携帯端末向けLinuxのリソース管理の実現

稗田諭士[†] 才田好則[†] 中山義孝[††]
千嶋博[†] 中本幸一[†,☆]

携帯端末のOSとしてLinuxの採用が検討されており，サードパーティプログラムの導入容易化や端末外からのプログラムダウンロードなどによる新サービスが期待される．しかし，携帯端末ではハードディスクがない，搭載されるメモリ量がPCに比べてはるかに小さいなどの理由から，Linuxにおいてアプリケーションプログラムがメモリをはじめとするリソースを過度に使用した場合には，電話としての安定動作を損なう可能性がある．そこで本論文では，携帯端末向けLinuxにおいてリソース制御を実現するための方式について述べる．本方式では，Linux上の各プロセスが使用するカーネル管理下のリソース量，ユーザ空間のリソース量，ファイル使用量をLinuxカーネル内で管理することで，ユーザプログラムに修正を加えることなく，Linuxのリソース管理を強化することが可能である．これにより，携帯端末内の各プロセスが使用可能な端末内リソースの上限が設定でき，端末システム全体として安定したソフトウェア実行環境が保証される．

Resource Management in Linux for Mobile Terminals

Satoshi Hieda,[†] Yoshinori Saida,[†] Yoshitaka Nakayama,[††]
Hiroshi Chishima[†] and Yukikazu Nakamoto[†,☆]

Linux is expected as the next generation mobile terminal OS. Using Linux, easy introduction of third party programs and new services by downloading programs will be able to be realized. However, since the mobile terminals provide no hard disks and less memory space than PC, too large resource consumption by application programs prevents stable behaviors as a phone. In this paper, we propose methods to manage various kinds of resources on Linux. In the methods, Linux manages amounts of kernel-managed resource, resources in user spaces, and files, which are consumed by processes. They enhance resource management functionalities without modifying user program and libraries. Using these methods, available resource usage for each process on Linux is controlled and stable software runtime environment is realized.

1. はじめに

近年，Webブラウザや Java 実行環境の搭載など，携帯端末の高機能化によって，携帯端末上に実装されるソフトウェアの規模が増大している．これまで多くのアプリケーションソフトウェアやミドルウェアが汎用OS上で開発・製品化され，携帯端末上で利用されるためにリアルタイムOS（RTOS）上に改造，あるいは移植されてきている．こうしたRTOSへの移植コストが大きい，加えて携帯端末向けクロスソフトウェア開発環境が容易に入手できない，などの理由から，従来のRTOS上の開発では生産性の向上は難しいといわれている．こうした状況に対応するために携帯端末のOSとしてLinuxの採用が検討されており，このような高機能かつ汎用的なOSを利用することで，開発効率が向上する，サードパーティの作成したアプリケーションの導入が容易になる，などのメリットが見込まれている．またLinuxを採用することで，将来には通信を介しての端末外からのソフトウェアダウンロードやアップデートなどの新サービスが考えられる．しかしながら，悪意のあるソフトウェアから確実に端末内の個人情報を保護し，端末内ソフトウェアの安定動作を保証するための機能強化が必要となる．

携帯端末用OSとしてLinuxを採用することは，上述したメリットがあるものの，セキュリティ面では以下のような問題点があげられる．

- ルート権限を持つプロセスがすべてのファイルを改竄することができる．

† NEC システムプラットフォーム研究所
NEC System Platforms Research Laboratories
†† NEC ユビキタス基盤開発本部
NEC Ubiquitous Platform Development Division
☆ 現在，兵庫県立大学大学院
Presently with Graduate School of Applied Informatics, University of Hyogo

● 携帯端末でもサーバ機能が必要とされている．こうしたサービスを実現するためには，DOS 攻撃など外部攻撃に対する保護強化が必要である．

● ネイティブプログラムダウンロードサービスを実現するためには，ダウンロードされたプログラムの悪意ある動作，あるいは過失による誤動作から，携帯端末の端末システムを保護し，安定動作させることが必要である．

そこで携帯端末用 Linux のセキュリティ機能として，厳密なアクセス制御とリソース制御の仕組みが求められている．

アクセス制御とは，システム上のプロセスがファイルなどのリソースにアクセスを行う際，そのアクセスが適切なものであるかを検証する機能である[1)]．たとえばセキュリティが強化された Linux では，アクセスの検証をアクセスポリシに基づいて行うため，ポリシに記述されていないアクセスの実行は許可されない（たとえば，文献 2)）．これにより，ネットワークを介して携帯端末にダウンロードされたプログラムに対して，システムファイルを修正する権限を与えないようアクセスポリシを記述することができ，そのプログラムによるシステムファイルの改竄，破壊を防止できる．また，ルートユーザが持つすべてのファイルに書き込める権限も制限することができる．さらに，カーネル以外に X ウィンドウシステムにもアクセス制御の機能強化が行われている[3)]．

このようにアクセス制御に関しては多くの研究開発がなされている一方，携帯端末ではハードディスクがない，搭載されるメモリ量も PC に比べてはるかに小さい状況にある．メモリをはじめとするリソースの過度な使用は，電話としての安定動作を損なうことになるため，これを管理するリソース管理機能が求められている．そこで本論文では，携帯端末用 Linux において端末内リソースの使用量管理を実現するための R2ELinux（Resource Reservation Enhanced Linux）について述べる．R2ELinux は，Linux 上の各プロセスが使用するカーネル管理下のリソース量，ユーザ空間のリソース量，ファイル使用量を Linux カーネル内で管理することで，ユーザプログラムに何ら修正を加えることなく，Linux のリソース管理を強化することが可能である．本論文の構成は以下のとおりである．2 章では携帯端末用 Linux のリソース制御に対する要件をまとめる．3 章ではカーネルで管理するリソース制御の仕様と設計を，4 章ではサーバで管理するリソース制御の仕様と設計を，それぞれ述べる．5 章では，前章で述べた実装とその評価を，6 章では R2ELinux の適用事例を，7 章でまとめを述べる．

2. 携帯端末用 Linux のリソース制御に対する要件

携帯端末のメニューから起動される機能をアプリケーションと呼ぶことにする．メニューからアプリケーションが選択されると，プログラムを起動してプロセスを生成する，あるいはすでに起動されているプロセスを起こす．これらのプロセスは必要に応じて他のプロセスと通信して，アプリケーションの機能を実現する．このとき，携帯端末は PC やワークステーションと比較しリソースが大きく制限されており，1 つのプロセスが大量のリソースを使用することはシステム全体の安定性を損なうことにつながる．リソースを消費するケースとして次の場合が考えられる．

C1： プログラムのバグによる使用リソース量の上限チェックもれにより，もしくはメモリリークにより携帯端末内リソースを不用意に使い尽くす．

C2： 悪意のあるプログラムが携帯端末内リソースを使い尽くす．

ここでは **C1** に対するリソース制御機能を考える．

上述のような Linux 内でリソース制御を行うためには以下の 4 つの機能が必要である．

(1) カーネルリソースの管理

カーネルリソースとは，CPU 時間やヒープ，スタックなど，カーネル内で管理される様々なリソースのことをいう．あるプログラムの実行によりこれらのリソースを消費し尽くすと他のプログラムの実行に支障をきたす．プログラム実行にあたってこうしたリソース消費量の上限を設けることにより，システムの安定性を保証することが必要である．

(2) ファイル使用量の管理

悪意のあるプロセスがファイルシステム内に大きなファイルを作成すると，携帯端末の Linux システム全体の安定性に悪影響を及ぼす．携帯端末はハードディスクを備えておらず，代わりに RAM ディスクを使用するなど，携帯端末は PC と比較しリソース量に大きな制約があるため，この問題はシステムの安定動作に大きな影響を及ぼしかねない．Linux ではユーザ単位でファイルの使用量の上限を制限するクオータの機能があるが，携帯端末ソフトウェアにおいては，むしろプログラム単位でファイルの使用量を管理，制限する必要がある．

(3) 汎用リソースの管理

Linux のような高機能を搭載した携帯端末では，サーバにより今後多様な機能が実現されると考えられる[4)]．

たとえば，X サーバはその例である．こうしたサーバを利用して，クライアントの要求をサーバで実現するときに，その実行によりサーバ内のメモリが消費され，他のクライアントの実行に支障をきたす場合がある．このような状況を回避するため，汎用リソースという概念を導入し，これをカーネルリソース同様に管理することを考えた．汎用リソースとは，カーネル以外で量的に管理すべきユーザ空間のリソースのことである．メモリリソースに関しては，カーネルリソースではプロセス単位で管理されるのに対して，汎用リソースではカーネルリソースで与えられたメモリリソースをさらに小さい単位で管理しようとするものであり，カーネルリソースより管理粒度を細かくすることができる．X サーバで確保・解放される GUI リソースなどのメモリリソースは汎用リソースの例である．

(4) 既存のアクセス強化 Linux の組み込みが容易な構成

上述したリソース制御機能は，アクセス制御で許可され操作に対して量的制限を与えるものであり，アクセス制御と一体となってセキュリティ機能を実現することになる．Linux では，MAC（Mandatory Access Control）や RBAC（Role-Based Access Control）などのアクセス制御機能の強化が行われており，SELinux, PitBull，LIDS といった例がある[2),5)~10)] また最近の Linux 用セキュア OS は LSM（Linux Security Modules)[11)] に準拠したものが多い．LSM とは，Linux カーネルからセキュリティ機能を持ったモジュールへのフック関数群である．従来セキュア OS を Linux カーネルに適用する場合，カーネル内のセキュリティチェック機能をフックするために，カーネルに対して別途パッチを当てる必要があった．しかし Linux カーネル 2.6 からはカーネル内に標準実装されている，この LSM のフレームワークを使用することで，Linux カーネルにセキュリティ機能を組み込むことが容易になっている．

上述した要求事項を満たすべく Linux と X ウィンドウシステムを基にした携帯端末向けソフトウェアプラットフォームのリソース制御を強化した R2ELinux を開発した．

R2ELinux の機能仕様を設計するにあたって，以下のような設計方針をたてた．

P1： 実現にあたって，可能な限り既存の機能を利用すること．特に，アクセス制御機能に関してはすでに述べたように多くの既存のセキュア Linux が存在するのでこれを利用できるようにする．

P2： リソース制御機能の実現にあたって，リソース量を制限する機能とその機能をどう利用するかを司るポリシとは分離すること．これにより，柔軟にリソース量が制御できるようにする．

P3： リソース制御機能を利用する際に，プログラムは変更しなくてもいいようにすること．携帯端末用 Linux には，Web ブラウザやメーラなど既存のプログラムがすでに数多く存在しており，これらのソースコードに改造を加えるのは困難なためである．

なお，R2ELinux において組み込み可能としたアクセス制御の実装と評価については，付録 A.1 で述べる．

表 1 setrlimit で管理可能なリソース
Table 1 Resources managed in setrlimit.

- 使用できる CPU 時間の上限（RLIMIT_CPU）
- 使用できるファイルサイズの最大値（RLIMIT_FSIZE）
- 使用できるヒープサイズの最大値（RLIMIT_DATA）
- 使用できるスタックサイズの最大値（RLIMIT_STACK）
- 同時に生成できる（RLIMIT_NPROC）
- プロセス空間の最大値（RLIMIT_AS）
- コアダンプ時に生成されるコアファイルの最大値（RLIMIT_CORE）
- 所有可能なページフレーム数（RLIMIT_RSS）
- オープンできるファイルの最大数（RLIMIT_NOFILE）
- スワップ対象外にするメモリの最大値（RLIMIT_MEMLOCK）
- 使用できるファイルロック数の最大値（RLIMIT_LOCKS）

3. カーネルリソース制御

3.1 機能仕様

(1) カーネルリソースの管理

Linux ではカーネルリソース制御を行うための仕組みとして，setrlimit というシステムコールが用意されている．このシステムコールは，呼び出し元のプロセスに対して，**表 1** のようなカーネルリソースの使用量を設定することができる．R2ELinux のリソース制御対象は，方針 **P1** に基づき，プロセスごとに使用できるスタックサイズやオープンできるファイルの数などのリソース制御に関して，setrlimit システムコールを利用した．このとき，以下のような機能の追加，変更を行った．

- メモリの消費を防ぐためにコアダンプが発生しないようにした．このために，RLIMIT_CORE の値は 0 とした．これ以外の項目の値は次に述べるポリシファイルで規定される．
- Linux ではユーザ単位でプロセス生成個数を制限することができる．しかし，携帯端末ではアプリケーションから起動されるプログラムが管理主体であり，プログラム単位でこれを制限する必要がある．そこで，R2ELinux ではシステム内の各プログラムにそれぞれ異なるユーザ ID を割り振る

ことによって，各プログラムで生成できるプロセス数を RLIMIT_NPROC で制限できるようにした．

- 表 1 の管理項目に加えて，最大同時実行プログラム数を追加した．最大同時実行プログラム数とは，同一プログラムが同時に起動できる最大個数であり，プログラム単位に定義できる．これはあらかじめ決められた数以上のプログラムが起動され，メモリなどのリソースが消費されるのを防ぐために導入した．通常の携帯端末の運用では，各プログラム 1 個を想定している．

なお，現在の実装では同一プログラムが多重起動された場合，プロセスが fork した場合には同一のリソース量が設定される．

(2) ファイル使用量の管理

Linux ではクォータの形でユーザ単位に，ファイルの使用量の上限を設定できる．2 章で述べたように携帯端末では，むしろプログラム単位でファイルの使用量を管理，制限する必要がある．上述したように，R2ELinux ではシステム内の各プログラムにそれぞれ異なるユーザ ID を割り振っているので，各プロセスが使用するファイル使用量をクォータの仕組みで管理することができる．これにより，プログラムがファイルシステム上で，あらかじめ決められた使用量を超えてディスク書き込みを行うことを防止できる．使用できるファイル使用量は，Linux システム内のファイルシステム単位にリソース制御ポリシファイルに記述する．

(3) リソース制御ポリシファイル

方針 **P2** に基づき，カーネルリソース，汎用リソース，ファイル使用量の最大値を記述するリソース制御ポリシファイル（以下，ポリシファイルと略する）と呼ばれるファイルを設けた．ポリシファイルには，プログラムごとにカーネルリソースやファイル使用量，汎用リソースの最大使用量が記述されており，各プログラムのプロセスは，ここに記述された使用量を超えてそれらリソースを使用することはできない．ポリシファイル内に記述されているリソースの項目は以下のとおりである．

- プログラム実行時に設定するユーザ ID
- プログラムの実行ファイルへの絶対パス
- （1）で示したカーネルリソースの各項目
- クォータ管理対象のファイルシステムの数
- クォータ管理のファイルシステムと利用できる最大ファイル使用量

上述の各項目は CSV 形式で記述し，使用量を指定しない項目については空欄とする．

表 2 R2ELinux で利用する LSM フック関数
Table 2 The LSM fook functions used in R2ELinux.

LSM フック関数名	機能
bprm_check_security	execve 実行前に呼ばれる．
task_free_security	プロセスが終了したときに呼ばれる．
sb_post_mountroot	ルートファイルシステムがマウントされた後に呼ばれる．
sb_post_addmount	ルートファイルシステム以外のファイルシステムがマウントされた後に呼ばれる．
sb_umount	ファイルシステムがアンマウントされる前に呼ばれる．
sys_security	R2ELinux システムコールを呼び出す．

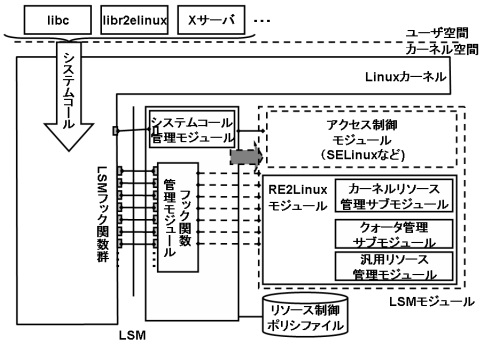

図 1 R2ELinux アーキテクチャ
Fig. 1 R2ELinux architecture.

3.2 設　　計

方針 **P1** に基づき，リソース制御機能は LSM（Linux Security Modules）[12] を利用することにした．LSM を基本にすることによって，SELinux などの既存のアクセス制御用 LSM モジュールとリソース制御モジュールを登録し，アクセス制御とリソース制御の両方を実現することができる．R2ELinux で使用する LSM フック関数を**表 2** に示す．

3.1 節で述べた機能を実現するために，R2ELinux は**図 1** のようなアーキテクチャで構成される．R2ELinux はリソース管理を行う LSM モジュールとして実現され，以下のサブモジュールから構成される．

- フック関数管理サブモジュール：LSM フック関数群を R2ELinux で実装したものである．
- カーネルリソース管理サブモジュール：プログラムごとに使用できるカーネルリソースの量的制御を行う．このために，プログラムの絶対パスと最大リソース量を規定したカーネルリソース管理テーブルを持つ．
- クォータ管理サブモジュール：プログラムごとに使用できるファイル使用量を管理する．
- 汎用リソース管理モジュール：汎用リソースを管

理する．詳細は 4 章で述べる．

以下に R2ELinux の振舞いを述べる．

- Linux カーネルが初期化されるとき

R2ELinux モジュールと SELinux などのアクセス制御モジュールを LSM モジュールとして登録する．

- ルートファイルシステムがマウントされるとき

ルートファイルシステムがマウントされると，LSM フック関数 `sb_post_rootmount` が呼び出され，以下の処理を行う．

(1) ルートファイルシステム上に存在するポリシファイルのロードを行う．

(2) カーネルリソース管理テーブルで管理されているプログラム実行ファイルへの絶対パスをもとに，その実行ファイルがルートファイルシステム上に存在している場合，そのファイルを識別するためにファイルシステムのデバイス番号とファイルの inode 番号を求め，テーブル内の項目として格納する．

(3) クォータ管理サブモジュールを呼び出し，マウントされたファイルシステムに対して，クォータの設定を行う．

- 他のファイルシステムがマウントされるとき

ファイルシステムがマウントされるたびに，LSM フック関数 `sb_post_addmount` が呼ばれ，ルートファイルシステムと同様の処理が行われる．

- プログラムが起動されるとき

`execve` システムコールによりプログラムが実行される前に，LSM フック関数 `bprm_check_security` によってフック関数管理モジュールを通じてカーネルリソース管理モジュールが呼ばれる．このとき，以下のことを行う．i) 最大同時実行プログラム数の上限チェックを行い，上限以下であれば現在実行中のプログラム数の値を 1 増やす．ii) 実行されるファイルのデバイス番号と i ノード番号を使って，このファイルのエントリがカーネルリソース管理テーブル中に存在するか調べる．存在すれば，そのエントリのカーネルリソース使用量を，`setrlimit` システムコールの実装関数である `sys_setrlimit()` を呼びプロセスに設定する．これにより，3.1 節の機能を実現すると同時に，プログラムの改造は不要となるため方針 **P3** が実現されることになる．iii) 実行するプログラムのユーザ ID をポリシファイルで定義したものに変更する．

- プログラムが終了するとき

`exit` システムコールによりプログラムが終了する際，LSM フック関数 `task_free_security` によってカーネルリソース管理モジュールを呼び，現在実行中のプログラム数の値を 1 減らす．

4. 汎用リソースの管理

4.1 機能仕様

サーバ系プログラムは，そのサーバプログラム自体で使用可能なリソース量ということよりも，そのサーバを使用するクライアントごとにリソース使用量を管理したいという要求がある．なぜなら，サーバにおいてあるクライアントがリソースを過度に消費した場合に他のクライアントの処理に支障をきたすからである．Linux を搭載した携帯端末では X サーバがこの例である．

そこで，R2ELinux では汎用リソース管理という機能を設けた．汎用リソース管理では以下のパラメータを指定してリソースの上限管理を行う．

- クライアント識別子：サーバから見るとサーバに要求を出すクライアントにはプロセス，スレッドなど多様である．また，プログラム，サービスのような抽象的な要求主体もクライアントととらえる場合もある．クライアント識別子はこれらを抽象化した識別子である．
- リソース型：クライアントが消費するある論理的なまとまり．

たとえば，X サーバにおいては，クライアントはプロセス識別子で識別され，リソースは X サーバ内で消費されるメモリが対応する．

これに基づき R2ELinux では，汎用リソース確保用のシステムコール `r2elinux_gralloc` とリソース解放用のシステムコール `r2elinux_grfree` を用意した（**図 2** 参照）．チェック対象のリソースの最大使用量をポリシファイルでタイプごとに規定する．汎用リソースのシステムコールのパラメータでチェック対象のリソースのタイプを指定する．プログラムから上述した

```
int r2elinux_gralloc(unsigned int cid, unsigned int type,
    size_t size)
/* 識別子が CID のクライアントから，type で指定されたリソースにおいて，これまでに確保したリソース量と size で指定されたサイズの合計が，ポリシファイルで指定された使用量を超えてないかチェックする．超えていなければ R2ELINUX_TRUE を返却し，超えていれば R2ELINUX_FALSE を返却する．*/

int r2elinux_grfree(unsigned int cid, unsigned int type,
    size_t size)
/* 識別子が CID のクライアントから，type で指定されたリソースにおいて，これまでに確保したリソース量と size で指定されたサイズの差が，0 以上かどうかをチェックする．超えていなければ R2ELINUX_TRUE を返却し，超えていれば R2ELINUX_FALSE を返却する．*/
```

図 2 汎用リソース管理システムコールの API 仕様
Fig. 2 API specification of generic resource management system call.

汎用リソースのシステムコールが呼ばれるたびに，汎用リソース管理テーブルでそのプログラムが現在使用している汎用リソース量の値を変更し，かつその値が汎用リソースの最大使用量を超えていないかをチェックする．もし超えている場合は，プログラムに対してエラーを返却する．

汎用リソース管理機能は，1つのプロセスにおいてサービスごとのリソース上限量を制御する機能を提供する．さらに，複数のプロセスで1つのサービスを実現する場合にその実現に必要なリソースを全体として制限するような場合にも利用できる．

汎用リソース制御を行うためにポリシファイルに以下の項目を追加した．

- 制限対象の汎用リソース数
- 汎用リソースの種別とその最大使用量

4.2 設　　計

ここでは，携帯端末向けXウィンドウシステムにおける汎用リソース管理機能の実現について述べる．

Xサーバでは，`malloc()` をはじめとして21種類のメモリを動的確保する関数が，`free()` をはじめとして3種類のメモリ解放関数が存在する．これらの関数を調べ以下のように分類した．

M1： Xクライアントごとに確保されるメモリ．

M2： Xクライアントには依存せずに確保されるメモリのうち，一時的に取られてすぐに解放されるメモリ．具体的にはXクライアントからXサーバのAPIが呼ばれてXクライアントに戻るまでに確保，解放がなされるメモリ領域で，Xサーバ用のメモリ領域として管理される．

M3： 上のいずれでもないもの：Xクライアントには依存せずに確保されるメモリのうち永続的に確保されたままのメモリ領域．Xサーバ用のメモリ領域として管理される．

Xサーバが起動するときに，**M1** に属するメモリ確保関数の中でXクライアントのプロセス識別子を引数として（`r2elinux_gralloc`）を呼んで，この関数の確保するリソースの上限をチェックし，**M2**，**M3** に属するメモリ確保関数の中でXサーバのプロセス識別子を引数として（`r2elinux_gralloc`）を呼んで，この関数の確保するリソースの上限のチェックを行う．メモリ解放関数の中で（`r2elinux_grfree`）を呼びプロセスの消費する当該リソースを減らす．プログラムが終了するときに，汎用リソース管理テーブルの該当するリソースデータを消去する．

Xウィンドウシステムの汎用リソース管理では，クライアント識別子としてXクライアントのプロセス識別子を利用した．Xクライアントのプロセス識別子は，`XOpenDisplay` 内でXサーバに接続するときに送付され，Xクライアント管理情報の一部としてXサーバ側で保持される．Xクライアント管理情報はプロセス識別子とリソース上限値を保持し，プロセス識別子によりキャッシュ制御される．これは，同じXクライアントからXリクエストを連続して発行することが多く，Xクライアント管理情報をプロセス識別子によってキャッシュ制御することで，リソースの上限チェックが高速になると期待できるからである．

汎用リソース管理機能は，Xサーバだけでなく，他のサーバでも適用可能である．たとえば，ESD（The Enlightened Sound Daemon）☆があげられる．これは，複数のプロセスからのサウンド鳴動要求をミキシングして音を鳴らすものであり，サウンドデータを蓄積しておき，要求により再生する機能がある．この機能を利用する場合，1つのクライアントが多くのサウンドデータを蓄積してしまうと，メモリ不足で他のクライアント向けのサービスができなくなる．このように，1つのサーバプロセスが，他のクライアントプロセスのために，メモリを消費するようなモデルでのクライアント別リソース管理に汎用リソース管理は有効である．また，汎用リソース管理は，メモリだけではなく，クォータ機能のないRAMファイルシステムのプロセス別利用容量制限などにも有効である．

汎用リソース管理の利用方法を述べる．サーバプログラムでこれを利用する場合はXサーバと同様である．サーバプログラム以外で汎用リソース管理機能を利用する場合は，メモリ確保関数やメモリ解放関数を改造して，呼び出し側のプロセス識別子を付与し `r2elinux_gralloc`，`r2elinux_grfree` を呼び出して，リソース量の上限チェックを行うことになる．

5. 実装と評価

R2ELinux の妥当性と性能を評価するために，R2ELinux を**表3**の環境で実装し，評価を行った．実装に要したソース行の概数は**表4**に示すとおりである．

5.1 評 価 内 容

以下の3つの内容に関して評価を行った．

(1)　プログラムのメモリ使用量比較

メモリを確保し続けるプログラムを．通常のLinuxとR2ELinux上で実行し，R2ELinuxでは一定量以上のメモリが確保されないことを確認する．なおR2ELinux上で，プログラムが使用できるメモリの上限はここで

☆ http://www.tux.org/ ricdude/EsounD.html

表 3 実装環境の仕様
Table 3 Specification of implementation environment.

hardware	iPAQ H3660 (CPU: Strong ARM 206 MHz, RAM: 64 MB, ROM: 32 MB)
OS	Linux 2.4.19

表 4 R2ELinux の実装規模概数
Table 4 Implementation size of R2ELinux.

カーネル部分		5,050 行
	フック関数管理	1,600 行
	カーネルリソース管理	900 行
	汎用リソース管理	1,100 行
	クォータリソース管理	450 行
	その他（ヘッダファイル等）	1,000 行
ユーザ空間部分		200 行
R2ELinux 全体		5,250 行

は 60 MB としている．

(2) プログラムの実行時間比較（カーネルリソース管理）

`sys_setrlimit()` によるカーネルリソースの上限チェックは通常の Linux でも行われており，これによるシステムコール実行時のオーバヘッドの増加はない．R2ELinux では，3 章で述べたように，(i) `exec` 時の各種リソース量の上限設定，(ii) ルートファイルシステムマウント時のポリシファイルのロード，を行うのでこれらのオーバヘッドを測定する．(i) によるオーバヘッドの増加を調べるために，通常の Linux と R2ELinux において 10 万回 fork，exec を繰り返すプログラムを実行する．(ii) については，ルートファイルシステムマウント時に呼ばれる LSM フック関数 `sb_post_rootmount` を 5,000 回実行して，その実行時間を 10 個のプログラム（リソース制御ポリシが 10 個）がある場合と 100 個のプログラム（同 100 個）がある場合とで測定した．

(3) プログラムの実行時間比較（汎用リソース管理）

X サーバでは汎用リソース管理は `malloc()` などのメモリ確保関数実行時，`free()` などのメモリ確保関数実行時に汎用リソース管理が実行される．汎用リソース管理を利用する場合，利用しない場合において，メモリの確保，解放を 10 万回行うプログラムを実行し，そのオーバヘッドを測定する．このとき，X クライアントのプロセス識別子による X クライアント管理情報のキャッシュ制御の有効性を検証する．

(4) カーネルの起動時間比較

通常の Linux と R2ELinux の起動時間を比較する．携帯端末はカーネルの起動が頻繁に起こるため，起動時間の短縮は重要な課題である．

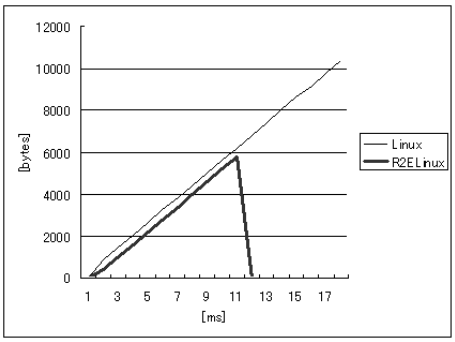

図 3 プログラムのメモリ使用量
Fig. 3 Amount of memory consumption in application.

表 5 プログラムの実行時間の比較（`exec` 時）
Table 5 Comparison of application execution time (in a case of executing `exec`).

	実行時間（全体）	実行時間（1 回あたり）	増加率
通常の Linux	704 s	7.04 ms	—
R2ELinux	717s	7.17 ms	1.8%

表 6 プログラムの実行時間の比較（ファイルシステムマウント時）
Table 6 Comparison of application execution time (in a case of mounting file system).

	実行時間（全体）	実行時間（1 回あたり）
通常の Linux	0	0
R2ELinux（10 プログラム）	4 s	0.8 ms
R2ELinux（100 プログラム）	29 s	5.8 ms

5.2 評価結果と考察

(1) プログラムのメモリ使用量比較

図 3 のとおり，R2ELinux を使用する場合，12 ミリ秒でプログラムがメモリの確保に失敗していることが分かる．このように R2ELinux では，プログラムが使用できる最大使用量をポリシファイルに記述することができ，これにより不正なプログラムが大量のメモリを確保することを防ぐことができる．またメモリ使用量が 60 MB に達する前に収束しているが，これは明示的に確保したメモリ領域以外に glibc をメモリにマッピングするために，メモリ領域が使用されているためである．

(2) プログラムの実行時間比較（カーネルリソース管理）

(i) `exec` 時の各種リソース量の上限設定，(ii) ルートファイルシステムマウント時のポリシファイルのロードのオーバヘッドは**表 5**，**表 6** にそれぞれ示すとおりである．`exec` の 1 回あたりの実行時間の増分は 130 マイクロ秒である．また，ルートファイルシステムマ

表 7 プログラムの実行時間の比較（汎用リソース管理）
Table 7 Comparison of application execution time.

	実行時間（全体）	実行時間（1 回あたり）	増加率
通常の Linux	14 s	1.4 s	—
R2ELinux（キャッシュあり）	50 s	5.0 s	3.6 倍
R2ELinux（キャッシュなし）	106 s	10.6 s	7.6 倍

表 8 カーネルの起動時間の比較
Table 8 Comparison of kernel bootup time.

	実行時間	増加率
通常の Linux	7.466 s	—
R2ELinux	7.546 s	1.1%

ウント時のオーバヘッドは，ポリシファイル中のルール数に比例して増加するが，ルートファイルシステムのマウントはシステム立ち上がりに 1 回だけ行われるだけである．これらから両者とも実用上問題はないと思われる．

(3) プログラムの実行時間比較（汎用リソース管理）
表 7 に示すように，メモリの確保・解放の 1 回あたりの実行時間は汎用リソース管理が行われた場合，ない場合に比べて 3.6 倍もの時間がかかっている．しかし，X サーバの実行においてメモリの確保・解放はごくわずかの時間しか占めないと考えられるため実用上問題ないと考えられる．また，メモリの確保・解放の 1 回あたりの実行時間はキャッシュ制御により 5.0 マイクロ秒となり，これがない場合は 10.6 マイクロ秒であることから，X クライアント情報のキャッシュ制御が汎用リソース管理の実装では有効であることを示している．

(4) カーネルの起動時間比較
表 8 のとおり，R2ELinux を使用することによる増加率は 1.1%となった．R2ELinux がカーネル起動時の大部分はポリシファイルのロードであり，その記述量によって起動時間に与えるインパクトは異なる．よって，携帯端末においては，適切にカスタマイズされたポリシファイルの使用が不可欠である．

6. 応　　用

R2ELinux の携帯端末製品への適用シナリオを説明する．適用形態の 1 つとして，携帯端末へのダウンロードなどによるプログラム追加サービスが考えられる．ここでは信頼できるベンダが提供するプログラムのみを対象とする．追加されるプログラムには悪意はないが，使用するリソース量の制限は行われておらず，またバグによりメモリリーク，GUI リソースリーク，スタックの使いすぎなどが生じる可能性がある．これらにより，携帯電話端末全体の動作に悪影響を与える可能性がある．R2ELinux を使って，外部から提供されたプログラムに対して，適切なリソース制限をかければ，上述の理由により携帯端末システム全体がおかしくなることを防ぐことができる．ただし，追加されたプログラムがサーバプログラムの場合に，クライアントの要求をスレッドで処理をする場合がある．現在の R2ELinux ではスレッド単位でスタックの上限チェックができないという制限がある．なお，プログラム追加サービスでは，どのようなプログラムが提供されるか事前には分からないので，ドメインと呼ばれるプログラムカテゴリを導入し，ポリシファイルには，ドメインごとに利用できるリソース量を記述することになる．

7. おわりに

本論文では，携帯端末向け Linux においてリソース制御を実現するための方式に述べた．本方式では，Linux 上の各プロセスが使用するカーネル管理下のリソース量，ユーザ空間のリソース量，ファイル使用量を Linux カーネル内で管理することで，ユーザプログラムに何ら修正を加えることなく，Linux のリソース管理を強化することが可能である．

携帯端末向け Linux におけるリソース制御の課題として以下が残されていると考えられる．まず，本方式では実現できていないリソース制御の実現があげられる．これには，本文で述べた，プログラム多重起動時に異なるリソース量を設定する方法，スレッドスタックをはじめとするその他のカーネルリソースの制御に加えて，単位時間あたりのリソース使用量（CPU や通信データ量など）の制御がある．特に，単位時間あたりのリソース制御の実現にはカーネル内バッファなどを管理する機能の追加，リソースカーネル[13] などの一定周期でリソース管理機能を呼び出すなどの機構が必要である．

現在の R2ELinux の実装は，プログラム間で事前にリソース量を決め，携帯端末稼動中は変更しないものとしている．現在の携帯端末の利用形態ではこれで十分であると考えられる．しかし，今後利用状況に応じて QoS 制御などのために動的に携帯端末稼動中にプログラム間のリソース量を変更・調停することが考えられる．この場合，必要なリソース量を管理するリソースマネジャプログラムが必要になる．各プログラムがリソースマネジャに必要リソース量を申請し，リソースマネジャがこれを調整し，R2ELinux 機能により調停された値をカーネルに設定し，管理する．この

場合の調整方法として，原田らの提案している QoS レベルの平均による動的リソース制御方式[14] の適用が考えられる．

第 3 にリソース管理機能の適用対象として本論文では対象外とした悪意のあるプログラムへの適用が考えられる．これには R2ELinux 機能だけでは十分でなく，Linux カーネル本体でのメモリ管理部の改造，SELinux 機能との連携が必要となる．

最後に，リソース制御だけでなく，現在の Linux のアクセス制御は複雑なポリシ記述を必要としており，ユーザがこれを正確に記述するのは難しいという問題がある．このようなポリシ記述の簡単化，あるいはポリシ記述をユーザは知らなくてもいいような運用環境が求められる．

参 考 文 献

1) Silbeschatz, A., Galvin, P. and Gagne, G.: *Operating System Concepts*, 6th edition, John Wiley & Sons, Inc. (2002).
2) Loscocco, P. and Smalley, S.: Integrating Flexible Support for Security policies into Linux Operating Systems, *Proc. 2001 USENIX Annual Technical Conference* (2001).
3) Kilpatrick, D., Salamon, W. and Vance, C.: Securing the X Window System with SELinux, Technical report, NSA (2003).
4) 太田 賢，吉川 貴，中川智尋，倉掛正治：携帯機用モバイルサーバフレームワークの提案，情報処理学会技術報告 OS, Vol.2003, No.42, pp.127–134 (2003).
5) Argus Systems: PitBull Linux. http://www.argus-systems.com/product/overview/lx/
6) Openwall Project: OWL. http://www.openwall.com/Owl/
7) LIDS Project: LIDS. http://www.lids.org/
8) SkLUG: Medusa DS9. http://medusa.fornax.sk/
9) RSBAC Project: RSBAC. http://www.rsbac.de/
10) 日本総研：LBSM. http://www.jri.co.jp/contents/press/report/jri-press030319.pdf
11) LSM Project: LSM. http://lsm.immunix.org/
12) Wright, C., Cowan, C., Smalley, S., Morris, J. and Kroah-Hartman, G.: Linux Security Module Framework, *Proc. Ottawa Linux Symposium*, pp.604–617 (2002).
13) Oikawa, S. and Rajkumar, R.: Portable RK: A Portable Resource Kernel for Guaranteed and Enforced Timing Behavior, *Proc. 5th IEEE Real-Time Technology and Applications Symposium*, pp.111–120 (1999).

表 9 SELinux 組み込み時のプログラムの実行時間の比較
Table 9 Comparison of application execution time in SELinux.

	実行時間（全体）	実行時間（1 回あたり）	増加率
通常の Linux	310 s	3.10 ms	—
SELinux	316 s	3.16 ms	1.9%

表 10 SELinux 組み込み時のカーネルの起動時間の比較
Table 10 Comparison of kernel bootup time in SELinux.

	実行時間	増加率
通常の Linux	9.1 s	—
R2ELinux	9.5 s	4.4%

14) 原田史子，潮 俊光，中本幸一：QoS レベル公平化に基づくリアルタイムシステムの QoS 適応制御，電子情報通信学会論文誌，Vol.J87-D-I, No.3, pp.364–371 (2004).

付 録

A.1 アクセス制御の実装と評価

R2ELinux のリソース制御機能を補完するアクセス制御機能は，SELinux（Linux Kernel 2.4.19 用パッチ）は用いて表 3 の環境で実装した．SELinux ではカーネル起動時にアクセス制御ポリシファイルをロードし，システムコール実行時にアクセス権のチェックを行う．このため，(i) プログラム実行時間と (ii) カーネルの起動時間の比較を行うことにした．アクセス制御ために用意したポリシファイルは約 37200 行である．(i) は `open`，`write`，`read`，`close` を 10 万回繰り返すプログラムを，SELinux を組み込んである場合，組み込んでいない場合で実行した．(ii) はポリシファイルを読み込んで起動する時間を測定した．その結果を，**表 9**，**表 10** に示す．いずれも数%程度の増分であり実用に耐えうると考える．

(平成 16 年 5 月 18 日受付)
(平成 16 年 10 月 19 日採録)

稗田 諭士（正会員）

2001 年早稲田大学大学院理工学研究科修士課程修了．同年 NEC 入社．現在システムプラットフォーム研究所所属．組み込みソフトウェアの研究開発に従事．

才田　好則

1993 年九州大学大学院工学研究科電子工学専攻修士課程修了．同年 NEC 入社．現在システムプラットフォーム研究所所属．組み込みソフトウェアの研究開発に従事．電子情報通信学会会員．

中山　義孝

1992 年広島工業大学機械工学科課程卒業．同年 NEC ホームエレクトロニクスに入社．現在 NEC ソリューション開発研究本部所属．組み込みミドルウェアの研究開発に従事．

千嶋　博

1990 年法政大学工学部電気電子専攻学士課程修了．同年 NEC 入社．現在システムプラットフォーム研究所所属．組み込みソフトウェアの研究開発に従事．

中本　幸一（正会員）

1982 年大阪大学大学院基礎工学研究科前期課程修了．同年 NEC 入社．組み込みソフトウェア，モバイルシステムソフトウェアの研究開発に従事．1990〜1991 年 Cornell 大学計算機科学科客員研究員．1997 年大阪大学大学院情報数理系専攻博士課程入学．2000 年単位取得退学．博士（工学）．2003〜2004 年電気通信大学客員教授．2004 年より現職．電子情報通信学会，日本ソフトウェア科学会，IEEE Computer Society，ACM 各会員．