공학석사학위논문

선사상 기법을 통한 안드로이드 스마트폰의 응답성 향상

Improving Responsiveness of Android Smartphone via Premapping Mechanism

2013년 2월

서울대학교 대학원

전기 컴퓨터 공학부

김 정 호

선사상 기법을 통한
안드로이드 스마트폰의 응답성 향상

Improving Responsiveness of Android Smartphone via Premapping Mechanism

지도교수 홍 성 수

이 논문을 공학석사학위논문으로 제출함

2013년 2월

서울대학교 대학원

전기 컴퓨터 공학부

김 정 호

김정호의 석사학위논문을 인준함

2013년 2월

위 원 장 문 수묵 (인)

부 위 원 장 홍 성수 (인)

위 원 김 태환 (인)

초록

안드로이드 스마트폰의 대표적인 킬러 응용인 웹 브라우저의 성능은 사용자 경험과 서비스 품질에 중요한 요소이다. 웹 브라우저는 수행 도중 메모리 집약적 작업을 빈번하게 수행하기 때문에 응답 시간 중 리눅스 커널의 메모리 관리 서브시스템의 수행 시간이 큰 비중을 차지한다. 구체적으로, 웹 브라우저가 새로운 웹 페이지를 읽어 들일 때 수천 번의 페이지 부재가 발생하고 이를 처리하는데 걸리는 시간이 전체 응답 시간에서 20% 가량의 비중을 차지한다. 따라서 페이지 부재 발생 횟수를 줄일 수 있다면 웹 브라우저의 응답 시간을 단축시킬 수 있다. 페이지 부재가 많이 발생하는 것은 현재의 리눅스 커널이 가지고 있는 동적 메모리 관리 기법인 요구 사상 기법 때문이다. 선사상 기법은 이러한 페이지 부재 발생을 효과적으로 줄일 수 있는 기법이지만, 선사상할 페이지를 예측하기 어렵기 때문에 기존 안드로이드 스마트폰에서는 요구 사상 기법이 사용되고 있다. 본 논문은 응답성 향상을 위해 커널이 안드로이드 런타임과 라이브러리의 도움을 받아 선사상할 페이지를 예측하는 선사상 기법을 제안한다. 실험 결과 제안된 기법은 기존 시스템에 비해 웹 브라우저 응용의 응답 시간을 최대 3.25% 단축할 수 있었다.

주요어: 안드로이드, 선사상, 동적 힙, 메모리 관리

학 번 : 2010-23251

목차

그림 목차

제 1 장 서론

안드로이드 스마트폰의 대표적인 킬러 응용인 웹 브라우저의 성능은 사용자 경험과 서비스 품질에 중요한 요소이다[1]. 웹 브라우저는 수행 도중 메모리 집약적 작업을 빈번하게 수행하기 때문에 응답 시간 중 리눅스 커널의 메모리 관리 서브시스템의 수행 시간이 큰 비중을 차지한다. 구체적으로, 웹 브라우저가 새로운 웹 페이지를 읽어 들일 때 수천 번의 페이지 부재가 발생한다. 이는 웹 브라우저의 렌더링 엔진이 HTML, XML 그리고 CSS 같은 웹 콘텐츠를 스크린에 그릴 때 필요한 수십 MB의 물리 메모리 공간이 페이지 부재 처리기에 의해 할당되기 때문이다. 페이지 처리기는 페이지 부재가 발생한 페이지에 대하여 하나의 프레임 4KB를 사상하며 약 0.03msec 정도가 소요된다. 따라서 웹 브라우저의 렌더링 엔진이 요구하는 수십 MB의 물리 메모리 공간을 할당하기 위해 수천 번의 페이지 부재 처리기가 수행되며 이는 전체 응답 시간에서 20% 가량의 비중을 차지한다.

페이지 부재 발생 횟수를 줄일 수 있다면 웹 브라우저의 응답 시간을 단축시킬 수 있다. 페이지 부재가 많이 발생하는 것은 현재의 리눅스 커널이 가지고 있는 동적 메모리 관리 기법인 요구 사상 기법 때문이다. 요구 사상 기법은 커널이 각 페이지에 실제 접근이 일어날 때 그 페이지만 페이지 부재 처리기에 의해 물리 메모리에 사상하는 기법이다. 따라서 웹 브라우저가 할당한 페이지들에 대한 접근이 실제로 일어나지 않을 때 메모리 오버헤드 감축에 유리하다.

선사상 기법은 가상 메모리의 페이지들을 물리 메모리에 미리 사상함으로써 페이지 부재 발생을 감소시킬 수 있는 기법이다. 따라서 이 기법

은 할당된 페이지들에 대한 접근이 일어날 확률이 높을 때 메모리 할당 시간 감축에 유리하다. 웹 브라우저 응용의 경우 동적으로 할당한 페이지들 대부분이 높은 확률로 가까운 미래에 접근되기 때문에 선사상 기법이 기존의 요구 사상 기법보다 효과적이다. 그러나 기존 안드로이드의 리눅스 커널은 웹 브라우저 응용을 다른 응용과 구분하지 못하고 선사상할 페이지를 예측하기 어렵기 때문에 단지 기존의 요구 사상 기법을 사용한다.

기존 연구들은 선사상할 페이지를 예측하기 위해 요구 선페이징(Demand Prepaging)을 사용한다. 요구 선페이징은 부재 페이지 시 부재 페이지 외에도 참조되지 않은 추가 페이지들을 한꺼번에 프레임들 연결하는 기법이다. 기존의 연구들은 순차 메모리 접근(Sequential Memory Access)을 위한 연구와 임의 메모리 접근(Random Memory Access)를 위한 연구로 나뉘는데 각각은 언급한 문제에 적용되기에 한계점들이 있다. 전자는 임의 메모리 접근에 방식을 갖는 웹 브라우저의 메모리 사용 방식에 적용되기 어렵다. 후자는 미래의 메모리 접근을 예측하기 위해 정적 분석 방법을 사용하였다. 이 연구들은 새로운 응용에 동적으로 적용되기 어렵다.

본 논문은 이 같은 한계를 극복하기 위해 커널의 페이지 부재 처리기가 런타임에 달빅 가상 머신과 바이오닉 C 라이브러리의 힌트를 받아서 선사상 할 응용과 페이지 주소를 예측하는 선사상 기법을 제안한다. 제안된 기법에서 페이지 부재 처리기는 현재 수행 중인 태스크가 웹 브라우저라는 사실과 동적으로 할당된 가상 메모리 공간의 힙 영역 주소를 힌트로 전달받아 이 영역에 대해 선사상을 수행한다. 실험 결과 제안된 기법은 스와이프 동작 중 페이지 부재 발생을 16% 감소시키고 이를 통해 응답시간을 3% 단축시킬 수 있었다.

본 논문의 나머지 부분은 다음과 같이 구성된다. 2장은 안드로이드 아키텍처에 대해 전반적으로 살펴본 후 메모리 관리에 대해 분석한다. 3장은 본 논문의 연구 동기를 설명하고 4장은 비효율적인 힙 사용 문제를 해결하기 위한 기존 연구들을 살펴본다. 5장은 비효율적인 힙 사용에 대해 문제를 정의한다. 6장은 정의된 문제를 해결하기 위해 적응적 요구 선사상을 제안한다. 7장은 제안된 해결책을 실험을 통해 검증한다. 8장 논문의 결론을 맺는다.

제 2 장 동기

본 장에서는 연구 동기가 된 실험을 설명한다. 이를 위해 사용된 실험 구성을 설명하고 추출된 실험 결과를 분석한다.

2.1 개괄

웹 브라우저는 대표적인 메모리 집약적 응용 중 하나이다. 웹 브라우저에서 웹 페이지 렌더링 도중 웹 페이지 당 수천 번의 페이지 부재가 발생한다. 본 연구팀은 먼저 웹 브라우저의 응답 시간 중에서 페이지 부재 처리기의 수행시간이 차지하는 비중을 분석하기 위해 다음과 같은 실험을 수행하였다.

2.2 실험 구성

대상 시스템은 안드로이드가 탑재된 Galaxy Nexus 스마트폰이다. 대상 시스템의 하드웨어는 Dual-core TI OMAP 4460 (@1.2 GHz) 프로세서와 1GB 램으로 구성된다. 한편 소프트웨어는 안드로이드 버전 AOSP (안드로이드 Open Source Project)의 Jellybean 4.0.9.99.999.9999.99999과 리눅스 커널 3.0-tuna가 사용되었다. 실험에 사용된 웹 브라우저는 안드로이드 기본 브라우저와 Firefox V15.0 브라우저 두 가지이다.

대상 시나리오는 네이버의 메인 웹 페이지를 스와이프 동작으로 넘기는 경우이다. 그림 1은 대상 시나리오를 그림으로 나타낸다. 본 연구팀은 이 시나리오의 응답 시간, 페이지 부재 발생 횟수, 각 페이지 부재 처리

기의 평균 수행 시간을 측정하였다. 네트워크에 의한 간섭을 배제하기 위해 웹 페이지를 미리 로딩하고 네트워크 연결을 끊은 채로 그 값들이 측정되었다.

그림 1. 웹 브라우저의 스와이프(Swipe) 동작.

실험에 사용된 측정 방법들은 proc 파일 시스템, TraceView, DDMS (Dalvik Debug Monitor Server) 그리고 KernelShark 이다. Proc 파일 시스템은 페이지 부재 발생 횟수를 측정하기 위해 사용되었으며, TraceView와 DDMS는 응답 시간을 측정하기 위해 사용되었다. 그리고 각 페이지 부재 처리기의 평균 수행 시간을 측정하기 위해 KernelShark 도구가 사용되었다. 우리는 페이지 부재 처리기의 호출 횟수를 측정하기 위해 proc 파일 시스템에 pagefaultinfo 파일을 생성하고 커널의 페이지 부재 처리기가 이 파일에 페이지 부재 발생 횟수를 기록하도록 하였다. pagefaultinfo에 기록되는 페이지 부재 처리기의 호출 횟수는 페이지 부재 시 entry 함수로 호출되는 do_fault_page() 함수가 호출될 때마다 1씩 증가된다. 이를 위해 수정된 파일은 /kernel/mm/vmstat.c, /kernel/mm/memory.c 그리고 /kernel/arch/arm/mm/fault.c이다.

2.3 실험 결과

실험 결과는 그림 2에 나타난 것과 같다. 응답시간 중 무려 약 21%가 페이지 부재 처리기의 수행시간으로 나타났다. 이는 새로운 웹 페이지를 로딩하기 위해 수많은 동적 메모리 할당이 일어나고 새로 할당된 각 페이지에 접근할 때 마다 페이지 부재가 발생해 총 만 번에 가까운 페이지 부재가 발생하기 때문이다.

웹 브라우저 종류	페이지 부재 발생 횟수 (번)	페이지 부재 처리기의 한 번 수행 시간 (ms)	페이지 부재 처리기의 총 수행 시간 (ms)	응답 시간 (ms)
기본 브라우저	9,326	0.03	279.78	1238.46
Firefox V15	13,637	0.03	409.11	1943 .28

그림 2. 실험 결과.

제 3 장 배경

본 장에서는 페이지 부재 발생 원인을 파악하기 위해 안드로이드 아키텍처를 전반적으로 살펴보고, 안드로이드의 메모리 관리 방식을 분석한다.

용어 혼란을 방지하기 위해 페이지와 프레임을 정의할 필요가 있다. 페이지는 가상 메모리 공간의 기본 단위이며 프레임은 물리 메모리 공간의 기본 단위이다.

3.1 안드로이드 아키텍처

3.1.1 개괄

안드로이드는 OHA(Open Handset Alliance)에 의해 개발된 모바일 기기용을 위한 리눅스 기반 운영체제이다.[2] 안드로이드는 크게 세 가지 계층들로 구성된 계층화 아키텍처이다. 그 세 가지 계층들은 응용 계층, 응용 프레임워크 계층, 리눅스 커널 계층이다. 그림 3은 안드로이드의 계층화 아키텍처를 나타낸다.

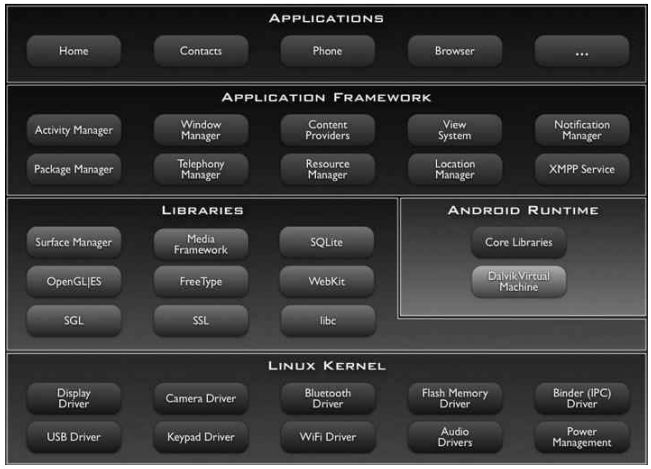

그림 3. 안드로이드의 계층화 아키텍처.

안드로이드 아키텍처에서 생성하는 리눅스 태스크는 크게 일곱 가지로 구분할 수 있다. 리눅스 태스크의 생성 순서는 init, 데몬, 서비스 매니저, Zygote, 시스템 서버, 런처 그리고 응용 순이다. 그림 4는 안드로이드의 리눅스 태스크 및 자바 프로세스 모델을 나타낸다. init, 데몬, 서비스 매니저는 달빅 가상 머신을 수행 중이지 않은 반면 Zygote, 시스템 서버, 런처 그리고 응용은 달빅 가상 머신을 수행한다. 따라서 Zygote, 시스템 서버, 런처 그리고 응용은 자바 코드를 수행할 수 있다.

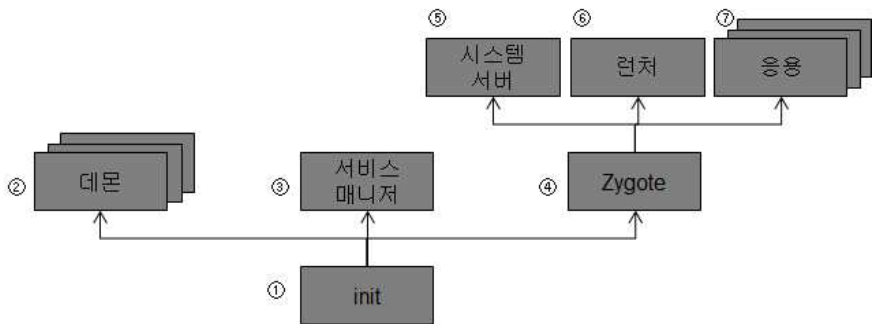

그림 4. 안드로이드에서 생성되는 태스크들.

3.1.2 리눅스 커널 계층

안드로이드는 리눅스 커널을 운영체제의 기능들을 제공하기 위해 사용한다. 그 핵심 서비스들은 메모리 관리, 네트워크 관리, 디스크 관리, 프로세스 관리, 장치 관리이다. 안드로이드에는 모바일 기기의 특성을 살리기 위해 여러 가지 기능들을 리눅스 커널 계층에 추가로 탑재하였다. 그 기능들로는 알람, 바인더 IPC(Inter-Process Communication), Ashmem(Anonymous Memory), 전력 관리 등이 있다.

부팅 시 리눅스 커널은 크게 세 가지 리눅스 태스크를 생성한다. 그 세 가지 리눅스 태스크는 init, 데몬 그리고 서비스 매니저이다. init와 데몬은 기존 리눅스 커널의 init의 역할을 갖고 있다. init는 각종 장치를 초기화하고 시스템에 필요한 프로세스를 생성한다. init에 의해 생성된 데몬은 런타임에 시스템에 필요한 기능들을 수행한다. 데몬으로는 ueventd, ksoftirqd, kworker, kthreadd 등이 있다. 서비스 매니저는 안드로이드 아키텍처에서 바인더 IPC를 위해 생성된다. 그것은 바인더 IPC를 사용하는 클라이언트와 서버 간의 관계 사상 테이블을 관리한다.

3.1.3 프레임워크 계층

응용 프레임워크 계층은 시스템 자원을 관리한다. 그 시스템 자원으로는 응용의 컴포넌트 생명 주기 관리, 디스플레이 관리, 오디오 관리, 무선 네트워크 관리 그리고 입력 이벤트 관리 등이 있다. 그리고 이는 응용 개발자에게 자바 개발 환경을 제공한다. 응용 프레임워크 계층은 자바 개발 환경을 응용 개발자에게 제공하기 위해 달빅 가상 머신을 갖는

다.

응용 프레임워크 계층의 서비스를 위해 생성되는 자바 프로세스들은 크게 네 가지가 있다. 그 두 가지 자바 프로세스는 Zygote, 시스템 서버이다. Zygote는 새로운 자바 프로세스를 빠르게 생성하기 위해 존재한다. Zygote는 자바 코드를 수행하기 위한 달빅 가상 머신 프로그램을 실행하여 꼭 필요한 클래스와 자원들을 적재한 후 자바 프로세스 요청을 기다린다. 시스템 서버는 모바일 기기의 시스템 자원을 관리하기 위한 자바 프로세스들이다. 시스템 서버의 자바 스레드로는 InputManager, WindowManager, WifiManager, AudioFlinger, SurfaceFlinger 등이 있다.

3.1.4 응용 계층

응용 계층은 사용자가 사용할 자바 프로세스를 제공한다. 기본적으로 안드로이드는 이메일 관리 응용, SMS 응용, 달력 응용, 지도 응용, 브라우저 응용, 연락처 응용 등을 제공한다. 모든 응용은 기본적으로 자바 언어로 작성되며 필요에 따라 C 언어로 작성된다.

런처는 안드로이드가 최초로 실행하는 응용이다. 마지막으로 런타임에 사용자의 요청에 따라 설치, 제거, 실행 그리고 종료가 가능한 응용이 있다.

3.1.5 리눅스 태스크와 자바 스레드

안드로이드의 메모리 동작 원리를 이해하기 위해 리눅스 태스크와 자바 스레드 또는 리눅스 태스크와 자바 프로세스 간 사상 관계를 파악하

는 것은 필수이다. 메모리를 표현하는 데이터 구조가 어느 태스크 간에 공유되는지 파악하기 위해 이 배경지식이 필요하다.

일반적으로 두 가지 프로세스 모델이 있다. 하나는 단일 스레드화 프로세스 모델(Single-Threaded Process Model)이며 다른 하나는 멀티 스레드화 프로세스 모델(Multi-Threaded Process Model)이다.[3] 그림 5는 두 가지 프로세스 모델을 나타낸다. 단일 스레드화 프로세스 모델은 하나의 프로세스가 하나의 스택을 갖는 반면 멀티 스레드화 프로세스 모델은 하나의 프로세스가 다수의 스레드 스택을 갖는다. 안드로이드에서 자바 프로세스와 자바 스레드 간에 모델은 멀티 스레드화 프로세스 모델이다.

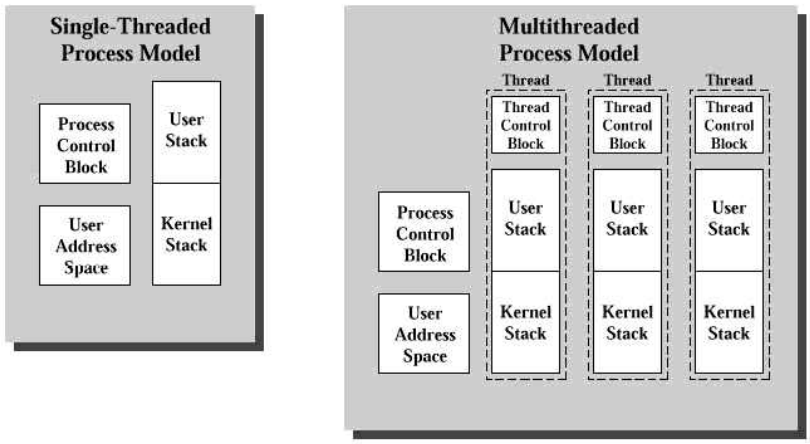

그림 5. 프로세스 모델[3].

자바 가상 머신에서 두 가지 주요한 두 가지 스레드 모델이 있다. 하나는 Green 스레드 모델이며 다른 하나는 Native 스레드 모델이다.[4] Green 스레드 모델은 하나의 리눅스 태스크가 다수의 자바 스레드와 사상 관계를 갖는 모델이며, Native 스레드 모델은 하나의 리눅스 태스크

는 하나의 자바 스레드와 사상 관계를 갖는 모델이다. 안드로이드의 달빅 가상 머신은 Native 스레드 모델을 갖는다.

안드로이드에서 리눅스 태스크, 자바 프로세스 그리고 자바 스레드를 생성하는 시스템 콜은 다음과 같다. 리눅스 태스크는 일반적으로 fork() 또는 clone() 시스템 콜에 의해서 생성된다. clone() 시스템 콜에 할당된 옵션에 따라 리눅스 태스크 간 공유 자원이 결정된다. 안드로이드에서 자바 프로세스 생성 시 Zygote 자바 프로세스를 fork() 한다. 자바 프로세스 생성 요청을 하기 위한 IPC 메커니즘은 소켓 인터페이스를 통한 IPC이다. Zygote는 select() 소켓 API 함수를 호출하고 수면 상태에 있다가 생성 요청에 따라 깨어나 자신을 fork()하고 복제된 자바 프로세스의 식별자를 요청한 자바 프로세스에게 전달한다. 마지막으로 자바 스레드 생성 요청 시 pthread_create() 함수가 호출되며 이 함수는 결국 clone() 시스템 콜을 호출한다. 호출되는 clone() 시스템 콜에 들어가는 옵션은 CLONE_FILES, CLONE_FS, CLONE_VM, CLONE_SIGHAND, CLONE_THREAD, CLONE_SYSVSEM이다. 이 옵션은 부모 스레드와 공유되는 자원은 파일 기술자, 파일 시스템 정보, 메모리 영역, 시그널 핸들러, 스레드 그룹, 세마포어 리스트이다.

3.2 달빅 가상 머신의 동적 힙 관리

3.2.1 개괄

달빅 가상 머신은 런타임에 생성되는 모든 자바 객체들을 자신이 관리하는 동적 힙에 할당한다. 런타임에 달빅 가상 머신이 자바 new 키워드를 만나면 자바 객체 할당자의 함수를 호출한다. 자바 객체 할당자는 동

적 힙에서 요청된 자바 객체 크기에 맞는 메모리 공간을 찾고 그 공간에 자바 객체를 할당한다.

동적 힙은 요구 사상 기법에 의해 물리 메모리로 사상된다. 자바 응용이 동적 힙의 특정 페이지에 접근했을 때 그 페이지에 대하여 페이지 부재가 발생한다. 커널의 페이지 부재 처리기가 프레임 하나를 할당하여 페이지 부재가 발생한 페이지와 사상한다.

3.2.2 디폴트 힙과 동적 힙

디폴트 힙은 응용 실행 시 로더에 의해 할당되는 연속적인 가상 메모리 공간인 반면 동적 힙은 자바 프로세스 생성 시 달빅 가상 머신에 의해 동적으로 할당되는 연속적인 가상 메모리 공간이다. 힙 할당 시 디폴트 힙은 힙의 크기를 요구하며 동적 힙은 힙의 크기와 힙의 초기 위치를 요구한다. 디폴트 힙의 가상 메모리 공간에서 초기 위치는 로더에 의해 정해져있다. 디폴트 힙은 brk() 또는 sbrk() 시스템콜로 할당되며 동적 힙은 mmap() 시스템콜로 할당된다.

응용 프로그래머가 작성한 코드가 수행될 때 네이티브 응용은 디폴트 힙을 사용하고 자바 응용은 동적 힙을 사용한다. 달빅 가상 머신은 네이티브 응용이기 때문에 디폴트 힙을 사용한다. 네이티브 응용이 동적 메모리 할당 함수를 호출하면 디폴트 힙에 할당하며 자바 응용이 자바 객체를 생성하면 동적 힙에 할당된다.

네이티브 응용의 전역 변수는 데이터 섹션에 할당되고 동적으로 할당된 변수는 디폴트 힙에 할당된다. 반면 자바 응용의 모든 변수는 동적 힙에 할당된다. 따라서 일반적으로 자바 응용이 필요로 하는 동적 힙의 크기는 네이티브 응용이 필요로 하는 디폴트 힙의 크기보다 크다.

디폴트 힙은 커널의 메모리 관리에 의존적이며 동적 힙은 커널의 메모리 관리에 독립적이다. 따라서 자바 가상 머신들은 커널에 독립적으로 모든 플랫폼에서 동일한 프로그램 구동 환경을 제공하기 위해 커널에 독립적인 동적 힙으로 구현하였다. 동적 힙은 자바 가상 머신이 원하는 메모리 정책으로 사용될 수 있다. 반면 디폴트 힙은 커널에 의존적이므로 자바 가상 머신이 원하는 메모리 정책을 사용할 수 없다.

3.2.3 동적 힙의 리눅스 커널 데이터 구조

동적 힙은 가상 메모리 공간을 나타내는 커널의 데이터 구조로 표현된다. 그림6은 가상 메모리 공간 데이터 구조를 나타낸다. 그 커널의 데이터 구조로는 task_struct, mm_struct 그리고 vm_area_struct 세 가지가 있다. task_struct는 리눅스 태스크 실행을 위한 정보들을 담고 있으며 리눅스 하나 당 하나가 생성된다. mm_struct는 리눅스 태스크의 가상 메모리 영역에 관련된 정보들을 담고 있으며 task_struct하나 당 하나가 생성된다. vm_area_struct는 각각의 연속적인 가상 메모리 공간을 기술하며 하나의 연속적인 가상 메모리 공간 하나당 하나가 생성된다. vm_area_struct는 연속적인 가상 메모리 공간을 표현하기 위한 포인터들이 있는데 그 중 vm_start와 vm_end는 각각 그 공간의 시작과 끝 주소를 갖고 있다.

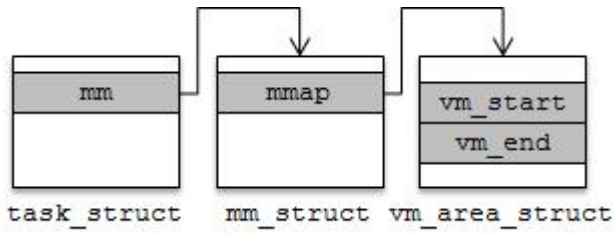

그림 6. 가상 메모리 공간 데이터 구조.

3.2.4 동적 힙 초기화

Zygote가 달빅 가상 머신을 실행하고 초기화 과정 중 동적 힙을 할당한다. Zygote는 안드로이드에서 부팅 시 달빅 가상 머신을 실행하고 자주 사용되는 클래스와 자원을 메모리에 미리 적재한 후 fork() 시스템 콜로 복제되길 기다리는 자바 스레드이다. Zygote가 달빅 가상 머신을 실행하고 이를 초기화 하는 과정에서 동적 힙을 anonymous 메모리 영역으로 할당한다. Anonymous 메모리 영역은 특정 파일과 사상 되지 않은 가상 메모리 영역을 말한다. 이 영역은 MAP_ANONYMOUS 플래그를 갖는 mmap() 시스템 콜에 의해 할당된다. 이는 특정 파일과 사상되지 않는 연속적인 메모리 공간을 할당하도록 요청한다.

그림 7을 통해 확인할 수 있듯이 mmap() 시스템콜을 통해 할당된 가상 메모리 공간은 vm_area_struct로 표현되어 커널이 관리한다. mmap() 시스템 콜은 가상 메모리 공간에서 연속적인 가상 메모리 공간을 찾을 때 최초 적합 할당(First Fit Allocation)을 한다. 그 할당된 메모리 공간은 응용 종료 시 munmap() 시스템콜을 호출하여 해제된다.

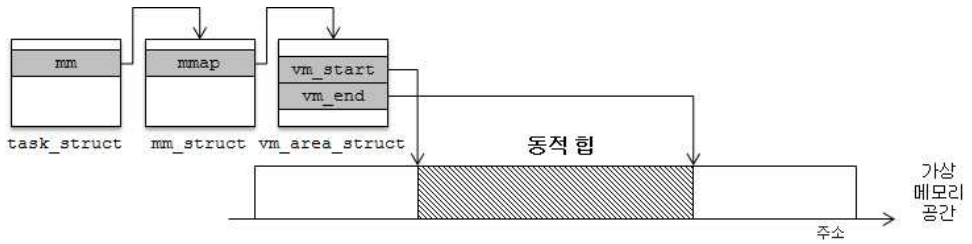

그림 7. vm_area_struct로 표현되는 동적 힙.

안드로이드에서 모든 자바 스레드는 Zygote에 의해 복제되기 때문에 모든 자바 스레드는 생성 시 추가 동적 힙 초기화 과정 없이 가상 메모리 공간에서 동적 힙을 갖고 있다.

3.2.5 자바 객체 할당 및 해제

달빅 가상 머신은 new 키워드에 해당하는 바이트 코드를 만나면 할당 요청한 객체의 크기대로 동적 힙에 할당한다. 런타입에 요청되는 크기는 최소 32B에서 수십 MB까지 다양하다. 달빅 가상 머신은 자바 객체를 동적 힙에 할당하기 위해 바이오닉 C 라이브러리의 API(Application Programming Interface)인 malloc() 함수를 사용한다. 달빅 가상 머신의 쓰레기 수집(Garbage Collector)은 표시하고 쓸기(Mark-and-Sweep) 알고리즘으로 자바 객체들을 해제한다. 해제 시 호출되는 함수는 바이오닉 C 라이브러리의 API인 free()함수이다.

3.2.6 동적 힙에 대한 물리 메모리 사상 기법

힙을 물리 메모리로 사상하는 기법에는 두 가지가 있다. 하나는 요구 사상이며 다른 하나는 선사상이다. 요구 사상은 각 가상 페이지에 실제 접근이 발생했을 때 페이지 부재에 의해 그 페이지만을 물리 메모리로

사상하는 기법이다. 이 기법은 동적으로 할당된 페이지들에 대한 접근 분포가 희박할 때 사상에 필요한 성능 오버헤드를 줄일 뿐만 아니라 물리 페이지를 절약할 수 있다는 장점이 있다. 한편 선사상은 실제 접근이 일어나지 않더라도 앞으로 사용될 것으로 예측되는 페이지들을 미리 물리 메모리에 사상하는 기법이다 [2][3][4][5]. 이 기법은 런타임에 페이지 부재에 의한 성능 오버헤드를 없앨 수 있다는 장점이 있지만, 가까운 미래에 접근될 페이지들의 주소와 개수를 쉽게 알 수 없는 경우 프레임들의 불필요한 낭비를 초래한다.

웹 브라우저 응용은 페이지의 렌더링을 수행하는 도중에 수많은 크고 작은 자바 객체를 동적으로 할당하고 이들을 높은 확률로 접근한다. 따라서 리눅스/안드로이드의 메모리 관리 서브시스템은 응용의 특성과 상관 없이 일률적으로 요구 사상 기법에 의해 가상 메모리의 사상을 수행하기 때문에 웹 브라우저가 새롭게 할당된 페이지에 접근할 때마다 페이지 부재가 발생한다.

그림 8은 2장에서 수행된 실험의 페이지 부재 발생 과정을 나타낸다. 사용자가 새 웹 페이지를 보기 위해 스와이프 동작을 하면 전체 화면 갱신을 위한 렌더링이 수행된다. 렌더링 과정 중 수 많은 작은 크기 자바 객체들이 동적 할당된다. 이 자바 객체들은 웹 브라우저 가상 메모리의 힙 영역에 할당되고 SurfaceFlinger에 의해 접근된다. 이 때, 페이지 부재 처리기에 의해 페이지 부재가 발생한 페이지가 물리 메모리로 사상된다.

그림 8. 2장에서 수행된 실험의 페이지 부재 발생 과정 개관.

3.2.7 안드로이드 응용의 동적 메모리 할당 과정

기존 리눅스/안드로이드는 언급한 두 가지 동적 메모리 사상 기법들 중 요구 사상 기법을 사용한다. 그림 9은 가상 주소 공간과 물리 주소 공간에서 동적 메모리 할당 과정을 나타낸다. 그림 9의 ①번 과정은 달빅 가상 머신이 힙을 초기화하는 과정을 나타낸다. 각 안드로이드 응용을 수행하는 달빅 가상 머신은 초기화 과정 중에 mmap() 시스템 콜을 통해 자신의 힙 영역을 가상 주소 공간에 할당한다. 이때 mmap() 시스템 콜의 인자로 MAP_ANONYMOUS 플래그가 넘겨지는데, 이는 특정 파일과 사상되지 않는 연속적인 메모리 공간을 할당하도록 요청한다.

그림 9의 ②번 과정은 응용이 동적으로 할당한 자바 객체를 달빅 가상 머신이 malloc() API를 통해 힙 영역에 할당하는 상황을 나타낸다. 자바 응용이 new 키워드를 통해 자바 객체를 할당한다. 이 응용의 바이트코드를 수행하는 달빅 가상 머신은 new 키워드가 호출된 것을 파악하면 dvmAllocObject() 함수를 호출한다. 이 함수는 malloc() 를 호출하여 자바 객체 할당을 위한 메모리 공간을 확보한다. malloc() 은 요청된 크기의 메모리 공간을 가상 주소 공간에 할당하고 시작 주소 값을 반환한다. 그림 9의 ③번 과정은 malloc() 에 의해 할당된 메모리 공간의 한 페이지에 실제 접근이 발생한 상황을 나타낸다. 이 경우 그 페이지에 대한

페이지 부재가 발생하여 페이지 부재 처리기의 시작 함수인 do_page_fault() 를 호출한다. 페이지 부재 처리기는 하나의 프레임을 할당하고 이를 그 페이지와 사상한다.

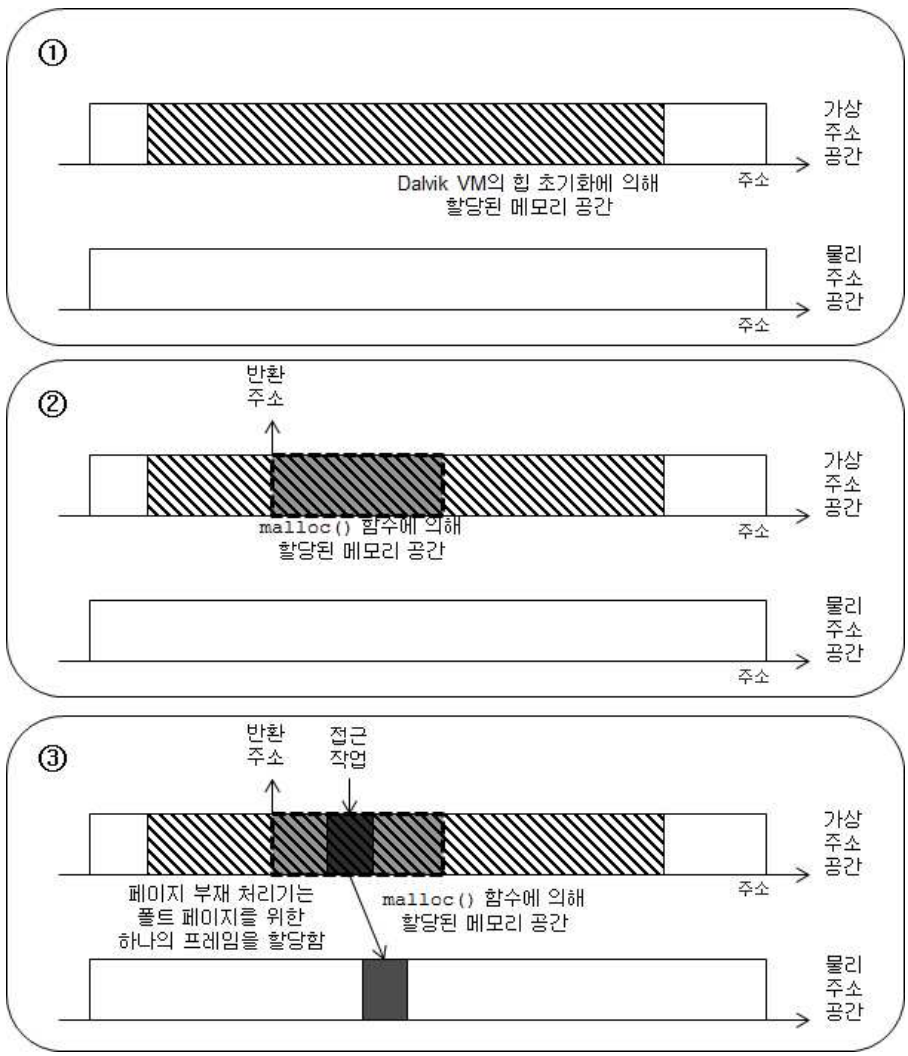

그림 9. 안드로이드 응용의 동적 메모리 할당 과정.

3.2.8 안드로이드 응용의 동적 힙의 확장 및 축소

그림 10는 달빅 가상 머신이 관리하는 힙 영역의 확장과 축소 과정을 나타낸다. 그림 10의 ①번 과정은 달빅 가상 머신이 관리하는 힙 영역이 확장되는 과정을 나타낸다. 어떤 자바 프로세스가 malloc() 으로 동적 메모리 할당을 요청했을 때 힙 영역에 그 할당을 위한 연속적인 메모리 공간이 부족한 경우 달빅 가상 머신은 그 힙 영역을 확장을 한다. 추가된 힙 영역은 MAP_ANONYMOUS 플래그를 갖는 mmap() 시스템 콜을 통해 확보된다. 이 확보된 영역은 기존의 힙 영역과 연속적인 메모리 공간일 수도 있고 불연속적인 메모리 공간일 수도 있다.

그림 10의 ②번 과정은 달빅 가상 머신이 관리하는 힙 영역이 축소되는 과정을 나타낸다. 어떤 자바 프로세스가 free() 로 동적 메모리 해제를 요청했을 때 힙 영역의 top-most 프리 공간이 2MB보다 큰 경우 힙 영역을 축소한다. 힙 영역은 munmap() 시스템 콜을 통해 축소된다.

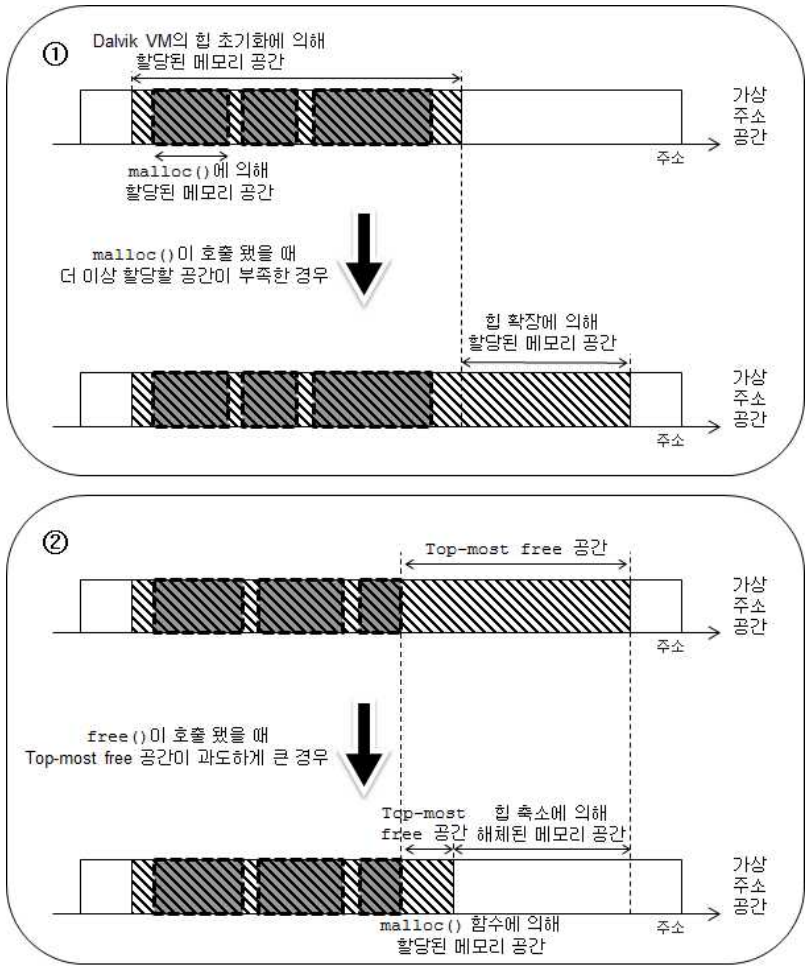

그림 10. 달빅 가상 머신이 관리하는 힙의 확장과 축소 과정.

3.3 리눅스에서 페이지 부재

3.3.1 개괄

페이지 부재는 요구 페이징을 위해 사용되는 기법이다. 일반적으로 요구 페이징은 프로세스가 디스크 값 참조 시에만 그 디스크 값을 물리 메

모리로 프레임 단위만큼 적재하는 가상 메모리 시스템 기법이다. 요구 페이징은 프로세스 구동을 위한 최소한의 크기만을 물리 메모리에 적재하기 때문에 요구 페이징 사용 전보다 더 많은 프로세스를 물리 메모리에 적재하여 실행할 수 있다.

리눅스 구현상에서 페이지 부재는 디스크 작업뿐만 아니라 동적 힙에 대한 작업도 처리한다. 디스크 작업을 위한 페이지 부재는 부재 페이지에 대하여 프레임을 할당하고 그 페이지와 프레임을 사상한 후 그 프레임에 디스크에서 읽은 값을 적재한다. 반면 동적 힙에 대한 페이지 부재는 부재 페이지에 대하여 프레임을 할당하고 그 페이지와 프레임을 사상하는 작업만을 수행한다.

3.3.2 페이지 부재 처리기를 통한 프레임 할당

리눅스에서 프로세스가 접근한 페이지에 대하여 페이지 부재가 발생하면 인터럽트에 의해 do_page_fault() 함수가 호출된다. 이 함수는 페이지 부재 종류에 따라 다른 작업을 수행한다. 페이지 부재 처리기는 힙을 위한 페이지 부재 작업을 처리하기 위해 do_anonymous_page() 함수를 호출한다. 이는 anonymous 메모리 영역에 대한 처리를 의미한다. 이 함수는 페이지 부재가 발생한 페이지에 사상할 프레임을 하나 할당한다.

페이지 부재 처리기는 프레임 할당을 하기 위해 alloc_pages() 함수를 호출한다. 이 함수의 인자 값들 중에 요청하는 프레임의 개수를 지정할 수 있는데, 페이지 부재에서 호출되는 함수들은 단 하나의 프레임만을 요청한다. 이 함수는 __alloc_pages_nodemask() 함수를 호출한다. 프레임을 할당하기 위한 시작 함수인 이 함수는 리눅스 커널의 버디 할당자(Buddy Allocator)에게 프레임을 요청한다.

제 4 장 관련 연구

본 장에서는 2장에서 실험 관찰을 통해 도출된 과도한 페이지 부재 발생을 줄일 수 있는 선사상 관련 연구들을 설명하고 한계점을 분석한다.

4.1 선페이징 개괄

선페이징은 아직 참조되지 않은 다수의 프레임을 물리메모리로 할당하는 기법이다. 선페이징은 여러 상황에 사용된다. 그 중 선페이징은 페이지 부재 시 사용되는 선페이징이다.

선페이징에서 결정해야 할 중요 요소로 선페이징 개수, 선페이징 주소가 있다. 선페이징 개수는 선페이징 할 페이지의 개수를 결정하며 선페이징 주소는 선페이징 할 페이지의 주소를 결정한다.

선페이징의 종류로는 선반입과 선사상이 있다. 선반입은 디스크 입출력을 수반하는 페이지 부재를 위한 선페이징이며 선사상은 디스크 입출력을 수반하지 않는 가능한 페이지 부재를 위한 선페이징이다.

4.2 관련 연구

관련 연구는 선반입에 관련된 연구와 선사상에 관련된 연구로 나눌 수 있다. 선반입에 관련된 연구로는 OBL(One Block Lookahead) 선반입[5], 페이지 클러스터 기반 선반입[6] 그리고 적응적 선반입[7]이 있다. 선사상에 관련된 연구로는 메모리 접근 패턴 로그 기반 선사상[8, 9]이 있다.

4.2.1 선반입 관련 연구

OBL 선반입은 페이지 부재 시 부재 페이지 뿐만 아니라 그 다음 페이지까지 적재하는 기법이다. 따라서 선페이징 개수는 1이며 선페이징 주소은 부재 페이지의 다음 페이지이다.

페이지 클러스터 기반 선반입은 페이지 클러스터 중 하나의 페이지에 페이지 부재가 발생하면 부재 페이지 외에도 페이지 클러스터의 나머지 페이지들을 적재하는 것이다. 페이지 클러스터는 디스크에 연속적으로 저장되어있고 가상 주소 공간에서 인접한 페이지들이다. 이 기법은 디스크 입출력 작업의 공간 구역성(Spatial Locality) 특성을 활용한 기법이다. 따라서 선페이징 개수는 페이지 클러스터의 크기에서 1개를 뺀 값이며 선페이징 주소는 페이지 클러스터의 위치이다.

적응적 선반입은 이전에 선반입한 프레임의 참조된 횟수를 보고 선반입 할 선페이징 개수를 결정하는 기법이다. 주기적으로 이전에 선반입한 프레임의 참조된 횟수에 비례하여 선페이징 개수를 결정한다. 선페이징 주소는 부재 페이지의 다음 페이지들이다.

4.2.2 선사상 관련 연구

메모리 접근 패턴 로그 기반 선사상은 메모리 접근 패턴 로그 값을 통해 미래에 접근할 페이지를 선사상하는 기법이다. 메모리 접근 패턴 로그는 한 프로세스의 메모리 접근 기록을 갖는 로그 메시지이며 정적으로 생성된다. 선페이징 개수는 메모리 접근 패턴 로그에 따라 달라지며 선페이징 주소는 로그 메시지의 메모리 접근 주소 값들로 결정된다.

4.3 한계점

기존 연구들은 안드로이드에 적용되기에 두 가지 한계점들이 있다. 선반입에 관련된 연구들은 주로 순차 메모리 접근을 하는 디스크 작업을 위한 해결책들이다. 본 논문에서 제시한 문제 상황은 임의 메모리 접근을 요구하는 동적 힙에 대한 메모리 접근이므로 기존의 선반입 연구들이 적용되기 어렵다. 기존의 힙을 위한 선사상 연구들은 정적으로 로그 값을 얻어야하기 때문에 응용마다 로그 값을 추출하는 작업이 필요하다. 이는 다양한 응용들을 갖는 안드로이드 플랫폼에서 적용되기에 어려움이 있다.

제 5 장 문제 정의

본 장에서는 2장에서 과도한 페이지 부재 발생 횟수에 대한 문제를 정의한다.

5.1 개괄

본 논문은 선사상을 통해 페이지 부재 처리기의 총 수행 시간을 줄이고자 한다. 대상 시스템 모델을 살펴보고 그 모델에서 총 페이지 부재 비용을 정의한다.

5.2 대상 시스템 모델

그림11은 대상 시스템 모델을 나타낸다. 하나의 물리 메모리는 여러 개의 태스크들 $\tau_1, \tau_1, \ldots, \tau_N$에 의해 사용된다. 하나의 태스크는 달빅 가상 머신을 수행 중이며 주어진 벤치마크에 대하여 여러 개의 페이지 부재 $pf_1, pf_2, \ldots, pf_{K_i}$를 가진다. 그리고 주어진 벤치마크에 대하여 페이지 부재 비용 c_i를 갖는다.

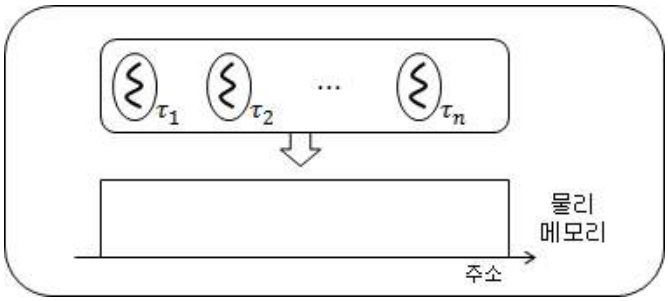

그림 11. 대상 시스템 모델.

여기서 N은 대상 시스템의 태스크들의 개수이며 K_i는 주어진 시나리오에 대하여 태스크 τ_i에서 발생한 페이지 부재의 총 개수를 나타낸다.

5.3 문제 정의

본 절은 페이지 부재 처리기의 수행 시간을 태스크 별로 수식화 하고 이를 토대로 총 수행 시간을 정의한다.

5.3.1 태스크 τ_i의 페이지 부재 처리기의 수행 시간

주어진 벤치마크에 대하여 i번째 태스크 τ_i에 대한 페이지 부재 처리기의 수행 시간 c_i는 페이지 부재에 대한 함수로 정의된다.

$$c_i = \sum_{j=1}^{K_i} TC(pf_j)$$

여기서 $TC(x)$ 함수는 그 페이지 타입에 따른 페이지 부재 x의 수행 시간을 계산하는 함수이며 K_i는 주어진 벤치마크에 대하여 i번째 태스크 τ_i에서 발생한 총 페이지 부재 횟수이다.

5.3.2 페이지 부재 처리기의 총 수행 시간

모든 태스크에 대한 페이지 부재에 의한 총 비용을 C라 하면 그 비용은 c_i들의 합으로 정의된다.

$$C = \sum_{i=1}^{N} c_i$$

여기서 N는 대상 시스템에서 태스크의 개수를 나타낸다.

제 6 장 프레임워크 지원을 통한 선사상 기법

본 장에서는 4장에서 언급한 기존 연구들의 한계점을 극복하면서 5장에서 정의한 문제를 해결할 수 있는 메커니즘들을 구체적으로 설명한다.

6.1 선사상 기법

그림 12는 제안된 선사상 개관을 나타낸다. 달빅 가상 머신은 모든 자바 객체 할당을 위하여 반드시 바이오닉 C 라이브러리의 동적 메모리 할당 함수를 사용한다. 제안된 기법은 이러한 사실에 착안하여 선사상 시 커널에 전달할 힌트를 생성하기 위해 대상 시스템의 바이오닉 C 라이브러리가 확장된다. 그리고 달빅 가상 머신의 자바 객체 할당자와 바이오닉 C 라이브러리가 선사상 힌트를 각각의 하위 계층에 전달할 수 있도록 확장된다.

그림 12. 제안된 선사상 기법 개관.

제안된 기법에서 힌트는 선사상 시점 그리고 선사상 할 페이지의 개수 및 주소 값으로 구성된다. 이 힌트들은 런타임에 힙이 확장되는 메커니즘에 따라 결정된다. 선사상 시점은 자바 객체가 할당되는 힙이 확장되는 시점이다. 선사상할 페이지의 개수 및 주소 값은 확장된 힙 영역에 해당하는 페이지들이다.

그림 13은 제안된 선사상 기법의 구성 요소들을 나타낸다. 제안된 선사상 기법은 대상 시스템의 달빅 가상 머신, 라이브러리 그리고 커널 계층에 각각 식별자 생성자 모듈, 힌트 생성자 모듈 그리고 선사상 모듈로 구성된다. 식별자 생성자 모듈은 바이오닉 C 라이브러리에게 동적 메모리 할당 요청의 출처가 웹 브라우저의 자바 객체 할당자 임을 알려준다. 힌트 생성자 모듈은 힙이 확장될 때 그 확장된 영역의 기준 주소와 길이를 포함하는 힌트를 메모리 할당자의 선사상 모듈에게 전달한다. 선사상 모듈은 전달 받은 힌트를 토대로 얻은 대상 페이지들을 물리 메모리로 선사상한다.

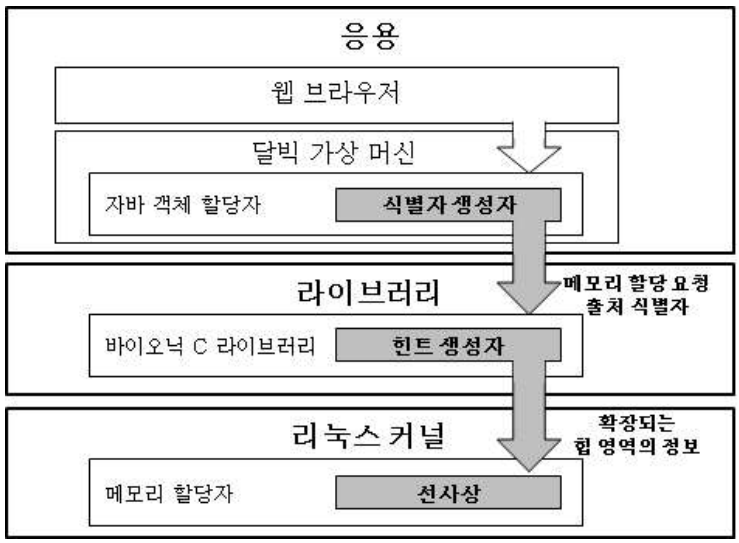

그림 13. 제안된 선사상 기법의 구성 요소들.

6.2 식별자 생성자 모듈

식별자 생성자 모듈은 바이오닉 C 라이브러리에게 동적 메모리 할당 요청의 출처가 웹 브라우저의 자바 객체 할당자 임을 알려준다. 이 모듈은 달빅 가상 머신의 자바 객체 할당자 내에 존재하며 동적 메모리 할당을 할 때 바이오닉 C 라이브러리에게 메모리 할당 요청 출처 식별자를 전달한다. 이 식별자는 동적 힙에 할당하는 경우와 그렇지 않은 경우를 식별한다.

6.3 힌트 생성자 모듈

힌트 생성자 모듈은 힙이 확장될 때 그 확장된 영역의 기준 주소와 길이를 포함하는 힌트를 메모리 할당자의 선사상 모듈에게 전달한다. 이 모듈은 바이오닉 C 라이브러리 내에 존재한다. 이 모듈은 우선 식별자 생성자가 전달한 식별자를 받아서 요청된 동적 메모리 할당이 동적 힙에 할당하는 경우인지 판단한다. 동적 힙에 할당하는 경우에 한해서 이 모듈은 동적 힙이 확장될 때 확장된 힙 영역에 해당하는 연속적인 페이지들에 대한 정보를 선사상 모듈에게 전달한다. 이 정보는 확장된 힙의 첫 번째 페이지의 주소와 총 페이지의 개수를 포함하며 추가로 현재 선사상을 필요로 하는 태스크 정보를 포함한다.

6.4 선사상 모듈

선사상 모듈은 전달 받은 힌트를 토대로 얻은 대상 페이지들을 물리 메모리로 선사상한다. 이 모듈은 리눅스 커널에 존재한다. 이 모듈은 우선 힌트 생성자로부터 건네받은 힌트를 토대로 선사상할 태스크를 구별하고 선사상이 필요할 경우 그 힌트에서 선사상할 페이지들 정보를 토대로 선사상을 시도한다.

제 7 장 구현 및 실험 평가

이번 장에서는 제안된 선사상을 구현하고 그 유효성을 평가한다. 이를 위해서 6장에서 소개한 선사상 기법을 안드로이드에 적용하고 대상 시스템에 대상 시나리오를 수행하여 성능 향상 정도를 측정한다. 마지막으로 결과에 대해 토의 하고 앞으로의 개선 방향에 대해 논의한다.

7.1 구현

그림 14는 제안된 선사상의 구현을 나타낸다. 제안된 선사상을 구현하기 위해 우리는 달빅 가상 머신의 자바 객체 할당자와 라이브러리의 바이오닉 C 라이브러리에 각각 힌트 생성을 위한 식별자 생성자 모듈과 힌트 생성자 모듈을 추가하였고 각각 하위 계층으로 힌트를 전달하기 위한 함수를 추가하였다. 선사상 모듈은 버전 2.5.46 이상의 리눅스 커널에서 제공된다.

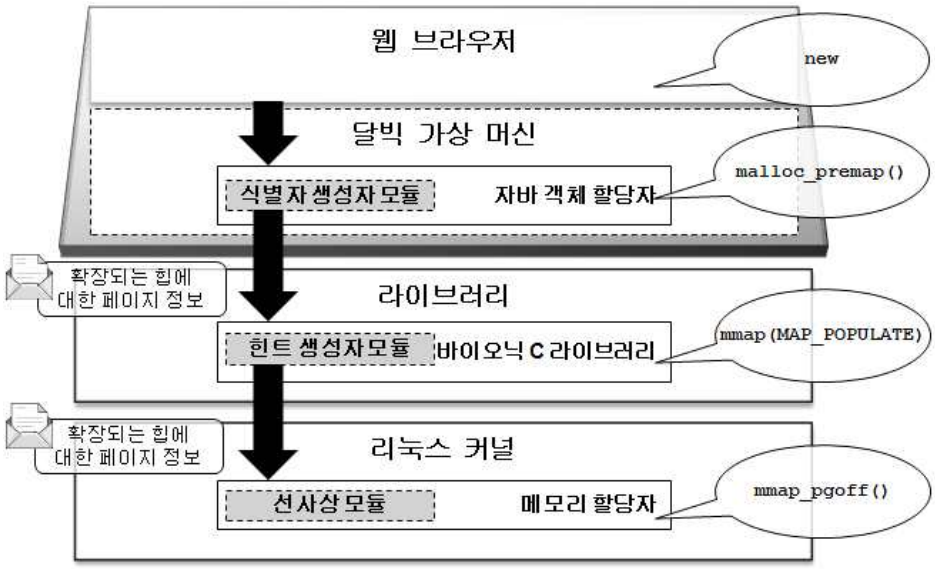

그림 14. 제안된 선사상 기법 구현.

식별자 생성자 모듈은 호출된 malloc() 함수가 달빅 가상 머신이 관리하는 동적 힙에 할당하는지 파악한다. 이를 바이오닉 C 라이브러리에 전달하기 위해 식별자 생성자 모듈은 그 라이브러리에 추가된 malloc_premap() 함수를 호출한다. 바이오닉 C 라이브러리에 구현된 힌트 생성자 모듈은 malloc_premap() 함수가 호출됐을 때 힙 영역의 확장 시기와 확장된 힙 영역에 해당하는 page들을 파악한다. 힙 영역이 확장된 시기에 그 페이지들을 커널에게 전달하기 위해 힌트 생성자 모듈은 그 페이지들을 인자로 갖는 mmap() 시스템 콜을 사용한다. 이때 mmap() 시스템 콜의 인자로 MAP_POPULATE 플래그가 넘겨지는데 이는 그 시스템 콜의 인자로 넘어간 페이지들을 미리 물리 메모리에 미리 사상하도록 선사상 모듈에게 요청한다. mmap_pgoff() 함수는 요청받은 페이지들을 미리 프레임들에 사상한다.

7.2 실험 환경

4.1절의 실험 구성과 동일하되 제안된 기법 적용 이전과 이후에 대한 실험값을 측정한다.

7.3 실험 결과

그림 15는 대상 시나리오를 수행할 때 제안하는 기법을 대상 시스템에 적용하기 이전과 이후에 얻은 실험 결과를 나타낸다. 안드로이드 기본 브라우저와 Firefox V15 브라우저의 페이지 부재 횟수는 각각 14.65%, 16.74% 감소하였으며 응답 시간은 각각 2.83%, 3.25% 감소하였다. 한편 제안된 기법의 메모리 오버헤드는 두 브라우저에서 각각 5.04%와 3.34%에 불과하였다.

웹 브라우저 종류	페이지 부재 발생 횟수 (번)	페이지 부재 처리기의 총 수행 시간 (ms)	물리 메모리 사용량 (KB)	응답 시간 (ms)
기본 브라우저	9,326	279.78	98,004	1238.46
기본 브라우저 (선사상 기법 적용)	7,960	238.8	102,945	1203.35
향상 정도 (%)	14.65%		-5.04% 오버헤드	2.83%
Firefox V15	13,637	409.11	121,641	1943 .28
Firefox V15 (선사상 기법 적용)	11,354	340.62	125,704	1880.12
향상 정도 (%)	16.74%		-3.34% 오버헤드	3.25%

그림 15. 실험 결과.

제 8 장 결론

본 논문은 안드로이드에서 웹 브라우저 사용 시 발생하는 과도한 페이지 부재 수를 줄여 페이지 부재 처리기의 총 수행 시간을 줄이기 위해 선사상 기법을 제안하였다. 이를 위해 안드로이드의 프로세스 모델을 분석하고 안드로이드의 프레임워크 단부터 커널 단까지 메모리 관리 방식을 분석하였다. 이를 토대로 문제가 정의되고 해결책이 제안되었다. 그리고 주어진 벤치마크의 비효율적인 힙 사용에 대해 정량적으로 나타낼 수 있는 페이지 부재 처리기의 총 수행 시간에 대해 정의하였다. 제안된 해결책은 런타임에 확장되는 동적 힙에 대해 선사상을 수행하여 응답 시간을 최대 3.25% 감소 시켰다.

참고 문헌

[1] J. West and M. Mace, "Browsing as the killer app: Explaining the rapid success of Apple's iPhone," in Telecommunications Policy, 2010. [국외 학술지]

[2] 안드로이드 developers http://developer.android.com, 2012. [웹 사이트]

[3] Dave Marshall, Programming in C UNIX System Calls and Subroutines using C, 1999. [국외 학술지]

[4] Rahul Mehta, Configurable JVM Threading, A Dissertation Submitted to The University of Manchester for The Degree of Master of Science in The Faculty of Engineering and Physical Sciences, 2007. [국제 논문지]

[5] M. Joseph. An analysis of paging and program behaviour. Computer Journal, 1970. [국제 논문지]

[6] D. Black, J. Carter, G. Feinberg, R. MacDonald, S. Mangalat, E. Sheinbrood, J. Sciver, and P. Wang. OSF/1 virtual memory improvements. In Proceedings of the USENIX Mac Symposium, 1991. [국제 논문지]

[7] 안우현, 양종철, 그리고 오재원. APC: 가상 메모리 시스템에서 적응적 페이지 선반입 제어 기법, 정보 과학회 논문지, 2010. [국내 논문지]

[8] Seongje Cho, Yookun Cho, 페이지 부재 Behavior and Two Prepaging Schemes, In Proceedings of the 1996 IEEE 15th

Annual International Phoenix Conference on Computers and Communications, 1996. [국제 논문지]

[9] Heejin Ahn, Seongjin Cho, Hyunik Na, and Hwansoo Han Access Pattern Based Stream Buffer Management Scheme for Portable Media Players, IEEE Transactions on Consumer Electronics, 2009 [국제 논문지]

Abstract

Performance of the web browser which is a representative killer application of Android based smartphone is an important factor for user experience and service quality. Since the web browser often runs memory-intensive works, execution time for the memory management subsystem of Linux kernel occupies large portion. Specifically, when the web browser reads a new web page, thousands of page faults occur. To deal with the page faults, the web browser spends almost 20% of total response time. Therefore, we can reduce total response time by reducing the number of page faults. The reason why a lot of page faults occur is the memory allocation method of the demand paging which is mechanism for dynamic memory allocation in Linux kernel. Pre-mapping can reduce the number of page faults, but it is difficult to predict the target pages for the mechanism. This paper proposes the pre-mapping mechanism which predicts the pages with hints from Android runtime and libraries. The experiment results demonstrate that the proposed technique reduces the response time up to 3.25% for a target scenario.

Keywords: Android, Pre-mapping, Dynamic heap, Dalvik VM
Student number: 2010-23251

감사의 글

석사 과정 동안 공학도로서 갖추어야 할 자세와 마음가짐을 많이 가르쳐주신 분들에게 감사의 뜻을 전하고자 합니다. 공학 햇병아리였던 저에게 석사 과정은 결코 순탄하지 않았습니다. 여러분들의 관심, 가르침과 격려 덕분에 제가 무사히 석사 과정을 마칠 수 있었습니다.

우선 저의 지도 교수님이신 홍성수 교수님께 깊이 감사드립니다. 교수님께서는 저에게 공학도가 가져야 할 자세로 자기 주도적 문제 해결 능력을 가르쳐주셨습니다. 교수님께서는 이를 토대로 학부 시절 지식의 소비자를 벗어나 이제는 지식의 생산자로 발돋움해 나가야 함을 저에게 일깨워 주셨습니다. 이는 앞으로 저의 연구자 일생에 가장 중요한 지침이 될 것입니다. 다가올 융합과학기술 대학원 박사 과정 동안 제가 국가 산업 발전에 기여할 수 있는 공학도로 발전해 나가는 모습으로 교수님의 은혜에 보답하겠습니다. 저의 석사 졸업 논문 심사를 맡아주셨고 학문과 연구에 조언을 아끼지 않으셨던 문수묵 교수님과 김태환 교수님께도 깊은 감사를 드립니다.

연구실 구성원들께도 감사드립니다. 항상 연구실에 가장 먼저 출근하시고 매사에 열정적인 혜준씨, 제가 공학도로서 가장 닮고 싶은 롤 모델인 샤프한 연구실 맏형 종훈이 형, 묵묵히 맡은 업무를 잘 수행하고 똑똑한 대동이 형, 연구실의 구성원들을 아끼고 보살피며 연구에 있어서는 스토리 텔링의 대가이신 승주 형, 연구원의 사회생활과 회사에서의 최근 연구 방향성들을 많이 알려 주신 명선이 형, 앞으로 함께 융대원 생활을 해쳐 나갈 넉살 좋은 원석이 형, 웃는 얼굴이 매력적인 정희 형, 만난 기간이 짧지만 많은 이야기를 해주신 병수 형, 삶의 행복이란 무엇인가 생각해보게 해주시고 스스로도 꿈을 가지고 열정적으로 삶을 주도하는 명

이 형, 저에게 여러모로 도움을 많이 주신 정 많은 제동이 형, 엠티 가서도 한국어 공부를 열심히 하던 자제력 으뜸 알렉세이, 점심때마다 전자레인지로 밥을 데워 301동 1층에 내려가 혼자 식사하던 강철 심장 비제타, 거친 유머 감각 뛰어나며 드라이브 스피드를 즐기고 코딩을 빠르고 신속하게 잘하지만 나보다 고집이 약한 장원이, 카푸치노를 좋아하고 사람들을 만나 이야기하는 걸 좋아하고 자신이 가진 꿈을 이루기 위해 노력하는 용석이 형, 도움을 요청하면 군말 없이 도와주던 승우 형, 따뜻한 말들로 불안해하던 제게 마음의 안정을 주신 종환이 형, 비제타의 큰 힘이 되어준 우마, 연구실에 어려운 일 있으면 가장 먼저 솔선수범하며 내가 짓궂은 장난을 쳐도 잘 받아주는 고집 센 천사 중현이 형, 열정적으로 하루하루 살아가고 거친 드라이브 스피드를 즐기는 쾌활 건전 총각 동진이 형, 연구실에서 맡은 일을 하느라 응급실에 갈 정도로 연구에 열정적이신 두산맨 재훈이 형, 연구실에서 맡은 일을 끝까지 책임지고 해내려고 하는 헌신적인 성원이 형, 공학도로서 발전하기 위해 스스로 열정적으로 많은 시도를 하며 노력하는 순현이 형, 당장 미국 드라마에 나올 법한 유머스러운 안토니, 할리우드 배우 뺨치게 잘생긴 맨체스터 출신 축구 열정맨 리차드, 앞으로 같이 융대원 생활할 한빈이와 윤일권씨 모두 감사드립니다.

장애인들의 정보 접근성 향상을 위한 QoLT (Quality of Life Technology) 과제에서 같이 연구했던 분들도 소중한 인연들입니다. 힘들 때마다 위로의 말을 건네어 저를 격려해 주신 박정훈 책임님, 출장 개발 시 많이 배려해주신 안성주 책임님, 맡은 바 업무에 열정적인 신항식 책임님, 물심양면으로 필요한 물품들을 친절하게 공급해주신 유현국 사원님 덕분에 무사히 과제를 마칠 수 있었습니다.

그리고 마지막으로 저를 항상 믿고 아껴주시며 제가 힘들 때마다 묵묵

히 뒤에서 응원해주시는 김기학 아버지와 이희자 어머니께 감사드립니다. 그리고 이제 군대에서 전역할 날이 멀지 않은 멋쟁이 김영호 남동생에게도 감사합니다.

석사 과정 동안 제게 도움을 주신 많은 분들께 다시 한 번 감사드립니다. 제가 다른 분들에게 많이 얻고 배운 만큼 다른 분들에게 베풀도록 노력하겠습니다. 아무쪼록 모두 건강하시고 하시는 일 잘 풀리시길 응원하겠습니다. 다시 한 번 감사하며 사랑하고 고맙습니다.