(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2019년03월07일
(11) 등록번호 10-1955737
(24) 등록일자 2019년02월28일

(51) 국제특허분류(Int. Cl.)
G06F 9/06 (2018.01) *G06F 12/00* (2016.01)
G06F 9/44 (2018.01)
(21) 출원번호 10-2013-7033916
(22) 출원일자(국제) 2012년06월18일
심사청구일자 2017년05월31일
(85) 번역문제출일자 2013년12월19일
(65) 공개번호 10-2014-0035416
(43) 공개일자 2014년03월21일
(86) 국제출원번호 PCT/US2012/043036
(87) 국제공개번호 WO 2012/177579
국제공개일자 2012년12월27일
(30) 우선권주장
13/163,741 2011년06월20일 미국(US)
(56) 선행기술조사문헌
KR1020060033606 A*
(뒷면에 계속)

(73) 특허권자
마이크로소프트 테크놀로지 라이센싱, 엘엘씨
미국 워싱턴주 (우편번호 : 98052) 레드몬드 원 마이크로소프트 웨이
(72) 발명자
마일렛 스티븐
미국 워싱턴주 98052-6399 레드몬드 원 마이크로소프트 웨이 엘씨에이 - 인터내셔널 페이턴츠 마이크로소프트 코포레이션
홀 마이클
미국 워싱턴주 98052-6399 레드몬드 원 마이크로소프트 웨이 엘씨에이 - 인터내셔널 페이턴츠 마이크로소프트 코포레이션
(뒷면에 계속)
(74) 대리인
제일특허법인(유)

전체 청구항 수 : 총 19 항 심사관 : 유진태

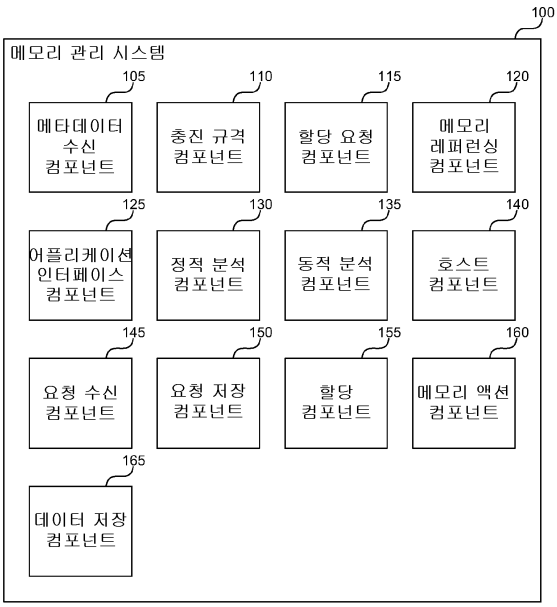
(54) 발명의 명칭 **강화된 어플리케이션 메타데이터를 갖는 메모리 매니저**

(57) 요 약

메모리가 어떻게 사용되고 있는지를 나타내는 정보를 어플리케이션들로부터 수신하고 또한 어플리케이션 호스트로 하여금 메모리를 사용하기 위한 어플리케이션 요청들에 대해 더 많은 제어를 행사하게 하는 메모리 관리 시스템이 본 명세서에서 설명된다. 그 시스템은, 어플리케이션으로 하여금 더 나중에 메모리를 관리하는 것에 유용

(뒷면에 계속)

대 표 도 - 도1



한 메모리 할당들에 관한 더 많은 정보를 명시하게 하는 어플리케이션 메모리 관리 어플리케이션-프로그래밍 인터페이스(API)를 제공한다. 그 시스템은 또한, 더 효과적인 메모리 관리에 참여하기 위한 일부 능력을 그 시스템으로 작업하도록 수정되지 않는 어플리케이션들을 제공하기 위해 레거시 어플리케이션들을 정적으로 및/또는 동적으로 분석하기 위한 능력을 제공한다. 그 시스템은, 어플리케이션들에 의해 제공된 정보를 레버리징하고 또한 정보를 이용하여 메모리를 더 효과적으로 관리하기 위한 변화들을 어플리케이션 호스트에 제공하며, 어플리케이션의 메모리 사용으로 후킹한다. 따라서, 그 시스템은, 어플리케이션 호스트 거동을 개선시키고 어플리케이션들로 하여금 컴퓨팅 리소스들을 더 효율적으로 사용하게 하는, 메모리를 관리하기 위한 새로운 모델을 제공한다.

(72) 발명자

라루스 제임스

미국 워싱턴주 98052-6399 레드몬드 원 마이크로소프트 웨이 엘씨에이 - 인터내셔널 페이턴츠 마이크로소프트 코포레이션

스프라들린 제레미아 씨

미국 워싱턴주 98052-6399 레드몬드 원 마이크로소프트 웨이 엘씨에이 - 인터내셔널 페이턴츠 마이크로소프트 코포레이션

(56) 선행기술조사문헌

US20080168235 A1*

JP2010231684 A

KR1020060010679 A

KR1020090131142 A

KR1020100087138 A

US20090113444 A1

*는 심사관에 의하여 인용된 문헌

명 세 서

청구범위

청구항 1

어플리케이션으로부터 메모리 우선순위 정보를 수신하는 어플리케이션 호스트에서 메모리 관리를 핸들링하는 컴퓨터-구현된 방법으로서,

복수의 메모리 할당 요청, 및 메모리 할당이 하나 이상의 어플리케이션에 의해 어떻게 사용되는지를 나타내는 정보를 어플리케이션 호스트에서 수신하는 단계와,

상기 어플리케이션 호스트 상에서 어플리케이션을 계속해서 효율적으로 실행시키기 위해 액션을 취할 필요성을 나타내는 적어도 하나의 메모리 상태를 검출하는 단계와,

액션을 취할 할당을 결정하기 위해, 상기 수신된 할당 요청을 열거하는 단계와,

열거된 할당 중 액션을 취할 하나 이상의 할당을 선택하는 단계와,

상기 검출된 메모리 상태를 개선(alleviate)시키기 위해, 상기 선택된 할당에 대해 액션을 수행하는 단계

를 포함하고,

상기 단계들은 적어도 하나의 프로세서에 의해 수행되는

메모리 관리를 핸들링하는 컴퓨터-구현된 방법.

청구항 2

제 1 항에 있어서,

상기 정보를 어플리케이션 호스트에서 수신하는 단계는, 상기 요청이 상기 어플리케이션 호스트에 의해 어플리케이션으로부터 수신될 때 저장되는 할당들의 리스트 또는 다른 데이터 구조에 액세스하는 단계를 포함하는

메모리 관리를 핸들링하는 컴퓨터-구현된 방법.

청구항 3

제 1 항에 있어서,

상기 정보를 어플리케이션 호스트에서 수신하는 단계는, 이용가능한 메모리를 효율적으로 관리하기 위해 어느 할당이 해제되거나, 페이징되거나, 또는 핸들링될 수 있는지를 상기 어플리케이션 호스트가 결정하는 것을 돕는 우선순위 또는 다른 정보에 액세스하는 단계를 포함하는

메모리 관리를 핸들링하는 컴퓨터-구현된 방법.

청구항 4

제 1 항에 있어서,

상기 메모리 상태를 검출하는 단계는, 어플리케이션이 실행에 필요한 메모리가 부족하다고 결정하는 단계와, 다른 실행 어플리케이션으로부터 메모리를 취하여 상기 어플리케이션에 제공하는 단계를 포함하는

메모리 관리를 핸들링하는 컴퓨터-구현된 방법.

청구항 5

제 1 항에 있어서,

상기 메모리 상태를 검출하는 단계는, 당면한 어플리케이션 메모리 요청이 충족될 수 없음을 결정하는 단계와, 상기 호스트가 상기 요청을 수신할 때까지 상기 어플리케이션의 요청을 충족시키도록 더 많은 메모리를 선제적으로 이용가능하게 하는 단계를 포함하는

메모리 관리를 핸들링하는 컴퓨터-구현된 방법.

청구항 6

제 1 항에 있어서,

상기 요청을 열거하는 단계는, 할당의 리스트 또는 다른 데이터 구조를 검사(traverse)하여, 해제될 필요가 있을 경우 용이하게 복원될 수 있는 할당을 결정하는 단계를 포함하는

메모리 관리를 핸들링하는 컴퓨터-구현된 방법.

청구항 7

제 1 항에 있어서,

상기 요청을 열거하는 단계는, 할당의 리스트 또는 다른 데이터 구조를 검사하여, 어플리케이션이 바로 사용할 가능성이 없는 할당을 결정하는 단계를 포함하는

메모리 관리를 핸들링하는 컴퓨터-구현된 방법.

청구항 8

제 1 항에 있어서,

상기 할당을 선택하는 단계는, 상기 할당을 요청하였던 상기 어플리케이션에 의해 상기 할당이 어떻게 사용되는지를 나타내는 상기 수신된 정보에 기초하여 할당을 선택하는 단계를 포함하는

메모리 관리를 핸들링하는 컴퓨터-구현된 방법.

청구항 9

제 1 항에 있어서,

상기 할당을 선택하는 단계는, 메모리의 특정 총 사이즈에 대해 액션을 취할 복수의 할당을 선택하는 단계와, 상기 사이즈까지 또는 상기 사이즈를 초과하는 할당을 선택하는 단계를 포함하는

메모리 관리를 핸들링하는 컴퓨터-구현된 방법.

청구항 10

제 1 항에 있어서,

상기 액션을 수행하는 단계는, 이전에 할당된 메모리를 해제하는 단계 또는 메모리 콘텐츠를 디스크 또는 다른 저장부로 스와핑하는 단계를 포함하는

메모리 관리를 핸들링하는 컴퓨터-구현된 방법.

청구항 11

제 1 항에 있어서,

상기 액션을 수행하는 단계는, 상기 액션을 상기 어플리케이션에게 통지하여, 상기 어플리케이션이 상기 할당에 의존하는 거동을 수정할 수 있게 하는 단계를 포함하는

메모리 관리를 핸들링하는 컴퓨터-구현된 방법.

청구항 12

호스트 상에서 실행되는 어플리케이션으로부터 수신된 할당 메타데이터를 사용하여 상기 호스트 내에서 메모리를 효율적으로 관리하는 컴퓨터 시스템으로서,

하기의 컴포넌트들 내에서 구현되는 소프트웨어 명령어를 실행하도록 구성된 프로세서 및 메모리와,

하나 이상의 어플리케이션과 상기 호스트 간에 통신 인터페이스를 제공하여 메모리 할당의 사용을 협상하는 어플리케이션 인터페이스 컴포넌트와,

어플리케이션이 실행되는 환경을 포함하고 상기 시스템에 의해 제공되는 메모리 관리로의 액세스를 제공하는 호스트 컴포넌트와,

메모리를 할당하기 위한 어플리케이션으로부터의 요청을 수신하는 요청 수신 컴포넌트 - 각각의 요청은, 상기 어플리케이션이 할당된 메모리를 어떻게 사용하는지를 나타내는 정보를 상기 호스트에 제공하는 메타데이터를 포함함 - 와,

메모리 관리 결정을 행하는 동안 후속 사용을 위해 수신된 요청 및 관련 메타데이터 정보를 저장하는 요청 저장 컴포넌트와,

메모리를 할당하기 위한 실제 액션을 수행하여 하나 이상의 수신된 어플리케이션 할당 요청을 충족시키는 할당 컴포넌트와,

디바이스 상의 메모리를 관리하기 위한 액션을 수행하고 이전에 저장된 수신 요청 정보에 액세스하여 상기 수행된 액션에 의해 영향을 받을 하나 이상의 적절한 메모리 할당을 결정하는 메모리 액션 컴포넌트를 포함하는

컴퓨터 시스템.

청구항 13

제 12 항에 있어서,

상기 어플리케이션 인터페이스 컴포넌트는, 상기 호스트가 상기 어플리케이션에게 제공하는 어플리케이션 프로그래밍 인터페이스(API) - 상기 API를 통해 상기 어플리케이션은 상기 할당을 나타내는 정보를 제공하여 상기 호스트로 하여금 상기 어플리케이션 대신에 메모리를 관리하게 함 - 를 포함하는

컴퓨터 시스템.

청구항 14

제 12 항에 있어서,

상기 어플리케이션 인터페이스 컴포넌트는 어플리케이션 바이너리 모듈을 정적으로 또는 동적으로 분석하여 메모리 사용 거동 정보를 상기 호스트에 제공하는

컴퓨터 시스템.

청구항 15

제 12 항에 있어서,

상기 요청 수신 컴포넌트, 상기 요청 저장 컴포넌트, 상기 할당 컴포넌트, 및 상기 메모리 액션 컴포넌트는, 어플리케이션 메모리를 관리하기 위해 상기 호스트로 액세스가능한 메모리 매니저의 컴포넌트들을 구성하는

컴퓨터 시스템.

청구항 16

제 12 항에 있어서,

상기 요청 수신 컴포넌트는, 상기 호스트 상에서 실행되는 복수의 어플리케이션으로부터 요청을 수신하고, 상기 호스트는 메모리 관련 액션을 취하도록 판정하는데 어느 할당이 영향을 미치는지 우선 순위화할 수 있는

컴퓨터 시스템.

청구항 17

제 12 항에 있어서,

상기 요청 저장 컴포넌트는, 상기 요청이 수신될 때 상기 호스트가 주어진 할당 요청을 만족할지 여부를 판정하고, 메모리 액션이 적절한 시점 이후에 이용 가능한 정보를 상기 호스트가 가질 수 있도록, 상기 할당과 연관하여 상기 어플리케이션에 의해 제공되는 정보를 저장하는

컴퓨터 시스템.

청구항 18

제 12 항에 있어서,

상기 요청 저장 컴포넌트는, 상기 어플리케이션이 메모리를 할당한 이후에 상기 어플리케이션으로부터 별도의 통신으로 할당 메타데이터를 수신하는

컴퓨터 시스템.

청구항 19

제 12 항에 있어서,

상기 메모리 액션 컴포넌트는

메모리 콘텐츠를 스왑 파일로 스와핑하는 액션과,

이전에 할당된 메모리를 해제하는 액션과,

어플리케이션이 상기 어플리케이션의 메모리 풋프린트를 감소시키는 것을 요청하는 액션

을 취하도록 구성되는,

컴퓨터 시스템.

발명의 설명

기 술 분 야

배 경 기 술

[0001] 컴퓨터 시스템들에 있어서의 메모리 관리는, 복수의 어플리케이션들 및 오퍼레이팅 시스템이 메모리의 사용에 합의하는 방식을 지칭한다. 각각의 컴퓨터 시스템이 고정된 양의 물리적 랜덤 액세스 메모리(RAM) 또는 다른 메모리를 갖지만, 오퍼레이팅 시스템은 물리적 메모리와는 상이한 메모리의 사이즈를 표현하는 가상 메모리를 어플리케이션들 및 오퍼레이팅 시스템 컴포넌트들에 제시할 수도 있다. 일부 경우들에 있어서, 가상 메모리는 오퍼레이팅 시스템으로 하여금 메모리의 특정 부분에 액세스하도록 각각의 어플리케이션을 제한하게 하여, 하나의 어플리케이션이 다른 어플리케이션의 메모리를 우발적으로 또는 의도적으로 수정함으로써 다른 어플리케이션의 동작을 간섭하는 것을 방지한다. 오퍼레이팅 시스템들은 일반적으로, 어플리케이션 및 오퍼레이팅 시스템 컴포넌트 요청들에 응답하여 메모리를 할당 및 해제(free)하기 위한 하나 이상의 기능들을 제공한다. 오퍼레이팅 시스템은 어플리케이션에게 메모리 풀을 제공할 수도 있으며, 그 메모리 풀로부터, 어플리케이션은 상당한 양의 메모리를 할당할 수 있다. 인스톨된 물리적 메모리의 양보다 더 많은 가상 메모리를 어플리케이션이 사용하거나 어플리케이션들의 그룹이 함께 사용하면, 오퍼레이팅 시스템은, 페이징 또는 디스크 스왑핑(즉, 메모리의 페이지들을 디스크에 저장하고 취출하는 것)으로 지칭되는 프로세스에서 스왑 파일을 통해 메모리의 외관상 사이즈를 연장하기 위해 더 느린 디스크-기반 저장부를 이용할 수도 있다.

[0002] 제공된 할당 및 해제 기능들을 제외하면, 오퍼레이팅 시스템들은, 어플리케이션들이 어떻게 메모리를 사용하는지에 대해 거의 알 수 없다. 다수의 컴퓨팅 디바이스들은 메모리를 둘러싼 특정한 한계들을 포함한다. 예를 들어, 모바일 컴퓨팅 디바이스들은 데스크탑 컴퓨터 시스템에 대해 통상적으로 이용가능한 것보다 훨씬 더 적은 양의 메모리를 포함할 수도 있어서(또는 시스템은 에너지 소모를 감소하기 위해 일부 메모리를 전력차단하길 원할 수도 있음), 얼마나 많은 어플리케이션들이 동시에 구동할 수 있는지, 각각의 어플리케이션이 얼마나 많은 메모리를 요청/소비할 수 있는지 등등과 관련된 디바이스에 대한 한계들을 형성한다. 특정 컴퓨팅 시스템 내의 어플리케이션 코드를 호스팅하는 다른 컴퓨팅 환경들은 또한 그 환경의 메모리 사용에 대한 한계 또는 상한들을 강화할 수도 있다. VMware 및 마이크로소프트 TM 가상 PC, 하이퍼바이저들, 오퍼레이팅 시스템들, 및 기타와 같은 호스트들은 한정된 리소스들을 할당받을 수도 있다. 이러한 상황들의 모두에 있어서, 유효 메모리 관리는 더 주목할만 하게 된다.

[0003] 새로운 컴퓨팅 플랫폼들은 신 기술들을 도입하였거나 구 기술들을 용도변경하여 어플리케이션들 간에 공유되는 한정된 메모리의 문제를 해결하였다. 예를 들어, 모바일 폰 오퍼레이팅 시스템들은, 어플리케이션이 (예를 들어, 활성적으로 이용되는) 전경에 있지 않을 경우에 오퍼레이팅 시스템이 그 어플리케이션을 셧다운하고 어플리케이션의 메모리의 이미지를 더 느린 저장부(예를 들어, 플래시 메모리 또는 다른 저장부) 상에 저장하도록, 각 어플리케이션의 메모리 스냅샷을 생성할 수도 있다. 어플리케이션이 선택될 경우, 오퍼레이팅 시스템은 저장된 이미지를 메모리로 다시 로딩하고 어플리케이션을 시작한다. 어플리케이션은 셧다운되었음을 알지도 못할 수도 있다. 그러한 기술들이 도움이 되지만, 오퍼레이팅 시스템은 여전히, 메모리를 사용하기 위한 어플리케이션의 불명료한 요청들을 받는다. 현재, 동적 메모리 사용에 관하여 행해진 대부분의 결정들은 런 타임 동안 발견된 정보에 기초하여 행해진다. 그러한 정보의 예로는 할당된 메모리 세그먼트에 대한 레퍼런스들의 사이즈 및 개수가 있다. 그 후, 이러한 정보는 어떤 할당들이 디스크에 페이징되거나, 고성능 메모리에 캐싱되거나, 또는 자동 메모리 관리 시스템의 일부 종류에 의해 해제될 것인지를 결정하는데 사용될 수 있다. 불행하게도, 임의의 플랫폼은 잠재적으로 수년의 레거시 어플리케이션들에 의해 제한되어, 메모리 관리만큼 광범위한 분야에서 새로운 모델들을 채택하는 것은 어렵다.

발명의 내용

해결하려는 과제

과제의 해결 수단

[0004] 메모리가 어떻게 사용되고 있는지를 나타내는 정보를 어플리케이션들로부터 수신하고 또한 어플리케이션 호스트로 하여금 메모리를 사용하기 위한 어플리케이션 요청들에 대해 더 많은 제어를 행사하게 하는 메모리 관리 시스템이 본 명세서에서 설명된다. 오늘날, 어플리케이션 호스트는, 어플리케이션이 얼마나 많은 메모리 요청들을 실시하였는지 및 각각의 요청에 의해 얼마만의 메모리 사이즈가 요청되었는지 이외에, 어플리케이션의 메모리 사용에 관하여 극히 적게 안다. 하지만, 어플리케이션 호스트는 각각의 메모리 할당의 목적, 어느 메모리 할당들이 바로 사용될 것인지, 어플리케이션 호스트가 더 많은 메모리를 요구하였으면 어느 메모리 할당들이 용이하게 재생될 수 있는지, 어플리케이션의 성능에 영향을 주지 않으면서, 어느 메모리 할당들이 잠시동안 사용되지 않을 것이고 따라서 디스크에 페이징될 수 있는지 등을 알지 못한다. 불행하게도, 어플리케이션 호스트가 이러한 타입들의 결정들을 실시하는 것으로 태스킹되지만, 이러한 결정들을 효과적으로 실시하는 것에 관한 대부분의 정보를 어플리케이션이 소유한다.

[0005] 메모리 관리 시스템은 이러한 문제들을 여러 방식들로 극복한다. 첫째, 그 시스템은, 어플리케이션으로 하여금 더 나중에 메모리를 관리하는 것에 유용한 메모리 할당들에 관한 더 많은 정보를 명시하게 하는 어플리케이션 메모리 관리 어플리케이션-프로그래밍 인터페이스(API)를 제공한다. API는, 또한 메모리가 요구될 때를 어플리케이션에게 통지하고 또한 어플리케이션의 상호작용없이 요구될 경우에 메모리 할당들을 순방향으로 해제하고 재생하기 위한 능력을, 어플리케이션 호스트에게 제공할 수도 있다. 둘째, 그 시스템은, 더 효과적인 메모리 관리에 참여하기 위한 일부 능력을 그 시스템으로 작업하도록 수정되지 않은 어플리케이션들을 제공하기 위해 레거시 어플리케이션들을 정적으로 및/또는 동적으로 분석하기 위한 능력을 제공한다. 셋째, 그 시스템은, 어플리케이션들에 의해 제공된 정보를 레버리징하고 또한 그 정보를 이용하여 메모리를 더 효과적으로 관리하기 위한 변화들을 어플리케이션 호스트에 제공하며, 어플리케이션의 메모리 사용으로 후킹(hook)한다. 따라서, 메모리 관리 시스템은, 어플리케이션 호스트 거동을 개선시키고 잠재적으로 어플리케이션들로 하여금 컴퓨팅 리소스들을 더 효율적으로 사용하게 하는, 메모리를 관리하기 위한 새로운 모델을 제공한다.

[0006] 본 요약부, 상세한 설명에서 이하 더 설명되는 개념들의 선택을 단순화된 형태로 도입하도록 제공된다. 본 요약부는 청구 대상의 핵심 특징들 또는 중요한 특징들을 식별하도록 의도되지 않으며 청구 대상의 범위를 한정하는데 사용되도록 의도되지도 않는다.

도면의 간단한 설명

[0007] 도 1은 일 실시예에 있어서, 메모리 관리 시스템의 컴포넌트들을 도시한 블록 다이어그램이다.

도 2는 일 실시예에 있어서, 메모리 관리 시스템의 오퍼레이팅 환경을 도시한 블록 다이어그램이다.

도 3은 일 실시예에 있어서, 메모리의 할당 및 사용을 요청하기 위한 소프트웨어 어플리케이션 내에서의 메모리 관리 시스템의 프로세싱을 도시한 플로우 다이어그램이다.

도 4는 일 실시예에 있어서, 어플리케이션 요청들을 수신하여 메모리를 할당 및 사용하기 위한 호스트 내에서의 메모리 관리 시스템의 프로세싱을 도시한 플로우 다이어그램이다.

도 5는 일 실시예에 있어서, 메모리 할당 정보를 제공하기 위해 구체적으로 설계되지 않은 어플리케이션을 분석하기 위한 메모리 관리 시스템의 프로세싱을 도시한 플로우 다이어그램이다.

도 6은 일 실시예에 있어서, 어플리케이션을 정적으로 분석하고 강화된 메모리 정보에 대한 매니페스트(manifest)를 제공하기 위한 메모리 관리 시스템의 프로세싱을 도시한 플로우 다이어그램이다.

도 7은 일 실시예에 있어서, 검출된 메모리 압력에 응답하여 메모리에 관련된 액션을 취하기 위한 메모리 관리 시스템의 프로세싱을 도시한 플로우 다이어그램이다.

도 8은 일 실시예에 있어서, 메모리가 호스트에 의해 이전에 수정되었던 어플리케이션을 활성화하기 위한 메모리 관리 시스템의 프로세싱을 도시한 플로우 다이어그램이다.

발명을 실시하기 위한 구체적인 내용

[0008] 메모리가 어떻게 사용되고 있는지를 나타내는 정보를 어플리케이션들로부터 수신하고 또한 어플리케이션 호스트로 하여금 메모리를 사용하기 위한 어플리케이션 요청들에 대해 더 많은 제어를 행사하게 하는 메모리 관리 시스템이 본 명세서에서 설명된다. 오늘날, 어플리케이션 호스트는, 어플리케이션이 얼마나 많은 메모리 요청들

을 실시하였는지 및 각각의 요청에 의해 얼마만의 메모리 사이즈가 요청되었는지 이외에, 어플리케이션의 메모리 사용에 관하여 극히 적게 안다. 하지만, 어플리케이션 호스트는 각각의 메모리 할당의 목적, 어느 메모리 할당들이 바로 사용될 것인지, 어플리케이션 호스트가 더 많은 메모리를 요구하였으면 어느 메모리 할당들이 용이하게 재생될 수 있는지, 어플리케이션의 성능에 영향을 주지 않으면서, 어느 메모리 할당들이 잠시동안 사용되지 않을 것이고 따라서 디스크에 페이징될 수 있는지 등을 알지 못한다. 불행하게도, 어플리케이션 호스트가 이러한 타입들의 결정들을 실시하는 것으로 태스킹되지만, 이러한 결정들을 효과적으로 실시하는 것에 관한 대부분의 정보를 어플리케이션이 소유한다. 이러한 모순은, 오늘날, 기능의 기저 레벨을 제공하고 또한 어느 액션들이 취해질 지를 추측하는 어플리케이션 호스트에 의해 해결된다. 다수의 경우들에 있어서, 어플리케이션 호스트는 어플리케이션이 메모리를 필요로 하기 직전에 메모리를 디스크에 페이징할 수도 있거나, 또는 어플리케이션 호스트는, 어플리케이션에 거의 중요하지 않은 메모리를 관리하는 막대한 노력을 낭비할 수도 있다.

[0009] 메모리 관리 시스템은 이러한 문제들을, 본 명세서에서 더 설명되는 수개의 방식들로 극복한다. 첫째, 그 시스템은, 어플리케이션으로 하여금 더 나중에 메모리를 관리하는 것에 유용한 메모리 할당들에 관한 더 많은 정보를 명시하게 하는 어플리케이션 메모리 관리 어플리케이션-프로그래밍 인터페이스(API)를 제공한다. API는, 또한 메모리가 요구될 때를 어플리케이션에게 통지하고 또한 어플리케이션의 상호작용없이 요구될 경우에 메모리 할당들을 순방향으로 해제하고 재생하기 위한 능력을, 어플리케이션 호스트에게 제공할 수도 있다. 둘째, 그 시스템은, 더 효과적인 메모리 관리에 참여하기 위한 일부 능력을 그 시스템으로 작업하도록 수정되지 않은 어플리케이션들을 제공하기 위해 레거시 어플리케이션들을 정적으로 및/또는 동적으로 분석하기 위한 능력을 제공한다. 셋째, 그 시스템은, 어플리케이션들에 의해 제공된 정보를 레버리징하고 또한 그 정보를 이용하여 메모리를 더 효과적으로 관리하기 위한 변화들을 커널-레벨 오퍼레이팅 시스템(또는 호스트)에 제공하며, 어플리케이션의 메모리 사용으로 후킹한다. 따라서, 메모리 관리 시스템은, 어플리케이션 호스트 거동을 개선시키고 잠재적으로 어플리케이션들로 하여금 컴퓨팅 리소스들을 더 효율적으로 사용하게 하는, 메모리를 관리하기 위한 새로운 모델을 제공한다. 본 명세서에서 설명되는 바와 같은 어플리케이션 호스트는 어플리케이션, 또는 실버라이트 TM, .NET, 고유의 Win32 호스트, 또는 다른 호스트들에 의해 제공된 런타임들과 같은 다른 타입의 호스트(예를 들어, 오퍼레이팅 시스템 또는 가상화 서브시스템 상에서 자체 구동하는 어플리케이션)을 실행하는 오퍼레이팅 시스템, 또는 VMware 및 가상 PC에 의해 제공된 가상 머신들을 지칭할 수도 있다. 상기 소개된 3개의 영역들 각각은 다음의 섹션들에서 더 상세히 설명된다.

[0010] 수정된 어플리케이션들

[0011] 다수의 경우들에 있어서, 소프트웨어 개발자들이 소프트웨어 어플리케이션들을 수정하여 메모리 관리 시스템과 상호작용하는 것이 가능할 수도 있다. 능동적으로 개발된 어플리케이션들에 있어서, 소프트웨어 개발자는, 어플리케이션이 메모리 관리 시스템을 채택하도록 동작하는 특정 플랫폼에 의해 지시될 수도 있거나 제공하는 이점들을 위해 메모리 관리 시스템을 채택하도록 선택할 수도 있다. 다수의 경우들에 있어서, 어플리케이션은, 그 어플리케이션이 사용할 가능성이 없는 메모리를 할당된 채로 유지할 수도 있다. 예를 들어, 사용자가 어플리케이션의 사용자 인터페이스의 일 부분으로부터 다른 부분으로 천이할 경우, 어플리케이션은 이전 인터페이스로부터의 정보를, 사용자가 그 인터페이스를 재방문할 경우에 유지하고 있을 수도 있다. 오늘날, 이러한 메모리는 다른 능동적으로 사용된 메모리와 같이 호스트에 요구된 것처럼 보인다. 메모리 관리 시스템은 어플리케이션이 이들과 같은 상황들을 호스트에게 통지하는 방식을 제공하여, 그러한 메모리가 우선순위 정도를 낮출 수 있다. 이에 대해, 호스트는 그러한 메모리를 페이징을 위한 양호한 후보로서 선택하거나, 또는 어플리케이션으로부터의 부가된 정보 때문에 더 효율적인 다른 메모리 관리 결정들을 실시할 수도 있다.

[0012] 일부 실시예들에 있어서, 메모리 관리 시스템은, 메모리 매니저로 하여금 런타임 동안 메모리 사용을 최적화하는 것에 관한 지능적 결정들을 실시할 수 있게 하는 어플리케이션-프로그래밍 모델 또는 프레임워크를 제공한다. 이는, 임의의 소정의 메모리 오브젝트들에 대해 명시된 메모리 할당에 어플리케이션 요청들을 할당 및 충진하는데 사용되는 액션들 및 메타데이터 양자를 수신하는 어플리케이션 프로그래밍 모델/프레임워크를 활용함으로써 달성된다. 메타데이터는, 메모리의 우선순위, 할당되는 메모리의 양, 스크래치로부터 메모리의 콘텐츠를 재생하는 용이성(예를 들어, 그 콘텐츠는 파일로부터 로딩되거나 알고리즘에 의해 계산될 수도 있음), 액세스의 빈도, 어플리케이션이 얼마나 신속히 메모리를 사용할 수 있는지 등과 같이, 메모리 할당의 본질 또는 목적을 설명하는, 호스트에 어플리케이션이 통신하길 원하는 정보를 제공한다. 메모리를 할당 및 충진하는데 사용되는 액션들은, 어플리케이션의 요청 시 메모리를 해제하고 해제된 메모리를 후속적으로 재생할 수 있기에 충분한 정보를 호스트에게 제공할 수도 있다. 개발자로 하여금 메모리를 충진하는데 사용되는 액션들 및 메타데이터를 명시하게 함으로써, 메모리 관리 시스템은 어플리케이션에 의해 명시된 원하는 사용과 부합하는 메모

리 사용을 최적화할 수 있다.

[0013] 메모리 관리 시스템에 의해 어플리케이션에 제공된 API는, 어플리케이션 프레임워크 또는 프로그래밍 모델을 통해, 메모리 할당들의 사용을 설명하는 메타데이터를 명시하기 위한 수단을 어플리케이션 개발자에게 제공한다. 부가적으로, API는, 어플리케이션 프레임워크로 하여금, 널리 공지된 함수들을 통해 메모리를 충진 또는 수정하기 위한 표준 수단을 개발자가 활용할 것을 지시하게 한다. 이는 메모리 관리 시스템으로 하여금 성능 이유로 메모리를 우발적으로 충진하게 하고, 다른 목적들로, 낮은 메모리 가용도의 기간 동안 (즉, 메모리를 해제하기 위한 기회 비용이 더 나중에 메모리를 재할당 및 재-정주시키기 위한 비용보다 클 경우) 메모리를 해제하게 한다.

[0014] 메모리 사용을 최적화하는 것은 당업계에 공지된 다수의 기술들을 포괄할 수 있지만, 일반적으로, 성능 또는 사이즈 중 어느 하나를 최적화하는 것을 의미할 것이다. 성능의 경우, 최적화하는 것은, 아직 불필요한 메모리 할당들을 이행하기 위한 수단이 이용가능하면, 아직 불필요한 메모리 할당들을 발생하게 하는 것을 의미할 수도 있다. 이는, 현재의 CPU 사용이 낮고 어플리케이션이 유휴상태이라면 바람직할 수도 있다. 일부 경우들에 있어서, 메모리를 할당하기 위한 어플리케이션의 요청들은, 호스트가 그 때에 아무것도 할당하지 않고 추후의 레퍼런스를 위해 저장함을 주목하게 될 수도 있다. 더 나중에, 어플리케이션이 API를 통해 메모리를 사용할 것을 요청할 경우 또는 호스트가 그 요청을 이행할 적절한 시간을 결정할 경우, 메모리 관리 시스템은 요청된 메모리를 실제로 할당한다. 사이즈의 경우, 최적화하는 것은 현재 할당된 메모리 및 추후 메모리 필요성들에 기초하여 메모리 풋프린트를 감소시키거나 결정을 행하는 것을 의미한다.

[0015] 어플리케이션과 호스트 사이의 실제 인터페이스는, 당업자에 의해 인식될 다양한 형태들을 취할 수도 있다. 예를 들어, 어플리케이션은 할당의 각 타입에 대한 할당 함수를 제공할 수도 있고, 할당 요청에 있어서, 할당에 대한 포인터 또는 레퍼런스를 호스트로 전달할 수도 있다. 호스트가 할당을 수행할 준비가 되면, 호스트는 제공된 할당 함수를 호출하고 어플리케이션은 정규의 메모리 할당 함수들을 사용하여 메모리를 생성한다. 유사하게, 어플리케이션은 다른 함수들로 레퍼런스들을 전달할 수도 있어서, 호스트가 메모리를 해제하는 것, 메모리를 이동시키는 것, 메모리 콘텐츠를 상이한 타입의 저장부로 스위칭시키는 것 등을 요청할 수 있다. 동일한 개념이 또한 할당용으로 이용될 수 있으며, 어플리케이션이 오퍼레이팅 시스템으로부터의 메모리를 요청할 경우, 오퍼레이팅 시스템은 다수의 팩터들에 기초하여 할당을 지연시킬 수도 있다. 오퍼레이팅 시스템이 할당할 준비가 되면, 함수 레퍼런스가 (할당된 메모리로) 다시 호출되거나 또는 이벤트(또는 유사한 메커니즘)가 야기된다. 그 인터페이스는 또한 (요청된 사이즈와는 상이할 수도 있는) 메모리 사이즈, 우선순위, 캐싱 선호도들, 페이지 가능성, 메모리가 어떻게 충진되는지, 메모리에 대한 의존성들 또는 레퍼런스들, 메모리가 업데이트되는지 여부 등과 같은 메타데이터를 수신할 수도 있다. 일부 실시예들에 있어서, 그 시스템은, 할당이 메모리 할당의 각 타입을 정의하도록 도출하는 메모리 인터페이스 클래스를 제공한다. 그 클래스는 어플리케이션 특정 할당 함수들을 취출하기 위한 GetPointer 함수, 또는 다른 메모리 핸들링 태스크들을 수행하기 위한 함수들을 취출하기 위한 다른 GetX 함수들을 포함할 수도 있다. 대안적으로 또는 부가적으로, 어플리케이션은 통상적인 방식으로 할당들을 실시하고, 할당된 메모리를 호스트에 등록하며 본 명세서에서 설명되는 부가 정보를 명시하여 할당된 메모리와 연관시키는 RegisterPointer 함수들을 호출할 수도 있다. 다음의 의사코드는 어플리케이션이 이용할 수 있는 하나의 메모리 클래스의 일 예를 제공한다.

[0016] CMemChunk // 모든 메모리 할당들을 위한 기저 클래스

[0017] {

[0018] <할당들의 글로벌 리스트>

[0019] …

[0020] };

[0021] CMyMemory : public CMemChunk

[0022] {

[0023] Allocate (size);

[0024] Fill () { <메모리 충진 방법> } // 오버라이드

[0025] Attributes <예를 들어, 우선순위, 페이지 가능성 등>

[0026] };

[0027] 대안적으로 또는 부가적으로, 개발자는, 기존의 메모리 할당들을 식별하기 위해 그리고 각각의 할당, 액세스, 또는 다른 메모리 상호작용에 관련된 부가적인 파라미터들 및 메타데이터를 명시하기 위해 본 명세서에서 설명된 프레임워크를 SAL(source annotation language) 또는 다른 마크업을 사용하여 어플리케이션 코드로 도입할 수도 있다.

[0028] 일부 실시예들에 있어서, 메모리 관리 시스템은 단일 어플리케이션 내에서 동작하고, 커널 또는 다른 호스트와 공유되지 않을 수도 있다. 어플리케이션은, 그 자신의 내부 메모리 매니저가 본 명세서에서 설명되는 프레임워크를 사용함으로써 제공된 부가적인 정보로 수행할 수 있는 개선된 메모리 관리로부터 이익을 얻을 수 있다. 일부 경우들에 있어서, 그 후, 호스트는, 어플리케이션이 크로스-어플리케이션 이점들을 획득하고 또한 호스트로 하여금 잘 정의된 메모리 할당들 및 사용을 또한 사용하게 하도록 호출할 수 있는 등록 함수를 제공할 수도 있다. 일 예로서, 그 시스템은 멀웨어(malware) 스캔 전에 어플리케이션에게 통지하여, 어플리케이션이 그 스캔을 가속하기 위해 임의의 덜 관련된 메모리를 언로딩할 수 있다. 다른 예로서, 어플리케이션은, 사용자가 어떤 것을 하고 CPU가 실행을 재개(전력 상태 변화들에 대한 응답도를 향상시킴)할 경우에 어플리케이션이 이벤트에 신속하게 응답하도록, CPU가 유휴 상태에 진입하기 전에 메모리를 선제적으로 할당할 수도 있다.

[0029] 수정되지 않은 어플리케이션들

[0030] 소프트웨어 개발자들이 소프트웨어 어플리케이션들을 수정하여 메모리 관리 시스템과 상호작용하는 것이 가능하지 않는 경우 또는 그 시스템이 수정되지 않은 어플리케이션들(즉, 그 시스템과 작동하도록 구체적으로 설계되지 않은 어플리케이션들)과 동작하도록 구현되는 경우에 있어서, 그 시스템이 메모리 관리 이점들을 제공하는 것은 여전히 가능하다. 그렇게 행하기 위해, 시스템은 (예를 들어, 정적 분석 및/또는 어플리케이션을 프로파일링하는 것 - 어플리케이션을 구동하게 하는 것, 메모리 할당들을 차단하는 것, 및 어플리케이션의 런타임에 걸쳐 사용을 살피는 것 - 에 기초하여) 어플리케이션이 메모리 자체를 어떻게 사용하는지를 나타내는 정보를 수집한다. 이러한 정보는 성능 특성들을 결정할 경우에 유용하고, 어플리케이션의 호스트의 메모리 매니저에 의해 지능적으로 사용될 수도 있다. 이러한 정보가 사용될 수도 있는 방식의 예들은 지능적 가비지 수집, 디스크로의 지능적 페이징, 더 높은 성능 메모리 캐시로의 지능적 캐싱, 및 어플리케이션이 조우할 수도 있는 잠재적인 메모리 한계들에 관하여 사용자에게 심지어 경고하는 것을 포함한다.

[0031] 바이너리의 거동의 바이너리 런타임 분석의 정적 분석을 활용하여 그리고 바이너리를 장비함으로써, 바이너리의 소정 메모리 할당들 중 임의의 할당 및 그 할당들의 사용에 관한 부가적인 정보를 수집하는 것이 가능하다. 그 후, 이러한 정보는, 물리적 메모리 내 할당의 위치 및 로딩/언로딩에 관한 더 지능적 거동을 유도하는데 사용될 수도 있다. 메모리 관리 시스템은, 할당 자체의 잠재적인 또는 실제의 사용을 설명하는 메타데이터로 어플리케이션 메모리 할당들에 자동으로 주석을 다는 수단을 제공한다. 이러한 분석은, 어떠한 개발자 상호작용이나 기존 어플리케이션들의 재작성을 요구하지 않고, 런타임 동안 바이너리로 정적으로 또는 동적으로 자동 실행될 수 있다. 일단 수행되면, 분석은 시스템에 의해 캐싱될 수도 있어서, 호스트 오퍼레이팅 시스템은 추후에 어플리케이션을 처리하는 방법을 안다. 분석은 또한, 단지 국부적으로 캐싱되는 것이 아니라, 다른 클라이언트들에 의한 발견을 위해 공개될 수도 있다. 부가적으로, 정보가 옵션적인 편집을 위해 사용자에게 노출되는 것이 가능하여, 관리자 또는 사용자로 하여금 어플리케이션의 메타데이터 및 어플리케이션 호스트가 어떻게 어플리케이션의 메모리 할당들을 처리할 것인지를 맞춤화하게 한다.

[0032] 정적 및 동적 분석을 사용하여 어플리케이션의 메모리 할당들에 관한 부가적인 정보를 도출하는 것이 가능하며, 그 후, 이러한 정보는 어플리케이션 및 시스템 메모리 할당들의 전체 관리를 지시하는 것을 돕는데 사용될 수도 있다. 일 예는 메모리가 터치되고 있을 때이며; 다른 메모리 세그먼트가 이러한 메모리 할당을 충진하는 것을 돕는데 사용되는지 여부를 결정하는 것이 가능하다. 그 후, 이러한 메모리 할당이 다른 할당에 의존하면, 이러한 의존성은 표시될 수 있거나, 또는 메모리 할당은, 입력없이 생성되지 않았었음을 나타내기 위해 비트에 의해 단순히 표기될 수도 있다. 정적 분석은 소프트웨어 코드가 어디에서 메모리를 터치하는지, 메모리가 어떻게 사용되는지, 메모리가 어떻게 충진되는지, 코드 경로가 얼마나 공통인지(예를 들어, WORM(write once/read many)인 경우), 및 얼마나 자주 메모리가 사용되는지를 결정할 수 있다. 동적 분석은 모든 할당들 및/또는 액세스들(프로파일러와 유사)을 가질 수 있으며, 코드의 동작에 대한 시스템 환경 영향들, 동작에 영향을 주는 사용자 설정들, 및 정적으로 결정하기 어렵거나 이용할 수 없는 다른 데이터를 포착할 수 있다.

[0033] 일부 실시예들에 있어서, 메모리 관리 시스템은, 어플리케이션이 분석된 이후, 어플리케이션의 메모리 사용의 매니페스트 또는 다른 디스크립션을 출력한다. 이는, 시스템으로 하여금 분석의 결과들을 캐싱하게 하고 어플

리케이션의 추후 실행을 위해 결과들을 재사용하게 한다. 오퍼레이팅 시스템은, 어플리케이션 바이너리가 실행하는 처음에(또는 어플리케이션 가상화의 시퀀싱과 같은 프로세스들 동안) 이러한 분석을 수행하고, 그 후, 바이너리가 실행할 때마다 사용하기 위해 그 분석을 저장하도록 시스템으로 설계될 수도 있다. 커널 또는 다른 호스트는 매니페스트 데이터를 판독하고, 어플리케이션이 메모리를 어떻게 사용하는지를 설명하는 부가적인 정보를 통해 적절한 액션을 취할 수 있다. 어플리케이션 사용은 시간에 걸쳐 변할 수도 있어서, 매니페스트 또는 다른 캐시는 가끔 동적으로 업데이트될 수도 있다. 일부 경우들에 있어서, 시스템은 메모리의 전체 페이지들을 각각의 할당에 대해 할당할 수도 있어서, 각각의 메모리 액세스는 커널로 하여금 메모리가 어떻게 사용되는지를 제어하게 하고 또한 메모리에 대한 어플리케이션 레퍼런스들과 본 명세서에서 더 설명되는 실제 메모리 할당들 간의 인디렉션(indirection)의 타입을 제공하게 하는 페이지 결함을 트리거한다.

[0034] 일부 실시예들에 있어서, 시스템은 어플리케이션 메모리 거동에 대한 리포트들을 제공할 수도 있다. 이는 분석에 대한 다른 사용이고, 어느 서버 또는 가상 머신에 대해 잘 구동할 것인지와 같이 어플리케이션에 관한 결정을 행하도록 관리자를 도울 수도 있다. 시스템은 또한 메모리 사용의 시장 평가를 제공할 수 있어서, 예를 들어, 모바일 전화 어플리케이션 스토어의 사용자들은, 어플리케이션을 다운로드하기 전에, 어플리케이션이 어떻게 모바일 전화의 가용 메모리를 사용할 것인지(예를 들어, 높은 사용, 낮은 사용 등)를 알 수 있다. 관리자는 또한, 메모리 소비에 관한 수집된 메타데이터에 기초하여 다양한 시스템들에 걸친 소프트웨어의 IT 배치를 위해 이러한 정보를 이용할 수 있다.

[0035] 메모리 관리 시스템은 메모리 사용 분석으로부터 도출된 정보를 사용하여, 시스템이 유휴상태이거나 낮게 활용될 경우에 메모리를 선제적으로 할당하고 잠재적으로 충진할 수 있다. 메모리에 관한 어플리케이션의 추후 사용을 아는 것은, 시스템으로 하여금 메모리를 더 효율적으로 할당하게 하고 또한 시스템이 더 무거운 부하 하에 있을 경우에 사용될 수도 있는 메모리를 시스템이 할당하도록 낮게 활용되는 시간들을 레버리징하게 한다. 이는, 더 무거운 부하로 하여금 더 이르게 완료되었던 작업에 의해 이용가능하게 된 더 많은 프로세싱 또는 다른 리소스들로 액세스하게 한다.

[0036] 호스트 수정들

[0037] 메모리 관리 시스템은 커널 또는 어플리케이션 호스트에 대한 수정들을 포함하여, 어플리케이션에 의해 제공되거나 어플리케이션의 분석을 통해 결정된 메모리 사용에 관한 부가적인 정보를 수신한다. 통상적인 소프트웨어 메모리 관리와 달리, 커널은, 본 명세서에서 설명된 어플리케이션으로부터 통신된 메타데이터 또는 다른 정보에 의해 제공된 메모리 사용으로의 부가된 통찰력들 때문에, 어플리케이션의 지식 또는 액션없이 메모리를 효율적으로 관리하기 위해 더 많이 실시할 수 있다. 커널은 더 나중에, 어떤 액션들이 취해졌었는지를 어플리케이션에게 통지하거나 메모리를 관리하여, 어플리케이션이 메모리를 필요로 하는 시간까지 제자리에 있게 하고, 따라서, 어플리케이션은 커널의 메모리 관련 액션들을 알지 못하거나 그 액션들과 관련되지 않는다. 그 후, 커널은 (예를 들어, 덜 중요하거나 사용될 가능성이 없는 메모리를 더 느린 디스크 저장부로 오프로딩함으로써) 더 양호한 페이징을 수행할 수 있고, 메모리 압력이 발생하면 메모리를 해제할 수 있으며, 커널에 의해 제공된 오퍼레이팅 환경을 사용하여 하나 이상의 어플리케이션들 대신에 메모리를 관리하기 위한 다른 액션들을 취할 수 있다. 예를 들어, 커널은 더 적은 프래그먼트화를 위해 메모리를 더 양호하게 할당할 수 있다.

[0038] 메모리는 오퍼레이팅 시스템에 있어서의 제약된 리소스이며; 따라서, 커널이 메모리가 어디에 할당되는지를 적절히 추적하여 필요에 따라 어플리케이션들로부터 메모리를 복원할 수 있는 것이 중요하다. 하나의 솔루션은 어플리케이션들로 하여금 그 어플리케이션들에게 할당된 바와 같은 메모리에 대한 우선순위를 할당하게 하는 것이다. 따라서, 시스템이 메모리 상에서 낮다고 결정할 경우, 커널은 최저 우선순위 메모리가 어디에 있는지를 결정하고, 그 메모리를, 더 높은 우선순위 메모리를 갖는 다른 어플리케이션들의 성능에 영향을 주지 않으면서 할당해제하거나 페이징할 수 있다.

[0039] 통상적으로, 메모리의 우선순위 및 어플리케이션들로의 그 분배는 커널에 의해 결정된다. 시스템이 메모리 상에서 낮게 구동할 경우, 커널은 더 높은 우선순위를 갖는 어플리케이션에 의해 사용되는 메모리를 임의로 해제할 수도 있으며, 따라서, 어플리케이션의 성능을 방해하거나 열화시킬 수도 있다. 대신, 더 낮은 우선순위 메모리는 먼저 할당해제되었어야 한다. 어느 어플리케이션이 메모리를 해제하는지를 커널이 어떻게 결정하는지와 관련된 흥미있는 양태들이 존재한다. 하나의 솔루션은 어플리케이션들로 하여금 그 어플리케이션들 사이에서, 메모리 리소스들에 대해 우선순위 순서가 무엇인지를 결정하게 하는 것이다. 메모리 우선순위 방식은, 어플리케이션들이 할당 또는 할당해제 시 그 어플리케이션들에 할당되거나 할당해제되는 메모리의 우선순위를 할당할 것을 요청한다. 따라서, 메모리 압력이 존재할 경우, 커널은 우선순위들에 의해 랭크된 메모리 맵을 가지며 더

낮은 우선순위 메모리를 먼저 해제하고 위치지정할 수도 있다. 대안적으로, 커널은, 해제될 필요가 있는 할당들의 리스트를 갖는 통지를 어플리케이션으로 전송할 수도 있다. 메모리 관리 시스템은 오퍼레이팅 시스템에 의해 완전히 호스팅될 수도 있거나, 또는 오퍼레이팅과 어플리케이션 간에 협력할 수도 있다.

[0040] 우선순위 모델은, 메모리 요청이 실시될 때마다 어플리케이션을 캡슐화하는 메모리 할당 API를 생성함으로써 구현될 수도 있다. 이는 또한, 등록하지 않은 어플리케이션들에게 이익을 줄 수 있다. 이러한 API를 활용함으로써, 어플리케이션 및 우선순위는, 어플리케이션으로 하여금 메모리 우선순위를 능동적으로 관리하게 하지 않으면서, 서브시스템에 의해 자동으로 추적된다. 커널 오브젝트는 등록, 계산, 시그널링, 및 메모리 관리 기능 모두를 통합하여, 각각의 어플리케이션은 오직 이 오브젝트만을 호출해야 한다(또는 수정되지 않은 어플리케이션들의 경우, 어플리케이션을 대신하여 호출되었음).

[0041] 대안적인 솔루션은, 현재 구동하고 있는 다양한 어플리케이션들의 트랙을 유지하는데 사용되는 커널 외부의 마스터 어플리케이션을 갖는 것이다. 어플리케이션들이 메모리를 요청할 경우, 어플리케이션들은 마스터 어플리케이션에 등록된다. 따라서, 어플리케이션이 무응답이 될 경우, 마스터 어플리케이션은 좀비 어플리케이션을 킬링하여 메모리를 재생할지 여부를 결정하거나, 또는 메모리가 요구될 경우에 메모리가 어플리케이션으로 리턴될 수 있도록 미결상태로 유지할 수도 있다.

[0042] 커널 또는 호스트는 강화된 메모리 정보를 사용하여, 다양한 메모리 관리 결정들을 더 효율적으로 실시할 수 있다. 예를 들어, 저전력 디바이스의 커널은, 배터리 수명을 절약하기 위해 메모리의 뱅크들을 턴오프하도록 선택할 수도 있다. 시스템은, 턴오프될 뱅크들 내의 메모리를 갖는 프로세스들을 킬링하거나 교환할 수도 있다. 시스템은 또한 수개의 뱅크들에 걸쳐 확산된 메모리를 디프래그먼트화하여 셧오프될 수 있는 일부 빈 뱅크들을 성취할 수도 있다. 다른 예로서, 커널은 일부 메모리 할당들을 (클라우드 기반 저장 설비를 포함한) 다른 디바이스들로 푸시하도록 선택할 수도 있다. 다수의 작은 풋프린트, 저전력 오퍼레이팅 시스템들은 페이징의 개념을 갖지 않아서, 이 경우, 메모리 할당은 다른 디바이스/서비스에 의해 호스팅된 REST(Representational State Transfer) 인터페이스를 통해 지원될 수도 있다. 데이터베이스 스냅샷을 통해 스크롤링하는 것과 유사한 방식으로, 어플리케이션은 전체 데이터베이스를 메모리에 갖지 않지만, 데이터베이스 서비스/서버로부터 전달된 피스들이다. 어플리케이션은, 전체 데이터베이스보다는 스냅샷을 보고 있다는 것을 알지 못한다. 메모리를 자신있게 이동하거나 메모리를 할당해제할 수 있는 것은 커널/호스트로 하여금 메모리를 더 양호하게 관리하게 하고 메모리 가용도의 어플리케이션 기대들을 여전히 갖게 한다.

[0043] 일부 실시예들에 있어서, 시스템은 메모리에 부가하여 다른 호스트-레벨 오브젝트들 또는 컴포넌트들과 함께 사용된다. 예를 들어, 시스템은 디스플레이가 오프될 경우에 그래픽 프로세싱 유닛(GPU)을 셧오프하거나, 또는 그 관련된 하드웨어가 어떤 시간 동안 사용되고 있지 않거나 사용되지 않을 경우에 개별 드라이버들을 셧오프할 수도 있다. 시스템은 또한 시스템의 하이버네이션용으로 사용되어, 먼저 메모리를 디프래그먼트화하고, 그 후, 시스템이 웨이크 업할 시간인 경우에 용이하게 리로딩될 수 있는 디스크로 메모리 할당의 선형 스트림을 스트리밍할 수 있다. 전체 물리적 메모리와 사이즈가 동일한 파일을 저장하는 통상적인 하이버네이션과 달리, 메모리 관리 시스템은 불필요하거나 용이하게 복원된 메모리를 하이버네이션 전에 해제하도록 어플리케이션들을 권장할 수 있고(또는 어플리케이션들에 대해 그렇게 실시할 수 있음), 따라서, 더 적은 양의 하이버네이션 데이터를 저장할 수 있다. 메모리에 부가하여 관리되는 다른 아이템들은 파일 핸들들, 타이머들, 커널 오브젝트들 등을 포함할 수도 있다. 시스템은, 메모리 사용과 같이, 이들 아이템들에 관한 사용 정보를 수신할 수 있다(또는 정적 및 동적 분석을 통해 결정할 수 있음).

[0044] 일부 실시예들에 있어서, 메모리 관리 시스템은 각각의 메모리 할당과 연관된 전력 상태들을 수신한다. 예를 들어, 전력 레벨 X에서, 시스템은, 일부 메모리 할당들이 더 이상 필요하지 않다고 결정할 수도 있다. 이는, 배터리 제약형 디바이스로 하여금 그 전력 레벨에서 필요하지 않는 임의의 메모리를 오프로딩함으로써 저전력 모드로 스위칭하게 할 수 있다. 예를 들어, 모바일 전화기는 메모리에 충분한 어플리케이션 데이터를 유지하여 전화 호출들에 응답하거나 사용자를 새로운 이메일에 경보할 수도 있지만, 다른 더 낮은 우선순위 함수들 또는 어플리케이션들을 언로딩할 수도 있다. 일부 경우들에 있어서, 어플리케이션들은 오직 일부 더 낮은 전력 상태들에서의 메모리를 사용하기 전에 포인터 유효성을 검증할 필요가 있을 수도 있지만, 더 높은 전력 상태들에서의 메모리로의 통상의 액세스를 가질 수도 있다. 이는, 임의의 소정 플랫폼에 대하여 어플리케이션과 호스트 간의 접촉에서의 보증으로서 제공될 수 있다.

[0045] 시스템 컴포넌트들 및 오퍼레이팅 환경

[0046] 도 1은 일 실시예에 있어서, 메모리 관리 시스템의 컴포넌트들을 도시한 블록 다이어그램이다. 시스템(100)은

메타데이터 수신 컴포넌트(105), 충진 규격 컴포넌트(110), 할당 요청 컴포넌트(115), 메모리 레퍼런싱 컴포넌트(120), 어플리케이션 인터페이스 컴포넌트(125), 정적 분석 컴포넌트(130), 동적 분석 컴포넌트(135), 호스트 컴포넌트(140), 요청 수신 컴포넌트(145), 요청 저장 컴포넌트(150), 할당 컴포넌트(155), 메모리 액션 컴포넌트(160), 및 데이터 저장 컴포넌트(165)를 포함한다. 이들 컴포넌트들 각각은 본 명세서에서 더 상세히 설명된다.

[0047] 메타데이터 수신 컴포넌트(105), 충진 규격 컴포넌트(110), 할당 요청 컴포넌트(115), 및 메모리 레퍼런싱 컴포넌트(120)는 함께, 시스템(100)이 어플리케이션들에 노출하는 메모리 프레임워크를 구성한다. 더 효율적인 메모리 관리를 위해 그 어플리케이션 코드를 수정하려 하는 개발자들은 이들 컴포넌트들을 그 어플리케이션들로부터 레버리징하여, 호스트 또는 커널로 하여금 어플리케이션을 대신하여 메모리를 더 효과적으로 관리하게 하는 어플리케이션들을 생성할 수 있다.

[0048] 메타데이터 수신 컴포넌트(105)는, 메모리가 어플리케이션에 의해 어떻게 사용되는지를 나타내는 정보를 어플리케이션 호스트에 제공하는 각각의 메모리 할당과 연관된 정보를 수신한다. 예를 들어, 메타데이터는, 어플리케이션에 대해 할당이 얼마나 용이하게 액세스가능해야 하는지 또는 어플리케이션 플랜들이 할당과 연관된 메모리에 얼마나 자주 액세스하는지를 나타낼 수도 있다. 메타데이터는 또한, 메모리 콘텐츠들이 생성하기에 얼마나 어려운지, 따라서, 콘텐츠가 해제되거나 디스크로 페이징되면 어플리케이션 또는 호스트가 메모리 콘텐츠를 대체하기가 얼마나 어려운지를 나타낼 수도 있다. 메타데이터 수신 컴포넌트(105)는, 메모리 할당 API로의 호출 시 또는 메모리가 메타데이터를 제공하기 위한 팔로우-업 API에서 이미 할당된 이후에, 메타데이터를 수신할 수도 있다.

[0049] 충진 규격 컴포넌트(110)는 특정 메모리 할당의 콘텐츠가 어떻게 충진되는지를 나타내는 정보를 수신한다. 메모리의 콘텐츠는 다양한 소스들로부터, 예를 들어, 디스크 상에 저장된 파일의 콘텐츠를 판독하는 것으로부터, 입력 데이터에 대한 하나 이상의 계산들을 수행하는 것으로부터, 사용자 입력으로부터, 네트워크 액세스된 정보(예를 들어, 데이터베이스 또는 인터넷)로부터 등등으로부터 유래할 수도 있다. 일부 실시예들에 있어서, 어플리케이션은 메모리 충진 함수를 호스트로 전달하여, 호스트는 그 호스트에 의해 결정된 시간에 메모리 콘텐츠를 충진하도록 그 함수를 호출할 수 있다. 프로세싱 및 다른 리소스들을 효율적으로 사용하기 위해, 호스트는 리소스들이 낮게 활용되거나 유휴상태일 때까지 메모리를 충진하는 것을 지연하도록 선택할 수도 있다. 부가적으로, 호스트는 또한 이전에 충진된 메모리를 다른 사용들을 위해 릴리스하거나 해제하기 위한 자유를 가지며, 그 후, 어플리케이션이 메모리를 사용하길 기대하기 전에 메모리를 더 나중에 재할당 및 재충진할 수도 있다. 수신된 메타데이터는, 어플리케이션이 메모리를 사용할 때를 알도록 호스트가 이용하는 정보를 제공할 수도 있거나, 또는 어플리케이션은 각각이 메모리를 사용하려고 시도하기 전에 호스트에게 통지할 수도 있어서 호스트가 메모리의 현재 상태를 체크할 수 있다.

[0050] 할당 요청 컴포넌트(115)는 수신된 메타데이터 및 충진 규격에 기초하여 메모리를 할당하기 위한 요청을 어플리케이션으로부터 호스트로 제출한다. 호스트가 그 요청을 수신하지만, 응답으로 그 요청을 즉시 서비스할 지 또는 다른 적절한 시간까지 대기할 지는 호스트에게 달려 있음을 유의한다. 극단적인 경우, 호스트는 액세스될 준비가 될 때까지 어떠한 메모리도 할당하지 않을 수도 있어서, 메모리가 어플리케이션에 의해 요구되고 어플리케이션이 그 작업을 수행할 수 있도록 할당되어야 할 마지막 가능한 순간까지 호스트로 하여금 한정된 리소스들을 보존하게 할 수도 있다. 할당 요청 컴포넌트(115)는 제출된 요청을, 호스트에 의해 관리되는 데이터 저장부에 저장하고, 더 나중의 메모리 관리 액션들에서의 사용을 위해 수신된 메타데이터 및 충진 규격을 포함한다.

[0051] 메모리 레퍼런싱 컴포넌트(120)는, 이전에 제출된 할당 요청의 대상이었던 메모리에 어플리케이션이 액세스하기 전, 어플리케이션으로부터 표시를 수신한다. 호스트가 메모리로 하여금 선택할 때까지 실제 할당 메모리를 이용불가능하거나 지연하게 할 수 있기 때문에, 어플리케이션은, 어플리케이션이 메모리를 이용할 준비가 될 경우에 메모리가 이용가능함을 보장하기 위한 방법을 필요로 한다. 메모리 레퍼런싱 컴포넌트(120)는 이러한 목적을 제공하고, 어플리케이션으로 하여금 특정 메모리 할당에 액세스할 준비가 됨을 표시하게 한다. 응답으로, 호스트는 (메모리가 이미 이용가능하다면) 실제 메모리 위치에 대한 포인터를 어플리케이션에 전달할 수도 있거나, 또는 (메모리가 현재 할당되어 있지 않으면) 수신된 충진 규격 및 메타데이터에 기초하여 메모리를 생성 및 충진할 수도 있다. 어플리케이션은 메모리를 릴리스하기 위한 표시를 전송할 수도 있어서, 호스트는 메모리 관리 결정들에 있어서 메모리를 한번 더 포함하기 위해 해제된다.

[0052] 어플리케이션 인터페이스 컴포넌트(125)는 어플리케이션과 호스트 간의 통신 인터페이스를 제공하여 메모리 할당들의 사용을 협상한다. 인터페이스는 메모리 할당들을 요청하고 그 할당들에 관한 정보를 명시하기 위해 어

플리케이션에 의해 사용된 하나 이상의 함수들 또는 기저 클래스들뿐 아니라 어플리케이션에 의해 제공된 사용자-정의 함수들을 사용하여 어플리케이션의 메모리와 상호작용하기 위해 호스트에 의해 사용된 함수들 또는 기저 클래스들을 포함할 수도 있다. 어플리케이션과 호스트 간의 강화된 통신은 호스트로 하여금 어플리케이션의 메모리 사용으로의 통상보다 훨씬 더 높은 가시 레벨을 갖게 하고, 호스트로 하여금 유한한 리소스들을 더 효과적으로 공유하는 다양한 어플리케이션들 대신에 메모리를 관리하게 한다.

[0053] 메모리 관리 시스템(100)을 사용하기 위해 구체적으로 형성되지 않은 어플리케이션들에 있어서, 어플리케이션 인터페이스 컴포넌트(125)는 정적 및/또는 동적 분석에 의해 결정된 임의의 장비된 어플리케이션 코드 간의 상호작용을 제공하고, 이전에 논의된 바와 유사한 방식으로 호스트와 상호작용한다. 시스템(100)을 사용하도록 형성되지 않은 어플리케이션들에 있어서, 시스템(100)은 메모리 할당의 목적 또는 다른 규격에 관한 더 적은 정보를 가질 수도 있으며, 어플리케이션의 정적 및 동적 분석을 통해 시스템(100)이 발견할 수 있는 정보로 제한될 수도 있다. 하지만, 그러한 분석은, 여전히, 레거시 어플리케이션들에 의해 할당된 더 효과적으로 관리하는 메모리에 유용한 메타데이터를 발견할 수 있다. 일부 경우들에 있어서, 시스템(100)은 어플리케이션이 메모리를 어떻게 충진하고 있는지를 자동으로 검출할 수도 있고, 어플리케이션의 명시적인 협력없이 이전에 설명된 메모리 콘텐츠의 동일한 종류의 주문식 충진을 수행할 수도 있다.

[0054] 정적 분석 컴포넌트(130)는 어플리케이션 바이너리 또는 다른 어플리케이션 코드를 정적으로 분석하여, 어플리케이션이 메모리를 어떻게 사용하는지를 결정한다. 컴포넌트(130)는 바이너리 코드, 중간 코드(예를 들어, 마이크로소프트 TM 중간 언어(IL) 코드), 또는 어플리케이션의 다른 컴파일된 또는 구동가능한 버전들을 분석할 수도 있다. 정적 분석은 지난 수년에 걸쳐 실질적으로 진보하였으며, 다수의 기술들이, 어플리케이션 바이너리가 무엇을 행하는지 및 어떻게 행하는지를 결정하는 업계에서 널리 공지되어 있다. 메모리 관리 시스템(100)은 이들 기술들을 이용하여, 어플리케이션이 메모리를 할당하고 사용하는 영역에 특히 집중한다. 컴포넌트(130)는 어플리케이션 바이너리를 장비하여 정보를 수신하거나 어플리케이션의 특정 액션들을 차단할 수도 있으며, 차단된 액션들을 새롭거나 부가적인 액션들로 대체할 수도 있다. 예를 들어, 컴포넌트(130)가 메모리 할당 함수에 대한 호출을 발견하면, 컴포넌트(130)는 정적 분석으로부터 이용가능한 임의의 메타데이터를 수집하고, 메타데이터를 파라미터로서 수신할 수 있는 할당 함수의 버전을 호출할 수도 있다. 이러한 방식으로, 호스트는, 마치 어플리케이션이 그러한 정보를 제공하기 위해 그 개발자에 의해 수정된 것처럼, 메타데이터 정보를 어플리케이션으로부터 수신한다. 유사하게, 시스템은 어플리케이션이 어떻게 특정 메모리 할당을 충진하거나 액세스하는지를 결정할 수도 있어서, 메모리 레퍼런스들을 나타내는 정보 및 충진 규격이 호스트에 제공될 수 있다.

[0055] 동적 분석 컴포넌트(135)는 구동하는 어플리케이션을 동적으로 분석하여, 정적 분석으로 결정하기 어려운 어플리케이션의 메모리 사용에 관련된 부가적인 정보를 수집한다. 종종, 어플리케이션들은 정적 분석을 좌절시키는 프로그래밍 단계들을 (의도적으로, 또는 단순히 그 단계들이 그러한 방식으로 되기 때문에) 포함한다. 동적 분석은 어플리케이션에 의해 사용된 메모리의 실제 콘텐츠 및 외부 콘텐츠로부터 수신된 응답들의 콘텐츠와 같이 이용가능한 정보를 가지며, 이에 대해 정적 분석 동안에는 오직 추측들 또는 근사화들만이 이용가능하다. 따라서, 동적 분석은, 정적 분석동안 발견되지 않은 메모리 사용 및 다른 어플리케이션 활동을 발견할 수 있다. 컴포넌트(135)는 또한 동적 분석을 사용하여 정적 분석의 결과들을 확인할 수 있다. 동적 분석 컴포넌트(135)는 결정된 정보를 호스트에 제공하여, 추가로, 레거시 어플리케이션들로 하여금 시스템(100)에 의해 제공된 적어도 일부 기능을 레버리징하게 한다.

[0056] 호스트 컴포넌트(140)는, 어플리케이션이 구동하고 시스템(100)에 의해 제공된 메모리 관리로의 액세스를 제공하는 환경을 포함한다. 플랫폼에 의존하여, 호스트는 다양한 소프트웨어 엔터티들을 포함할 수 있다. 통상적인 데스크탑 컴퓨팅 시스템들에 있어서, 호스트는, 드라이버들과 같이 커널 및 다른 커널-모드 소프트웨어 코드를 포함할 수도 있는 오퍼레이팅 시스템이다. 관리된 소프트웨어 환경에 있어서, 호스트는 마이크로소프트 TM .NET TM 런타임 또는 마이크로소프트 TM 실버라이트 TM 런타임과 같은 런타임을 포함할 수도 있다. 웹 어플리케이션들은 또한 어플리케이션 호스트를 포함할 수도 있고, 다른 서버 또는 데스크탑 컴퓨팅 환경들 내에서 구동하는 샌드박스형 환경에서 구동할 수도 있다. 서버 컴퓨팅 시스템들은 가상화 하드웨어 또는 소프트웨어를 포함할 수도 있으며, 하나 이상의 오퍼레이팅 시스템 커널들보다 더 높은 레벨에서 동작하는 하이퍼바이저를 포함할 수도 있다. 메모리 관리 시스템(100)은 다양한 레벨들의 소프트웨어 코드로 구현될 수 있으며, 따라서, 임의의 특정 타입의 호스트로 한정되지 않는다. 호스트 컴포넌트(140)는, 어플리케이션들과 상호작용하고 또한 시스템(100)이 구동하고 있는 특정 플랫폼에 대한 리소스들을 관리하는 것을 책임지는 컴포넌트를 나타낸다.

[0057] 요청 수신 컴포넌트(145), 요청 저장 컴포넌트(150), 할당 컴포넌트(155), 및 메모리 액션 컴포넌트(160)는 함

께, 본 명세서에서 설명된 시스템(100)을 구현하도록 수정된 메모리 매니저의 컴포넌트들을 구성한다.

[0058] 요청 수신 컴포넌트(145)는 메모리를 할당하기 위해 어플리케이션으로부터의 요청들을 수신한다. 각각의 요청은, 어플리케이션이 할당된 메모리를 어떻게 사용하는지를 나타내는 정보를 호스트에 제공하는 메타데이터를 포함한다. 요청 수신 컴포넌트(145)는 특정 호스트 상에서 구동하는 복수의 또는 모든 어플리케이션들로부터의 요청들을 수신할 수도 있어서, 호스트는 메모리가 어떻게 사용되는지를 설명하는 광범위한 정보를 갖는다. 그 후, 호스트는 특정 액션(예를 들어, 메모리 압력을 감소하기 위해 메모리를 페이지 아웃하거나 해제하는 것)을 취하도록 결정할 시에 어느 할당들이 영향을 주는지를 우선순위화할 수 있다.

[0059] 요청 저장 컴포넌트(150)는, 실시한 메모리 관리 결정 동안 후속 사용을 위해 수신된 요청들 및 관련 메타데이터 정보를 저장한다. 호스트는, 요청이 수신된 시간에, 할당 요청을 충족시킬 수도 있거나 충족시키지 않을 수도 있다. 하지만, 호스트가 요청을 즉시 충족시키든지 또는 더 나중에 충족시키든지, 호스트가 메모리 액션이 적절할 경우 더 나중에 이용가능한 정보를 가질 수 있도록, 할당과 연관하여 어플리케이션에 의해 제공된 정보를 저장한다. 일부 실시예들에 있어서, 호스트는, 어플리케이션이 메모리를 할당한 이후 어플리케이션으로부터의 별도의 통신에 있어서 메타데이터 및 다른 정보를 수신한다. 그러한 경우들에 있어서, 호스트는 할당 정보를, 복수의 시간들에서 또는 유용한 정보가 이용가능할 때에 저장할 수도 있다.

[0060] 할당 컴포넌트(155)는 메모리를 할당하기 위한 실제 액션을 수행하여, 하나 이상의 수신된 어플리케이션 할당 요청들을 충족시킨다. 할당 컴포넌트(155)는, 호스트에 액세스가능하고 호스트에 의해 관리되는 컴퓨터 시스템에 인스톨된 물리적 메모리로부터 직접 또는 어플리케이션 힙으로부터 메모리를 할당할 수도 있다. 할당 이후, 메모리는 특정 어플리케이션에 전념하고, 다른 어플리케이션들에 의해 사용될 수 없다(통상의 공유된 메모리 기술들을 통해 그 다른 어플리케이션들이 그렇게 실시하도록 요청하지 않으면). 할당 컴포넌트(155)는 페이지 테이블들 및 가상 메모리와 작용하여 윈도우를 할당을 위한 물리적 메모리로 제공할 수도 있으며, 제공된 메모리는 스왑 파일을 통해 디스크 또는 다른 저장부에 의해 지원될 수도 있다.

[0061] 메모리 액션 컴포넌트(160)는 디바이스 상의 메모리를 관리하기 위한 액션을 수행하고, 이전에 저장된 수신 요청 정보에 액세스하여 수행된 액션에 의해 영향을 받을 하나 이상의 적절한 메모리 할당들을 결정한다. 액션들은 스왑 파일을 통해 메모리 콘텐츠를 스와핑하는 것, 이전에 할당된 메모리를 해제하는 것, 어플리케이션이 그 메모리 풋프린트를 감소시킬 것을 요청하는 것, 또는 하나 이상의 어플리케이션들에 의해 사용된 메모리에 영향을 주는 임의의 다른 액션을 포함할 수도 있다. 통상적으로, 메모리 관리 액션을 수행함에 있어서의 호스트의 목적은 낮은 메모리(예를 들어, 메모리 압력) 와 같은 현재의 또는 임박한 조건을 핸들링하는 것, 어플리케이션 요구를 충족시키기에 충분한 메모리를 수집하는 것 등등이다.

[0062] 데이터 저장 컴포넌트(165)는 하나 이상의 어플리케이션들 대신 메모리를 관리하기 위해 호스트에 의해 사용된 정보를 저장한다. 데이터 저장 컴포넌트(165)는 시스템(100)에 의한 사용을 위해 확보된 메모리, 별도의 플래시 또는 디스크 저장부, 또는 데이터를 저장하기 위한 다른 설비들을 포함할 수도 있다. 데이터 저장 컴포넌트(165)는 또한, 그 어플리케이션에 관련된 할당들을 추적하는 각각의 어플리케이션의 메모리 공간에 저장된 데이터, 및 어플리케이션을 대신하여 메모리를 관리하기 위해 호스트에 의해 사용된 다른 정보를 포함할 수도 있다.

[0063] 메모리 관리 시스템이 구현되는 컴퓨팅 디바이스는 중앙 프로세싱 유닛, 메모리, 입력 디바이스(예를 들어, 키보드 및 포인팅 디바이스들), 출력 디바이스(예를 들어, 디스플레이 디바이스들), 및 저장 디바이스들(예를 들어, 디스크 드라이브들 또는 다른 비휘발성 저장 매체)을 포함할 수도 있다. 메모리 및 저장 디바이스들은, 시스템을 구현하거나 인에이블시키는 컴퓨터-실행가능 명령들(예를 들어, 소프트웨어)로 인코딩될 수도 있는 컴퓨터-판독가능 저장 매체이다. 부가적으로, 데이터 구조들 및 메시지 구조들은, 통신 링크 상의 신호와 같이, 데이터 송신 매체를 통해 저장 또는 송신될 수도 있다. 인터넷, 로컬 영역 네트워크, 광역 네트워크, 포인트-투-포인트 다이얼-업 커넥션, 셀 폰 네트워크 등과 같은 다양한 통신 링크들이 사용될 수도 있다.

[0064] 시스템의 실시예들은, 개인용 컴퓨터들, 서버 컴퓨터들, 핸드헬드 또는 랩탑 디바이스들, 멀티프로세서 시스템들, 마이크로프로세서 기반 시스템들, 프로그래머블 소비자 전자기기들, 디지털 카메라들, 네트워크 PC들, 미니 컴퓨터들, 메인프레임 컴퓨터들, 임의의 상기 시스템들 또는 디바이스들을 포함하는 분산 컴퓨팅 환경들, 셋탑 박스들, 시스템 온 칩(SOCs) 등을 포함한 다양한 오퍼레이팅 환경들에서 구현될 수도 있다. 컴퓨터 시스템들은 셀 폰들, 개인용 디지털 보조기들, 스마트 폰들, 개인용 컴퓨터들, 프로그래머블 소비자 전자기기들, 디지털 카메라들 등일 수도 있다.

[0065] 시스템들은, 하나 이상의 컴퓨터들 또는 다른 디바이스들에 의해 실행되는 프로그램 모듈들과 같은 컴퓨터-실행

가능 명령들의 일반적인 문맥에서 설명될 수도 있다. 일반적으로, 프로그램 모듈들은, 특정 태스크들을 수행하거나 특정 추상적 데이터 타입들을 구현하는 루틴들, 프로그램들, 오브젝트들, 컴포넌트들, 데이터 구조들 등을 포함한다. 통상적으로, 프로그램 모듈들의 기능은 다양한 실시예들에서 요구된 바와 같이 결합되거나 분배될 수도 있다.

[0066] 도 2는 일 실시예에 있어서, 메모리 관리 시스템의 오퍼레이팅 환경을 도시한 블록 다이어그램이다. 그 환경은 커널/호스트(210), 하나 이상의 어플리케이션들(220), 메모리 핸들링 프레임워크(230), 저장된 할당 메타데이터(240), 및 하나 이상의 툴들(250)을 포함한다. 커널/호스트(210)는 하나 이상의 어플리케이션들(220)에 의해 공유된 메모리를 관리한다. 메모리 핸들링 프레임워크(230)는, 어플리케이션들(220)이 메모리를 할당하기 위해 호출하고 메모리 할당 메타데이터를 제공할 수 있는 하나 이상의 API들을 제공한다. 메모리 핸들링 프레임워크(230)는 수신된 메타데이터(240)를 저장한다. 툴들(250)은, 정적 및 동적 분석을 통해 그러한 정보를 추출하기 위해 할당 정보를 고유하게 제공하도록 설계되지 않은 어플리케이션들에 대해 동작한다. 그 후, 임의의 추출된 정보는 할당 메타데이터(240)로서 저장된다.

[0067] 시스템 동작

[0068] 도 3은 일 실시예에 있어서, 메모리의 할당 및 사용을 요청하기 위한 소프트웨어 어플리케이션 내에서의 메모리 관리 시스템의 프로세싱을 도시한 플로우 다이어그램이다.

[0069] 블록 310에서 시작하여, 어플리케이션은, 어플리케이션이 메모리 할당을 어떻게 사용할 것인지를 설명하는 할당 메타데이터를 설정한다. 어플리케이션은 할당 함수로 전달하기 위한 파라미터 구조를 형성하거나, 할당 메타데이터를 제공하기 위한 API를 호출하거나, 또는 본 명세서에서 설명된 메모리 관리 시스템에 따라 메모리를 할당하기 위한 도출된 클래스의 파라미터를 설정할 수도 있다. 할당 메타데이터는 어플리케이션에 대한 메모리의 우선순위 레벨, 할당된 메모리를 사용하기 위해 어플리케이션이 얼마나 자주 계획되는지, 어플리케이션이 할당의 콘텐츠, 및 어플리케이션의 메모리 사용에 관련된 다른 정보를 대체하기가 얼마나 어려운지, 및 복수의 어플리케이션들에 의해 공유된 물리적인 메모리 리소스들을 효과적으로 사용하기 위해 호스트가 메모리를 어떻게 조작할 수 있는지를 포함할 수도 있다.

[0070] 블록 320에서 계속하면, 어플리케이션은, 메모리 할당의 콘텐츠를 정주시키는 메모리 충진 함수를 설정한다. 메모리 충진 함수는 파일로부터의 정보에 액세스하거나, 메모리 할당에 저장된 결과들을 생성하기 위해 하나 이상의 계산들을 수행하거나, 또는 메모리 할당을 정주시키는 다른 액션들을 수행할 수도 있다. 할당 직후에 충진 함수를 호출하는 것 대신에 메모리 충진 함수를 호스트에 제공함으로써, 어플리케이션은, 요구될 경우 메모리를 릴리스 및 재정주시키거나 또는 호스트에 대한 더 적절한 시간까지 메모리의 할당 및 초기 정주를 연기하는데 요구되는 정보를 호스트에게 제공한다.

[0071] 블록 330에서 계속하면, 어플리케이션은, 복수의 어플리케이션들에 걸쳐 공유된 물리적 메모리를 관리하는 호스트에 의해 제공된 할당 인터페이스를 호출하며, 여기서, 어플리케이션은 설정된 할당 메타데이터 및 메모리 충진 함수를 할당 인터페이스를 통해 호스트에 제공한다. 할당 인터페이스는 오퍼레이팅 시스템 API, 런타임 함수, 또는 메모리 관리를 책임지는 호스트와 어플리케이션 간의 정보를 전달하는 다른 방식을 포함할 수도 있다. 할당 인터페이스는 오늘날 메모리 할당들에 통상 제공되는 것보다 더 많은 정보를 제공하며, 컴퓨팅 디바이스 상의 메모리를 더 효율적으로 관리하기에 유용한 정보를 호스트에 제공한다.

[0072] 블록 340에서 계속하면, 어플리케이션은 어플리케이션 인터페이스를 호출하는 것에 응답하여 레퍼런스를 수신하며, 여기서, 레퍼런스들은 어플리케이션에 의한 할당된 메모리의 후속적인 사용을 위한 간접 식별자로서 기능한다. 호스트는, 요청을 수신 시 메모리를 즉시 할당할 수도 있거나 즉시 할당하지 않을 수도 있다. 부가적으로, 호스트는, 어플리케이션이 메모리를 사용하기 전에 호스트가 다시 제어를 수신하는 것이 필요한 방식으로, 때때로(예를 들어, 메모리를 릴리스할 때, 디스크에 페이징할 때 등등) 메모리를 수정할 수도 있다. 따라서, 호스트는 메모리에 대한 레퍼런스를 어플리케이션에게 제공하며, 어플리케이션이 메모리를 사용하길 원할 경우, 어플리케이션은 그 레퍼런스를 제공하여 메모리로의 직접 액세스(예를 들어, 포인터)를 획득한다.

[0073] 결정 블록 350에서 계속하면, 어플리케이션은 어플리케이션이 할당된 메모리로 종료됨을 검출하면, 어플리케이션은 완료하고, 그렇지 않으면, 어플리케이션은 블록 360에서 계속한다. 완료하기 전, 어플리케이션은 할당 인터페이스를 호출하여 이전에 할당된 메모리를 할당해제(또는 해제)할 수도 있다. 어플리케이션이 메모리를 사용하지 않았고 호스트가 메모리를 실제로 할당하지 않았으면, 이 액션은 단순히, 할당에 관련된 호스트의 저장된 엔트리를 클린업하고 제어를 어플리케이션에 리턴할 수도 있다.

[0074] 결정 블록 360에서 계속하면, 어플리케이션이 어플리케이션에 있어서의 할당된 메모리의 사용을 검출하면, 어플리케이션은 블록 370에서 계속하고, 그렇지 않으면 어플리케이션은 350으로 돌아가서 메모리 할당의 목적의 메모리 액세스 또는 완료를 대기한다. 어플리케이션은 메모리가 어플리케이션 소프트웨어 코드에서 액세스될 때마다 특정 함수를 호출할 수도 있거나, 메모리를 이용가능하게 하도록 하는 호스트에 대한 적절한 호출들을 사용하여 메모리의 액세스들을 랩핑하는 클래스에서 메모리 할당을 캡슐화할 수도 있다.

[0075] 블록 370에서 계속하면, 어플리케이션은 호스트로부터의 메모리 할당에 대한 직접 액세스를 요청한다. 호스트가 메모리를 이미 할당하였으면, 호스트는, 메모리가 액세스될 수 있는 어플리케이션으로 포인터를 리턴한다. 호스트가 메모리를 할당하지 않았거나 메모리를 할당해제 및 재사용하였으면, 호스트는 어플리케이션 요청에 응답하여 메모리를 할당하고, 메모리 콘텐츠를 정주시키기 위해 수신된 충진 함수를 호출하며, 그 후, 메모리에 액세스하는 포인터 또는 다른 수단을 어플리케이션에 리턴한다.

[0076] 블록 380에서 계속하면, 어플리케이션은 호스트로부터의 메모리에 대한 수신된 어드레스를 통해 메모리 할당에 액세스한다. 어플리케이션은 메모리를 수정하거나, 메모리로부터 정보를 판독하거나, 또는 다른 통상적인 메모리 동작들을 수행할 수도 있다. 어플리케이션이 메모리에 액세스하는 것을 통할 경우, 어플리케이션은, 메모리를 이용불가능하게 할 수도 있는 관리 동작들을 호스트가 다시 수행할 수 있음을 호스트에게 표시할 수도 있다. 어플리케이션은 충진 함수에 의해 사용된 데이터를 업데이트할 수도 있어서, 호스트가 다시 메모리를 재정주하면, 메모리는 최근의 변화들(있다면)을 포함할 것이다. 블록 380 이후, 이들 단계들은 종료한다.

[0077] 도 4는 일 실시예에 있어서, 어플리케이션 요청들을 수신하여 메모리를 할당 및 사용하기 위한 호스트 내에서의 메모리 관리 시스템의 프로세싱을 도시한 플로우 다이어그램이다. 블록 410에서 시작하여, 호스트는 어플리케이션으로부터 메모리 할당 요청을 수신한다. 호스트는 복수의 어플리케이션들로부터의 요청들을 서비스할 수도 있으며, 호스트 플랫폼 상에서 구동하는 복수의 어플리케이션들 중에서 제한된 물리적 메모리를 슬라이싱하거나 공유하는 메모리 매니저를 포함할 수도 있다. 메모리 할당 요청은 요청된 메모리 사이즈, 또는 요청된 메모리에 관련된 다른 파라미터들과 같은 정보를 포함할 수도 있다.

[0078] 블록 420에서 계속하면, 호스트는, 어플리케이션이 요청된 메모리 할당을 사용하기 위해 계획한 방식을 명시하는 메모리 할당 메타데이터를 수신한다. 메타데이터는 어플리케이션으로의 메모리 할당의 중요도 또는 우선순위, 할당이 디스크로의 페이징에 적합한지 여부, 호스트가 할당을 해제해야 한다면 어플리케이션이 메모리의 콘텐츠를 복원할 수 있는지 여부 등등과 같은 정보를 포함할 수도 있다.

[0079] 블록 430에서 계속하면, 호스트는 요청된 메모리 할당의 콘텐츠를 정주시키기 위해 호스트에 의해 호출될 수 있는 메모리 충진 함수를 어플리케이션으로부터 수신한다. 메모리 할당을 충진하기 위한 수단을 갖는 것은 호스트로 하여금 메모리의 할당을 연기하게 할 뿐 아니라, 호스트가 다른 어플리케이션 요청을 충족시키거나 다른 메모리 압력을 경감시키기 위해 메모리 할당을 해제해야 할 필요가 있으면 메모리를 복원하게 한다.

[0080] 결정 블록 440에서 계속하면, 호스트가 요청에 응답하여 메모리를 직접 할당할 수 있음을 호스트가 결정하면, 호스트는 블록 450에서 계속하고, 그렇지 않으면, 호스트는 블록 460에서 계속한다. 호스트는, 메모리가 어플리케이션에 대한 것임을 어플리케이션이 표시한 것이 얼마나 중요한지 및 메모리에 액세스하도록 어플리케이션이 얼마나 자주 계획하는지에 기초하여 메모리를 직접 할당할지를 결정할 수도 있다. 호스트는 또한, 호스트가 구동하고 있는 컴퓨팅 디바이스가 요청 시에 얼마나 바쁜지와 같은 다른 팩터들을 고려할 수도 있다.

[0081] 블록 450에서 계속하면, 호스트는, 호스트에서 이용가능한 물리적 메모리로부터의 요청된 메모리를 할당한다. 호스트는, 어플리케이션이 물리적 메모리에 액세스하는 페이지 테이블들 또는 다른 가상 메모리를 셋업할 수도 있다. 통상적인 호스트들은 어플리케이션 요청들에 대한 직접적인 응답으로 메모리를 할당하거나, 호스트가 그 요청을 충족할 수 없으면 그 요청을 실패시킨다. 하지만, 메모리 관리 시스템은 요청들을 연기하거나 다른 메모리 관리 액션들을 수행하여, 메모리 및 다른 호스트 리소스들을 더 효율적으로 사용할 수도 있다.

[0082] 블록 460에서 계속하면, 호스트는 수신된 메타데이터 및 충진 함수와 함께, 더 나중의 할당을 위해 수신된 요청을 저장한다. 호스트는, (더 많은 메모리를 이용가능하게 하는 것과 같은) 메모리 관리 액션이 요구될 경우를 선택하기 위한 다양한 어플리케이션들에 의해 요청된 메모리의 데이터 저장을 유지할 수도 있다. 어플리케이션에 의해 제공된 메타데이터는, 어느 어플리케이션들 및 어플리케이션들 내 할당들이 호스트의 메모리 관리 액션에 의해 적어도 부정적으로 영향을 받을 지를 결정하도록 호스트를 돕는다. 블록 460 이후, 이들 단계들은 종료한다.

[0083] 일부 경우들에 있어서, 어플리케이션의 개발자는 새로운 오퍼레이팅 시스템 특징들을 지원하기 위해 변경하는

것을 원하지 않거나 이용할 수 없다. 그러한 경우들에 있어서, 메모리 관리 시스템은, 도 5 및 도 6을 참조하여 더 설명되는 바와 같이, 메모리 할당들에 관한 정보를 발견하기 위해 어플리케이션 바이너리 정보를 분석함으로써 일부 강화된 메모리 관리를 여전히 제공할 수 있을 수도 있다.

[0084] 도 5는 일 실시예에 있어서, 메모리 할당 정보를 제공하기 위해 구체적으로 설계되지 않은 어플리케이션을 분석하기 위한 메모리 관리 시스템의 프로세싱을 도시한 플로우 다이어그램이다.

[0085] 블록 510에서 시작하여, 시스템은, 어플리케이션에 의해 실시된 메모리 할당들을 나타내는 정보가 이용가능하지 않은 어플리케이션을 검출한다. 그 어플리케이션은, 메모리 관리 시스템에 의해 제공된 메모리 모델에 그 어플리케이션이 참여하는지 여부를 명시한 그 바이너리 모듈에 관한 플래그 또는 다른 정보를 포함할 수도 있거나, 또는 다른 표시를 제공할 수도 있다. 시스템은 이전에 결정된 어플리케이션 할당 정보를 캐싱할 수도 있어서, 각각의 어플리케이션에 대한 분석이 오직 1회 수행된다. 따라서, 이 단계에 있어서, 시스템은 또한, 분석이 이전에 수행되지 않았음을 결정한다.

[0086] 블록 520에서 계속하면, 시스템은, 어플리케이션에 의해 실시된 메모리 할당들을 결정하기 위해 어플리케이션에 관한 분석을 수행한다. 그 분석은 어플리케이션의 정적 및/또는 동적 분석의 다양한 형태들을 포함할 수도 있다. 정적 분석은 어플리케이션을 구동시키지 않고 발생하며, 어플리케이션의 바이너리 코드를 조사하여 어플리케이션의 거동을 결정한다. 동적 분석은 어플리케이션이 구동하고 있는 동안에 발생하며, 어플리케이션의 메모리 내 코드, 데이터 구조들, 및 다른 정보를 조사하여 어플리케이션의 거동을 결정한다. 시스템은 메모리를 할당, 액세스, 및 해제하기 위한 특정 호스트 API들에 대한 호출들을 찾을 수도 있으며, 위치들, 주위의 보충 정보, 할당된 메모리를 충진하는데 사용된 단계들 등등을 주목할 수도 있다.

[0087] 블록 530에서 계속하면, 시스템은, 어플리케이션이 메모리를 할당하는 어플리케이션 내 위치들에서 할당 정보를 제공하기 위해 어플리케이션을 후킹한다. 프로그램에 있어서의 임의의 포인트에서 어플리케이션의 통상의 거동을 차단하거나 증대시키기 위해 어플리케이션 바이너리 코드의 재지향을 허용하는 마이크로소프트 TM 우회들 및 다른 것들과 같은 다양한 형태들의 어플리케이션 후킹 기술들이 이용가능하다. 예를 들어, 시스템은, 어플리케이션에 의해 원래 호출된 메타데이터없는 표준 할당 함수 대신 할당 메타데이터를 제공하는 할당 함수를 호출하는 어플리케이션 후크를 제공할 수도 있다.

[0088] 블록 540에서 계속하면, 시스템은 메모리를 할당하기 위한 어플리케이션으로부터의 요청, 및 후킹된 어플리케이션 코드에 의해 제공된 관련 할당 정보를 수신한다. 그 요청은, 할당 정보를 제공하기 위해 구체적으로 설계되지 않은 어플리케이션으로부터 할당 정보가 유래하고 있음을 결정할 수도 있는 호스트에 의해 수신된다. 호스트는, 시스템과 작용하도록 설계된 어플리케이션들보다는 그러한 어플리케이션들에 대해 상이하게 거동할 수도 있다. 예를 들어, 호스트는, 시스템과 작용하도록 설계되지 않은 어플리케이션들에 의해 제공되는 잠재적으로 더 적은 정보를 설명할 수도 있다. 일부 실시예들에 있어서, 새로운 메모리 모델들의 채택을 권장하기 위해, 시스템은, 디스크에 대한 그 데이터를 교환하거나 또는 메모리 압력이 발생할 경우에 다른 메모리 관리 액션들을 수행하는 것을 선호함으로써 새로운 메모리 모델을 따르지 않는 어플리케이션들에게 페널티를 줄 수도 있다. 블록 540 이후, 이들 단계들은 종료한다.

[0089] 도 6은 일 실시예에 있어서, 어플리케이션을 정적으로 분석하고 강화된 메모리 정보에 대한 매니페스트를 제공하기 위한 메모리 관리 시스템의 프로세싱을 도시한 플로우 다이어그램이다.

[0090] 블록 610에서 시작하여, 시스템은, 메모리 할당 함수들에 대한 호출들과 관련하여 메모리 할당 정보를 제공하지 않는 컴파일된 어플리케이션 코드를 수신한다. 시스템은 어플리케이션을 구동하기 위한 제 1 요청 시, 컴퓨팅 디바이스의 하드 드라이브 또는 다른 저장부의 스캔으로부터의 벌크 프로세싱 동안, 또는 특정 사용자 또는 관리자 요청에 응답하여, 어플리케이션 코드를 수신할 수도 있다. 컴파일된 어플리케이션 코드는 특정 프로세서에 대한 바이너리 코드(예를 들어, x86 또는 x64 바이너리 코드), 특정 런타임에 대한 중간 언어 코드, 또는 다른 형태들의 비-소스 어플리케이션 코드를 포함할 수도 있다. 소스가 이용가능할 경우, 어플리케이션은 본 명세서의 다른 곳에서 설명된 바와 같이 시스템을 사용하기 위해 직접 수정될 수 있다.

[0091] 블록 620에서 계속하면, 시스템은, 어플리케이션이 메모리를 할당하는 코드 내 위치들을 결정하기 위해 수신된 어플리케이션 코드에 관한 정적 분석을 수행한다. 정적 분석은, 예를 들어, 임포팅된 모듈들 또는 다른 수단을 통해, 메모리 할당 함수들에 대한 호출들을 찾을 수도 있다. 정적 분석은 함수들에 대한 호출들뿐 아니라 일부 경우들에 있어서 전달된 파라미터들을 찾는데 익숙하다. 일부 실시예들에 있어서, 시스템은 정적 분석을 동적 분석으로 증대시키고, 정적으로 이용가능하지 않거나 용이하게 검출가능하지 않는 정보를 식별하기 위해 어플리

케이션을 구동한다.

[0092] 블록 630에서 계속하면, 시스템은 분석된 어플리케이션 코드에 있어서 하나 이상의 메모리 관련된 코드 액션들을 식별한다. 코드 액션은 메모리 할당, 메모리 액세스, 메모리 릴리스, 및 다른 메모리 동작들을 포함할 수도 있다. 시스템은 액션들이 발생하는 어드레스에 의해, 또는 호스트에게 더 나중에 제공되거나 디폴트 어플리케이션 거동을 변경하도록 수정될 수 있는 다른 식별에 의해, 코드 액션들을 식별할 수도 있다.

[0093] 블록 640에서 계속하면, 시스템은, 할당된 메모리가 어플리케이션에 의해 어떻게 사용되는지를 설명하는 부가적인 정보를 제공하는 각각의 식별된 코드 액션에 관련된 주위 정보를 식별한다. 주위 정보는 할당이 얼마나 오래 저장되는지(예를 들어, 동일 함수에서 1회 사용되는지 또는 반복된 후속 사용을 위해 글로벌 변수에 저장되는지), 어플리케이션이 메모리 할당의 콘텐츠를 얼마나 용이하게 정주시키는지 등등을 시스템에게 알릴 수도 있다. 주위 정보는 정적 및/또는 동적 분석을 통해 식별될 수 있다.

[0094] 블록 650에서 계속하면, 시스템은 식별된 메모리 관련 코드 액션들 및 임의의 식별된 주위 정보를, 어플리케이션을 구동할 시 호스트에 의한 후속적인 취출을 위해 데이터 저장부에 저장한다. 일부 실시예들에 있어서, 시스템은 식별된 정보를, 어플리케이션 모듈로 저장된 매니페스트에 저장하여, 어플리케이션을 로딩하는 호스트가 어플리케이션에 의해 실시된 메모리 할당들을 설명하는 더 많은 정보를 호스트에 제공하도록 식별된 위치들의 임의의 차단 또는 후킹을 수행할 수 있다. 호스트는 이러한 정보를 이용하여, 예를 들어, 시스템이 유휴상태이거나 낮게 활용되는 동안, 어플리케이션이 분석에 의해 결정된 바와 같이 추후의 어떤 시간에 필요할 것인 메모리를 선제적으로 할당할 수 있다. 이는 레거시 어플리케이션들로 하여금 메모리 관리 시스템에 의해 제공된 메모리 모델에 참여하게 한다. 블록 650 이후, 이들 단계들은 종료한다.

[0095] 도 7은 일 실시예에 있어서, 검출된 메모리 압력에 응답하여 메모리에 관련된 액션을 취하기 위한 메모리 관리 시스템의 프로세싱을 도시한 플로우 다이어그램이다. 메모리 압력은, 한정된 메모리를 갖는 디바이스가 그 이용가능한 메모리 근처에 있을 경우에 발생할 수 있다. 예를 들어, 시스템 오퍼레이팅 시스템들은, 물리적 RAM의 90%가 사용되었을 경우에 발생하는 것으로서 메모리 압력을 정의할 수도 있다.

[0096] 블록 710에서 시작하여, 시스템은 복수의 메모리 할당 요청들, 및 메모리 할당들이 하나 이상의 어플리케이션들에 의해 어떻게 사용되는지를 나타내는 정보를 수신한다. 일부 경우들에 있어서, 시스템은, 어플리케이션들로부터 수신될 때 할당들의 리스트 또는 다른 데이터 구조뿐 아니라, 각각의 할당이 어떻게 사용되는지를 설명하는 관련 메타데이터를 저장한다. 할당 사용 정보는, 이용가능한 메모리를 효율적으로 관리하기 위해 어느 할당들이 해제되거나, 페이징되거나, 또는 그렇지 않으면 핸들링될 수 있는지를 시스템이 결정하는 것을 도울 수 있는 우선순위 또는 다른 정보를 포함할 수도 있다.

[0097] 블록 720에서 계속하면, 시스템은, 어플리케이션들을 효율적으로 계속해서 구동시키기 위한 동작의 필요성을 나타내는 메모리 압력을 검출한다. 검출된 메모리 압력은, 커널 또는 다른 호스트가 취할 수 있는 다양한 조건들 또는 액션들을 포함할 수도 있다. 예를 들어, 커널은 전력을 절약하기 위해 메모리의 뱅크를 스위치 오프하길 원할 수도 있으며, 그 뱅크 상에 저장된 할당들을 검출하고 그 할당들을 릴리스 또는 교환할 수도 있어서 그 뱅크가 스위치 오프될 수 있게 한다. 다른 예로서, 시스템은 메모리 뱅크들을 디프래그먼트화하여, 스위치 오프될 수 있는 빈 뱅크들을 성취할 수도 있다. 호스트가 메모리의 사용의 완전한 제어에 있지 않은 경우, 어플리케이션 협력없이 메모리를 이동 또는 릴리스하는 동작들을 수행하는 것은 종종 어렵다. 하지만, 본 명세서에서 설명된 메모리 관리 시스템은, 메모리가 어떻게 관리되는지에 비해, 호스트에게 훨씬 더 많은 정보 및 제어를 허용한다.

[0098] 블록 730에서 계속하면, 시스템은 수신된 할당 요청들을 열거하여, 작용하는 할당들을 결정한다. 시스템은 할당들의 리스트 또는 다른 데이터 구조를 검사하여, 해제될 필요가 있을 경우 용이하게 복원될 수 있는 할당들 또는 어플리케이션이 다시 사용할 가능성이 없는 할당들을 결정할 수도 있다.

[0099] 블록 740에서 계속하면, 시스템은 작동하는 열거된 할당들 중 하나 이상을 선택한다. 시스템은, 할당을 요청하였던 어플리케이션에 의해 할당이 어떻게 사용되는지를 설명하는 수신된 정보에 기초하여 할당들을 선택할 수도 있다. 일부 경우들에 있어서, 시스템은 메모리 매니저의 특정 목표들(예를 들어, 메모리의 특정 총 사이즈에 대해 작용하는 것 및 그 사이즈까지 또는 그 사이즈를 초과하여 부가하는 할당들을 찾는 것)에 의존하여 복수의 할당들을 선택할 수도 있다.

[0100] 블록 750에서 계속하면, 시스템은 선택된 할당들에 관한 액션을 수행하여 메모리 압력을 경감시킨다. 그 액션들은 이전에 할당된 메모리를 해제하는 것, 메모리 콘텐츠를 디스크 또는 다른 저장부로 스와핑하는 것, 메모리

를 이전 위치로부터 새로운 위치로 이동시키는 것 등을 포함할 수도 있다. 일부 경우들에 있어서, 시스템은 액션을 어플리케이션에게 통지하여, 어플리케이션은 할당에 의존하는 거동을 수정할 수 있다. 예를 들어, 어플리케이션은, 그 어플리케이션에 의해 사용된 다음 시간에 메모리를 재할당함으로써 응답할 수도 있다. 블록 750 이후, 이들 단계들은 종료한다.

[0101] 도 8은 일 실시예에 있어서, 메모리가 호스트에 의해 이전에 수정되었던 어플리케이션을 활성화하기 위한 메모리 관리 시스템의 프로세싱을 도시한 플로우 다이어그램이다. 다수의 멀티태스킹 시스템들에 있어서, 어플리케이션들은 백그라운드로 푸쉬되고, 더 나중에, 사용자 또는 오퍼레이팅 시스템에 의해 재활성화된다. 모바일 디바이스들 상에서, 사용자들은 한번에 하나의 어플리케이션과 상호작용할 수도 있으며, 오퍼레이팅 시스템은 특정 어플리케이션이 전방에 있는 동안 다른 어플리케이션들을 알아챌 수도 있다. 오퍼레이팅 시스템은 다른 어플리케이션들의 메모리를 해제하거나 디스크 또는 다른 저장부로 스트리밍할 수도 있다. 활성 시, 오퍼레이팅 시스템은 다시 구동할 어플리케이션을 준비하도록 작동할 수도 있다.

[0102] 블록 810에서 시작하여, 시스템은 활성화하기 위한 어플리케이션 요청을 수신한다. 그 요청은 어플리케이션 자체로부터, 사용자로부터, 또는 오퍼레이팅 시스템으로부터 유래할 수도 있다. 활성화하기 위한 요청은, 어플리케이션이 이전에 할당된 메모리를 요청할 수 있도록 사용자가 그 어플리케이션과 한번더 상호작용하고 있음을 단순히 나타낼 수도 있다.

[0103] 블록 820에서 계속하면, 시스템은 하나 이상의 이전에 수신된 어플리케이션 메모리 할당들을 식별한다. 시스템은 하나 이상의 할당들에 의해 요청된 각각의 할당의 리스트 또는 다른 데이터 구조를 유지할 수도 있어서, 시스템은 그 리스트를 검사함으로써 이전에 수신된 메모리 할당들을 식별할 수 있다. 시스템은, 할당에 관련된 메모리가 여전히 이용가능한지 여부 및 어플리케이션에 의해 거기에 마지막으로 배치된 콘텐츠를 여전히 포함하는지 여부와 같이 각 할당의 상태를 체크할 수도 있다.

[0104] 결정 블록 830에서 계속하면, 어플리케이션의 할당들이 모두 준비된다고 시스템이 결정하면, 시스템은 블록 860으로 점프하고, 그렇지 않으면 시스템은 블록 840에서 계속한다. 시스템은 할당들을 해제하거나, 이동시키거나, 페이징하거나, 또는 그렇지 않으면 어플리케이션 상에서 작동시켜, 다른 어플리케이션들을 구동시키는 것과 같은 다른 태스크들에 대해 더 많은 메모리를 제공할 수도 있다. 따라서, 어플리케이션을 다시 구동할 때, 시스템은 이전에 작동된 할당들을 복원할 수도 있거나, 액션들을 어플리케이션에게 통지하여 그 어플리케이션이 적절한 액션을 취할 수 있다.

[0105] 블록 840에서 계속하면, 시스템은 준비되지 않은 할당들을 할당하여, 그 할당들을 어플리케이션에 의해 기대된 상태로 둔다. 시스템은 할당을 충족시키기 위해 이용가능한 물리적 메모리를 할당함으로써 물리적 메모리를 할당들에 할당할 수도 있다. 일부 경우들에 있어서, 시스템은, 예를 들어, 물리적 메모리 및 스왑 파일에 의해 지원되는 가상 메모리를 제공할 수도 있다.

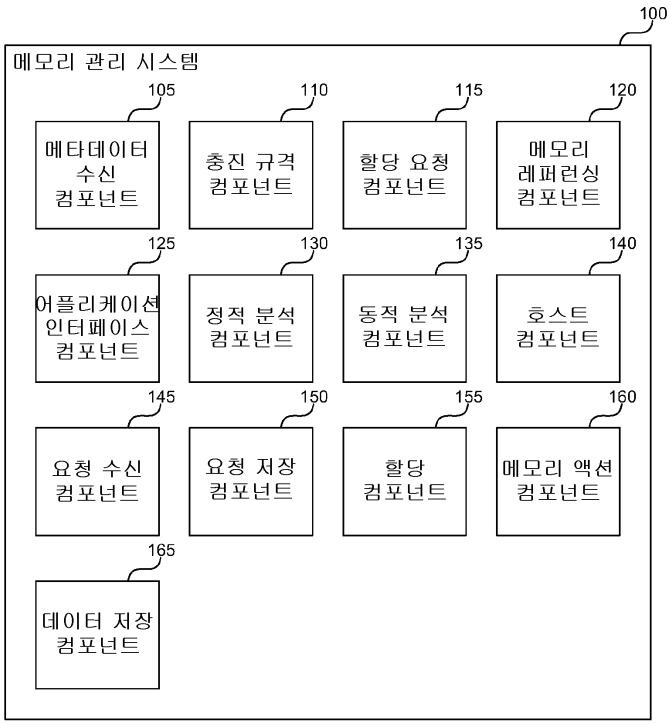
[0106] 블록 850에서 계속하면, 시스템은 어플리케이션에 의해 제공된 충진 함수를 사용하여 할당된 메모리 콘텐츠를 충진한다. 어플리케이션은 시스템에 충분한 정보를 제공하여, 시스템은 어플리케이션들의 메모리 할당들을 릴리스 및 재생할 수 있다. 이는 시스템으로 하여금 어플리케이션들에 대한 부정적 효과들을 낮게 유지시키면서 리소스들을 어플리케이션들 간에 공유하도록 효율적인 결정들을 행하게 한다. 이상적으로, 시스템이 메모리 또는 다른 압력 하에 있을 경우, 시스템은 어쨌든 바로 사용되지 않았을 메모리를 해제하거나 이동시키고, 그 후, 다시 요구되기 전에 메모리를 복원할 시간을 갖는다. 통상적인 시스템들에 있어서, 호스트가 할 수 있는 최상은 추측이지만, 본 명세서의 메모리 관리 시스템을 이용하여, 호스트는 작동하는 하나 이상의 메모리 할당들을 매우 효과적으로 선택할 수 있다.

[0107] 블록 860에서 계속하면, 시스템은 요청된 어플리케이션을 활성화하고, 어플리케이션에 의해 기대된 메모리 할당들을 어플리케이션에 제공한다. 시스템은 어플리케이션에 대한 메모리를 준비하면서 어플리케이션을 중지시키고, 그 후, 어플리케이션의 메모리 할당들 모두가 준비될 경우에 어플리케이션을 재개할 수도 있다. 블록 860 이후, 이들 단계들은 종료한다.

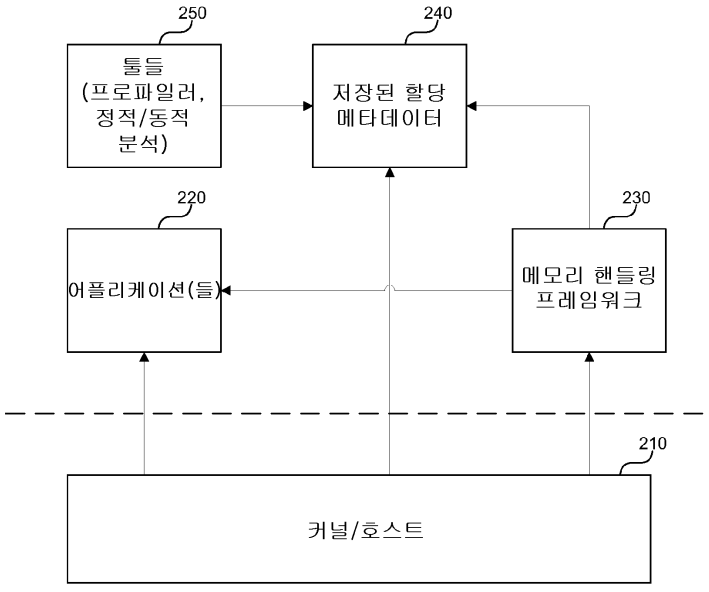
[0108] 전술한 바로부터, 메모리 관리 시스템의 특정 실시예들이 예시의 목적으로 본 명세서에서 설명되었지만, 다양한 변형들이 본 발명의 사상 및 범위로부터 일탈함없이 행해질 수 있음이 이해될 것이다. 이에 따라, 본 발명은, 첨부된 청구항에 의한 것을 제외하고 한정되지 않는다.

도면

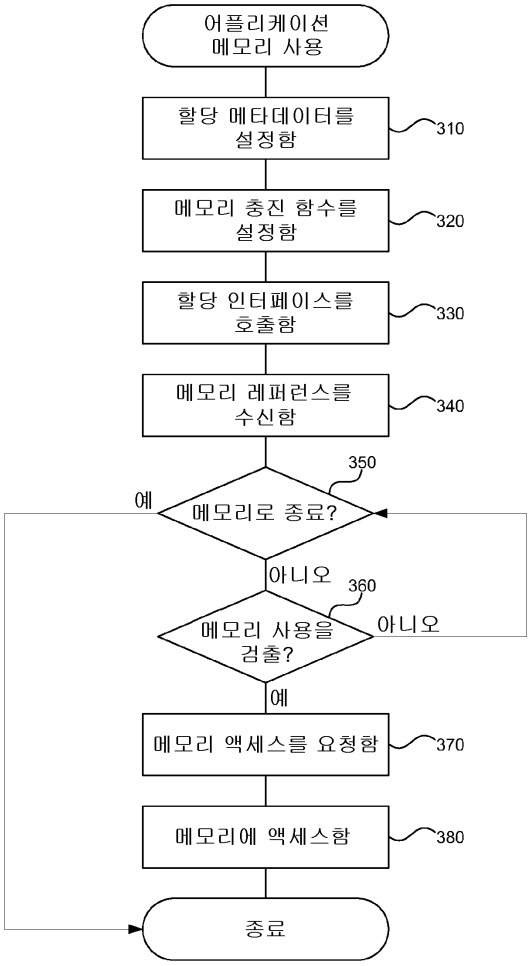
도면1


100
메모리 관리 시스템
105
메타데이터 수신 컴포넌트
110
충진 규격 컴포넌트
115
할당 요청 컴포넌트
120
메모리 레퍼런싱 컴포넌트
125
어플리케이션 인터페이스 컴포넌트
130
정적 분석 컴포넌트
135
동적 분석 컴포넌트
140
호스트 컴포넌트
145
요청 수신 컴포넌트
150
요청 저장 컴포넌트
155
할당 컴포넌트
160
메모리 액션 컴포넌트
165
데이터 저장 컴포넌트

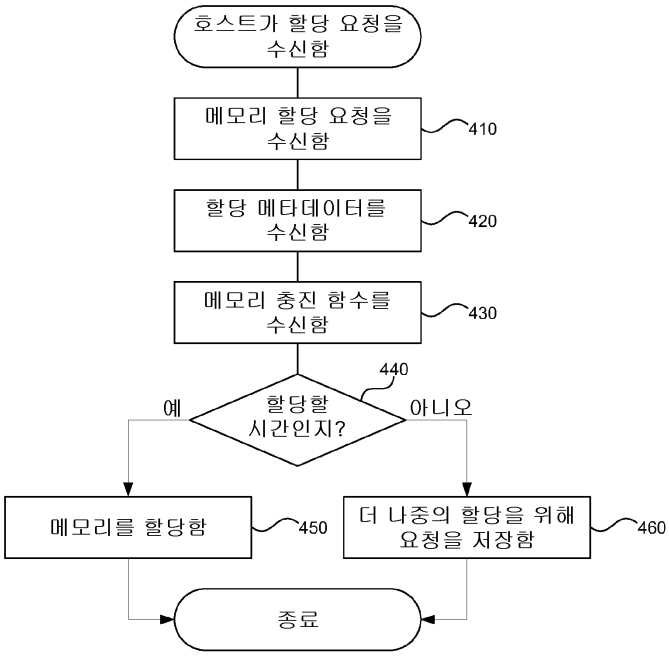
도면2


250
툴들 (프로파일러, 정적/동적 분석)
240
저장된 할당 메타데이터
220
어플리케이션(들)
230
메모리 핸들링 프레임워크
210
커널/호스트

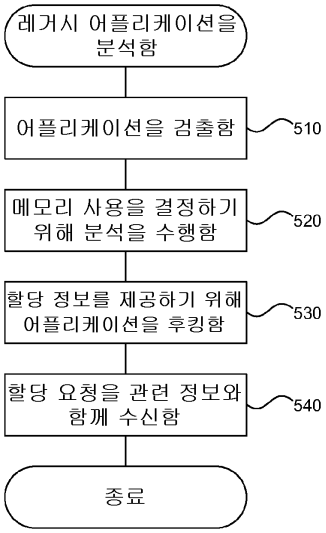
도면3


어플리케이션 메모리 사용
할당 메타데이터를 설정함
310
메모리 충진 함수를 설정함
320
할당 인터페이스를 호출함
330
메모리 레퍼런스를 수신함
340
350
메모리로 종료?
예
아니오
360
메모리 사용을 검출?
아니오
예
메모리 액세스를 요청함
370
메모리에 액세스함
380
종료

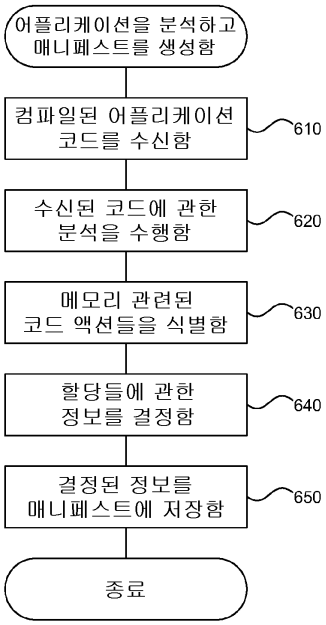
도면4


호스트가 할당 요청을 수신함
메모리 할당 요청을 수신함
410
할당 메타데이터를 수신함
420
메모리 충진 함수를 수신함
430
440
할당할 시간인지?
예
아니오
메모리를 할당함
450
더 나중의 할당을 위해 요청을 저장함
460
종료

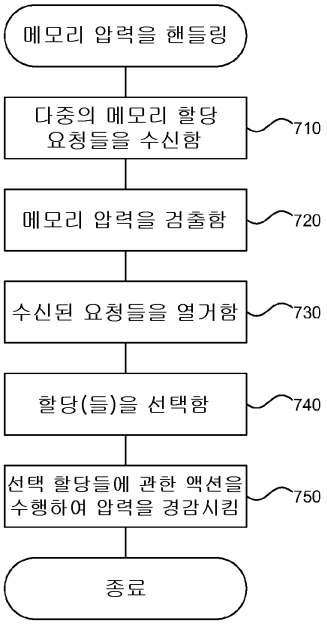
도면5


레거시 어플리케이션을 분석함
어플리케이션을 검출함
510
메모리 사용을 결정하기 위해 분석을 수행함
520
할당 정보를 제공하기 위해 어플리케이션을 후킹함
530
할당 요청을 관련 정보와 함께 수신함
540
종료

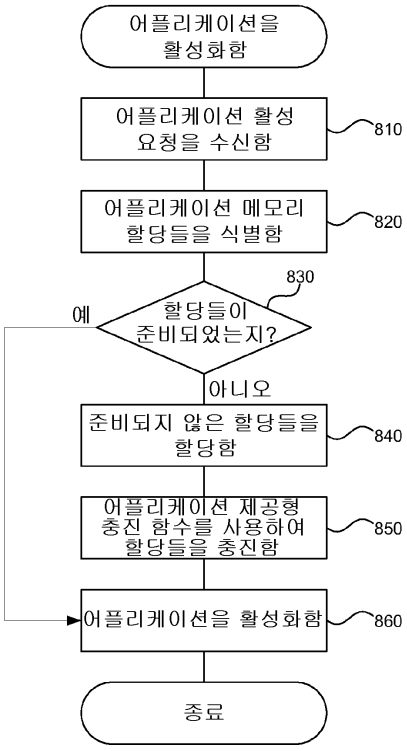
도면6


어플리케이션을 분석하고 매니페스트를 생성함
컴파일된 어플리케이션 코드를 수신함
610
수신된 코드에 관한 분석을 수행함
620
메모리 관련된 코드 액션들을 식별함
630
할당들에 관한 정보를 결정함
640
결정된 정보를 매니페스트에 저장함
650
종료

도면7


메모리 압력을 핸들링
다중의 메모리 할당 요청들을 수신함
710
메모리 압력을 검출함
720
수신된 요청들을 열거함
730
할당(들)을 선택함
740
선택 할당들에 관한 액션을 수행하여 압력을 경감시킴
750
종료

도면8


어플리케이션을 활성화함
어플리케이션 활성 요청을 수신함
810
어플리케이션 메모리 할당들을 식별함
820
830
할당들이 준비되었는지?
예
아니오
준비되지 않은 할당들을 할당함
840
어플리케이션 제공형 충진 함수를 사용하여 할당들을 충진함
850
어플리케이션을 활성화함
860
종료