Journal of The Korea Society of Computer and Information
Vol. 18, No. 3, March 2013
www.ksci.re.kr
http://dx.doi.org/10.9708/jksci.2013.18.3.001

어플리케이션 처리속도 개선을 위한 임베디드 리눅스 경량화 기법

이 태 우*, 조 지 용*, 조 용 환*

A Method of Embedded Linux Light-Weight for Efficient Application Execution

Tae-Woo Lee *, Ji-Yong Cho *, Yong-Hwan Cho*

요 약

본 논문에서는 임베디드 시스템에서 실행하는 어플리케이션의 처리 효율을 개선하기 위하여 임베디드 리눅스 경량화 기법을 제안한다. 임베디드 리눅스 경량화를 위하여 Hibernation 기법을 응용한 부팅 시간 단축, Symbolic Link 기법과 가상주소매핑 기법을 적용한 JFFS2 파일시스템 최적화, 범용성을 보장하는 커널 경량화 등 세 가지 방법을 적용하였다. 이후, 의존성 검사 및 커널 이미지를 타깃 임베디드 키트에 맞게 생성하여 전송한 후 기존 임베디드 리눅스를 사용한 시스템과 본 논문에서 제안하는 경량화 기법이 적용된 임베디드 리눅스를 사용하는 시스템의 성능을 비교해 보았다.

실험결과, 커널 사이즈는 기존대비 9.6% 개선되었고, 부팅 완료시간은 18% 개선되었음을 확인하였다. 그리고 타깃 임베디드 키트에서 동작하는 어플리케이션의 처리 속도 또한 최적의 경우 11%, 최악의 경우 66%의 처리 속도가 개선되었음을 확인할 수 있었다. 이 결과는 본 논문에서 제안하는 경량화 기법을 통해 부팅 속도의 향상 및 기존 커널 대비 많은 리소스 확보가 가능하다는 것을 보여주며, 이렇게 확보된 시스템 자원이 임베디드 시스템에서 동작하는 어플리케이션 처리속도 개선에도 좋은 영향을 준다는 것을 의미한다.

▸ Keywords : 임베디드 리눅스, 경량화, 임베디드 시스템

Abstract

In this paper, we propose a method of embedded linux light-weight to improve efficiency of application running on embedded systems. Three methods including fast booting scheme applying the Hibernation technique, JFFS2 file system optimization applying the Symbolic Link and virtual address mapping, kernel light-weight that guarantees the general purpose was applied. Since then

• 제1저자 : 이태우 • 교신저자 : 조용환
• 투고일 : 2011. 12. 16, 심사일 : 2013. 1. 9, 게재확정일 : 2013. 3. 1.
* 충북대학교 컴퓨터공학과(Dept. of Computer Science, Korea University)
※ 이 논문은 2011년도 충북대학교 학술연구지원사업의 연구비 지원에 의하여 연구되었음
(This paper was supported by the research grant of the Chungbuk National University in 2011)

check the system dependency and generate kernel image according to the target embedded kit. And embedded system performance of existing linux and linux which the method proposed in this paper was compared.

In experimental result, the kernel size was 9.6% improved and the system booting time was 18% improved. And application processing speed on target embedded kit was improved 11% in the best case, 66% in the worst case. This result show that the light-weight method proposed in this paper is guarantee fast booting time and securing resources and it is good for the application processing speed improvement.

▸ Keywords : Embedded Linux, Light-Weihgt, Embedded System

I. 서 론

임베디드 시스템이란 미리 정해진 특정 기능을 수행하기 위해 컴퓨터의 하드웨어와 소프트웨어가 조합된 전자 제어 시스템을 말하며, 필요에 따라 기계의 일부가 포함될 수 있다. 즉, 단순히 회로로만 구성된 것이 아니라 마이크로프로세서가 내장되어 있고 그 마이크로프로세서를 구동하여 특정한 기능을 수행하도록 프로그램이 내장되어 있는 시스템을 가리키는 것이다. 이러한 임베디드 시스템은 최근 IT기술의 발달로 마이크로프로세서가 저가, 소형화, 고성능화 되어가는 것과 맞물려 지속적으로 성장하고 있다[1].

임베디드 시스템이라는 것은 통상적으로 일반적인 시스템과는 달리 특정한 작업만을 수행하도록 설계되어지며, 초기의 임베디드 시스템은 비교적 단순해서 특별한 운영체제가 필요없이 순차적인 프로그램을 작성해서 실행하도록 설계되었다. 하지만 최근 임베디드 시스템 자체가 상당히 커지게 되었고 네트워크나 멀티미디어가 시스템에 기본적으로 자리 잡으면서 순차적인 프로그램으로 작성하기가 매우 어려워졌다. 이에 따라 임베디드 시스템에서도 운영체제의 개념이 필요하게 되었고, 현재 임베디드 시스템을 위해 가장 많이 사용되고 있는 운영체제가 임베디드 리눅스이다[1].

하지만 임베디드 리눅스는 임베디드 시스템을 위한 범용적인 운영체제로써 그 자체로 모든 응용시스템에 적용하기에는 용량, 부팅속도, 기능 등에서 몇 가지 문제점을 가지고 있다. 특히, 이미지 프로세싱이나 단순한 산술연산 및 데이터 입·출력을 처리하는 임베디드 시스템 환경에서는 부팅속도 및 루트 파일시스템, 그리고 커널 경량화 등이 중요한 요소로 부각되고 있다[1-7].

따라서 본 논문에서는 기존의 임베디드 리눅스를 경량화하는 방법을 개선하고 통합하여 임베디드 시스템에서 동작하는 어플리케이션의 처리 속도 개선에 어떤 효과가 있는지를 검증하고자 한다. 이를 위해 Hibernation 기법을 응용한 부팅시간 단축, Symbolic Link 기법과 가상주소매핑 기법을 적용한 JFFS2 파일시스템 최적화, 범용성을 보장하는 커널 경량화 방법을 적용하였으며 실험을 통해 본 논문에서 제안하는 경량화 기법이 임베디드 시스템의 성능 향상에 어떤 효과가 있는지를 확인하였다.

II. 관련 연구

1. 임베디드 리눅스 경량화 방법

임베디드 리눅스는 임베디드 시스템을 위한 범용적인 운영체제로써 그 자체로 모든 응용시스템에 적용하기에는 몇 가지 문제점을 갖고 있다. 따라서 이러한 문제를 해결하기 위하여 시스템에서 사용하는 임베디드 리눅스를 임베디드 시스템 자체 혹은 해당 시스템에서 특정 목적으로 사용하도록 설계되어진 응용어플리케이션의 목적에 맞게 수정·보완해서 사용해야 한다. 이를 임베디드 리눅스 경량화(Light-Weight) 혹은 최적화라고 부르며 일반적인 임베디드 시스템에서 사용하는 운영체제를 경량화 하는 방법은 크게 부팅 메커니즘 개선, 루트 파일시스템 경량화, 그리고 리눅스 커널 경량화 등 세 가지 방향으로 진행된다.

첫 번째 경량화 방향인 부팅 메커니즘 개선은 범용 운영체로의 부팅 시간에 영향을 주는 요소를 최적화하여 부팅 시간을 최소화하는 것이다. 부팅 시간 최소화의 일반적인 방법은 임베디드 장치의 하드웨어 구성과 어플리케이션의 목적 등을

고려하여 부팅 요소를 구성하는 모듈 중에서 불필요한 부분을 제거하는 것이다. 다른 방법으로는 정상적인 부팅 과정에서 필요한 요소만으로 분리하여 우선 적재하고 부가적인 요소에 대해서는 필요할 때 적재하도록 함으로써 부팅 시간을 단축하는 것이다.

두 번째 경량화 방향은 루트파일시스템의 효율적인 적용이다. 임베디드 리눅스는 기본적으로 여러 가지 파일시스템을 제공하고 있다. 또한 이들 각각은 특성에 맞는 장단점을 보유하고 있기 때문에 현재 임베디드 시스템에서 사용하고 있는 플래시메모리 및 SDRAM의 종류, 응용어플리케이션의 목적에 따라 각기 다른 파일시스템을 선택하여 사용한다. 이때, 페이징 시간 단축 및 I/O 처리 속도를 개선할 수 있는 방법들을 적용함으로써 효율성을 높이고 있다.

마지막으로 살펴볼 경량화 방향은 리눅스 커널을 경량화하는 것이다. 이 방법 또한 응용어플리케이션의 목적에 따라 크게 필요하지 않은 기능을 적재하지 않거나, 기능 중 선별적으로 적재함으로써 임베디드 시스템의 커널 크기를 줄이고 시스템의 효율을 높이는 것이다. 리눅스 커널 경량화는 일반적으로 커널 Configuration 최적화를 통해 수행한다. 즉, 임베디드 시스템이 구동하는데 꼭 필요한 메모리 및 프로세스 관리, 인터럽트, 시그널 관리 및 파일시스템 기능만이 포함되도록 커널 이미지를 생성하는 것이다. 이때, 커널의 각 기능 간의 의존성을 해치지 않는 범위 내에서 이루어져야 하는데 그 접점을 찾는 것은 매우 어렵다. 따라서 일반적으로 필수 기능을 제외한 나머지 부수적인 기능들 중 시스템에서 동작하는 어플리케이션의 목적이나 성격에 맞게 추가적으로 선택하여 커널 이미지를 생성한다. 위에서처럼 기능 전체를 적재하느냐 하지 않느냐를 결정하기도 하지만 오픈된 리눅스 커널 소스를 직접적으로 변경하여 불필요한 요소를 제거하는 방법도 있다. 또한, 커널 이미지를 생성할 때 사용하는 gcc 컴파일러의 최적화 옵션을 사용해 리눅스 커널 경량화를 수행하기도 한다.

앞에서 살펴본 세 가지의 임베디드 리눅스 경량화 방법은 독립적으로 연구되기도 하지만 서로 유기적으로 연결되어 다른 방법의 결과에 영향을 미친다. 따라서 각각의 경량화 방법을 개선하고자 할 때, 다른 부분에 어떤 영향을 미치는지 면밀히 분석한 후 적용하여야 한다.

2. 임베디드 리눅스 경량화에 대한 기존 연구

본 절에서는 임베디드 시스템 환경에서 리눅스 운영체제 경량화를 위한 선행 연구를 살펴보고, 각 연구들의 장단점에 대하여 살펴본다.

2.1 빠른 부팅을 위한 기존 연구

임베디드 시스템에서 빠른 부팅 및 로딩을 구현하기 위하여 XIP 기법에 관한 연구가 진행되었다[2-3]. XIP 기법은 실행 코드를 RAM으로 복사하지 않고 플래시 메모리 영역에서 바로 실행하는 기법으로 플래시 메모리를 보조 저장장치로 채택하고 있는 임베디드 시스템 환경에서 쉽게 적용될 수 있다. 그러나 플래시 메모리는 SDRAM에 비해 10배 정도 느린 접근 시간을 가지고 있어 운영체제와 같은 실행빈도가 높은 코드를 플래시 메모리에서 실행하는 것은 시스템 전체의 성능을 저하시킬 수 있는 문제점이 있다. 따라서 최근에는 XIP 기법을 통해 부팅 시간을 감소시키는 연구보다는 주기억장치의 사용량을 줄이고 파일 입·출력의 효율을 높일 수 있는 기법으로 연구되고 있다[2-3].

다음으로 임베디드 리눅스의 빠른 부팅을 위하여 활발히 연구되는 분야는 스냅샷을 이용한 Hibernation 방식이다[4]. 그림 1과 같이 이 방식은 현재의 시스템 상태를 기록한 이미지를 스냅샷 형태로 플래시 메모리에 기록해 두었다가 부팅 시 저장된 상태를 그대로 복원함으로써 일반적인 부팅과정에서 발생하는 초기화 과정을 감소시켜 부팅 시간을 단축한다.

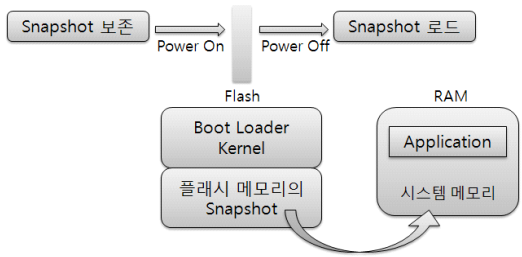

그림 1. Hibernation 기법을 이용한 빠른 부팅 원리
Fig.1. Fast Booting Concept using Hibernation

Hibernation 기법이 활발히 연구되는 이유는 임베디드 리눅스를 부팅할 때 부트로더가 플래시 메모리에서 RAM으로 커널 이미지를 복사해야 하는데 이때, 전체 부팅 시 소요되는 시간 중 상대적으로 많은 시간이 소요되기 때문이다. 아울러 각종 초기화 과정을 수행하면서 부트로더가 커널에 포함될 기능과 디바이스들에 대한 정보를 플래시 메모리에서 RAM으로 빈번히 이동시켜야 한다. 여기서 현재의 시스템 상태를 통합된 이미지로 저장하였다가 그대로 RAM 영역으로 복사함으로써 기존의 부트로더의 로드를 감소시킬 수 있다. 그러나 Hibernation 기법을 이용한 빠른 부팅 방식은 항상 저장된 상태로 복원되므로 시스템 사용 중 변경되거나 기록된 사항들이 모두 초기화 된다는 단점을 가지고 있다. 이는 시스

템 사용 중 변경되는 항목들에 대하여 유연하게 대처할 수 없다는 뜻이다. 또한 하드웨어에 매우 큰 의존성을 가지고 있어 만들어진 스냅샷 이미지를 다른 시스템에 적용하기 쉽지 않다는 단점도 가지고 있다.

2.2 효율적인 파일시스템 적용을 위한 기존 연구

앞에서 살펴본 빠른 부팅 못지않게 임베디드 리눅스 경량화를 위하여 효율적인 파일시스템 적용도 중요한 요소이다. 임베디드 시스템에서 비용 효율성은 가장 중요한 요소 중의 하나이며, 시스템 목적에 적합한 파일시스템을 적용함으로써 비용 효율성을 높일 수 있기 때문이다. 임베디드 리눅스는 기본적으로 범용적으로 설계되었기 때문에 개발자로 하여금 시스템에 가장 적합한 파일시스템을 선택할 수 있도록 다양한 형태의 파일시스템을 제공하고 있다. 따라서 효율적인 파일시스템 적용을 위한 기존 연구들은 선택한 파일시스템을 동작하면서 발생하는 오버헤드를 어떤 방법을 이용하여 감소시키는지에 초점이 맞춰져 있다[5].

이런 연구들 중 가장 활발히 연구되는 분야는 압축파일시스템을 사용하여 효율성을 높이며 제한된 저장장치를 효과적으로 사용하는 방법을 검증하는 것이다. 특히, 비효율적인 압축·해제 구조 개선을 통해 불필요한 오버헤드를 해결하고 효율적인 압축·해제 기법을 제시하는 연구가 진행되고 있다. 그 중 높은 압축률이 큰 장점인 SquashFS의 성능향상 연구는 압축해제를 위해 사용되는 비연속적인 내부버퍼를 그림 2에서와 같이 각 버퍼에 저장된 압축 블록들을 하나씩 압축 해제하여 내부버퍼에 임시 저장하였다가 압축해제가 완료되면 내부버퍼의 데이터들을 파일의 주소 공간으로 해당 페이지를 복사하여 효율을 높여주었다.

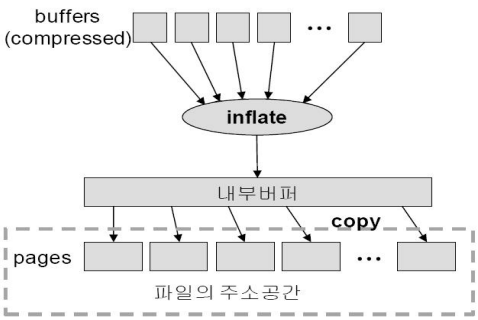

그림 2. SquashFS의 압축 해제 과정
Fig. 2. Decompression Process of SquashFS

그러나 위 방법은 다음과 같은 문제점을 갖고 있다. 첫째, 압축해제 함수의 반복적인 호출로 인한 오버헤드이다. 그림 2에서 보는바와 같이 압축 블록들이 저장된 버퍼공간이 비연속적으로 존재하기 때문에 각 버퍼 별로 압축해제를 실행하여야 하는 한계를 가지고 있다. 두 번째 문제점으로는 불필요한 복사 오버헤드의 발생이다. 압축 해제된 데이터를 내부버퍼에 임시로 저장하였다가 모두 완료된 후 파일의 주소 공간으로 복사하는데 이때, 주소 공간에 복사하는 페이지의 크기가 고정되어 있어 여러 번 플래시 메모리에서 데이터를 읽어와야 하는 불필요한 시간의 낭비가 발생한다.

2.3 리눅스 커널 경량화를 위한 기존 연구

리눅스 커널 경량화를 위한 연구 방향은 단순한 형태로 진행되었다. 임베디드 시스템의 하드웨어 구성과 응용어플리케이션의 목적에 따라 크게 필요하지 않은 기능을 적재하지 않는 방식이 그것이다. 이때 임베디드 리눅스에서 제공하는 Configuration 옵션을 이용하여 설정하는데 개발자에 의해 선별적으로 적재함으로써 임베디드 시스템의 효율을 높일 수 있다[1][6-7].

하지만 리눅스 커널 경량화를 위하여 옵션을 조정할 때, 임베디드 시스템이 구동하는데 꼭 필요한 기능들은 반드시 적재되어야 하고 개발자의 요구에 의해 불필요한 기능들을 제거하는 방식이기 때문에 이렇게 설정된 커널 경량화 옵션을 다른 임베디드 시스템에 그대로 적용할 수 없다는 한계를 가지고 있다. 즉, 표준화된 리눅스 커널 경량화 옵션을 결정하기 어렵다. 이는 시스템에서 동작하는 응용어플리케이션과 하드웨어 구성에 의해 선택적으로 결정할 수밖에 없기 때문에 쉽게 해결할 수 없는 한계이다. 이와 같은 한계를 극복하기 위하여 오픈된 리눅스 소스를 직접적으로 변경하여 불필요한 요소를 제거하는 방법이 연구되고 있지만[7] 여전히 타깃 시스템에 종속된 형태로 진행되고 있다.

III. 임베디드 리눅스 경량화 기법

본 논문에서는 임베디드 시스템에서 동작하는 어플리케이션의 처리 속도를 개선하기 위하여 임베디드 리눅스 경량화 기법을 제안한다. 본 논문에서 제안하고 있는 경량화 기법은 크게 세 가지 형태로 진행되며 이들 각각은 서로 유기적으로 연결되어 자원 효율과 처리 속도를 개선할 수 있도록 설계되었다. 다음 그림 3은 본 논문에서 제안하고 있는 임베디드 리눅스 경량화 기법의 전체 구성도이다.

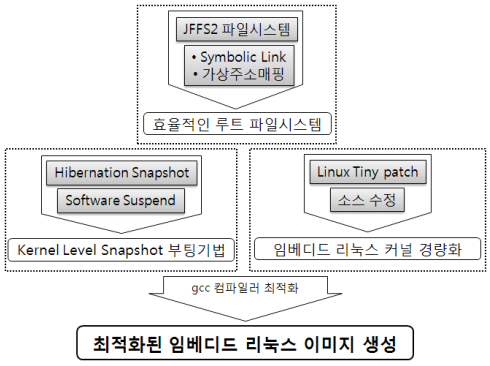

그림 3. 제안하는 경량화 기법 전체 구성도
Fig. 3. Proposed Light-Weight Diagram

그림 3에서와 같이 효율적인 루트파일시스템, 빠른 부팅 시간을 위한 부팅 기법 및 리눅스 커널 경량화를 수행한 후, gcc 컴파일러 최적화를 이용하여 임베디드 리눅스 이미지를 생성하면 RAM으로 복사 시간, 압축해제 시간, 프로세스 초기화 시간, 자원의 효율적인 이용, 입출력 관련 오버헤드 제거 등과 같은 효과를 얻을 수 있으며 이는 곧 임베디드 시스템에서 동작하는 어플리케이션의 처리 속도를 향상시킬 수 있다.

1. 부팅 최적화 메커니즘

본 논문에서는 부팅 메커니즘을 최적화하고 이를 통해 빠른 부팅 시간을 확보하기 위하여 다음 두 가지 방법을 적용하였다. 첫째, 2.1절에서 살펴본 Hibernation 방법이다. 앞서 설명하였듯이 Hibernation 기법은 스냅샷 이미지를 이용하여 부팅과정에서 발생하는 초기화 과정을 감소시켜 부팅 시간을 줄여주지만 하드웨어 의존성이 강하고 시스템 상황에 유연하게 대처할 수 없다는 단점을 가지고 있다. 따라서 본 논문에서는 Hibernation 기법이 가지고 있는 초기화 과정을 감소시키는 장점은 수용하면서 단점을 보완하기 위하여 Software Suspend 알고리즘을 적용하였다. Software Suspend 알고리즘이란 프로그램 혹은 프로세스(Task)가 Running 상태에서 Suspend 상태로 변경될 때의 메모리 상태 및 PC레지스터 등이 변경되는 원리를 이용하여 멀티쓰레딩, 멀티태스킹을 지원하는 방법으로 이를 임베디드 리눅스 커널 환경에 맞게 변경하여 적용하였다.

일반적인 Hibernation 기법은 커널 이미지 복사 및 초기화 과정을 부트로더가 담당한다면, 본 논문에서 제안하는 부팅 최적화 메커니즘에서는 최초 커널 이미지 복사는 부트 로더가 담당을 하되, 커널이 RAM에 복사되어 실행할 때 부트로더를 Suspend 시키고 이후 커널이 주체가 되어 다시 한 번 수행을 시키는 구조를 갖는다. 이 방식은 일반적인 Hibernation 기법을 이용한 부팅보다 느리지만 커널 레벨에서 초기화를 수행하기 때문에 높은 하드웨어 의존성을 낮출 수 있으며 시스템에 유연하게 대처할 수 있어 높은 이식성을 얻을 수 있다. 다음 리스트 1은 Software Suspend 알고리즘을 Hibernation 기법에 적용한 의사코드이다.

의사코드에서와 같이 본 논문에서 제안하는 부팅 메커니즘에서는 시작은 부트로더가 담당을 하지만 최초 커널 이미지를 로드한 후에는 커널에서 모두 담당하도록 설계하였다. 이를 통해 범용 임베디드 리눅스가 갖고 있는 유연함과 높은 이식성을, Hibernation 기법이 갖고 있는 빠른 부팅 시간을 모두 획득할 수 있었다.

2. 루트 파일시스템 경량화 기법

임베디드 리눅스는 루트 파일시스템을 통해 하드디스크나 I/O 디바이스 등에 접근하는 구조를 가지고 있다. 매우 빈번한 접근이 발생할 뿐만 아니라 Read-Only 파일시스템 특성 등을 고려하여 시스템의 성능을 향상시키기 위해 RAM에 루트 파일스세템을 구축하는 구조를 갖는다. 이 루트 파일시스템도 커널과 동일하게 압축된 상태로 플래시 메모리에 저장되어 있으며 부팅 시에 RAM으로 복사되어 압축을 해제하는 과정을 거치므로 루트 파일시스템의 압축해제 효율을 개선하여 경량화를 시킬 수 있으며, 이를 통해 부팅 속도뿐만 아니라 전반적인 임베디드 시스템의 성능을 향상시킬 수 있다.

본 논문에서는 JFFS2 파일시스템을 루트 파일시스템으로 결정하고 Symbolic Link 기법과 가상주소매핑 기법을 적용하여 효율을 높이고자 하였다. JFFS2 파일시스템은 임베디드 리눅스에서 기본적으로 제공하는 파일시스템으로 저널링 방식을 사용한다. 따라서 가비지 컬렉션에 시간이 많이 걸린다는 단점을 가지고 있지만, 높은 압축 효율을 가지고 있으며 작은 비용으로 파일 시스템의 향상성을 유지할 수 있다는 큰 장점을 가지고 있다. 본 절에서 제안하는 루트 파일시스템 경량화는 메모리에 적재되는 루트 파일시스템의 크기를 최소화하기 위하여 Symbolic Link를 사용하고, 분리된 메모리 구조를 갖고 있음으로 인해 발생하는 빈번한 I/O 시 효율을 높이기 위하여 vmap()과 vunmap()을 이용한 가상주소매핑 방법을 적용하였다. 첫 번째 Symbolic Link를 사용함으로 인해 메모리 자원을 효율적으로 사용할 수 있지만, 반복적인 I/O 발생과 압축해제 함수의 호출이라는 오버헤드가 발생한다. 여기서 이러한 오버헤드를 감소시키기 위하여 가상주소매핑 방식을 도입한 것이다. 다음 그림 4는 JFFS2 파일시스템을 기본으로 RC 스크립터 단계에서 Symbolic Link를 구성

리스트 1. Software Suspend in Hibernation Pseudo-code
List 1. Software Suspend in Hibernation Pseudo-code

```
int hibernate_nvs_register
(unsigned long start, unsigned long size)
{
struct nvs_page *entry, *next;
while (size > 0) {
커널사이즈 계산 및 할당();
if (!entry)
goto Error;
페이지 사이즈에 맞게 동적 구성();
// Linked List로 구현하여 가장주소
할당이 가능하도록 설계
}
return 0;
Error:
list_for_each_entry_safe(entry, next, &nvs_list, node)
{
시스템에 가상 페이지 주소 반납();
}
return -ENOMEM;
}
sector_t alloc_swapdev_block(int swap)
{
unsigned long offset;
offset = swap 페이지 타입에 따른 오프셋 계산();
allocate a swap page();  // 페이지 할당
register allocation();  // 레지스터 변경
return 0;
}

int swsusp_swap_in_use(void)
{
// 커널 레벨에서 변경된 내용이 있을 때
이미지를 새로 저장하도록 처리
return (swsusp_extents.rb_node != NULL);
}

static int swsusp_extents_insert
(unsigned long swap_offset)
{
struct rb_node **new = &(swsusp_extents.rb_node);
struct rb_node *parent = NULL;
struct swsusp_extent *ext;

while (*new)
// 디바이스 혹은 시스템 변경에 따른 수정 사항 체크
ext = container_of(*new, struct swsusp_extent,
node);
종료조건 체크();
rebalance the tree();
// swsusp 블록 처리를 위한 노드 초기화
}
```

하여 루트 파일시스템을 최소화한 메모리 맵 구조를 도식화한 것이다. Symbolic Link를 이용하여 루트 파일시스템을 확장 루트파일시스템과 기본루트파일시스템으로 분리하여 설정한 후 부팅 시 기본루트파일시스템만 적재하여 수정전과 비교하였을 때 메모리 사용량이 작아졌음을 확인할 수 있었다.

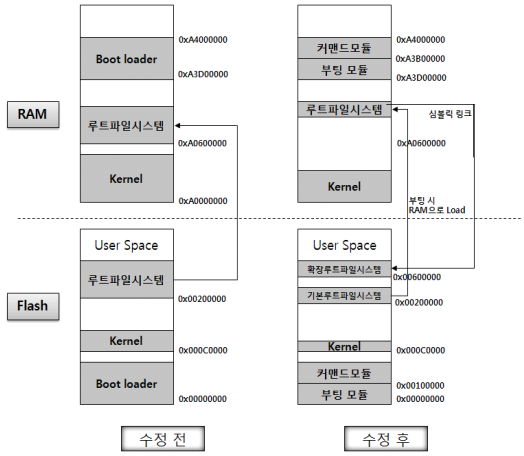

그림 4. 수정전과 Symbolic Link 수행 후 메모리 맵 구조
Fig. 4. Memory Map after Symbolic Link

다음으로 Symbolic Link를 이용하여 파일시스템을 분리 적재함으로 발생하는 오버헤드를 제거하기 위한 가상주소매핑 기법을 적용하였다. 가상주소매핑 기법은 임베디드 리눅스 커널에서 제공되는 API인 vmap()과 vunmap()을 사용하여 구현할 수 있다. 필요한 가상주소 공간을 할당받고, 할당받은 가상주소 공간에 페이지들을 매핑 하는 방법으로 간단히 구현된다. 필요한 작업이 끝나면 할당받은 가상주소 공간에 매핑된 페이지들을 매핑 해제시키고 할당받은 가상주소 공간을 반환해야 한다. 이 과정은 페이지들을 매핑 시키는 과정이 RAM 영역에서 동작하기 때문에 순서가 틀리면 압축해제를 실행할 때마다 메모리 누수가 발행하므로 매우 중요하다. 다음 그림 5는 vmap/vunmap을 사용하여 가상주소매핑 기법을 이벤트 흐름에 따라 보여주고 있다. 기존의 JFFS2 방식은 읽기 작업을 수행할 때마다 반복적으로 새로운 가상주소를 할당받아야 하는 문제를 가지고 있었다. 이와 같은 오버헤드를 감소하기 위하여 본 논문에서는 메모리상에 비연속적인 공간에 흩어져 있는 압축 블록들을 연속된 가상공산으로 연결시켜 한꺼번에 접근할 수 있도록 개선하였으며 이를 통해 한 번의 압축해제 함수 호출로 압축을 해제할 수 있으며, 내부버퍼를 사용하는 대신 주소공간의 페이지들을 연속된 공간처럼 사용할 수 있어 내부버퍼에서 데이터를 복사하는 오버헤드를 제거하였다.

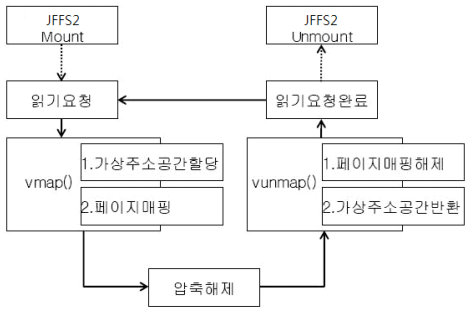

그림 5. vmap/vunmap을 이용한 가상주소매핑 흐름도
Fig. 5. Flow of Virtual Address Mapping

또한 기존 파일시스템에서 읽기 작업을 수행할 때마다 반복적으로 새로운 주소를 할당받아야 하는 오버헤드도 필요한 공간을 플래시 메모리와 RAM 영역에서 가상주소 할당과 페이지 매핑을 따로 구분하고 동작시켜 해결하였다. 그림 6에서 보는바와 같이 가상주소 공간을 할당/반환하는 작업은 마운트와 언마운트 시점에서만 수행하므로 연속적인 공간에 페이지와 가상주소를 매핑 시킬 수 있으며 빈번히 발생하는 가상주소 매핑으로 인한 오버헤드를 제거하였다.

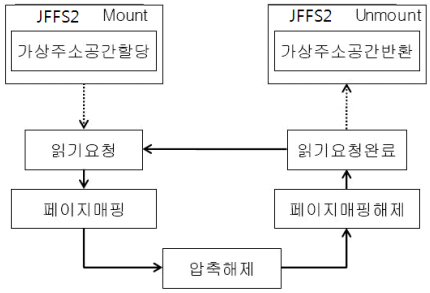

그림 6. 최적화된 가상주소 매핑 방식
Fig. 6. Optimization Virtual Address Mapping

3. 리눅스 커널 경량화 기법

본 논문에서 제안하는 리눅스 커널 경량화 기법은 크게 두 가지로 나누어 설계하였다. 첫 번째 방향은 Linux Tiny 프로젝트에서 수행한 패치를 타깃 임베디드 리눅스에 포팅한 것이며, 두 번째 방향은 gcc 컴파일러의 최적화 옵션을 사용하여 선택적으로 커널 이미지를 빌드한 것이다.

3.1 Linux Tiny Patch 적용

Linux Tiny 프로젝트[13]는 리눅스가 소형시스템에서도 동작할 수 있도록 커널 2.6을 기반으로 커널의 메모리와 디스크 사용량을 경량화한 패치셋이다. Linux Tiny 프로젝트가 소형 디바이스, 레거시 시스템을 대상으로 설계되었기 때문에 본 논문에서는 여러 가지 패치 내용 중 범용성을 해치지 않고 다양한 시스템에 적용할 수 있도록 재사용성을 고려하여 선별한 후 적용하였다. 다음은 본 논문에서 커널을 경량화 하기 위하여 설정한 항목과 그 효과를 설명하였다.

- kill-printk.patch : 커널 이미지에서 메시지 출력을 제거한다. printk 함수를 return 부분만을 가지는 inline printk 함수로 대체하고, kernel/printk.c 파일의 log_buf 관련 변수들과 console, log 관련 함수들을 제거하여 커널의 플래시 메모리와 RAM 사용량을 줄이고 실행속도를 높인다.
- no-aio.patch : 쓰레드 애플리케이션에서 성능을 높이기 위해 사용되는 POSIX 비동기 IO 함수들을 제거하여 커널 크기를 줄인다. fs/aio.c 파일의 aio_complete 함수, exit_aio 함수 등과 kernel/sysctl.c 파일 fs_table 변수의 aio 관련 부분을 제거한다.
- no-attr.patch :파일 시스템 확장 애트리뷰트 시스템콜에 관련된 함수를 제거한다. fs/Makefile에서 xattr.o 오프젝트 파일 관련 부분이 제거된다.
- posix-timers.patch : 실시간 어플리케이션에서 사용되는 POSIX 타이머 관련 함수를 제거한다. kernel/posix-timer.c 파일에서 posix_timer_id, posix_clocks, init_posix_timer와 같은 POSIX 타이머 관련 선언과 변수, 함수들을 모두 제거하여 커널의 크기를 줄인다.

위에 적용된 사항들은 임베디드 시스템에서 어떤 응용어플리케이션이 동작하여도 어플리케이션 실행과는 전혀 무관한 항목들이다. 따라서 본 논문에서 제안한 적용 범위를 포함하여 어플리케이션 목적에 따라 불필요한 요소들을 효과적으로 분리 적재하면 높은 커널 경량화 효과를 얻을 수 있을 것이다. 이때 기본적으로 제공하는 make menuconfiguration 기능을 사용하면 된다. 하지만 본 논문에서는 범용성과 재사용성에 초점을 맞추었기 때문에 위에 열거한 항목들만 제거하여 커널 경량화를 꾀하였다.

3.2 gcc 컴파일러 최적화 옵션

다음으로 gcc 컴파일러 최적화 옵션을 사용하여 실행 파일의 크기, 컴파일 시간, 실행 속도와 같은 조건을 고려하여 선택적으로 커널 이미지를 빌드하였다. gcc 컴파일러는 표 1과 같은 여러 단계의 최적화 옵션을 제공하고 있으며, 코드 크기를 최소화할 수 있는 최적화 옵션을 사용해 커널 이미지의 ROM 크기를 경량화 할 수 있다. 본 논문에서는 최적화 옵션 중 -O1을 이용하였다. 본 논문에서와 같이 타깃 임베디드 시

스템이 다양한 어플리케이션이 동작된다면, 컴파일러에서 지원하는 모든 최적화를 수행하였을 경우, 어플리케이션에 따라 컴파일 시간이 다소 오래 걸리고 코드의 크기가 증가하여 오히려 역효과를 초래하는 경우도 발생한다. 따라서 다양한 어플리케이션이 동작되는 임베디드 시스템이라면 효과는 비교적 다른 옵션에 비해 작지만 어플리케이션 실행 속도를 향상시키고 루트 파일시스템 및 부팅 메커니즘에도 좋은 영향을 주는 -O1 옵션을 선택하길 권장한다.

표 1. gcc 컴파일러 최적화 옵션
Table 1. gcc Compiler Optimization Option

-O	-O1 옵션과 동일
-O1	컴파일 시간은 고려하지 않고 코드 크기를 줄이고 실행 속도를 향상시키는 것을 목표로 최적화
-O2	컴파일러에서 지원되는 거의 모든 최적화를 수행. 루프 전개, 함수의 인라인 전개, 레지스터 리네이밍 부분은 제외. 컴파일 시간이 다소 오래 걸리고 생성 코드의 크기가 증가.
-O3	-O2 옵션의 모든 최적화를 수행하고 루프 전개, 함수의 인라인 전개, 레지스터 리네이밍도 수행.
-O0	최적화를 수행하지 않는다.
-Os	코드 크기를 최적화. -O2 옵션에서 수행되는 최적화 중 코드 크기를 증가시키지 않는 최적화를 모두 수행하며, 코드 크기를 줄이기 위해 설계된 추가적인 최적화 옵션을 수행.

IV. 실험 및 결과분석

본 논문에서 사용할 임베디드 시스템은 휴인스에서 제작한 XP-100이다. XP-100은 10만 게이트의 FPGA가 집적되었으며, LCD, Ethernet, Camera I/F 등 많은 주변기기들을 보유하고 있으며 이 시스템의 세부사항은 다음 표 2와 같다.

표 2. Hunins XP-100 임베디드 시스템 주요 사양
Table 2. Huins XP-100 Embedded System Specification

항목	세부스펙	기타
Processor	ARM922T	
FPGA Gate	10만 Gate	
Core Speed	133Mhz	
Internal Memory	SRAM:32KB, DPRAM:16KB	
Internal Logic	UART, SDRAM, Timer, Interrunpt, Watchdog Timer, EBI Controller	
FLASH	16MB(8MB × 2EA)	EBI
SDRAM	32MB	

1. 커널 및 점유 메모리 크기 비교

제한된 리소스를 사용해야만 하는 임베디드 시스템의 경우, 속도도 중요한 요소이긴 하지만, 메모리를 비롯한 제한된 리소스를 얼마나 잘 활용하느냐가 경량화의 중요한 요소이다. 바꾸어 말하면 속도는 기다리면 해결되지만 한정된 리소스를 효율적으로 활용하지 않으면 아예 수행 자체가 불가능하기 때문에 효율적인 자원 활용은 매우 중요하다.

본 논문에서 제안한 임베디드 리눅스 경량화 기법을 적용한 후 기존 커널과 비교한 결과는 다음 표 3과 같다. 기존의 커널은 일반적인 임베디드 리눅스 커널을 사용하였다. 여기에 본 논문에서 제안한 경량화 기법은 범용성과 재사용성을 고려하여 설계되었기 때문에 기존 임베디드 리눅스가 지원하던 대부분의 기능을 모두 지원하면서도 기존 커널의 크기를 9.6% 개선할 수가 있었다.

표 3. 기존 커널과 경량화를 수행한 커널의 크기 비교
Table 3. Kernel Size Comparison

	기존	경량화	비교
커널사이즈 (Byte)	997,640	901,866	9.6%

다음 표 4는 부팅 과정 중에 나오는 Freeing init Memory의 크기와 부팅을 완료하는데 까지 소요되는 시간을 비교해 놓은 표이다. 초기화 직전까지 사용된 이후, 반환되는 메모리의 크기로 경량화를 수행한 커널이 약 4Kb 정도 적게 소모되는 것을 알 수 있다. 또한 Hibernation 기법을 응용한 부팅 방식과 루트 파일시스템 경량화 기법에 사용된 Symbolic Link 기법을 통해 부팅 시 발생할 수 있는 오버헤드를 대부분 줄여주었기 때문에 만족할 만한 결과를 얻을 수 있었다.

표 4. 기존 커널과 경량화를 수행한 커널의 사용 메모리 및 부팅 시간 비교
Table 4. Kernel Using Memory Size and Booting Time Comparison

구분	기존	경량화	비교
Freeing init Memory	52Kb	44Kb	7.3% 개선
부팅 완료 시간	93초	76초	18% 개선

2. 어플리케이션 처리 속도 비교

앞에서 부팅 완료 시간, 커널의 크기, Freeing init

Memory의 크기 비교를 통하여 임베디드 리눅스 운영체제가 범용성과 재사용성을 해치지 않는 범위에서 얼마나 효율적으로 경량화가 가능한지를 확인하였다. 본 절에서는 임베디드 시스템에서 동작하는 어플리케이션의 처리 속도를 비교해 보고 본 논문에서 제안한 경량화 기법을 통해 얼마나 개선하였는지를 확인한다.

본 논문에서는 타깃 시스템에서 동작을 검증하는 어플리케이션으로 에지검출 프로그램을 이용하여 BMP 파일의 변환 시간을 측정하였다. 경량화를 적용하여 만들어진 임베디드 리눅스 운영체제 하에서 처리 속도가 개선되었는지를 확인하였다. 에지검출 프로그램을 크로스 컴파일러를 이용해 컴파일한 후 시리얼 케이블을 통해 전송하였고, 입력으로는 526KB 표준 BMP 파일을 이용하였다. 실험은 BMP 파일 변환 시간에 있어서 각각 기존 커널 사용 시의 최적, 최악의 경우와 경량화 된 커널 적용 후 최적, 최악의 경우를 선정하여 비교하였다. 최적의 경우는 수차례의 테스트 중 가장 짧은 시간이 걸린 경우이며, 최악의 경우는 가장 긴 시간이 걸린 것이다. 다음 그림 7은 에지검출 프로그램의 실행결과를 정리한 것이다. 최적의 경우 기존 대비 1초가 줄어 11%의 성능 향상으로 다소 미비하지만 최악의 경우는 기존 임베디드 시스템 환경에서 61초가 소요됨에 반해, 경량화가 적용된 시스템에서는 21초가 소요되어 약 66%의 처리 속도 향상이 있음을 확인할 수 있었다. 이는 본 논문에서 제안한 경량화 기법을 적용함으로써 어플리케이션의 처리 속도 개선에 효과가 있음을 보여주고 있다.

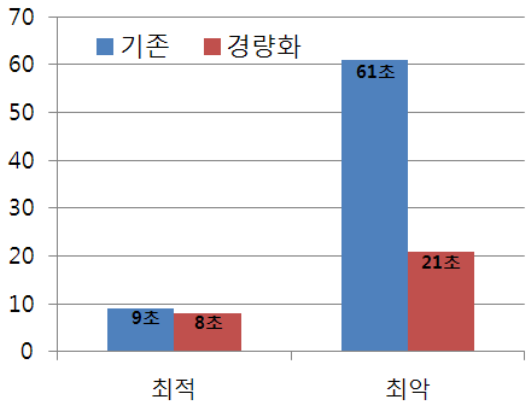

그림 7. 에지검출 프로그램을 이용한 성능 비교
Fig. 7. Performance Comparison using Edge Detecting

V. 결론

본 논문에서는 리눅스를 탑재한 임베디드 시스템에서 임베디드 리눅스 경량화 기법을 제안하고 적용함으로써 효율적인 자원 활용과 빠른 부팅 시간 및 시스템에서 동작하는 어플리케이션의 처리 속도가 개선됨을 확인하였다.

빠른 시간 안에 부팅을 완료하기 위하여 Hibernation 기법에 Software Suspend 알고리즘을 리눅스 환경에 맞게 적용하였으며, 효율적인 루트 파일시스템을 적용하기 위하여 기본적으로 제공하는 JFFS2 파일시스템에 메모리 사용 효율을 높일 수 있는 Symbolic Link 기법과 압축해제 과정에서 발생할 수 있는 오버헤드를 줄이기 위하여 vmap()과 vunmap()을 이용한 가상주소매핑 기법을 제안하였다. 마지막으로 일반적인 임베디드 리눅스가 갖고 있는 범용성과 재사용이 가능한 리눅스 커널 경량화를 위하여 Linux Tiny 프로젝트에서 수행된 패치 항목 중 선별하여 제거하였다.

실험결과 경량화를 적용한 후에도 시스템에서 동작하는 어플리케이션이 정상적으로 동작하였음은 물론이며, 부팅 및 어플리케이션 실행 속도의 향상 및 기존 커널 대비 많은 리소스 확보가 가능하였다. 그러나 시스템에서 동작하고 있는 어플리케이션들이 안정적으로 운용되고 있을 때에는 경량화를 통한 효과는 크게 나타나지 않았다. 따라서 정밀한 커널 Configuration 조정으로 커널 사이즈 최적화하는 연구 및 소스 레벨에서 커널 사이즈를 줄일 수 있는 연구, 루트 파일시스템에서 동적라이브러리의 효율적인 관리 기법 등이 이 문제를 해결할 수 있을 것이며 이는 중요한 향후 연구과제가 될 것이다.

참고문헌

[1] Huins, "System On Chip and Embedded System Design using ARM9 Core," HongRung Press, 2004.

[2] B. Roman, "Tips and Tricks for Implementing Software Execute-In-Place with Windows CE.NET, Datalight, 2003.

[3] In Hwan Doh, "Implementation of the Hibernation-based Boot Mechanism on an Embedded Linux System," Journal of the Korea society of computer and information, Vol. 16,

No. 5, pp23-31, 2011.
[4] S. Park, J. Song, C. Park, "A Fast Booting Technique using Improved Snapshot Boot in Embedded Linux," Journal of KISS, Vol. 14, No. 6, pp594-598, 2008.
[5] S. Ahn, S. Hyun, K. Koh, "Improving Read Performance of SquashFS for Multimedia Embedded System," Conf. Korea Multimedia Society, pp117-120, 2008.
[6] M. Mackall, "Linux-tiny and directions for small systems," Proc. of the Linux Symposium, Jul. 2004.
[7] LinuxTiny, http://www.selenic.com/linux-tiby/

저 자 소 개

이 태 우

2004~2008: (주)LG전자 단말연구소 선임연구원
2011: 충북대학교 컴퓨터공학과 박사
현 재: 충북대학교 컴퓨터공학과 강의전담교수
관심분야: 임베디드시스템, 유비쿼터스센서 네트워크, 멀티미디어 통신
Email : dickgap@gmail.com

조 지 용

2012년: 충북대학교 컴퓨터공학과 석사 졸업
2012년-현재: 충북대학교 컴퓨터공학과 박사과정
2010년-현재: (주)디노ICT 대표이사
관심분야: 임베디드시스템, 지식베이스구축, 멀티미디어 통신
Email : jycho@dinomedia.co.kr

조 용 환

1989년: 고려대학교 박사 졸업
1978년-1980년: 한국전자통신연구원(ETRI)선임연구원
1982년-현재: 충북대학교 컴퓨터공학과 교수
2007년-현재: (사)엔터테인먼트산업학회 수석부회장
관심분야: 엔터테인먼트기반기술, USN, 멀티미디어 통신, 정보통신정책
Email : yhcho@cbnu.ac.kr