2005－OS －98（1）
2005／2／22

プライバシ保護を実現するオペレーティングシステムのファイルアクセス制御手法

一柳 淑美† 鈴来 和久† 毛利 公一†† 大久保 英嗣††
† 立命館大学大学院理工学研究科 †† 立命館大学情報理工学部

企業の計算機から顧客情報が漏洩し，顧客のプライバシが侵害されるようなプライバシ情報の漏洩が問題となっている．漏洩を防ぐ技術として，セキュリティ技術がある．しかし，既存のセキュリティ技術は，ユーザや計算機を単位としてファイルを保護するため，プライバシ情報を保護するには粒度が粗い．そこで，本論文では，OS において粒度の細いプライバシ情報の保護手法を提案する．OS で保護することにより，信頼性と汎用性のある保護も可能となる．また，提案するファイル保護機構を Linux カーネルに実装し，その性能を評価した．その結果，ファイルオープンのオーバヘッドは約 250μs，ファイルの読出しと書込みでは，最大で，それぞれ 390μs と 490μs のオーバヘッドであった．

A Control Method of File Access in Privacy-Aware Operating System

Yoshimi Ichiyanagi† Kazuhisa Suzuki† Koichi Mouri†† Eiji Okubo††
†Graduate School of Science and Engineering, Ritsumeikan University
††College of Information Science and Engineering, Ritsumeikan University

The leakage of privacy information such as customer's information in the computers of enterprises is a serious problem. The security technology is one of prevention methods of leakage. However, in the conventional security technology, the granularity of protection is too coarse because a unit of protection is a user or a computer. In this paper, we propose a fine-grained privacy protection method in the operating system's layer. By this method, the operating system can provide any application with reliable and general protection functions. Furthermore, in this paper, we have developed the mechanism for file protection in Linux kernel, and evaluated it's performance. Experimental results show that the overhead to open, read, and write a file are approximately 250μs, 390μs, and 490μs respectively.

1　はじめに

現在，企業の計算機から顧客情報が漏洩することによって，顧客のプライバシが侵害されることが問題となっている．この問題を解決するため，計算機のハードディスクに保存されているデータが漏洩することを防ぐ技術が必要である．このデータ漏洩を防ぐ技術として，セキュリティ技術がある．セキュリティ技術は，悪意のある第三者の攻撃から計算機資源を守ることを目的としている．そのため，人為的ミスによるデータ漏洩を防ぐことが困難である．しかし，ユーザは，企業にプライバシ情報を提供することで，種々のサービスを受けている．このとき，企業のデータ管理者へ計算機に保存されているプライバシ情報のアクセスを許可しながら，データ管理者によるプライバシ情報の漏洩を防ぐ必要がある．すなわち，プライバシ情報の漏洩を防ぐためには，ユーザ，計算機単位で保護するのではなく，より細い粒度での保護が必要である．

以上の背景から，現在，より細い粒度でプライバシ情報を保護するオペレーティングシステム (以下，OS と記す) を開発している [1]．また，本 OS では，より細い粒度でプライバシ情報を保護するため，プライバシ情報を，データとそれをどのように保護すべきかを記述したデータ保護ポリシの組で表現する．すなわち，ファイルごとに保護ポリシを設定可能としている．また，本 OS では，ある瞬間の状態をコンテキストとして利用可能としている．コンテキストには，時刻，位置，プロセスの動作履歴がある．これによって，ユビキタス環境などの，人や計算機が移動するような場合にも対応可能としている．

以下，本論文では，2 章で本 OS におけるプライバシ情報の定義について，3 章でデータ保護に使用するコンテキストについて，4 章で本 OS の特徴，適用例，コンテキストに適応したファイル保護手法について述べる．5 章でファイル保護機能を実装した Linux カーネルについて述べ，さらに 6 章で関連研究について述べる．

2　プライバシ情報の保護

2.1　プライバシ情報

本 OS のデータ保護機構は，保護ポリシに従ってデータを保護する．また，本 OS では，保護ポリシが改竄されることも防ぐ．

プライバシ情報をファイルとして保存する場合，保護ポリシは，このファイルと対にしてファイルに保存される．本 OS は，保護するファイルと保護ポリシファイルを一対のものとして管理する．保護ポリシは，データ発信者の意思，または，データ発信者とデータ管理者の合意に基づいて作成される．

2.2　OS によるプライバシ保護

データ保護を行う場合，まず，データ発信者は，データを提供するか否かを検討し，提供する場合には，データをどのように保護したいのかを考え，この保護方法をデータ管理者に伝える必要がある．このとき，このデータ発信者の意図を計算機システムにどのように反映するかが問題となる．この問題を解決するためには，アプリケーション，システムソフトウェア，ハードウェアのいずれかにおいて解決することが考えられる．次に，それぞれにおけるファイル保護の信頼性と運用における汎用性の有無について議論する．

アプリケーションにおいてデータを保護する場合を考える．アプリケーションにおいて保護する場合，コンピュータウイルスやスパイウェアなどにより，データが漏洩する危険性が高い．また，アプリケーションごとにプライバシ情報を保護する機構が必要となる．

システムソフトウェアにおいてデータを保護する場合を考える．OS において保護する場合，アプリケーションに対して統一的な保護を提供できるため，汎用性が高い．また，アプリケーションにおける保護より，保護に対する信頼性は高くなる．

ハードウェアにおいてデータを保護する場合を考える．ソフトウェアレベルでの保護よりも，保護に対する信頼性は高い．しかし，特殊なハードウェアを必要とするため，汎用性が低く，経費がかかる．

以上の考察より，データの保護に対して信頼性があり，運用においても汎用性が高い OS でデータを保護する手法を採用する．しかし，従来の OS では，ユーザごとに読出し，書込み，実行の各アクセス権限を設定するファイル保護しか行っていない．また，人為的ミスによるデータの漏洩も考慮していない．そこで，本 OS では，ファイルを単位としたデータ保護機能を提供する．

2.3　保護モデル

本 OS では，プライバシ情報の漏洩と改竄，ユーザの同意なくユーザのプライバシ情報を第三者に提供することを防ぐことを目的としている．このため，プロセスによる計算機のハードディスクに保存されているデータを読み出す処理が，データを漏洩させる処理であるか否かを判断する必要がある．そこで，本 OS は，データ保護のために，プロセス動作の履歴を取得する機能 (History Logger) とデータを保護する機能 (Action Contoroller) を持つ (図 1 参照)．プロセス動作の履歴を取得する機能とは，プロセスによるファイルに記述されているデータを漏洩させる危険性がある処理を検出するための機能である．具体的には，プロセスが発行するシステムコールを記録する．データを保護する機能とは，プロセスがファイルにアクセスしたときにコンテキストを取得し，保護ポリシとコンテキストに適応してファイルを保護する機能である．

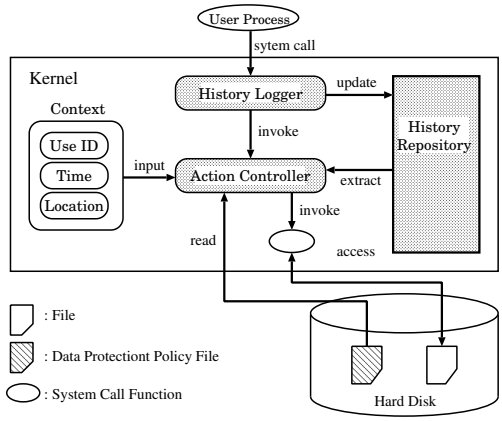

図 1　データ保護モデル

3　コンテキスト

本 OS では，データ保護のため，以下をコンテキストとする．

- OS 内部で管理されるコンテキスト
  実ユーザ ID・実効ユーザ ID・プロセス ID・相対時刻
- アプリケーションが関与するコンテキスト
  システムコールの番号・引数・返り値とプロセスごとのシステムコール履歴
- ハードウェアから得られるコンテキスト
  絶対時刻・位置

また，本 OS では，コンテキストを用いてデータを保護するため，コンテキストの信頼性が問題となる．そこで，OS がコンテキストの取得，管理を行う．ただし，コンテキストをどのように取得しているかによって，コンテキストの信頼性は異なる．

OS 内部で管理されるコンテキストは，プロセスの状況を表す．このコンテキストに適応してデータを保護することにより，プロセスごとにデータ保護ポリシを設定できる．また，このコンテキストは，データを保護する上において，最も信頼できるパラメータであると位置づけられる．

アプリケーションが関与するコンテキストは，どのような処理を要求しようとしているのか，または，要求したのかを表すものである．アプリケーションがデータ漏洩を引き起こす可能性がある処理を要求したとき，このコンテキストに適応してデータを保護することにより，この処理を制限することができる．したがって，このコンテキストは，関与しているアプリケーションの信頼性に依存する．しかし，計算機資源にアクセスするためには，アプリケーションがこれらのパラメータを正しく指定する必要がある．したがって，このコンテキストも，データを保護する場合に用いるパラメータとして，信頼できる．

ハードウェアから得られるコンテキストは，本 OS が動作している計算機以外の計算機，デバイスから取得するため，このコンテキストの信頼性を保証することは困難である．このため，このコンテキストを用いてデータを保護する場合，データ発信者は，このコンテキストが信頼できるのか否かを検討し，データ保護に使用するか否かを考える必要性がある．例えば，GPS(Global Positioning System) , RFID(Radio Frequency Identification System) タグから ID を取得し対応する場所を検索できる位置情報システムが，使用できる環境を想定する．このとき，RFID タグの ID を書き換えたり，偽造することによって，位置情報を容易に偽ることができる．しかし，GPS から取得する位置情報を偽ることは困難である．したがって，GPS から取得できる位置情報は信頼性が高いが，RFID タグから取得できる位置情報は GPS より信頼性が低くなる．データ発信者は，GPS と RFID タグから取得できる位置情報を信頼するか否かを判断し，この 2 つの位置情報を使用してデータ保護を行うか否かを決める必要がある．

本 OS では，システムコール履歴は，プロセスがファイルを読み出したか否かを検索するためのものである．したがって，プロセスが保護するファイルをオープンしたことを契機に取得し始める．それ以外のコンテキストは，保護するファイルにアクセスするたびに取得される．

4　ファイル保護機能

4.1　特徴

本 OS は，ファイルとして保存されているデータをコピーするプロセスの動作を制御する．本 OS のファイル保護の特徴を以下に示す．

- ファイルシステムの種類に依存しない．
  本 OS のデータ保護機構は，ファイルシステムを変更せずに実現している．このため，本 OS の導入を容易に行うことができる．
- 対となる保護ポリシファイルが存在するファイルを保護する．

本 OS では，保護するファイルと対となる保護ポリシファイルを作成する．この保護ポリシファイルが存在することにより，動的な環境の変化に適応してデータを保護することができる．

- ファイルへのアクセスを制御する．
  本 OS では，ファイル保護として，ファイルへのアクセスを監視し，それを制御する．

4.2 保護ポリシファイル

ユーザは，以下の項目を保護ポリシファイルに記述する．

- デフォルトのファイルアクセスポリシ
- コンテキストに適応して制御するためのファイルアクセスポリシ
- ファイルを削除するポリシ

これらのポリシは，いずれも，着目すべきコンテキスト，コンテキストに対する条件，条件を満たした場合と満たさなかった場合の動作を組として，アクセス制御リストとして記述される．デフォルトのファイルアクセスポリシとファイルを削除するポリシは，1 つの項目のみが記述できる．コンテキストに適応して制御するためのファイルアクセスポリシは，複数項目を記述できる．これらの項目のうち，データ発信者がコンテキストに適応して制御するためのファイルアクセスポリシの項目を記述していない場合，デフォルトのアクセスポリシに記述されているアクセス制御を行う．また，デフォルトのファイルアクセスポリシを記述していない場合，コンテキストに適応して制御するためのファイルアクセスポリシの項目に一致しないファイルへの読出しと書込みのアクセスは禁止される．さらに，ファイルアクセスに関しての項目を記述していない場合，保護するファイルへの読出しと書込みのアクセスは禁止される．ファイルを削除するポリシを記述していない場合，データの消失を防止するためには，このファイルを削除しない．

4.3 システムコールフックによるアクセス制御

プロセスによるファイルへのアクセスを制御するために，ファイルをオープンするシステムコール，ファイルをクローズするシステムコール，ファイルに書き込むシステムコール，ファイルを読み出すシステムコールをフックする．以下に，それぞれのシステムコールにおけるデータ保護のための処理を示す．

- open
  オープンするファイルと対になる保護ポリシファイルを検索する．ポリシを発見した場合はそのポリシファイルを読み出し，OS のみ参照できるアクセス制御リストに登録する．このアクセス制御リストを参照してファイルへの書込みと読出しを制御する．また，保護するファイルが読み出されていた場合，保護ポリシによっては，保護しないファイルへの書込みを禁止する．
- close
  本 OS がアクセス制御をしているファイルをクローズする場合，このファイルをオープンしたときに登録したこのファイルに対する保護ポリシを，アクセス制御リストから削除する．
- write
  コンテキストを取得し，このコンテキストに対応する保護ポリシをアクセス制御リストから検索する．この保護ポリシに応じてアクセスを制御する．
- read
  ファイルに書き込むシステムコールと同様のアクセス制御を行う．さらに，ファイルの保護ポリシによっては，このプロセスのファイルへの書込みを禁止するため，システムコール履歴にこの書込みを禁止する情報を付加する．

4.4 不要となったデータの削除

データ発信者が保護するファイルに対するアクセスを許可している間は，そのファイルがハードディスクに保存されている必要がある．しかし，必要としなくなったときにそのファイルを削除することにより，データ漏洩を予防することができる．そこで，不要となったデータが格納されているファイルを削除する．以下にその処理を示す．

1. 削除するファイルをオープンしているプロセスが存在した場合には，強制的にこのファイルをクローズする．
2. 削除するファイルをゼロで初期化する．
3. ファイルをアンリンクする．

ファイルを削除する条件となるコンテキストは，保護するファイルを作成したときに本 OS に登録され，保護ポリシファイルに書き込まれる．その後，削除する条件となるコンテキストが切り替わり，この削除する条件となるコンテキストのパラメータの値と一致したときにファイルを削除する．しかし，現在時刻のように，割込みなどの明確なイベントがないものを条件として削除するような場合が考えられる．このとき，そのコンテキストが切り替わるタイミングでファイルを削除することが困難である場面が存在する．削除する時刻に計算機の電源が入っていない場合にファイルを削除することは不可能である．したがって，ファイルをオープンするときにおいても，保護ポリシからファイルを削除する条件となるコンテキストを検索し，削除する時点を過ぎているか否かを調べる．この時点を経過していた場合は，ファイルを削除する．

4.5 適用例

動的な環境の変化に適応してデータを保護する例を次に示す．

学校において，生徒がノート PC を携帯している環境を想定する．このとき，授業中の試験を受ける教室内では生徒にレジュメを見せるが，授業以外の時間帯や，授業中でもその教室以外でレジュメを見せないようにする場合を考える．このとき，レジュメにアクセスしようとする時刻，生徒の位置に応じて，ファイルへのアクセスを制限する必要がある．また，授業が終了した後，生徒の PC にレジュメが保存されている場合，この PC からレジュメがコピーされる危険性がある．このため，授業が終了したときに，生徒の PC からレジュメを削除する．

また，企業において，データ管理者が，業務上，顧客情報を閲覧し，更新する必要性がある場合を想定する．この場合，顧客情報を読み出した後には，他のファイルに顧客情報を他のファイルに書き込む処理を禁止する必要がある．そこで，プロセスの動作履歴に基づいたデータ保護が必要である．

以上より，動的に変化する環境においてデータを保護するためには，より細い粒度のデータ保護が必要である．しかし，従来のデータ保護は，コンテキストに基づく保護ができないため，動的な環境の変化に対応することは不可能である．本 OS では，従来のデータ保護に加えて，コンテキストに基づいてデータを保護することが可能である．

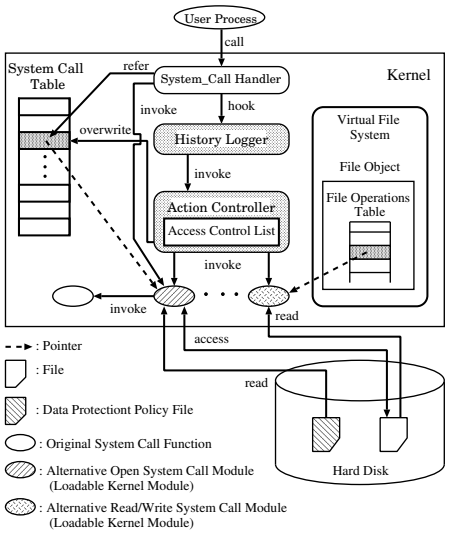

図 2　システムコールフック

5 実装と評価

5.1 実装

データ保護機能を Linux-2.6.6 内へ実装している．その構成を図 2 に示す．プロセスは，ファイル・ネットワーク・メモリなどの計算機資源を使用する場合，システムコールを発行する．そこで，本 OS では，プロセスのデータ漏洩が発生する可能性がある計算機資源を使用する処理を制御するために，システムコールをフックする．本 OS では，ファイルアクセス制御のために，open，close，read，write システムコールをフックしている．

read/write システムコールは，VFS(Virtual File System) のファイルオブジェクト内のファイル操作テーブルに登録された関数からファイルにアクセスしている．そこで，この関数を置き換えることによりフックする．これにより，制御するアクセスのみをフックし，制御する対象ではないアクセス処理のオーバヘッドをゼロとしている．

一方，close システムコールでは，read/write システムコールのような仕組みで処理を行わない．また，open システムコールでは，アクセスしようとしているファイルへの読出しと書込みを制御するか否かを判断する必要がある．このため，open システムコールでは，すべてのアクセスをチェックしなければならない．そこで，open/close システムコールは，システムコールテーブルに登録されている関

数のアドレスを，ファイルアクセスを制御する関数と置き換えることによりフックする．システムコールテーブルを置き換えることにより，すべてのアクセスを制御することができる．

本 OS では，システムコールの履歴に基づくファイルアクセス制御を実装している．以下のシステムコールが発行されているか否かをこの履歴から検索し，アクセスを制御している．

read システムコール　このシステムコールに着目することにより，保護するファイルのコピーを作成する可能性があるアクセスを制御できる．具体的には，プロセスが保護するファイルを読み出した後，このファイル以外に書き込む処理を禁止することができる．

write システムコール　このシステムコールに着目することにより，ユーザが保護したいデータを書き込んだファイルを保護することができる．具体的には，ファイルにデータを書き込んだ場合，他のプロセスからの，このファイルへのアクセスを禁止する．これは，書き込んだファイルへの保護ポリシの継承を意味する．

5.2　評価

本 OS には，以下のコンテキストに基づくファイルアクセス制御を実装している．

- ユーザ ID・実効ユーザ ID
- OS 内部で管理されている RTC から取得できる絶対時刻・相対時刻
- 位置情報として，通信している無線 LAN の ESSID(Extended Service Set ID)
- 保護するファイルを読み出したシステムコールの履歴
- 保護するファイルに書き込んだシステムコールの履歴

本 OS の性能を評価するため，以下のアクセスを比較した．

- 標準 Linux-2.6.6 におけるファイルアクセス
- 本 OS における保護しないファイルへのアクセス
- 本 OS における保護するファイルへのアクセス

これらのアクセスについて，シングルユーザモードで，ユーザプロセスが open，close，write，read システムコールを発行してから，ユーザプロセスに応答が返ってくるまでの時間を計測した．実験環境は，表 1 に示す計算機を用いた．また，データ保護ポリシには，以下の条件を記述し，ファイルの読出しと書込みを許可する場合について 10 回計測した．

表 1　計算機構成

計算機	IBM PC/AT 互換機
CPU	Celeron 900Mhz
メモリ	256MB
ファイルシステム	ReiserFS

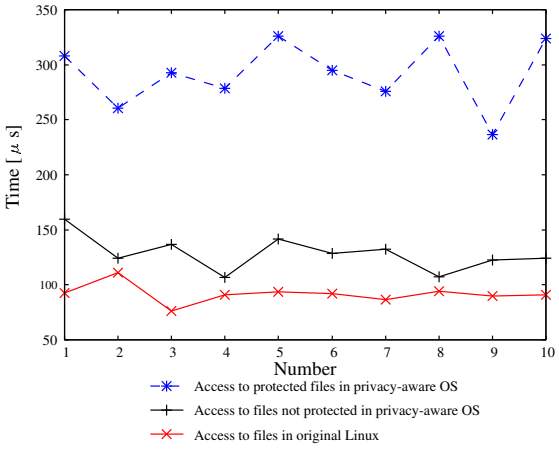

図 3　ファイルオープンのオーバヘッド

- 実ユーザ ID と実効ユーザ ID が データ発信者，または，データ管理者であるか否か
- ファイルをオープンしてから，5 秒以内であるか否か
- 通信している無線 LAN の ESSID が実験環境で使用している ESSID と一致し，最大値が 88 である電波強度の値が 10 以上であるか否か

計測に使用した保護ポリシファイルのサイズは，183 バイトである．また，アクセスするファイルのデータがキャッシュにない状態で読み出す場合と，書き込む場合について，それぞれ異なるファイルに対して計測した．

実験結果として，図 3 に open システムコール，図 4 に close システムコール，図 5 に read システムコール，図 6 に write システムコールの計測した結果を示す．

本 OS における保護しないファイルへのアクセス処理のオーバヘッドは，図 3 より，ファイルをオープンする場合では，約 30μs のオーバヘッドがある．

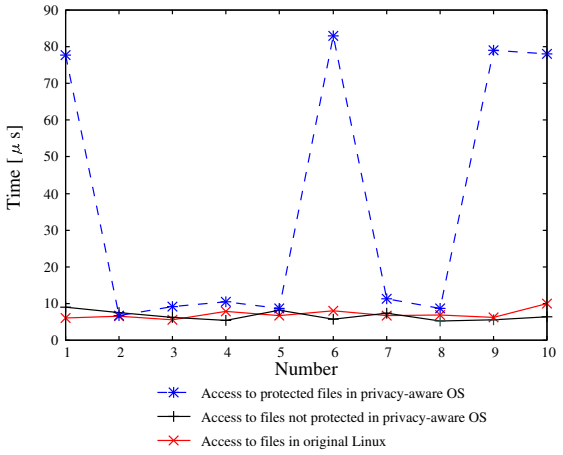

図 4　ファイルクローズのオーバヘッド

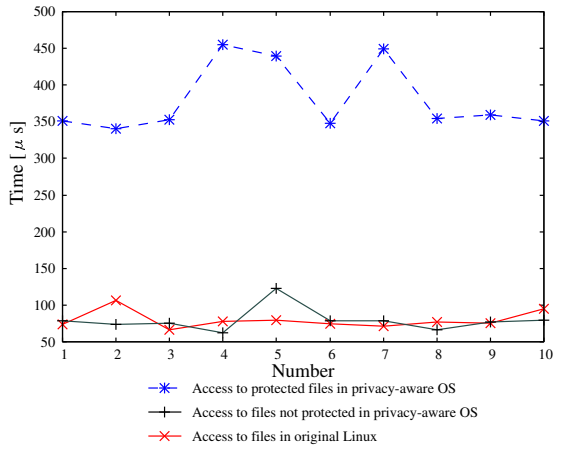

図 5　ファイル読出しのオーバヘッド

これは，本 OS では，ファイルをオープンする際に，オープンしようとするファイルの保護ポリシファイルを検索するオーバヘッドである．一方，保護しないファイルの読出しと書込み処理は，標準の Linux と同一の処理をしているため，オーバヘッドはないと考えるられる．このことは，図 5 と図 6 のそれぞれの値を比較した結果より，証明された．クローズの場合では，このファイルが保護するファイルか否かを判断する処理がオーバヘッドになると考えられる．しかし，この処理によるオーバヘッドは，図 4 のそれぞれの値を比較した結果，問題にならないといえる．以上より，保護しないファイルへのアクセスのオーバヘッドはファイルをオープンするときのみである．したがって，4.3 節で述べたシステムコールをフックするオーバヘッドはない．

本 OS における保護するファイルへのアクセスのオーバヘッドは，ファイルオープンでは，図 3 より 250μs 以内，ファイルクローズでは図 4 より 78μs 以内であった．ファイルオープン処理のオーバヘッドの要因は，保護ポリシファイルを読み出す処理である．したがって，このオーバヘッドは保護ポリシファイルに依存する．

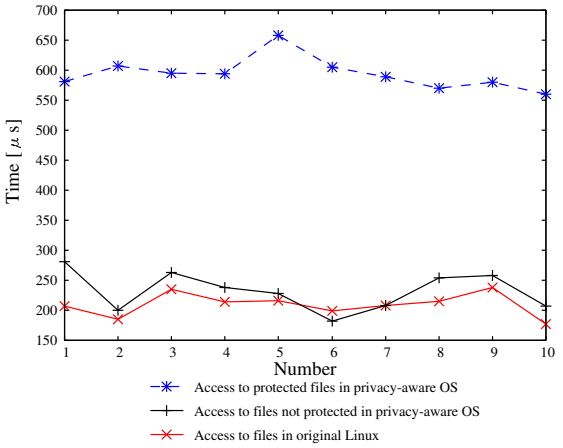

図 6　ファイル書込みのオーバヘッド

また，ファイルの読出しでは，図 5 より 390μs 以内，ファイルへの書込みでは図 6 より 490μs 以内であった．このオーバヘッドの要因として，コンテキストを取得する処理にかかるオーバヘッドがある．そこで，OS において無線 LAN の ESSID と電波強度を取得するのにかかる時間と，絶対時刻を取得する時間を計測した．その結果，無線 LAN の ESSID と電波強度の取得にかかる時間は，300μs ミリ秒程度，時刻の取得にかかるオーバヘッドは 0.8μs 以下となった．したがって，ファイルの読出しと書込みにおける最大のオーバヘッドは，無線 LAN から ESSID と電波強度を取得する処理にかかる時間である．

以上より，本 OS における保護しないファイルへのアクセス処理のオーバヘッドは，ユーザが使用する上で，不便さを感じない程度であった．一方，保護するファイルへのアクセスのオーバヘッドは，保護ポリシファイルのサイズとコンテキストの取得にかかる時間に依存し，増加する．そこで，ファイルサイズを小さくするための保護ポリシの記述法を検討することと，コンテキスト取得にかかる時間を小さくする実装を行う必要がある．また，保護するファイルへのアクセス処理にかかる時間には，約 77μs から 98μs のゆらぎがある．このゆらぎは，何らかの割込みであると考えらる．この割込みの種類に関しては，調査中である．

6　関連研究

システムコールフックを利用したファイル保護手法は，数多く存在する．例えば，Linux カーネルに組み込まれている LSM (Linux Security Modules)[2] がある．LSM は，悪意ある攻撃から計算機を守ることを目的とした，OS が管理する資源に対してアクセス制御をするカーネルフレームワークである．LSM は，ユーザごとにファイルシステムの操作の可否を細く指定できる．LSM のようなシステムコールをフックしてファイルを保護する手法と本 OS のデータ保護手法の違いは，本 OS では，人や計算機が移動する環境でのファイル保護を考慮している点である．したがって，システムコールフックを利用したファイル保護において，本 OS の特徴はファイル保護の仕方が動的に切り替わる点であるといえる．

また，ユーザのプライバシを考慮している暗号化ファイルシステムとして，TCFS(Transparent Cryptographic File System)[3] がある．この TCFS は，OS で NFS(Netwark File System) サーバに保存されているファイルを保護する．アプリケーションにおいてファイルを暗号化するよりも，ユーザにファイルを暗号化，複合化するという行為を意識させないため，使いやすいと考えられる．また，ユーザが使用している計算機において暗号化，複号化するため，NFS サーバの信頼性を問わない．また，ネットワークを介して通信されるファイルデータは，すべて暗号化されるため，盗聴によるデータ漏洩を防ぐことができる．しかし，ファイルの暗号化鍵はユーザごとに管理される．このため，プライバシ情報を保護するためには，より細い粒度の保護が必要である．また，本 OS では，ネットワークを介して保護するファイルを送信する場合，このファイルのデータを盗聴から守ることを考慮していない．そのため，暗号化ファイルシステムと本 OS を組み合わせることによって，より信頼性の高いプライバシ情報の保護を提供できると考えられる．

7　おわりに

本論文では，ユーザのプライバシ情報が書き込まれたファイルを保護する手法，この手法を実現する OS について述べた．本 OS では，細い粒度でファイルを保護する．このとき，データ発信者は，データ保護機構にどのようにファイルを保護したいのかを伝えるため，保護ポリシファイルを作成する．本 OS は，この保護ポリシファイルに記述されたポリシに従ってファイルを保護する．この保護ポリシにコンテキストという概念を組み込むことで，粒度の細い保護を実現している．また，本論文では，本 OS の評価を行った．本 OS を用いて，ユーザ ID，時刻，位置，プロセスごとのシステムコール履歴に適応したプロセスのファイルアクセスを制御することができた．また，そのオーバヘッドを計測した場合，オープンで約 $300\mu s$，読出しと書込みで，それぞれ，$390\mu s$，$490\mu s$ 程度であった．このオーバヘッドの要因として，保護ポリシファイルを読み出す処理と，コンテキストの構成要素であるパラメータの値を取得処理が挙げられる．

今後の課題としては，保護ポリシファイルのサイズをより小さくするための保護ポリシの記述法とコンテキストの構成要素であるパラメータの値を取得する処理のオーバヘッドを減らす仕組みを考えることが挙げられる．

参考文献

[1] 鈴来 和久，一柳 淑美，毛利 公一，大久保 英嗣: プライバシ保護を実現するオペレーティングシステムにおけるコンテキスト管理，情報処理学会研究報告 2004-OS-96，Vol.2004，No.63，pp. 7–14 (2004).

[2] Linux Security Modules: http://lsm.immunix.org/

[3] Giuseppe Cattaneo, Luigi Catuogno, Aniello Del Sorbo, Pino Persiano: The Design and Implementation of a Transparent Cryptographic File System for UNIX，USENIX Annual Technical Conference 2001，pp .199–212 (2001).