사용 빈도가 높은 애플리케이션 사용 기록을 이용한 안드로이드 프로세스 로딩 개선

곽현호, 안진우, 신동군

성균관대학교 컴퓨터공학과

gusghrhkr@gmail.com, creator.ahn@gmail.com, dongkun.skku.edu

Improving process loading by using History of frequently-used applications

Hyunho Gwak, Jinwoo Ahn, Dongkun Shin

Department of Computer Engineering, Sungkyunkwan University

요 약

애플리케이션 시장이 확대되면서 사용자가 동시에 사용하는 애플리케이션 수가 늘었다. 가용 메모리가 적은 안드로이드 기기에서는 메모리를 확보하기 위해 프로세스를 종료시킨다. 이 때 프로세스가 자주 사용되는지, 앞으로 사용할 것인지 여부는 반영되지 않는다. 때문에 전에 아무리 많이 사용되었더라도 상태값이 높으면 종료될 가능성이 높다. 이 논문에서는 프로세스의 사용 기록을 저장하여 위와 같은 상황에서 보호하는 방법을 제시한다.

1. 서 론

안드로이드 기기의 보급으로 인해 애플리케이션 시장이 확대되면서 다양한 애플리케이션들이 등장하였다. 이에 따라 사용자가 사용하는 애플리케이션의 종류가 많아졌으며 동시에 사용하는 애플리케이션의 수가 증가하였다. 그런데 특히 모바일 기기같이 작은 메모리 공간에서 실행할 수 있는 양은 한정되어 있다. 그래서 안드로이드는 자신만의 방식을 통해 종료시킬 프로세스를 선택하며 메모리를 관리한다.

안드로이드가 메모리를 관리할 때, 프로세스의 상태값을 참조하는 방법 두가지가 있다. 상태값이란 프로세스의 현재 상태를 나타내는 값으로, 현재 화면에서 실행되고 있는지, 백그라운드 상태에 있는지 등을 나타낸다. 첫번째 방법으로 LMK(Low Memory Killer)가 있고 두번째 방법으로 HIDDEN 상태 프로세스 최대 개수를 제한하는 것이 있다.

첫번째 LMK는 안드로이드에서 사용되는 메모리 관리 모듈로서 메모리가 부족한 상황에서 호출되어 프로세스 상태값이 높은 프로세스를 우선적으로 종료시킨다.

두번째 방법은 백그라운드 프로세스의 최대 개수를 제한하는 것으로 HIDDEN 상태 프로세스 개수가 지정된 MAX_HIDDEN_APPS값을 넘게 되면 프로세스를 종료시킨다.

이 논문은 2010년도 정부(미래창조과학부)의 재원으로 한국연구재단-차세대정보·컴퓨팅기술개발사업의 지원을 받아 수행된 연구임(No. 2010-0020724)

위 두 방법은 프로세스 상태값에 의존하여 종료시킬 프로세스를 선택한다. 프로세스의 상태값은 프로세스의 상태를 나타내며, 각 프로세스마다 계속 계산되어 상태를 갱신한다.

유저 애플리케이션 프로세스들은 FOREGROUND 상태부터 시작해서 사용되지 않으면 점점 HIDDEN 상태로 올라간다.

이 프로세스 상태값은 계산될 때 그 프로세스가 앞으로 사용될 가능성은 반영하지 않는다. 때문에 프로세스 상태값에 의존하는 앞에 두 방법에서는 바로 사용될 프로세스를 잠시 사용되지 않아 프로세스 상태값이 높아졌다면 그 프로세스를 종료시킨다. 그래서 매번 사용할 때마다 종료되어 있어서 로딩시켜야 하는 상황이 생길 수 있다.

이 논문에서는 위와 같은 상황을 해결하는 방법을 제시한다. 사용자가 앞으로 사용할 것이라고 예상되는 프로세스들은 상태값을 낮게 유지시킴으로써 해당 프로세스가 상태값으로 인해 종료되지 않도록 우선순위를 낮춘다.

관련 연구로 모바일 장치에서 사용 패턴을 분석하는 연구들이 있다. 사용 패턴을 기반으로 사용될 애플리케이션을 미리 로딩시키는 연구[1]와, 스마트폰 사용의 다양성[2], 스마트폰 사용 패턴 분석[3] 등이 있다.

2. 문제 시나리오

프로세스의 현재 상태값을 참조하여 종료시킬

프로세스를 선정하기 때문에 상태값이 높아진다면 자주 사용되는 프로세스라도 종료될 수 있다. [표 1]는 앵그리버드를 자주 사용하는 시나리오의 애플리케이션 기록으로 그 상황이 발생한다.

표 1 애플리케이션 실행 기록

순서	애플리케이션
1	com.android.chrome
2	com.rovio.angrybirds
3	com.google.android.talk
4	com.rovio.angrybirds
5	com.google.android.talk
6	com.facebook.katana:nodex
7	com.facebook.katana
8	com.rovio.angrybirds
9	com.google.android.apps.maps
10	com.rovio.angrybirds
…	…
22	com.rovio.angrybirds
23	com.google.android.talk
24	com.rovio.angrybirds
25	com.facebook.katana:nodex
26	com.facebook.katana
27	com.rovio.angrybirds
28	com.google.android.talk
29	com.rovio.angrybirds
…	…
40	com.rovio.angrybirds
41	com.facebook.katana:nodex
42	com.facebook.katana
43	com.rovio.angrybirds

앵그리버드는 총 10번 실행되는, 가장 많이, 자주 사용되는 애플리케이션이다. 그런데 10번째와 22번째 사이, 29번째와 40번째 사이에서는 앵그리버드가 사용되지 않고 다른 프로세스들이 많이 사용되었다. 때문에 그때 높은 상태값을 가진 앵그리버드가 종료되었다. 그래서 22번째와 40번째 앵그리버드 실행 시 앵그리버드가 종료되어 있어서 새로 로딩하게 된다. 하지만 그때 앵그리버드는 가장 많이 사용한 애플리케이션으로 앞으로도 사용될 가능성이 높다.

앞으로 사용될 가능성이 높다면 종료시키지 않고 대신 다른 가능성이 낮은 것을 종료시키는 것이 더 로딩을 적게 하여 로딩에 소모되는 비용을 줄일 수 있다. 예를 들어 위 시나리오에서 앵그리버드 대신 다른 한두 번 사용한 프로세스들을 종료시켰다면 앵그리버드를 다시 로딩하지 않아도 됐었다.

3. 프로세스 선택과 보호

자주 사용되는 프로세스가 상태값이 높을 때 종료되어서 자주 로딩 시켜 줘야 하는 문제가 일어날 수 있다. 이를 해결하기 위해 자주 사용된 프로세스들은 상태값을 낮게 유지시켜 해당 문제로부터 안전하게 한다.

3.1. 보호할 프로세스 선택

많이 사용되었으면 앞으로 사용할 가능성이 높다고 보고 가장 많이 사용한 프로세스를 보호한다. 현재 화면을 실행 중인 프로세스가 바뀔 때마다 그 바뀐 프로세스의 카운터를 증가시켜서 저장한다. 또한 카운터는 프로세스가 100번 바뀔 때 마다 반으로 나누어서 오래 전 실행된 프로세스가 계속 보호되지 않게 한다. 그리하여 계산된 프로세스들의 카운터가 가장 큰 3개의 프로세스를 보호한다.

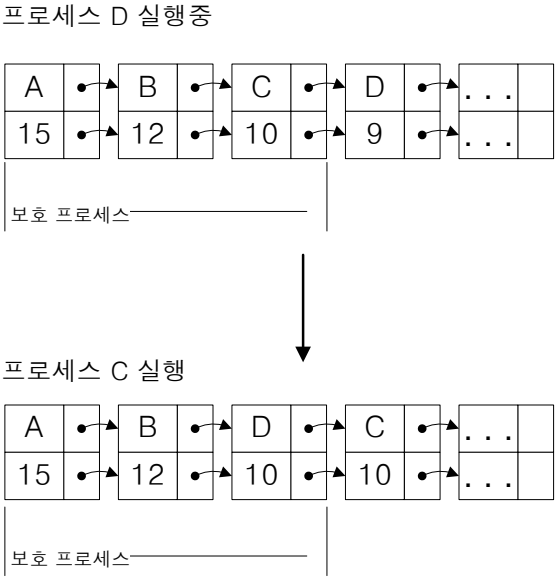

그림 1 프로세스 카운터

[그림 1]은 프로세스 C에서 프로세스 D로 화면 실행 프로세스가 바뀌는 상황으로, 프로세스 D의 카운터가 증가하여 보호 프로세스가 A, B, C에서 A, B, D로 바뀌는 것을 보여 준다.

3.2. 프로세스 보호 방법

프로세스의 maxAdj값을 FOREGROUND 상태값으로 설정하여서 상태값이 그보다 높아지지 않게 하는 방법으로 보호한다. 그리하여 보호되는 프로세스가 상태값이 높아서 종료되는 문제가 발생하지 않는다.

4. 실험

실제 성능 향상을 확인하기 위해 기존 안드로이드와 위 보호 방법을 적용한 안드로이드를 비교 실험하였다.

4.1. 실험 설계

넥서스10, 안드로이드 4.2.2버전에서 진행하였으며 기존 안드로이드와 수정한 안드로이드 모두에서 [표 1]에 나와 있는 시나리오로 실험하여 그 결과를 비교하였다.

4.2. 실험 결과

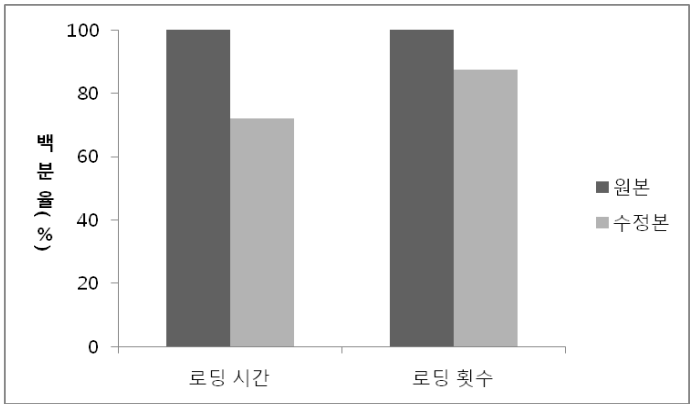

그림 2 실험 결과

표 2 실험 결과

	원본	수정본
로딩 시간	22517ms	16244ms
로딩 횟수	24	21

실험 결과 보호 알고리즘이 잘 적용된 것을 확인하였다. 기존 안드로이드와 달리 앵그리버드가 한번만 로딩되었다. [표 1] 시나리오에서 22번째와 40번째 앵그리버드 실행 시 앵그리버드는 사용 빈도가 높은 프로세스로 카운터 값이 컸기 때문에 종료되지 않았다. 그래서 앵그리버드 실행 시에 추가로 로딩하지 않았고 총 로딩에 소모되는 비용을 줄일 수 있었다.

5. 결론

프로세스의 카운터를 저장하여 앞으로 사용될 것이라 예상되는 프로세스를 보호함으로써 로딩에 소모되는 비용을 줄일 수 있었다. 하지만 단순히 카운터뿐만 아니라 프로세스의 사용 순서나 사용하는 메모리 크기 등을 같이 고려하여 앞으로 사용될 프로세스를 선정한다면 더 좋은 효율을 기대할 수 있을 것이다.

참 고 문 헌

[1]Hokwon Song, et al., “Usage Pattern-Based Prefetching: Quick Application Launch on Mobile Devices”, ICCSA, LNCS 7335PART III, 227-237, 2012

[2] Falaki, Hossein, et al., “Diversity in smartphone usage”, Proceedings of the 8th international conference on Mobile systems, applications, and services, ACM, 2010.

[3] Kang, Joon-Myung, et al., "Usage pattern analysis of smartphones", Network Operations and Management Symposium (APNOMS), 2011 13th Asia-Pacific, IEEE, 2011.