리눅스기반 저전력 ARM 임베디드 장비의 부팅과정 최적화

김종석* · 양진영 · 김대영

*한국과학기술원 전산학과

Optimizing Boot Stage of Linux for Low-power ARM Embedded Devices

Jongseok Kim* · Jinyoung Yang · Daeyoung Kim

*Dept. of Computer Engineering, KAIST

E-mail : {paldad*, jyyang0308, kimd}@.kaist.ac.kr

요 약

전통적으로 임베디드 장비에서는 단순한 펌웨어 수준의 운영체제를 사용해 왔다. 그러나 최근 하드웨어 성능의 향상과 이에 따른 사용자들의 다양한 요구사항을 만족시킬 수 있도록 범용 운영체제인 리눅스를 사용하고 있는 임베디드 장비들이 활성화되고 있다. 리눅스를 운영체제로 사용하는 경우 확장성, 범용성, 이식성 등의 장점이 있지만 리눅스의 이식성을 고려한 설계로 인한 복잡성의 증가로 인한 오버헤드가 존재한다. 대다수의 임베디드 시스템에서 전원이 인가된 후 필요한 기능을 정상적으로 수행하기까지의 시간을 최소화하는 것은 필수적인 요구사항이기 때문에, 이들 특성은 임베디드 장비에서는 불필요할 수 있으며 적절히 재구성 또는 제거되어야 한다. 본 논문에서는 Corelogic 사의 CLM9722 DTK 를 대상으로 하여 알려진 소프트웨어 최적화 기법을 리눅스 부팅 각 단계에 적용, 결과를 측정함과 동시에 하드웨어 종속성으로 인한 범용 프레임워크 내의 최적화 제약사항에 대하여 연구하였다. 결과적으로 장비 전원 인가 후 부팅 시간을 약 33% 정도 단축할 수 있었다.

ABSTRACT

Conventionally embedded devices used simple operating system (OS); however, the number of embedded devices using Linux as OS is increasing to keep up with hardware's performance improvement and customer's various needs. While embedded devices using Linux can take advantage of expandability, generality, portability, Linux's flexibility nature may cause undesirable overheads because of its increased complexity. One such overhead makes boot stage optimization essential in most embedded systems, where many features are redundant and possible to be removed or reconfigured. This paper applies well-known software optimization technique for Linux's boot stage to an CLM9722 DTK, measures the results, and studies about limitation of such techniques from hardware dependancy on the standard framework of Linux. The booting time from power-on until completion were decreased by 33% approximately.

키워드

ARM, Embedded Linux, Fastboot, Boot optimization

Ⅰ. 서 론

최근 임베디드 장비에서 리눅스와 같은 범용 운영체제를 사용하는 장비들이 늘어나고 있으나 이 경우 리눅스의 이식성을 고려한 설계로 복잡성이 증가하여 초기화 지연이 발생할 수 있다. 저전력 센서 노드들로 구성되는 VSN(Visual Sensor Networks)[1] 의 경우 센서의 수명을 늘리기 위해 센서 노드 내 이벤트 감지기를 두고 보통 때는 power off mode로 유지되다가 특정 이벤트를 감지한 경우에만 전원이 공급되어 동작하고 다시 power off mode로 돌아간다. 이런 경우 센서 노드가 동작하기 위한 초기화가 매번 필요하게 된다. 리눅스를 이용하는 데스크탑 또는 서버의 초

기화 시간은 20초에서부터 1분 이상이 소요되며 센서 노드와 같은 임베디드 장비에서는 상대적으로 열악한 자원과 낮은 컴퓨팅 파워로 인해 더 소요될 수 있다. 이는 빠른 응답시간과 실시간 요구사항을 만족시켜야 하는 임베디드 장비에서 큰 제약사항이다.

본 연구에서는 ARM 프로세서와 임베디드 리눅스를 사용하는 스마트 센서 시스템에서 리눅스가 제공하는 표준 개발 프레임워크를 사용하여 전원이 인가된 후 특정 동작을 수행하기까지의 시간을 최소화하는 부트과정 최적화에 대한 연구를 진행하였다.

본 논문의 구성은 다음과 같다. 2장에서는 본 논문과 관련된 기존의 연구를 살펴보고, 3장에서 ARM 임베디드 시스템의 부팅 과정을 설명한다. 4장에서는 리눅스 표준 개발 프레임워크를 사용한 최적화 설계를 제안한다. 그리고 5장에서는 실제 개발 보드를 대상으로 제안된 방법을 적용한 후의 결과를 평가하며 마지막으로 6장에서 결론 및 향후 과제를 기술한다.

II. 관련 연구

임베디드 리눅스 환경에서 전원이 인가된 후 타겟 머신에서 특정 어플리케이션이 동작되기까지의 시간을 줄이는 방식은 크게 스냅샷부트 방식(warm boot)[2]와 부트과정 최적화 방식(cold boot)[3]으로 나눌 수 있다. 스냅샷부트방식의 경우, 리눅스의 Power Management 프레임워크의 도움을 받아 현재 동작하고 있는 시스템의 상태를 스냅샷으로 보존하여 주기억장치 또는 보조기억장치의 특정 영역에 저장, 필요할 때 원 상태로 복구하는 방법을 사용할 수 있다. 그러나 ARM 기반 시스템에서는 각 디바이스 명세의 상이함으로 인해 현실적으로 이들 프레임워크의 도움을 받기 어렵다. 그리고 이러한 스냅샷 복귀 기반 방식의 가장 큰 문제점은 타겟 시스템의 프로세서, 그리고 주기억장치와 보조기억장치 등에 계속 전원이 공급되고 있어야 한다는 것이다. 따라서 스냅샷부트 방식들은 배터리 기반 임베디드 장비에서의 에너지 소모를 비약적으로 줄이기 어려우며 이러한 장비에서는 부트과정 최적화를 사용해야 한다.

III. 임베디드 리눅스의 부트 과정

임베디드 리눅스의 부트과정은 크게 3부분으로써, 부트로더 초기화 과정과 리눅스 커널 초기화 과정, 그리고 Userspace 초기화 과정으로 나누어진다.

보드에 전원을 인가하면 Secure boot 과정에 의해 첫 번째로 SOC 안의 iROM안에 있는 BL0가 실행된다. BL0은 SOC의 boot mode 를 감지한 후 Primary boot media를 선택하여 정해진 위치에 있는 BL1을 iRAM에 올린다. BL1 은 주 메모리를 사용하기 위한 기초적인 초기화를 수행 후 BL2 를 찾아 주 메모리에 올린다. BL2는 흔히 부트로더로 불리며 기초적인 하드웨어의 초기화를 수행하는 역할을 맡게 된다. 그 후 부트로더는 커널이미지를 보조 기억장치로부터 읽어 들여 주 기억장치의 정해진 위치로 위치시킨 후 컨트롤을 이전시킨다.

부트로더가 로드한 커널 이미지는 보통 압축되어 있으며, 처음으로 하는 작업은 커널이미지의 압축을 해제하여 정의된 장소에 위치시키는 작업이다. 압축 해제 후 커널은 자신이 필요한 자료구조들을 초기화하고 하드웨어에서 사용할 필수적인 디바이스들을 등록 후 이들 디바이스와 드라이버들을 초기화 한다. 등록되는 디바이스와 드라이버들은 타겟 보드의 머신 드라이버에 구현되어 있다. 그 후 커널은 인자로 넘어온 파티션 정보와 파일시스템 정보를 사용하여 루트 파일 시스템을 찾아 마운트 한다. 루트 파일 시스템은 리눅스 커널이 인식 가능한 파일 시스템과 디렉토리 구조를 가지고 있어야 하며, 정해진 위치에 init 프로세스를 위한 RC 스크립트를 가지고 있어야 한다.

리눅스 커널은 마지막에 init 프로세스를 생성하여 정해진 위치에 저장되어 있는 RC 스크립트를 실행하게 된다. RC 스크립트에는 타겟 어플리케이션이 실행되기까지의 여러 가지 서비스들의 초기화 스크립트들로 이루어져 있으며, 이들은 스크립트에 정의된 순서로 타겟 어플리케이션이 동작하기 위한 각종 서비스를 초기화한 후 타겟 어플리케이션을 동작시킨다.

IV. 임베디드 리눅스 부트 최적화 설계

본 연구에서 택한 접근법은 임베디드 리눅스의 표준 개발 프레임워크에서 제공하는 인터페이스만을 사용하여 임베디드 장비에 전원이 인가된 후 타겟 어플리케이션이 동작할 때까지의 각 과정을 분류하여 네 단계로 나누고, 각각을 최적화 시켜 소요되는 시간을 최소화 시키는 것이다. 표준 개발 프레임워크에서 제공하는 인터페이스만을 사용할 경우 소스코드의 수정을 최소화할 수 있을 뿐만 아니라 기존에 존재하는 각 기능 사이의 의존성을 해치지 않기 때문에 개발 시간과 유지보수적인 면에서 이득을 얻을 수 있다.

이를 위하여 부트 과정을 분석 후 최적화가 가능한 부분을 분석하여 부트로더 최적화, 커널 임지 로딩 및 압축 해제, 커널 최적화, Userspace

최적화의 4 부분으로 분류하였다. 분류된 부분과 각 부분에서 사용될 수 있는 방법들을 그림 1 에 나타내었다.

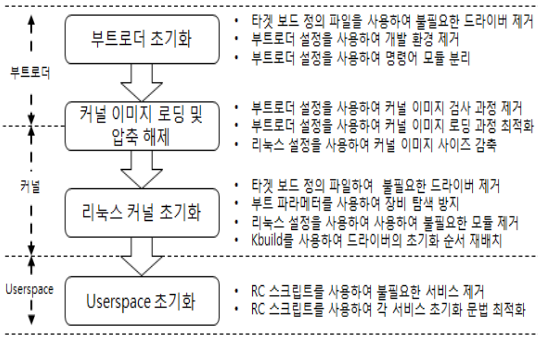

그림 1. 리눅스 개발 프레임워크를 통한 최적화 설계

4.1 부트로더 초기화 단계에서의 최적화

본 연구에서 사용되는 Uboot 는 부트로더 레벨에서의 개발을 위해 디버깅 루틴, 커맨드 라인 인터페이스와 각종 명령어를 제공한다. 또한 개발 환경에서 사용되는 각종 디바이스 탐색 및 초기화 과정 또한 시간을 소모하게 된다. 이들은 하드웨어 제조사에서 제공하는 보드 설정 파일을 수정함으로써 위 환경이 필요 없는 최종 릴리즈 버전에서는 제거하여 부트로더의 초기화 시간을 최적화 할 수 있다.

리눅스 이미지를 로딩하기 위한 필수 기능만을 유지하고 Uboot 에서 소요되는 시간을 감소하기 위해, 개발 환경에 사용되는 키보드 및 LCD 컨트롤러 초기화, 메시지 출력을 위한 리얼 인터페이스 초기화, 네크워크 부트를 위한 이더넷 컨트롤러 초기화 과정을 제거하였다. 그리고 커맨드 라인 인터페이스를 제거하고 업데이트 과정을 생략하여 커널 이미지를 바로 로드하고 부팅하도록 하였다.

4.2 커널 이미지 로딩 및 압축 해제의 최적화

커널 이미지의 로딩 시간과 압축해제 시간은 부트로더와 커널의 설정 두 부분에서 영향을 받는다. 이미지 로딩 시간은 커널 이미지 사이즈와 비례하며 커널 설정에서 커널 이미지의 사이즈를 감소시킬 수 있다. 주된 방법은 하드웨어에서 사용하지 않는 드라이버들과 관련된 코드들과 각종 디버그 심볼들을 제거하는 것이다. 커널 이미지 사이즈를 작게 유지하고 커널 이미지를 불러올 때 커널 이미지가 있는 파티션 전체를 불러오는 대신 정확한 이미지 크기만을 로드하여 보조기억장치에서 주기억장치로 커널 이미지를 로드하는 I/O 동작에 소모되는 시간을 줄일 수 있다.

본 연구에서 제거한 드라이버들은 대표적으로 IDE, SCSI 디바이스와 ext 파일시스템과 같은 블록 디바이스 관련 드라이버들과 같이 개발 보드에서 사용하지 않는 부분들이다. 그리고 cgroup, power management 와 같이 타겟 시스템에서 중요하지 않은 서브시스템과 기능들을 제거하였다.

init 램디스크는 보조기억장치에 들어 있는 루트 디렉토리를 마운트하기 전 필요한 드라이버를 로드하는 역할을 수행한다. 네트워크 부트 등을 사용하는 경우에는 이를 위한 필수적인 서비스가 들어 있는 파일시스템을 주기억장치로부터 빠르게 읽어오기 위해 사용되지만, CLM9722 DTK와 같은 보조기억장치에 루트 디렉토리가 존재하는 경우에는 결과적으로 루트 디렉토리를 두 번 마운트하게 되며, 전체 커널 이미지 사이즈가 증가하게 된다. 따라서 init 램디스크를 제거하였다.

4.3 커널 초기화 과정에서의 최적화

리눅스 커널에서 사용할 수 있는 방법으로는 커널 설정을 사용한 세부 옵션 변경, 커널 부트 파라메터를 사용한 내부 변수들의 초기값 지정, 그리고 Kbuild 시스템을 사용한 각 모듈의 초기화 순서를 재지정하는 방법으로 최적화가 가능하다.

리눅스 커널 부팅 과정에서 가장 많은 시간을 소모하는 과정은 타겟 보드에서 사용하는 각종 디바이스들을 초기화하는 과정에서 발생한다. 리눅스 커널이 생성한 로그 메시지를 분석해보면, 리눅스 부팅 과정에서 80% 이상의 시간이 디바이스를 초기화하는 과정에서 발생하는 것을 알 수 있다. 이 과정에서 소모되는 시간은 각 모듈의 초기화 과정 순서를 재배치하여 상대적으로 오랜 시간이 거리는 디바이스의 초기화 루틴을 더 일찍 실행하거나, 타겟 보드에 존재하지만 타겟 어플리케이션에서 실제로 사용하지 않는 디바이스를 제거함으로써 줄일 수 있다. 주 지연 루틴을 개별적으로 초기화하는 것 또한 부팅 시간 감소에 도움이 될 수 있지만 각 드라이버 코드를 수정해야 하는 작업이 필요하다.

4.4 Userspace 과정에서의 최적화

RC 스크립트에는 타겟 어플리케이션이 실행되기까지의 여러 가지 서비스들의 초기화 스크립트들로 이루어져 있으며, 이들 중 사용하지 않는 서비스들이 존재한다면 제거하여 타겟 어플리케이션이 실행되기까지의 시간을 감소시킬 수 있다. 본 연구에서 제거한 서비스들로는 ssh 서버, udev 데몬 등 타겟 어플리케이션이 동작하는데 관련 없는 서비스들이다. 그리고 네트워크 서비스 등 특정 시점에만 필요할 때에만 필요한 서비스들은 필요할 때에만 동작시킬 수 있도록 하였다.

V. 실험 및 성능 평가

구현 및 실험을 위해 Corelogic 사의 CLM9722 DTK 를 사용하였다. CLM9722 DTK 는 ARM cortex A8, DRAM 256MB, 128M의 NAND Flash 를 사용하며, 주변장치로는 카메라, 터치 기능이 있는 LCD 패널과 스피커, USB, 시리얼 인터페이스 그리고 카메라 동작을 트리거하기 위한 각종 센서 들이 존재한다.

CLM9722 DTK 는 부트로더로 Uboot를 사용하고 있다. 리눅스 커널의 버전은 2.6.35 이며, Corelogic 사에서 제작한 보드 초기화 코드가 포함되어 있다.

부트로더에서 소요된 시간은 Grabserial[4] 유틸리티를 사용하여 측정하였다. 리눅스 커널에서 소요되는 시간을 측정하기 위해 설정에서 PRINTK_TIME 를 사용하여 각 출력 루틴에 타이밍 정보를 포함시켰다. 그리고 각 모듈 및 디바이스들의 초기화 과정에서 소요되는 상대적인 초기화 시간을 측정하기 위해 설정에서 BOOT_TRACER 와 시각화 유틸리티인 bootgraph 를 이용하였다. Userspace에서는 각 데몬과 서비스들이 초기화되는 시간을 측정하기 위해서 Bootchart[5] 유틸리티를 사용하였다.

각 과정에서의 최적화 작업 전과 후의 부팅 시간을 측정하여 비교하였다. CLM9722 DTK 에서 사용하는 버전의 리눅스 커널 인자에 기본적으로 preset-LPJ[6] 와 quiet[7] 가 적용되어 있으므로 이를 적용한 결과를 사용하였다. 표 1 에 보는 바와 같이 최적화되기 전에는 3992 ms 가 소요되었으나 최적화 후 2656 ms 가 소요되어 부팅 시간이 33.4% 감소하였다.

주된 소요시간 감소는 보조기억장치와 주 기억장치의 보조기억장치의 이미지 로딩에 소요되는 시간에서 발생하였다. 커널 이미지 사이즈가 최적화하는 과정에서 전 6.8MB 에서 3.1MB 로 약 46% 감소하였으므로 이미지 로딩 시 상당수의 I/O 동작이 감소한 결과이다.

반면에, 커널 초기화 단계에서는 비약적인 시간 감소 효과를 얻기 어려웠다. 이는 대부분의 시간 소요가 보드에 존재하는 필수 디바이스들의 초기화 과정에서 발생했기 때문에 표준 개발 프레임워크에서 제공하는 방법만으로는 더 이상 시간 감소 효과를 얻기 힘들기 때문이다.

표 1. 최적화 전·후 부팅 시간

단계		최적화 전	최적화 후
부트로더	부트로더 초기화	373 ms	213 ms
	커널 이미지 로딩	647 ms	347 ms
커널	커널 압축 해제	93 ms	73 ms
	커널 초기화	1733 ms	1421 ms
Userspace	서비스 초기화	1146 ms	602 ms
Total		3922 ms	2656 ms

VI. 결 론

본 연구에서는 임베디드 리눅스를 사용하는 ARM 개발 보드를 사용하여 전원 인가 후 타겟 어플리케이션이 실행되기까지의 부팅시간을 단축시키기 위한 여러 기법들을 적용하여 부팅 시간을 약 33% 감소시켰다. 사용된 방법들은 코드에 수정을 최소화하기 위해 리눅스가 제공하는 표준 개발 프레임워크를 사용하여 필수 드라이버들과 서비스들이 초기화될 때까지의 시간 지연을 최소화 하는 방법들이며, 개별 드라이버나 타겟 어플리케이션, 하드웨어 최적화에 대해서는 고려하지 않았다. 그 결과 대부분의 시간 소요가 보드에 존재하는 디바이스들의 초기화와 주기억장치와 보조 기억 장치 사이의 I/O 동작에서 발생하는 것으로 나타났기 때문에 소프트웨어 적인 방법론만으론 비약적인 시간 감소 효과를 얻기 어려웠다. 따라서 향후 각 하드웨어의 영향을 고려한 최적화 기법에 대한 연구를 계속 해 나갈 계획이다.

참고문헌

[1] Y. Charfi, N. Wakamiya, and M. Murata, “Challenging issues in visual sensor networks”, Tech. Rep., Advanced Network Architecture Laboratory, Osaka University, 2007.
[2] Hiroki Kaminaga, “Improving Linux Startup Time Using Software Resume”, In proceedings of the Linux Symposium, 2006
[3] Tim R. Bird, “Method to Improve Bootup Time in Linux”, In proceedings of the Linux Symposium, 2004
[4] Grabserial, http://elinux.org/Grabserial
[5] Bootchart, http://www.bootchart.org
[6] Preset-LPJ, http://elinux.org/Preset_LPJ
[7] Disable console, http://elinux.org/Disable_Console