RTEMS 커널 API 기반 경량 실시간 모니터링 시스템

(Lightweight Real-time Monitoring System based on RTEMS Kernel API)

신대현*, 박주찬*, 장준혁**

(Daehyeon Shin, Juchan Park, Joonhyouk Jang)

요 약

RTEMS는 우주 임무에 널리 사용되는 실시간 운영체제로서, 모니터링을 위한 RTEMS Record 기능을 제공한다. 하지만 RTEMS Record는 실시간 모니터링이 아니라 사후 분석을 위한 기능이며, 데이터 전송 시 수집한 데이터보다 훨씬 큰 메타데이터를 함께 전송한다. 이에 본 연구는 RTEMS 커널 API 기반의 경량 실시간 모니터링 시스템을 제안한다. 제안 시스템은 모니터링을 위한 최소한의 정보를 추출하고, UART 전송에 용이한 데이터 패킷 구조를 설계하여 모니터링 오버헤드를 최소화한다. 또한, 수집한 정보를 시각적으로 제공하는 모니터링 도구를 개발하였다. SIS 시뮬레이터 환경에서 RTEMS Record 방식과 제안 방식을 비교하여, 제안 방식이 전송량(bytes/s)을 크게 감소시키고 모니터링 태스크의 CPU 점유율을 낮춰 실행 간섭을 완화함을 확인하였다.

■ 중심어 : 우주임무 ; 실시간 운영체제 ; RTEMS ; 모니터링 ; 시각화

Abstract

RTEMS is a real-time operating system widely used in space missions and provides the RTEMS Record functionality for monitoring. However, RTEMS Record is a non-real-time method, and it transfers extensive metadata that can be substantially larger than the collected payload. To address this limitation, this paper proposes a lightweight real-time monitoring system based on RTEMS kernel APIs. The proposed system extracts minimized the information required for monitoring and adopts a packet format designed for UART transmission to minimize monitoring overhead. In addition, we implement a host-side monitoring tool that visualizes the collected data in real time. Using the SIS emulator environment, we compare the proposed monitoring system with the RTEMS Record-based monitoring and show that the proposed system significantly reduces the transmission volume (bytes/s) and lowers the CPU utilization of the monitoring task, thereby mitigating execution interference.

■ keywords : space mission ; real-time operating system ; RTEMS ; monitoring ; visualization

Ⅰ. 서 론

우주 임무에 활용되는 실시간 운영체제는 탑재체 발사 이후에는 소프트웨어 갱신이 제한적이고, 결함의 사후 대응 비용이 매우 크다. 이러한 우주 환경의 특수성을 고려하면, 개발·통합 단계에서 운영체제의 스케줄링 및 자원 관리 동작을 체계적으로 관찰하고, 사전 검증할 수 있는 모니터링 과정이 필수적이다. 하지만 모니터링은 계

* 준회원, 한남대학교 우주공학과

** 종신회원, 한남대학교 컴퓨터공학과

본 과제는 2026(결과물)는 2026년도 교육부 및 대전광역시의 재원으로 대전RISE센터의 지원을 받아 수행된 지역혁신중심 대학지원체계(RISE)의 결과입니다.(2026-RISE-06-013)

접수일자 : 2026년 03월 03일 게재확정일 : 2026년 03월 24일

교신저자 : 장준혁 e-mail : jhjang@hnu.kr

측을 위한 추가 연산과 데이터 전송을 포함하므로, 모니터링 자체가 시스템의 실행에 간섭할 수 있다. 모니터링으로 인한 간섭을 줄이기 위해 일반 운영체제인 Linux는 per-CPU 버퍼를 사용해 CPU 오버헤드를 낮추고[9][10], 실시간 운영체제인 FreeRTOS는 계측 이벤트 필터링을 통해 모니터링 오버헤드를 낮추는[11] 등 경량화 기법을 적용하고 있다.

RTEMS(Real-Time Executive for Multiprocessor Systems)는 우주 임무에 널리 사용되는 실시간 운영체제로서 ESA(European Space Agency), NASA(National Aeronautics and Space Administration)를 비롯한 다양한 우주 임무에서 활용된다[1][16]. RTEMS에서 운영체제 내부 동작을 분석하기 위해 제공하는 RTEMS Record는 이벤트 기반 트레이싱을 통해 내부 동작을 분석하는 기능을 제공한다. 그러나 해당 기능은 데이터 시간 축 정렬 등 분석을 위한 메타데이터를 함께 포함하므로, 실제 수집된 정보량에 비해 전송 데이터 크기가 증가한다.

본 연구는 이러한 문제를 해결하기 위해 RTEMS 커널 API 기반의 경량 실시간 모니터링 시스템을 제안한다. 제안 시스템은 모니터링에 필요한 최소 항목만 정의하여 불필요한 메타데이터를 제거하고, 전송에 용이한 데이터 프레임 구조를 설계하여 모니터링 오버헤드를 줄인다. 또한 수집한 데이터를 시각화하는 응용프로그램을 통해 RTEMS 기반 개발·통합 단계에서 작은 오버헤드로 내부 동작을 직관적으로 관찰할 수 있게 하였다.

본 논문의 구성은 다음과 같다. 2장에서는 연구 배경 및 관련 연구를 소개한다. 3장에서는 RTEMS 커널 API 기반 경량 모니터링 시스템의 아키텍처와 데이터 프레임 설계를 설명한다. 4장에서는 RTEMS Record 방식과 제안 방식을 SIS 시뮬레이터 환경에서 비교 실험하고 모니터 태스크의 데이터 전송량 및 CPU 사용률을 비교한다. 5장에서는 결론과 한계에 대해 논의한다.

Ⅱ. 배경 및 관련 연구

1. 연구 배경

가. 우주 소프트웨어 특수성

우주 임무용 소프트웨어는 비행 이전 지상 단계에서의 체계적 시험과 검증이 규범화되어 있다. 이와 관련하여 NASA는 소프트웨어 획득·개발·시험·운영·유지보수 전 과정에 걸쳐 소프트웨어 보증, 안전, IV&V를 체계적으로 수행하고, 그 결과를 객관적 증거로 관리할 것을 표준으로 요구한다[12][13]. IV&V는 개발 조직과 분리된 독립 기관이 요구사항 준수와 의도 적합성을 문서·시험 결과를 통해 확인하는 활동을 뜻한다[15].

대표적으로 NASA-STD-8739.8B는 보증 활동의 목적·역할·산출물을 규정하며, 분석과 시험을 통해 수집한 증거를 독립적 평가에 활용하도록 명시한다[12]. 한편, NPR 7150.2D는 NASA 소프트웨어 엔지니어링 최소 요구사항을 제시하며, 각 개발 단계에서 V&V(검증, 타당성 확인) 활동과 산출물의 확인을 요구한다[13]. 실무 지침인 Software Engineering Handbook는 이러한 요구에 따라 SWE-066(시험 수행), SWE-141(소프트웨어 IV&V 적용)등 조항을 통해 테스트 결과와 근거 자료의 확보를 요구한다[14]. 이는 개발·통합 단계에서 운영체제와 응용프로그램의 실제 동작을 데이터로 확인할 수 있는 모니터링·시각화 수단이 필요함을 시사한다.

이러한 요구사항들을 충족하기 위해서 실시간 운영체제 상에서 태스크 스케줄링과 자원 관리가 설계 의도대로 동작하는지를 실행 결과를 통해 확인할 수 있어야 한다. 선행연구[18]에서는 RTEMS의 SMP 지원과 스케줄링 기법을 분석하여, RTEMS 내부 동작 분석의 필요성을 제시하였다. 특히 개발·통합 단계에서는 주기 미준수, 우선순위 역전, 자원 부족과 같은 문제가 실행 중에만 드러나는 경우가 많아, 운영체제 내부 동작을 관찰할 수 있는 모니터링 수단이 필요하다.

2. 관련 연구

가. 계측 오버헤드 및 실행 간섭

최근 실시간 시스템의 모니터링으로 인한 실행 간섭을 최소화하기 위한 경량화 계측 접근이 연구 주제로 제안되고 있다[4][5][7]. 실시간 시스템에서 모니터링은 실행 중인 태스크의 스케줄링 흐름과 자원 사용 상태를 관찰할 수 있으나, 모니터링을 위해 삽입된 코드는 CPU 실행 시간, I/O 처리, 버퍼 사용을 추가로 유발한다. 따라서, 모니터링 자체가 시스템 상태를 교란할 수 있으며[2], 모니터링이 런타임 오버헤드를 유발하고 그 영향이 무시 가능한 수준부터 치명적 수준까지 커질 수 있다.

이러한 문제를 해결하기 위한 접근은 크게 두 축으로 나뉜다. 첫째, 수집량 자체를 제한하는 방식으로 수집 정보를 선택적으로 좁혀 모니터링 비용을 감소시키는 전략이다. Craun 등은 eBPF 훅 포인트를 프로세스별로 분리해 모니터링 오버헤드를 감소시키고[6], Percepio의 RTOS 트레이싱 도구는 이벤트 필터링 및 스냅샷/스트리밍 모드 선택으로 기록량을 제어한다[11].

둘째, 동일한 관측 목표는 유지하면서 수집 경로를 단순화 하거나, 수집 주기를 조절함으로써 모니터링 비용을 감소시키는 전략이다. Forlina 등은 FreeRTOS의 portable layer에 계측을 통합하는 저오버헤드 모니터링을 제안하며[4], Chen 등은 마일스톤 기반 진행 평가로 런타임 관측을 최소화한다[5]. 또한 Jiang 등은 자율주행 시스템 전 계층을 대상으로 수집·분석 경로를 표준화한 저오버헤드 트레이싱 프레임워크를 제안하고 있다[7]. 따라서 실시간 시스템 모니터링은 관측의 유용성을 유지하면서도 실행 간섭을 최소화하는 경량화 설계가 필수적이다[8]. 이러한 요구는 RTEMS 환경에서도 동일하다. RTEMS Record는 메타데이터를 포함하여 자원 제약 환경에서 오버헤드가 증가할 수 있으므로, 이를 최소화하는 경량 모니터링 체계가 필요하다.

Ⅲ. 본론

1. 제안 시스템 개요

그림 1은 RTEMS 커널 API 기반 경량 실시간 모니터링 시스템의 동작 흐름을 나타낸다. 제안 시스템은 RTEMS 타깃에서 스케줄링 및 자원 사용과 관련된 핵심 정보를 최소 항목으로 수집한 뒤, UART 전송에 적합한 데이터 패킷으로 구성하여 링버퍼에 적재한다. 이후 모니터 태스크가 주기적으로(1000ms) 링버퍼에 적재된 데이터를 SIS 시뮬레이터 콘솔로 flush하고 파일로 저장한다. 호스트 PC는 파일을 읽어 파싱/태깅하여 GUI로 시각화한다. 본 아키텍처는 RTEMS Record 기반 트레이싱에서 발생하는 부가 메타데이터로 인한 모니터링 오버헤드를 감소시키기 위해 모니터링에 필요한 최소 항목만 추출하여 링버퍼에 적재하고 주기적으로 flush하는 구조로 설계하였다. 또한 로그를 실행 종료 후 일괄 분석하는 방식이 아니라, 실행 중 출력되는 스트림을 호스트 측 모니터링 도구가 연속적으로 수신·해석하여 화면을 갱신하는 방식을 사용함으로써 실시간 모니터링이 가능하다. 이를 통해 개발·통합 단계에서 스케줄링 이상이나 자원 부족 징후를 조기에 확인할 수 있다.

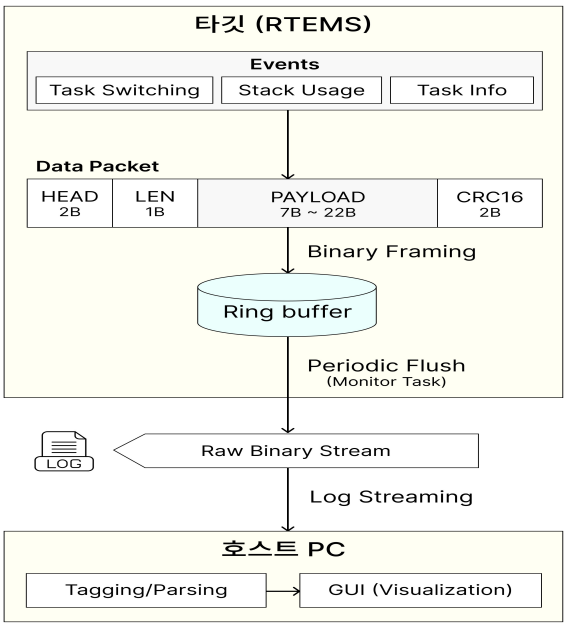

그림 1. 제안 시스템 개요

2. 타깃 이벤트 추출 및 시각화

가. 타깃 이벤트 추출

RTEMS 타깃에서 스케줄링 및 자원 관리 동작을 관찰하기 위해 타깃 내부 이벤트를 최소 항목으로 추출한다. 태스크 전환(Task Switching) 이벤트는 스케줄링 흐름을 기록하기 위해 계측하며, 스택 사용량(Stack Usage)은 RTEMS의 스택 점검 기능을 이용해 태스크별 사용량을 확인한다. 또한 Task Info는 호스트 분석 도구에서 태스크 ID - 이름 매핑 테이블을 구성하기 위한 기준 정보로 사용된다.

수집된 정보는 헤더(HEAD), 길이(LEN), PAYLOAD, 무결성 검증 값(CRC16)을 포함하는 데이터 패킷으로 구성된다. 생성된 데이터 패킷은 즉시 출력하지 않고 링버퍼에 적재하여 I/O 호출 빈도를 제한함으로써 실행 간섭을 완화한다. 링버퍼에 누적된 데이터는 모니터 태스크가 설정된 주기에 따라 출력한다. 이후 해당 출력은 SIS 시뮬레이터 콘솔 출력(stdout)으로 전달되어 로그 파일로 저장한다.

나. 데이터 패킷 처리 및 시각화

저장되는 로그 파일의 경로를 호스트 측 Python 도구에 입력하면 도구는 해당 파일을 바이트 단위로 연속적으로 읽어 들인다. 이후 헤더(HEAD)를 탐색하여 각 데이터 패킷의 시작점을 식별한다. 또한 길이 필드(LEN)로 결정된 PAYLOAD에 대해 CRC16을 계산하여 수신된 CRC16과 비교하여 데이터 패킷의 무결성을 검증한다.

검증된 데이터 패킷은 PAYLOAD 필드의 공통 헤더를 기준으로 이벤트 타입과 타임스탬프를 추출한 뒤, 태스크 전환, 스택 사용량, 태스크 메타정보(Task Info)로 분류된다. Task Info 프레임으로 태스크 ID, 이름, 우선순위, 스택 크기 및 인덱스 매핑을 구성하고, 이를 이용해 스위칭 및 스택 이벤트를 태깅한다. 또한 태스크 전환 이벤트의 타임스탬프 차이(전환 간격)를 이용해 태스크별 실행시간을 누적하여 CPU 사용률을 계산하며, 스택 사용량의 경우 사용량과 총 스택 크기를 이용해 스택 사용률을 산출한다. 최종적으로 태스크별 CPU 사용률, 스택 사용량, 태스크 전환 타임라인을 표와 그래프로 시각화한다(그림 2).

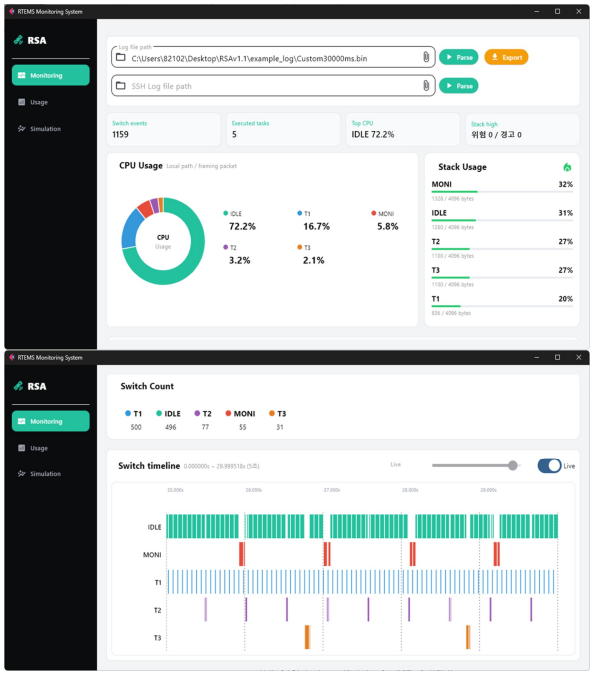

그림 2. 모니터링 도구

3. 데이터 패킷

가. 데이터 패킷 구조

그림 3의 상단은 UART 전송을 위한 데이터 패킷 구조를 나타낸다. 프레임은 경계 식별을 위한 HEAD1과 HEAD2로 시작하며, LEN은 PAYLOAD의 길이를 나타낸다. CRC16은 CRC-CCITT-FALSE 방식으로 계산되며, big-endian으로 전송된다. 본 논문에서 데이터 패킷 1개당 전송 크기는 HEAD1+HEAD2+LEN+PAYLOAD+CRC16으로 정의하며, 데이터 패킷 고정 오버헤드는 HEAD1(1B)+HEAD2(1B)+LEN(1B)+CRC16(2B)로 총 5B이다. 이러한 구조는 전송 과정에서 바이트 손상/혼입이 발생하더라도 헤더 재탐색을 통해 프레임 경계를 식별하고 CRC 검증으로 손상 프레임을 검출할 수 있게 한다.

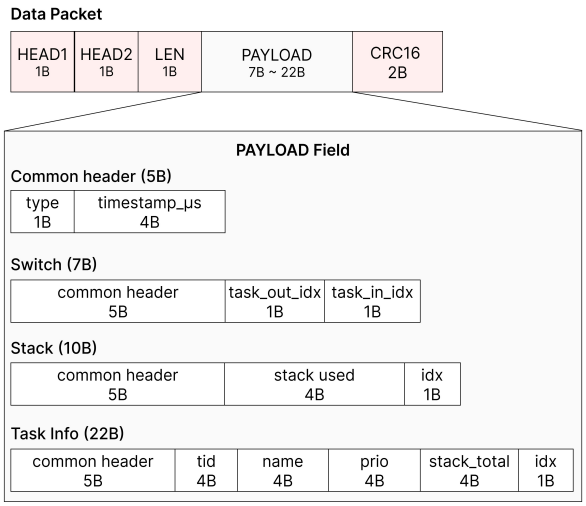

그림 3. 데이터 패킷 및 PAYLOAD 필드 구조

그림 3의 하단은 PAYLOAD 필드에 포함되는 이벤트 타입별 데이터 크기를 정의한다. 포맷은 공통적으로 이벤트 타입을 나타내는 type(1B)와 시간을 나타내는 timestamp_µs(4B)를 포함하여 총 5B로 구성되며, SWITCH는 전환 전/후 태스크의 식별자(out_idx, in_idx)로 구성된다. STACK은 태스크 식별자와 스택 사용량을 포함하여 태스크별 스택 사용량으로 구성된다. TASKINFO는 태스크의 tid, name, priority, stack_total, idx를 전달하며, 이 정보는 이후 SWITCH/STACK 이벤트를 태깅하기 위한 ID-이름 매핑 정보로 사용된다.

나. 제안 기법과 RTEMS Record의 데이터 패킷 크기 비교

본 절에서는 제안 방식과 RTEMS Record의 전송량 차이를 이벤트 1회당 전송 크기(bytes/event)를 비교한다. 표 1은 동일 관측 항목에 대해 이벤트 단위 전송 크기를 비교한 결과이며, item은 RTEMS Record 스트림에서 하나의 관측 정보를 구성하는 기본 레코드 단위(8B)를 의미한다. 표 2는 RTEMS Record에서 사후 분석을 위해 추가로 포함되는 메타데이터이다.

표 1. 이벤트 1회당 bytes/event 비교

항목	Proposed	크기	Record API	크기
Switch	Switch(7B) + 패킷 헤더(5B)	12B	Task_Out(8B) + Task_In(8B)	16B
Stack	Stack(10B) + 패킷 헤더(5B)	15B	Thread_ID(8B) + Stack_Usage(8B)	16B
Task Meta	Task Info(22B) + 패킷 헤더(5B)	27B	Thread_ID(8B) + Prio(8B) + Stack_Size(8B)	24B

표 2. RTEMS Record의 추가 메타데이터

항목	설명	크기
Stream Header	스트림 식별/해석 정보	~184B
Uptime Sync	절대 시간 앵커	16B / 주기(20ms)
Drain Meta	버퍼 drain 시 CPU 상태 정보	24B / drain 1회

표 1에 따르면, Switch 이벤트는 제안 방식이 12B인 반면 RTEMS Record는 SWITCH_OUT과 SWITCH_IN의 2개 item으로 구성되어 총 16B이다. Stack 이벤트는 제안 방식 15B, RTEMS Record 16B로 유사하며, 두 방식 모두 태스크 식별과 스택 사용 상태를 전달한다. Task Meta(TaskInfo)는 제안 방식 27B, RTEMS Record는 24B로 제안 방식이 다소 크게 나타나는데, 이는 태스크 태깅을 위한 식별 정보를 포함하는 설계 선택에 기인한다. 그러나 Task Meta는 일반적으로 초기 구간 또는 낮은 빈도로 전송되는 메타 정보이므로, 전체 전송량은 고빈도 이벤트(Switch)의 발생률에 의해 크게 좌우될 수 있다.

한편 표 2는 Record API에서 이벤트 레코드 외에도 스트림 시작 시점의 Stream Header(약 184B), 시간 축 정렬을 위한 Uptime Sync(16B/주기), 버퍼 drain 시점의 Drain Meta(24B/drain 1회) 등 추가 메타데이터가 포함된다.

Ⅳ. 실험

1. 실험 환경 및 구성

본 실험은 RTEMS Record 기반 모니터링 방식과 제안 방식의 전송량(bytes/s)과 CPU 사용률을 비교하기 위해 수행하였다.

가. 실험 환경

실험 환경은 표 3과 같이 RTEMS QDP(Qualification Data Package) GR712RC-UNI(단일 코어) 환경에서 SIS 시뮬레이터를 통해 수행하였다. SIS 시뮬레이터는 Frontgrade Gaisler사의 SPARC/LEON 기반 시스템의 명령어 수준 동작을 모사하는 오픈소스 시뮬레이터이며, 실제 UART 직렬화 지연을 반영하지 않으므로, 본 연구에서는 UART 8-N-1 및 115,200 bps를 가정하여 바이트당 전송 시간(≈86.8 μs/byte)에 기반한 소프트웨어 지연을 삽입하였다.

표 3. 실험 환경

항목	설정
Host PC	AMD Ryzen 5 5700X3D, RAM 32GB, Windows 11
실행 환경	VS Code SSH 원격(Ubuntu 22.04)에서 빌드/실행
타깃	RTEMS QDP GR712RC-UNI, SIS Simulator

표 4. 실험 조건

항목	설정
비교 대상	제안 방식 / RTEMS Record 방식
UART 가정	8-N-1, 115,200 bps
지연 반영	출력 바이트 수에 비례한 소프트웨어 지연 삽입
FLUSH 주기	1000ms

나. 실험 워크로드 구성

실험 워크로드 구성은 표 5와 같다. 워크로드의 총 실행 시간은 60초이며, 세 개의 주기 태스크(T1~T3)는 서로 다른 주기와 실행 시간을 갖도록 설정하여 태스크 전환이 반복적으로 발생하도록 하였다. MONI 태스크(모니터링 태스크)는 타깃에서 수집한 이벤트를 주기적으로 flush하며, 워크로드 실행을 방해하지 않도록 가장 낮은 우선순위(150)로 설정하였다. 또한 UART 전송 지연을 반영하기 위해 MONI 태스크의 출력 바이트 수에 비례한 지연을 포함하여 동작한다.

두 방식 모두 SIS 시뮬레이터 콘솔 출력을 통해 로그 파일로 저장한 후, Python 기반 분석 도구로 처리하여 지표를 산출하였다. 제안 방식 로그는 프레임 식별(HEAD1/HEAD2), 길이(LEN), CRC 검증을 기반으로 이벤트를 복원하였고, RTEMS Record 방식 로그는 RTEMS Record 데이터 구조 정의(recorddata.h)를 참고하여 파싱 로직을 구현하였으며[17], 이후 로그 파일 크기와 실행 시간을 기반으로 시간당 전송량(bytes/s)과 CPU 사용률을 산출하여 두 방식을 비교하였다.

표 5. 워크로드 구성(60초)

태스크	우선순위	실행 시간(ms)	주기(ms)
T1	10	10	50
T2	20	20	500
T3	40	50	2000
MONI	150	출력 바이트 수에 비례	1000

2. 실험 결과

SIS 시뮬레이터 환경 특성상 반복 실행에 따른 시간당 전송량(bytes/s)과 CPU 사용률의 변동이 발생하지 않는다. 따라서 본 절에서는 평균값을 별도로 산출하지 않고 단일 실행에서 측정한 전송량과 CPU 사용률을 기준으로 제안 방식과 RTEMS Record 방식의 결과를 비교하였다.

가. 전송량 비교

시간당 전송량(bytes/s)은 타깃에서 생성한 로그 파일 크기를 총 실행시간으로 나누어 계산한다.

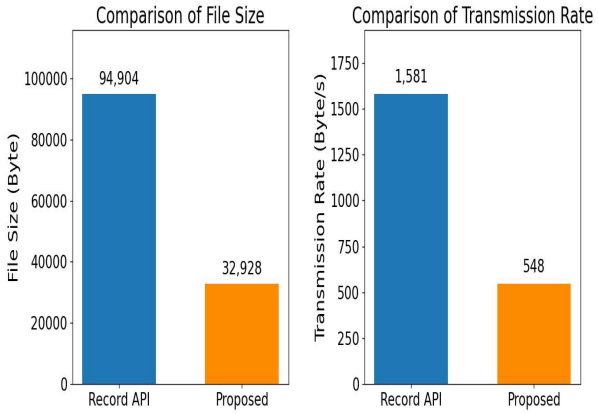

그림 4. 총 전송량 및 시간당 전송량 비교

그림 4는 제안 방식과 RTEMS Record 방식의 총 전송량 및 시간당 전송량을 비교한 결과이다. 제안 방식은 548 bytes/s, RTEMS Record 방식

의 시간당 전송량은 1,581 bytes/s로 측정되었으며, 제안 방식이 RTEMS Record 방식 대비 65.3% 감소하였다. 이는 RTEMS Record 방식이 모니터링 이벤트 외에도 데이터 시간 축 정렬 및 스트림 관리와 같은 사후 분석 목적의 부가 메타데이터가 함께 포함되어 전송량이 증가하는 반면, 제안 방식은 모니터링 목적에 필요한 최소 항목만 추출하여 전송함으로써 전송 부담을 완화한 결과로 해석된다.

나. CPU 사용률 비교

본 절에서는 로그 출력이 RTEMS 실행에 미치는 영향을 평가하기 위해 MONI 태스크의 CPU 사용률을 비교한다. CPU 사용률은 타깃에서 계산하지 않고, 태스크 전환 이벤트 간 시간 정보를 기반으로 모니터링 도구에서 태스크별 실행 구간을 누적한 뒤 전체 실행시간으로 산출하였다. 이를 통해 CPU 사용률 계산 자체로 인한 타깃 측 추가 연산을 최소화하면서도, 로그 출력 방식에 따른 실행 간섭을 정량적으로 비교할 수 있다.

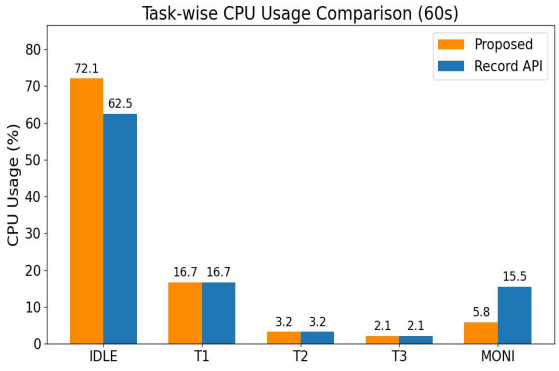

그림 5. 태스크별 CPU 사용률

그림 5는 제안 방식과 RTEMS Record 방식의 태스크별 CPU 사용률 결과를 나타낸다. 두 방식 모두 워크로드 태스크(T1, T2, T3)의 CPU 사용률은 동일하게 유지되었으나, 로그 출력/전송을 담당하는 MONI 태스크의 점유율과 IDLE 비율에서 유의미한 차이가 관측되었다.

제안 방식의 CPU 점유율은 RTEMS Record 방식 대비 5.8% 감소하였고, IDLE 비율은 9.6% 증가하였다. 이는 제안 방식이 RTEMS Record 방식 대비 시간당 전송량(bytes/s)을 감소시켜 MONI 태스크가 처리해야 하는 출력 바이트 양이 줄어든 결과로 해석된다. 특히 본 실험에서는 flush 단계에서 바이트당 전송 지연을 적용하였으므로, 출력 바이트 양 감소는 MONI 태스크의 점유율 감소로 직접적으로 반영된다. 결과적으로, 제안 방식은 RTEMS Record 방식 대비 MONI 태스크가 소비하는 CPU 시간을 크게 절감하고, 모니터링 경로에서 발생하는 실행 간섭을 낮추는 효과를 보였다.

V. 결론

우주 임무에서는 발사 이후 갱신이 제한되고 사후 대응 비용이 크므로, 개발·통합 단계에서 운영체제 내부 동작을 관찰·검증할 모니터링이 필수적이다. 하지만, 모니터링은 계측·전송 오버헤드로 실행 간섭을 유발할 수 있어, 경량 설계가 필수적이다. 본 연구에서는 RTEMS QDP 환경에서 RTEMS Record가 부가 메타데이터를 포함하여 전송량이 증가할 수 있다는 점에 주목하고, RTEMS 커널 API 기반의 경량 실시간 모니터링 시스템을 설계·구현하였다. 제안 방식은 태스크 전환·스택·태스크 정보 등 모니터링에 필요한 최소 항목만을 바이너리 패킷으로 구성해 링버퍼에 누적한 뒤 주기적으로 flush하는 구조를 적용하였다. 실험 결과, 제안 방식은 RTEMS Record 대비 총 전송량을 크게 감소시켰으며, 로그 출력 태스크(MONI)의 CPU 점유율도 낮아져 실행 간섭이 완화됨을 확인하였다. 다만, 실제 하드웨어에서는 UART 직렬화 지연, FIFO 처리, 인터럽트 기반 I/O 처리 비용이 오버헤드에 영향을 주기 때문에, 제안 방식을 실제 하드웨어 환경에서 동일 조건으로 추가 실험을 통해 검증할 필요가 있다.

REFERENCES

[1] C. R. Tooley et al., “The Magnetospheric Multiscale Constellation,” *Space Science Reviews*, vol. 199, pp. 23-76, 2016.

[2] A. D. Malony et al., “Performance Measurement Intrusion and Perturbation Analysis,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 3, no. 4, pp. 433-450, Jul. 1992.

[3] S. Fischmeister and P. Lam, “Time-Aware Instrumentation of Embedded Software,” *IEEE Transactions on Industrial Informatics*, vol. 6, no. 4, pp. 652-663, Nov. 2010.

[4] B. Forlina et al., “Lightweight Instrumentation for Accurate Performance Monitoring in RTOSes,” *Proc. of 2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2024.

[5] W. Chen et al., “Low-Overhead Online Assessment of Timely Progress as a System Commodity,” *Proc. of 35th Euromicro Conference on Real-Time Systems (ECRTS 2023)*, pp. 13:1-13:26, 2023.

[6] M. Craun et al., “Eliminating eBPF Tracing Overhead on Untraced Processes,” *Proc. of Workshop on eBPF and Kernel Extensions (eBPF '24)*, pp. 1-7, Aug. 2024.

[7] B. Jiang et al., “AutoTracer: A Low-Overhead Tracing Framework for Autonomous Driving System,” *Proc. of the 33rd ACM International Conference on the Foundations of Software Engineering (FSE Companion '25)*, pp. 389-399, Jun. 2025.

[8] B. Djika et al., “A POSIX/RTEMS Monitoring Tool and a Benchmark to Detect Real-Time Scheduling Anomalies,” *ACM SIGAda Ada Letters*, vol. 43, no. 2, pp. 62-68, 2024.

[9] M. Desnoyers and M. R. Dagenais, “The LTTng tracer: A low impact performance and behavior monitor for GNU/Linux,” *Proc. of Ottawa Linux Symposium (OLS)*, 2006.

[10] ftrace.txt. https://www.kernel.org/doc/Documentation/trace/ftrace.txt (accessed Jan., 08, 2026).

[11] RTOS Tracing, your way(2022). https://percepio.com/rtos-tracing/ (accessed Jan., 08, 2026).

[12] NASA, “*Software Assurance and Software Safety Standard (NASA-STD-8739.8B)*,” Tech. Rep., Sep. 2022.

[13] NASA Office of the Chief Engineer, “*NASA Software Engineering Requirements (NPR 7150.2D)*,” Tech. Rep., Mar. 2022.

[14] NASA, “*NASA Software Engineering Handbook (NASA-HDBK-2203)*,” Tech. Rep., Apr. 2020.

[15] About NASA's IV&V Program(2025). https://www.nasa.gov/about-nasas-ivv-program/ (accessed Jan., 08, 2026).

[16] Herschel & Planck. https://www.rtems.org/applications/space/herschel_and_planck/ (accessed Jan., 08, 2026).

[17] rtems_record_item_32 Struct Reference. https://docs.rtems.org/doxygen/main/structrtems__record__item__32.html (accessed Feb., 21, 2026).

[18] Juchan. Park et al., “Analysis of RTEMS SMP Support Status and Scheduling Techniques,” *Smart Media Journal*, vol. 14, no. 12, pp. 28-39, Dec. 2025.

[19] M.W. Kang et al., "Design and Implementation of a Development Automation Framework for RTEMS," *Smart Media Journal*, Vol. 14, No.9, pp. 17-24, Sep. 2025

[20] K.S. Kim et al., “Loan/Redemption Scheme for I/O performance improvement of Virtual Machine Scheduler,” *Smart Media Journal*, vol. 5, no. 4, pp. 18-25, 2016.

저 자 소 개

신대현(준회원)

2020년~현재 한남대학교 컴퓨터공학과, AI융합학과(다전공) 학사 졸업

2025년~현재 한남대학교 우주공학과 석사 재학(학석사연계과정)

<주관심분야 : 운영체제, 시스템소프트웨어, 인공위성>

박주찬(준회원)

2020년~현재 한남대학교 전기전자공학과 학사 졸업.

2025년~현재 한남대학교 우주공학과 석사 재학(학석사연계과정)

<주관심분야 : 운영체제, 시스템소프트웨어, 인공위성>

장준혁(종신회원)

2009년 서울대학교 전기·컴퓨터공학부 학사 졸업.

2015년 서울대학교 전기·컴퓨터공학 박사 졸업(석박사통합).

2021년~현재 한남대학교 컴퓨터공학과 교수

<주관심분야 : 임베디드시스템, 운영체제, 시스템소프트웨어>