ISSN 2383-6318(Print) / ISSN 2383-6326(Online)
KIISE Transactions on Computing Practices, Vol. 21, No. 12, pp. 786-791, 2015. 12
http://dx.doi.org/10.5626/KTCP.2015.21.12.786

Linux 기반에서 빠른 프로그램 실행을 위한 주소 공간 유지 기법

(Address Space Maintaining Scheme for Fast Program Execution in Linux-based Systems)

노 승 준 † 강 동 현 † 김 정 한 † 김 인 혁 † 엄 영 익 ††
(Seung Joon Noh) (Dong Hyun Kang) (Junghan Kim) (Inhyeok Kim) (Young Ik Eom)

요 약 최근 사물에 네트워크 기능을 추가하여 인터넷을 통해 자료를 교환하는 사물 인터넷(IoT, Internet of Things) 환경의 개발이 활발히 진행되고 있다. 이에 따라 기존의 컴퓨팅 환경이 데스크톱이나 모바일로부터 다양한 디바이스의 컴퓨팅 환경으로 확장되고 있다. 이러한 환경에서 프로그램의 응답성은 사용자 경험(UX: User Experience) 측면에서 중요하기 때문에 디바이스에서의 응답성을 위한 빠른 프로그램 실행 기술이 주목 받고 있다. 본 논문에서는 안드로이드 모바일 환경에서 어플리케이션의 빠른 실행을 위한 Zygote 기술을 분석하고, 이를 바탕으로 범용적인 리눅스 환경에서 프로그램의 빠른 실행을 위한 주소 공간 유지 기법을 제안한다. 본 기법은 리눅스에서 사용하고 있는 COW(Copy On Write) 정책을 활용하고 안드로이드의 Zygote 기술을 응용한 기법이다. 제안한 기법을 평가하기 위해 리눅스 상에서 실험하여 성능을 측정한 결과, 일반적인 프로그램과 비교하여 최대 99%의 실행 시간이 단축됨을 확인하였다.

키워드: 리눅스, Android Zygote 기술, Copy on Write 정책, 주소 공간 유지 기법

Abstract The environment of Internet of Things (IoT) wherein various devices are connected through the Internet with value-added network functions, is currently a subject of active study. Accordingly, the existing computing environment based on desktop or mobile systems is being expanded into a computing environment of more diverse devices. Because the response of program launching is important in terms of User Experience (UX) in IoT environments, the technology for guaranteeing rapid response of program launching in IoT devices is getting the focus of much current research. In this paper we analyze the Zygote technique, which is being used for faster program execution in Android systems, and, based on our results, we propose an address space maintaining scheme for the rapid launching of programs for use in Linux-based systems. Our scheme utilizes the Copy on Write (CoW) technique in Linux systems as well as the Zygote technique of Android systems. In order to evaluate the proposed scheme, we implemented our scheme on Linux systems and performed several experiments. The experimental results show that the proposed scheme shortens the launching time up to 99%, compared to the existing technique.

Keywords: Linux, Android Zygote technique, Copy on Write technique, address space maintaining technique

· 본 연구는 미래창조과학부 및 정보통신기술연구진흥센터의 정보통신·방송 연구개발사업의 일환으로 수행하였음. [R0126-15-1065, (SW 스타랩) 중대형 디스플레이 기반 동시 다중 사용자 지원 UX 플랫폼 SW 개발]
· 이 논문은 2015 한국컴퓨터종합학술대회에서 'Linux 기반에서 빠른 프로그램 실행을 위한 Zygote 기법을 응용한 주소 공간 유지 기술'의 제목으로 발표된 논문을 확장한 것임

논문접수 : 2015년 9월 8일
(Received 8 September 2015)
논문수정 : 2015년 9월 23일
(Revised 23 September 2015)
심사완료 : 2015년 10월 12일
(Accepted 12 October 2015)

† 학생회원 : 성균관대학교 정보통신대학
p38l@skku.edu
kkangsu@skku.edu
junghan@skku.edu
kkojiband@skku.edu
†† 종신회원 : 성균관대학교 정보통신대학 교수(Sungkyunkwan Univ.)
yieom@skku.edu
(Corresponding author임)

1. 서 론

폼 팩터(Form Factor)란 컴퓨팅 환경에서 하드웨어의 구조화된 형태를 의미한다. 예를 들어, 폼 팩터에는 모바일 및 데스크톱과 스마트 센서(Smart Sensor), 스마트 암 밴드(Smart Armband) 등의 소형 기기와 스마트 카(Smart Car), 스마트 테이블(Smart Table) 등의 대형 기기 등이 있다.

IoT의 등장으로 각종 다양한 임베디드 시스템들이 인터넷으로 연결되어 상호작용을 할 수 있게 되었다. 이로써 기존의 모바일과 데스크톱 환경의 대표적인 폼 팩터로부터 스마트 와치(Smart Watch), 스마트 월(Smart Wall) 등의 다양한 폼 팩터로 확장되고 있는 추세이다. IoT로 연결된 다양한 기기들은 주로 사용자와 상호작용을 하기 때문에 사용자 경험을 위한 프로그램 실행의 응답성이 보장되어야 한다[1]. 본 논문에서는 이러한 빠른 프로그램의 실행을 위한 주소 공간 유지 기법을 제안한다. 빠른 프로그램의 실행을 위한 주소 공간 유지 기법은 사용자에게 새로운 프로그램이 시작될 때, 기존 프로그램의 실행 속도 보다 빠른 프로그램의 실행을 통해 향상된 응답 속도를 제공하기 위한 기법이다.

안드로이드의 경우, Dalvik 가상 머신 기반의 모바일 환경에서 빠른 어플리케이션의 실행을 위하여 Zygote 기술을 개발했다[2]. 그러나 Zygote 기술은 Dalvik 가상 머신 환경에서만 작동하기 때문에 일반 리눅스 환경에 적용할 수 없는 한계점이 있다. 본 논문에서는 Zygote 기술과 리눅스의 COW 정책을 응용하여 범용적인 리눅스 환경에서 보다 빠르게 프로그램을 실행하기 위한 주소 공간 유지 기법을 제안한다.

본 논문의 구성은 다음과 같다. 1장의 서론에 이어 2장에서는 Zygote 기술을 살펴보고 한계에 대해 파악한다. 3장에서는 본 논문에서 제안한 주소 공간 유지 기법을 자세히 설명하고 리눅스 환경에서 구현한 방법을 설명한다. 4장에서는 여러 워크로드에서 벤치마크 툴을 이용하여 기존의 프로그램 실행 방법과 제안한 주소 유지 기법의 성능 차이를 비교 및 분석한다. 마지막으로 5장에서는 본 논문의 결론과 향후 연구에 대하여 도출함으로 본 논문을 마무리한다.

2. 관련 연구

안드로이드의 Zygote는 어플리케이션을 실행하기 위한 데몬 프로세스(Daemon Process)이다. Zygote 프로세스는 안드로이드 시스템이 부팅된 후에 실행되어 이후 시스템에서 실행되는 모든 어플리케이션 프로세스의 실행에 관여한다. Zygote 프로세스가 실행되면 다음과 같은 작업들을 수행한다. 먼저, 새로운 Dalvik 가상 메모리를 할당하여 소켓 통신을 위한 모든 자바 클래스와 이후에 실행될 어플리케이션을 위한 자원들을 로드 해 놓는다. 이후 사용자가 어플리케이션을 실행하면, 액티비티 매니저(Activity Manager)에게 통신 소켓을 통해 실행 요청이 전달되고, 액티비티 매니저는 Zygote 프로세스가 fork 함수를 호출하게 한다. Zygote에서 fork 함수를 호출하여 Zygote의 자식 프로세스가 생기면 자식 프로세스의 Dalvik 가상 메모리에 해당 어플리케이션을 로드 한다[3]. 리눅스의 COW 정책에 의해 실제 메모리의 복사는 바로 일어나지 않으며 Zygote 프로세스는 자식 프로세스와 주소 공간(Address Space)를 공유하게 된다. 이후 Zygote 프로세스와 공유 중이던 자식 프로세스의 주소 공간에 수정에 대한 동작이 일어날 때에만 커널이 실제 메모리를 할당하여 수정 사항을 적용한다. Zygote프로세스의 주소 영역에는 쓰기 불가(Unwritable) 라이브러리를 로드 해 놓기 때문에 이후에 여러 어플리케이션이 실행되어도 COW 정책에 의해 같은 주소 공간에 있는 라이브러리와 자원을 공유한다. Zygote 프로세스는 위와 같은 방식으로 라이브러리를 COW 정책을 통해 주소 공간을 공유함으로써 어플리케이션이 실행될 때마다 라이브러리를 새로운 주소 공간에 로드하는 비용을 줄일 수 있다. 이를 통해 Zygote 기술은 어플리케이션의 실행 시간을 단축시킬 수 있으며 실제 메모리를 새로 할당하지 않기 때문에 메모리의 사용량 또한 줄일 수 있다.

그러나 Zygote기술은 Dalvik 가상 머신 환경에서 메모리에 라이브러리, 자원 및 어플리케이션을 로드하기 때문에 일반 리눅스 환경에 적용할 수 없다[4].

3. 제안 기법

본 논문에서는 Dalvik 가상 머신 환경에서만 작동하는 Zygote 기술을 일반 리눅스 환경에서 적용할 수 있는 주소 공간 유지 기법을 제안한다.

본 논문에서 제안하는 주소 공간 유지 기법은 새로 실행될 자식 프로세스 자체의 코드로 만든 라이브러리와 프로세스에게 필요한 라이브러리들을 부모 프로세스의 주소공간에 로드 해 놓는 기법이다[5]. 하지만 위 기법[5]에서 제시되었던 기법은 자식 프로세스가 만들어진 이후에 자식 프로세스의 주소 공간에 라이브러리를 로드하는 기법이며 본 논문에서 제시한 주소 공간 유지 기법은 자식 프로세스가 만들어지기 전에 부모 프로세스의 주소 공간에 라이브러리를 로드하는 기법이다. 즉, 본 논문에서 제안하는 주소 공간 유지 기법은 자식 프로세스가 실행된 이후에는 자식 프로세스의 주소 공간에 라이브러리를 로드하는 작업 없이 자식 프로세스를 바로 실행하는 기법이다. 이로써 Zygote 기술에서 Zygote

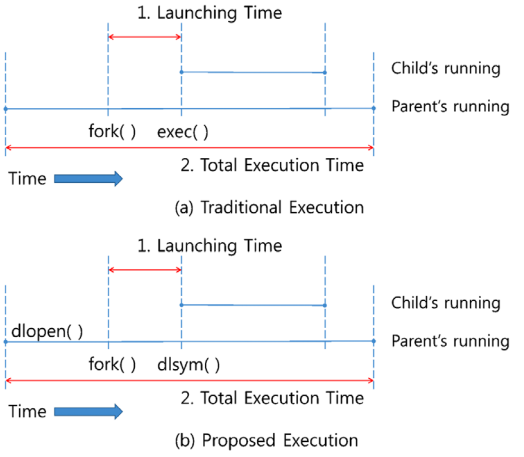

그림 1 프로세스 실행 방식 및 실행 시간 구분

Fig. 1 Process Execution Method and Time Classification

의 주소 공간에 어플리케이션의 라이브러리와 자원들을 로드해 놓는 기술을 일반 리눅스 환경에 적용할 수 있다.

그림 1은 프로세스에서 다른 하나의 프로세스를 실행하기 위한 기존의 방식과 본 논문에서 제안하는 주소 공간 유지 기법의 다이어그램을 보여준다. 기존 방식은 그림 1의 (a)처럼 부모 프로세스에서 fork 함수를 호출 후, exec 함수를 통해 새로운 자식 프로세스를 실행하는 방식이다. 그림 1의 (b)는 본 논문에서 제안하는 주소 공간 유지 기법을 구현하기 위하여 수행되는 핵심 과정을 다이어그램으로 나타낸 것이다. 주소 공간 유지 기법은 다음과 같은 과정을 수행한다. 첫째, 실행될 자식 프로세스의 함수들을 라이브러리로 생성한다. 둘째, 부모 프로세스에서 fork 함수를 호출하기 전에 dlopen 함수를 호출하여 부모 프로세스의 주소 공간에 첫 번째 단계에서 생성한 라이브러리를 로드 한다. 마지막으로, fork 함수를 호출한 후 dlsym 함수를 통하여 라이브러리화 된 자식 프로세스를 실행한다.

자식 프로세스에서 라이브러리가 실행될 때, 자식 프로세스의 주소 공간에 쓰기 작업이 발생하게 되면 리눅스의 COW 정책에 의해 부모 프로세스와 공유되던 메모리 구역에서 다른 새로운 메모리 구역으로 복사 및 쓰기 작업이 발생한다. 복사 및 쓰기 작업으로 인해 주소 공간 유지 기법의 이점이 사라지기 때문에 쓰기 작업으로 인한 새로운 메모리 구역으로의 복사를 최소화하는 것이 중요하다.

4. 실험 환경

본 실험에서는 주소 공간 유지 기법의 빠른 프로그램 실행에 대한 평가를 하기 위해 다음과 같이 총 다섯 개의 워크로드와 두 개의 실험을 구성하였다.

먼저 자식 프로세스가 될 네 개의 워크로드의 경우 다음의 두 가지의 기준으로 구분할 수 있다. 첫 번째 기준은 CPU bounded한 수학적 연산을 위주로 하는 워크로드와 I/O 요청을 하는 워크로드로 구성한다. 또한 각각의 기준에 의해 구분된 워크로드에 대해서 적용된 두 번째의 기준은 추가적인 라이브러리를 사용한 여부에 따라 구분하여 총 네 개의 워크로드를 구현하였다. 마지막으로 dlopen 함수와 exec 함수의 성능차이를 분석하기 위해 Null function으로 이루어진 워크로드를 구현하였다[6].

첫 번째로 CPU bounded인 수학적 계산이 많은 두 개의 워크로드들을 구성하였다. 하나의 워크로드는 RSA 라이브러리를 이용한 2048bit 암호화 프로그램이다. 랜덤으로 키를 생성하고 문자열을 암호화 및 복호화를 다수 반복하는 CPU bounded한 워크로드를 구성하였다[7]. 다른 워크로드는 라이브러리를 사용하지 않는 프로그램으로 N Queens 문제의 해를 백 트래킹(Back Tracking) 알고리즘을 이용하여 찾는 워크로드를 구성하였다[8]. 두 번째로 I/O 요청이 있는 두 개의 워크로드를 구성하였다. 하나의 워크로드는 ZIP 라이브러리를 이용한 파일 압축 및 해제 하는 프로그램이다. 파일을 다수 압축 및 압축 해제를 반복하여 I/O 요청을 하는 워크로드를 구성하였다. 다른 워크로드는 ZIP 라이브러리 보다 작은 크기의 라이브러리를 사용하는 프로그램이다. TCP/IP 통신을 위한 소켓 라이브러리를 이용하여 클라이언트로부터 서버로 파일을 보내는 작업을 수행한다. 마지막으로, Null function으로 이루어진 워크로드의 경우 라이브러리가 추가되어 있지 않으며 내부에서 다른 작업을 수행하지 않는다.

두 개의 실험의 경우 실험1은 그림 1의 (a)와 같이 기존 방식의 프로세스 실행방식으로 구성하였다. 실험2는 그림 1의 (b)와 같이 본 논문에서 제안한 주소 공간 유지 기법을 구현한 방법으로 구성하였다.

실험1의 경우 부모 프로세스에서 fork 함수를 호출한 후 exec 함수를 호출하여 자식 프로세스에서 워크로드를 실행하는 코드로 작성하였다. 실험2의 경우 먼저 워크로드를 라이브러리로 만든다. 이후 부모 프로세스에서 dlopen 함수를 호출하여 라이브러리로 만든 워크로드를 메모리에 로드 하고 난 뒤에 fork 함수를 호출하여 만들어진 자식 프로세스에서 dlsym 함수를 통해 워크로드를 실행하는 코드로 구현하였다.

위 두 실험을 통해 주소 공간 유지 기법이 보다 빠른 프로그램의 실행을 보장하는지 검증했다. 실험 결과를 얻기 위해 각각의 실험에 대해 다섯 개의 워크로드를 자식 프로세스에서 실행하여 Linux Perf Tools를 사용

해 실행해서 종료하는 시점까지의 Page fault 수, Instruction 수와 총 실행 시간을 비교하였다. 또한, 다음과 같은 방법으로 실행 시간을 측정하였다. 그림 1은 실험1과 실험2의 실행 시간(1. Launching time)과 총 실행 시간(2. Total execution time)을 구분하기 위한 다이어그램이다. 실행 시간을 기록하기 위해 gettimeofday 함수를 부모 프로세스에서의 fork 함수 실행 직전과 자식 프로세스가 실행된 직후에 호출하여 타임 스탬프(Time Stamp) 값의 차이로 측정하였다[9].

5. 성능 평가 및 분석

본 논문에서 제안한 주소 공간 유지 기법을 평가하기 위해 모든 실험은 Intel i7-3770K 3.50GHz CPU, 4GB RAM의 하드웨어 시스템에서 Linux Kernel 3.13.0-62의 커널 환경에서 실험하였다. Linux 벤치마크 도구인 Perf는 Perf Tools 3.13.11버전을 사용해 성능을 측정하였다.

그림 2는 실험1과 실험2의 실행 시간의 평균과 총 실행 시간의 평균을 나타낸 막대 그래프이다. 실행 시간 그래프의 Y 축은 각 워크로드의 실험1의 실행 시간 대비 실험2의 실행 시간의 비율을 퍼센트로 나타냈으며 총 실행 시간 그래프의 Y축의 단위는 초(sec)이다.

본 논문에서 제안하는 주소 공간 유지 기법이 모든 워크로드에서 실행 시간이 실험1 대비 25%~99%의 실

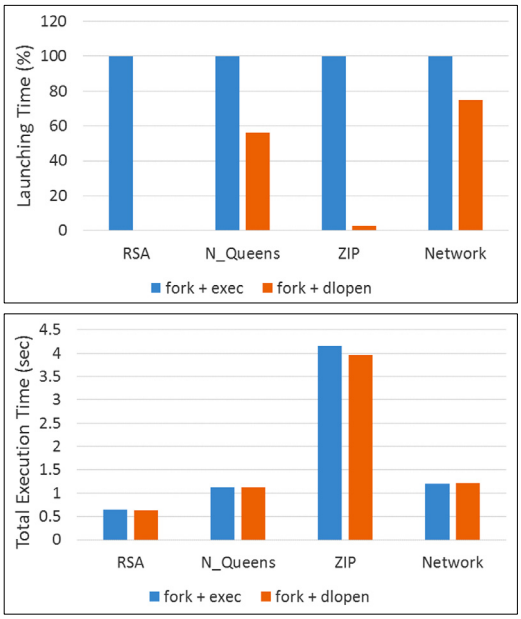

그림 2 실행 시간 및 총 실행 시간 비교

Fig. 2 Comparison of Launching Time and Total ExecutionTime between Experience Groups

행 시간이 단축함을 확인하였다.

RSA와 ZIP 워크로드를 통해 자식 프로세스에서 사용하는 라이브러리의 크기가 커질수록 실행 속도의 성능 향상이 더욱 크게 나타남을 확인하였다. 라이브러리가 큰 워크로드의 경우 주소 공간 유지 기법이 실행 시간이 기존 방식보다 96%~99%의 실행 시간 단축을 확인하였다. 하지만 라이브러리가 작은 워크로드의 경우 주소 공간 유지 기법이 실행 시간이 기존 방식보다 25%~43%의 시간 단축을 나타내어 라이브러리의 크기가 성능 향상에 미치는 영향을 확인하였다. 이는 실험1에서 exec 함수로 자식 프로세스에서 워크로드를 실행하기 위해 라이브러리를 로드하는 시간이 필요하지만 실험2는 자식 프로세스에서 부모 프로세스의 주소 공간에 로드 되어있는 워크로드를 바로 실행하기 때문이다. 그러므로 자식 프로세스에서 실행될 워크로드의 크기가 클수록 실험1에 비해 실험2의 실행 속도의 향상이 크게 나타남을 알 수 있다.

또한, 실험1의 실행 시간은 각 워크로드 마다 차이가 나이가 크게 나타났다. CPU bounded한 워크로드의 경우 라이브러리가 있는 RSA워크로드의 실행 시간이 N Queens워크로드 보다 2497% 더 시간이 측정되었다. 하지만 실험2의 경우에는 RSA워크로드의 실행 시간이 N Queens 워크로드 보다 658% 더 시간이 걸렸다. 이는 dlopen함수가 exec함수보다 추가되는 라이브러리에 대해 더 적은 오버헤드를 가지는 것을 알 수 있다.

하지만 총 실행 시간은 CPU bounded나 I/O요청이 있는 워크로드의 특성이나 라이브러리의 크기와 상관없이 무시 가능할 정도의 실행 시간 단축을 확인하였다. 이는 실행 속도 향상으로 얻어낸 시간의 단축이었으며 워크로드 라이브러리의 크기가 커질수록 총 실행 시간에서의 실행 시간으로 획득한 이점이 차지하는 비율의 크기는 더욱 작아져 기존 방식과 제안한 방식의 총 실행 시간은 유사한 성능을 나타낼 것이다.

그림 3은 실험1과 실험2의 Page fault의 수와 실험1을 기준으로 한 실험2의 Instruction의 수를 비율로써 나타낸 그래프이다. NULL function의 워크로드를 제외한 나머지 워크로드에서 실험2의 Instruction의 수는 실험1과 무시 가능할 정도의 차이가 났다. 하지만, Instruction의 수가 20% 감소한 Null function의 워크로드를 통해 exec 함수가 dlopen 함수 보다 추가적인 작업이 발생하는 것을 확인할 수 있었다. 나머지 워크로드에서 Instruction의 수의 차이가 적은 이유는 총 실행 시간의 경우와 마찬가지로 각 워크로드에서의 CPU나 I/O 작업을 위한 Instruction의 수로 인해 주소 공간 유지 기법으로 얻은 이점의 비중이 낮아지기 때문이다.

Page fault의 수는 실험2가 실험1 보다 15%~25%

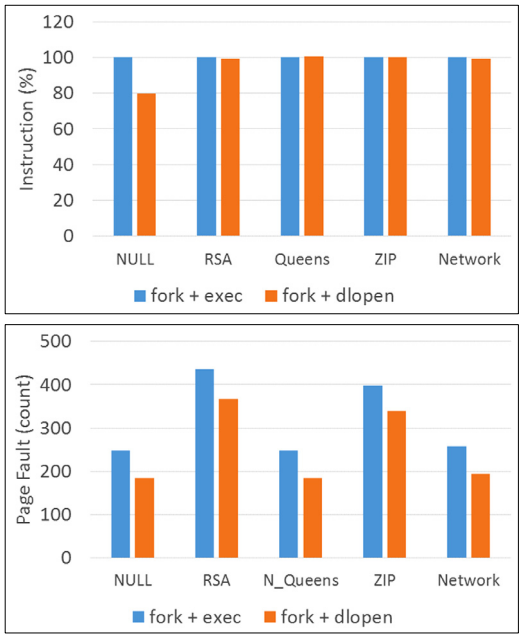

그림 3 실험 성능 비교

Fig. 3 Comparison of Performance between Experience Groups

적게 발생하는 것을 확인할 수 있다. Page fault의 수를 통해 실험1이 실험2보다 Page fault handling을 더 적게 하는 것을 알 수 있다. 이는 본 논문에서 제안한 주소 공간 유지 기법에서 부모 프로세스에 라이브러리를 로드 해 놓음으로써 부모 프로세스의 주소 공간을 자식 프로세스가 공유하여 사용함을 증명한다[10]. 그러므로 본 연구에서 제안한 주소 공간 유지 기법은 자식 프로세스가 실행될 때의 Page fault를 줄임으로써 실행 시간을 단축했음을 알 수 있다.

6. 결론 및 향후 연구

본 논문에서는 안드로이드 Zygote 기술을 응용하여 일반 리눅스 환경에서 프로그램의 실행 시간을 단축하기 위한 주소 공간 유지 기법을 제안하였다. 본 논문에서 제안한 주소 공간 유지 기법은 안드로이드 Zygote 기술과 리눅스의 COW 정책을 응용하여 dlopen 함수를 통해 부모 프로세스의 주소 공간에 실행될 프로그램을 라이브러리 형태로 로드 해 놓음으로써 자식 프로세스가 부모 프로세스의 주소 공간을 공유하도록 하는 기법이다. 이를 통해 exec 함수를 사용하는 일반적인 방법보다 dlopen 함수를 사용하는 주소 공간 유지 기법이 Page fault의 수를 줄임으로써 프로그램의 실행 시간을 최대 99%까지 단축할 수 있음을 실험을 통해 확인하였다. 본 논문에서 제안한 주소 공간 유지 기법을 통해 향후 리눅스 기반의 다양한 폼 팩터에서 기존 방식보다 빠른 사용자 경험을 위한 프로그램 실행의 응답성을 보장할 수 있음을 확인할 수 있었다.

향후에는 본 논문에서 제안한 주소 공간 유지 기법과 실험 결과를 바탕으로 실제 상황에서 리눅스 기반의 폼 팩터에서 빠른 사용자 경험을 이끌어 내기 위한 운영체제 단계에서의 기법에 대해 연구 및 분석해 볼 예정이다.

References

[1] N. Tolia, D. G. Andersen, and M. Satyanarayanan, "Quantifying Interactive User Experience on Thin Clients," *Journal of Computer*, Vol. 39, No. 3, Mar. 2006.

[2] R. West and G. Parmer, "Application-Specific Service Technologies for Commodity Operating Systems in Real-Time Environments," *Journal of the ACM Transactions on Embedded Computing Systems*, Vol. 10, No. 30, Apr. 2011.

[3] Zygote | Anatomy of Android [Online]. Available: http://anatomyofandroid.com/2013/10/15/zygote (Sep, 28, 2015)

[4] L. K. Yan and H. Yin, "DroidScope: Seamlessly Reconstructing the OS and Dalvik Semantic Views for Dynamic Android Malware Analysis," *Proc. of the USENIX Security Symposium 2012*, pp. 569-584, 2012.

[5] C. Jung, D. Woo, K. Kim, and S. Lim, "Performance Characterization of Prelinking and Preloading for Embedded Systems," *Proc. of the 7th ACM & IEEE International Conference on Embedded software* 2012, pp. 213-220, 2007.

[6] J. M. Redondo and F. Ortin, "A Comprehensive Evaluation of Common Python Implementations," *Journal of the Software, IEEE*, Vol. 32, No. 4, Aug. 2014.

[7] Y. Chen and R. Sion, *Costs and Security in Clouds*, pp. 50. Springer, New York, 2014.

[8] N. J. Nilsson, *Principles of Artificial Intelligence*, pp. 55. Springer, New York, 1982.

[9] S. A. Finney, "Real-time Data Collection in Linux: A Case Study," *Journal of the Behavior Research Methods, Instruments, & Computers*, Vol. 33, No. 2, May. 2001.

[10] D. P. Bovet and M. Cesati, *Understanding the Linux Kernel*, 3nd ed., pp. 115. O'Reilly Media, Inc. California, 2005.

노 승 준

2015년 성균관대학교 컴퓨터공학과학사 2015년~현재 성균관대학교 전자전기 컴퓨터공학과 석사과정. 관심분야는 운영체제, 스토리지 시스템, UI/UX

강 동 현

정보과학회 컴퓨팅의 실제 논문지 제 21 권 제 1 호 참조

김 정 한

2008년 세종대학교 컴퓨터소프트웨어 공학과 학사. 2010년 성균관대학교 전자 전기 컴퓨터공학과 석사. 2010년~현재 성균관대학교 전자전기컴퓨터공학과 박사과정. 관심분야는 가상화, 운영 체제

김 인 혁

2006년 성균관대학교 컴퓨터공학과 학사 2010년 성균관대학교 전기전자 컴퓨터공학과 석사. 2010년~현재 성균관대학교 전자전기컴퓨터공학과 박사과정. 관심분야는 시스템 소프트 웨어, 운영체제, 가상화

엄 영 익

정보과학회 컴퓨팅의 실제 논문지 제 21 권 제 1 호 참조