情報処理学会論文誌 Vol. 50 No. 3 1043–1052 (Mar. 2009)

認証デバイスを用いたOSの起動・終了制御

高田真吾[†1] 佐藤聡[†1] 新城靖[†1]
中井央[†2] 板野肯三[†1]

企業や大学など多数のコンピュータが導入されている組織では，利用者に対する計算機利用の自由度を高く維持しつつ，稼動しているOSの種類とその最大稼動数を適切に管理する必要がある．本研究では認証デバイスを用いて，管理者により承認された複数のOSから適切なOSの起動と終了を制御できるシステムを提案する．提案システムの特徴は，OSの種類とは独立に，起動されるOSの数を把握するために認証デバイスを用いる点にある．利用者は，システム利用時に利用したいハードウェアに認証デバイスを挿入する．提案システムは，デバイスの挿入を検出するとハードウェアと認証デバイスの組合せにより，管理者が定めた起動すべきOSを決定し，起動する．認証デバイスが抜かれると，起動したOSを停止する．これらの動作は，制御対象のOSより下位に配置するOS制御レイヤにおいて行う．このOS制御レイヤを仮想計算機モニタXenを用いて実装し，動作実験を行い，提案システムの有効性を確認した．

OS Boot-shutdown Control System Using Authentication Devices

Shingo Takada,[†1] Akira Sato,[†1] Yasushi Shinjo,[†1]
Hisashi Nakai[†2] and Kozo Itano[†1]

In large enterprises or university campuses, it is necessary both to support customizable flexibility of computer environments for users and to manage the kind and the maximum number which OS (Operating System) is running on these environments. So we propose a Boot and Shutdown Control System which allows a user to choose and boot a suitable OS from certified OSes using an authentication device. One of characteristics of our proposed system is to use the authentication device in order to manage the maximum number of OS instances with an OS independent manner. Before using our system, a user inserts an authentication device into the hardware that he or she wants to use. Our system selects an OS using the combination of certificates stored in the authentication devices and hardware, and boots the selected OS. When the authentication device is removed, our system halts its OS. These processes are operated by OS Control Layer which runs under the OS. In order to examine the advantage of this system, we implement OS Control Layer by using the Virtual Machine Monitor Xen. Using this way, we don't have to modify our hardware architecture to realize the boot controlling. We show the experimental result of boot-controlling and confirm the effectiveness of our proposed system.

1. はじめに

企業や大学など多数のコンピュータが導入されている組織では，コンピュータを適切に管理することは非常に重要な課題である．多数のコンピュータの管理方法は管理主体の違いにより大きく2種類に分類される．

1つは，利用者が主体となって分散的に管理する方法である．この方法では，利用者が自由にOSや主要となるアプリケーションを選択可能であるため，計算機環境構築の自由度は非常に高い．しかし，コンピュータの管理を行う利用者の数が非常に多くなると，インストールされているソフトウェアを適切に管理できなくなったり，セキュリティ対策が万全とならなくなったりし，その結果組織全体としてのリスクが高くなる．

もう1つは，管理者が集中的に管理する方法である．これは，管理者がOSや主要となるアプリケーションを統一し，セキュリティ対策やインストールされているソフトウェアに関する管理を集中的に行う方法である．この方法では，管理者が一元的にソフトウェアの管理を行うため管理コストを削減することができる．しかし利用者が利用できる計算機について制限を受けるなど，利用者の利便性が損なわれる．

前者の欠点，利点は，後者のそれと相反してあてはまっている．大学や企業などにおいては利用者の所属する学部や部署の教育・業務目的に適した計算機環境がある．組織全体としては複数の環境を提供するが，利用者や利用場所ごとに利用する計算機環境を使い分けたり，ある特定の環境を，特定の人数だけに限定的に利用させたりしたいという要望もある．大学において，ネットワークやオペレーティングシステムの講義を履修している学生に対してのみそれらの理解を助けるようなOS環境を利用できるようにしたいという要望がある．

†1 筑波大学大学院システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba
†2 筑波大学大学院図書館情報メディア研究科
Graduate School of Library, Information and Media Studies, University of Tsukuba

企業においても，ネットワーク管理者のみがネットワークのトラブルを解決を行いやすい OS を利用したいという要望がある．そのほか，大学などの公開講座の受講者や，企業における来訪者に対してのみ，特定の部屋に設置してある計算機を用いて，利用できる数を受講者・来訪者数に応じた数に制限したうえで特定の OS を利用させたいといった要望もある．したがって，我々は，多数のコンピュータを導入する組織においては前述の両者の利点をあわせ持つような管理を実現するために，以下に示す条件をすべて満たす管理支援の仕組みが今後必要になると考えている．

- 利用者が，望むときに，望む場所に設置されている計算機上で，要求する OS を稼動させて利用することができる．
- 利用者の利便性を損なわないように，数多くの種類の OS が稼動できる．
- 管理者が，どの種類の OS をどの計算機上で稼動可能かを適切に管理できる．
- 管理者が，OS の種類ごとの最大稼動台数を適切に管理できる．
- 利用者が利用する OS 自体に変更を加える必要がない．

現在までに，上記の条件のうちのいくつかを支援する仕組みが提案されている．特定の用途に特化した OS イメージを CD や DVD などの外部メディアに格納しておき，そのメディアから OS を起動するという LiveCD [4] のような仕組みがある．この仕組みは，利用者の利便性を損なわずに複数種類の OS を稼動可能な計算機環境を提供できているが，組織全体で利用されている OS の種類を把握することができない．また，PXE [7] は，利用者の要求にあわせて作成した複数種類の OS のイメージをネットワーク上のディスクサーバにあらかじめ作成しておき，利用者にその選択を委ねる仕組みである．この仕組みでは，起動ディスクイメージの配信数をカウントすることにより実行されている OS のインスタンス数を把握できるが，利用者がシャットダウン処理を行わずに席を立ってしまうとその分のカウントダウンを行うことができず，最大稼動台数を満たすように管理できなくなってしまうという問題がある．このように，現在までに前述のすべての条件を満たす運用支援の仕組みは提案されていない．

そこで我々は上記の条件をすべて満たすネットワークブートのシステムを提案する．

提案システムの特徴は，起動される OS の数を把握するために認証デバイスを用いる点にある．提案システムでは，利用者がシステムを利用するときに，利用したいハードウェアに認証デバイスを挿入する．提案システムは，ハードウェアと認証デバイスの双方を認証し，起動すべき OS を決定し，起動する．認証デバイスが抜かれると，起動した OS を停止する．すなわち，提案システムでは，認証サーバに登録されている認証デバイスを用いて同時刻に起動可能な OS をたかだか 1 つとすることができる．提案システムでは，認証デバイスとハードウェアの認証については公開鍵認証を用いる．それぞれの公開鍵情報は管理者があらかじめシステムに登録する．また，利用可能な OS のイメージについても管理者があらかじめシステムに登録する．そして，ハードウェア，認証デバイス，起動する OS の組合せを管理者がシステムに登録する．

このシステムを実現するには，起動・終了制御対象の OS より下位のレイヤに，起動後も継続的に動作する OS 制御レイヤと呼ばれる制御レイヤを挿入し，そのレイヤにおいて起動・終了の制御や認証デバイスの操作などを行う．認証情報や OS イメージはネットワークを用いて転送する．このとき，盗聴や改竄といった脅威から保護するために，その通信経路は SSL を用いて暗号化する．なお，本論文では，利用者に認証デバイスをどのように配布するかといった認証デバイス自体の管理方法については議論の対象としない．

この仕組みの有効性を確かめるために，本研究では起動・終了制御を仮想計算機モニタ Xen [3] を用いて実現した．また認証デバイスとして USB トークンを用いた．ハードウェアを個々に識別するためのデバイスとして，TPM [2] のように複製がきわめて困難なハードウェアモジュールを利用するべきであるが，プロトタイプの作成，検証を行ううえで TPM の利用が困難であったため，同等の処理をソフトウェアで行うことができるようファイル形式の証明書を用いた．

2 章では，提案手法に関連した研究を取り上げ，提案手法の優位性について述べる．3 章では提案手法について述べる．4 章では，提案手法の実装について述べ，5 章では実装したシステムの評価について述べ，その考察を行う．最後に 6 章で本論文をまとめる．

2. 関 連 研 究

2.1 PXE—Preboot eXecution Environment

クライアントコンピュータには必要最小限のプログラムだけを配置し，起動に必要なファイルをネットワーク経由で取得することにより，ディスクレスブートを実現するための規格として，Intel 社が策定した PXE がある．PXE では，NBP（Network Boot Program）を tftp を用いて取得し，この NBP が OS を起動する．利用者に対して複数の起動候補を提示，選択させることが可能であるほか，MAC アドレスに応じて起動する OS を変更することも可能である．

PXE のような手法では，提供する環境のバリエーションの数だけ利用者に選択肢を提供するため，利用者はその中から適切な環境を選択する必要がある．これは煩雑な手間であ

り，また利用者が誤って終了方法を知らない OS を起動してしまうといった操作ミスが起こりうる．また，PXE はブートローダのように OS の起動のみを制御するため，OS の起動後に何か処理を行うことができない．

提案システムでは，認証デバイスとハードウェアの組合せから起動する OS を決定するため，利用者が利用できない OS が起動することは起こらない．また，提案システムでは OS の起動後も OS から独立して継続的に動作するため，OS の終了まで制御を行うことが可能である．

2.2　起動処理におけるコンポーネント検証による認証

電源投入直後から OS の起動までの間に処理が行われるコンポーネントの状態値を検証することにより，不正なコンポーネントの挿入や改竄といった脅威からシステムを保護し，起動処理の正当性を検証可能とする試みがある．

AEGIS[1] は，拡張ボードに状態値を格納しておき，起動時にコンポーネントの状態値を比較することでシステムの正当性を検証する．sAEGIS[9] はこの AEGIS をさらに拡張したもので，ブートローダとして grub をサポートしたことにより起動できる OS の種類が増え，また状態値の保存にスマートカードを採用したことにより，システムの正当性のより高度な検証を実現している．

Intel TXT[8] は，システムの正当性の検証を CPU や TPM を用いて実現する実装である．

CD や DVD などのメディアに起動に必要なファイルだけを格納しておき，それ以外に必要となるファイルをネットワーク経由で取得することで OS を起動するシステムとして，HTTP-FUSE KNOPPIX[12] がある．中村ら[11] は，この FUSE を TPM を搭載したコンピュータ上で利用する際に，システムの正当性を記録する手法を提案，実装している．この手法では，同様の方法で TPM にコンポーネントの状態値を記録することで，システムの正当性を検証することを可能としている．また，ファイルを取得する際に利用するサーバについては SSL サーバ認証により確認を行い，取得したファイルについても，ハッシュ値を用いて検証を行っている．

これらの研究は，起動に必要なコンポーネントが想定した状態で存在していることを検証することを可能としている．この仕組みを利用して，起動時に認証デバイスを必要とするシステムを構築することは可能であるが，OS の起動後に認証デバイスが抜かれるという構成の変更を検出するためには起動する OS に監視を行うような変更を加える必要がある．

それに対し，提案システムは，OS の起動後も認証デバイスが挿入された状態が継続していることを監視する．これにより，特定の認証デバイスと特定のハードウェアの組合せでのみ許可された OS が稼動することを保証するだけでなく，起動する OS に変更を加えることなく，OS の起動後もその組合せに変更がないことを保証している．

2.3　ハードウェアの固有情報による実行制御

システムのハードウェアが持っている固有情報を用いることにより，特定のハードウェア上での特定のアプリケーションの動作を制御する仕組みを実現することができる．

XOM[10] は CPU が固有の鍵を持つことでこの仕組みを実現しているアーキテクチャである．この CPU 上で動作するプログラムをあらかじめ CPU の公開鍵で暗号化しておくことで，プログラムがその CPU 上でのみ動作するということが実現可能となる．これにより，OS が稼動する計算機（CPU）を制限する仕組みを実現することができる．

これに対し，提案システムでは，認証デバイスとハードウェア固有情報の組合せにより，OS の稼動を制御する．さらに，ハードウェアと認証デバイスの組合せが異なると，異なる OS が起動するように制御可能である．

XOM では，このアーキテクチャ用に OS やアプリケーションを変更する必要があるが，提案システムでは，アーキテクチャに変更の必要がないため，既存の OS に変更を加えることなく起動制御を行うことができる．

3. 提 案 方 式

本研究では，認証デバイスを用いて，あらかじめ登録されている OS の中から，認証デバイスとハードウェアの組合せをもとにして適切な OS を選択し，その OS の起動を制御するシステムを提案する．提案システムでは，登録された OS の種類ごとの最大稼動数を管理するために，認証デバイスを用いる．そして，次のような要求を満たすように設計と実装を行う．

(1)　OS の起動には，認証デバイスが必要となること
(2)　ハードウェアの管理者が，認証デバイスとその認証デバイスで起動できる OS を指定できること
(3)　ハードウェアと認証デバイスの組合せで，起動できる OS がたかだか 1 つに定まること[⋆1]
(4)　認証デバイスがハードウェアから抜かれた場合，起動中の OS が停止すること

⋆1 これは，利用者主体で管理を行う場合には一般に「誰が」と「どのハードウェアで」の要素が定まれば利用する計算機環境が一意に定まるという理由による．

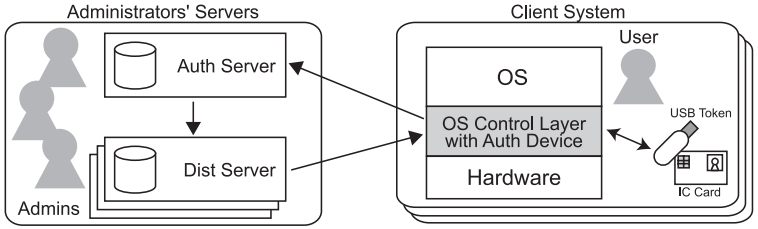

図 1　提案システム概要
Fig. 1　Overview of proposed system.

提案システムの概要図を**図 1** に示す．提案システムでは，認可された認証デバイスのことを単に**認証デバイス**と呼び，利用者が要求する OS を単に **OS** と呼ぶ．また，その OS が起動されるハードウェアを単に**ハードウェア**と呼ぶ．提案システムは認証デバイスとハードウェア，OS についての起動許可情報の管理を行う**認証サーバ**や，ディスクイメージの配布を行う**配布サーバ**と，利用者用の端末である**クライアントシステム**から構成される．提案システムでは，認証サーバはすべてのクライアントシステムで 1 つとすることができ，かつ配布サーバは複数台のクライアントシステムごとに 1 つとすることができる．これにより，提案システムの管理・運用上のコストを削減することが可能となる．

3.1　クライアントシステム

提案方式では，クライアントシステムにおいて，OS の起動・終了を制御するために制御対象の OS より下位のレイヤに OS 制御レイヤを配置する．従来の構成ではここには BIOS が位置しており，デバイスの管理や IPL（Initial Program Loader）の提供を行っている．本研究では，BIOS のかわりに OS 制御レイヤを配置し，起動・終了の制御や認証デバイスの抜き取り監視，認証処理を行う．これにより，この上で動作する OS に変更を加えることなく，提案方式を実現することが可能となる．

提案方式ではクライアントシステムの電源を投入しても認証デバイスが挿入されていない場合には OS は起動しない．OS 制御レイヤは認証デバイスが挿入されたことを検知すると，認証デバイスとハードウェア情報に応じて適切な OS のイメージを取得し，その OS を起動する．また，物理的認証デバイスが抜かれたことを検知すると，起動した OS の停止処理を行う．これにより，提案システムでは認証サーバに公開鍵情報が登録されている認証デバイスを用いて同時刻に起動可能な OS をたかだか 1 つとすることができる．

従来のコンピュータにおいて，ハードウェア上に搭載されている BIOS が OS の起動を行っている．したがって，提案方式では OS 制御レイヤが BIOS にかわって起動・終了制御を行うものであり，従来のコンピュータシステムとの親和性が高い．BIOS は起動時のみ動作するのに対し，本制御レイヤは起動後も動作する．これにより，認証デバイスの抜き取りを監視することができる．また，ハードウェアや OS に対する変更点が少ないという利点がある．下位のレイヤから OS に対し状態変化を伝える例として，ACPI[6] がある．ACPI では，主に電源状態の変化を OS に伝える手段を提供しており，同様の手法を用いることで必要に応じて起動中の OS を停止させることが可能である．

3.2　認証サーバと配布サーバ

認証デバイスやハードウェアの情報，OS のディスクイメージなどを一括管理するために，提案方式では認証サーバと配布サーバを設置する．認証サーバでは，次のような情報を管理，格納する．

(a)　ポリシを記述した ACL（Access Control List）
(b)　認証デバイスに対応する公開鍵
(c)　OS イメージ名と，その OS イメージを管理する配布サーバの URI

一方，配布サーバでは次のような情報を管理，格納する．

(d)　OS イメージ名と OS イメージファイルの対応リスト
(e)　OS イメージファイル

ハードウェアについては，SSL（Secure Socket Layer）クライアント認証により認証を行う．その正当性の検証は「クライアント証明書に署名を行った認証局が信頼できる認証局であるかどうか」について行われるため，配布サーバは認証局の証明書だけを持てばよい．

(a) では，利用を許可するハードウェア名，認証デバイス名，OS イメージ名の組合せを保持する．(b) では，認証デバイスに格納された証明書に対応する公開鍵情報を保持する．これにより，あらかじめ公開鍵情報を認証サーバに登録してある認証デバイスのみを提案システムにおいて利用が許可された認証デバイスとすることができる．(c) では，OS イメージ名からその OS イメージを担当する配布サーバの URI を取得するために，その担当関係を保持する．最後に，(d)，(e) は，クライアントに送信するディスクイメージの名前と実際のファイルとの対応を保持する．これにより，ディスクイメージファイルの管理が容易になる．

なお，本配布サーバでは，利用者に提供する OS 環境をディスクイメージとして管理する．これにより，種類の異なる OS を異なる OS 環境として利用者に提供することができるだけでなく，同一種類のオペレーティングシステムであっても，その上にインストールされているソフトウェアが異なれば，異なる OS 環境として利用者に提供することができる．

3.3　認証処理を含む OS イメージの転送方法

ネットワークにおいて，安全に認証やデータ転送を行うプロトコルとして，HTTPS（Hyper Text Transfer Protocol Security）がある．HTTPS では，SSL サーバ認証とクライアント認証の両方を同時に用いることにより，サーバはクライアントを，クライアントはサーバを互いに識別することができる．このとき，提案方式ではサーバ側の証明書としては認証サーバまたは配布サーバを識別できる証明書を用い，クライアント側の証明書としてはクライアントシステムが実行されているハードウェアに関連づけられたハードウェア用証明書を用いる．これにより，通信経路が暗号化されるほか，認証サーバ側でクライアントシステムが実行されているハードウェアについての認証を行うことができる．同様に，認証デバイスの証明書を用いて SSL クライアント認証を行えば，認証デバイスについても認証サーバ側で認証を行うことができるが，ハードウェアについて認証を行ったセッション上でそれを行うことはできない．そこで，本実装ではハードウェアについて認証を行ったセッション上で，HTTP を用いた独自プロトコルにより認証デバイスを用いた認証を行うこととした．

提案するプロトコルにおける認証とイメージ転送の流れを次に示す．ここで，認証デバイス名とは，認証デバイスに格納された証明書の CN（Common Name）を指す．

(1)　OS 制御レイヤは認証デバイスから認証デバイス名を取得する．

(2)　OS 制御レイヤは，認証サーバに対しハードウェア用証明書を用いた HTTPS SSL クライアント認証セッションを確立する．このセッションを用いて，以下の処理を行う．
- (a)　OS 制御レイヤは (1) で取得した認証デバイス名を名乗り，認証要求を出す．
- (b)　認証サーバはこのセッションを確立したハードウェア名（ハードウェア用証明書の CN）を取得する．
- (c)　認証サーバは認証デバイス名とハードウェア名から，起動すべき OS とその OS イメージを担当する配布サーバの URI を決定する．
- (d)　認証サーバはランダムに作成したデータをチャレンジデータとして，起動すべき OS 名とともに保存する．そのチャレンジデータとその OS イメージを担当する配布サーバの URI の組を HTTP 応答としてクライアントに送信する．起動すべき OS が指定されていない場合，そのハードウェアに対しその認証デバイスによる起動が許可されないことを表すため，HTTP エラー応答を送信する．

(3)　OS 制御レイヤは受け取ったチャレンジデータを，認証デバイスの署名機能を用いて署名し，レスポンスデータを作成する．

(4)　OS 制御レイヤは配布サーバに対し新規 HTTPS セッションを確立する．
- (a)　OS 制御レイヤは (1) で取得した認証デバイス名とともに，レスポンスデータを配布サーバに送る．
- (b)　配布サーバは，受け取った認証デバイス名とレスポンスデータを認証サーバに送信する．
  - (i)　認証サーバは，受け取ったレスポンスデータをその認証デバイスに対応する公開鍵を用いて復号化する．
  - (ii)　復号化したデータと，認証要求時に保存しておいたデータを比較し，一致すれば保存しておいた OS 名を返す．一致しない場合はエラーを返す．
- (c)　配布サーバは，認証サーバから受け取った OS 名に対応する OS イメージを送信する．認証サーバからエラーを受け取った場合は HTTP エラー応答を送信する．

この認証プロトコルでは，通信に SSL を用いているため，鍵の機密情報が漏洩しない限り通信の安全性が保証される．また公開鍵暗号の理論から，認証サーバにおいてチャレンジデータとレスポンスデータの比較に成功した場合，クライアントシステムには起動を許可された認証デバイスが間違いなく挿入されていることが保証される．

4. 実　　装

提案した方式の有効性を検証するために，クライアントシステムを仮想計算機モニタ Xen を用いて実装し，認証・配布サーバについては Web サーバ上の CGI プログラムとして実装した．

1 台のハードウェア上に複数の仮想計算機（Virtual Machine: VM）を構築する仮想計算機技術の中で，Type-I VMM[5] と呼ばれるものは，BIOS と OS の間で仮想計算機モニタ（Virtual Machine Monitor: VMM）が動作する．VMM は実計算機資源の VM への割当てや，VM の管理などを行うため，VM 上で動作するゲスト OS にとっては BIOS に相当するレイヤと見なすことができる．この特徴を利用し，今回は検証のために Type-I VMM を用いることとした．

提案手法を検証するために用いる VMM は，前述の Type-I VMM であればよい．今回は，オープンソースであり，ホスト OS として Linux を利用できるという理由から Xen を用いる．OS 制御レイヤをホスト OS の Linux を利用して実装することで，Linux が持つ HTTP や SSL などのライブラリ，ソフトウェアやデバイスドライバなどを利用して実現す

ることができる．

Xen ではゲスト OS を DomainU と呼ばれる領域で起動し，DomainU は Domain0 と呼ばれる VMM 部分により管理される．利用者が要求した OS を DomainU として起動することにより，Domain0 で起動・終了制御を行うことができるようになる．この点に着目し，本研究では OS 制御レイヤを Xen の Domain0 上で実装することとした．ここでは，OS 制御レイヤを実装する Domain0 のことを**ホスト OS** と呼び，利用者が要求した OS のことを**利用者用 OS** と呼ぶ．

本実装では認証デバイスとして飛天ジャパン社の USB トークン ePass2000 を用いた．Domain0 の OS としては Linux 2.6.18（Fedora7）を，VMM としては Xen 3.2 を用いた．認証・配布サーバの OS としては Linux 2.6.18（Fedora7）を用い，HTTP デーモンとしては Apache 2.2.6 を，HTTP 上の認証プロトコル処理の実装には PHP 5.2.4 を用いた．また，認証サーバと配布サーバは同一ハードウェア上に実装した．

4.1　利用者用 OS の制御の流れ

提案システムにおける利用者用 OS の起動の流れを**図 2** に示す．電源が投入されると，提案システムは，利用者に対してまず USB トークンの挿入を要求する．USB トークンが挿入されたことを検知すると，提案システムは USB トークンから取得した認証デバイス名と，あらかじめ Domain0 内に保存されているハードウェア用証明書を用いて，認証をともなう利用者用 OS イメージの取得処理を行う．

認証・配布サーバからの利用者用 OS イメージの取得が完了すると，そのイメージを仮想計算機上で起動し，利用者がその OS を利用できるようにする．また，同時にホスト OS 上で USB トークンの抜き取りを監視するデーモンを実行する．

利用者用 OS の稼動中に USB トークンが抜かれた場合，監視デーモンがそれを検知し VMM のシャットダウン命令を利用して利用者用 OS を停止させる（**図 3**）．

4.2　クライアントシステムの実装

提案システムでは，クライアントシステムのハードウェアを識別するためにそれぞれのハードウェアに対してユニークな証明書をあらかじめ取得しておく．このハードウェア用証明書は，クライアントシステムに配置するホスト OS 内に同梱する．この証明書は，SSL クライアント認証で使用することができる X.509 証明書を用いた．また，証明書・秘密鍵の形式は PEM 形式とした．ハードウェア用証明書の格納については，将来 TPM を用いることにより，より安全に実現可能となる．

認証デバイスがハードウェアから抜かれたときに，起動中の OS が停止されることを保証

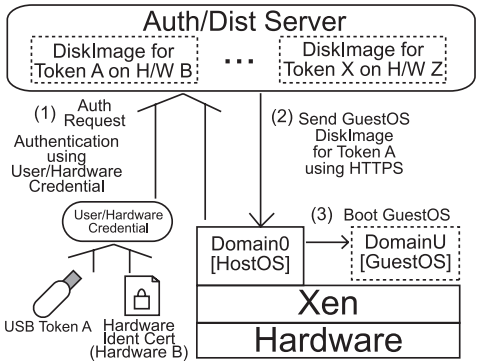

図 2　システム起動の流れ
Fig. 2　System booting flow.

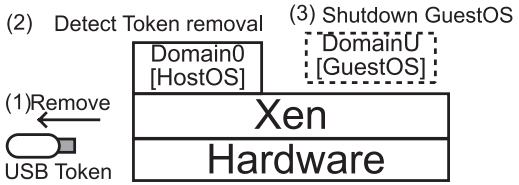

図 3　USB トークンが抜かれた場合
Fig. 3　Process flow on USB token has been removed.

するために，ホスト OS では USB トークンの監視処理を行う．1 度 USB トークンが抜かれたことが検出された場合，利用者用 OS はシャットダウンされる．

この処理は次のような流れで動作する．まず，異なる USB トークンが挿された場合にそのことを検出できるように，利用者用 OS を起動する前に認証デバイス名をホスト OS 上に記録しておく．その後，ホスト OS 上で定期的に USB トークンにアクセスし，認証デバイス名が記録されているものと一致するかを確認する（ポーリング方式）．USB トークンにアクセスできない場合や，取得した認証デバイス名が記録されているものと異なる場合，起動に使用した USB トークンが抜かれたものとして判定する．

USB トークンが抜かれたことを検出した場合，まず DomainU をシャットダウンさせるコマンドを実行する．実際には，次のコマンドを実行している．ここで，`domain` は DomainU の名前である．

```
# xm shutdown domain
```

コマンドを実行すると DomainU にシャットダウン命令が送られ，このコマンドを受け取った DomainU はシャットダウンを開始する．

上記のコマンドを実行した後も定期的な確認を行い，DomainU のシャットダウンが確認できない場合には DomainU の強制終了コマンドを実行する．このコマンドは DomainU の状態に関係なく DomainU を強制終了させる．これは実際の PC における電源断に相当する．実際には，次のコマンドを実行している．

```
# xm destroy domain
```

ここで，`domain` は DomainU の名前である．

4.3　認証サーバと配布サーバの実装

3.3 節で示した方式を実現するために，認証サーバおよび配布サーバ上で動作する CGI プログラム，および認証サーバ・配布サーバ間の通信処理を PHP を用いて作成した．認証サーバ用 CGI プログラムは，クライアントシステムからの認証リクエストを処理し，配布サーバ用 CGI プログラムでは，クライアントシステムから送られたレスポンスデータの検証や，ディスクイメージの送信を行う．

認証リクエストの処理では，クライアントシステムから送出されるリクエストのパラメータとして認証デバイス名を受け取り，また環境変数からハードウェア用証明書の CN をハードウェア名として取得する．認証デバイス名とハードウェア名から起動すべき OS 名を ACL から検索する．また，OS 名とハードウェア名から，担当する配布サーバの URI を決定する．

クライアントに返すデータは，その OS を担当する配布サーバの URI とチャレンジデータをつなげたバイナリデータであり，それらは LF（LineFeed）を用いて区切ることとした．これにより，クライアントシステムは受け取ったデータを LF で分割し，前半を URI，後半をチャレンジデータとして扱うことが可能となる．なお，認証情報の偽造に対する強度が向上するため，チャレンジデータの長さは USB トークンで署名を行うことができる最長のデータ長である 48 バイトとした．

起動が許可される組合せであった場合，前述の URI・チャレンジデータを OS 制御レイヤに返す．また，配布サーバからの問合せのために，OS 制御レイヤが起動しようとしている OS の名前と，送信したチャレンジデータをサーバ上に保存しておく．

もし ACL からの検索結果に対応する OS がない場合には，その認証デバイスとハードウェアの組合せでは OS の起動が許可されていないことを表すため，この処理の HTTP 応答としてステータスコード 404 NotFound を返す．

認証サーバ・配布サーバ間の通信処理では，認証サーバは配布サーバから送出されるリクエストのパラメータとして，認証デバイス名とレスポンスデータを受け取る．認証サーバはそのパラメータをもとに，その認証デバイスに対応する公開鍵を検索する．この公開鍵を用いてレスポンスデータの復号化処理を行い，保存してあるチャレンジデータとの比較を行う．比較の結果によらず，復号化処理を行った場合には一時的に保管したチャレンジデータは削除する．比較処理の結果，一致した場合には，この処理の応答として，一緒に保存していた OS 名を返す．一致しない場合は，HTTP 応答としてステータスコード 404 NotFound を返す．

レスポンス処理では，クライアントシステムから送出されたリクエストのパラメータとして認証デバイス名とレスポンスデータを受け取る．受け取ったパラメータを用い，認証サーバ・配布サーバ間の通信処理を実行する．正常な HTTP 応答が返ってきた場合には，そのボディから OS イメージ名を取得し，クライアントに対する応答として対応するイメージをボディとして送信する．エラー応答が返ってきた場合には，クライアントに対し HTTP 応答としてステータスコード 404 NotFound を返す．

5. 評　　価

提案方式の有効性の評価を行うため，異なる証明書が格納された 3 つの USB トークンと，2 つのハードウェアを用意し，実装されたシステムを用いて利用者用 OS の起動制御実験を行った．また，本システム特有の処理にかかる時間の計測を行った．

5.1　ハードウェア構成

実験には，次のようなハードウェア構成のサーバ・クライアントを用いた．それぞれのハードウェアは，**図 4** に示すように，1000BASE-T の Ethernet で接続されており，USB トークンとして Token A，B，C の 3 つと，クライアントシステムとして Hardware α，β の 2 台から構成される．

- クライアント Hardware α
  - CPU: Intel PentiumD 820 2.8 GHz
  - Memory: DDR2 SDRAM 1,024 MB
  - HDD: SerialATA 160 GB
- 認証・配布サーバ / クライアント Hardware β
  - CPU: Intel Core2Quad Q6600 2.4 GHz
  - Memory: DDR2 SDRAM 4,096 MB

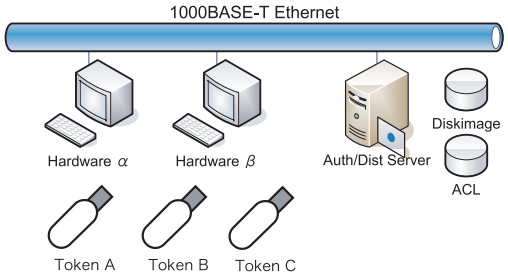

図 4　実験環境概略図
Fig. 4　Experimentation environment.

– HDD: SerialATA 250 GB

5.2 実験方法

まず初めに，**表 1** に示すような起動許可設定を行った．また，異なる証明書が格納された USB トークン Token A，B，C を用意し，対応する公開鍵をそれぞれ認証サーバに格納した．

次に，Hardware α 上でクライアントシステムを稼動させ，Token A，B，C をそれぞれ挿入し，どの利用者用 OS が起動するかを観察した．また，その OS の起動処理にかかる時間を計測した．最後に，Hardware β 上でクライアントシステムを稼動させ，Token A，B，C をそれぞれ挿入し，どの利用者用 OS が起動するかを観察した．

利用者から見た起動所要時間とは，USB トークンを挿入してから，利用者が利用者用 OS にアクセスできるようになるまでの時間である．その時間は，認証にかかる時間，OS イメージの転送にかかる時間，利用者用 OS の起動時間に分けることができる．しかし，利用者用 OS の種類や構成によりその起動時間は様々である．そこで，提案システムの評価として，認証にかかる時間を計測した．これは OS 制御レイヤに処理が移ってから，OS イメージの受信が始まるまでの時間で，ここでは認証フェーズ所要時間と呼ぶ．この所要時間が OS イメージの転送にかかる時間に比べどの程度かかるかを比較するため，OS イメージの受信開始から受信完了までの時間の計測も行った．ここではこれを転送フェーズ所要時間と呼ぶ．所要時間の測定は，それぞれの USB トークンについて 3 回ずつ行った．なお，OS A，B，C の OS イメージサイズは，それぞれ 540 MB，1,080 MB，1,540 MB とした．

5.3 実験結果

Hardware α を用いた起動実験では，Token A の挿入時には OS A が，Token B の挿入時には OS B が，Token C の挿入時には OS C が，それぞれ起動した．Hardware β を用いた起動実験では，Token A を挿入したときは OS A が，Token B の挿入時には OS C が起動したが，Token C を挿入したときには OS は起動しなかった．これにより，表 1 の設定どおりに起動制御が行われていることが確認できた．

表 1　OS 起動許可設定
Table 1　OS boot control table.

	Hardware α	Hardware β
Token A	OS A	OS A
Token B	OS B	OS C
Token C	OS C	No OS permitted

表 2　所要時間計測結果
Table 2　Average process time.

Image	Auth Phase[s]				Trans Phase[s]			
Size[MB]	1st	2nd	3rd	Avg	1st	2nd	3rd	Avg
540	11	11	11	11	15	16	17	16
1,080	13	11	11	12	31	32	33	32
1,540	11	10	11	11	52	52	53	52

また，Hardware α を用いた起動実験において，ディスクイメージのサイズが 540 MB，1,080 MB，1,540 MB である 3 種類の OS（OS A，B，C）について，認証フェーズと転送フェーズの所要時間をそれぞれ 3 回計測した．その実験結果をまとめたものを**表 2** に，それをグラフに表したものを**図 5** に示す．結果より，すべてのイメージサイズで認証フェーズの所要時間は約 11 秒と一定であるが，転送フェーズの所要時間はディスクイメージサイズにほぼ比例することが読み取れる．

5.4 考　　察

5.4.1 柔軟な起動許可設定

表 1 の設定では，Token A に着目すると，すべてのハードウェアで OS A が起動した．一方，OS A に着目すると，OS A の起動を許可されているトークンは Token A だけであるため，OS A の最大稼動数は 1 となる．これにより，最大稼動数を管理されている OS がハードウェアの制限なくどこでも稼動可能となっていることが分かる．

それに対し，Token B では，Hardware α では OS B が起動し，Hardware β では OS C が起動した．これにより，認証デバイスに対して，異なるハードウェアでは異なる OS が起

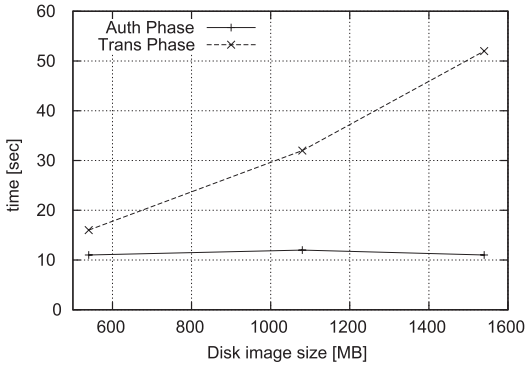

図 5　平均所要時間計測結果
Fig. 5　Average process time graph.

動するような柔軟な起動制御が実現されていることが確認できた．

Token C と OS C に着目すると，Hardware α でのみ OS が起動した．これにより，「起動を許可しない」という設定も可能であることが確認できた．

以上より，非常に柔軟な OS の起動制御が実現できているといえる．

5.4.2　認証フェーズ所要時間

所要時間の計測結果では，認証フェーズにおける平均所要時間は約 11 秒であった．今回使用した USB トークンでは，認証デバイス名の取得に約 4 秒かかり，チャレンジデータの署名に約 7 秒かかった．両者を合わせると約 11 秒となり，認証フェーズにおける平均所要時間の大半が USB トークンアクセスにかかる時間である．USB トークンに対する処理以外に要した時間は 1 秒以下であり，無視できる程度であった．

5.4.3　システムのスケーラビリティ

提案システムでは，配布サーバがどのトークンとハードウェアの組合せを担当するかという情報を認証サーバが一元管理しているため，クライアントシステムの数が増加した場合にも，配布サーバを追加することで対応することができる．

このとき，1 台の配布サーバがどの程度の数のクライアントシステムを処理できるかが重要になる．配布サーバの性能により，CPU の SSL 処理に要する処理能力または回線の帯域幅によるボトルネックが生じる．そのため，それぞれの利用者用 OS イメージの転送に要する時間の許容量から，配布サーバの台数や性能を見積もったうえで設置する必要がある．

5.4.4　完全仮想化と準仮想化

本論文では，組織内のコンピュータの管理に関して，管理支援の仕組みが備えるべき条件について考察し，仮想計算機の起動・終了を制御することによって実現する手法を示した．仮想計算機の実現方法には，完全仮想化（Full-Virtualization）と準仮想化（Para-Virtualization）があり，準仮想化の場合は OS 自体への変更が必要となるため利用可能な OS に制限が生じる場合がある．

今回の実装では，使用したハードウェアの制約から準仮想化による実装を行ったが，本論文の手法は完全仮想化に対応したハードウェアを使用して完全仮想化を行った場合にも適用可能である．

6. ま と め

本論文では，認証デバイスを用いた OS の起動・終了制御システムの提案を行った．提案システムでは，認証デバイスが挿入されたことにより適切な OS を起動させ，かつ，その認証デバイスが抜かれたときに，起動した OS を停止させるために，OS 制御レイヤを用いた．また，システムにおいて利用者の要求に応じて各種 OS のイメージを管理するために，配布サーバを用いた．また提案システムで用いる認証デバイス，ハードウェアの管理，および，認証デバイスとハードウェアの組合せに対して起動を許可する OS を管理するために認証サーバを用いた．OS 制御レイヤは，認証サーバに対し認証デバイスとハードウェアの認証を行い，配布サーバに対し OS イメージを要求し，OS の起動・終了制御を行う．

提案システムの有効性を検証するために，仮想計算機モニタ Xen と USB トークンを用いて提案システムを実装した．その実装システムを用いて OS の起動制御実験，および，本システムの起動に要する時間を測定した．その結果，適切に起動・終了制御が行えることが分かり，提案システムが有効であることを確認した．

参 考 文 献

1) Arbaugh, W.A., Farber, D.J. and Smith, J.M.: A secure and reliable bootstrap architecture, *SP '97: Proc. 1997 IEEE Symposium on Security and Privacy*, Washington, DC, USA, IEEE Computer Society, pp.65–71 (1997).
2) Bajikar, S.: Trusted Platform Module (TPM) based Security on Notebook PCs-White Paper (2002). http://www.intel.com/design/mobile/platform/downloads/Trusted_Platform_Module_White_Paper.pdf (accessed 2008-01-10)
3) Barham, P., Dragovic, B., Fraser, K., et al.: Xen and the art of virtualization,

SIGOPS Oper. Syst. Rev., Vol.37, No.5, pp.164–177 (2003).
4) Barlow, D.: Building your own live CD, *Linux J.*, Vol.2005, No.132, p.2 (2005).
5) Goldberg, R.: Survey of Virtual Machine Research, *IEEE Computer Magazine*, Vol.7, pp.34–45 (1974).
6) Hewlett-Packard Corp., et al.: Advanced Configuration and Power Interface Specification (2006). http://www.acpi.info/DOWNLOADS/ACPIspec30b.pdf (accessed 2008-05-23)
7) Intel Corporation: Preboot Execution Environment (PXE) Specification Version 2.1 (1999). http://download.intel.com/design/archives/wfm/downloads/pxespec.pdf (accessed 2008-01-10)
8) Intel Corporation: Intel Trusted Execution Technology Preliminary Architecture Specification (2007). http://download.intel.com/technology/security/downloads/31516804.pdf (accessed 2008-01-10)
9) Itoi, N., Arbaugh, W.A., Pollack, S.J. and Reeves, D.M.: Personal Secure Booting, *ACISP*, pp.130–144 (2001).
10) Lie, D., Thekkath, C.A. and Horowitz, M.: Implementing an untrusted operating system on trusted hardware, *SOSP '03: Proc. 19th ACM symposium on Operating systems principles*, ACM, pp.178–192 (2003).
11) 中村めぐみ，宗藤誠治，須崎有康ほか：トラステッド・コンピューティングによる HTTP-FUSE KNOPPIX クライアントのセキュリティ強化，情報処理学会研究報告，Vol.2006-CSEC-34, pp.223–230 (2006).
12) 須崎有康，八木豊志樹，飯島賢吾ほか：ネットワークに対応した分割圧縮ループバックデバイス HTTP-FUSE-CLOOP とそれから起動する Linux，インターネットコンファレンス 2005 (2005).

(平成 20 年 6 月 10 日受付)
(平成 20 年 12 月 5 日採録)

高田　真吾（学生会員）

1985 年生．2008 年筑波大学第三学群情報学類卒業．現在，筑波大学大学院システム情報工学研究科コンピュータサイエンス専攻博士前期課程に在学中．仮想計算機モニタを用いた OS の起動制御の研究に従事．

佐藤　　聡（正会員）

1967 年生．1991 年筑波大学第三学群情報学類卒業．1996 年筑波大学大学院工学研究科単位取得退学．同年広島市立大学情報科学部助手．2001 年筑波大学大学院システム情報工学研究科講師．現在，同大学学術情報メディアセンター勤務．博士（工学）．キャンパスネットワークの企画管理運用，ネットワークデータベース，言語処理等の研究に従事．電子情報通信学会，ACM-SIGMOD-JAPN 各会員．

新城　　靖（正会員）

1965 年生．1993 年筑波大学大学院工学研究科電子・情報工学専攻博士課程修了．同年琉球大学工学部情報工学科助手．1995 年筑波大学電子・情報工学系講師，2003 年同助教授．2007 年准助教授．オペレーティング・システム，分散システム，仮想システム，並行システム，情報セキュリティの研究に従事．1995 年情報処理学会山下記念研究賞受賞．博士（工学）．日本ソフトウェア科学会，ACM，IEEE CS 各会員．

中井　　央（正会員）

1968 年生．筑波大学第三学群情報学類卒業，同大学大学院工学研究科修了（博士（工学））．1997 年 10 月図書館情報大学助手，2001 年 8 月同総合情報処理センター講師，2002 年 8 月同助教授，2002 年 10 月の筑波大学との統合により，筑波大学図書館情報メディア研究科助教授（学術情報メディアセンター勤務）．日本ソフトウェア科学会，ACM，ACM-SIGMOD-JAPAN 各会員．

板野　肯三（正会員）

1948 年生．1977 年東京大学大学院理学系研究科物理学専門課程単位取得後退学．1993 年より筑波大学電子・情報工学系教授．計算機アーキテクチャ，分散システム，プログラミングシステム等に関する研究に従事．理学博士．日本ソフトウェア科学会各会員，電子情報通信学会，ACM，IEEE CS 各会員．