情報処理学会論文誌　コンシューマ・デバイス & システム　Vol.5　No.1　9–19 (Feb. 2015)

コンシューマ・デバイス論文

Androidにおける LRUを用いた終了プロセスの選定

野村 駿[1,a)]　中村 優太[1]　坂本 寛和[1]　山口 実靖[1]

受付日 2014年4月24日, 採録日 2014年10月15日

概要：近年，Android は様々なデバイスで採用されており，普及率も高く重要性が高まっている．Android には，low memory killer と呼ばれる独自のプロセスメモリ管理システムが搭載されており，メモリ不足時には既定の基準に従いプロセスを強制終了し空きメモリを確保する．このプロセスの強制終了により，再度同じアプリケーションを使用する場合にプロセスの再起動の時間が必要となり，ユーザの利便性を低下させることがある．本研究では，LRU に基づく強制終了プロセス選定手法，アプリケーションの起動時間に基づく手法，メモリ使用量に基づく手法，およびこれらを組み合わせた手法を提案する．Android 標準手法とこれら提案手法を，ユーザから許可を得て取得した実際のアプリケーションの起動履歴を用いて評価をしたところ，すべての提案手法においてアプリケーションの合計起動時間を標準手法よりも低減させることができることが確認され，提案手法の有効性が確認された．

キーワード：Android，low memory killer

LRU Based Memory Termination in Android Low Memory Killer

Shun Nomura[1,a)]　Yuta Nakamura[1]　Hirokazu Sakamoto[1]　Saneyasu Yamaguchi[1]

Received: April 24, 2014, Accepted: October 15, 2014

Abstract: Android has become one of the most important platforms in the current computer systems. Android has its original process memory management system, which is called low memory killer. When amount of free memory is less than a threshold, it selects processes based on its own policy and terminates them for allocating free memory. This process termination sometime declines user convenience. That is, re-launching of a process is required for re-using an application if its process has been terminated. Thus, selecting policy for termination can be considered important. In this paper, we have proposed several selecting methods for process termination. One is based on LRU, one is based on application launching time, and the other is based on memory size. We have implemented these proposed methods and evaluate them with practical application launching log. Our experimental results have demonstrated that our proposed method have been able to decrease application launching time.

Keywords: Android, low memory killer

1. はじめに

Android はスマートフォン，タブレット PC，音楽プレイヤなど様々なデバイスの OS として採用されており，2013 年第 2 四半期においてそのシェアは 81.3% [1] となっており，今も増加中である．このように Android は広く普及しており，重要性が高まっていると考えられる．

Android には，low memory killer と呼ばれるプロセスメモリ管理システムが搭載されており，独自のルールに従ってメモリの管理を行っている．low memory killer は空きメモリ量が少なくなると起動され，メモリの空き容量を確保するためにプロセスを強制終了する．このプロセスの強制終了により，ユーザが再度同じアプリケーションを使用する場合に，プロセスの再起動が必要となりユーザの待ち時間を増加させることがある．

本研究では，強制終了するプロセスの選定の改善によりプロセス再起動にともなうユーザの待ち時間を低減させる

[1] 工学院大学大学院電気電子工学専攻
Electrical Engineering and Electronics, Kogakuin University Graduate School, Shinjuku, Tokyo 163–8677, Japan
[a)] frelia.enja@gmail.com

© 2015 Information Processing Society of Japan

ことを目的として，新しい選定手法を提案した．そして提案手法に基づいて改修した Android をスマートフォンに実装し，評価を行った．

2. Android

2.1 アーキテクチャ

Android はアプリケーション，アプリケーションフレームワーク，ライブラリ，Android ランタイム，Linux カーネルで構成されている [2]．

Android はベースは Linux を採用しているが，モバイル端末特有のハードウェア制約のため，独自の修正をしている．アプリケーション，アプリケーションフレームワーク，ライブラリ，Android ランタイムには Android のために新規に開発された実装が多く含まれている．たとえば，標準 C ライブラリとしては GNU libc ではなく bionic を用いており，アプリケーションは基本的に Android アプリケーションフレームワークと Dalvik VM を用いて動作し，待ち受け時の UI も Android 特有のホームアプリケーションにより処理されている．

一方，カーネルには Linux カーネルが使用されており，多くの部分は変更されることなくそのまま使用されている．ただし，次節で述べる low memory killer など Android 固有の機能も追加されている．

2.2 low memory killer

Android には，low memory killer という独自のプロセスメモリ管理システムが搭載されている．これは，システム全体のメモリ空き容量が閾値以下まで下がった場合に起動され，後述の *adj* と *minfree* の関係に基づいてプロセスを選定し強制終了するプログラムである．多くの携帯端末同様に，Android はユーザがアプリケーションの終了処理を行わずに使用できるように設計されている．すなわち，ユーザはそのときに使用したいアプリケーションの起動（開始）のみを行えばよく，アプリケーションの終了操作を行う必要がない．よって，ユーザがアプリケーションを終了させることなく稼働アプリケーション数を増やし続けると OS が管理する空きメモリ量が単調に減り続け，必然的にシステムはメモリ不足の状態に陥る．low memory killer はこの問題を解決するために（換言すると，ユーザにアプリケーション終了操作を要求しないシステムを実現するために）用意されており，システム全体の空きメモリ量が少なくなると自動的にアプリケーションを終了する．

システム内のプロセスには *adj* と呼ばれる値が設定されており，*adj* の値が小さいプロセスほど強制終了されにくく，重要なプロセスほど小さな *adj* が設定されている．**表 1** に Android 4.0.3 におけるプロセスの状態と *adj* の値の関係を示す．最も強制終了されにくいフォアグラウンドのアプリケーションに 0 が，サービスアプリケーションに 5 が，最も強制終了されやすいバックグラウンドのアプリケーションに 15 が割り当てられている．

表 1　プロセスの状態，種類による *adj* の値
Table 1　*adj* and process status/type.

adj	プロセスの状態，種類
0	FOREGROUND_APP
1	VISIBLE_APP
2	PERCEPTIBLE_APP
3	HEAVY_WEIGHT_APP
4	BACKUP_APP
5	SERVICE
6	HOME_APP
7	PREVIOUS_APP
8	SERVICE_B
9	HIDDEN_APP_MIN
15	HIDDEN_APP_MAX

low memory killer が起動されると，2 段階で強制終了プロセスの選定が行われる．まず第 1 次判定として，*adj* を用いた比較が行われる．第 1 次判定で強制終了対象を一意に定めることができない場合は第 2 次判定としてメモリ使用量による比較が行われる．具体的には，起動しているすべてのプロセスの *adj* を比較し，*adj* の数値がより高いプロセスを強制終了する候補にあげる（第 1 次判定）．最高 *adj* のプロセスが 1 個しかない場合は第 1 次判定で選定は終了する．最高 *adj* のプロセスが複数存在する場合，メモリ使用量を比較し，メモリ使用量のより多いプロセスを強制終了する候補にあげる（第 2 次判定）．

前述のとおり，low memory killer はシステムの空きメモリ量が少なくなると起動され，空きメモリ量が指定の値を超えるまで *adj* が大きなプロセスを強制終了する．low memory killer が起動する空きメモリ量の閾値（*minfree* と呼ばれる）および強制終了対象とする *adj* の閾値の組合せは，`/init.rc` で定義されており，low memory killer において強制終了するプロセスを選定する際に参照される．*minfree* は，プロセス強制終了実行の閾値となる空きページ数であり，*adj* によってランク分けされている．たとえば Android 4.0.3 では $minfree = 9607\,[4\,\mathrm{KB}] = 38428\,[\mathrm{KB}]$ と $adj = 9$ の組合せが定義されており，メモリ空き容量が 38428 [KB] 以下になると *adj* の値が 9 以上の数値を持つプロセスが強制終了される候補に上がる．そして，空きメモリ量が 38428 [KB] 以上になるか，*adj* が 9 以上のプロセスがなくなるまでプロセスを強制終了する．

3. アプリケーションの起動手順

Android においてアプリケーションのプロセスは「プロセスが存在しない」「プロセスがフォアグラウンド状態で存

© 2015 Information Processing Society of Japan

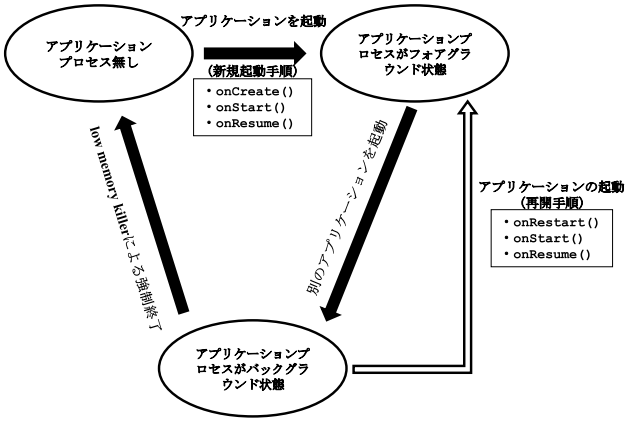

図 1　アプリケーション起動状態遷移図
Fig. 1　Application process status transition.

在する」「プロセスがバックグラウンド状態で存在する」の 3 種類の状態がありうる．また，アプリケーション起動の方法は「新規起動」と「再開」の 2 種類がある．前者はアプリケーションプロセスが存在しない状態からアプリケーションプロセスを生成する起動方法であり，後者はバックグラウンド状態のアプリケーションプロセスが存在する状態から既存プロセスを再利用して起動する方法である．

これらの 3 種類のアプリケーションプロセスの状態と，2 種類の起動方法を**図 1** に示す．OS が起動した直後はアプリケーションのプロセスは存在しておらず「アプリケーションプロセスなし」の状態である．ここで，ユーザがアプリケーションを起動すると「新規起動」の方法を用いてアプリケーションプロセスが生成され，「アプリケーションプロセスがフォアグラウンド状態」に移行する．次にユーザが別のアプリケーションを起動すると，当該アプリケーションのプロセスは「バックグラウンド状態」に移行する．バックグラウンド状態に移行している間にシステム全体のメモリ量が不足すると，当該プロセスは low memory killer により強制終了されてしまい「プロセスなし」の状態に移行する．メモリ不足が発生しなければ当該プロセスは「バックグラウンド状態」で存在し続ける．その後にユーザが当該アプリケーションを再度使用しようとしたときの動作は，アプリケーションプロセスの状態により異なる．「プロセスなし」の状態であれば「新規起動」の手順により起動され，「バックグラウンド状態」であれば「再開」の手順により起動される．

Android のアプリケーションを新規に起動する場合，**図 2** のような手順に従って起動される．① ユーザがアプリケーションのアイコンをタッチするとホームアプリケーション（Launcher）がアプリケーション起動要求のインテントを ActivityManager に送信する．② ActivityManager が Zygote にプロセス生成要求を送信する．③ Zygote が自分自身を fork し，子プロセスを生成する．④ 新しいプロセスが初期化される．⑤ アプリケーションのライフサイク

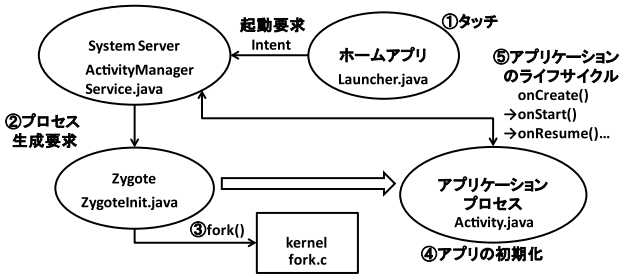

図 2　アプリケーションの起動手順
Fig. 2　Application launching procedure of re-forking.

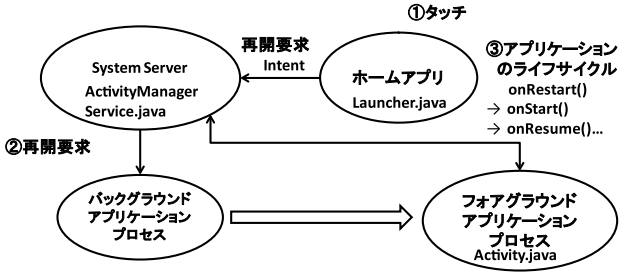

図 3　アプリケーションの再開手順
Fig. 3　Application launching procedure of re-using.

ルに従って `onCreate()`，`onStart()`，`onResume()` が呼び出される．

また，起動済みであるがバックグラウンド状態にあるプロセスの再開は，**図 3** のような手順に従って行われる．① ユーザがアプリケーションのアイコンにタッチするとホームアプリケーションが再開要求のインテントを ActivityManager に送信する．② ActivityManager は対象のバックグラウンドアプリケーションプロセスに再開要求を出す．③ バックグラウンドアプリケーションプロセスは再開要求を受けてアプリケーションプロセスとして起動しフォアグラウンド状態となる [3]．本稿では前者を「新規起動」，後者を「再開」と呼ぶ．

起動済みプロセスが low memory killer によって強制終了されることなく再利用された場合は，上記の「再開」の手続きが行われるが，プロセスが強制終了された後に再利用された場合は「新規起動」の手続きが行われる．通常，「再開」よりも「新規起動」の方が要する時間が長いため，後に再利用するプロセスの強制終了はユーザの待ち時間を増加させてしまう．

4. 提案手法

本章で，low memory killer における新しい強制終了プロセス選定手法を提案する．提案手法は標準 low memory killer をベースとしており，標準手法と同様に第 1 次判定，第 2 次判定により構成される．すなわち，標準手法の第 1 次判定および第 2 次判定の判定方法を改変し，提案手法を実現する．

具体的な提案としては，LRU に基づいて強制終了プロセ

© 2015 Information Processing Society of Japan

スを選定する LRU 手法と，アプリケーションの新規起動時間をもとに選定する起動時間手法と，これらを組み合わせた手法を提案する．LRU アルゴリズムの詳細に関しては付録を参照されたい．

以下における F-標準，F-LRU，F-起動時間，F-LRU × 起動時間手法は第 1 次（First）判定に LRU や起動時間を用いる手法であり，S-標準手法および S-メモリ × LRU 手法は第 2 次（Second）判定にメモリ量やメモリ量 × LRU を用いる手法である．4.1 節および 4.2 節の手法は我々の提案手法ではなく標準の low memory killer の手法であるが，比較のために本稿ではこれらを F-標準手法，S-標準手法と呼ぶ．

4.1　F-標準手法

比較のため，第 1 次判定を *adj* で行う標準 low memory killer の手法を本稿では F-標準手法と呼ぶ．標準 low memory killer の動作の説明は付録 A.2 に記す．

4.2　S-標準手法

強制終了プロセス選定の第 2 次判定をプロセスのメモリ使用量で行い，よりメモリ使用量の多いプロセスを強制終了する対象とする手法を S-標準手法と呼ぶ．

本手法は，標準 low memory killer の動作である．比較のため，これを S-標準手法と呼ぶ．標準 low memory killer の動作の説明は付録 A.2 に記す．

4.3　F-LRU 手法

本節以下（4.3 節から 4.6 節）が我々の提案手法となる．

本節で，強制終了するアプリケーションの第 1 次判定を LRU 方式を用いて行う F-LRU 手法を提案する．

本手法では，ユーザが実行したアプリケーションを実行履歴として一定数記録しておく．履歴は LRU により管理し，最後に参照されてからの時間が短いものの *adj* を下げ強制終了の対象となりにくくする．

後述の評価用実装では以下のように実装されている．長さ 15 の配列を用意し，ユーザが起動したアプリケーション履歴を保持する．配列は LRU で管理する．そして，low memory killer における強制終了プロセス選定の際，アプリケーションの履歴内の順位を調査し，履歴の n 番目にあるプロセスの *adj* を $n/2$ に変更する．順位は新しい方から数え，最小は 0 とし，小数点以下は切り捨てとする．ただし，変更前の *adj* が変更後の *adj* より低い場合は，変更前の *adj* を優先する．この手法により，最近使用されたアプリケーションのプロセスが強制終了されにくくなり，ユーザの待ち時間を低減させることができると期待される．履歴内に記録のないアプリケーションの *adj* は変更を加えない．履歴の長さや，*adj* の下げ幅はチューニングパラメータである．ただし本稿では，履歴内のアプリケーションを履歴外のアプリケーションよりも優先するために，履歴の最後尾のアプリケーションの調整後の *adj* が履歴外のアプリケーションの *adj* より小さくなるように設定することを推奨する．本稿の場合，評価実験を Android 4.0.3 を用いて行っており，同 OS で SERVICE_B や HIDDEN_APP_MIN の *adj* が 8 や 9 である．よって，履歴長を 15，*adj* 下げ幅を $n/2$ として，履歴最後尾のアプリケーションの調整後の *adj*（15/2）が 8 や 9 を下回るように設定している．

4.4　F-起動時間手法

本節で，アプリケーションの新規起動時間を考慮して強制終了するプロセスの第 1 次判定を行う F-起動時間手法を提案する．本手法では，アプリケーションの新規起動時間を計測する．そして，新規起動時間が長いアプリケーションの *adj* を下げ強制終了の対象となりにくくする．新規起動時間が長いアプリケーションの新規起動を積極的に避けることにより，ユーザの待ち時間を大きく増加させることを回避できると期待される．

アプリケーションの新規起動時間は，アプリケーションのライフサイクルにおける `onCreate()` の呼び出しから `onResume()` の呼び出しまでの時間とし，提案システムでは Android アプリケーションフレームワークを改変してアプリケーションごとの新規起動時間の履歴をとる．そして，履歴内のアプリケーションに対して新規起動時間の長い順に順位（rank）をつける．たとえば，新規起動時間の一番長いアプリケーションのプロセスは，rank = 0 となる．low memory killer による強制終了プロセス選定の際，アプリケーション履歴内に存在するアプリケーションのプロセスの rank を参照し，*adj* を rank/2 にすることで，新規起動時間の長いアプリケーションを強制終了されにくくする．本手法では rank と *adj* の 2 種類の値が使用されているが，強制終了対象の決定は *adj* の値のみの比較により行われる．rank は提案手法の内部で算出される「起動時間の順位」であり，rank と *adj* が直接比較されることはない．ただし，履歴内のアプリケーションの *adj* は rank/2 に変更されるため強制終了されにくくなる．すなわち，rank は *adj* に変換されることにより選定に影響を及ぼす．

本手法では履歴は FIFO（First-In First-Out）で管理され，後述の評価用実装では最新の 15 個のアプリケーションの新規起動時間を保持し，それより古いものは破棄する．すなわち，起動時間の大小にかかわらずアプリケーションが履歴に入ってから一定回数（履歴長と同数）の更新でアプリケーションは履歴から破棄される．よって，長時間前に起動し最近使用していないアプリケーションが長期にわたり強制終了の対象とならないことは生じない．履歴の長さと *adj* の下げ幅はパラメータである．ただし前節同様に，履歴最後尾のアプリケーションの調整後の *adj* が履歴外のアプリケーションの *adj* より小さくなるように履歴長と *adj* 下げ幅を設定することを推奨する．

© 2015 Information Processing Society of Japan

4.5　F-LRU × 起動時間手法

本節で，F-LRU 手法と F-起動時間手法を組み合わせて第 1 次判定を行う F-LRU × 起動時間手法を提案する．

本手法では，F-LRU 手法と F-起動時間手法を並列して動作させ，両値の小さい方をそのプロセスの *adj* とする．

4.6　S-メモリ × LRU 手法

本節で，強制終了プロセス選定の第 2 次判定を「プロセスのメモリ使用量」と「最後に使用されてからの時間が長い順の順位」の積で行う手法を提案する．アプリケーションが LRU の履歴内に存在しない場合は，最後に使用されてからの時間が長い順の順位として (履歴長 + 1) を用いる．これにより，履歴内に存在しないアプリケーションの評価を履歴内に存在するすべてのアプリケーションよりも低い評価（履歴の最後尾のアプリケーションの次に低い評価）とすることができる．

5.　評価

本章で，標準 low memory killer（以下標準手法と呼ぶ）および提案手法のアプリケーションの合計起動時間の評価を行う．

5.1　評価方法

標準手法および提案手法が実装された Android スマートフォンにおいて，複数のアプリケーションを順に起動し，その起動に要した時間を測定した．実験は，**表 2** に示す仕様のスマートフォンを用いて行った．

比較は，Android の標準の手法として F-標準-S-標準手法の評価を，第 1 次判定に各提案手法を用いたものとして F-LRU-S-標準手法，F-起動時間-S-標準手法，F-LRU × 起動時間-S-標準手法の 3 個の手法の評価を，第 2 次判定に提案手法を用いたものとして F-標準-S-メモリ × LRU 手法の評価を行った．また，第 1 次判定と第 2 次判定の両方に提案手法を用いた場合の例として本稿で着目している LRU を両者に適用した F-LRU-S-メモリ × LRU 手法の評価を行った．

5.2　評価シナリオ

本稿では，起動するアプリケーションの順番を定めたものを “シナリオ” と呼ぶ．本実験では，第三者から許可を得て取得したユーザの実際の 1 日のアプリケーション使用履歴であるシナリオを 2 つ（シナリオ A，B）用意し，標準手法とすべての提案手法の評価を行った（シナリオ A，B のアプリケーション起動順は**表 5** と**表 6** 参照）．

また，シナリオではアプリケーション起動とアプリケーション起動の間にホームアプリケーション（Launcher）を起動し，実際の使用順に近いものとしている．

表 2　実験環境

Table 2　Experimental setup.

Device name	Nexus S
OS	Android4.0.3
CPU	Cortex A8 (Hummingbird) Processer 1GHz
Memory	512MB

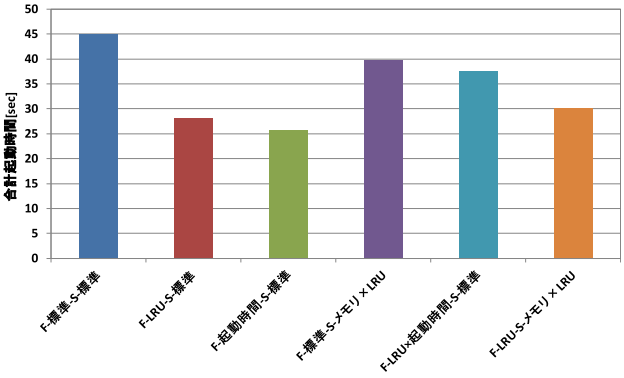

図 4　シナリオ A における評価結果

Fig. 4　Experimental result (scenario A).

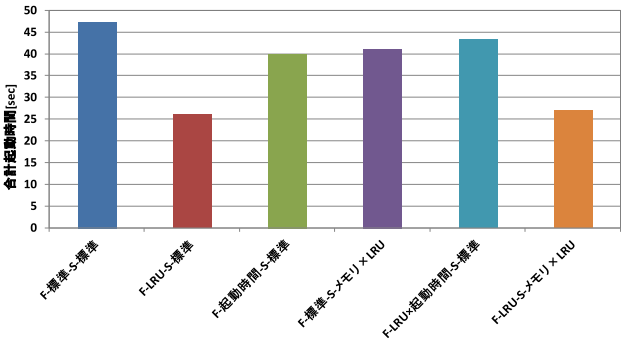

図 5　シナリオ B における評価結果

Fig. 5　Experimental result (scenario B).

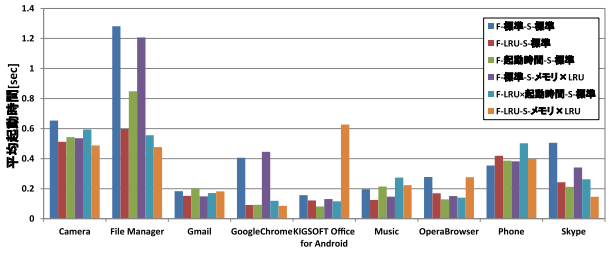

図 6　シナリオ A におけるアプリの平均起動時間

Fig. 6　Average launching time of applications (scenario A).

5.3　評価結果（終了プロセス選定手法）

本節で，各終了プロセス選定手法の性能の比較を行う．シナリオ A，B における評価結果を**図 4**，**図 5**，**図 6**，**図 7** に示す．図 4，5 の縦軸は，シナリオのホームアプリケーション（Launcher）以外の全アプリケーションの起動時間の合計であり，少ないほど優れているといえる．図 6，7 の縦軸は各アプリケーションの起動時間の平均値であり，同様に少ないほど優れているといえる．

© 2015 Information Processing Society of Japan

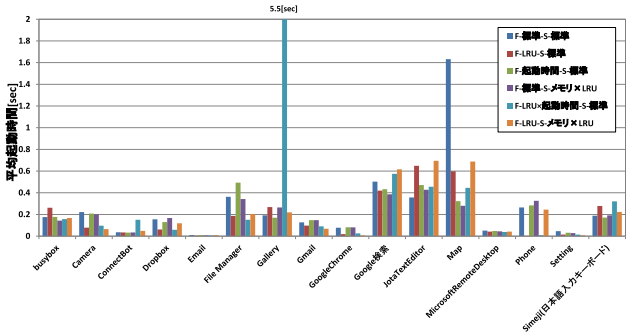

図 7　シナリオ B におけるアプリの平均起動時間
Fig. 7　Average launching time of applications (scenario B).

表 3　シナリオ A における新規起動，再開の数
Table 3　Re-forking and re-using times (scenario A).

手法	新規起動回数	再開回数
F-標準-S-標準	53	101
F-LRU-S-標準	41	113
F-起動時間-S-標準	39	115
F-標準-S-メモリ×LRU	48	106
F-LRU×起動時間-S-標準	44	110
F-LRU-S-メモリ×LRU	42	112

表 4　シナリオ B における新規起動，再開の数
Table 4　Re-forking and re-using times (scenario B).

手法	新規起動回数	再開回数
F-標準-S-標準	76	321
F-LRU-S-標準	33	364
F-起動時間-S-標準	75	322
F-標準-S-メモリ×LRU	76	321
F-LRU×起動時間-S-標準	31	366
F-LRU-S-メモリ×LRU	32	365

また，シナリオ A，B におけるそれぞれの手法のアプリケーションの新規起動，再開の数を**表 3**，**表 4** に示す．

通常，新規起動による起動時間の方が再開による起動時間より長いため，基本的に新規起動の回数が少ないほど合計起動時間が短くなり，優れた手法であると考えることができる．ただし，新規起動時間はアプリケーションにより異なるため，新規起動の回数が同一でも合計起動時間に差が生じる．

図と表より，両シナリオにおいて標準手法よりすべての提案手法の方が合計起動時間が短く，また新規起動の回数も少ないまたは同等であることが確認でき，標準手法よりも優れた手法であることが分かる．

提案手法どうしを比較すると，F-LRU-S-標準手法と F-LRU-S-メモリ × LRU 手法が両シナリオにおいて優れた

表 5　シナリオ A におけるアプリケーション起動順（上から下に，左から右に向かって順に起動）
Table 5　Scenario A.

Setting	File Manager	Gmail	KINGSOFT Office for Android
Phone	Skype	Google Chrome	Gmail
Opera Browser	Gmail	Gmail	KINGSOFT Office for Android
Gmail	Skype	Google Chrome	Gmail
Music	Gmail	Gmail	Google Chrome
Gmail	Google Chrome	Google Chrome	Gmail
Google Chrome	Gmail	Gmail	Google Chrome
Gmail	Google Chrome	Skype	Gmail
Opera Browser	Gmail	Google Chrome	Google Chrome
Skype	File Manager	Gmail	Gmail
Gmail	Google Chrome	Google Chrome	Google Chrome
Skype	Gmail	Skype	Gmail
Gmail	Skype	Google Chrome	KINGSOFT Office for Android
Google Chrome	Gmail	Skype	Gmail
Gmail	Google Chrome	Google Chrome	KINGSOFT Office for Android
KINGSOFT Office for Android	Gmail	Skype	Gmail
Gmail	Google Chrome	Google Chrome	KINGSOFT Office for Android
Phone	Gmail	Skype	Gmail
Google Chrome	Google Chrome	Google Chrome	Google Chrome
Gmail	Skype	Skype	Gmail
Google Chrome	Google Chrome	Google Chrome	Google Chrome
Gmail	Gmail	Skype	Gmail
Google Chrome	Google Chrome	Google Chrome	Skype
Gmail	Gmail	Skype	Gmail
Google Chrome	Google Chrome	Gmail	KINGSOFT Office for Android
Gmail	Gmail	Google Chrome	Skype
Google Chrome	Google Chrome	Gmail	Gmail
Gmail	Skype	Skype	Google Chrome
KINGSOFT Office for Android	Google Chrome	Gmail	Gmail
Google Chrome	Gmail	Google Chrome	Google Chrome
Gmail	Google Chrome	Gmail	KINGSOFT Office for Android
Phone	Skype	Google Chrome	Google Chrome
Google Chrome	Gmail	Gmail	Gmail
Phone	Google Chrome	Google Chrome	Google Chrome
Google Chrome	Gmail	Gmail	Gmail
Gmail	Skype	KINGSOFT Office for Android	Skype
Skype	Gmail	Gmail	Google Chrome
Google Chrome	Google Chrome	KINGSOFT Office for Android	
Gmail	Camera	Gmail	

性能を示しており，第 1 次判定においては LRU を用いることが好ましいことが分かる．

次に，各手法の動作に基づき性能に関する考察を記す．表 3，表 4 よりシナリオ B の F-標準-S-メモリ × LRU 手法を除くすべての提案手法において新規起動回数が Android の標準手法（F-標準-S-標準手法）よりも減少されていることを確認でき，新規起動回数の削減が合計起動時間の削減につながっていることが分かる．同様に，シナリオ B の F-起動時間-S-標準手法およびシナリオ B の F-標準-S-メモリ × LRU 手法においては新規起動回数の削減がほとんどない，あるいはまったくない状況であり，新規起動回数の削減の程度が小さいと合計起動時間の削減の程度も小さいことが分かる．

5.4　評価結果（チューニングパラメータ）

本節で，提案手法のチューニングパラメータ（履歴長および *adj* の下げ幅）の影響の評価を行う．履歴長を 5，10，15，20 と変更し，各提案手法の性能を評価した．ただし 4 章の提案のとおり，履歴最後尾の *adj* が一般アプリケーションの *adj* より小さくなるように，履歴最後尾アプリケーションの調整後の *adj* が標準設定（履歴長 = 15，下げ幅 = $n/2$）と等しくなるように *adj* 下げ幅を定めた．すなわち，履歴長 5，10，15，20 の *adj* 下げ幅は，順に $n*3/2$，$n*3/4$，$n/2$，$n*3/8$ とした．

各履歴長および下げ幅のシナリオ A における評価結果を**図 8** に示す．図より以下のことを確認することができる．まず，履歴長を調整することにより提案手法の性能に変化があるが，いずれの値に調整しても Android 標準の手法（44.9 [sec]）よりも性能が優れており，提案手法はチューニ

© 2015 Information Processing Society of Japan

表 6　シナリオ B におけるアプリケーション起動順（上から下に，左から右に向かって順に起動）

Table 6　Scenario B.

File Manager	Email	Google Chrome	Setting
JotaTextEditor	Email	Dropbox	Gmail
File Manager	Email	Dropbox	Gmail
Setting	Setting	Dropbox	Gmail
Setting	Setting	Google Chrome	Gmail
Setting	Setting	Dropbox	Google Chrome
Setting	Setting	Dropbox	Gmail
Setting	Setting	Dropbox	Gmail
Setting	Setting	Dropbox	Gmail
Google Chrome	Setting	Dropbox	Gmail
Dropbox	Setting	Dropbox	Gmail
Google Chrome	Email	Map	Gmail
Gmail	Email	Gmail	Google Chrome
Gmail	Setting	Gmail	Gmail
Gmail	Setting	Gmail	Google Chrome
Google Chrome	Setting	Gmail	Gmail
Setting	Email	Gmail	Gmail
Setting	Setting	Setting	Gmail
Google Chrome	Setting	Setting	Gmail
Google Chrome	Setting	Gmail	Gmail
Gmail	Gmail	JotaTextEditor	Gmail
Gmail	Google Chrome	Gmail	Gmail
Gmail	Google Chrome	Gmail	Gmail
Setting	Setting	Gmail	Camera
Setting	Setting	Gmail	Gmail
Gmail	Google Chrome	Gmail	Gmail
Google Chrome	Google Chrome	Map	Camera
Google Chrome	Google Chrome	ConnectBot	File Manager
Google Chrome	Setting	ConnectBot	Gallery
Google Chrome	Setting	ConnectBot	File Manager
Gmail	Setting	ConnectBot	File Manager
Google Chrome	Setting	ConnectBot	Gmail
Google Chrome	Setting	ConnectBot	File Manager
Gmail	Setting	ConnectBot	Google Chrome
Google Chrome	Setting	ConnectBot	Setting
Google Chrome	Setting	Google Chrome	Setting
Google Chrome	Setting	Setting	Setting
Gmail	Setting	Setting	Setting
Gmail	Setting	Setting	Gmail
Gmail	Setting	Setting	Gmail
Gmail	Setting	Google Chrome	Gmail
Gmail	Google Chrome	ConnectBot	Gmail
Gmail	Setting	ConnectBot	Gmail
Gmail	Setting	ConnectBot	Gmail
Gmail	Setting	ConnectBot	Gmail
Gmail	Setting	ConnectBot	Google Chrome
Google Chrome	Setting	ConnectBot	Gmail
Google Chrome	Phone	ConnectBot	Gmail
Google Chrome	Setting	MicrosoftRemoteDesktop	Gmail
Google Chrome	Setting	MicrosoftRemoteDesktop	Camera
Google Chrome	Setting	MicrosoftRemoteDesktop	File Manager
Gmail	Gmail	MicrosoftRemoteDesktop	Camera
Google Chrome	Gmail	ConnectBot	File Manager
Google Chrome	Gmail	ConnectBot	Gallery
Google Chrome	Google Chrome	MicrosoftRemoteDesktop	File Manager
Google Chrome	Setting	MicrosoftRemoteDesktop	Gallery
Google Chrome	Setting	Google Chrome	File Manager
Setting	Setting	Setting	Gallery
Setting	Setting	Setting	File Manager
Setting	Gmail	Setting	File Manager
Setting	Gmail	Setting	Gmail
Setting	Gmail	Setting	Gmail
Setting	Gmail	Gmail	Gmail
Setting	Gmail	Gmail	Gmail
Setting	Gmail	Gmail	Gmail
Setting	Gmail	Gmail	Gmail
Setting	Gmail	Gmail	Simeji(日本語入力キーボード)
Setting	Gmail	Gmail	Gmail
Gmail	Gmail	Gmail	Gmail
Gmail	Gmail	Google Chrome	Gmail
Gmail	Gmail	Google Chrome	Gmail
Gmail	Gmail	Google Chrome	Gmail
Gmail	Gmail	Google Chrome	Gmail
Gmail	Gmail	Gmail	Google Chrome
Gmail	Gmail	Gmail	Google Chrome
Gmail	JotaTextEditor	Gmail	Google Chrome
Gmail	Setting	Gmail	Gmail
Gmail	Setting	Gmail	busybox
Setting	Google Chrome	Gmail	Google Chrome
Setting	busybox	Gmail	Google Chrome
Setting	Setting	Setting	Gmail
Email	Setting	Setting	Google Chrome
Setting	Setting	Setting	Google Chrome
Email	Setting	Setting	Google Chrome
Setting	Setting	Setting	Google Chrome
Email	Setting	Camera	Google Chrome
Setting	Setting	Camera	Google Chrome
Email	Gmail	Setting	Google Chrome
Email	Google Chrome	Setting	Google検索
Email	Gmail	Setting	Gmail
Email	Gmail	Gmail	Gmail
Email	Google Chrome	Gmail	File Manager
Gmail	Setting	Setting	Gmail
Email	Setting	Setting	Gmail
Email	Setting	Setting	Gmail
Email	Setting	Setting	Gmail
Email	Google Chrome	Camera	Gmail
Email	Google Chrome	Camera	File Manager
Email	Google Chrome	Setting	Gmail

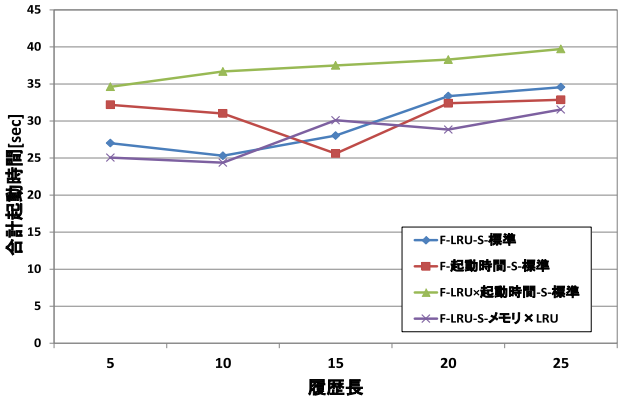

図 8　シナリオ A における履歴長ごとの評価結果

Fig. 8　Experimental result (log length and total launching time, scenario A).

ングパラメータの調整を行わなくても有効な手法であることが分かる．次に，提案手法はチューニング（履歴長の調整）を行うことによりさらなる性能向上が可能であることが分かる．

6. 考察

6.1 手法の優劣の変化

前章で各手法の性能評価を行い，多くの例において LRU を用いる手法が最高あるいは最高に近い性能を示すことを確認した．ただし，図 4 のように LRU が最高の性能を示さない例も確認された．図 5 のシナリオ B の結果や，図 8 より，起動時間手法は必ずしも高い性能を示さないことが分かるが，図 4 のシナリオ A の結果では最高の性能を示している．これは F-起動時間-S-標準手法の性能がシナリオ内の新規起動時間が長いアプリケーションの使用頻度に依存するからである．シナリオ A において起動時間の長いアプリケーションは Gmail，Google Chrome，Skype であり，シナリオ B において起動時間の長いアプリケーションは Map（Google Map），Gallery，Gmail，File Manager であった．シナリオ A において起動時間が長いアプリケーションは，使用頻度も高くなっている．よって，起動時間手法によりこれらのアプリケーションの強制終了を回避したことは，今後高頻度で使用されるアプリケーションの強制終了を回避する目的からもずれず，結果的に本手法の欠点は表面化しなかった．一方シナリオ B では起動時間手法が強制終了を回避したアプリケーションは使用頻度がきわめて低く，今後使用される可能性や頻度が低いアプリケーションの強制終了を回避してしまっており，LRU 手法より劣る結果になっている．換言すると，LRU 手法と起動時間手法が重要と考え強制終了を回避しようとするアプリケーションが類似している場合は両手法の性能は同程度となり，両手法が強制終了を回避しようとするアプリケーションに差異がある場合は LRU 手法の方が良い性能を示すことができることとなる．このことから，環境に依存せ

© 2015 Information Processing Society of Japan

ず最高に近い性能を提供できる LRU が多くの場合には適しており，起動時間が長く高頻度で使用されるアプリケーションの存在が既知である場合などは起動時間手法を用いることにより LRU 手法よりもさらに良い性能を実現できることがあるといえる．

F-LRU × 起動時間-S-標準手法は，シナリオ B において新規回数が少ないにもかかわらず合計起動時間が大きくなっている．また，シナリオ B において当該手法を除き新規起動回数と合計起動時間に強い相関があるが，当該手法のみ新規起動回数から予想される値と異なる合計起動時間となっている．これは図 7 のように Gallery アプリケーションが大きな起動時間を示したことが原因である．Gallery アプリケーションは起動時にストレージから画像データを読み込むなどを行い，ページキャッシュの状態などによりその起動時間に大きな分散が生じることが予想できる．これに対して F-LRU × 起動時間手法は過去の新規起動時間の履歴より当該アプリケーションの大きな新規起動時間の発生を回避するような動作をすることが期待されたが，Gallery アプリケーションの使用間隔が長すぎて結果的に回避をすることができなかった．すなわち，今回用いた履歴のように使用アプリケーションに時間的局所性が存在する場合は，起動時間を考慮する手法を用いても必ずしも効果的に動作しないことが分かる．このことからも，多くの場合において LRU が優れていると考えることができる．

6.2　標準手法のチューニング

標準の手法においても閾値（*adj* と *minfree*）の調整により動作を変更することができる．標準手法のチューニングにより実現できることについて考察する．

前述のとおり，low memory killer は空きメモリ量が少なくなったときにアプリケーションプロセスを強制終了する．*minfree* は low memory killer が発動する空きメモリ量を定め，*adj* は発動時に強制終了されるプロセスを定める．よって，これらを調整することにより low memory killer 発動の積極性を調整することができる．すなわち，*minfree* の値を増やすことにより，より大きな空きメモリ量で発動させるように変更することができ，*adj* 値を下げることにより，発動時により多くのアプリケーションを強制終了することが可能となる．より積極的にプロセスを強制終了することによりバックグラウンドアプリケーションおよびそれらにより消費される計算資源を減少させることが可能となり，結果的にフォアグラウンドアプリケーションに多くの計算資源を与えフォアグラウンドアプリケーションをより快適に動作させることが可能となる．ただし，積極的にアプリケーションを強制終了するとアプリケーション再利用時にプロセスの新規起動が必要になる確率が増えるため，アプリケーション切替え時により長い時間待たされる確率が増える欠点につながる．同様に low memory killer 発動の積極性を下げると，アプリケーション切替え時にプロセス再開となり待ち時間が短くなる確率を増加させることができるが，フォアグラウンドアプリケーションの動作の快適さを低減させることなる．すなわち，プロセスが再開となる確率とフォアグラウンドアプリケーションの動作の快適さはトレードオフの関係にあり，両閾値（*adj* と *minfree*）の調整によりこれを制御することができる．

一方，これら閾値の調整により強制終了されるアプリケーションプロセスの選定や順位を制御することは不可能であり，本稿で提案した手法のように LRU などに基づいて今後再利用される可能性が高いと期待されるアプリケーションを強制終了対象から外すなどの制御を実現することはできない．

また，提案手法と両閾値の制御は排他的な関係にはないため，強制終了対象の調整と積極性の調整は独立に行うことができる．

6.3　提案手法のチューニングパラメータ

端末のメモリサイズとチューニングパラメータ（履歴長と *adj* の下げ幅）との関係に関する考察を行う．

履歴はカーネルのメモリ内に記録されており，履歴 1 個につき 12 バイト消費する．Android が搭載されている現在の多くの端末では履歴長を 1,000 程度（12 KB）としても問題ないと予想される．必要な履歴長はユーザの使用方法に依存するが，一般的なユーザであれば使用するアプリケーションの数は 1,000 未満であり現在の端末であれば十分な履歴長を用意することが可能であると我々は考える．

adj の下げ幅は優先度を制御するパラメータであるが，前章で示したように多くの場合は本稿で示した設定（履歴内で最も優先度が低く削除対象となりやすいアプリケーションの *adj* が，バックグラウンドアプリケーションの *adj* と同等以下）で十分な性能が得られると考えられる．また *adj* の下げ幅を調整することにより，バックグラウンドアプリケーションの優先度をホームアプリケーションやサービスプログラムより高くし強制終了されにくくすることが可能である．これは，履歴の最後尾の（優先度が最も低い）アプリケーションの *adj* が，ホームアプリケーションやサービスプログラムの *adj*（本稿の例では 6 と 5）よりも小さな値となるようにすればよい．たとえば，メモリが極端に不足している端末でユーザがホームアプリケーションの使用を犠牲にしてでもアプリケーションを強制終了したくない場合などに有効になる．

6.4　起動時間の定義

今回提案した手法では，アプリケーションの新規起動時間は `onCreate()` の開始時刻から `onResume()` の開始時刻までとしており，再開時間は `onRestart()` の開始時刻か

© 2015 Information Processing Society of Japan

ら onResume() の開始時刻までとしている．しかし，アプリケーションによっては onResume() が開始した後にデータの読み込みやページの読み込みを行うことがあり，これによりユーザには待ち時間が生じてしまうことが考えられる．このような場合，実際のユーザの待ち時間は，本稿で定義した「アプリケーション新規起動時間」を大きく上まわることになり，提案手法により得られるユーザの体感的待ち時間の低減は前章の評価をさらに上まわるものになると期待できる．たとえば，Web ブラウザアプリケーションの多くがこれに該当し，プロセスが強制終了されてしまった後にアプリケーションを使用すると onResume() の開始後に以前閲覧していたページの再読み込みが発生し，ユーザはこの間待たされることとなる．また，タブ型のブラウザの場合は閲覧タブの変更を行うたびに以前は読み込み済みであったページの再読み込みが生じユーザが待たされることとなる．

そのため，本稿で定めたアプリケーションの起動ではなく，onResume() の開始後に発生する処理に要する時間も考慮した手法を適用することでユーザの待ち時間のさらなる軽減を図ることができると予想される．

7. 関連研究

Android アプリケーションの起動時間の調査方法に関する研究としては，文献 [3], [4] の研究がある．これらにおいて，ユーザによるスクリーンのタッチからアプリケーション起動の完了までの時間の計測方法が示されている．また，実際のアプリケーション起動時間の測定結果と考察が示されている．本稿における起動時間の測定は当該研究で提案されている手法に基づき行われている．

Linux カーネルのモニタリングツールとしては，FTrace [5], [6]，SystemTap [7], [8]，LTTng [9], [10]，OProfile [11] がある．これらは Linux 用に構築されている．しかし，Linux 標準のアプリケーションプロセスの設計思想や実装と Android のアプリケーションプロセスの設計思想や実装は大きく異なるため，これらツールは Android の解析には適さない．具体的には本研究で必要となるアプリケーションフレームワークや Dalvik VM の動作の解析ができない．たとえば，アプリケーションフレームワーク内におけるアプリケーションライフサイクル（onCreate()，onRestart()，onStart()，onResume()）の時刻を取得することができず，アプリケーション起動時間の調査をすることができない．

アプリケーション起動時間の短縮に関する研究としては，Zygote の preload クラスの増加による起動時間の短縮の研究 [12] がある．当該研究は，Zygote による読み込み済みクラスの数を増加させることによりアプリケーション起動時におけるストレージからのクラス読み込み量を減少させ，起動時間の短縮を実現している．しかし，強制終了されたアプリケーションの起動時間の短縮を考える当該研究の手法と，強制終了するアプリケーションの最適化を考える本稿の手法はアプローチが大きく異なっている．また両手法は排他的な関係になく，両手法を同時に用いていくことが好ましいと考えられる．

8. おわりに

本稿で我々は low memory killer における強制終了プロセスの選定手法に着目し，LRU による選定手法とアプリケーションの起動時間を考慮した選定手法などを提案した．そしてこれらの手法の評価結果を示し，提案手法が標準手法よりもアプリケーション起動時間を短縮できることを示した．また，多くの例において LRU 手法が優れた性能を提供できることを示した．

今後は，onResume() の開始後に発生する処理に要する時間も考慮した選定手法について考察していく予定である．

謝辞　本研究は JSPS 科研費 24300034，25280022，26730040 の助成を受けたものである．

参考文献

[1] Android Captures Record 81 Percent Share of Global Smartphone Shipments in Q3 2013: available from ⟨https://blogs.strategyanalytics.com/WSS/post/2013/10/31/Android-Captures-Record-81-Percent-Share-of-Global-Smartphone-Shipments-in-Q3-2013.aspx http://www.idc.com/getdoc.jsp?containerId=prUS23638712⟩

[2] What is Android? | Android Developers: available from ⟨http://developer.android.com/guide/basics/what-is-android.html⟩

[3] 永田恭輔，山口実靖：Android アプリケーションの起動性能解析システムとその評価，マルチメディア，分散，協調とモバイル（DICOMO2012）シンポジウム，pp.83–90 (2012).

[4] Nagata, K. and Yamaguchi, S.: An Android Application Launch Analyzing System, *8th ICCM: 2012 International Conference on Computing Technology and Information Management* (2012/04/24).

[5] Rostedt, S.: ftrace tracing inftrastructure, available from ⟨http://lwn.net/Articles/270971/. SystemTap http://sourceware.org/systemtap/⟩.

[6] Eigler, F.C.: Problem solving with systemtap, *Proc. Ottawa Linux Symposium 2006* (2006).

[7] LTTng Project, available from ⟨http://lttng.org/⟩.

[8] Bird, T.: Measuring Function Duration with Ftrace, *Proc. Japan Linux Symposium* (2009).

[9] Desnoyers, M. and Dagenais, M.R.: The LTTng Tracer: A low impact performance and behavior monitor of GNU/Linux, *Proc. Ottawa Linux Symposium 2006*, pp.209–223 (2006).

[10] Levon, J. and Elie, P.: Oprofile: A system profiler for linux, available from ⟨http://oprofile.sf.net⟩ (Sep. 2004).

[11] Valgrind, available from ⟨http://valgrind.org⟩.

[12] Nagata, K., Nakamura, Y., Nomura, S. and Yamaguchi, S.: Measuring and Improving Application Launching Performance on Android Devices, *4th International Workshop on Advances in Networking and Computing* (*WANC '13*) (2013).

© 2015 Information Processing Society of Japan

付　　録

A.1　LRU

LRU（Least Recently Used）はキャッシュなどに用いられる置換対象決定のアルゴリズムである．最後に参照されてからの時間が最も長いデータを置換対象とする．**図 A·1** の例を用いてその動作を説明する．図内では，左のデータほど最後に参照されてから時間が長く，最右が MRU（Most Recently Used）の状態であり，最左が LRU（Least Recently Used）の状態である．t = 0 でキャッシュ内に A，B，C，D のデータが保持されており，最後に参照されてからの時間が長い順に A，B，C，D であるとする．

この後データ E にアクセスが行われると，キャッシュ内のデータが 1 個破棄されデータ E がキャッシュ内に格納されるが，最後に参照されてからの時間が最も長いデータ A が破棄対象となり，キャッシュ内のデータは t = 1 の状態に変わる．たった今（t = 0 に）アクセスされたデータ E は，最後にアクセスされてからの時間が最も短いため，キャッシュ内の最右ブロックに格納される．それ以外のデータ（B，C，D）は今回アクセスされていないため，これら（B，C，D）の中での順位を保ったまま残りのブロックに格納される．結果として t = 1 のように B，C，D，E の順となる．

同様に t = 1 の後にデータ F にアクセスされると，データ B が破棄され，データ F が最右ブロックに格納される．結果，t = 2 のように C，D，E，F の順になる．

次にデータ D にアクセスされると，データ D が最後にアクセスされてからの時間が最短となるため，データ D が最右ブロックに格納され，これ以外（C，E，F）は相対順位を保ったまま残りのブロックに格納され，t = 3 のように C，E，F，D の順となる．

次にデータ G，データ H にアクセスされると，最後に参照されてからの時間が長いデータ C，データ E が破棄され，t = 4，t = 5 の状態に遷移する．

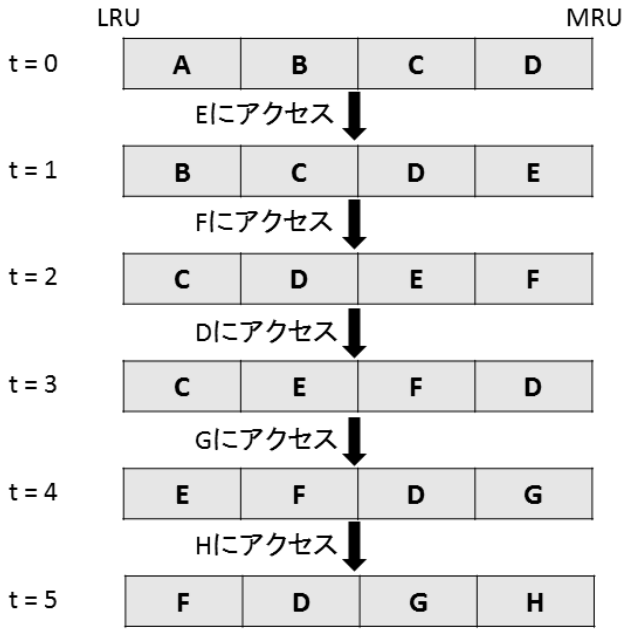

図 A·1　LRU の動作
Fig. A·1　LRU.

A.2　low memory killer 実装

本稿で用いた low memory killer の実装の抜粋と解説を記す．

以下は，`lowmemorykiller.c` 内の `lowmem_shrink` 関数の抜粋である．行番号が記載されている行は `lowmemorykiller.c` からの引用であり，行番号がない行は我々が追加した解説である．low memory killer が *adj* が大きいプロセスを，*adj* が同じであれば使用メモリ量が大きいプロセスを強制終了対象にしていることを確認できる．

```
82 static int lowmem_shrink(struct shrinker *s, struct
shrink_control *sc)
 83 {
          :
 90       int selected_tasksize = 0;
          :
135       for_each_process(p) {
                // ↑全てのプロセスを調査するループ
                :
141             mm = p->mm;
142             sig = p->signal;
                :
147             oom_adj = sig->oom_adj;
                :
152             tasksize = get_mm_rss(mm);
                // ↑今回の調査対象のプロセスの
                //   使用メモリ量を変数 tasksize に，
                //   adj を 変数 oom_adj に格納する．
                :
156             if (selected) {
157                 if (oom_adj < selected_oom_adj)
158                     continue;
                    // ↑今回の調査対象プロセスの adj が
                    //   選択済みプロセス の adj より小さいなら，
                    //   今回の調査対象プロセスは対象外

159                 if (oom_adj == selected_oom_adj &&
160                     tasksize <= selected_tasksize)
161                     continue;
                    // ↑今回の調査対象プロセスの adj が
                    //   選択済みプロセス の adj と等しく かつ
                    //   今回の調査対象プロセスの使用メモリ量が
                    //   選択済みプロセスの使用メモリ量以下なら，
                    //   今回の調査対象プロセスは対象外
162             }
                // 「今回の調査対象プロセス の adj が
                // 選択済みプロセスの adj より小さい」あるいは，
                // 「adj が等しいく かつ 使用メモリが大きい」場合のみ，
                // 選択済みプロセスを更新
163             selected = p;
164             selected_tasksize = tasksize;
165             selected_oom_adj = oom_adj;
                :
168       }
169       if (selected) {
                :
175             force_sig(SIGKILL, selected);
                // 選択済みプロセスを強制終了
                :
177       }

182 }
```

© 2015 Information Processing Society of Japan

野村 駿　（学生会員）

2013年工学院大学工学部情報通信工学科卒業．同年同大学大学院工学研究科電気・電子工学専攻入学．AndroidOSの研究に従事．

中村 優太　（学生会員）

2013年工学院大学工学部情報通信工学科卒業．同年同大学大学院工学研究科電気・電子工学専攻入学．AndroidOSの研究に従事．

坂本 寛和　（学生会員）

2014年工学院大学工学部情報通信工学科卒業．同年同大学大学院工学研究科電気・電子工学専攻入学．AndroidOSの研究に従事．

山口 実靖　（正会員）

2002年東京大学大学院工学系研究科電子情報工学専攻博士課程修了．博士（工学）．同年より東京大学生産技術研究所学術研究支援員，産学官連携研究員，日本学術振興会特別研究員．2006年工学院大学工学部講師．2007年同大学同学部准教授．オペレーティングシステム，I/O高速化の研究に従事．電子情報通信学会，日本データベース学会各会員．

© 2015 Information Processing Society of Japan