Vol.2010-SLDM-144 No.39
Vol.2010-EMB-16 No.39
Vol.2010-MBL-53 No.39
Vol.2010-UBI-25 No.39
2010/3/27

組込みシステム学習支援環境「港」における Linux プロセス可視化環境の開発

本橋大樹[†]　西野洋介[††]　早川栄一[†††]

本研究は，組込みシステム学習支援環境「港」の可視化環境を用いて，Linux のプロセス管理を可視化するものである．組込みシステムにおける Linux 利用の増加よって，Linux のプロセス管理機構の理解は重要になってきている．本研究では，学習支援環境「港」上に OS の概念レベルから動作レベルまでの連続的な学習を可能にする可視化環境を開発した．特徴としては，ftrace を用いてオーバヘッドの少ないプロセス情報取得機構を開発し，そこで得られたログを元に「港」上の可視化環境を動作させるようにした．また，可視化ツールのユーザインタフェースを改良し，学習者が容易に操作可能な環境を提供した．

Development of Visualization Environment for Linux Process on Embedded System Learning Environment Minato

Daiki Motohashi[†], Yosuke Nishino[††] and Eiichi Hayakawa[††]

Abstruct : This paper describes the development of visualization environment about Linux process management using system learning environment Minato. Increasing the use of Linux in embedded systems, it is important to understand the behavior of process management. We developed the visualization environment to learn from conceptual level to behavior level continuously. System log level visualization is available to present the ftrace based low overhead process information acquiring mechanism. The user interface of the visualization system is improved from the previous system.

† 拓殖大学　大学院　工学研究科
†† 東京都立八王子桑志高等学校
††† 拓殖大学　工学部　情報工学科

1. 背景と目的

近年，サーバや組込みシステムにおいて Linux を用いる機会が増えてきている[1][2]．実際のシステム開発において，スレッドプログラミングを用いる場面は多いことから，Linux のプロセス管理などの動作の仕組みを知ることは重要である．このような要求に対して，Linux などの OS のプロセス管理の可視化を行っている研究[3][4]も存在する．しかし，表示される情報が多く，プロセスの状態遷移などを簡単に把握することができないことや，教材環境とし統合化されていないことから，学習環境としての利用は難しい．

これに対して，現在，拓殖大学工学部情報工学科早川研究室ではオペレーティングシステム（以下 OS）の学習支援環境として「港」システム[5][6]がある．これは，OS の動作を可視化することで動作のしくみを学習者が容易に理解できるようにするためのものである．

本研究の目的は「港」システムを Linux に対応させて，プロセス管理機構の可視化を行うものである．さらに，可視化環境を整備し操作や表示を統一し，概念レベルの学習から実際の Linux の動作の学習までを連続的に行えるシステムを提供する．

2. 「港」システム

「港」システムには，ロボットを用いた組込みシステムの統合型の学習支援環境である．次の学習環境を提供している．

- 教材ロボットおよびロボット上での OS ログ取得機構
- ロボット動作と OS ログとを合わせてモニタリングするログトレーサ
- ロボット上でのプログラムをシミュレートする CPU シミュレータ
- 仮想空間内でロボットの動作をシミュレートする 3D シミュレータ
- ユーザ操作および OS ログを元に OS の動作を可視化する OS 可視化ツール

OS 可視化ツールには図 1 のようにユーザ操作によりタスクの生成やスケジューラの選択を行い，それをもとに可視化を行う概念レベルのツール[4]と，図 2 のように実際の OS から取得した動作ログを利用して実際の OS 上にあるタスクの状態遷移の様子を可視化している動作レベルのツール[3]の二つのツールが存在する．

Vol.2010-SLDM-144 No.39
Vol.2010-EMB-16 No.39
Vol.2010-MBL-53 No.39
Vol.2010-UBI-25 No.39
2010/3/27

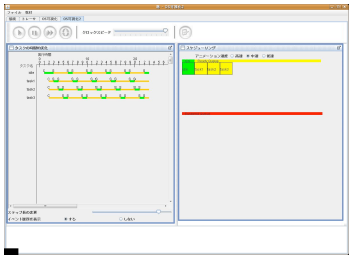

図 1　概念レベルの可視化ツール

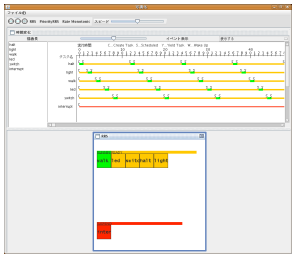

図 2　動作レベルの可視化ツール

概念レベルツールでは，教授者は理想的な状態の可視化を学習者に提供できる．再現が難しい同期やスケジューリングの問題を可視化して見せることが可能である．これに対して，動作レベルツールは，学習者自身が課題の演習を通して，問題を発見する手助けのための可視化ツールである．学習者がプログラムを実行した結果のログを取得し，可視化を行う．

3.　要求分析

3.1　Linux 対応における要求

本システムは，これまで miniOS（学習用に開発した OS）および TOPPERS/JSP，LEGO Mindstorms NXT を対象としたログ取得機構を用意してきた．これらは OS コードおよびライブラリの変更で対応可能であったが，Linux の場合，OS の規模が大きいこと，複数のアーキテクチャで動作しコードの変更が行われることから，ターゲット OS のコード変更をできるかぎりなくしたい．

また，マルチコアの利用が始まっていることから，これに対応したログ取得および可視化機構が必要となる．

3.2　可視化環境の要求

概念レベルのツールと，動作レベルのツールは，開発時期が異なることから，ユーザインタフェースが異なっている．このために，学習者は操作を 2 回覚えなければならず，学習負荷が高い．著者らが過去の行った実験でも，本システムのユーザインタフェースについては低スコアであり，学習者に使いやすく分かりやすいインタフェースを提供する必要がある．

4.　「港」システムの可視化環境の改良

4.1　全体構成

図 3 に全体構成を示す．

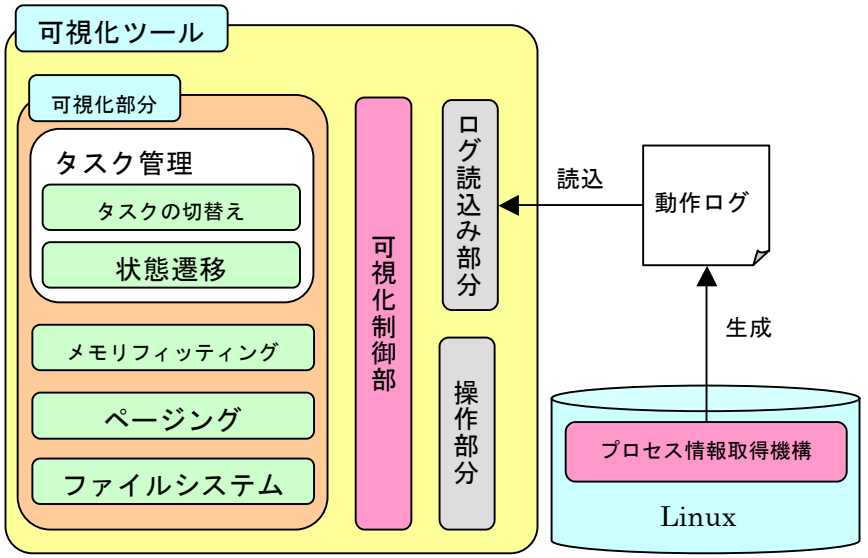

図 3　全体構成図

本システムでは，OS からプロセスの動作ログを取得するプロセス情報取得機構と，そこで得られたログを元に可視化を行う可視化ツールから構成する．可視化ツールでは，これまで概念レベルと動作レベルで個別に実現されてきたシステムを統合して扱えるように，可視化制御部を設けた．

GUI によるタスク生成などのユーザの操作は，操作インタフェースを通して可視化制

Vol.2010-SLDM-144 No.39
Vol.2010-EMB-16 No.39
Vol.2010-MBL-53 No.39
Vol.2010-UBI-25 No.39
2010/3/27

御部を呼び出す．ログについても，ログ解析後同様に可視化制御部を呼び出す．このようにすべての操作が可視化制御部を通すようにすることで，一つの可視化ツールを，状況に応じて利用することが可能になる．

4.2 可視化制御部

可視化制御部は，可視化ツールの制御を行うものである．可視化制御部が提供する機能は次に挙げるものである．

(1) 仮想的な OS のインタフェース

プロセスに関するユーザによる操作は次のようなものがある．

- タスク生成，終了
- 優先度の決定スケジューラの選択
- タスクスイッチ

これらの操作は実際には OS 内部で行われる処理であるが，本システムではユーザが自分でスケジューラの選択やタスクの生成などを行い，可視化を行うことでスケジューリングなどの動作の仕組みを容易に理解するために仮想インタフェースを提供する．実際の OS は，仮想インタフェースに対応づけられる．

(2) ログファイルの読込みとログによる動作

実際に動作する OS と可視化ツールとの連動を実現するために，ログファイルの読込みとそのログによる可視化の動作を実現する．

読み込まれたログファイルの内容を解析して，生成されているタスクの個数や現在の状態などのデータを一つのまとまりとして格納しておく．格納されたデータを可視化部分に渡すことで，同一時間のすべてのタスクの状態を表示することができる．

(3) システム全体の制御

システム全体の制御は次に挙げるものがある．

- 実行開始
- 実行停止
- 可視化速度の変更
- 表示のリセット

これらの操作は，ボタン操作などの GUI による操作で行われる．可視化制御部ではこれらの操作に対応する処理を各可視化画面に対して行う．

4.3 OS との連動

可視化ツールと実際に動作している OS との連動を行うために，本システムではログファイルを利用しそのログを読み込むことで実現する．ログの取得は，実際の OS を動作させてその後に OS がログファイルを出力するようになっている．動作ログの内容を図 4 に示す．

ログは，複数の OS に対応可能なように一般的な OS の状態を提供する．UNUSED は，タスクが生成される前の状態を表現するものであり，可視化に影響を与える．これについては，5.3 で後述する．また，リアルタイム OS で重要となる，優先度や周期，デッドライン情報のフィールドを持つ．全体は，ログ取得の粒度によって決まったタイムスタンプによって，時間の制御を行う．

```
<time> ::= “timestamp” <時間>
<state> ::= “state” <状態> [<args>]*
        |<priority> ::= “priority” <優先度>
        |<rate> ::= “rate” <space>
        |<deadline> ::= “deadline” <デッドライン>
        |<name> ::= “name” <タスクの名前>
<stop> ::= “STOP”
<id>   ::= <タスク id>
<状態> ::= “RUNNING” | “READY” | “SUSPEND” | “UNUSED”
```

図 4 ログファイルのフォーマット

図 5 にログファイルの実行例を示す．state に続いて状態を表し，その状態を取るタスク id が次に続く形式となっている．

Vol.2010-SLDM-144 No.39
Vol.2010-EMB-16 No.39
Vol.2010-MBL-53 No.39
Vol.2010-UBI-25 No.39
2010/3/27

```
timestamp 434405
state READY
3612
state UNUSED
3613
state UNUSED
3614
state UNUSED
3615
STOP
timestamp 434405
state RUNNING 0
3612
state READY
```

図５　ログファイルの例

5. Linux 可視化機構の設計

5.1 プロセス情報取得機構

図 6 にプロセス情報取得機構の構成図を示す.

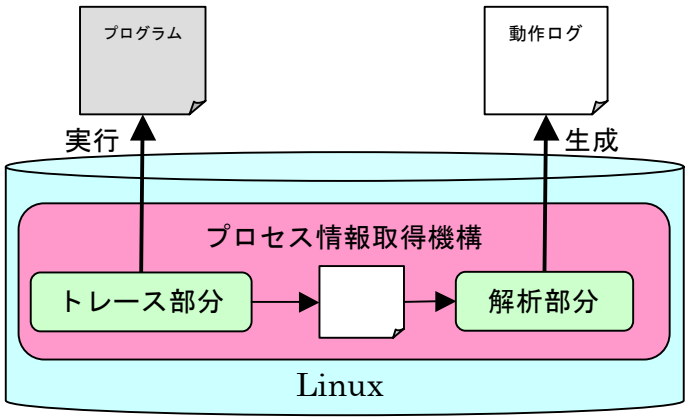

図６　プロセス情報取得機構

プロセス情報取得機構は，トレース部分と解析部分から構成されている．トレース部分では，プログラムを指定して実行させ，そのプログラムのトレースを行い，トレース結果を出力する．解析部分ではトレース部分によりトレースした結果を解析し，「港」システムで利用できる形のログファイルを出力する．このように分割することで，組込みシステム開発のように，開発マシンとターゲットマシンとが分離している場合でも，トレース部分だけをターゲットマシン上で実行することが可能である．

5.2 ftrace の利用によるプロセス情報の取得

プロセス情報取得機構では, Linux の機能である ftrace[6]を利用して情報の取得を行う．ftrace は Linux にある機能であり，関数のトレースやメモリ内容のトレース，プロセス状態のトレースなど，システム内部の多くの情報を，コードを変更せずにトレースできる．ftrace を利用することで得られる，プロセスに関係する情報は次のものである．

- 実行中のプロセス名，ID，優先度
- 割り当てられている CPU
- 実行開始時間
- トレースしている状態
- スイッチ後に遷移する状態
- スイッチ先のプロセス名，ID，優先度

表１にトレースしている状態の一覧とその状態に対する意味を示す．ftrace では，プロセス生成，待ち状態から実行可能状態への遷移，実行可能状態から実行状態への遷移の３状態を得ることができる（Linux では，実行可能状態と実行状態は区別されていない）.

表１　トレースしている状態

トレースしている状態	意味
sched_wakeup_new	新たにプロセスが生成された
sched_wakeup	待ち状態から実行可能状態への遷移
sched_switch	実行可能状態から実行状態への遷移（コンテキストスイッチ）

Vol.2010-SLDM-144 No.39
Vol.2010-EMB-16 No.39
Vol.2010-MBL-53 No.39
Vol.2010-UBI-25 No.39
2010/3/27

ftrace を実行すると，トレース対象であるプロセスが実行を終了して消滅するまでの間，プロセスの状態遷移の情報を取得する．この ftrace を利用して本システムでは，対象となるプログラムをプロセス情報取得機構から実行させて，その際に生成されたプロセスのトレースを行う．トレース結果は，proc ファイルシステムを通して，テキストでアクセス可能である．テキストでアクセスできることから，プログラムでのアクセスおよび後段での加工が容易になっている．

図 7 に ftrace を利用してプログラムをトレースした際に出力される内容を示す．

```
sh-3613  [000]  1097.435157: sched_wakeup_new: task sh:3614 [120] success=1
sh-3613  [000]  1097.435160: sched_switch: task sh:3613 [120] (R) ==> sh:3614 [120]
test-3614  [000]  1097.435718: sched_wakeup_new: task test:3615 [120] success=1
test-3614  [000]  1097.435721: sched_switch: task test:3614 [120] (R) ==> test:3615
[120]
echo-3615  [000]  1097.437413: sched_wakeup: task echo:3615 [120] success=0
echo-3615  [000]  1097.437505: sched_switch: task echo:3615 [120] (x) ==> test:3614
[120]
```

図 7　ftrace の実行結果

各行は，プロセス名，プロセス id，CPU 番号，タイムスタンプ，状態，状態遷移情報を含んでいる．

図 7 において，1 行目はプロセス ID : 3613 で sh というプロセスがプロセス ID : 3614 で sh というプロセスを生成したことを意味している．5 行目はプロセス ID : 3615 で echo というプロセスがプロセス ID : 3615 で echo というプロセスを実行可能状態に遷移させようとしたが同じプロセスを指しているため失敗していることを意味している．6 行目は，プロセス ID : 3615 で echo というプロセスから，プロセス ID : 3614 で test というプロセスに実行が切り替わり，切り替わった後にプロセスが消滅することを意味している．

5.3 「港」システムとの連動

前節で述べたように，ftrace を利用することでプロセスの状態遷移や切り替えの様子をトレースすることができるが，図 9 のような出力結果のままでは「港」システムで利用できない．そこで，プロセス情報取得機構では，出力結果を解析して「港」システムで利用できるログフォーマットでログファイルを出力することで「港」システムとの連動を行う．また，プロセス管理の可視化を行うために次の機能を「港」システムに追加した．

(1)　マルチコアへの対応

マルチコアの CPU での Linux のプロセスは，CPU の数だけ複数のプロセスが同時に実行されている．そこで，ログファイルの実行状態に CPU 番号を引数として追加した．また，それぞれの色を対応させることで複数の CPU で実行されていることをわかりやすく示している．表 2 に新たに定義する状態と色を示す．

表 2　新たに定義する状態と色

状態	意味	色
RUNNING 0	0 番の CPU で実行	緑
RUNNING 1	1 番の CPU で実行	青
RUNNING 2	2 番の CPU で実行	水色
RUNNING 3	3 番の CPU で実行	紫

既存のシステムでは，実行状態を緑で示していた．新たに設けた実行状態を示す状態についても同系色を利用することで，それが実行状態であり，かつどの CPU で実行されているか示すようにする．こうすることで，他の連動している OS の可視化を行った際と同じような表現にすることができ，学習者にとって理解しやすくなる．

(2)　プロセスの親子関係を示す

Linux のプロセスには必ず親子関係が存在する．そのため，Linux のプロセス管理を可視化する際には，生成されたプロセスの親子関係を示す必要がある．本システムでは，プロセスが生成される前の状態を UNUSED という状態で定義しておく．この状態の場合，可視化画面には何も表示されないようにする．また，生成されたプロセスの親プロセスがどのプロセスなのか明確に示すため，可視化画面上で親子関係にあるプロセスを矢印でつなぐようにする．こうすることで，学習者に対してプロセスの親子関係を明確

Vol.2010-SLDM-144 No.39
Vol.2010-EMB-16 No.39
Vol.2010-MBL-53 No.39
Vol.2010-UBI-25 No.39
2010/3/27

に示すことができる．

5.4 ユーザインタフェース

図 8 に改良した可視化ツールの実行画面を示す．既存の可視化ツールを統合することで，一つのツールで連続した学習が行える環境を実現する．そのために，可視化画面や操作部分のレイアウトなどユーザインタフェースを統一させる必要がある．

また，それぞれの可視化において共通の操作である実行開始や停止，可視化速度の変更は可視化全体で共通して利用する操作なので，図 9 に示すようにツールバーとして常に画面に表示させておく．

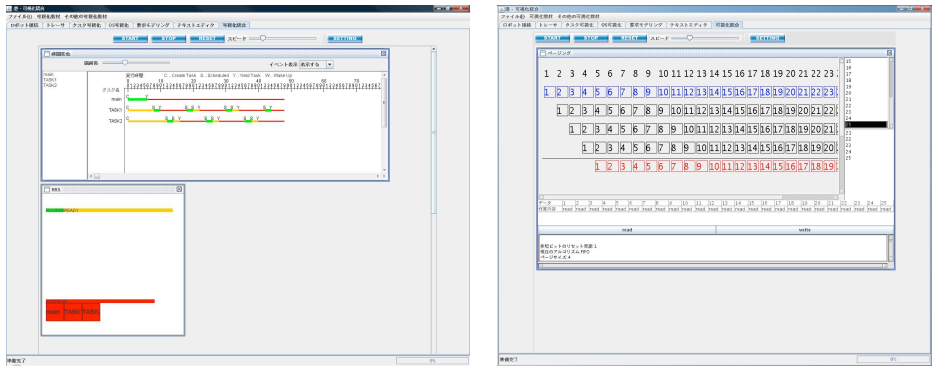

図 8　可視化ツールの実行画面

図 9　操作部分

ユーザ操作による可視化の設定は，ツールバー上に設定ボタンを設けることで，どこで設定を行うか明確に示すようにする．設定ボタンは，可視化制御部で監視されており，表示されている可視化画面に対応した画面が図 10，図 11 のようにポップアップで表示されるようになっている．

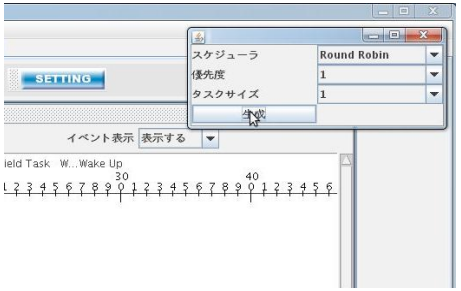

図 10　タスク管理の設定画面

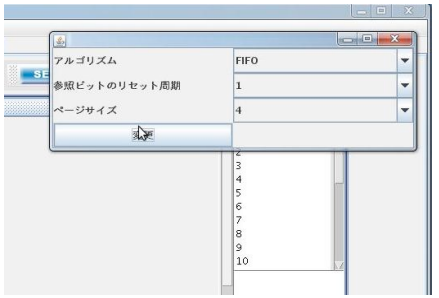

図 11 ページングの設定画面

ログファイルの読込みや可視化教材の選択はそれほど頻繁に行われる操作ではないため，ファイルメニュー内に表示する．こうすることで，可視化画面の表示領域を大きく確保することができ，より可視化画面を見やすくすることができる．

5.5 プロセス可視化の実行例

本システムでは，C 言語で記述されているプログラムなら内容に関わらずトレースすることが可能となっている．また，対象となるプログラムはプロセス情報取得機構から実行されるため，あらかじめコンパイルさせておく必要がある．図 12 にトレース対象となるプログラムの一例を示す．

```
// 新しいプロセスを作る
  // カーネルは，同じプロセスをもう 1 つ作る
  pid = fork();

// 子プロセスの pid は，0
  if (pid == 0)
  {
    execl("/bin/echo", "echo", "test", NULL);
    exit(-1);
  }
```

図 12　トレースするプログラム例

Vol.2010-SLDM-144 No.39
Vol.2010-EMB-16 No.39
Vol.2010-MBL-53 No.39
Vol.2010-UBI-25 No.39
2010/3/27

プログラム例では，プロセスを生成し，子プロセスがこのプログラムを，ftrace を利用してプロセスのトレースを行う．さらにトレース結果を解析することで「港」システムで利用するログファイルの出力を行う．出力されるログファイルは図 5 のように出力される．また，先の改良方針にしたがって改良した「港」の可視化画面における Linux のプロセス管理の可視化画面を図 13 に示す．

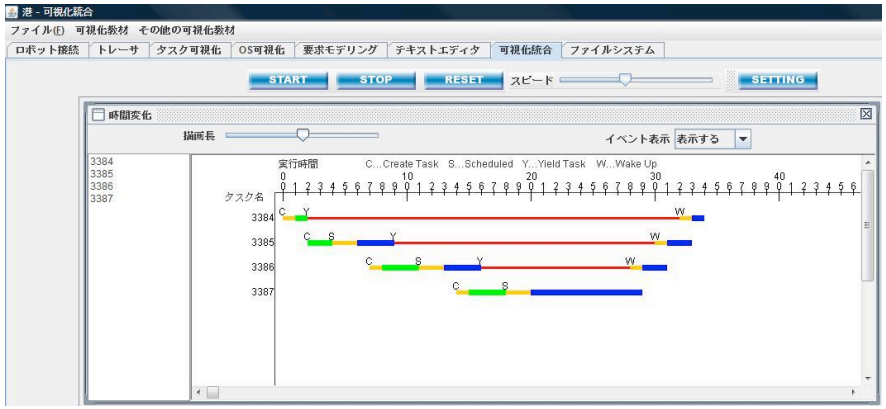

図 13　改良した可視化画面

この図では，プロセスがどの CPU で実行されているかを，色を変えることで分かるようになっている．また，プロセスが生成されたタイミングで可視化画面に表示されるようになっているため，学習者はプロセスの生成の流れを容易に知ることができる．また，全体と局所とを切り替えて見ることができるように，全体の描画長をスライダで切り替えることができる．また，上部にあるスピードのスライダを使って実行スピードを調節することができる．このスライダや START,STOP などのボタンは，概念レベル可視化ツールでも同様の操作を提供する．

このような機能を提供することによって，プロセスを概念レベルで可視化，操作して，学習したのち，自分たちでマルチプロセス／スレッドのプログラムを作成して，システムがどのようにスケジューリングするかを確認することが容易になっている．

6. おわりに

本システムでは，「港」システムにおける OS の可視化環境の統合を行い，概念レベルの学習から動作レベルまでの学習を連続的に一つのツール内で行えるようにした．また，プロセス管理の可視化を行うためにプロセス情報取得機構の実装を行い，さらに取得した情報を「港」システム上で可視化するために可視化環境の改良を行った．

今後の課題は，統合した「港」システムの有用性の検証や，Linux のほかの機能の可視化機構の実現である．

参考文献

[1] 経済産業省 : 2009 年度組込みソフトウェア産業実態調査報告書（プロジェクト責任者向け調査）
http://www.meti.go.jp/policy/mono_info_service/joho/downloadfiles/2009software_research/09project_houkokusyo.pdf

[2] Mathieu Desnoyers and Michel Dagenas, “OS Tracing for Hardware, Driver and Binary Reverse Engeneering in Linux”, CodeBreakers Journal Article, vol.4, no.1, 2007.

[3] 後藤隼弐，本田晋也，長尾卓哉，高田広章，“トレースログ可視化ツールの開発”，情報処理学会報告書, vol.2009, no.22, pp.73-78, 2009

[4] 大角他，システムソフトウェア教育支援環境「港」における実装レベルの OS 学習支援システム，Vol.2005，　No.15，pp.9-14，2005-CE-78-(2)

[5] 坂本他，OS 学習支援環境「港」の学習教材におけるユーザインタフェースに関する検討，第 70 回情報処理学会全国大会，2008

[6] ftrace tutorial
http://people.redhat.com/srostedt/