韓國컴퓨터情報學會 論文誌
第16卷 第5號, 2011. 05.

2011-16-5-1-3

임베디드 리눅스 시스템에서 하이버네이션 기반 부팅 방식 구현

도 인 환*

Implementation of the Hibernation-based Boot Mechanism on an Embedded Linux System

In Hwan Doh*

요 약

컴퓨팅 시스템의 부팅 시간 지연 문제는 시스템 소프트웨어 분야에서 중요한 이슈로 부각되어 왔다. 최근 임베디드 시스템 환경에서도 리눅스의 부팅 속도 개선에 많은 관심이 집중되고 있다. 본 연구에서는 임베디드 리눅스 시스템의 대안적인 부팅 방식으로써 하이버네이션을 기반으로 하는 부팅 방식에 주목한다. 본 논문은 다음의 두 가지 측면에서 그 의의를 찾을 수 있다. 첫째, 실제 모바일 전자기기에 널리 활용되는 ARM 임베디드 개발 보드에서 리눅스 버전 2.6.21에 하이버네이션 기반 부팅 방식을 구현한다. 둘째, 하이버네이션 기반 부팅 과정을 시간대 별로 관찰하고 부팅 속도 개선의 여지에 대해서 논의한다. 실제 구현을 통한 부팅 과정 분석 결과, 다양한 최적화 기법이 적용될 경우 하이버네이션 기반 부팅 방식은 전통적인 부팅 방식보다 최대 3.1배 정도 빠른 부팅을 제공할 수 있을 것으로 기대된다.

▸ Keyword : 부팅, 하이버네이션, 임베디드 리눅스

Abstract

Improving system boot time has become one of the most important issues in the system software arena. As Linux is widely used in the embedded system environment, extensive research has been conducted in order to mitigate Linux boot time delay. In this respect, this paper mainly focuses on the Hibernation-based boot mechanism, which is the boot mechanism based on Hibernation, as an alternative to the conventional boot sequence. The contributions of this work are as follows. First, we implement the Hibernation-based boot mechanism on a real embedded Linux system and describe the implementation details. Second, we observe the Hibernation-based boot procedures so that we can investigate the possibility whether the boot mechanism has room for improvement in terms of the boot time. Through the in-depth observation and analysis based on the real

• 제1저자/교신저자 : 도인환
• 투고일 : 2011. 01. 01, 심사일 : 2011. 01. 24, 게재확정일 : 2011. 02. 14.
* (주)프롬나이 (Peromnii Inc.)

implementation, we anticipate that the Hibernation-based boot mechanism which adopts various optimization methods can provide maximum of 3.1 times faster booting performance compared to the conventional way.

▸ Keyword : Booting, Hibernation, Embedded Linux

I. 서 론

컴퓨팅 시스템의 부팅 시간 지연 문제는 시스템 소프트웨어 분야에서 중요한 이슈로 부각되어 왔다. 시스템의 부팅 속도가 개선되면 부팅 시간 지연의 감소로 사용자 편의가 증대됨은 물론 종료된 시스템의 서비스 응답성이 높아져 대기전력 절감에도 효과적일 수 있다. 사용자 입장에서 시스템이 꺼져 있더라도 만족할만한 서비스 응답성을 제공받을 수 있다면 사용자는 유휴 시스템을 대기전력 모드로 전환하는 대신 완전히 종료함으로써 대기전력을 절감할 수 있다.

최근 임베디드 시스템 환경에서도 리눅스의 부팅 속도 개선에 많은 관심이 집중되고 있다. 공개된 운영체제인 리눅스는 서버와 PC 환경뿐만 아니라 휴대전화, 디지털 TV, 셋톱박스와 같은 임베디드 시스템 환경에서도 널리 채택되는 추세이다. 특히 최근 등장한 스마트폰과 타블렛 PC는 안드로이드 플랫폼과 결합된 임베디드 리눅스를 탑재한다. 이와 같은 모바일 전자기기 시장에서 사용자 편의를 향상시킴과 동시에 대기 전력을 절감하려는 목적으로 임베디드 리눅스의 부팅 시간을 단축하려는 움직임들이 다양한 방면으로 전개되고 있다[1-5].

임베디드 리눅스 시스템의 부팅 속도를 개선하려는 노력들은 크게 두 가지 방식으로 구분된다. 첫째는 전통적인 부팅 절차를 최적화하는 방식이다. 예를 들면, 리눅스 커널 컴파일 설정 단계에서 불필요한 커널의 구성요소들을 제거하여 커널 실행 이미지 크기를 축소함으로써 부팅 시 커널 적재에 소요되는 시간을 줄일 수 있다. 둘째, 최대절전모드로 잘 알려진 하이버네이션(Hibernation) 기법을 빠른 부팅의 용도로 활용하는 방식이 있다. 이는 특정 시점의 시스템 상태(State)를 비휘발성 저장매체에 기록해 두고 시스템 종료 후 전원이 재 인가되면 저장되어 있는 시스템 상태로 복원함으로써 기존의 절차적 부팅과정을 생략하고 빠른 시간 내에 시스템을 구동하는 방식이다. 이후, 논문에서는 이와 같은 부팅 방식을 “하이버네이션 기반 부팅”이라는 용어로 참조한다.

전통적인 부팅 절차를 최적화하는 방식에는 두 가지 측면에서 한계가 있다. 하나는 리눅스의 부팅 절차를 분석하여 필수적인 커널 구성요소와 그렇지 않은 구성요소를 분류하여 제거하는데 드는 엔지니어링 비용이 높다는 것이다. 더구나 이러한 작업은 매번 리눅스의 커널 버전이 변하거나 새로운 패치가 추가될 때마다 이루어져야 한다. 다른 하나는 리눅스의 온전한 기능을 모두 제공하지 못한다는 근본적인 한계점이다. 리눅스를 탑재한 최신의 스마트폰이나 타블렛 PC는 모바일 전자기기임에도 불구하고 범용 시스템에 가깝다는 점을 고려한다면 시스템이 진보함에 따라 커널 이미지를 축소함으로써 얻을 수 있는 부팅 속도 개선의 여지는 줄어들 것으로 판단된다.

본 연구에서는 임베디드 리눅스 시스템의 대안적인 부팅 방식으로써 하이버네이션을 기반으로 하는 부팅 방식에 주목한다. 하이버네이션 기반 부팅 방식은 리눅스 커널 버전에 상관없이 동일한 메커니즘이 적용되므로 비교적 적은 엔지니어링 비용을 동반한다. 그리고 리눅스가 제공하는 온전한 기능들을 모두 사용하더라도 여전히 빠른 부팅이 가능하다. 이러한 점을 고려한다면 하이버네이션 기반 부팅 방식은 지속적으로 복잡해지고 있는 임베디드 리눅스 시스템 환경에서 대안적인 부팅 방식으로 주목할 만하다.

본 연구에서는 실제 임베디드 시스템 환경에서 하이버네이션 기반 부팅의 실현 가능성(Feasibility)을 살펴본다. 본 논문이 기여하는 바는 다음의 두 가지로 요약될 수 있다. 첫째, 실제 모바일 전자기기에 널리 활용되는 ARM 임베디드 개발 보드에서 리눅스 버전 2.6.21에 기존하는 하이버네이션 기반 부팅 방식을 구현한다. 둘째, 하이버네이션 기반 부팅 과정을 시간대 별로 관찰하고 부팅 속도 개선의 여지에 대해서 논의한다. 본 논문은 기존 하이버네이션 기반 부팅 방식을 리눅스 커널에 구현해 본 경험을 공유한다는 점에서 그 의의를 찾을 수 있다.

이후 논문의 구성은 다음과 같다. Ⅱ장에서는 관련 연구들을 언급하면서 본 연구를 수행하게 된 동기를 설명한다. 이어서 Ⅲ장에서는 하이버네이션에 대해서 개괄적으로 설명한다. Ⅳ장에서는 리눅스에서 하이버네이션의 동작을 코드 수준에서 분석하고, Ⅴ장에서는 타겟 시스템 환경에 하이버네이션 기반 부팅 방식을 구현하기 위해 추가한 코드를 자세하게 설명한다. Ⅵ장에서는 실제 임베디드 리눅스 시스템 환경에서 하이버네이션 기반 부팅 과정을 분석하고 최적화 여지에 대해서 논의한다. Ⅶ장에서는 향후 연구 방향을 제시하면서 결론을 맺는다.

II. 관련 연구

임베디드 리눅스 시스템 환경에서 부팅 속도를 개선하려는 노력은 다양한 방면으로 진행되고 있다. 특히, 기존의 절전모드를 시스템 종료 상태로 간주하고 절전모드로부터 실행모드로 전환하는 과정을 부팅 과정으로 대응시켜 빠른 부팅을 실현하려는 연구들이 기존한다[1-5]. 이들 연구는 시스템 종료 상태로써 ACPI (Advanced Configuration Power Interface) 표준에 따른 절전모드인 S3과 S4 상태를 채택한다[6]. 참고로, 절전모드로 ACPI S3와 S4를 채택하는 절전 방식은 STR (Suspend-to-RAM)과 하이버네이션 (Hibernation, 또는 STD (Suspend-to-Disk))으로 각각 널리 알려져 있다. ACPI S3 상태는 CPU 레지스터 정보와 주변장치가 사용하는 버퍼 및 레지스터 정보 같은 시스템 상태 정보를 휘발성 메인메모리에 기록하고 메인메모리에만 최소한의 전력을 공급하는 대기전력 모드 상태이다. Bai와 Hsu는 ACPI S3 상태로부터 실행상태로 전환하는 시간이 매우 짧다는 것을 감안하여 홈 네트워크 서버를 종료하는 대신 ACPI S3 상태로 만들고 시스템의 부팅과정을 S3 상태로부터 복원하는 과정으로 대체한다[1]. 이때 기존의 S3 상태에서의 전력공급회로를 개선함으로써 대기전력 소비를 최소화하여 S3 상태가 에너지 효율측면에서 기존의 종료상태에 가까워질 수 있다고 주장한다.

최근 하이버네이션을 부팅에 활용하는 연구들은 임베디드 시스템 환경에서 증가하는 추세이다[2-5]. 최대절전모드로 잘 알려진 하이버네이션 기법은 STR처럼 시스템 상태를 메인메모리에 기록한 다음, 메인메모리 전체 내용을 비휘발성 저장매체에 저장하고 시스템에 전력 공급을 완전히 차단한다. 시스템 종료 후 전원이 재 인가되면 미리 저장되어 있는 시스템 상태로 복원하는 것으로 전통적인 부팅 과정을 대체한다. Kaminaga는 리눅스 버전 2.6.11을 탑재한 ARM 기반 OMAP 플랫폼 환경에서 하이버네이션을 구현하고 이를 개선한 스냇샵(Snapshot) 부팅 기법을 소개한다[2]. 그 이듬해에 스냅샷 부팅을 상용 제품에 적용한 사례가 소개되면서 하이버네이션 기반 부팅 방식이 주목받기 시작한다[3]. 최근 Jo 등은 디지털 TV에 하이버네이션 기반 부팅 방식을 적용한 사례 연구를 실시한 바 있다[4]. Baik 등은 스마트폰에 하이버네이션 기반 부팅 방식을 도입하여 부팅 속도를 개선하고자 하는 연구를 수행한 바 있다[5]. 이와 같이 하이버네이션 기반 부팅 방식은 임베디드 시스템 환경에서 주목받고 있다.

하이버네이션은 리눅스에서 Swsusp (Software Suspend) 또는 TuxOnIce라는 공개 프로젝트 형식으로 제공되기 시작하였다[7]. 최신 커널 버전은 Swsusp를 커널에 포함하여 x86 계열 시스템에 대해 하이버네이션을 제공한다. 리눅스는 하이버네이션을 위해서 x86 계열 시스템에 종속적인 코드 부분만을 제공하고 ARM 계열 시스템은 고려하지 않고 있어 ARM 시스템 환경에서는 하이버네이션이 지원되지 않는다.

ARM 계열 시스템에서 하이버네이션을 실행하기 위해서는 ARM 시스템 종속적인 코드 부분을 필요로 한다. 여기서 시스템 종속적인 코드란 시스템의 CPU 구조와 ISA (Instruction Set Architecture)에 밀접하게 연관되어 있는 코드를 의미한다. 예를 들어, ARM CPU와 x86 CPU는 서로 다른 종류의 레지스터와 ISA를 지원하기 때문에 이들 CPU 레지스터 정보를 저장하고 복원하는 코드는 CPU 계열에 의존적으로 작성되어야 한다. 기존 연구에서 OMAP 플랫폼과 리눅스 버전 2.6.11에 특화된 ARM 시스템 종속적인 하이버네이션 코드가 패치형태로 제공된다[8]. 불행히도 해당 패치를 수정 없이 사용하기 위해서는 해당 패치가 작성된 시스템 환경을 재현해야 하는데 이는 현실적으로 의미가 낮다. 따라서 해당 패치를 다른 시스템 환경으로 포팅하는 것은 불가피하다.

본 연구의 목적은 하이버네이션 기반 부팅 방식을 도입함에 따른 부팅 속도 개선의 여지를 확인하는 것이다. 이를 위해서는 임베디드 리눅스 시스템에서 하이버네이션 지원이 필수적임에도 불구하고 기존 연구들로부터 세부적인 구현 및 포팅에 관한 정보를 얻기가 쉽지 않다. 이에 본 연구에서는 먼저 ARM 환경에서 하이버네이션 기반 부팅 방식의 구현에 대해 상세하게 설명한다. 이를 통해 기존의 연구 결과들을 재현하고 개선하는데 기여할 수 있을 것으로 기대한다. 또한 하이버네이션 기반 부팅 과정을 관찰하고 이를 바탕으로 부팅 속도 개선의 여지에 대해서 논의한다는 점에서 기존 연구들과 차별된다.

III. 하이버네이션 개요

최신 버전의 리눅스에는 하이버네이션을 지원하기 위한 관련 코드가 포함되어 있다. 하이버네이션은 크게 서스펜드(Suspend)와 리줌(Resume)과정으로 구분된다. 서스펜드는 사용자 요청으로부터 현재의 시스템 상태를 비휘발성 매체에 저장하고 시스템 전원을 차단하기까지의 일련의 과정이다. 리줌은 시스템에 전원 공급이 재개됨에 따라 미리 저장되어 있던 종전의 시스템 상태를 복원하여 시스템 구동을 재개하는 과정이다.

하이버네이션의 서스펜드과정은 다음과 같다. 먼저, 사용자는 리눅스가 제공하는 특정 인터페이스를 통해서 서스펜드

를 명령한다. 리눅스 커널은 현재 동작중인 모든 프로세스를 실행상태에서 슬립(Sleep) 상태로 전환한다. 커널에 등록되어 있는 모든 주변장치들에 대해서 각 장치드라이버가 제공하는 장치 서스펜드과정을 수행한다. 이 때, 각 장치의 상태를 복원하기 위해서 필요한 장치 레지스터 내용이나 버퍼 내용을 메인메모리에 저장한다. 현재 상태의 CPU 레지스터와 필요하다면 코프로세서(Coprocessor) 레지스터 내용을 메인메모리에 저장한다. 메인메모리에 저장된 시스템 상태정보를 바탕으로 하이버네이션 이미지를 생성한다. 생성된 하이버네이션 이미지를 비휘발성 저장매체(일반적으로 스왑(Swap)영역)에 기록하고 시스템의 전원을 차단한다.

하이버네이션 이미지는 유효한 메인메모리 내용과 각 주변장치의 상태정보, CPU와 코프로세서의 레지스터 내용을 모두 포함하고 있다. 유효한 메인메모리 내용이란 커널의 실행이미지와 부팅 과정에서 할당된 커널의 동적 메모리 공간을 의미한다, 커널의 동적 메모리 공간에는 파일시스템이나 메모리 관리자와 같은 커널 구성요소들을 관리하기 위한 메모리 영역과 사용자 프로세스들의 코드, 스택, 힙 영역이 존재한다.

저장되어 있는 하이버네이션 이미지를 바탕으로 시스템의 상태를 복원하는 하이버네이션의 리줌과정은 다음과 같다. 먼저, 시스템에 전원이 인가되면 전통적인 부팅 절차를 수행한다. 만약 특정 비휘발성 저장장치에 하이버네이션 이미지가 기록되어 있음을 확인하면 리줌을 명령한다. 스왑영역으로부터 메인메모리로 하이버네이션 이미지를 모두 복사한다. 하이버네이션 이미지를 바탕으로 코프로세서 상태와 CPU 상태를 복원한다. 각 주변장치 드라이버가 제공하는 장치 리줌과정을 수행하여 각 장치를 종전의 상태로 복원한다. 중지된 모든 프로세스들을 실행상태로 복원하여 최종적으로 서스펜드를 수행하기 직전의 상태로 리줌을 완료한다.

본 논문에서는 “하이버네이션“이라는 용어와 ”하이버네이션 기반 부팅“이라는 용어를 구분하여 사용하며 그 차이점은 다음과 같다. 하이버네이션은 매번 서스펜드를 수행하여 하이버네이션 이미지를 저장하고 해당 이미지를 바탕으로 리줌을 수행한다. 이 때문에 하이버네이션은 시스템을 매번 서스펜드 직전의 시점으로 구동된다. 이와 달리, 하이버네이션 기반 부팅 방식은 부팅 완료 시점이라고 지정된 특정 시점에서 한번만 서스펜드를 수행하여 하이버네이션 이미지를 생성하고 매번 같은 하이버네이션 이미지에 근거해서 리줌을 수행한다. 따라서 하이버네이션 기반 부팅 방식을 사용하면 시스템은 매번 동일한 부팅 시점으로 구동될 수 있다.

IV. 하이버네이션 동작 분석

하이버네이션 기반 부팅 방식을 구현하기 위해서는 기존하는 하이버네이션 코드에 대한 이해가 선행되어야 한다. 이에 본 장에서는 리눅스 2.6.21 커널에 포함되어 있는 기존의 하이버네이션 코드를 상세하게 분석한다.

1. 사용자 인터페이스

리눅스는 응용 계층과 커널 계층 간의 효율적인 인터페이스를 지원하기 위해서 SYS 파일시스템을 제공한다[10]. 이 SYS 파일시스템을 통해 응용 계층에서 서스펜드를 요청할 수 있다. 이와 관련하여, 커널 소스 파일 kernel/power/main.c에서는 매크로 decl_subsys(power, NULL)와 power_attr(state)를 사용하여 전역변수 power_subsys와 attr_group을 선언한다. 전역변수 power_subsys는 SYS 파일시스템의 하위 디렉터리 /sys/power/ 생성에 필요한 정보를 포함한다. 변수 attr_group는 SYS 파일시스템의 하위 파일 /sys/power/state를 생성하고 해당 파일에 접근할 경우 호출되는 함수에 대한 정보를 담고 있다. 리눅스 부팅과정에서 수행되는 초기화 함수 pm_init()에서는 전역변수 power_subsys와 attr_group를 인자 값으로 해서 함수 subsystem_register()와 sysfs_create_group()를 호출한다. 그 결과, 파일 /sys/power/state가 생성된다. 또한 해당 파일에 읽기 연산을 수행할 경우 함수 state_show()를 호출하고 쓰기 연산에 대해서는 state_store()를 호출하도록 지정한다. 이로써 하이버네이션을 위한 인터페이스가 구축된다.

하이버네이션의 서스펜드는 SYS 파일시스템 아래에 존재하는 파일 /sys/power/state에 문자열 “disk“를 기록함으로써 요청된다. 플래시 메모리의 특정 파티션(본 연구에서는 mtdblock2)을 하이버네이션 이미지 저장을 위한 스왑영역으로 사용할 경우, 다음과 같은 일련의 명령어를 리눅스 쉘 프롬프트 (Shell prompt)상에 입력함으로써 서스펜드를 요청한다.

```
# mkswap /dev/mtdblock2
# swapon /dev/mtdblock2
# echo disk > /sys/power/state
```

파일 /sys/power/state에 문자열 “disk“를 쓰면 커널 내부 함수 state_store()가 호출된다. 함수 state_store()는 인자로 받은 문자열 “disk“를 바탕으로 사용자가 다양한 전력관리 기법 중에서 하이버네이션을 지정하였음을 확인하고 함수 enter_state()를 호출한다.

2. 서스펜드(Suspend) 과정

함수 enter_state()가 호출하는 서스펜드 메인 함수인 함

수 pm_suspend_disk()의 동작은 크게 두 단계로 나뉜다. 먼저 하이버네이션 이미지를 생성하기 전 준비 단계로써, 프로세스와 콘솔을 비롯한 주변장치의 상태를 안정화하는 동시에 충분한 용량의 메인메모리 공간을 확보한다. 그 다음, 하이버네이션 이미지를 생성하고 이를 스왑공간에 저장한 후에 최종적으로 시스템 전원을 차단한다. 그림 1은 하이버네이션 서스펜드와 리줌과정에서 호출되는 주요 함수들에 대한 함수 호출 그래프이다. 각 함수들은 시스템 독립적인 코드와 시스템 종속적인 코드부분으로 구분되어 있다. 시스템 독립적인 코드들은 대다수가 커널 소스 디렉터리 kernel/power/ 아래의 파일들에 포함되어 있으며, 시스템 종속적인 코드는 커널 소스 디렉터리 arch/arm/power 아래에 추가로 구현된다. 특별한 언급이 없는 경우 함수들은 시스템 독립적으로 동작한다. 이제 함수 pm_suspend_disk()가 호출하는 주요 함수들에 대해서 살펴보도록 한다.

첫째, 하이버네이션 이미지를 생성하기 전 준비 단계에서 함수 pm_suspend_disk()는 주요 함수들 prepare_process(), swsusp_shrink_memory(), suspend_console(), 그리고 함수device_suspend()를 차례로 호출한다. 함수 prepare_ processes()는 서스펜드를 시작하고 리줌이 완료될 때까지 하나의 실행 흐름만이 CPU에서 실행되도록 한다. 함수 prepare_ processes()는 함수 freeze_process()를 호출하여 모든 태스크(Task)의 상태를 슬립 상태로 만드는데 이때 각 태스크는 인터럽트를 수신하지 못하도록 설정된다. 함수 freeze_process()를 수행하는 이유는 하이버네이션이 수행되는 동안 멀티태스킹 환경이 지속될 경우 예측하기 힘든 오류상황이 유발될 수 있기 때문에 이를 방지하고자 함이다.

함수 swsusp_shrink_memory()는 서스펜드를 수행하는 과정에서 요구되는 메인메모리 공간을 확보하기 위해서 리눅스의 메모리 관리 시스템이 제공하는 메모리 회수 메커니즘을 구동한다. 함수 suspend_console()은 락킹(Locking) 기법을 사용하여 콘솔(Console)을 통해서 더 이상 메시지가 출력되지 않도록 만든다. 함수 device_suspend()는 여러 주변장치들의 현재 상태를 저장하는 역할을 담당한다. 각 주변장치들을 구동하는 장치 드라이버는 일반적으로 해당 장치에 특화된 서스펜드 함수를 제공하는데 함수 device_suspend()는 모든 장치들에 대해서 이들 함수를 호출함으로써 각 장치들의 현재 상태를 메인메모리 공간에 저장할 수 있다.

함수 pm_suspend_disk()는 하이버네이션 이미지를 생성, 저장하고 서스펜드를 완료하기까지 함수 swsusp_suspend(), swsusp_write()와 함수 power_down()을 차례로 호출한다. 함수 swsusp_suspend()는 현재 시스템 상태 정보를 토대로 하이버네이션 이미지를 생성한다. 함수 swsusp_suspend()는 먼저 현재의 프로세서 상태를 메인메모리에 저장한다. 이 역할은 함수 save_processor_state()와 swsusp_arch_suspend()가 담당한다. 이들은 각각 코프로세서와 CPU 레지스터 값을 메인메모리로 복사한다. 이를 위해 당연히 CPU 종류나 칩셋에 특화된 시스템에 종속적인 코드들을 포함해야 하는데 관련 내용은 V장에서 자세하게 다룬다.

함수 swsusp_save()는 메인메모리 내용을 토대로 하이버네이션 이미지를 생성한다. 이미 주변 장치들의 상태 정보가 메인메모리에 기록되어 있음을 감안할 때, 프로세서의 상태 정보가 복사 완료되는 현 시점에서 메인메모리는 시스템 상태를 결정짓는 모든 정보를 담고 있다. 함수 swsusp_save()는 함수 swsusp_alloc()을 통해서 하이버네이션 이미지를 저장할 공간을 마련하고 함수 copy_data_pages()를 통해서 유

```
pm_suspend_disk()
    prepare_processes()
        freeze_processes()
    swsusp_shrink_memory()
    suspend_console()
    device_suspend()
    swsusp_suspend()
        save_processor_state()
        swsusp_arch_suspend()
            swsusp_save()
                swsusp_alloc()
                copy_data_pages()
    swsusp_write()
        save_image()
    power_down()
```

(a) 서스펜드 (Suspend)

```
software_resume()
    swsusp_check()
    prepare_processes()
        freeze_processes()
    swsusp_read()
        load_image()
    suspend_console()
    device_suspend()
    swsusp_resume()
        swsusp_arch_resume()
        restore_processor_state()
    device_resume()
    resume_console()
    unprepare_processes()
```

(b) 리줌 (Resume)

그림 1. 하이버네이션 함수 호출 그래프
Fig. 1. Function call graph for Hibernation

효한 메인메모리 내용을 하이버네이션 이미지를 위해 할당된 공간으로 복사한다. 함수 swsusp_write()는 함수 save_image()를 호출함으로써 생성된 하이버네이션 이미지를 스왑 공간으로 저장한다. 마지막으로 함수 power_down()을 수행함으로써 시스템을 종료상태로 전환한다.

3. 리줌(Resume) 과정

리줌을 위해서 리눅스는 전통적인 부팅 과정을 수행하다가 /etc/rc.d/rc.sysinit에 명세되어 있는 초기화 스크립트를 수행하기 직전에, 매크로 late_initcall()로 선언되어 있는 커널 초기화 함수 software_resume()을 호출한다. 해당 함수의 수행은 크게 두 단계로 구분된다. 먼저 하이버네이션 이미지를 메인메모리로 읽어 들이는 단계이다. 이 단계에서는 리줌과정 동안 시스템의 안정성을 높이기 위해 서스펜드의 초기 과정에서도 수행되었던 프로세스와 콘솔, 그리고 주변장치의 상태를 안정화하는 작업을 수행한다. 이어서 하이버네이션 이미지에 저장되어 있는 시스템 상태정보를 바탕으로 시스템을 서스펜드 직전의 상태로 복원한다. 이제 함수 software _resume()이 호출하는 주요 함수들에 대해서 살펴보고자 한다.

함수 software_resume()은 먼저 함수 swsusp_check()를 호출하여 하이버네이션 이미지가 스왑영역에 저장되어 있는지 확인한다. 하이버네이션 이미지가 저장되어 있는 장치명은 부트로더가 “resume=/dev/mtdblock2″라는 인자 값으로 커널에 전달한다. 함수 swsusp_ check()는 스왑영역의 첫 페이지를 읽어서 그곳에 문자열 ″S1SUSPEND″가 적혀 있는지 확인한다. 이는 하이버네이션 이미지가 저장되어 있는가를 결정하는 매직넘버(Magic number) 값이다. 매직 넘버 값을 발견하면 리줌과정을 수행한다. 이후 리줌과정 동안 시스템의 여러 태스크들을 안정적인 상태로 유지하기 위해서 함수 prepare_processes()를 수행한다. 그리고 함수 swsusp_read()를 호출한 후에 함수 suspend_console()와 device_suspend()를 호출한다. 함수 prepare_processes(), suspend_console(), device_suspend()는 서스펜드 초기 단계에서도 수행된 바 있는 함수들로써 이전 절에서 이미 다룬바 있다. 함수 swsusp_read()는 load_ image()를 호출함으로써 스왑영역에 저장되어 있는 하이버네이션 이미지를 메인메모리 영역으로 복사한다.

함수 swsusp_resume()은 메인메모리로 복사된 하이버네이션 이미지를 바탕으로 시스템 상태를 서스펜드 직전의 상태로 복원한다. 먼저 ARM 시스템 종속적인 함수 swsusp_arch_ resume()를 통해서 메인메모리 내용과 CPU 레지스터 내용을 서스펜드를 수행하던 당시의 상태로 만든다. 엄밀히 말해서, 함수 swsusp_arch_resume()가 수행 완료되어 반환되는 순간 CPU 제어권은 서스펜드과정에서 호출된 함수 swsusp_arch_ suspend()가 반환되는 시점으로 돌아간다. 이는 함수 swsusp_ arch_resume()에서 함수의 반환 주소값(Return address)인 CPU 레지스터 LR 값을 하이버네이션 이미지에 저장되어 있던 함수 swsusp_arch_suspend()에 대한 반환 주소값으로 덮어쓰기 때문이다. 이 시점부터 시스템의 CPU 제어권은 서스펜드과정에서 수행되었던 함수 swsusp_suspend()로 전환되어 해당 코드를 수행하게 된다. 함수 swsusp_suspend()는 함수 restore_processor_state()를 호출하여 코프로세서 레지스터들을 복구한다. 이어서 함수 device_resume()과 resume_ console()을 호출하여 주변장치와 콘솔 장치를 복구한다. 마지막으로 서스펜드 초기단계에서 수행되었던 함수 prepare_ process()에 대응되는 함수 unprepare_process()를 호출함으로써 슬립 상태의 태스크들을 실행 상태로 복원한다. 이와 같은 리줌과정을 통해서 시스템 상태는 서스펜드를 요청하기 직전의 상태로 복구된다.

V. 하이버네이션 기반 부팅 구현

본 연구에서 고려한 타겟(Target) 시스템은 리눅스 2.6.21을 운용하는 ARM 임베디드 시스템 개발보드 환경이다[9]. 타겟 시스템에 탑재된 프로세서는 네비게이션이나 휴대폰과 같은 이동식 전자기기에 널리 채택되고 있는 Marvell사의 XScale PXA270 프로세서이다. 메인메모리로 64MB 용량의 SDRAM을 사용하고 저장매체로는 최대 1GB 낸드 플래시 메모리를 장착하고 있다. 타겟 보드는 자체적으로 사운드와 LCD와 같은 다양한 주변장치를 지원하므로 실제 이동형 전자기기 제품에 대한 개발 참조(Reference) 보드로써 활용 가치가 충분하다.

본 연구에서는 타겟 시스템 환경에 하이버네이션 기반 부팅 방식을 구현하기 위해 다음의 두 가지 측면에서 리눅스 커널을 수정한다. 첫째, 타겟 시스템 환경에 하이버네이션의 서스펜드와 리줌을 지원하기 위해서 시스템 종속적인 코드 부분에 대한 포팅 작업을 수행한다. 둘째, 하이버네이션 기반 부팅을 지원하기 위해서 시스템 독립적인 코드 부분을 일부 수정한다.

1. 시스템 종속적인 함수 포팅

타겟 시스템의 메인 칩셋인 ARM XScale PXA270 칩셋에 종속적인 하이버네이션 함수로써 서스펜드과정에서 호출

되는 함수 save_processor_state()와 이에 대응되는 리줌 함수인 restore_processor_state()를 구현한다. 이들 함수는 XScale 계열 칩셋 내부에서 가상 메모리와 CPU 캐시 제어를 담당하는 코프로세서 CP15 내부의 레지스터 값을 각각 저장하고 복원하는 역할을 수행한다. 해당 함수는 커널 소스 코드 파일 arch/arm/power/cpu.c에 구현된다. 서스펜드를 위한 함수인 save_processor_state()는 CP15 레지스터들 중에서 읽기와 쓰기가 모두 허용되는 총 11개의 레지스터 값을 CP15로부터 읽어서 메인메모리 공간에 저장한다. 참고로 XScale 코어 개발자 지침서에 CP15 레지스터들에 관한 정보들이 상세히 명세되어 있다[11]. 리줌과정에서 수행되는 restore_processor_ state()는 하이버네이션 이미지에 저장되어 있는 CP15 레지스터 값들을 모두 복원한다. 이 함수가 수행되고 나면 CP15의 상태, 즉 가상 메모리와 CPU 캐시 관리 정보들이 모두 함수 save_processor_state()를 수행했던 시점의 상태로 복원된다.

또 다른 ARM CPU에 종속적인 하이버네이션 함수 swsusp_ arch_suspend()와 swsusp_arch_resume()의 구현은 다음과 같다. 이들 함수는 리눅스 2.6.11 커널 소스 파일 arch/arm/ power/swsusp.S에 구현된다. 함수 swsusp_arch_suspend()는 ARM CPU의 모든 레지스터 값을 미리 할당되어 있는 특정 메모리 공간에 저장한다. ARM 계열 CPU에서 가용한 총 36개의 레지스터들은 함수를 수행중인 CPU 모드에서 접근 가능한 17개 레지스터와 현재의 CPU 모드에서 접근 불가능한 19개의 뱅크 레지스터(Banked register)로 구성된다[12]. 뱅크 레지스터는 FIQ(Fast interrupt request) 모드, IRQ(Interrupt request) 모드, Supervisor 모드, Undefined 모드, Abort 모드에서만 각각 접근 가능한 레지스터를 의미한다. 이들 CPU 모드는 차례로 8개, 3개, 3개, 3개, 3개의 뱅크 레지스터를 제공한다. 따라서 함수 swsusp_arch_suspend()는 현재 수행중인 CPU 모드에서 접근 가능한 모든 레지스터 값을 메인메모리에 저장한 다음 5가지의 CPU 모드로 각각 전환하면서 뱅크 레지스터 값들을 저장한다.

함수 swsusp_arch_suspend()에 대응되는 리줌 함수인 swsusp_arch_resume()는 먼저 메인메모리에 존재하는 하이버네이션 이미지를 바탕으로 서스펜드 시점의 메인메모리 상태로 복원한다. 하이버네이션 이미지는 메인메모리 페이지 내용과 메타데이터 정보로 구성된다. 메타데이터는 각 유효 페이지에 대해서 해당 페이지가 서스펜드 당시에 실제 위치했던 주소정보와 하이버네이션 이미지 내에서의 주소정보를 담고 있다. 이들 메타데이터 단방향 연결 리스트로 연결되어 있으며 포인터 변수 restore_pblist를 통해서 접근 가능하다. 그림 2는 커널 버전 2.6.21을 위해서 수정된 핵심 코드 내용을 포함한다. 해당 코드 부분은 ARM 어셈블리어로 작성되어 있으며, 코드의 설명은 다음과 같다. 먼저 코드 라인 1부터 5는 변수 restore_pblist를 통해서 유효 페이지에 대한 하이버네이션 이미지 내의 주소 값과 해당 페이지가 복사될 목적지 주소 값을 얻어오는 코드이다. 라인 6부터 12까지에서 페이지 크기인 4KB 단위로 하이버네이션 이미지를 메인메모리로 복사한다. 라인 13부터 15의 코드는 이러한 작업을 하이버네이션 이미지내의 모든 유효 페이지에 대해서 수행하도록 만든다. 이로써 하이버네이션 이미지로부터 서스펜드 시점의 메인메모리 내용을 복원하는 것이 가능하다. 메인메모리 상태 복원을 완료한 후에 함수 swsusp_ arch_suspend()에 의해서 저장되었던 총 36개의 CPU 레지스터 값들을 현재의 CPU 레지스터에 덮어씀으로써 CPU 상태를 복원한다.

2. 하이버네이션 기반 부팅 지원

대기전력 절감을 목적으로 하는 하이버네이션은 시스템이 유휴 상태가 되면 서스펜드를 수행하여 시스템을 종료하고, 서스펜드과정에서 생성된 하이버네이션 이미지를 바탕으로 리줌을 수행하여 시스템을 서스펜드 시점의 상태로 재구동한다. 매번 종전의 서스펜드 상태로 시스템 상태를 복원하기 위해서 리줌과정에서 수행되는 함수 resume_check()는 스왑영역에 존재하는 하이버네이션 이미지가 재사용되지 않도록 구현되어 있다. 반면 하이버네이션 기반 부팅 방식은 부팅 시점이라고 합의된 특정 시점의 시스템 상태에 대해서 하이버네이션 이미지를 생성하여 스왑영역에 저장하고 전원이 인가되면 항상 해당 부팅 시점으로 복원함으로써 시스템을 부팅하는 기법으로 알려져 있다.

```
1:                      ldr     r1, .Lrestore_pblist
2:                      ldr     r6, [r1]
3: .Lcopy_loop:         ldr     r4, [r6]
4:                      ldr     r5, [r6, #4]
5:                      mov     r9, #1024
6: .Lcopy_one_page:     ldr     r8, [r4]
7:                      str     r8, [r5]
8:                      add     r4, r4, #4
9:                      add     r5, r5, #4
10:                     sub     r9, r9, #1
11:                     cmp     r9, #0
12:                     bne     .Lcopy_one_page
13:                     ldr     r6, [r6, #8]
14:                     cmp     r6, #0
15:                     bne     .Lcopy_loop
```

그림 2. 하이버네이션 이미지 적재를 위한 어셈블리 코드
Fig. 2. Assembly code for loading the Hibernation Image

본 연구는 하이버네이션 기반 부팅 방식을 지원하기 위해서 스왑영역에 존재하는 하이버네이션 이미지를 무효화하는 부분을 제거한다. 구체적으로 함수 resume_check()는 함수 bio_write_page()를 통해서 스왑영역의 첫 페이지에 기록되어 있는 매직넘버 값 S1SWSUSP를 소거하여 하이버네이션 이미지를 무효화한다. 해당 코드 부분을 제거함으로써 시스템 구동 시 매번 같은 하이버네이션 이미지로부터 리줌하여 동일한 시스템 상태, 즉 지정된 부팅 상태로 부팅할 수 있다. 이와 같이 하이버네이션 기반 부팅을 지원하기 위해 커널에서 직접 수정되는 부분은 미미하다. 이는 실제 상용화 가능성 측면에서 본다면 아주 매력적인 부분으로 작용될 수 있다.

VI. 결과 분석 및 논의

본 절에서는 실제 임베디드 리눅스 시스템 환경에 하이버네이션 기반 부팅 방식을 적용할 경우 시스템 부팅 과정에서 소요되는 시간을 관찰한다. 이를 바탕으로 하이버네이션 기반 부팅 방식을 개선할 수 있는 여지에 대해 살펴본다.

부팅 과정을 시간대별로 모니터링하기 위해서 본 연구에서는 Grabserial을 사용한다[13]. Grabserial은 타겟 시스템이 아닌 호스트 시스템에서 동작하므로 설치 및 구동이 간편하고 타겟 시스템의 성능저하를 유발하지 않기 때문에 본 연구에서는 이를 채택하였다.

본 연구에서는 리눅스의 로그인 메시지가 출력되는 순간을 부팅이 완료된 시점으로 간주한다. 전통적인 부팅 방식과 하이버네이션 기반 부팅 방식으로 타겟 시스템을 부팅 완료하는데 소요된 시간은 표 1에 요약되어 있다. 하이버네이션 기반 부팅 방식은 전통적인 부팅에 비해 2.166531초 더 빠르다. 표 1에서 전통적인 부팅 과정은 크게 부트로더 수행, 커널 적재(Loading), 커널 실행 이미지 압축 해제, 커널 초기화, 그리고 초기(Init) 스크립트 수행의 5단계로 구분된다. 하이버네이션 기반 부팅은 전통적인 부팅과정과 마찬가지로 부트로더 수행, 커널 적재 및 압축 해제과정을 수행하고 대부분의 커널 초기화 과정을 완료한 다음 리줌과정을 시작한다. 리줌 과정은 하이버네이션 이미지를 메인메모리로 적재하는 단계와 리줌 함수를 수행하는 단계로 나뉜다.

하이버네이션 기반 부팅은 크게 세 가지 측면에서 개선의 여지가 있다. 먼저, 하이버네이션 기반 부팅은 전통적인 부팅 과정을 대부분 수행한 시점에서 리줌을 호출한다. 리줌이 수행되면 앞서 전통적인 부팅 과정을 통해서 만들어진 시스템 상태, 즉 메인메모리 내용과 CPU 레지스터 정보 등은 하이버네이션 이미지에 저장되어 있는 시스템 상태로 덮어 써지므로 그 의미를 상실한다. 따라서 리줌이 호출되기 전에 수행되는 작업을 최소화함으로써 부팅 속도를 극대화할 수 있다. 리눅스 커널의 리줌 함수 swsusp_resume()을 수행할 수 있는 최소한의 여건을 만들어 주는 것이 리줌 함수 호출 이전에 해야 할 필수적인 작업이다. 이 작업에는 CPU, 코프로세서, 메인메모리, 그리고 저장매체에 대한 초기화 등이 포함된다. 이를 고려할 때, 가능한 최적의 리줌 시점으로는 부트로더의 장치 초기화 완료 직후 시점이나 커널 적재 이후 가상 메모리가 활성화된 직후 시점이 될 수 있다. 이와 같은 최적화 기법을 적용한다면 기존 하이버네이션 기반 부팅에서 커널 초기화에 소요되는 2.93초를 줄일 수 있어 총 부팅 시간은 12.4초 정도로 단축될 수 있다.

둘째, 하이버네이션 기반 부팅과정에서 저장매체에 있는 커널 실행 이미지는 두 차례 메인메모리로 복사된다. 하이버네이션 이미지에 커널 실행 이미지가 포함되어 있음을 고려할 때, 커널 실행 이미지는 리줌을 호출하기 전의 커널 적재과정에서 한번 복사되고 하이버네이션 이미지를 복사할 때 한 차례 더 복사된다. 만약 부트로더에서 커널 실행 이미지를 적재하고 압축을 해제하는 대신 하이버네이션 이미지를 앞당겨서 적재한다면 이 중복 적재 문제를 해결할 수 있다. 이 최적화 기법을 적용한다면 커널의 실행 이미지 적재와 압축해제에 소요되는 3.88초와 0.94초 정도의 시간을 각각 추가로 절약할 수 있다. 이를 통해 전통적인 부팅 방식 대비 2.3배 빠른 7.58초 정도의 시간에 타겟 시스템이 부팅 가능할 것으로 판단된다.

표 1. 부팅 시간 비교 (단위: 초)
Table 1. Comparisons of booting time delay (Unit: second)

전통적인 부팅 방식	부트로더	커널 적재	압축 해제	커널 초기화	초기 스크립트 수행		총 부팅시간
	0.748410	3.885154	0.945986	2.931110	8.995262		17.505922
하이버네이션 기반 부팅 방식	부트로더	커널 적재	압축 해제	커널 초기화	하이버네이션 이미지 적재	리줌 함수 수행	총 부팅시간
	0.749402	3.882975	0.945871	2.931381	5.940772	0.888990	15.339391

마지막으로, 하이버네이션 이미지의 크기를 줄여 하이버네이션 이미지 적재시간을 단축할 수 있다. 표 1에서 11.12MB의 하이버네이션 이미지는 1.87MB/s의 입출력 속도로 메모리에 적재된다. 리눅스 커널에 포함되어 있는 커널 XIP (eXecute In Place) 기능을 사용하면 커널 실행 이미지가 비휘발성 저장매체에서 직접 실행되므로 커널 실행 이미지는 하이버네이션 이미지에서 제외된다[14]. 압축되기 전 커널 실행 이미지가 3.6MB임을 감안한다면 커널 XIP를 통해서 하이버네이션 이미지는 7.52MB로 줄어들 것이다. 따라서 하이버네이션 이미지를 적재하는데 4.01초 정도 소요한다. 만약 하이버네이션 기반 부팅에서 리줌 시점을 최대한 앞당기고 커널 XIP 기법을 도입하면 최대 5.65초 정도 만에 타겟 시스템 부팅이 가능할 것으로 분석된다. 이와 같이, 타겟 시스템에 다양한 최적화 기법이 채택된 하이버네이션 기반 부팅 방식을 적용한다면 전통적인 부팅 방식보다 최대 3.1배 정도 빠른 부팅이 가능할 것으로 기대한다.

VII. 결 론

본 연구에서는 실제 임베디드 시스템 환경에서 하이버네이션 기반 부팅의 실현 가능성을 중점적으로 살펴보았다. 이를 위해 리눅스 2.6.21을 운용하는 ARM 임베디드 개발보드 환경에 하이버네이션 기반 부팅 방식을 구현하였다. 또한 타겟 시스템 환경에서 하이버네이션 기반 부팅 과정을 관찰하고 부팅 속도 개선의 여지에 대해서 논의하였다. 본 논문은 하이버네이션 기반 부팅에 관한 연구의 시작점을 제시하고 다양한 이슈들을 공유한다는 측면에서 그 의의를 찾을 수 있다.

향후 연구로는 본 연구에서 제시된 다양한 최적화 기법들을 적용하여 부팅 속도가 개선된 하이버네이션 기반 부팅 방식을 실제 구현하는 것이다. 또한 다양한 최신의 플랫폼에 적용하여 실제 상용화 가능성을 살펴보고자 한다. 그리고 본 연구에서 제시하는 다양한 최적화 기법들이 야기할 수 있는 부작용들을 파악하고 이를 해결해 나가고자 한다.

참고문헌

[1] E. Bai and H. Hsu, "Design and Implementation of an Instantaneous Turning-on Mechanism for PCs," IEEE Trans. on Consumer Electronics, Vol. 53, No. 4, pp. 1595-1601, Nov. 2007,

[2] H. Kaminaga, "Improving Linux Startup Time Using Software Resume," In Proc. of the Linux Symposium, 2006.

[3] V. Wool, "Linux Suspend-To-Disk Objectives for Consumer Electronic Devices," In Proc. of the Embedded Linux Conference, 2007.

[4] H. Jo, H. Kim, H. Roh, and J. Lee, "Improving the Startup Time of Digital TV," IEEE Trans. on Consumer Electronics, Vol. 55, No. 2, pp. 721-727, May 2009,

[5] K. Baik, S. Woo, S. Kim, and J. Choi, "Boosting up Embedded Linux device," In Proc. of the Linux Symposium, 2010.

[6] ACPI, http://www.acpi.info/spec.htm

[7] TuxOnIce, http://www.tuxonice.net

[8] SuspendToDiskForARM,http://tree.celinuxforum.org/CelfPubWiki/SuspendToDiskForARM

[9] Falinux, http://kshop.falinux.com/UsePage/tvell/Goods/GoodsDetail.php?goo_id=6457

[10] W. Mauerer, "Professional Linux Kernel Architecture," Wiley, pp. 689-706, 2008.

[11] Intel® XScaleTM Core Developer's Manual, http://www.intel.com/design/intelxscale/273473.htm

[12] A. Sloss, "ARM System Developer's Guide," Morgan Kaufmann, pp. 19-43, 2004.

[13] T. Bird, "Tools and Techniques for Reducing Bootup Time," In Proc. of the Embedded Linux Conference Europe, 2008.

[14] J. Hulbert, "Introducing the Advanced XIP File System," In Proc. of the Linux Symposium, 2008.

저 자 소 개

도 인 환

2003: 홍익대학교 컴퓨터공학과 공학사.
2006: 홍익대학교 컴퓨터공학과 공학석사.
2010: 홍익대학교 컴퓨터공학과 공학박사
현 재: (주)프롬나이 책임연구원 (연구개발팀장)
관심분야: 컴퓨터구조, 임베디드 시스템, 차세대메모리,운영체제, 파일시스템
Email : ihdoh@peromnii.com