Smart Media Journal / Vol.11,No.11 / ISSN:2287-1322
https://dx.doi.org/10.30693/SMJ.2022.11.11.84

베이스보드 매니지먼트 컨트롤러를 위한 부팅 과정 프로파일링 도구

(Booting Process Profiling Tool for Baseboard Management Controllers)

김재섭*, 박민호**, 홍지만***

(Jaeseop Kim, Minho Park, Jiman Hong)

요 약

베이스보드 매니지먼트 컨트롤러(BMC, Baseboard Management Controller)는 다양한 통신 인터페이스를 사용하여 서버 모니터링, 유지보수, 제어 기능을 지원한다. 그러나, 디바이스 드라이버 초기화 과정에서 예기치 못한 문제가 발생할 경우 BMC가 정상적으로 동작하지 않을 수 있기 때문에 디바이스 드라이버 초기화 과정을 정확하게 분석하고, 분석 결과를 확인할 수 있는 기능을 제공하는 부팅 과정 프로파일링 도구는 필수적이다.

기존 부팅 과정 프로파일링 도구들은 BMC 부팅 과정 분석에 필요한 디바이스 드라이버 초기화 과정과 결과를 구체적으로 제공하지 않아 개발자가 필요에 따라 여러 도구를 조합하여 사용해야 하는 불편함이 있다. 본 논문에서는 BMC의 부팅 과정 프로파일링 도구를 제안한다. 제안하는 도구는 디바이스 드라이버 초기화 과정 분석, CPU 및 메모리 사용률 분석, 커널 버전 관리 기능을 제공한다. 제안하는 도구를 사용하여 부팅 과정을 쉽게 분석할 수 있으며, 분석 결과는 부팅 시간 단축에 사용될 수 있다. 또한 제안한 도구를 Linux 기반의 BMC에 구현하고, 제안한 도구가 기존 프로파일링 도구에 비해 효율적임을 보인다.

■ 중심어 : 베이스보드 매니지먼트 컨트롤러 ; 프로파일링 ; 리눅스

Abstract

Baseboard Management Controller(BMC) supports server monitoring, maintenance, and control functions using various communication interfaces. However, if an unexpected problem occurs during the device driver initialization process, the BMC may not operate normally. Therefore, a boot process profiling tool that accurately analyzes the device driver initialization process and provides a function to check the analysis result is essential.

Existing boot process profiling tools do not specifically provide the device driver initialization process and results required for BMC boot process analysis, forcing developers to use a combination of tools to analyze the boot process in detail.

In this paper, we propose an integrated profiling tool for BMC's booting process. The proposed tool provides device driver initialization process analysis, CPU and memory usage analysis, and kernel version management functions. Users can easily analyze the booting process using the proposed tool, and the analysis result can be used to shorten the booting time. Also, the proposed tool is implemented in Linux-based BMC, and it is shown that the proposed tool is more efficient than the existing profiling tool.

■ keywords : Baseboard Management Controller ; Profiling ; Linux

Ⅰ. 서 론

베이스보드 매니지먼트 컨트롤러(BMC, Baseboard Management Controller)는 서버 재부팅, 운영체제 재설치, 팬 속도/온도 모니터링과 같이 데이터 센터를 물리적으로 방문해야 하

* 학생회원, 숭실대학교 컴퓨터학과
** 학생회원, 숭실대학교 컴퓨터학과
** 종신회원, 숭실대학교 컴퓨터학과
이 논문은 2022년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임(No. 2022-0-00202, 컴퓨팅 서버의 운영 전력 절감을 위한 지능형 메인보드 제어 소프트웨어 기술).

접수일자 : 2022년 11월 29일 게재확정일 : 2022년 12월 16일
교신저자 : 홍지만 e-mail : jiman@ssu.ac.kr

는 작업을 원격으로 수행할 수 있는 기능을 지원한다[1]. 이러한 기능을 제공하기 위해, BMC는 Inter-Integrated Circuit(I2C), Peripheral Component Interconnect Express(PCIE) 와 같은 통신 인터페이스를 사용하여 센서, 팬으로부터 서버 하드웨어 정보를 수집하고, 호스트보드를 제어한다[2,3]. 부팅 과정 중, 통신 인터페이스, 센서, 팬과 같은 하드웨어의 디바이스 드라이버 초기화 과정에서 문제 발생 시, BMC의 기능을 수행하지 못할 수 있다. 따라서, 부팅 과정에서 디바이스 드라이버 초기화 과정과 결과를 추적할 수 있는 프로파일링 도구가 필요하다.

부팅 과정을 분석하기 위해 기존에 많은 부팅 과정 프로파일링 도구가 제안되었다. 그러나, BMC 부팅 과정 프로파일링 도구에 필요한 기능을 제공하는 도구에 대한 기존 연구가 부족하다.

A. van de ven은 커널 영역 부팅 과정에서 호출되는 initcall 함수의 실행 시간을 수집 및 분석하고 결과를 그래프 형태로 제공하는 Bootgraph를 제안하였다[4]. initcall이란 디바이스 드라이버 및 파일 시스템을 올바른 순서로 초기화시키기 위한 메커니즘이다[5]. Bootgraph의 분석 결과에는 initcall 함수 수행 시간 및 함수명이 제공된다. 그러나, 자세한 함수 호출 추적이 제공되지 않아 자세한 분석에 사용하기에는 무리가 있다. 또한, 분석 결과는 initcall 함수 수행 시간을 블록으로 표현하고, 블록 내부에 함수명을 제공하는 형태로, 함수 수행 시간이 짧은 경우 함수명을 파악하기 어렵다는 단점이 있다.

A. van de ven 등은 시스템 정보와 Init Process 생성 이후부터, CPU 로드, 디스크 로드, 메모리 사용량, 데몬 및 프로세스 정보를 제공하는 Bootchart를 제안하였다[6]. Init Process 생성 이후 분석을 시작하기 때문에, 커널 부팅 과정의 디바이스 드라이버 초기화 과정은 분석할 수 없다. 따라서, 디바이스 드라이버 초기화 과정 분석이 중요한 BMC의 프로파일링 도구로는 적합하지 않다.

S. Rostedt 등은 커널 영역 부팅 시작 시점부터 시스템 콜, 스케줄링, 인터럽트에 대한 분석을 지원하는 Ftrace를 제안하였다[7]. Ftrace는 커널 부팅 영역을 포함하여, 부팅 이후까지 분석을 지원한다. 또한, 함수 필터링 기능이 포함되어 함수 호출 추적, 함수 수행 시간, 함수가 수행된 CPU 코어에 대한 정보를 제공한다. 그러나 분석 결과에 대한 통계화, 시각화 기능을 제공하지 않아, 도구의 프로파일링 결과에 대한 분석이 어렵다는 단점이 있다.

S. Peeters 등은 Init Process 생성 이후부터 초기화되는 서비스를 추적하고 분석을 지원하는 systemd-analyze를 제안하였다[8]. 추적한 서비스에 대해 소요된 시간 순으로 정렬을 지원하며, 분석 결과를 시각화하여 타임라인 형태로 변환하는 기능을 제공한다. 또한, 커널, 유저 영역 부팅에 소요된 시간을 제공한다. systemd-analyze는 Bootchart와 같이 Init Process 생성 이후 분석을 시작하기 때문에, BMC의 프로파일링 도구로 적합하지 않다.

기존 연구된 프로파일링 도구 중 Ftrace가 가장 많은 정보에 대한 분석을 제공한다. 그러나, 디바이스 드라이버 초기화 성공 여부에 대한 결과가 제공되지 않아 BMC 부팅 과정 분석을 위해, 단독으로 사용하기는 무리가 있다.

본 논문에서는 BMC를 위한 부팅 과정 프로파일링 도구를 제안한다. 제안하는 도구는 BMC 부팅 과정 분석에 필요한 디바이스 드라이버 초기화 과정을 중점으로 분석한다. 또한, CPU 및 메모리 사용률, 커널 버전 관리 기능과 버전 간 CPU, 메모리 사용률, 코드 비교 기능을 제공한다. 본 논문의 구성은 다음과 같다. 2장에서 제안하는 도구가 실행되는 환경, 프로파일링을 위한 정보 수집 기법 및 후처리 과정을 설명한다. 3장에서는 제안하는 도구가 부팅 과정 분석에 사용할 수 있는 예시를 보인다. 4장에서는 제안하는 도구와 기존 연구된 프로파일링 도구에 대한 비교를 수행하고, 마지막으로 결론을 맺는다.

Ⅱ. 본 론

본 장에서는 부팅 과정 프로파일링에 필요한 정보를 수집하는 환경과 과정을 설명한다. 또한, 부팅 과정을 분석하고 개발하는 과정을 단축하기 위해 제공하는 기능에 대해 설명한다.

1. 프로파일링

가. 정보 수집 환경

부팅 과정 분석 환경은 BMC 평가 보드인 AST2600-EVB를 사용한다[9]. AST2600-EVB의 부팅 과정에서 프로파일링에 필요한 정보를 수집하고, 수집한 정보는 후처리를 거쳐 제안하는 도구를 통해 개발자에게 제공한다. 표 1은 AST2600-EVB의 사양 및 커널 버전 정보이다.

표 1. AST2600-EVB 사양

카테고리	설명
CPU	Dual-core ARM Cortex A7 Embedded ARM Cortex M3
SDRAM 메모리	2GB DDR4 SDRAM with speed grade higher than DDR4-1600Mbps
Flash 메모리	SPI flash 메모리
BMC	BMC controller with IPMI 2.0/1.5 compliant
Kernel Version	5.4.62

나. 커널 소스코드 해시값

제안하는 도구는 부팅 과정을 분석하고, 개발을 수행하는 과정을 단축할 수 있는 기능을 제공하기 위해, 코드 변경에 의한 성능 변화 추이를 제공한다.

제안하는 도구에서 커널 소스코드 관리 기능과 소스코드 및 성능 비교 기능을 제공하기 위해, 커널 빌드 시 커널 소스코드를 별도의 저장소에 저장한다. 저장된 커널 소스코드는 제안하는 도구에서 해시 함수를 거쳐 해시값과 함께 저장된다. 커널 소스에 대한 해시값이 저장되는 예시는 표 2와 같다. 커널 소스코드를 해시값으로 변환하기 위해, 널리 쓰이는 해시 함수 중 하나인 MD5를 사용한다[10].

표 2. 커널 소스코드 경로 및 소스코드의 해시값 예시

파일 경로	소스코드 해시값
linux/lib/Makefile	36d535150dce9dd48eead0e7ecd1584f
linux/lib/argv_split.c	de54a0456da92416a1d0334c90fd3ffa
linux/lib/ashldi3.c	82b29e4bf917c3fa1d2ba4c3aa9ff0c5
linux/lib/ashrdi3.c	24ee697b066e06cb0d8836ff1e802c96
linux/lib/asn1_decoder.c	28fd0d6707d61cc0509508a09828fbb7
linux/lib/assoc_array.c	3f5db32658ca5bf175d7de92808ea9c0
linux/lib/atomic64.c	3fd58af52204db52a08a5068958e93d9

다. 초기화 수행 시간

디바이스 드라이버 초기화 과정에서는 인터럽트 처리 과정, 콘솔 출력과 같은 이유로 부팅 시간을 늦추는 요인이 발생할 수 있다. 이를 위해, 제안하는 도구는 디바이스 드라이버 초기화 시간을 수집하여 제공한다. 또한, BMC는 통신 인터페이스를 사용하여 센서, 팬으로부터 정보 수집 등의 기능을 수행하기 때문에, 디바이스 드라이버 초기화 과정에 문제가 생길 시 정상적인 동작이 불가능하다. 제안하는 도구는 디바이스 드라이버 초기화 과정에서 문제가 생기는 것을 추적을 위해, 디바이스 드라이버 초기화 과정과 초기화 함수의 반환 값을 수집하여 제공한다.

디바이스 드라이버 및 파일 시스템 초기화 시간을 수집하기 위해 initcall debug 기능을 사용한다[11]. initcall debug 기능을 사용하여 부팅 과정에 각 initcall 함수 수행 시간, initcall 함수명, 반환 값을 출력한 예시는 그림 1과 같다.

```
initcall armv7_pmu_driver_init+0x0/0x28 returned 0 after 2037 usecs
initcall ptdump_init+0x0/0x98 returned 0 after 3 usecs
initcall arch_uprobes_init+0x0/0x34 returned 0 after 5 usecs
initcall proc_execdomains_init+0x0/0x48 returned 0 after 22 usecs
initcall register_warn_debugfs+0x0/0x48 returned 0 after 58 usecs
initcall cpuhp_sysfs_init+0x0/0xac returned 0 after 147 usecs
initcall ioresources_init+0x0/0x78 returned 0 after 23 usecs
initcall psi_proc_init+0x0/0x7c returned 0 after 39 usecs
initcall irq_pm_init_ops+0x0/0x28 returned 0 after 4 usecs
initcall timekeeping_init_ops+0x0/0x28 returned 0 after 3 usecs
initcall init_clocksource_sysfs+0x0/0x3c returned 0 after 514 usecs
initcall init_timer_list_procfs+0x0/0x54 returned 0 after 13 usecs
initcall alarmtimer_init+0x0/0xb8 returned 0 after 284 usecs
```

그림 1. initcall debug가 포함된 부팅 메세지

라. 함수 호출 과정

디바이스 드라이버 초기화 과정에 대한 정보를 제공하기 위해 Ftrace를 사용한다. Ftrace는 부팅 과정에서 함수 필터링 기능을 통해 커널 함수 수행 시간 및 함수 호출 구조와 같은 정보를 제공한다. Ftrace의 함수 필터링 기능을 통해, 커널 영역 부팅 과정에서 호출되는 initcall 함수의 커널 함수 호출 정보를 수집한다. 그림 2는 Ftrace를 사용하여 initcall 함수의 커널 함수 호출 구조를 그래프 형태로 출력한 예시이다.

```
# tracer: function_graph
#
# CPU  DURATION                  FUNCTION CALLS
# |     |    |                     |   |   |   |
 0)               |  __probe_kernel_read() {
 0)               |    __check_object_size() {
 0)   1.885 us    |      check_stack_object();
 0)   7.128 us    |    }
 0) + 25.689 us   |  }
 0)               |  tracepoint_probe_register() {
 0)               |    tracepoint_probe_register_prio() {
 0)   1.730 us    |      mutex_lock();
 0)               |      __kmalloc() {
 0)   1.410 us    |        kmalloc_slab();
 0)   1.200 us    |        should_failslab();
 0)   6.760 us    |      }
```

그림 2. Ftrace의 함수 필터링 결과

마. CPU, 메모리 사용량

디바이스 드라이버 초기화 과정 중 하드웨어 인터럽트 처리 과정에서 대기가 발생할 경우, CPU, 메모리 사용률이 증가할 수 있다. 이러한 문제점을 찾아 제공하기 위해, 제안하는 도구에서는 CPU, 메모리 사용량을 수집하여, 변환을 거친 후 그래프 형태로 제공한다.

커널에서 CPU, 메모리 사용량은 “kernel_cpustat”, “sysinfo” 구조체에 저장하여 관리하도록 구현되어 있다. 이러한 CPU, 메모리 사용량 관리 알고리즘의 일부를 “linux/init/main.c”에 구현하여 커널 영역 부팅 과정에서 CPU, 메모리 사용량 정보를 수집할 수 있도록 한다. 표 3과 표 4는 CPU, 메모리 사용률 계산을 위한 “kernel_cpustat”, “sysinfo” 구조체 항목과 설명을 보여준다[12].

표 3. CPU 사용량 정보 (단위: HZ)

항목	설명
user	유저 모드에서 일반 프로세스가 수행된 시간
nice	유저 모드에서 낮은 우선순위를 가진 프로세스가 수행된 시간
system	system mode에서 프로세스가 수행된 시간
idle	idle task가 수행된 시간
io_wait	I/O를 마칠 때까지 대기한 시간
irq	interrupts를 수행한 시간
softirq	Soft IRQ를 수행한 시간

표 4. 메모리 사용량 정보 (단위: Kilobyte)

항목	설명
MemTotal	사용 가능한 총 RAM 용량
MemFree	현재 사용되고 있지 않는 RAM 용량

표 5는 부팅 과정에서 CPU, 메모리 사용량을 출력한 예시이다. 1, 5번째 줄에는 “calling” 키워드가 포함되며, 이는 디바이스 드라이버 및 파일 시스템의 초기화 시작 지점과 같다. 또한, 표 2, 3의 데이터는 2, 3, 6, 7번째에 출력되어 있다. “initcall” 키워드가 포함된 4, 8번째 줄은 초기화 작업이 종료된 메세지로 initcall 함수명, 반환 값, 초기화 수행 시간이 포함된다.

표 5. CPU, 메모리 사용량 정보가 포함된 부팅 메세지

줄	메세지
1	calling gpio_led_driver_init+0x0/0x28 @ 1
2	[OSLAB] user : 2, nice : 0, system : 553, idle : 521, iowait : 0, irq : 0, softirq : 0
3	[OSLAB] MemTotal : 924084, MemFree : 789560
4	initcall gpio_led_driver_init+0x0/0x28 returned 0 after 658 usecs
5	calling pca955x_driver_init+0x0/0x28 @ 1
6	[OSLAB] user : 2, nice : 0, system : 553, idle : 521, iowait : 0, irq : 0, softirq : 0
7	[OSLAB] MemTotal : 924084, MemFree : 789560
8	initcall pca955x_driver_init+0x0/0x28 returned 0 after 179 usecs

2. 제안하는 프로파일링 도구

제안하는 도구는 “Kernel Version Management”, “Code”, “Performance” 윈도우로 구성된다. “Kernel Version Management” 윈도우에서는 하나 또는 두 개의 커널 버전을 선택할 수 있다. 하나의 커널 버전 선택 시, 그림 3과 같

은 해당 버전에 대한 소스코드와 디렉토리 구조를 확인할 수 있다. 또한, 특정 소스코드 선택 시 그림 4와 같이 소스코드 내용을 확인할 수 있다. 그리고, Performance 윈도우에서 해당 버전에 대한 initcall 함수 호출 정보 및 CPU, 메모리 사용률을 그래프를 통해 확인할 수 있다. 두 개의 커널 버전 선택 시, 버전 간 소스코드 차이와 성능 차이를 확인할 수 있다.

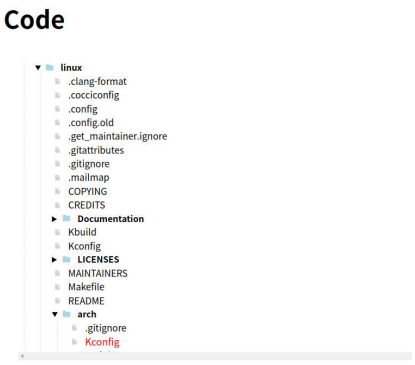

그림 3. 커널 소스코드 및 디렉토리 구조

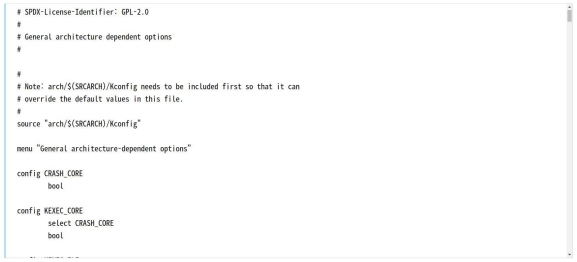

그림 4. Code 윈도우의 파일 내용 출력 기능

Performance 윈도우에서 커널 영역 부팅 과정에서 초기화되는 디바이스 드라이버 및 파일 시스템의 initcall 함수 수행 과정을 확인할 수 있으며, 이에 대한 예시는 그림 5와 같다. 그림 5의 X축은 커널 영역 부팅 시간 (단위: 초), Y축은 CPU, Memory 사용률이다. 그림 5 우측 영역에 표시되어 있는 것과 같이, 파란 영역은 CPU 사용률, 빨간 영역은 메모리 사용률을 표시한다.

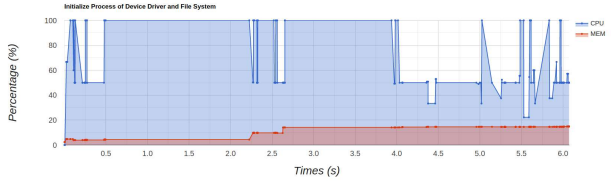

그림 5. Performance 윈도우의 커널 영역 부팅 과정

initcall 함수의 커널 함수 호출 추적을 위해 Ftrace를 사용할 경우, 추적 결과 크기가 방대하여 분석하기 어렵다는 단점이 있다. 그림 2의 경우, initcall 함수 일부에 포함되어 있는 키워드인 “probe”를 필터링하고, 함수 호출 추적 최대 깊이를 9로 설정했음에도 불구하고, 프로파일링 결과로 약 21만 줄이 출력된다. 방대한 데이터에 대한 분석을 지원하기 위해, 제안하는 도구는 initcall 함수의 커널 함수 호출 추적 정보를 통계화하여 제공한다. 그림 5의 X축에서 특정 디바이스 드라이버가 초기화가 시작된 지점 선택 시, initcall 함수의 CPU 사용률, 수행 시간, 함수 반환 값, 함수명, 커널 함수 호출 정보를 통계화한 테이블을 확인할 수 있으며, 이에 대한 예시는 표 6과 같다. 또한, 각 Function, Count, Time (μs) 항목에 대한 정렬을 지원한다. 표 6은 함수 호출 횟수 (Count)를 기준으로 내림차순 정렬한 결과이다.

표 6. Initcall 함수 정보

CPU Utilization : 100%
Initial Time : 1.282631s
Initcall Return Value : 0
Initcall Name : ast8250_platform_driver_init

Functions	Count	Time (μs)
aspeed_pinctrl_get_group_name()	8434	13444.04
pinctrl_dev_get_drvdata()	8090	9570.14
aspeed_pinmux_get_fn_name()	7794	27563.96
of_prop_next_string()	3821	4737.01
__of_find_property()	3514	5673.18
__of_device_is_compatible()	2596	19268.43

3. 제안하는 도구 사용 예시

제안하는 도구를 사용하는 예시와 커널 버전 간 비교 기능 예시를 설명하기 위해, 시리얼 디바이스 드라이버 출력을 제거하여 커널 부팅 과정 중 콘솔 출력을 제거한다. 또한, 부팅 과정을 자세하게 분석하기 위해 로그 버퍼를 확장한다. 이러한 수정사항을 반영한 커널과 수정 이전 커널을 비교를 수행한다.

그림 6~9는 실제 커널 부팅 로그를 수집하고, 부팅 시간 단축을 위해 커널을 수정한 버전과 수정 이전의 버전을 비교하기 위해 제안하는 도구를 사용한 예시이다. 제안하는 도구는 두 버전에 대

한 소스코드, 성능 비교를 지원함으로써 소스코드 변경 사항에 의한 성능 차이를 가시화된 그래프로 비교할 수 있다. 그림 6은 두 커널 버전 소스코드의 해시값이 저장된 파일을 비교한 예시이다. 이처럼, 각 버전의 해시값이 저장된 파일 비교하여 두 버전 간 소스코드 차이점을 찾을 수 있다. 그림 7은 제안하는 도구에서 두 버전 간 차이가 있는 소스코드 목록을 제공하는 예시이다. 그림 8은 그림 7에서 선택한 소스코드의 버전 간 차이점을 리눅스 diff 명령어를 사용하여, 제공하는 예시이다.

```
3c3
< linux/.config aabc4925db83824ebbcf2fb83c7909b7
---
> linux/.config c169be25b7f9beefb8bb90b74d071feb
52257c52257
<linux/drivers/tty/serial/8250/8250_core.c
9a26074cb5363ce54481accc1bfa8e9a
---
>linux/drivers/tty/serial/8250/8250_core.c
00baba7915a476e70af07c1b3b32af0
```

그림 6. 버전 간 해시값 차이

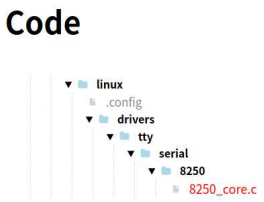

그림 7. 버전 간 차이가 있는 소스코드 목록

```
585,586d584
<
<     serial8250_console_write(up, s, count);
```

그림 8. 버전 간 소스코드 차이점 출력 결과

그림 9는 두 버전 간 CPU, Memory 사용률 차이를 시각화한 그래프이다. 그림 9를 통해, 수정한 커널의 커널 부팅 시간이 단축되었음을 확인할 수 있다. CPU_1, MEM_1은 부팅 시간 단축 전, CPU_2, MEM_2는 단축 후의 CPU, 메모리 사용률을 나타낸다.

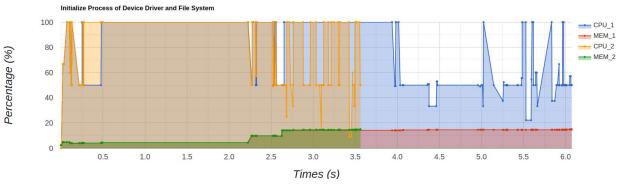

그림 9. 두 버전 간 CPU, Memory 사용률 차이

4. 프로파일링 도구 비교

표 7은 BMC 부팅 과정 분석에 필요한 영역, 정보, 기능 제공 여부를 비교하기 위해, 제안하는 도구와 기존 연구된 프로파일링 도구에 대해 비교를 수행한 결과이다. 부팅 과정 분석에 주로 사용되는 도구인 Bootgraph, Bootchart, Ftrace, systemd-analyze에 대해 프로파일링 영역, 프로파일링 정보, 도구가 제공하는 기능에 대해 비교를 수행한다.

BMC는 통신 인터페이스를 통해 센서, 팬으로부터 하드웨어 정보를 수집하기 때문에, 부팅 과정에서 디바이스 드라이버 초기화 과정이 중요하다. Bootgraph, Ftrace는 디바이스 드라이버 초기화 과정이 수행되는 커널 부팅 과정에 대해 분석을 지원한다. 그러나, Bootgraph는 프로파일링 가능한 정보 중 디바이스 드라이버 초기화 시간만 제공한다. 또한, Ftrace는 디바이스 드라이버 초기화 결과를 확인할 수 없고, CPU, 메모리 사용률을 제공하지 않는다. 따라서, Bootgraph, Ftrace는 BMC 부팅 과정 분석으로 필요한 정보를 모두 제공하지 못하기 때문에, 단독으로 사용하기 어렵다.

부팅 시간 단축 과정은 분석, 실험, 결과 확인의 반복으로 수행된다. 이러한 과정을 단축하기 위해 제안하는 도구는 BMC 부팅 과정 분석에 필요한 정보를 모두 제공함과 동시에 분석 결과에 대한 시각화와 커널 버전 관리 기능을 제공한다. 분석 결과 시각화와 커널 버전 관리 및 비교 기능을 통해 개발자는 부팅 시간 단축 과정에서 추가적인 결과 분석을 거치지 않고, 실험에 대한 결과를 확인할 수 있다.

표 7. 프로파일링 도구 비교

내용 \ 도구		Boot graph	Boot chart	Ftrace	systemd -analyze	제안하는 도구
영역	Kernel	O	X	O	X	O
	User	X	O	O	O	X
정보	CPU, 메모리	X	O	X	X	O
	initcall 소요 시간	O	X	O	X	O
	initcall 과정	X	X	O	X	O
	initcall 결과	X	X	X	X	O
기능	시각화	O	O	X	O	O
	커널 버전 비교	X	X	X	X	O

Ⅲ. 결 론

본 논문에서는 BMC의 부팅 과정을 분석하기 위한 프로파일링 도구를 제안하였다. 제안하는 도구에서는 BMC 부팅 과정 분석에 필요한 정보인 디바이스 드라이버 초기화 과정, 초기화 함수 반환 값을 제공한다. 이러한 기능은 통신 인터페이스를 통해 센서로부터 하드웨어 정보를 수집하는 BMC의 프로파일링 도구에는 필수적인 기능이다. 또한, 커널 버전 별 관리 기능을 제공하며, 이를 통해 버전 별로 소스코드, 성능을 확인할 수 있다. CPU 및 메모리 사용률과 초기화 시간을 확인할 수 있도록, 정보를 후처리하여 그래프 및 표 형태로 제공한다. 마지막으로, 제안된 도구에서는 커널 개발 버전 간 소스코드 및 부팅 과정의 차이점을 제공함으로써 개발 사항에 의한 성능 변화를 확인하는 데 도움을 줄 수 있는 기능을 제공한다. 제안한 도구를 구현하고, 기존 연구된 도구와의 비교를 수행함으로써 제안하는 도구가 BMC의 부팅 시간 분석에 효율적이며, 분석 결과는 부팅 시간 단축에 사용될 수 있음을 보였다.

REFERENCES

[1] J. Frazelle, “Opening Up the Baseboard Management Controller: If the CPU is the Brain of the Board, the BMC is the Brain Stem,” *Queue*, Vol. 17, No. 5, pp. 5-12, Sep-Oct, 2019.

[2] J. An, Y. Kim, and C. Park, “Design of Framework supporting IPMI and DCMI based on Open BMC,” *Proc. of the International Conference on Research in Adaptive and Convergent Systems (RACS '17)*, pp. 298-299, Sep, 2017.

[3] Using IPMI platform management in modular computer systems (2003), https://www.intel.my/content/dam/www/public/us/en/documents/product-briefs/modular-computer-systems.pdf (accessed Jun., 2, 2022).

[4] Bootgraph(2008), https://github.com/torvalds/linux/blob/master/scripts/bootgraph.pl (accessed Jun., 3, 2022).

[5] Demystifying Linux Kernel initcalls(2020), https://elinux.org/images/e/e8/2020_ELCE_initcalls_myjosserand.pdf (accessed Jun., 2, 2022).

[6] Bootchart(2021), https://elinux.org/Bootchart (accessed Jun., 4, 2022).

[7] The Linux Kernel/Boot-Time Tracing(2020), https://www.kernel.org/doc/html/v5.8/trace/boottime-trace.html (accessed Jun., 7, 2022).

[8] systemd-analyze(2022), https://github.com/systemd/systemd (accessed Dec., 5, 2022).

[9] AST2600(2019), https://www.aspeedtech.com/server_ast2600/ (accessed Sep, 1, 2022).

[10] RFC1321: The MD5 Message-digest Algorithm(1992), https://www.rfc-editor.org/rfc/rfc1321 (accessed Oct., 27, 2022).

[11] Initcall Debug(2012), https://elinux.org/Initcall_Debug (accessed June., 6, 2022).

[12] linux/fs/proc(2022), https://github.com/torvalds/linux/tree/master/fs/proc (accessed June., 7, 2022).

저 자 소 개

김재섭(학생회원)

2021년 숭실대학교 컴퓨터학부 학사 졸업.
2021년~현재 숭실대학교 컴퓨터학과 석사 재학.

<주관심분야 : 운영체제, 임베디드 시스템>

박민호(학생회원)

2022년 숭실대학교 컴퓨터학부 학사 졸업.
2022년~현재 숭실대학교 컴퓨터학과 석사 재학.

<주관심분야 : 운영체제, 임베디드 시스템>

홍지만(종신회원)

2003년 서울대학교 컴퓨터공학과 박사 졸업.
2004년~2007년 광운대학교 컴퓨터공학과 교수
2007년~현재 숭실대학교 컴퓨터학부 교수.

<주관심분야 : 운영체제, 임베디드 시스템>