特 別 研 究 報 告 書

題 目

Mint オペレーティングシステムにおける ヘルスチェック機能の実現

指導教員

乃 村 能 成

報 告 者

天野 正博

岡山大学工学部 情報工学科

平成 23 年 2 月 14 日 提出

要約

近年，計算機に対する不正な攻撃が増加し，巧妙化している．このため，オペレーティングシステム (OS) のセキュリティ向上への要求が高まっており，様々なシステムの研究が行われている．しかし，既存のセキュリティシステムの多くは攻撃対象となる OS 上に実装されている．このため，OS が乗っ取られた場合，セキュリティシステムを無効化される危険があり，信頼性の低下につながるという問題がある．これを防ぐためには，攻撃対象となる計算機を外部から監視する必要がある．

また，マルチコアプロセッサの登場やメモリの大容量化により計算機の性能が向上した．このため，1 台の計算機上で複数の OS を走行させる研究が活発であり，TwinOS や Mint オペレーティングシステム (Mint) が提案されている．

信頼性の高いセキュリティシステムを実現するため，TwinOS を用いたヘルスチェック機能が提案されている．ヘルスチェック機能は攻撃対象となる OS から独立に動作するセキュリティシステムである．セキュリティシステムが攻撃対象となる OS から分離されているため，OS が乗っ取られた場合でも，システムが無効化される危険が少ない．しかし，TwinOS はシングルコアプロセッサ上で実現されているため，TwinOS を用いたヘルスチェック機能はマルチコアプロセッサに対応していない．そこで，TwinOS の設計方針を引き継ぎ，マルチコアプロセッサを搭載した 1 台の計算機上で複数の OS を同時に走行させる Mint においてヘルスチェック機能の実現を行う．

本論文では，Mint におけるヘルスチェック機能の実現のための課題と対処を明確にした．具体的には，監視用 OS によるサービス用 OS の停止・再開，検査範囲や検査契機の検討，攻撃者によるサービス用 OS 停止の妨害，HDD を 1 台しかもたない計算機ではヘルスチェック機能が利用できないという課題がある．

課題のうち，監視用 OS によるサービス用 OS の停止・再開，HDD を 1 台しか持たない計算機上ではヘルスチェック機能を利用できないという 2 つの課題について対処を行い，実装した．

目次

図 目 次

表 目 次

第 1 章

はじめに

インタネットの普及により，ネットワークを介した計算機への攻撃が増加している．また，攻撃方法自体も単純な計算機環境の破壊目的からスパイウェアを用いた個人情報の収集や迷惑メール送信の踏み台などの営利目的の攻撃へ変わってきた．これらの攻撃はオペレーティングシステム (以降，OS と呼ぶ) 自体を改ざんし，管理者が容易に検出できないような攻撃方法をとる．このため，OS のセキュリティ向上への要求が高まっており，様々なシステムの研究が行われている．しかし，既存のセキュリティシステムの多くは攻撃対象となる OS 上に実装されている．このため，OS が乗っ取られた場合，セキュリティシステムを無効化される危険があり，信頼性の低下につながるという問題がある．これを防ぐためには，攻撃対象となる計算機を外部から監視する必要がある．

また，マルチコアプロセッサの登場やメモリの大容量化により計算機の性能が向上した．近年の計算機ではマルチコアプロセッサが主流となっている．このため，1 台の計算機上で複数の OS を走行させる研究が活発である．1 台の計算機上で複数の OS を走行させることで，計算機資源を余すことなく利用できる．代表的なものとして，仮想計算機 (VM：Virtual Machine) を利用する方式 (VM 方式) があり，Xen[1] や VMware[2] がある．また，実計算機に近い環境で複数の OS を走行させる方式として TwinOS[3] や Mint オペレーティングシステム [4](以降，Mint) が提案されている．

信頼性の高いセキュリティシステムを実現するため，TwinOS を用いたヘルスチェック機能 [5]，[6] が提案されている．ヘルスチェック機能は攻撃対象となる OS から独立に動作するセキュリティシステムである．セキュリティシステムが攻撃対象となる OS から分離されているため，OS が乗っ取られた場合でも，システムが無効化される危険が少ない．

TwinOS はシングルコアプロセッサ上で実現されているため，TwinOS を用いたヘルスチェック機能はマルチコアプロセッサに対応していない．このため，TwinOS の設計方針を

引き継ぎ，マルチコアプロセッサを搭載した 1 台の計算機上で，複数の OS を同時に走行させる Mint においてヘルスチェック機能の実現を行う．

本論文では，Mint におけるヘルスチェック機能の実現のための課題と対処を述べる．また，課題のうち，サービス用 OS 停止・再開機能についての設計と実装を示す．また，Mint の拡張によって先行 OS を HDD の占有なしで起動し，この先行 OS を監視用 OS とすることで HDD を 1 台しか持たない計算機でヘルスチェック機能を利用する方法を示す．

第 2 章

ヘルスチェック機能

本論文でのヘルスチェック機能とは，OS の検査を行い，異常を検出する機能である．異常を検出した場合，管理者に通知する．これにより，攻撃者による OS の不正な改ざんの検出と被害の拡大防止を行う．

ヘルスチェック機能は，ヘルスチェック機能が実装されている監視専用の OS(以降，監視用 OS) とサービス専用の OS(以降，サービス用 OS) の 2 つの OS を用いる．システムの管理者は，ヘルスチェック機能が実装されている監視用 OS を操作して OS の検査を行う．図 2.1 にヘルスチェック機能の流れを示し，以下で説明する．

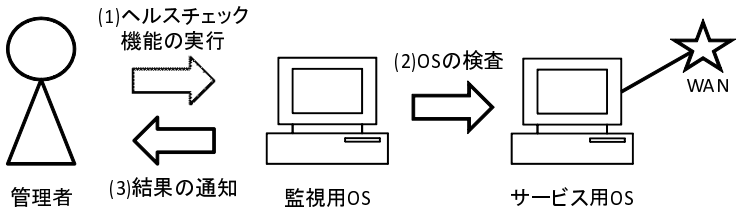

図 2.1 ヘルスチェック機能の流れ

(1) ヘルスチェック機能の実行

管理者が監視用 OS 上でヘルスチェック機能を実行し，監視用 OS はサービス用 OS の検査を行う．

(2) OS の検査

サービス用 OS のメモリの検査を行う．メモリ内のカーネルテキスト部などの通常変更されることがない領域を検査対象領域とする．このため，検査対象領域が変更されている場合，外部の攻撃者によって，OS が改ざんされていると判断する．

(3) 結果の通知

監視用 OS は，結果を管理者に通知する．

このように，ヘルスチェック機能は，監視用 OS がサービス用 OS を検査し，サービス用 OS が正常な状態か否かを管理者に通知する．しかし，監視用 OS がすでに外部から攻撃を受けていた場合など，監視用 OS が信頼できないと正しい検査を行えない．このため，監視用 OS を外部から分離することが重要である．したがって，監視用 OS はネットワークから切断した安全な環境で動作させる．

また，ヘルスチェック機能を利用するためには，監視用 OS とサービス用 OS の少なくとも 2 台の計算機が必要であり，コストが大きい．この問題を解決する方法として，1 台の計算機上で複数の OS を走行させる方式を利用することが挙げられる．

1 台の計算機上で複数の OS を走行させる方式は 2 つに分けられる．1 つ目は VM 方式であり，Xen や VMware がある．VM 方式は，仮想マシンモニタ (VMM：Virtual Machine Monitor) により，実計算機を複数の仮想的な計算機に見せかけ，この上でゲスト OS をそれぞれ走行させる．しかし，VM 方式はデバイスの仮想化によるオーバーヘッドが発生するほか，VM と VMM に依存性があり，実計算機と同等の性能では動作しない．

2 つ目は，実計算機に近い環境で複数の OS を動作させる方式であり，TwinOS や Mint が提案されている．各 OS で，CPU，メモリ，入出力機器といった計算機資源の効果的な共有や分割を行い，占有する．これにより，VM 方式に比べ，TwinOS や Mint 上で走行する各 OS は相互影響が少ない．この特徴を利用することで，ヘルスチェック機能をサービス用 OS から分離できる．

これらの方式のうち，TwinOS 上でヘルスチェック機能が実現されている．しかし，TwinOS はシングルコアプロセッサ上で実現されているため，TwinOS を用いたヘルスチェック機能はマルチコアプロセッサに対応していない．本研究では，TwinOS の設計方針を引き継ぎ，設計された Mint においてヘルスチェック機能の実現を行う．3 章で TwinOS と Mint について説明し，4 章の 4.2 節で TwinOS におけるヘルスチェック機能について説明する．

第 3 章

TwinOSとMintオペレーティングシステム

3.1 TwinOS

3.1.1 特徴と構成

1台の計算機上で2つのLinuxを走行させる方式としてTwinOSが提案されている．TwinOSは以下の設計方針に基づき実現されている．

(1) 相互に他 OS の処理負荷の影響を受けない．

(2) 両 OS とも入出力性能を十分に利用できる．

このため，1 台の計算機ハードウェアにおいて，CPU，メモリ，および入出力機器といったハードウェア資源の効果的な共有と占有を行う．2 つの OS の独立性を保つために，共有するハードウェアを最小限とすることが有効である．このため，CPU 以外のハードウェアは分割し，それぞれを各 OS に占有させる．図 3.1 に TwinOS の構成を示し，各ハードウェアの分割と共有方法を以下で述べる．

CPU

CPU については，シングルコアプロセッサを対象とし，時分割することによって共有する．このため，タイマ割り込みを契機として走行 OS を切り替える．

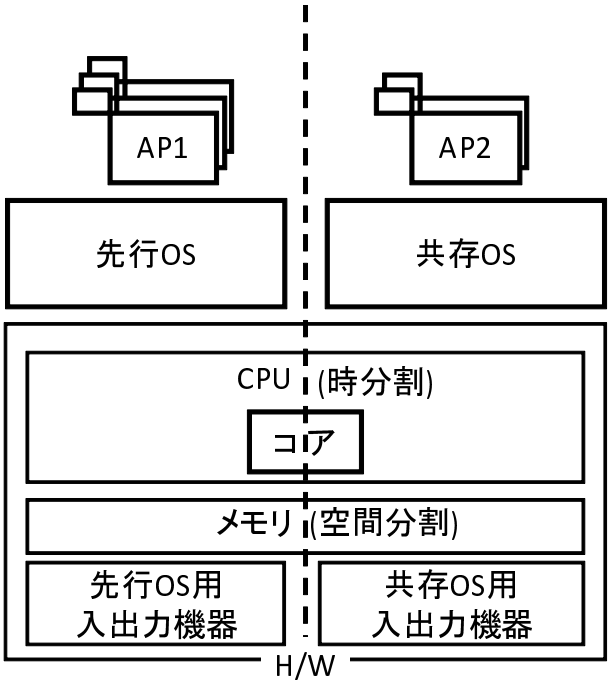

図　3.1　TwinOS の構成

メモリ

メモリについては，若番と老番に 2 分割する．先に起動する OS(以降，先行 OS と呼ぶ) にメモリの老番アドレス空間を，後から起動する OS(以降，共存 OS と呼ぶ) に若番アドレス空間を割り当てる．

入出力機器

入出力機器については，OS 毎に指定された入出力機器のみを占有制御する．現在走行していない OS への割り込みが発生した場合，OS を切り替えた後に割り込み処理を行う．

3.2　Mint オペレーティングシステム

3.2.1　特徴と構成

Mint は，TwinOS の設計方針を引き継ぎ，マルチコアプロセッサを搭載した 1 台の計算機上で複数の Linux を同時に走行させる．Mint は以下の設計方針に基づき実現されている．

(1) 全ての OS が相互に処理負荷の影響を与えない.

(2) 全ての OS が入出力性能を十分に利用できる.

このため，1 台の計算機上で CPU，メモリ，および入出力機器といったハードウェア資源を効果的に分割し，占有する必要がある．図 3.2 に Mint の構成を示し，各ハードウェアの分割と占有方法を以下で述べる.

CPU

CPU については，マルチコアプロセッサを対象とし，コア毎に分割する．そして，1 つ以上のコアを各 OS で占有する.

メモリ

メモリについては，走行させる OS の数だけ実メモリを空間分割し，各 OS で占有する.

入出力機器

入出力機器については，デバイス単位で分割し，各 OS は指定された入出力機器のみ占有する.

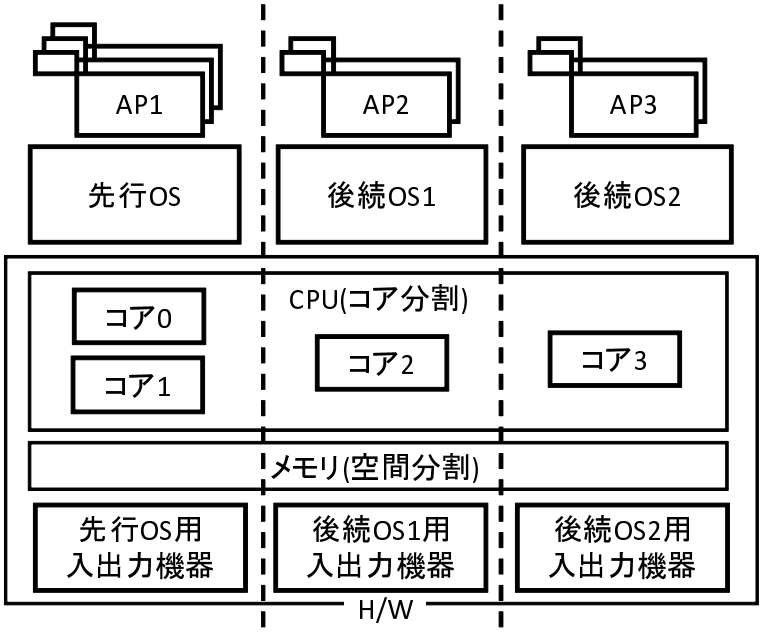

図 3.2 Mint の構成

第 4 章

Mint オペレーティングシステムにおけるヘルスチェック機能の実現

4.1 目的

Mint は，走行する全ての OS が相互に処理負荷の影響を与えないという特徴がある．これは，TwinOS と同様な特徴である．Mint 上でヘルスチェック機能を実現する際に，この特徴を利用することでヘルスチェック機能をサービス用 OS から分離することができる．

Mint は TwinOS をマルチコアプロセッサに対応させた方式であり，共通点が多い．そこで，TwinOS におけるヘルスチェック機能を参考に，Mint 上でヘルスチェック機能を実現する．Mint におけるヘルスチェック機能は，監視用 OS からサービス用 OS を監視する．サービス用 OS の監視により，攻撃者による OS への不正な改ざんの検出と被害拡大防止を目的とする．

また，Mint におけるヘルスチェック機能では，先行 OS を監視用 OS とし，後続 OS をサービス用 OS とする．検査範囲や検査契機は，TwinOS におけるヘルスチェック機能を参考にし，検討を行う．

4.2 TwinOS におけるヘルスチェック機能

TwinOS におけるヘルスチェック機能では，先行 OS を監視用 OS，共存 OS をサービス用 OS とし，監視用 OS にヘルスチェック機能を実装している．監視用 OS は監視専門に用いるため，ネットワークから切断した安全な環境で動作させる．サービス用 OS はサービス専門に用いるため，ネットワークと接続した環境で動作させる．

相互に他 OS の処理負荷の影響を受けないという TwinOS の特徴を利用し，TwinOS 上で動作する 2 つの OS を用いて，ヘルスチェック機能をサービス用 OS から分離することに成功している．また，TwinOS 上で動作する 2 つの OS は独立に走行するが，図 3.1 に示すようにメモリを共有している．このため，OS 間の検査データ授受において，OS 間通信を必要としない構成が可能である．したがって，監視処理を外部から隠ぺいできるため，先行 OS から共存 OS を監視することで信頼性の高いシステムの構築が可能になる．

ヘルスチェック機能は先行 OS 上のユーザ空間上で実現されている．管理者はヘルスチェック機能を呼び出し，共存 OS の検査を行う．検査は，正常データと検査データの 2 つのデータを扱う．正常データとは，サービス用 OS がインストールされた直後に，サービス用 OS のメモリから取得したデータである．検査前に，あらかじめ一度だけ作成される．また，検査データとは，走行中のサービス用 OS のメモリから取得したデータである．サービス用 OS が正常な場合，このデータは正常データと一致する．

正常データと検査データは同じ領域 (アドレス，サイズ) からデータを取得する．検査範囲，すなわち，どの領域を検査対象とすべきかについては 4.2.1 項で説明する．ヘルスチェック機能はこれら 2 つのデータの比較により検査を行う．サービス用 OS に異常がなければ，これら 2 つのデータは同じ値を示す．

また，検査にはメモリへアクセスする必要がある．しかし，ユーザ空間からカーネル空間のメモリへアクセスすることはできない．このため，正常データや検査データの作成等にはシステムコールを用い，OS の支援によって行う．

検査が終了すると，ヘルスチェック機能は検査の結果を管理者に通知する．共存 OS が，外部からの攻撃により改ざんされていた場合は，共存 OS を停止させる．

4.2.1 検査範囲

TwinOS におけるヘルスチェック機能は，メモリ上の以下の 3 つの領域を対象に検査を行う．また，以下の検査対象領域から算出したハッシュ値を正常データや検査データとして扱う．ハッシュ値の算出に使用するハッシュ関数には MD5 を用いる．

(1) カーネルテキスト部

通常，プログラムはテキスト部に配置され実行される．そこで，攻撃者は悪意のあるコードを実行するため，テキスト部への悪意あるコードの挿入，データ部に配置した悪意あるコードのアドレスへジャンプ先アドレスを変更するといった方法をとる．このため，物理メモリに展開されたカーネルテキスト部を検査範囲とする．

(2) システムコールテーブル
システムコールテーブルは，システムコールのジャンプ先アドレスを示すテーブルである．カーネルテキスト部と同じ理由で，ジャンプ先アドレスの変更のために用いられる．このため，物理メモリに展開されたシステムコールテーブルを検査範囲とする．

(3) LKM(Loadable Kernel Module) のテキスト部
不正な LKM をカーネルに組み込むと，カーネルテキスト部が変更されないまま OS が改ざんされる．LKM をロードすると，LKM が物理メモリに展開される．この展開された LKM のデータには，ヘッダとテキスト部とデータ部が含まれる．データ部はロードされる度に内容が変わる．このため，物理メモリに展開された LKM のテキスト部を検査範囲とする．

4.2.2 実装されている機能

TwinOS におけるヘルスチェック機能が提供する機能と実行契機を表 4.1 に示す．各機能について以下で説明する．

(1) 正常データ作成機能
正常データの作成を行う．共存 OS をネットワークから切断した安全な状態でこの機能を用いてハッシュ値を算出する．これを正常データとする．

(2) 正常起動検査機能
共存 OS の起動中の検査を行う．サービス用 OS のカーネルメモリ空間から検査データを取得する．あらかじめ取得していた正常データと新しく取得した検査データの比較を行い，共存 OS が正常か否か判定する．

(3) 正常走行検査機能
共存 OS 起動後に任意の契機で検査を行う．検査の処理は正常起動検査機能と同じである．

(4) 共存 OS 停止機能
停止処理を呼び出し，共存 OS の走行を一時停止する．正常データ作成や検査を行う際に使用され，検査中の走行 OS を停止し，変化しないようにする．また，共存 OS が異常であると判断された場合にも使用され，共存 OS の走行を停止する．

表 4.1 ヘルスチェック機能に実装されている機能と実行契機一覧

機能名	実行契機
(1) 正常データ作成機能	共存 OS の初回起動時
(2) 正常起動検査機能	正常データ作成後，共存 OS の起動時
(3) 正常走行検査機能	正常データ作成後，共存 OS の走行時
(4) 共存 OS 停止機能	検査時と検査で異常だった場合
(5) 共存 OS 再開機能	検査時と検査で正常だった場合
(6) 正常 LKM 登録機能	(1) の実行後
(7) 正常 LKM 検査機能	LKM のロード時

(5) 共存 OS 再開機能
再開処理を呼び出し，共存 OS 停止機能で停止した共存 OS の走行を再開する．

(6) 正常 LKM 登録機能
共存 OS 側で LKM をロードすると共存 OS のページテーブルが変更される．このページテーブルの変更点から LKM 展開先アドレスを算出して，物理メモリにあるページのハッシュ値を算出する．このハッシュ値を正常データとして登録する．

(7) 正常 LKM 検査機能
LKM のロード時にハッシュ値を取得し，正常データと比較する．

4.3 Mint オペレーティングシステムにおけるヘルスチェック機能

Mint におけるヘルスチェック機能は，TwinOS におけるヘルスチェック機能と同様の機能を提供する．ただし，Mint 上では複数のサービス用 OS が同時に走行するため，共存 OS 停止・再開機能はサービス用 OS 停止・再開機能とする．各機能の説明については 4.2.2 項と同様である．Mint におけるヘルスチェック機能の各機能は，ユーザ空間上で実現する．ユーザ空間上で実現不可能な処理はシステムコールとして OS 上に実装する．

サービス用 OS がネットワークに繋がっていない安全な状態で，サービス用 OS を起動し，正常データ作成を行う．それ以降は，サービス用 OS の起動時やサービス用 OS 走行中の任

意の契機で検査処理を行い，検査を行う．

Mint におけるヘルスチェック機能の検査の流れを図 4.1 に示し，以下で説明する．

(1) ヘルスチェック機能の実行

管理者は，監視用 OS 上でヘルスチェック機能を呼び出す．正常起動検査機能，正常走行検査機能，正常 LKM 検査機能によって，カーネルや LKM の検査を行う．

(2) 検査処理の呼び出し

ヘルスチェック機能は，ユーザ空間からカーネル空間のメモリへアクセスするため，システムコールを発行し，検査処理を呼び出す．

(3) 正常データの取得

正常データ作成機能と正常 LKM 登録機能によって，あらかじめ作成した正常データを監視用 OS が占有する HDD から取得する．

(4) メモリの検査

サービス用 OS が占有するメモリの検査を行う．

(5) 検査データの取得

検査データを取得する．

(6) 検査処理の結果返却

正常データと検査データの比較を行い，その結果を返却する．

(7) サービス用 OS の検査結果の返却

管理者にサービス用 OS が正常か否かの結果を返却する．

検査により，OS が異常だった場合は，サービス用 OS 停止機能によってサービス用 OS を停止する．

4.4 TwinOS と Mint オペレーティングシステムの違い

TwinOS と Mint は，同じ設計方針に基づき，TwinOS はシングルコアプロセッサ上で，Mint はマルチコアプロセッサ上で実現されている．また，TwinOS と Mint ではプロセッサの利用方法が異なる．以下に，Mint におけるヘルスチェック機能の実現において，問題となる TwinOS と Mint の違いについて示す．

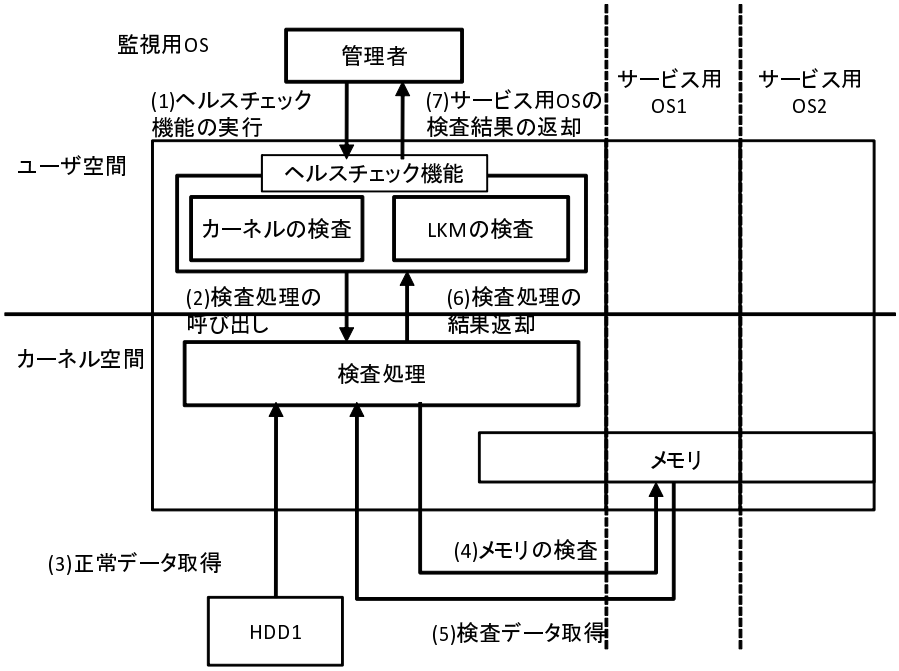

図　4.1　Mint におけるヘルスチェック機能の検査の流れ

コアの分割方法

コアの分割方法が異なる．TwinOS はシングルコアプロセッサを対象とし，1 つのコアを時分割し，2 つの OS で共有している．一方，Mint はマルチコアプロセッサを対象とし，コア毎に分割し，複数の OS でそれぞれ 1 つ以上のコアを占有している．

OS の走行方式

OS の走行方式が異なる．TwinOS は 1 つのコア上で，タイマ割り込みを契機に，走行する OS を切り替える．一方，Mint は，複数の OS が自身の占有するコア上で同時に走行する．

4.5　課題

4.4 節で示した TwinOS と Mint の違いから，以下の課題が生じる．

(課題 1) 監視用 OS によるサービス用 OS の停止・再開

サービス用 OS を乗っ取られた場合，自計算機だけでなく，ネットワークを介して周囲の計算機にまで被害が拡大する恐れがある．このため，サービス用 OS に異常が検出

された場合，速やかにサービス用 OS を停止させたいという要求がある．この要求に対して，TwinOS におけるヘルスチェック機能では，共存 OS 停止機能を用意することで対処している．Mint におけるヘルスチェック機能でも，異常が検出されたサービス用 OS を停止させることで被害の拡大防止を行う．TwinOS と Mint のコアの分割方法と OS の走行方式の違いから OS の停止・再開方法が異なる．TwinOS は 1 つのシングルコアプロセッサを 2 つの OS が共有し，タイマ割り込みを契機に OS を切り替える．このため先行 OS から共存 OS への切り替えを停止することで，共存 OS の停止を行う．一方，Mint は OS の切り替えは行わず，マルチコアプロセッサの各コア上でそれぞれの OS が同時に走行する．このため，TwinOS におけるヘルスチェック機能と同様の方法では OS を停止できない．また，検査中に OS が変化することを防ぐため，検査中は OS を停止させる．このため，OS の停止機能は再開を考慮する必要がある．

(課題 2) 検査範囲や検査契機の検討

TwinOS と Mint は実現されている OS の走行方式が異なる．このため，TwinOS におけるヘルスチェック機能と同様の検査範囲や検査契機で サービス用 OS の検査が十分に行えるか検討する必要がある．

(課題 3) 攻撃者によるサービス用 OS 停止の妨害

Mint 上では，マルチコアプロセッサの各コア上でそれぞれの OS が独立に，そして同時に走行する．このため，OS の停止を行う場合，監視用 OS がサービス用 OS に停止処理を行うように信号を送る必要がある．しかし，この信号を妨害されてしまうと OS を停止することができない．ヘルスチェック機能によって OS が異常であると判断した場合は，被害拡大防止のために確実に OS を停止する必要がある．

また，TwinOS にもともと存在する以下の課題も挙げられる．

(課題 4) HDD を 1 台しか持たない計算機上ではヘルスチェック機能を利用できない

Mint 上で複数の OS を走行させるためには，走行させる OS の数だけ HDD が必要となる．Mint 上でヘルスチェック機能を利用するためには 2 台以上の HDD が必要である．このため，HDD を 1 台しか搭載していない計算機ではヘルスチェック機能を利用することができない．

4.6 対処

4.5 節で示した (課題 1) から (課題 4) の中で，(課題 1) の監視用 OS によるサービス用 OS の停止・再開と (課題 4) の HDD を 1 台しか持たない計算機上ではヘルスチェック機能を利用できないという 2 つの課題に対処する．

前者は，IPI の送信により，サービス用 OS の停止・再開を行うことで対処する．複数コア間の通信を実現する仕組みとして，IPI(Inter-Processor Interrupt) が APIC(Advanced Programmable Interrupt Controller) により提供されている．IPI は任意のコアへ割り込みを発生させる．監視用 OS が占有するコアからサービス用 OS が占有するコアへ IPI を送信し，コアを停止，あるいは再開させる割り込みを発生させる．これによって，サービス用 OS を停止，あるいは再開させる．

後者は，initrd の拡張により，先行 OS を HDD の占有なしで起動することで対処する．これによって，HDD を 1 台しか搭載しない計算機でもヘルスチェック機能を利用することができる．先行 OS を HDD の占有なしで起動するため，先行 OS はルートファイルシステムをマウントせず，initrd(initial ramdisk) だけで動作させる．initrd は実際のルートファイルシステムが使用できるようになる前に，マウントされる初期ルートファイルシステムのことであり，必要最小限のファイルやコマンドを備えている．initrd を拡張することで先行 OS を HDD の占有なしで起動する．

第 5 章

実現

5.1 サービス用 OS 停止・再開機能

5.1.1 サービス用 OS 停止・再開機能における課題とその対処

Linux にはコアを停止させる IPI を呼び出す関数が用意されている．この IPI を受信した OS は自身のコアを停止させる関数を呼び出し，コアを停止する．監視用 OS からこの IPI をサービス用 OS へ送信することで，サービス用 OS が占有するコアを停止させ，サービス用 OS を停止させる．また，Mint 上では各 OS が同時に走行する．サービス用 OS 検査中に OS に変化があった場合，正しく検査を行うことができない．このため，サービス用 OS 停止機能は，異常なサービス用 OS を停止させる場合だけでなく検査の際にも使用する．したがって，サービス用 OS の再開を考慮して停止を行う必要がある．

コアを停止させる IPI を受信したコアは以下のような処理を行い，コアを停止する．

(1) 自身のローカル APIC を無効化する．

(2) hlt 命令を無限ループで実行させる．hlt 命令とは，割り込みを受け取るまでコアを一時停止させる命令である．

この IPI を用いて走行中の後続 OS を停止させる機能を先行 OS のカーネル上にシステムコールとして実装した．しかし，従来の関数を使った場合，コアの再開を考慮しないため，以下のような課題が生じる．

(課題 1) ローカル APIC の無効化による割り込みの禁止
ローカル APIC を無効化し割り込みを禁止するため，外部から IPI を送信して停止したサービス用 OS を再開させることができない．

(課題 2) 無限ループで実行されている hlt 命令によるコアの停止
hlt 命令を無限ループで実行させてコアを停止させる場合，IPI を送信してもコアが再開しない．

これらの課題への対処を以下に示す．

(対処 1) NMI モードでの IPI の送信
ローカル APIC の無効化はソフトウェアで行っている．このため，ソフトウェアからマスク不可な割り込みである NMI(Non-Maskable Interrupt) モードで IPI を送信する．

(対処 2) hlt 命令の実行回数の変更
コアの停止は hlt 命令を無限ループで実行させず，一度だけ呼び出しコアを停止するように変更する．

これらの対処を行い，サービス用 OS 停止機能とサービス用 OS 再開機能を実装した．

5.1.2 サービス用 OS 停止機能

図 5.1 にサービス用 OS 停止機能の処理流れを示し，以下で説明する．コアを停止させる IPI を監視用 OS が占有するコアからサービス用 OS が占有するコアへ送信すると，IPI を受信した側のコアは OS の停止処理を行う．IPI の送信は，stop_os システムコールにより行う．監視用 OS は IPI を送信後，処理を終了する．OS の停止処理は IPI を受信したコアのローカル APIC を無効化した後，hlt 命令を一度だけ実行し，コアを停止する．コアの停止により，サービス用 OS が停止する．また，ローカル APIC を無効化することで割り込みを禁止し，割り込み発生によるサービス用 OS の勝手な再開を防ぐ．

今回実装したサービス用 OS 停止機能は，監視用 OS が IPI を送信し，サービス用 OS に割り込みを発生させ停止する．このため，攻撃者によって IPI による割り込みの発生を無効化された場合，サービス用 OS を停止することができない．4.5 節の (課題 3) で述べたが，確実にサービス用 OS を停止する機能も必要である．

5.1.3 サービス用 OS 再開機能

図 5.2 にサービス用 OS 再開機能の処理流れを示し，以下で説明する．サービス用 OS 停止機能によって停止しているコアを再開させ，サービス用 OS を再開させる．停止している

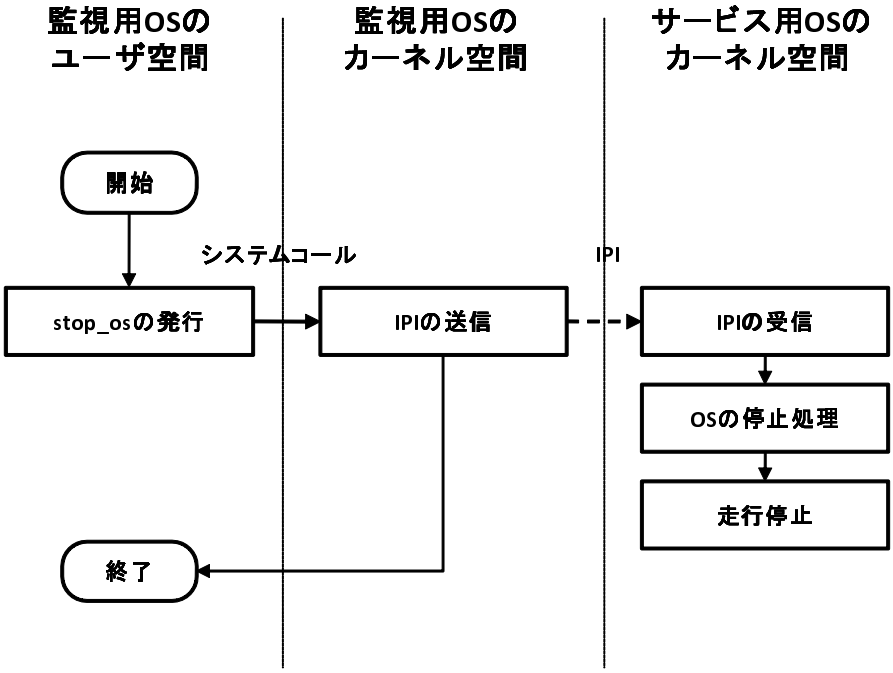

図 5.1 サービス用 OS 停止機能の処理流れ

OS は hlt 命令によって停止している．このため，サービス用 OS に割り込みを発生させることでサービス用 OS は再開する．しかし，割り込みが禁止された状態で停止しているため，ソフトウェアからマスク可能な割り込みである IPI では割り込みを発生させることができない．このため，NMI モードで IPI を送信し，ソフトウェアからマスク不可能な割り込みを発生させてサービス用 OS を起動させる．IPI の送信は，restart_os システムコールにより行う．監視用 OS が占有するコアからサービス用 OS が占有する停止しているコアへ NMI モードで IPI を送信する．監視用 OS は IPI を送信後，処理を終了する．サービス用 OS が占有するコアが NMI モードの IPI を受信することで，ソフトウェアからマスク不可能な割り込みが発生する．サービス用 OS が占有するコアは hlt 命令によって停止している．このため，サービス用 OS が占有するコアは再開する．

5.2 HDD を占有しない先行 OS によるヘルスチェック機能の支援

図 5.3 に先行 OS が HDD を占有しない場合の Mint の構成を示す．カーネルは起動時に initrd をマウントし，initrd の中のプログラムやモジュールを利用して HDD 上の本来のファ

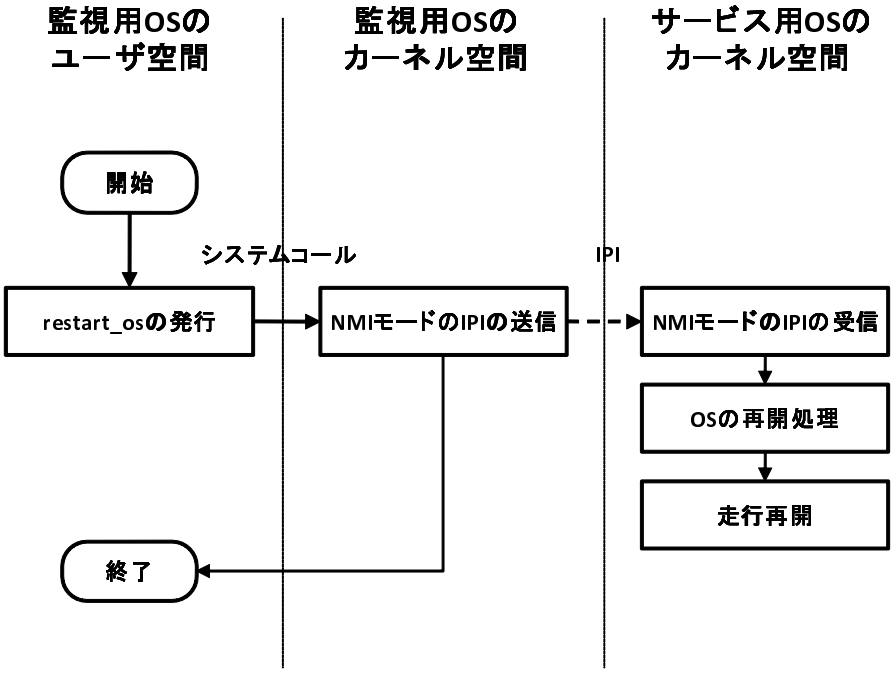

図 5.2 サービス用 OS 再開機能の処理流れ

イルシステムをマウントする．HDD を占有しない先行 OS では初期ルートファイルシステムである initrd をファイルシステムとして利用する．このため，HDD を占有せずに動作することができる．initrd は，必要最小限のファイルやコマンドを備えている．initrd 内に後続 OS の起動やヘルスチェック機能に必要なファイル等を用意し，initrd を拡張することで HDD を占有しない先行 OS からでも後続 OS の起動やヘルスチェック機能を実行することができる．

先行 OS が HDD の占有を行わずに起動し，initrd をファイルシステムとして利用するために以下の 3 つを行い，initrd の拡張を行った．

(1) initrd 内で作業が行えるように変更
initrd の環境内で作業が行えるように，シェルが利用できるように変更した．また，HDD にあるファイルシステムをマウントしないように変更した．

(2) 後続 OS の起動用スクリプトを追加
後続 OS の起動に必要なファイル等を initrd 内に用意する．具体的には，後続 OS の initrd イメージ，bzImage，先行 OS から後続 OS を起動するためのシェルスクリプト，実行ファイルを initrd 内に用意する．

(3) ヘルスチェック機能の実行ファイルの追加

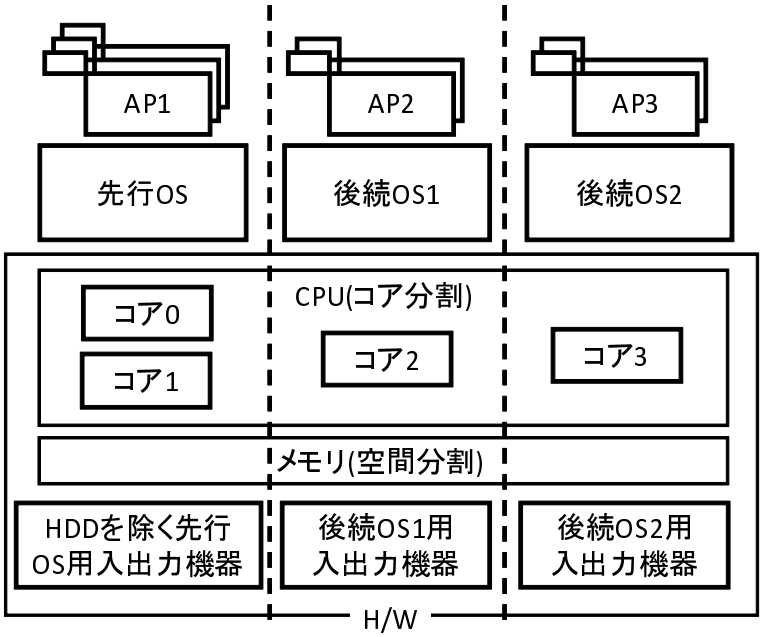

図 5.3 先行 OS が HDD を占有しない場合の Mint の構成

ヘルスチェック機能に必要なファイルを initrd 内に用意する．具体的にはヘルスチェック機能の実行ファイルを用意する．

これらによって，HDD の占有を行わずに先行 OS を起動することができる．また，HDD を占有しない先行 OS を監視用 OS とすることで，HDD を 1 台しか持たない計算機でもヘルスチェック機能を利用することができる．

第 6 章

おわりに

本論文では，Mint におけるヘルスチェック機能の実現のための課題と対処を明確にし，それを基にヘルスチェック機能の実装について述べた．

まず，TwinOS と Mint の違いを明確にした．具体的には，CPU の分割方法の違い，OS の走行方式の違いがあることを示した．

次に Mint におけるヘルスチェック機能の実現のための課題を明確にした．具体的には，課題として監視用 OS によるサービス用 OS の停止・再開，検査範囲と検査契機の検討，攻撃者によるサービス用 OS 停止の妨害について述べた．また，TwinOS にもともと存在する課題として，HDD を 1 台しか持たない計算機上ではヘルスチェック機能を利用できないことについて述べた．

そして，監視用 OS によるサービス用 OS の停止・再開，HDD を 1 台しか持たない計算機上ではヘルスチェック機能を利用できないという 2 つの課題について対処を行い，実装した．前者については，IPI を用いたコアの停止・再開による OS の停止・再開を行う方法について述べ，システムコールを実装した．後者については，initrd の拡張によって先行 OS を HDD の占有なしで起動し，この先行 OS を監視用 OS とすることで解決した．

残された課題として，検査範囲と検査契機の検討，攻撃者によるサービス用 OS 停止の妨害がある．また，正常データ作成機能，正常走行検査機能，正常 LKM 登録機能，および正常 LKM 検査機能の実装がある．

謝辞

本研究を進めるにあたり，懇切丁寧なご指導をして頂きました乃村能成准教授に心より感謝の意を表します．また，数々のご指導やご助言を頂きました谷口秀夫教授，山内利宏准教授，後藤佑介助教授に厚く御礼申し上げます．最後に，日頃の研究活動において，お世話になりました研究室の皆様ならびに本研究を行うにあたり，経済的，精神的な支えとなった家族に感謝いたします．

参考文献

[1] Paul Barham，Boris Dragovic，Keir Fraser，Steven Hand，Tim Harris，Alex Ho，Rolf Neugebauer，Ian Pratt，and Andrew Warfield,“ Xen and the Art of Virtualization，” Proc. of the 19th ACM Symposium on Operating Systems Principles，pp.164-177，2003.

[2] Jeremy Sugerman，Ganesh Venkitachalam，and Beng-Hong Lim,“ Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor，”Proc. of the General Track：2002 USENIX Annual Technical Conference，pp.1-14，2001.

[3] 田渕正樹，伊藤健一，乃村能成，谷口秀夫,“ 二つの Linux を共存走行させる機能の設計と評価，” 電子情報通信学会論文誌 (D-I)，Vol.J88-D-I，No.2，pp.251-262，2005.

[4] 千崎良太，中原大貴，牛尾裕，片岡哲也，粟田祐一，乃村能成，谷口秀夫,“ マルチコアにおいて複数の Linux カーネルを走行させる Mint オペレーティングシステムの設計と評価，”電子情報通信学会技術研究報告 vol.110，no.278，pp.29-34，2010.

[5] 池内達也,“ ヘルスチェック機能を支援する TwinOS の実現，” 岡山大学工学部情報工学科修士論文，2009.

[6] 長田一帆,“ TwinOS におけるヘルスチェック機能の実現と評価，” 岡山大学工学部情報工学科卒業論文，2010.