアドレスを元にしてアドレスを計算して解決する機能。
- 現在のLinuxではELFバイナリ形式が一般的に用いられている。
- pannusでのアドレス解決
  - パッチで置き換える関数や変数→ターゲットのシンボル情報から取得
  - パッチ内で使用している共有ライブラリ内の関数→共有ライブラリのシンボル情報から取得
  - パッチ内の未解決アドレス→パッチファイルのシンボル情報から取得

リゾルバ機能

- ターゲットのシンボル情報はターゲットのコンパイル時にmapfileを出力させ、パッチ実行時にmapfileを読み込んで検索する。
- パッチ内や共有ライブラリ内のシンボル情報は.dynamicセクションの.dynsymと.dynstrセクションを読み込んで検索する。
  - .dynsymセクションにはシンボル名、種類、値などが、.dynstrセクションには文字列が格納されている。

リゾルバ機能 –パッチ内のアドレス解決-

- パッチはELFの位置独立な共有ライブラリ形式でバイナリを構築する。
  - gcc で作成するには –shared　オプション付きで構築。
- この形式での.dynamicセクションは以下のような構造。

リゾルバ機能 –パッチ内のアドレス解決-

- 呼び出し関数のアドレスは.gotに格納される。
- 位置独立コードの場合、メモリにロードされてはじめてアドレス解決される関数もある。それらのアドレスも.gotに格納される。
- したがって、パッチ内のアドレス解決ではシンボルの検索で見つけたアドレスをパッチの.gotに書き込んでやることでアドレス解決がされる。.gotのどのアドレスに書き込むかは.rel.dynや.rel.pltセクションに値が格納されている。
- PannusではR_386_RELATIVEとR_386_GLOB_DAT/R_386_JMP_SLOTに対応

リゾルバ機能 –コマンドファイル-

- パッチ適用可能なものは以下。
  - 関数・・・パッチを適用される関数の先頭にjmpコード(E9)を埋め込む。
  - 変数・・・グローバルな変数に対し、指定したデータを書き込む。
  - ポインタ・・・グローバルなポインタ変数に対し、パッチ内の指定したシンボルを指すようにする。

- 関数へのパッチをType J、変数へのパッチをType V、ポインタへのパッチをType Pとよび、それぞれ必要な情報をテキストファイルとして準備する。このファイルをコマンドファイルと呼ぶ。

リゾルバ機能 –コマンドファイル-

- コマンドファイルの書式は以下。

```
J     target_symbol     patch_symbol
V     target_symbol     data
P     target_symbol     patch_symbol
```

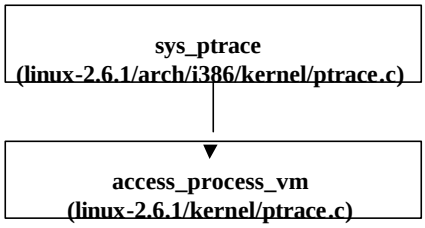
  - target_symbolとpatch_symbolはそれぞれターゲットとパッチのシンボル
  - dataは格納するデータを16進数表現したもの

パッチャ機能

- ターゲットプロセスのメモリ内容を書き換える機能。この機能を利用してパッチ対象の関数の先頭にマシン語でジャンプコードを上書きする。
- 既存の似たような機能として、ptraceというシステムコールがある。
  - ptraceシステムコールはpidを指定して他のプロセスのメモリ内容を書き換えることが可能。
  - ただし、ptraceシステムコールは書き換え対象のプロセスをptraceを実行したプロセスの子プロセスにしてしまう。それはデバッガがメモリ書き換え以外にstep実行やシグナルの配信などの機能を持つため(メモリ書き換えのみだとターゲットを子プロセスにする必要はない)。
- pannusではptraceのメモリ書き換え機能のみを切り出し、accesspvmという新たなシステムコールとして実装した。このシステムコールがパッチャ機能の実体である。

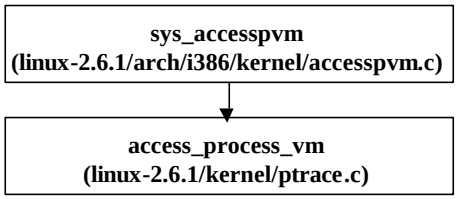
パッチャ機能 - ptraceでのメモリアクセス部の構造 -

- ptraceシステムコールでメモリ読み書きを行う場合、システムコール関数であるsys_ptraceからaccess_process_vmというメモリアクセス関数を呼び出している。
- 渡している情報はフラグ、読み書きされるメモリのアドレス、読み込んだメモリのコピー先又は書き替え内容のコピー元アドレス(ワード単位)、長さ、pid



パッチャ機能 - accesspvmの構造 -

- access_process_vmを直接呼び出すようなシステムコール関数を実装。
  - ptraceでのアタッチ処理を省略
  - ptraceではメモリの読み書きがワード単位だったが、任意長のメモリを指定できるように実装。



システムコール

```
readpvm(pid_t pid, void * address, void * data, int len)

writepvm(pid_t pid, void * address, void * data, int len)
```


コントローラ機能

- パッチャ機能でターゲットのメモリを書き換える際、安全のためにプロセスを一時停止したり再開したりする機能。
- ランキュー操作を行うことでプロセスの停止/再開を行う。
  - カーネル内ではLinux上のプロセス情報はtask_struct構造体に格納されている。
  - コンテキストスイッチでは、スケジューラのランキューに登録されているtask_struct構造体に対応するプロセスを一定時間単位で切り替えながら実行する。ランキューにtask_struct構造体が登録されていないプロセスは実行されない。
- コントローラとして、ターゲットのtask_struct構造体をランキューに登録したり登録を削除したりする機能を持つpcntlというシステムコールを実装。

コントローラ機能

- パッチャでメモリを書き換える直前にpcntlでターゲットプロセスをランキューから外し、プロセスが停止した位置が書き換え位置と重なっていないかチェックする。プロセスがランキューになかったり、停止位置が書き換え位置と重なっていればpcntlでプロセスを再開してリトライする。

- 重なっていなければ、メモリを書き換えた後にpcntlでプロセスをランキューに登録することで再開する。

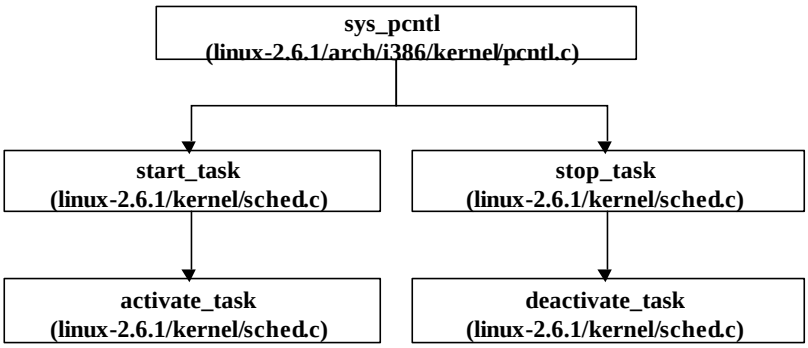
コントローラ機能 - pcntlの構造 -

- 2.6.1カーネルでタスクをランキューへ登録しているのはkernel/sched.cのactivate_taskというインライン関数。ランキューから削除しているのはdeactivate_taskというインライン関数。

- それぞれをsched.c外の関数から操作できるようにするため、start_task、stop_taskというインターフェース用関数を実装。

- システムコール関数sys_pcntlからstart_task、stop_taskを呼び出す。

コントローラ機能 - pcntlの構造 -



システムコール

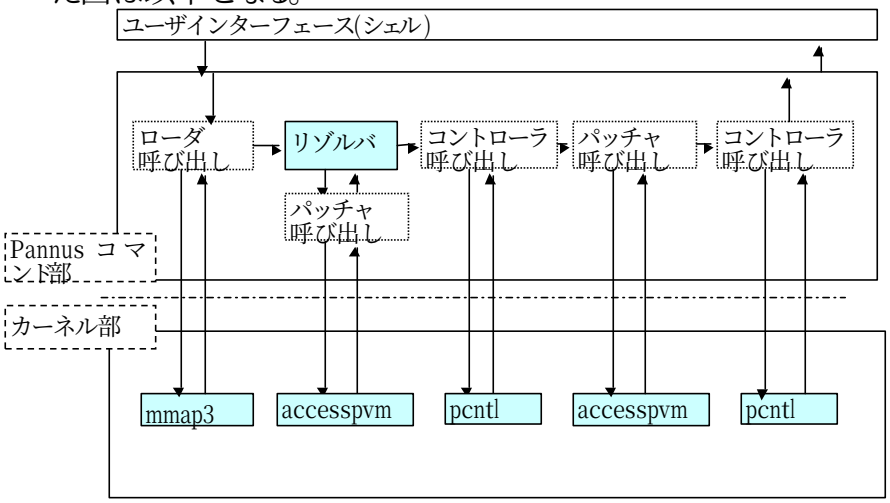
```
pcntl(long pid, int flag)
```


コントローラ機能 - pcntlの構造 -

- ランキュー操作でターゲットを停止させるためにはタスクがRUNNING状態にないといけない。→RUNNING状態にないとき、pcntlシステムコールはerrnoとしてEAGAINを返す。
- pannusコマンド部でEAGAINを検出した場合は、数ミリ秒待った後に再度ターゲットの停止を試みる。
- ターゲットが停止した場合は停止アドレスとパッチャによる書き換え位置を比較し、重なっていなければパッチャを実行する。重なっていればターゲットを再開させて再び停止を試みる。

機能構成

■ ローダ、リゾルバ、パッチャ、コントローラの各機能の関係を表した図は以下となる。



Pannusシステムコール一覧

ロード機能

```
mmap3(unsigned long start, size_t length,
      int prot, int flags, int fd, int pid)
```

パッチャ機能

```
readpvm(pid_t pid, void * address, void * data, int
len)
writepvm(pid_t pid, void * address, void * data, int
len)
```

コントローラ

```
pcntl(long pid, int flag)
```


Pannusの動作環境

- Pannusの動作環境は以下となる。
  - IA-32アーキテクチャのみ対応
  - ベースとなるカーネルはkernel.orgで公開されているlinux-2.6.1
  - gcc3.2.2
  - glibc2.3.2

- 前提条件
  - パッチを適用される対象はC言語でコーディングされたアプリケーションであること。
  - パッチを適用される対象はELF実行形式プログラム、パッチはELF共有ライブラリ。
  - カーネルには未対応。

インストール

- pannusのインストール方法
  1. Linux-2.6.1のカーネルソースツリーにパッチをあてる。

```
> patch -p1 kernel.2.6.1.pannus.patch
```

  2. カーネルの再構築と再起動。
  3. カーネルソースツリーのunistd.hを/usr/include配下にコピーする。

```
> cp include/asm-i386/unistd.h ¥
/usr/include/asm/unistd.h
```

  4. ライブラリのコンパイルとインストール

```
> cd ~/pannus/library
> make
> make install
```

  5. コマンドのコンパイルとインストール

```
> cd ~/pannus/pannus
> make
> make install
```

サンプルの実行

■ pannus付属サンプルの実行方法

1. サンプルのコンパイル。

```
> cd ~/pannus/sample
> make
```

2. ターミナルを2つ用意して、片方でターゲットプログラムを実行。

```
> ./target
```

3. もう一方のターミナルでpannusコマンドの実行

```
> pannus `pidof target` patch mapfile pannus.cmd
(pannus ターゲットのPID パッチファイル マップファイル コマンドファイル)
```

4. ターゲットの標準出力が変化したことを確認。

サンプルコード例 –ターゲット–

■ pannus付属サンプルのコード

□ ターゲット

```
char* func_J()
{
    return "func_J of target.";
}

char str_V[] = "str_V of target.";

char *str_P = NULL;

void (*func_L)(void)=NULL;


int main( int argc, char **argv )
{
…省略)…
```

パッチで置き換える関数 (Type J)

パッチで置き換える変数 (Type V)

パッチで置き換えるポインタ (Type P)

パッチ内のアドレス解決確認用ポインタ

サンプルコード例 –パッチ–

- pannus付属サンプルのコード
  - パッチ

```
char str_P_plug[] = "str_P of plug.";
char dummy[10000];

char* func_J_plug()
{
    return "func_J of plug.";
}

char        *func_J();
extern char str_V[];

…(次頁)…
```

パッチで置き換えるポインタ (Type P)

パッチで置き換える関数 (Type J)

サンプルコード例 –パッチ–

- pannus付属サンプルのコード
  - パッチ

```
…(続き)…
void func_L_plug(void)
{

    printf( "LIB    = (printf)¥n" );
    printf( "LIB    = %d¥n", errno );


    printf( "Plug   = %s¥n", str_P_plug );
    printf( "Plug   = %s¥n", func_J_plug() );


    printf( "Target = %s¥n", str_V );
    printf( "Target = %s¥n", func_J() );

}
```

パッチ内から共有ライブラリの関数呼び出しを確認

パッチ内のアドレス解決を確認

パッチ内からターゲットの関数呼び出しを確認

サンプルコード例 –コマンドファイル–

- pannus付属サンプルのコードに対するコマンドファイル

```
J       func_J func_J_plug
V       str_V  7374725f56206f6620706c75672e00
P       str_P  str_P_plug
P       func_L func_L_plug
```

  - 1行目 :関数に対するパッチについてターゲットのシンボル名とパッチのシンボル名を記述
  - 2行目 :ターゲットの変数に格納する値を記述
  - 3, 4行目 :ポインタ変数に対するパッチについて、ターゲットのシンボル名とパッチのシンボル名を記述

サンプルの実行比較

- サンプルについて、つぎの2つを比較
  - パッチによる変更内容を反映させた新しい実行ファイルを起動した場合
    - 古い実行ファイルを停止してから新しい実行ファイルを起動するまでの時間
    - ・・・1s以上
  - pannusによりプログラム実行中にパッチを当てた場合
    - pannusのコントローラ機能でサービスが停止している時間
    - ・・・50μs
- システムがクラスタなどの冗長構成をとっている場合、マシン切り替え時間より短い時間でパッチを適用できなければ意味がない。
  - CGLではフェイルオーバによるサービス再開を1s以内と規定している。

まとめ

- 比較結果より
  - Pannusによるパッチ適用ならば、サービスの高可用性にたいする要求の厳しいシステムにおいても、許容範囲な停止時間でパッチの適用が可能。

ご静聴ありがとうございました。

NTTコムウェア
オープンソースソフトウェア推進部
チーフエンジニア
中野 宏朗
Mail To： nakano.hiroaki@nttcom.co.jp
URL： http://www.nttcom.co.jp/