



(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2015-0094166

(43) 공개일자 2015년08월19일

(51) 국제특허분류(Int. Cl.)

G06F 9/45 (2006.01) G06F 9/44 (2006.01)

(21) 출원번호 10-2014-0015114

(22) 출원일자 2014년02월10일

심사청구일자 2014년02월10일

(71) 출원인

고려대학교 산학협력단

서울특별시 성북구 안암로 145, 고려대학교 (안암동5가)

(72) 발명자

김선욱

경기도 남양주시 도농로 34, 214동 2003호 (도농동, 부영그린타운)

조명진

서울특별시 강북구 한천로109길 83, 102동 2002호 (번동, 솔그린아파트)

(74) 대리인

전중학

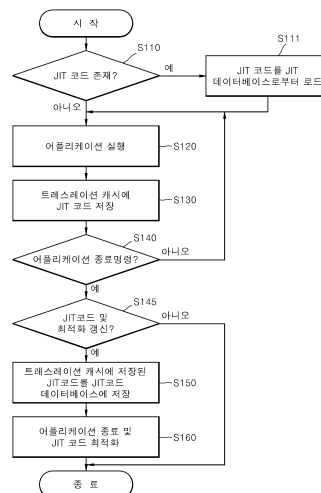
전체 청구항 수 : 총 10 항

(54) 발명의 명칭 어플리케이션 실행 장치 및 방법

(57) 요약

본 발명의 실시예에 따른 어플리케이션 실행 장치는 어플리케이션 실행에 따라 트랜스레이션 캐시에 생성된 핫스팟 트레이스의 JIT(Just in Time) 코드를 상기 어플리케이션 종료시 저장하는 JIT 코드 데이터베이스; 상기 어플리케이션 실행 시 상기 JIT 코드 데이터베이스에 저장된 상기 JIT 코드를 실행 주소에 대응시키는 주소 조정부; 및 상기 핫스팟 트레이스 판단을 위한 초기 프로파일링을 중단시키는 프로파일 설정부;를 포함한다.

대표도 - 도3



이 발명을 지원한 국가연구개발사업

과제고유번호 NRF-2011-0020128

부처명 미래창조과학부

연구관리전문기관 한국연구재단

연구사업명 (이공)기초연구실용성사업

연구과제명 [3차년도] 그린 스마트패치 및 응용 소프트웨어 연구

기 여 율 1/1

주관기관 고려대학교 산학협력단

연구기간 2013.09.01 ~ 2014.08.31

명세서

청구범위

청구항 1

어플리케이션 실행에 따라 트랜스레이션 캐시에 생성된 핫스팟 트레이스의 JIT(Just in Time) 코드를 상기 어플리케이션 종료시 저장하는 JIT 코드 데이터베이스;

상기 어플리케이션 실행 시 상기 JIT 코드 데이터베이스에 저장된 상기 JIT 코드를 실행 주소에 대응시키는 주소 조정부; 및

상기 핫스팟 트레이스 판단을 위한 초기 프로파일링을 중단시키는 프로파일 설정부;를 포함하는 것을 특징으로 하는 어플리케이션 실행 장치.

청구항 2

제1항에 있어서,

상기 JIT 코드 데이터베이스에 저장된 상기 어플리케이션에 따른 상기 JIT 코드를 병합하여 해당 어플리케이션 종료 및 미 실행시 최적화하는 JIT 최적화부;를 더 포함하는 것을 특징으로 하는 어플리케이션 실행 장치.

청구항 3

제1항에 있어서,

상기 어플리케이션 실행시 상기 JIT 코드 데이터베이스에 대응 저장된 상기 JIT 코드가 존재하는 경우, 이를 상기 트랜스레이션 캐시에 전달하고 상기 주소 조정부를 통해 주소 조정하며, 상기 프로파일 설정부를 통해 초기 프로파일링을 중단시키고, 상기 어플리케이션 종료시 상기 트랜스레이션 캐시의 상기 JIT 코드를 상기 JIT 코드 데이터베이스에 저장하는 제어부;를 더 포함하는 것을 특징으로 하는 어플리케이션 실행 장치.

청구항 4

제1항에 있어서,

상기 JIT 코드 메모리는 어플리케이션 프레임워크 모듈에 포함되는 것을 특징으로 하는 어플리케이션 실행 장치.

청구항 5

제1항에 있어서,

상기 JIT 코드는 상기 어플리케이션의 종료 명령에 의해 트랜스레이션 캐시에서 상기 JIT 코드 메모리로 이동되어 저장되는 것을 특징으로 하는 어플리케이션 실행 장치.

청구항 6

어플리케이션에 대응하는 JIT 코드의 존재 여부를 판단하여 상기 JIT 코드가 존재하는 경우 JIT 데이터베이스로부터 상기 JIT 코드를 로드하는 단계;

상기 어플리케이션을 실행하는 단계;

핫스팟 트레이스로 판단되어 새롭게 컴파일된 JIT 코드를 트랜스레이션 캐시에 저장하는 단계; 및

상기 어플리케이션의 종료 명령에 의해 상기 트랜스레이션 캐시에 저장된 상기 JIT 코드를 상기 JIT 코드 데이터베이스에 저장하는 단계;를 포함하는 것을 특징으로 하는 어플리케이션 실행 방법.

청구항 7

제6항에 있어서,

상기 JIT 코드를 JIT 코드 데이터베이스에 저장하는 단계 이후, 상기 어플리케이션을 종료하고, 상기 JIT 코드를 최적화하는 단계를 더 포함하는 것을 특징으로 하는 어플리케이션 실행 방법.

청구항 8

제6항에 있어서,

상기 JIT 코드를 JIT 코드 데이터베이스에 저장하는 단계 이후, 상기 JIT 코드를 최적화하고 상기 어플리케이션을 종료하는 단계를 더 포함하는 것을 특징으로 하는 어플리케이션 실행 방법.

청구항 9

제7항 또는 제8항에 있어서,

상기 JIT 코드를 최적화하는 단계는 상기 JIT 코드 데이터베이스에 저장된 상기 JIT 코드를 병합하여 이루어지는 것을 특징으로 하는 어플리케이션 실행 방법.

청구항 10

제6항에 있어서,

상기 JIT 코드를 트랜스레이션 캐시에 저장하는 단계는 상기 어플리케이션의 트레이스를 수행하는 과정에서 프로파일링을 실행하는 단계; 수집한 상기 트레이스 프로파일 정보를 바탕으로 핫스팟 트레이스를 선택하는 단계; 및 JIT 컴파일러를 통해 핫스팟 트레이스를 컴파일하고 JIT 코드를 저장하는 단계를 포함하는 것을 특징으로 하는 어플리케이션 실행 방법.

발명의 설명

기술 분야

[0001]

본 발명은 기계어 코드인 JIT 코드(Just in Time code)의 저장 및 재사용을 통해 최적화된 어플리케이션 실행 장치 및 방법에 관한 것으로, 보다 자세하게는 안드로이드에서 사용되는 달빅 가상 기계(Dalvik Virtual Machine, VM)에서 해석기(interpreter)에 의해 실행될 때의 가속 기술인 JIT 컴파일러(Just in Time Compiler, JITC)가 생성하는 네이티브(native) 코드를 어플리케이션 종료와 동시에 삭제하지 않고 어플리케이션 데이터베이스에 저장하여 다음 실행 시 재사용하고 어플리케이션이 실행되지 않는 오프라인(offline)에서 최적화를 통해 성능 향상을 도모할 수 있는 어플리케이션 실행 장치 및 방법에 관한 것이다.

배경 기술

[0002]

최근 다양한 기기에서 동일한 동작을 수행하는 응용 프로그램인 어플리케이션을 구현하기 위한 방안으로 가상 머신을 많이 활용하고 있다. 휴대전화나 디지털 TV에서 많이 쓰이는 자바 가상 머신, 자바 스크립트 엔진, 최근

엔 구글에서 개발하여 다양한 스마트폰에 탑재되어 배포되는 달빅 가상 머신까지 많은 영역에서 가상 머신이 활용되고 있다.

[0003] 어플리케이션은 하드웨어 비종속적인 중간 코드 형태로 배포되며, 배포된 어플리케이션은 하드웨어에서 직접 수행되는 것이 아니라 소프트웨어 수행 환경인 가상 머신을 통하여 수행되기 때문에, 다양한 하드웨어 환경에서도 일관적인 수행 환경을 보장할 수 있다. 하지만 중간 코드가 해석기를 통하여 수행되므로 네이티브 코드로 컴파일되어 실행되는 경우에 비해 어플리케이션 수행 성능이 떨어지는 문제점이 있다. 이러한 문제를 해결하기 위하여 흔히 사용되는 방법이 JIT 컴파일러이다.

[0004] JIT 컴파일러는 수행 중에 어플리케이션의 하드웨어 비종속적인 중간 코드를 하드웨어의 기계어 코드로 변환한다. 그리고 변환된 기계어 코드는 하드웨어에서 직접 수행할 수 있기 때문에 매우 빠르게 어플리케이션을 수행할 수 있다.

[0005] 달빅 가상 머신의 JIT 컴파일러 기술은 반복적으로 실행되는 핫스팟(Hot spot) 트레이스(Trace)를 판단하여 트레이스 단위로 코드를 컴파일하여 재사용한다. 이렇게 생성된 JIT 코드는 트랜스레이션 캐시(Translation cache)에 기록되고 어플리케이션이 동작하는 동안 해당 트레이스가 다시 실행될 경우 달빅 가상 머신의 해석기를 사용하지 않고 바로 네이티브 코드로 실행할 수 있다. 그러나 어플리케이션이 종료되면 메모리의 효율적 사용을 위해 트랜스레이션 캐시에 기록된 핫스팟 트레이스를 모두 삭제한다. 따라서 어플리케이션을 재실행하게 되면 처음부터 핫 트레이스를 판단하고 JIT 컴파일러를 통해 JIT 코드를 재생성하는 절차가 반복적으로 이루어짐에 따라 실행 속도가 느려지는 문제가 발생한다.

[0006] 또한, 기존의 가상 머신은 메소드보다 작은 단위인 트레이스를 빠르게 컴파일하여 사용하는 방식으로 기본 블록(Basic block) 단위의 트레이스 생성으로 다음 기본 블록이 다시 핫스팟 트레이스로 판단되어 JIT 코드가 생성되는데, 이러한 트레이스 체인을 병합하여 최적화를 수행하는 기술은 아직 적용되지 못하고 있는 실정이다.

발명의 내용

해결하려는 과제

[0007] 본 발명은 상기의 문제점을 감안하여 이루어진 것으로, 생성된 JIT 코드를 어플리케이션이 실행될 때마다 매번 다시 생성하여 실행 속도가 느려지는 문제를 해결하는 것을 목적으로 한다.

[0008] 또한, 생성된 JIT 코드를 보다 효율적인 코드로 변환하여 최적화된 코드를 실행할 수 있도록 하는 어플리케이션 실행 장치 및 방법을 제공하는 것을 목적으로 한다.

과제의 해결 수단

[0009] 본 발명의 실시예에 따른 어플리케이션 실행 장치는 어플리케이션 실행에 따라 트랜스레이션 캐시에 생성된 핫스팟 트레이스의 JIT(Just in Time) 코드를 상기 어플리케이션 종료시 저장하는 JIT 코드 데이터베이스; 상기 어플리케이션 실행 시 상기 JIT 코드 데이터베이스에 저장된 상기 JIT 코드를 실행 주소에 대응시키는 주소 조정부; 및 상기 핫스팟 트레이스 판단을 위한 초기 프로파일링을 중단시키는 프로파일 설정부;를 포함한다.

발명의 효과

[0010] 본 발명의 실시예에 따른 어플리케이션 실행 장치 및 방법에 따르면, JIT 코드를 기록 매체에 저장하여 어플리케이션의 재시작시 JIT 코드를 다시 생성하지 않고 재사용함에 따라 실행 속도가 향상되는 효과가 있다.

[0011] 또한 생성된 JIT 코드를 보다 효율적인 코드로 변환하여 최적화된 코드를 실행할 수 있는 효과가 있다.

도면의 간단한 설명

[0012] 도 1은 본 발명의 실시예에 따른 소프트웨어 스택의 구성을 나타낸 블록도이다.

도 2는 본 발명의 실시예에 따른 JIT 코드 로딩부의 구성을 나타낸 블록도이다.

도 3은 본 발명의 실시예에 따른 JIT 코드의 재사용 및 최적화 방법을 나타낸 순서도이다.

도 4는 본 발명의 실시예에 따른 JIT 코드의 생성 및 저장을 나타낸 순서도이다.

발명을 실시하기 위한 구체적인 내용

- [0013] 본 발명에서 사용되는 기술적 용어는 단지 특정한 실시 예를 설명하기 위해 사용된 것으로, 본 발명을 한정하려는 의도가 아님을 유의해야 한다. 또한, 본 발명에서 사용되는 기술적 용어는 본 발명에서 특별히 다른 의미로 정의되지 않는 한, 본 발명이 속하는 기술 분야에서 통상의 지식을 가진 자에 의해 일반적으로 이해되는 의미로 해석되어야 하며, 과도하게 포괄적인 의미로 해석되거나, 과도하게 축소된 의미로 해석되지 않아야 한다. 또한, 본 발명에서 사용되는 기술적인 용어가 본 발명의 사상을 정확하게 표현하지 못하는 잘못된 기술적 용어일 때에는, 당업자가 올바르게 이해할 수 있는 기술적 용어로 대체되어 이해되어야 할 것이다. 또한, 본 발명에서 사용되는 일반적인 용어는 사전에 정의되어 있는 바에 따라, 또는 전후 문맥상에 따라 해석되어야 하며, 과도하게 축소된 의미로 해석되지 않아야 한다.
- [0014] 또한, 본 발명에서 사용되는 단수의 표현은 문맥상 명백하게 다르게 뜻하지 않는 한 복수의 표현을 포함한다. 본 발명에서, "구성된다" 또는 "포함한다" 등의 용어는 발명에 기재된 여러 구성 요소들, 또는 여러 단계를 반드시 모두 포함하는 것으로 해석되지 않아야 하며, 그 중 일부 구성 요소들 또는 일부 단계들은 포함되지 않을 수도 있고, 또는 추가적인 구성 요소 또는 단계들을 더 포함할 수 있는 것으로 해석되어야 한다.
- [0015] 또한, 본 발명에서 사용되는 제1, 제2 등과 같이 서수를 포함하는 용어는 구성 요소들을 설명하는데 사용될 수 있지만, 구성 요소들은 용어들에 의해 한정되어서는 안 된다. 용어들은 하나의 구성 요소를 다른 구성 요소로부터 구별하는 목적으로만 사용된다. 예를 들어, 본 발명의 권리 범위를 벗어나지 않으면서 제1 구성 요소는 제2 구성 요소로 명명될 수 있고, 유사하게 제2 구성 요소도 제1 구성 요소로 명명될 수 있다.
- [0016] 이하, 첨부된 도면을 참조하여 본 발명에 따른 바람직한 실시 예를 상세히 설명하되, 도면 부호에 관계없이 동일하거나 유사한 구성 요소는 동일한 참조 번호를 부여하고 이에 대한 중복되는 설명은 생략하기로 한다.
- [0017] 또한, 본 발명을 설명함에 있어서 관련된 공지 기술에 대한 구체적인 설명이 본 발명의 요지를 흐릴 수 있다고 판단되는 경우 그 상세한 설명을 생략한다. 또한, 첨부된 도면은 본 발명의 사상을 쉽게 이해할 수 있도록 하기 위한 것일 뿐, 첨부된 도면에 의해 본 발명의 사상이 제한되는 것으로 해석되어서는 아니 됨을 유의해야 한다.
- [0018] 본 발명은 2013년도 정부(미래과학창조부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구이다(No.NRF-2011-0020128).
- [0019] 도 1은 본 발명의 실시예에 따른 소프트웨어 스택의 구성을 나타낸 블록도이고, 도 2는 본 발명의 실시예에 따른 JIT 코드 로딩부의 구성을 나타낸 블록도이다. 도 1을 참조하면, 단말장치에 포함되는 소프트웨어 스택은 각 계층별로 어플리케이션을 포함하는 어플리케이션 모듈(100), 어플리케이션 프레임워크 모듈(200), 네이티브 라이브러리 모듈(300) 및 포팅 모듈(400)을 포함한다.
- [0020] 단말장치는 일반적으로 휴대폰, MP3 등의 모바일 장치를 의미하지만, 그 밖에도 모든 정보처리 장치를 포함할 수 있다.
- [0021] 각 모듈은 아래쪽으로 갈수록 물리계층에 밀접해지는데, 최하단의 포팅 모듈(400)은 범용 운영체제(미도시)와 연결된다.
- [0022] 어플리케이션 모듈(100)은 단말장치의 플랫폼에서 바로 실행 가능한 네이티브 어플리케이션과, 가상 머신이 구비된 미들웨어를 통해 실행되는 어플리케이션을 포함한다.
- [0023] 어플리케이션 프레임워크 모듈(200)은 구글 플랫폼에서 어플리케이션 프레임워크 모듈을 수정하거나 재사용하여 사용할 수 있다. 구글의 어플리케이션 프레임워크 모듈에는 어플리케이션을 실행시키는데 필요한 클래스 라이브러리와 서비스들이 구비되어 있다. 이때, 일부 서비스는 어플리케이션의 호환에 필요가 없으므로 경량화를 위해 제거될 수 있다.
- [0024] 어플리케이션 프레임워크 모듈(200)은 전화 서비스, 위치 서비스, 블루투스 서비스, 무선랜 서비스 중 하나 이상의 서비스 모듈을 포함하여 어플리케이션의 실행에 따라 해당 서비스를 백그라운드로 실행시킬 수 있다.
- [0025] 어플리케이션 프레임워크 모듈(200)에는 JIT 코드 메모리(250)가 포함될 수 있다. JIT 코드 메모리(250)는 어플

리케이션 실행 시, JIT(Just in Time) 코드를 로드하는 기능을 수행한다.

- [0026] 도 2에 도시된 바와 같이, JIT 코드 메모리(250)는 JIT 코드 데이터베이스(251), JIT 최적화부(252), 주소 조정부(253), 프로파일 설정부(254) 및 제어부(255)를 포함한다.
- [0027] JIT 코드 데이터베이스(251)는 JIT 코드를 저장하는 기능을 수행한다. 일반적으로 어플리케이션 수행 중에 JIT 컴파일러 과정이 일어나므로 JIT 컴파일러의 오버헤드를 줄이는 것이 중요하기 때문에 어플리케이션의 모든 중간 코드를 기계어로 변환하지 않고, 자주 수행되며 JIT 컴파일러에 의한 효과가 클 것으로 예상되는 중간 코드를 선택적으로 컴파일하는 것이 바람직하다.
- [0028] JIT 컴파일러에서 이러한 선택의 단위는 메소드(method)(또는 function) 혹은 트레이스(trace)(코드 sequence)가 되며, 핫스팟(Hot Spot)인 메소드나 트레이스를 정확히 찾아내는 것이 중요하다.
- [0029] 일반적인 JIT 컴파일러는 어플리케이션을 중간 코드 트레이스 단위로 나누어서 각각의 트레이스가 몇 번 수행되는지 확인한 후에, 트레이스가 미리 정해진 특정 임계값 이상 수행되면 핫스팟 트레이스로 결정하고 기계어로 컴파일을 수행하여 JIT 코드를 생성한다.
- [0030] 이러한 과정에 의해 생성된 JIT 코드는 트랜스레이션 캐시에 기록되고 어플리케이션이 동작하는 동안 해당 트레이스가 다시 실행될 경우 달빅 가상 머신의 해석기를 사용하지 않고 바로 네이티브 코드로 실행될 수 있다.
- [0031] 어플리케이션이 종료되면 제한된 모바일 환경에서 메모리의 효율적 사용을 위해 트랜스레이션 캐시에 기록된 핫 트레이스를 모두 삭제하므로 어플리케이션을 다시 실행하게 되면 처음부터 핫 트레이스를 판단하고 다시 JIT 컴파일러를 통해 JIT 코드를 생성하는 절차가 반복적으로 이루어짐에 따라 실행 속도가 느려지는 문제가 발생한다.
- [0032] 본 발명에서는 이를 개선하기 위해 JIT 컴파일러를 통해 생성된 JIT 코드를 삭제하지 않고 JIT 코드 데이터베이스(251)에 저장한다. JIT 코드 데이터베이스(251)는 JIT 코드를 저장하며, 어플리케이션의 반복 실행에 따라 새로 생성되는 JIT 코드를 저장한다.
- [0033] JIT 최적화부(252)는 어플리케이션이 종료될 때나 종료된 후 시스템의 여유 자원을 이용하여 생성된 JIT 코드를 백그라운드(background)에서 병합하여 최적화한다.
- [0034] 기존의 JIT 컴파일러는 프로그램이 실행되고 있는 동안에 컴파일되므로 최적화에 제한적이므로 JIT 최적화부(252)는 어플리케이션을 백그라운드로 실행시킴으로써 JIT 코드 데이터베이스(251)에 기록된 트레이스 체인(Trace chain)을 분석하고, 이를 병합하여 보다 큰 단위인 슈퍼 트레이스(super trace)를 생성하여 컴파일러를 최적화한다. 이렇게 생성된 최적화된 코드는 어플리케이션의 시작과 동시에 트랜스레이션 캐시에 저장되어 성능 향상이 이루어질 수 있게 된다.
- [0035] 주소 조정부(253)는 기존에 생성된 JIT 코드의 주소와 현재 실행되고 있는 어플리케이션의 주소를 조정하여 일치시킨다. JIT 코드가 컴파일되면 메모리에 주소가 할당되는데, 어플리케이션이 실행될 때마다 동일한 주소가 사용되는 것이 아니므로 생성된 코드를 재사용하기 위해서는 주소 조정부(253)에 의해 기존에 생성된 JIT 코드의 주소와 현재 실행되고 있는 어플리케이션의 주소가 조정된다. 생성된 JIT 코드의 메모리 주소는 메타데이터에 저장될 수 있다.
- [0036] 프로파일 설정부(254)는 핫스팟 트레이스의 판단을 위한 초기 프로파일링을 중단시키는 기능을 수행하고, 제어부(255)는 어플리케이션 실행시 JIT 코드 데이터베이스(251)에 대응 저장된 JIT 코드가 존재하는 경우, 이를 트랜스레이션 캐시(260)에 전달하고 주소 조정부(253)를 통해 주소 조정하며, 프로파일 설정부(254)를 통해 초기 프로파일링을 중단시키고, 어플리케이션 종료시 트랜스레이션 캐시(260)의 JIT 코드를 JIT 코드 데이터베이스(251)에 저장하는 기능을 수행한다.
- [0037] 네이티브 라이브러리 모듈(300)은 가상머신(311)을 포함하는 런타임엔진(310)과 어플리케이션 프레임워크 모듈의 라이브러리를 구현하는데 기반이 되는 네이티브 라이브러리(320)를 구비한다. 이때, 네이티브 라이브러리 모듈(300)은 포팅 모듈과 연동하여 동작하는 하나 이상의 네이티브 라이브러리(320)를 구비할 수 있다.
- [0038] 가상머신(311)은 클래스 라이브러리와 자바 API를 통해 어플리케이션을 동작시키고, 동적 라이브러리 관리 모듈, 외부 함수 인터페이스, 동기화 모듈을 포함하며, 포팅 모듈과 연동하여 어플리케이션이 실행되도록 한다. 이때, 가상머신은 달빅(Dalvik) 가상머신일 수 있으며, 달빅 가상머신에서 리눅스 의존적인 모듈은 제거되고, 범용 운영체제에서 제공하지 않는 모듈로서 스레드 관리 모듈(pthread), 동적 라이브러리 관리 모듈(shared

object, DLL), 외부 함수 인터페이스(FFI: Foreign Function Interface), 동기화 모듈, 메모리 관리 모듈의 적어도 하나 이상을 추가 재구성할 수 있다.

- [0039] 또한, 가상머신(311)은 에뮬레이터(emulator)를 구비하여, 어플리케이션이나 가상머신에서 멀티 태스크가 요청되는 경우 싱글 태스크로 처리되도록 할 수 있다.
- [0040] 네이티브 라이브러리(320)는 프레임워크 모듈(200)을 구현하는데 필요한 각종의 네이티브 모듈을 제공하는데, 바람직하게는 공개 소스 기반의 소프트웨어를 사용하여 구성할 수 있다.
- [0041] 포팅 모듈(400)은 네이티브 라이브러리 모듈(300)과 범용 운영체제를 연결하며, 범용 운영체제를 통해 단말의 하드웨어를 관리하는 기능을 제공한다. 즉, 포팅 계층부(400)는 타 운영체제에 쉽게 이식할 수 있도록 운영체제 의존적인 코드를 내부에서 처리하여 상위 계층으로는 포팅 API(예: 리눅스 표준 API)를 제공하는 계층이다.
- [0042] 포팅 모듈(400)은 파일 시스템 모듈, 메모리 할당 모듈, 네트워크 모듈, 기본 라이브러리 모듈(stdio, stdlib, math 등), 타이머 모듈, 디바이스 제어 모듈(디스플레이, 미디어, 입력장치, 3D 등), 기타 유틸리티 등의 하드웨어 관리 모듈을 포함한다. 그리고, 포팅 계층부(400)는 하드웨어 관리 모듈을 통해 단말장치의 하드웨어에 접근하여 관리 및 제어한다. 예컨대, 포팅 모듈(400)에는 Kernel (DLL, pthread), System(file, memory, socket, stdio, stdlib, string, math, time), Device(framebuffer, media, input, 3D), Utils(log, debug) 등이 구비된다.
- [0043] 도 3은 본 발명의 실시예에 따른 JIT 코드의 재사용 및 최적화 방법을 나타낸 순서도이다. 도시된 바와 같이 JIT 코드를 재사용 및 최적화하는 방법은 JIT 코드의 존재 여부를 판단하는 단계(S110), JIT 코드가 존재하는 경우, JIT 데이터베이스로부터 JIT 코드를 로드하는 단계(S111), 어플리케이션을 실행하는 단계(S120), 트랜스레이션 캐시에 JIT 코드를 저장하는 단계(S130), 어플리케이션 종료명령이 입력되면(S140), 트랜스레이션 캐시에 저장된 JIT 코드를 JIT 코드 데이터베이스에 저장하는 단계(S150), 어플리케이션 종료 및 JIT 코드를 최적화하는 단계(S160)를 포함한다.
- [0044] JIT 코드의 존재 여부를 판단하는 단계(S110)에서는 어플리케이션 실행 명령이 입력되는 경우, 핫스팟 트레이스로 판단되어 컴파일된 JIT 코드가 JIT 데이터베이스에 존재하는지를 판단한다. 어플리케이션이 처음 실행되거나, 이전 어플리케이션의 실행에서 핫스팟 트레이스로 판단된 트레이스가 없는 경우, 컴파일된 JIT 코드가 JIT 데이터베이스에 존재하지 않게 된다.
- [0045] JIT 코드가 존재하는 경우, JIT 데이터베이스로부터 JIT 코드를 로드하게 된다(S111). 이에 따라 핫스팟 트레이스로 판단된 트레이스에 대해 JIT 컴파일러가 다시 JIT 코드를 생성하는 것을 방지할 수 있게 된다.
- [0046] 다음으로, 어플리케이션을 실행하게 된다(S120). 어플리케이션을 실행함에 있어서, 핫스팟 트레이스에 관한 정보를 포함하는 JIT 코드가 존재하는 트레이스에 대해서는 프로파일링 하지 않는다. 이를 위해 기존의 JIT 메카니즘에서 사용했던 프로파일 카운트값을 기존의 임계값(threshold) 이상으로 조정한다.
- [0047] 다음으로, 트랜스레이션 캐시에 JIT 코드를 저장한다(S130). 트랜스레이션 캐시에 저장되는 JIT 코드는 핫스팟 트레이스로 판단되어 컴파일된 JIT 코드이며, 트랜스레이션 캐시에 저장되는 과정에 대해서는 도 4를 참조하여 상세히 후술한다.
- [0048] 다음으로, 어플리케이션 종료명령이 입력되면(S140), JIT 코드 및 코드의 최적화 갱신이 필요한지를 판단한다(S145). JIT 코드 및 코드의 최적화 갱신은 선택적인 것으로 S130의 단계에서 트랜스레이션 캐시에 저장된 JIT 코드가 갱신되거나 최적화가 필요한 경우에는 트랜스레이션 캐시에 저장된 JIT 코드를 JIT 코드 데이터베이스에 저장한다(S150). 한편, JIT 코드 및 코드의 최적화 갱신이 필요하지 않거나 메모리 공간 등의 리소스 부족으로 JIT 코드를 저장할 수 없는 경우에는 바로 어플리케이션을 종료한다.
- [0049] 트랜스레이션 캐시에 저장된 JIT 코드는 어플리케이션 종료명령이 입력되기 전까지 핫스팟 트레이스로 판단되어 컴파일된 JIT 코드이며, 이는 어플리케이션 종료명령에 의해 JIT 코드 데이터베이스에 이동되어 저장된다. 따라서, 어플리케이션이 종료되어도, 핫스팟 트레이스로 판단되어 컴파일된 JIT 코드는 삭제되지 않고 다음 어플리케이션을 실행할 때 재사용된다. 이에 따라 JIT 컴파일러가 JIT 코드를 재생성하는 비용, 시간을 절감할 수 있게 된다.
- [0050] 다음으로, 어플리케이션을 종료하거나 종료후에 백그라운드로 실행시킴으로써 JIT 코드를 최적화한다(S160). JIT 최적화는 JIT 코드 데이터베이스(251)에 기록된 트레이스 체인(Trace chain)을 분석하고 보다 큰 단위인 슈퍼 트레이스(super trace)를 생성하여 컴파일러를 최적화한다. 여유 자원을 통해 오프라인에서 최적화를 수행하

기 때문에 최적화에 의한 어플리케이션 성능 저하가 발생하지 않으며, 시간제약이 없기 때문에 실행 시간에 빠르게 생성했던 코드보다 훨씬 효율적인 코드 생성이 가능하다. 이렇게 생성된 최적화된 코드는 어플리케이션의 시작과 함께 로딩되어 보다 큰 성능 향상을 얻을 수 있게 된다.

[0051] 도 4는 본 발명의 실시예에 따른 JIT 코드의 생성 및 저장을 나타낸 순서도이다. 도시된 바와 같이 JIT 코드의 생성 및 저장 방법은 트레이스를 수행하는 과정에서 프로파일링을 실행하는 단계(S131), 수집한 트레이스 프로파일 정보를 바탕으로 핫스팟 트레이스를 선택하는 단계(S132) 및 JIT 컴파일러를 통해 핫스팟 트레이스를 컴파일하고 JIT 코드를 저장하는 단계(S133)를 포함한다.

[0052] 프로파일링을 실행하는 단계(S131)는 트레이스가 한번 수행될 때마다 트레이스에 해당하는 카운터를 하나씩 증가시키는 방식을 사용할 수 있다. 기본적으로 가상 머신에는 트레이스를 위한 자료구조를 유지하고 있으므로, 이러한 자료구조 내에 카운터 변수를 추가함으로써 트레이스의 수행 횟수를 카운트할 수 있다.

[0053] 핫스팟 트레이스를 선택하는 단계(S132)에서는 상기 프로파일링 방법으로 트레이스 수행 시간에 프로파일링을 진행하여 트레이스의 수행 횟수가 일정 임계값을 초과하는 경우에는 해당 트레이스를 핫스팟 트레이스로 판단할 수 있다. 모든 트레이스에 대한 프로파일링을 원칙으로 하나, 메모리 공간 사용을 절감하고 오버헤드를 줄이기 위해 핫스팟 트레이스로 판단된 트레이스에 대해서는 프로파일링을 실행하지 않을 수 있다.

[0054] 핫스팟 트레이스를 컴파일하고 JIT 코드를 저장하는 단계(S133)에서는 JIT 컴파일러를 통해 핫스팟 트레이스를 컴파일(Compile)하게 되며, 컴파일이 종료되면 컴파일된 트레이스에 대해서는 다시 프로파일링을 수행하지 않고 트랜스레이션 캐시에 JIT 코드를 저장하게 된다.

[0055] 상기에서 기술한 바와 같이 JIT 컴파일러를 통해 생성된 JIT 코드를 삭제하지 않고 JIT 코드 데이터베이스(251)에 저장하고, 어플리케이션의 반복 실행으로 새로 생성되는 JIT 코드를 저장함에 따라 어플리케이션의 이전 실행에서 핫스팟 트레이스로 판단된 JIT 코드를 재생성하지 않게 되어 속도가 향상된다.

[0056] 또한, 어플리케이션이 종료된 후에도 슈퍼 트레이스 생성을 통해 최적화된 코드를 실행할 수 있는 효과가 있다.

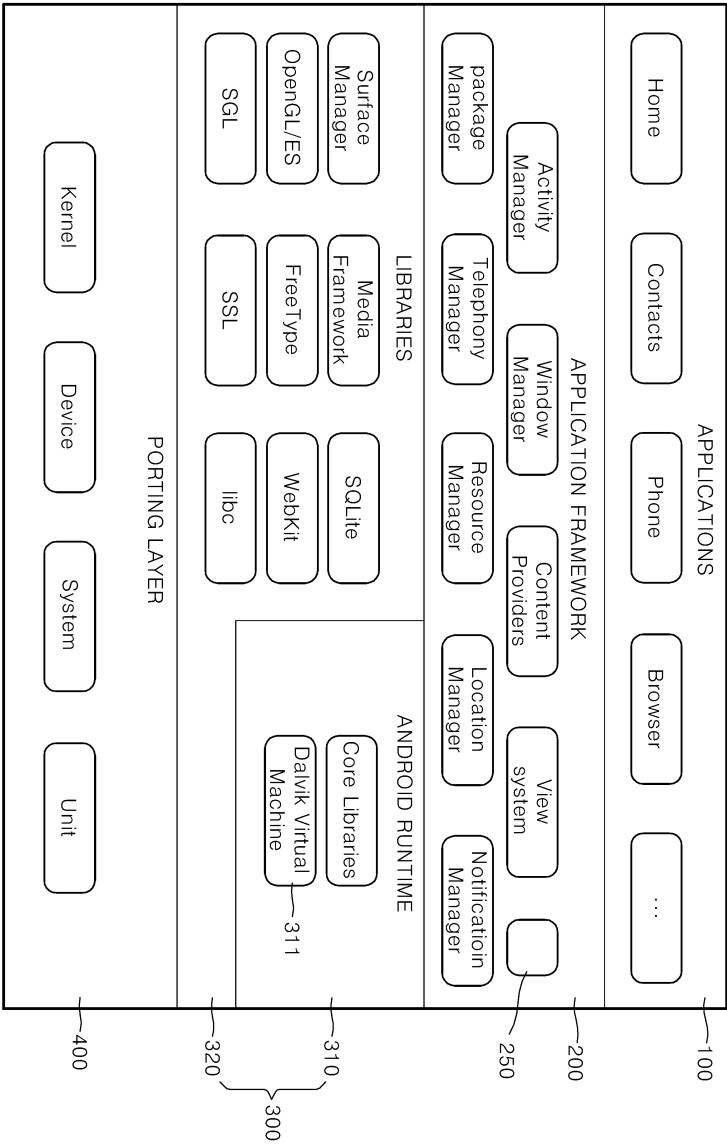
[0057] 전술한 내용은 본 발명이 속하는 기술 분야에서 통상의 지식을 가진 자라면 본 발명의 본질적인 특성에서 벗어나지 않는 범위에서 수정 및 변형이 가능할 것이다. 따라서, 본 발명에 개시된 실시예들은 본 발명의 기술 사상을 한정하기 위한 것이 아니라 설명하기 위한 것이고, 이러한 실시예에 의하여 본 발명의 기술 사상의 범위가 한정되는 것은 아니다. 본 발명의 보호 범위는 아래의 청구범위에 의하여 해석되어야 하며, 그와 동등한 범위 내에 있는 모든 기술 사상은 본 발명의 권리범위에 포함되는 것으로 해석되어야 할 것이다.

부호의 설명

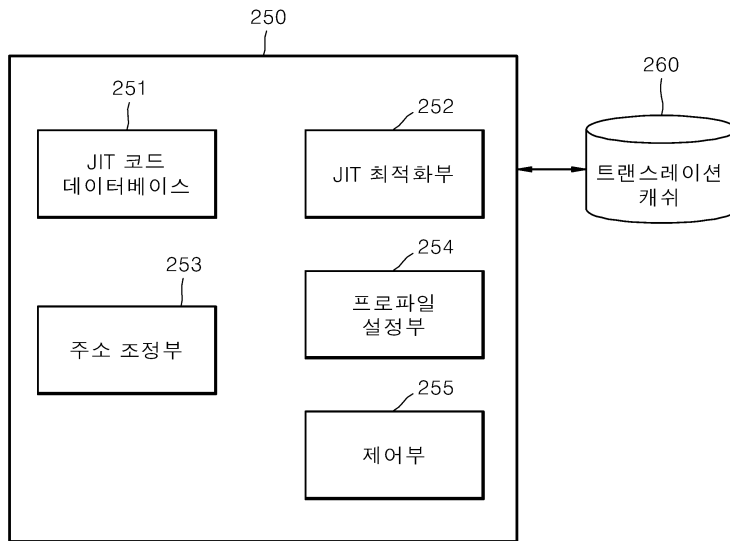
- [0058] 100 어플리케이션 모듈
200 어플리케이션 프레임워크 모듈
250 JIT 코드 메모리
300 네이티브 라이브러리 모듈
400 포팅 모듈

도면

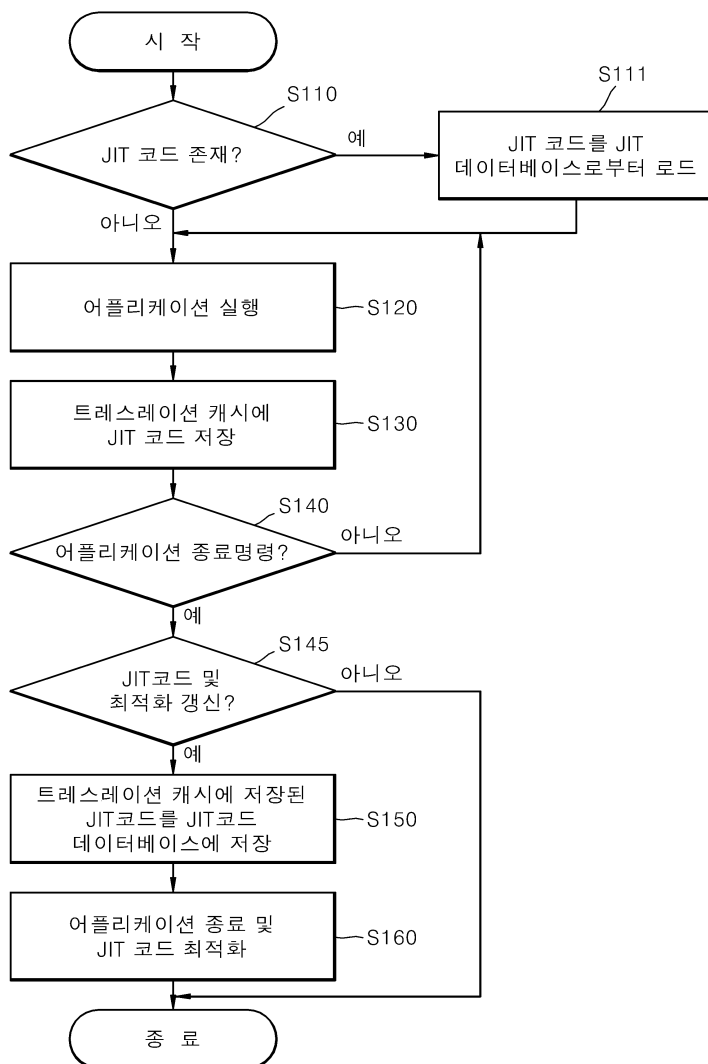
도면1



도면2



도면3



도면4

