2007－OS－106 (7)
2007／8／3

マルチサーバ型OSにおけるドライバ即時復旧手法の提案

尾崎亮太† 日高宗一郎†
児玉和也† 丸山勝巳†

オペレーティングシステム（OS）構成部品の中で故障の多いデバイスドライバの障害時に，OS全体を再起動させることなくドライバ復旧を可能とする手法が提案されている．ドライバはOS本体からメモリ保護機構などで隔離され，メモリ保護違反などによって起きる障害からはすみやかに再起動される．しかし，無限ループやデッドロックなどのエラーによるドライバ無反応障害は，定期的にドライバ状態を監視しなければ検出できない．即座に検出するためには監視間隔を短くする必要があるが，監視オーバヘッドが大きくならないように監視方法は軽量であることが求められる．本研究では，ドライバ動作特徴を基にした軽量な監視方法を提案する．本手法は，一定期間を越えても1回の仕事を完了しないドライバをエラー状態とみなし，ドライバを復旧処理へ移行させる．提案手法をマルチサーバ型OSであるMINIX 3に適用した．

Quick Recovery of Failed Drivers on Multi-server Operating Systems

Ryota OZAKI,† Soichiro HIDAKA,† Kazuya KODAMA†
and Katsumi MARUYAMA †

Drivers are failure-prone in OS components and OSs have to be failure resilient to them. Current dependable OSs isolate drivers and recover failed them without OS reboot. The OSs isolate drivers using MMU or type-safe language, and can detect drivers' protection violation at once. However, silent failures caused by infinite loops or dead locks are to be detected using periodical monitoring. Although quicker detection of the failures are realized shorter periodical monitoring, monitoring must be lightweight to keep cost low. We have proposed a lightweight monitoring method based on the characteristic that drivers in normal condition repeat a cyclic routine and complete their job within a constant short period. In our method, drivers which cannot complete their job within the period of the time quantum are considered as abnormal. We have implemented the method on MINIX 3.

1. はじめに

多数のアプリケーションの稼働基盤となるオペレーティングシステム（OS）の頑強性の向上は重要である[12]．OSは，多数の機能部品（サブシステム）で構成され，非常に大規模であるため故障要因も多いが，近年では，OSの障害としてデバイスドライバの故障（ソフトウェアのバグ）に起因するものが特に多いことが明らかになっている[3),10)]．そのため，特にデバイスドライバ故障を考慮した耐故障機能をOSへ付加することが頑強性向上には重要である．

現在提案されているシステムは，メモリ保護機構や言語レベルの安全性チェックにより，ドライバのエラーがOSの他の部分へ波及することを防ぐ機構を導入している[4),7),10),11),13),14)]．ドライバのメモリ保護違反や型安全違反を検出し，障害のあったドライバのみを再起動することで，OS全体を止めることなくドライバを復旧する．しかしながら，無限ループやデッドロックなどによりドライバが反応しなくなる障害（無反応障害）には，単純なタイムアウト待ちが必要となるため障害検出が遅れる．既存システムではこの遅れは数秒になり，人間に知覚できるOS動作停止を引き起こす．

そこで本研究では，デバイスドライバのエラーを即座に検出する手法を提案する[8),9)]．本手法は，正常なドライバは単純な処理を繰り返し，一定の短い時間でその処理を完了するという特徴に着目し，一定期間を越えても1回の仕事を完了しないドライバをエラー状態とみなし，ドライバを復旧処理へ移行させる．これにより，ドライバ障害復旧時間を短縮させる．提案手法をマルチサーバ型OSであるMINIX 3に適用した．

† 情報・システム研究機構 国立情報学研究所, National Institute of Informatics, Research Organization of Information and Systems.

以降，本論文は次のように構成される．2章で近年提案されているOSの頑強性向上手法について概観し，その問題点について述べる．3章で問題点を克服する提案手法の設計，4章で提案手法のMINIX 3への実装について述べる．5章で残された検討課題について述べ，6章で本論文をまとめる．

2. OS頑強性向上

2.1 ソフトウェアシステムの頑強性向上

ソフトウェア故障から障害発生を低減し，安定したサービスを提供する技術は長く研究されてきた．疎に結合した複数の部分システムからなる分散システムでは，冗長性を持たせ待機システムに切替えたり，チェックポイントを取り，部分システムの障害発生時に処理を途中から再開することでサービス復旧させる手法を取ることが可能である．しかし，単一大規模システムでは，故障はシステム全体に影響を及ぼし，サービスを復旧するためには，システム全体を再起動しなければならないことが多い．そのためシステムの頑強性は大幅に低下する．

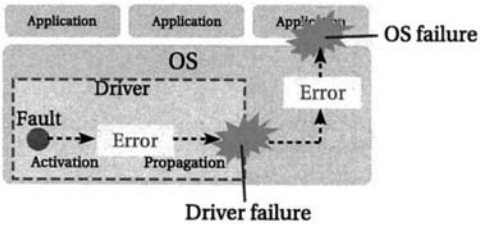
George Candeaらは，単一大規模システムを小さなサブシステムへ分割し，サブシステム単位で障害から高速復旧することによりシステム全体の可用性を向上する手法microrebootを提案している[2]．近年では，同様の設計思想をOSに対して適用した研究が多くなされている[4),7),10),11),13),14)]．

2.2 OS部分復旧

OSは多数の機能部品（サブシステム）で構成され，非常に大規模であるため故障要因も多いが，近年では，OSの障害としてデバイスドライバの故障（ソフトウェアのバグ）に起因するものが特に多いことが明らかにされた[3),10)]．例えば，Windows XPのクラッシュ原因の85%はドライバに依るものであり[10)]，Linuxのドライバ内のコーディングミスによるエラーの発生率はそれ以外の部分の3～7倍にも及ぶ[3)]．そのため，特にデバイスドライバ故障を考慮した耐故障機能を付加することがOS頑強性向上には有効である．

図1は，ドライバ故障とそれによってOS障害が引き起こされるまでを図に表わしたものである．ドライバ故障はドライバ内のエラーを発生させ，ドライバ障害につながる．ドライバ障害はOS内の他の部分へ波及し，最終的にアプリケーションに対するサービスが停止する．そのため，ドライバ故障に起因するOS障害を防止するには，まずドライバエラー波及を食い止めなければならない．

OS部分復旧手法では，メモリ保護機構や型安全性



ドライバ障害はOSのその他の部分へ伝搬し，最終的にOSの障害へと繋がる．

図1 OSドライバ障害モデル
Fig. 1 Driver failure model in an OS.

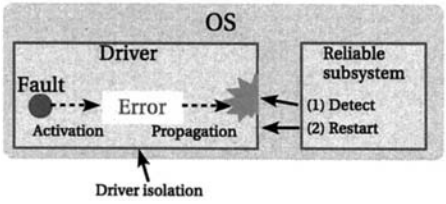


図2 ドライバ隔離と信頼サブシステム
Fig. 2 Driver isolation and the reliable subsystem.

言語などを用いてドライバをOS本体から論理的に隔離する（図2）．また信頼サブシステム（reliable subsystem）がOS本体とドライバ間の入出力の監視やドライバ状態の監視を行ない，障害につながるエラーが検出されたドライバを再起動する．障害のあったドライバのみを再起動させることで，OSを稼働したままドライバ障害から復旧する．これによりOS全体を再起動させる場合に比べ，障害期間を短く抑えることができる．

2.3 OS部分復旧のためのドライバ隔離手法

ドライバ隔離手法としては，型安全言語，仮想計算機，仮想メモリ機構による方法が提案されている．

SafeDrive[14)]では，言語ベース手法でOSの拡張機能（特にドライバ）の障害から復旧する．SafeDriveは，プログラマがポインタ変数に付けたアノテーションを基に故障検出コードを自動挿入するDeputyと，障害が起きた拡張機能を復旧するリカバリシステムで構成される．SafeDriveは，言語レベルで判明する配列の境界違反などのエラーを検出可能であるが，無限ループなどのエラーは検出対象ではない．

Joshua LeVasseurらはL4マイクロカーネル上にLinuxカーネルを2つ用意し，一方でアプリケーション，他方でデバイスドライバを動作させることでデバイスドライバの障害がアプリケーションへ波及することを防いでいる[7)]．既存のLinuxデバイスドライバの改変がほとんど必要ないため，コード追加による新た

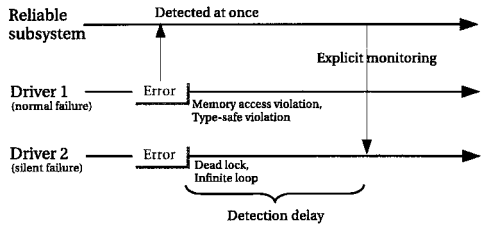


図 3 ドライバエラー検出遅延
Fig. 3 Detection delay of driver errors.

なバグが入ることは少ない．

Nooks[10] は，Linux カーネルの拡張機能をカーネル本体から分離することで，OS の信頼性を高める OS サブシステムである．拡張機能実行時にカーネル内の他のメモリ領域を書込み不可能にすることで，拡張機能障害のカーネル本体への波及を防ぐ．クラッシュした拡張機能は，復旧エージェントにより自動的に再起動される．Shadow Driver[11] は，Nooks をさらに発展させ，アプリケーションが設定するデバイス固有情報も復旧させることで，デバイスドライバ障害とその復旧処理をアプリケーションから隠蔽可能とする OS サブシステムである．

MINIX 3[4] は，マイクロカーネル＋マルチサーバ構成の OS である．ドライバは個別のユーザプロセスとして実行されるため高い耐障害性をもつ．一般的なユーザプロセスと同様に，ドライバの障害が他へ波及しない．またドライバ再起動はユーザプロセス再起動という形で実現できる．

Microdrivers[13] は，性能に影響する一部を除き，ドライバをユーザプロセスとして動作させる手法である．ユーザプロセスとして実行されるドライバの障害がカーネルの障害に即座には繋がらず，障害後のドライバ再起動も容易である．また頻繁に実行されるドライバコードはカーネル内に存在するため性能低下が少ない．

いずれのシステムもドライバエラーを検出した後，OS 全体を止めることなくドライバを再起動させることでサービスを継続し OS 頑強性を向上している．

2.4 ドライバエラー検出遅延

既存の手法はドライバ復旧に大きな遅れを発生させる可能を残す．不正なドライバ入出力やドライバのメモリ保護違反などのエラーによる一般的な障害は信頼サブシステムが即座に検出可能であるが，無限ループやデッドロックなどのエラーによる無反応障害は信頼サブシステムの定期的な監視を必要とする．監視間隔が長いとその分だけエラー検出が遅れ，結果としてドライバ復旧も遅れる．しかし，監視間隔を短くすると単位時間あたりの監視回数が増え，監視オーバヘッドが増加する．

3. 軽量なドライバエラー検出手法

本章では，短いドライバ監視間隔でも監視オーバヘッドが増加しない軽量なドライバエラー検出手法をの設計について述べる．

3.1 設　　計

本手法は，マルチサーバ型 OS を前提に設計する．ドライバは個別のユーザプロセス（以下タスクと呼ぶ）として実行され，ドライバへの処理要求や割込みはプロセス間通信（Inter Process Communication: IPC）でドライバへ通知される．

本手法は，デバイスドライバの正常動作である
（a）一連の動作を繰り返す
（b）一定の短い時間で処理を終える
という特徴を基に設計する．一定期間を越えても 1 回の仕事を完了しないドライバをエラー状態とみなし，ドライバを復旧処理へ移行させる．マルチサーバ型 OS のドライバタスクは
(i)　要求・イベント待ち
(ii)　実処理
(iii)　返答
の処理サイクルを繰り返す．そのため仕事の開始と終了はそれぞれ IPC の受信処理と送信処理とみなすことができる．受信処理の後に一定期間経っても送信処理が行なわれなかった場合，無限ループやデッドロックなどのエラーと判断する．この判断（以下エラー判定）は，タスクの CPU 割当て時間（タイムクォンタム）全消費時，タイマ割込み時，タスクスイッチ時などのタイミングで行なう．ドライバ IPC の監視やエラー判定はカーネル内で行なわれるため，エラー検出処理コストは小さい．

3.2 適用範囲

ドライバタスク復旧は，一時的なハードウェアエラーによるソフトウェアの異常動作や，無限ループやデッドロックなどのエラーに対して有効である．逆に，永続的なハードウエアエラーや悪意あるプログラム，例えば故意に正しくない値を返すようなプログラムからシステムを守ることは想定していない．

本手法は，以下の条件を満たすドライバに適用することができる
(1)　処理が単純なサイクルである．
(2)　処理の開始処理と終了処理が明確である．
(3)　1 つのリクエストの処理時間が予測可能である．

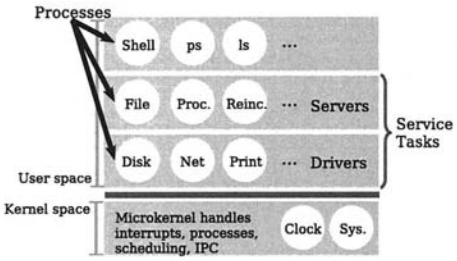


図 4　マルチサーバ型 OS のシステム構成（MINIX 3）
Fig. 4　Configuration of a multi-server OS : MINIX 3.

4. 提案手法の MINIX 3 への実装

提案手法をマルチサーバ型 OS である MINIX 3 へ適用する．

4.1 MINIX 3

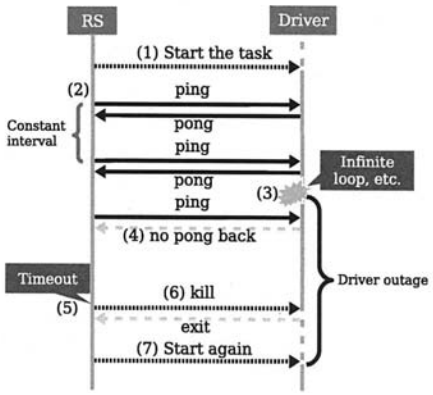
MINIX 3 は，カーネルモードで実行されるマイクロカーネルと，個別の論理空間をもちユーザモードで実行される各種の OS サービスタスクで構成される（図 4）．マイクロカーネルは，割込みハンドリング，CPU や MMU に関する特権処理，プロセスやスレッドのスケジューリング，IPC などを行なう．OS サービスタスクには，ファイルシステムやネットワーク処理の他に，ディスクデバイスやネットワークデバイスなどのデバイスドライバがある．

MINIX 3 では，サービスタスクを定期的に監視する Reincarnation Server（RS）によってタスク再生が行なわれる．サービスタスクは，親プロセスとなる RS によって定期的に IPC で監視され，返答がなければそのサービスタスクが異常状態になったと判断し，強制終了させたのち再起動させる（図 5）．

MINIX 3 における生存確認は IPC を用いた Heartbeat によるため，いくつかの問題点が存在する．

- 復旧開始までの遅延が長い：MINIX 3 では，確認間隔の 2 倍を越えても，サービスタスクからの返答がなかった場合，障害と判断する．確認間隔を T とした場合，最大 2T 期間だけ障害発見が遅れる．
- コードの改変が必要：サービスタスクには，RS の生存確認メッセージに返答するためにコードの改変が必要となる．
- サービスタスク数に従い処理量の増大する：サービスタスク数を n とすると，確認間隔 T の間に $n \times 2$ 回の IPC が発生する．

提案手法は，これらの問題点を克服するように設計されている．



(1)　RS がサービスタスクを生成する．
(2)　RS はサービスタスクに IPC で ping メッセージを送信し，サービスタスクはこれに pong メッセージで返答する．
(3)　サービスタスクが無限ループやデッドロックに陥る．
(4)　RS の ping にサービスタスクが pong を返さなくなる．
(5)　pong 待ちがタイムアウトし，RS はサービスタスクの障害を検出する．
(6)　RS はサービスタスクを kill，同タスクは exit する．
(7)　RS は exit を確認した後，サービスタスクを再度起動させる．

図 5　サービスタスク再生
Fig. 5　The sequence of recovery an anomalous driver on the MINIX 3.

4.2 タイムクォンタムベースの実装

ここでは，エラー判定のタイミングにタイムクォンタムを用いる実装について述べる．

MINIX 3 は，一般的な OS と同様にハードウェアクロック割込みを使い，時間（経過）を管理している．MINIX 3 のタイマ処理は，クロック割込み毎に呼ばれる **clock_handler** 関数と，必要なときにのみ呼び出される **do_clocktick** 関数の 2 つに分けられる．**clock_handler** は OS 内時間（tick）をカウントアップさせ，実行中のプロセスのタイムクォンタムを tick 分減らす．**do_clocktick** は，alarm(2) などで設定した時限イベントを発行する必要がある場合か，実行中のプロセスがタイムクォンタムを使い切った場合に **clock_handler** からのメッセージで呼び出される．**do_clocktick** は，タイムクォンタムを使い切ったプロセスに再度タイムクォンタムを割り当て，ペナルティとしてそのプロセスの優先度を下げる．

このように割り当てられた CPU 時間は，タスク切替えを跨いで残量がカウントされる．例えば，100 ms のタイムクォンタムを割り当てられたタスクが，10 ms だけ CPU 時間を消費してタスク切替えが起きた場合，次に起動した時のそのタスクの残り CPU 時間は 90 ms となる．そのため単純にタイムクォンタムを使い切ったからといって，エラーが起きたと断定することはできない．そこで監視対象ドライバタスクへ

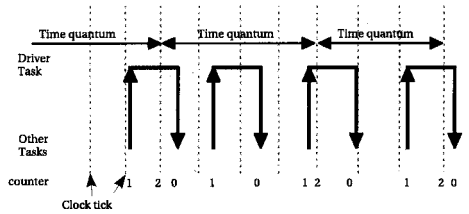


図 6 ドライバタスク処理と Tick（正常時）
Fig. 6 Driver task processing and clock ticks (normal condition).

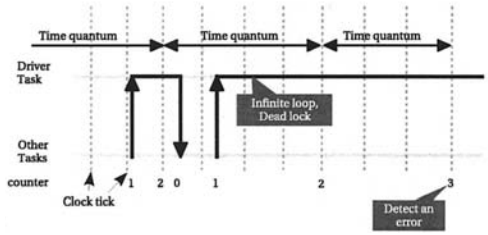


図 7 ドライバタスク処理と Tick（異常時）
Fig. 7 Driver task processing and clock ticks (anomalous condition).

counter を設け，以下のように counter の値を更新しエラー判定を行なう．

(1) ドライバタスクの処理開始箇所で counter を 1 にする．

(2) counter が 1 で，かつ，そのタスクがタイムクォンタムを使い切った場合 counter を 1 カウントアップさせる．正常動作時では，counter の値は一回しかカウントアップされない（図 6）．

(3) もし counter が規定数（現在は 3）以上になった場合，ドライバタスクが異常動作をしていると判断し，RS へ障害を通知する（図 7）．

(4) ドライバタスクの処理終了箇所で counter を 0 にする．

その他，提案手法を実現するために，MINIX 3 にいくつか修正を加えた．まず再生すべきドライバタスクをカーネルへ教えるための，カーネルコールを用意した．ドライバタスクの起動や，エラーのあったドライバタスクの停止，再起動などは RS の機能をそのまま利用した．

提案するエラー検出手法を用いた場合のドライバタスク復旧の流れは図 9 の通りである．

5. 議　　論

5.1 オリジナル手法との比較

表 1 に MINIX 3 オリジナルの手法との比較につい

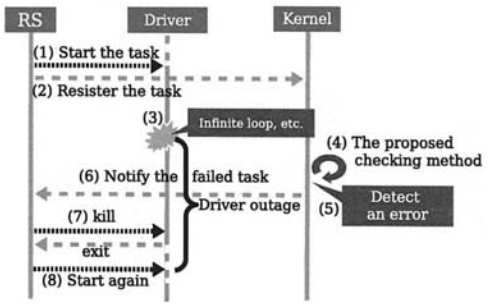
```
/********* proc.c **************/
// mini_send() : タスクが send を発行
task->counter = 0;

...

// mini_receive() : タスクが receive を発行
task->counter = 1;

/********* clock.c **************/
// do_clocktick() : タスクがタイムクォンタムを使い切った
if (task->counter > 0) {
  task->counter++;
  if (task->counter > 2) {
    // 障害発生を RS に通知する
  }
}
```

図 8 エラー判定のための追加コード（主要部分）
Fig. 8 Additional mainly code to detect driver errors.



(1) RS がドライバタスクを生成する．
(2) RS はドライバタスクの監視をカーネルに依頼する．
(3) ドライバタスクが無限ループに陥る．
(4) 提案する手法でドライバタスクをチェックする．
(5) カーネルがドライバタスクの異常を検出する．
(6) カーネルは，RS へタスクの異常を通知する．
(7) RS はドライバタスクを kill，同タスクは exit する．
(8) RS は exit を確認した後，ドライバタスクを再度起動させる．

図 9 提案する異常検出手法を用いた場合のドライバタスク復旧
Fig. 9 The sequence of recovery an anomalous driver on the proposed method.

表 1 MINIX 3 オリジナル手法との比較
Table 1 Comparison with the original.

	チェック方法	ドライバ修正	対タスク数コスト増加比
オリジナル	IPC	必要	大
提案手法	カーネル内	不要	小

てまとめた．オリジナルの手法では，チェック間隔は標準では 1,000 ms（最小値 100 ms）である．一方，提案手法では，チェックはタイムクォンタム（100 ms）全消費毎に行なわれる．チェックコストは，既存手法はチェック間隔毎に $IPC \times 2 \times n$（n：タスク数）である．一方，提案手法では，IPC 毎に代入文 1 回およびタイムクォンタム消費毎のエラー判定がカーネル内で行なわれる．

5.2 ドライバ障害誤認識

提案手法はドライバの振る舞いから推定するため，

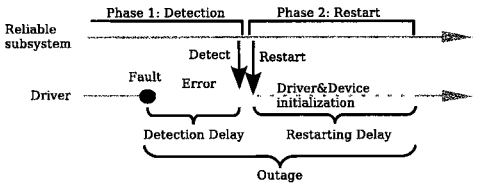


図 10 ドライバ復旧
Fig. 10 Recovery delay of a failed driver.

正常なドライバをエラーと誤認識し，そドライバを再起動してしまう可能性がある．そのため，エラー判定までの時間は適切に設定しなければならない．現在は，事前にドライバ処理時間を計測したのち値を決定しているが，現在の実装では 100 ms オーダの間隔のため，ドライバ処理がその判定時間を越えることはほとんどない．また後述のソフトウェア若化手法にみられるように，誤認識頻度が低いならば，定期的な再起動は悪影響しかないとは必ずしも言えない．しかしながら，タイマ割込み（MINIX 3 では 16.6 ms 間隔）を用いる実装ではより慎重にエラー判定時間を設定する必要がある．

5.3 ソフトウェア若化手法

部分復旧手法と同様にソフトウェアの頑強性向上を目的とする手法として，ソフトウェア若化を行なう手法が提案されている[5),6)]．ソフトウェア若化は，ソフトウェアエイジングにより発生するエラー（例えばメモリリーク）からシステムを守るために，障害発生前に定期的にシステムを再起動する予防的障害対策である．この手法は，システムが正常な段階で再起動を行なうため，OS 部分復旧手法のような障害時の問題（ドライバ隔離など）を考慮する必要がないため，実現すべき機能が少なくて良い．しかし，ドライバ障害要因としては，ソフトウェアエイジングだけではなくアプリケーションからの入力に起因するものも多い．そのため耐故障機能を提供する OS 部分復旧手法も併用することが望ましいと考えられる．

5.4 ドライバ初期化遅延

ドライバ復旧時間は，ドライバエラー検出遅延とドライバ初期化時間の和である（図 10）．ドライバ初期化時間を削減することでドライバ復旧時間をさらに短縮することができる．ドライバ初期化時間は，デバイス初期化時間の占める割合が大きい．ドライバエラーが起きたとしても，デバイスも同時に異常状態になるとは限らない．デバイス初期化を行なわず，ドライバ（ソフトウェア）の初期化のみ行なうだけでドライバ復旧できる可能性も存在する．このデバイス初期化を行なわないドライバ復旧手法については，今後検討していく予定である．

6. おわりに

OS の頑強性向上手法として，OS 部分復旧手法が提案されている．メモリ保護機構などでドライバを論理的に隔離し，障害のあったドライバのみを再起動することで，OS 全体を止めることなくドライバを復旧する．しかし，無限ループやデッドロックなどによる無反応障害は，信頼サブシステムの定期的な監視を必要とする．監視間隔が長いとその分だけエラー検出が遅れ，結果としてドライバ復旧も遅れる．

本論文では，ドライバタスク処理の正常終了時の状態が一定であることに着目した低遅延なタスク障害検出手法を提案した．提案手法によりドライバ無反応障害から素早く復旧することができる．また MINIX 3 を対象に，タイムクォンタムを用い 100 ms オーダでドライバエラー検出可能な提案手法の実装方法ついて述べた．

今後は，実システムによる評価，タイマ割込みベースの実装，5.4 節で述べたドライバ初期化時間の短縮手法の検討を行なう予定である．

参考文献

1) Avizienis, A., Laprie, J., Randell, B. and Landwehr, C.: Basic Concepts and Taxonomy of Dependable and Secure Computing, *IEEE Transactions on Dependable and Secure Computing*, Vol.1, No.1, pp.11–33 (2004).
2) Candea, G., Kawamoto, S., Fujiki, Y., Friedman, G. and Fox, A.: Microreboot – A Technique for Cheap Recovery, Proc. *the 6th Symposium on Operating Systems Design and Implementation (OSDI '04)*, pp.31–44 (2004).
3) Chou, A., Yang, J., Chelf, B., Hallem, S. and Engler, D.: An Empirical Study of Operating Systems Errors, *ACM SIGOPS Operating Systems Review*, Vol.35, No.5, pp.73–88 (2001).
4) Herder, J.N., Bos, H., Gras, B., Homburg, P. and Tanenbaum, A.S.: Failure Resilience for Device Drivers, Proc. *the 37th International Conference on Dependable Systems and Networks (DSN '07)*, pp.41–50 (2007).
5) Ishikawa, H., Nakajima, T., Oikawa, S. and Hirotsu, T.: Proactive Operating System Recovery, Proc. *the 20th ACM Symposium on Operating Systems Principles (SOSP '05)*, poster session, pp.1–2 (2005).
6) Kourai, K. and Chiba, S.: A Fast Rejuvenation Technique for Server Consolidation with

Virtual Machines, Proc. *the 37th International Conference on Dependable Systems and Networks (DSN '07)*, pp.245–254 (2007).
7) LeVasseur,, J., Uhlig, V., Stoess, J. and Gotz, S.: Unmodified Device Driver Reuse and Improved System Dependability via Virtual Machines, Proc. *the 6th Symposium on Operating Systems Design and Implementation (OSDI '04)*, pp.17–30 (2004).
8) Ozaki, R., Hidaka, S., Kodama, K. and Maruyama, K.: Quick and Lightweight Detection of Anomalous Drivers in Multi-server Operating Systems to Improve Availability, Proc. *the 37th International Conference on Dependable Systems and Networks (DSN '07)*, fast abstract, pp.378–380 (2007).
9) 尾崎亮太, 日高宗一郎, 児玉和也, 丸山勝巳: マルチサーバ型OSにおける可用性の向上, 電子情報通信学会 技術研究報告 ディペンダブルコンピューティング研究会 (DC2006-81), pp.7–11 (2007).
10) Swift, M.M., Bershad, B.N. and Levy, H.M.: Improving the Reliability of Commodity Operating Systems, Proc. *the 19th ACM Symposium on Operating Systems Principles (SOSP '03)*, pp.207–222 (2003).
11) Swift, M., Annamalai, M., Bershad, B. and Levy, H.: Recovering Device Drivers, Proc. *the 6th USENIX Symposium on Operating Systems Design and Implementation (OSDI '04)*, pp.1–16 (2004).
12) Tanenbaum, A.S., Herder, J.N. and Bos, H.: Can We Make Operating Systems Reliable and Secure?, IEEE Computer, Vol.39, No.5, pp.44–51 (2006).
13) V. Ganapathy, A. Balakrishnan, M.M. Swift and S. Jha: Microdrivers: A New Architecture for Device Drivers, Proc. *the 11th Workshop on Hot Topics in Operating Systems (HotOS '07)*, pp.85–90 (2007).
14) Zhou, F., Condit, J., Anderson, Z., Bagrak, I., Ennals, R., Harren, M., Necula, G. and Brewer, E.: SafeDrive: Safe and Recoverable Extensions Using Language-Based Techniques, Proc. *the 8th Symposium on Operating Systems Design and Implementation (OSDI '06)*, pp.45–60 (2006).