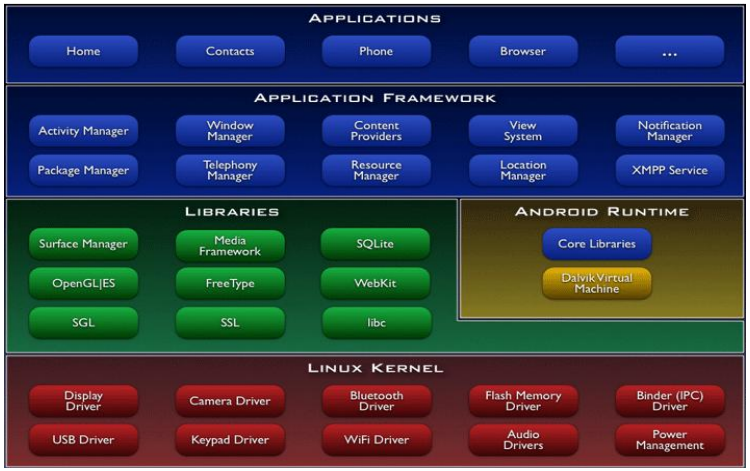
임베디드 리눅스 시스템을 위한 통합 성능 분석 도구

Integrated Performance Analysis Tool for Linux Based Embedded System

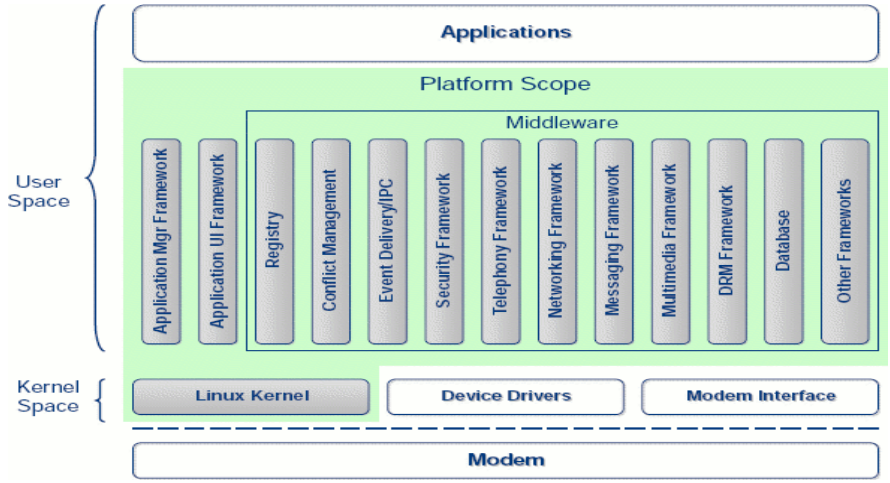
Contents

1. 개요 및 목표
2. 개발 내용
3. 활용 방향

개요



< Android Software Stack >

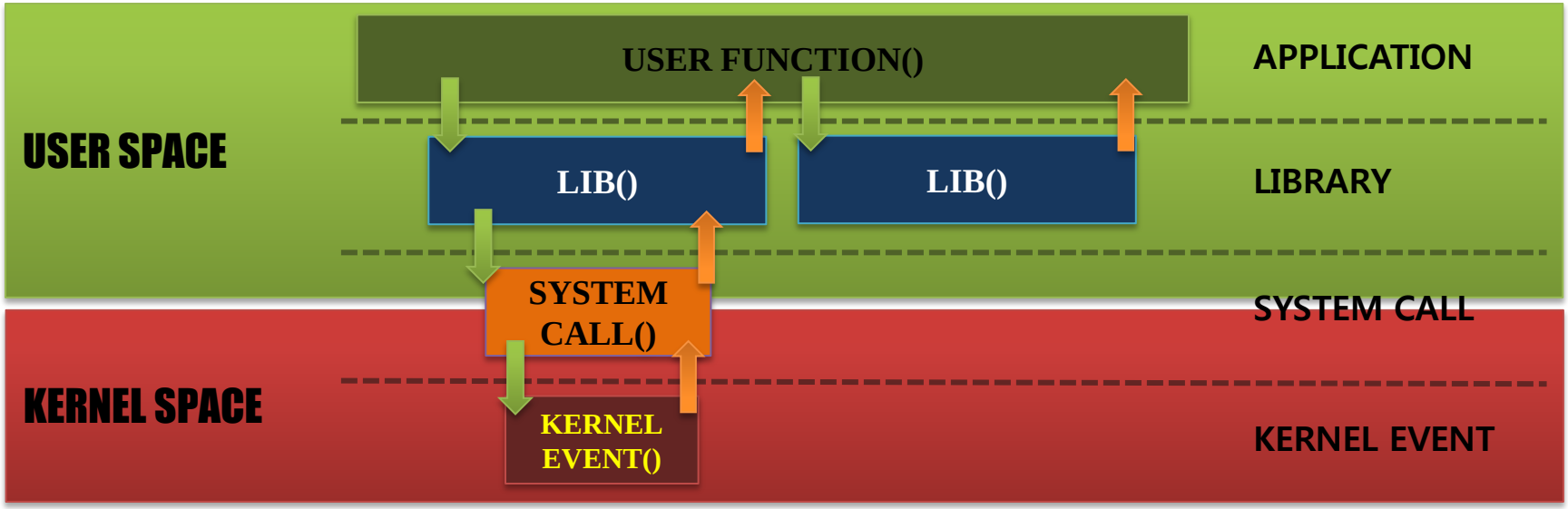


< LiMo Software Stack >

- 임베디드 시스템은 다양한 소프트웨어 계층들로 구성되어 있으며, 이 계층들을 모두 고려하여 성능상의 정확한 병목지점을 찾는 것은 어려움
  - 기존의 성능 및 전력 분석도구들은 이러한 다양한 계층들을 종합적으로 고려하지 못함

- 다양한 소프트웨어 계층을 고려하여 정확한 병목지점을 찾을 수 있는 계층별 성능 분석 도구의 요구가 증대됨

개요

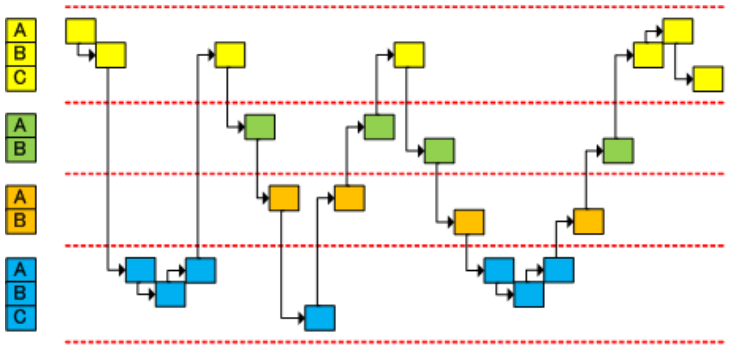


<리눅스의 계층적 소프트웨어 계층>

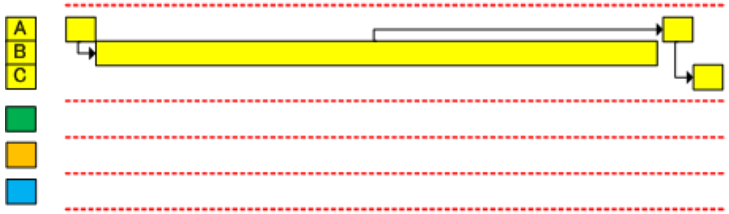
- 하나의 사용자 함수의 호출은 그 하위 계층 함수들의 호출을 야기한다.
  - 하나의 사용자 함수의 호출은 전체 시스템의 성능에 영향을 미친다.
- 하나의 함수에 대한 정확한 성능 분석은 그 하위 계층 함수들에 대한 성능 분석이 되어야만 가능하다.

관련 연구

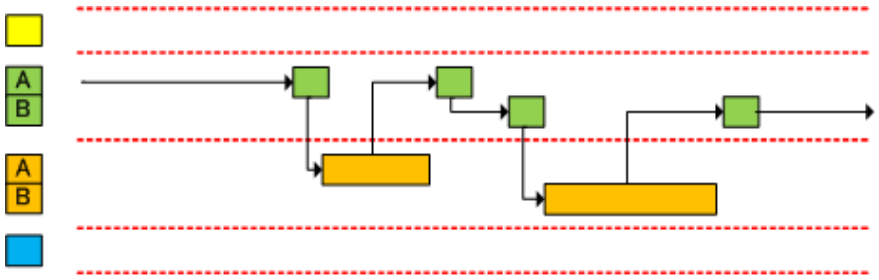
오픈 소스 기반의 성능 분석 도구



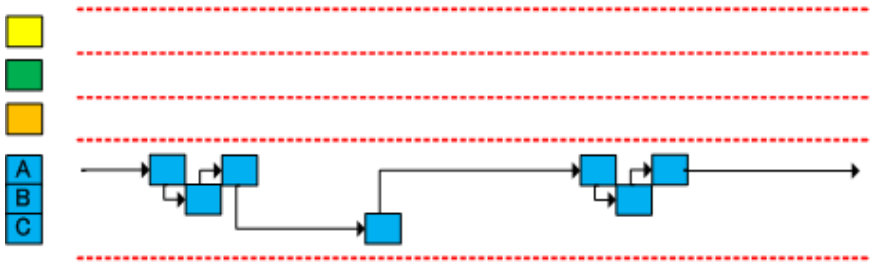
<가> 응용 소프트웨어의 계층별 실행 흐름



<나> Gprof의 응용 소프트웨어 분석 결과



<다> Strace, Ltrace 의 응용 소프트웨어 분석 결과



<라> LTTng의 응용 소프트웨어 분석 결과

User defined function (Layer)
Library function (Layer)
System call (Layer)
Kernel event (Layer)

- 한 소프트웨어 계층만을 고려하여 정확한 분석이 가능한가?

관련 연구

오픈 소스 기반의 성능 분석 도구

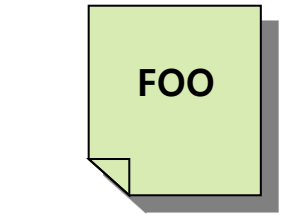
	GDB	Gprof	Strace	Ltrace	Lttng	Oprofile	Ps, top	본 도구
주목적	디버깅	사용자함수 추적	시스템호출 추적	라이브러리 함수 추적	커널이벤트 추적	커널이벤트 추적	프로세스의 상태 확인	계층간 실행흐름분석
설치를 위한 커널의 설정 및 패치	X	X	X	X	O	O	X	O
사용자정의 함수 분석	X	O	X	X	X	O	X	O
라이브러리 함수 분석	X	X	X	O	X	O	X	O
시스템 호출 분석	X	O	O	O	X	O	X	O
커널 이벤트 분석	X	X	X	X	O	O	X	O
사용자 영역 – 라이브러리 상관 관계 분석	X	X	X	X	X	O	X	O
사용자 함수 – 시스템 호출 상관 관계 분석	X	X	X	X	X	O	X	O
라이브러리 – 시스템호출 상관 관계 분석	X	X	X	X	X	O	X	O
시스템 호출 – 커널 이벤트 상관 관계 분석	X	X	X	X	X	O	X	O

관련 연구

오픈 소스 기반의 성능 분석 도구

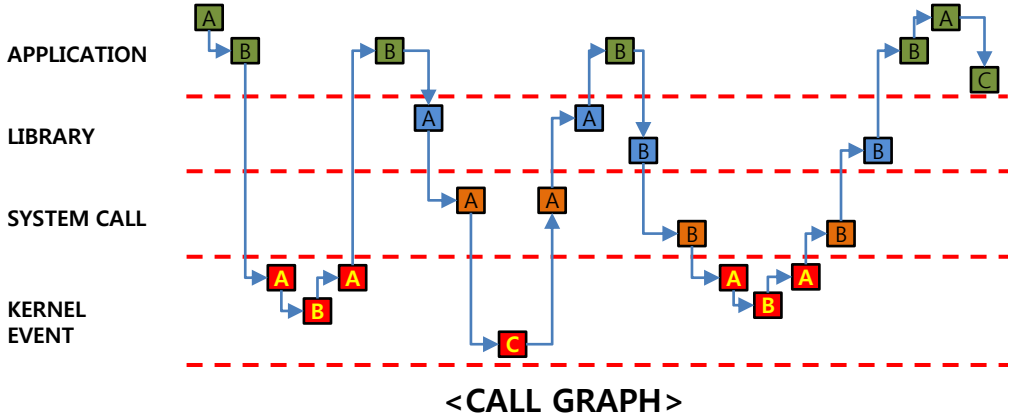
- 관련 기술들의 공통적인 특징
  - 특정 계층에 특화된 성능 분석만을 함
- 관련 기술들의 공통적인 맹점
  - 전체 시스템의 관점에서의 성능을 고려하지 않음

연구 목표

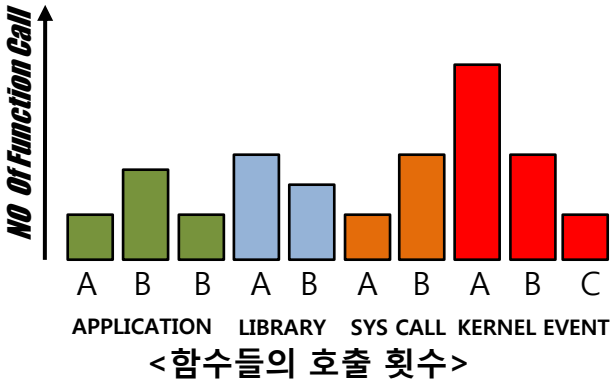


<APPLICATION 'FOO'>

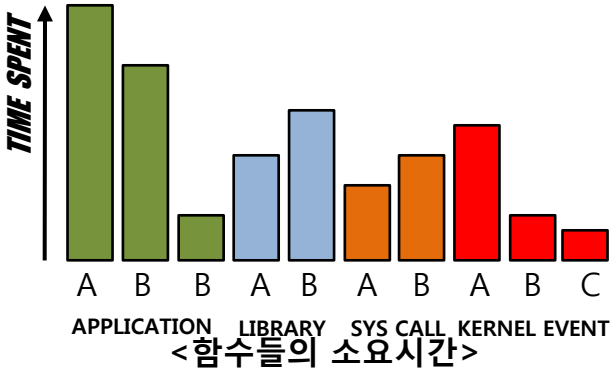
성능분석



<CALL GRAPH>



<함수들의 호출 횟수>

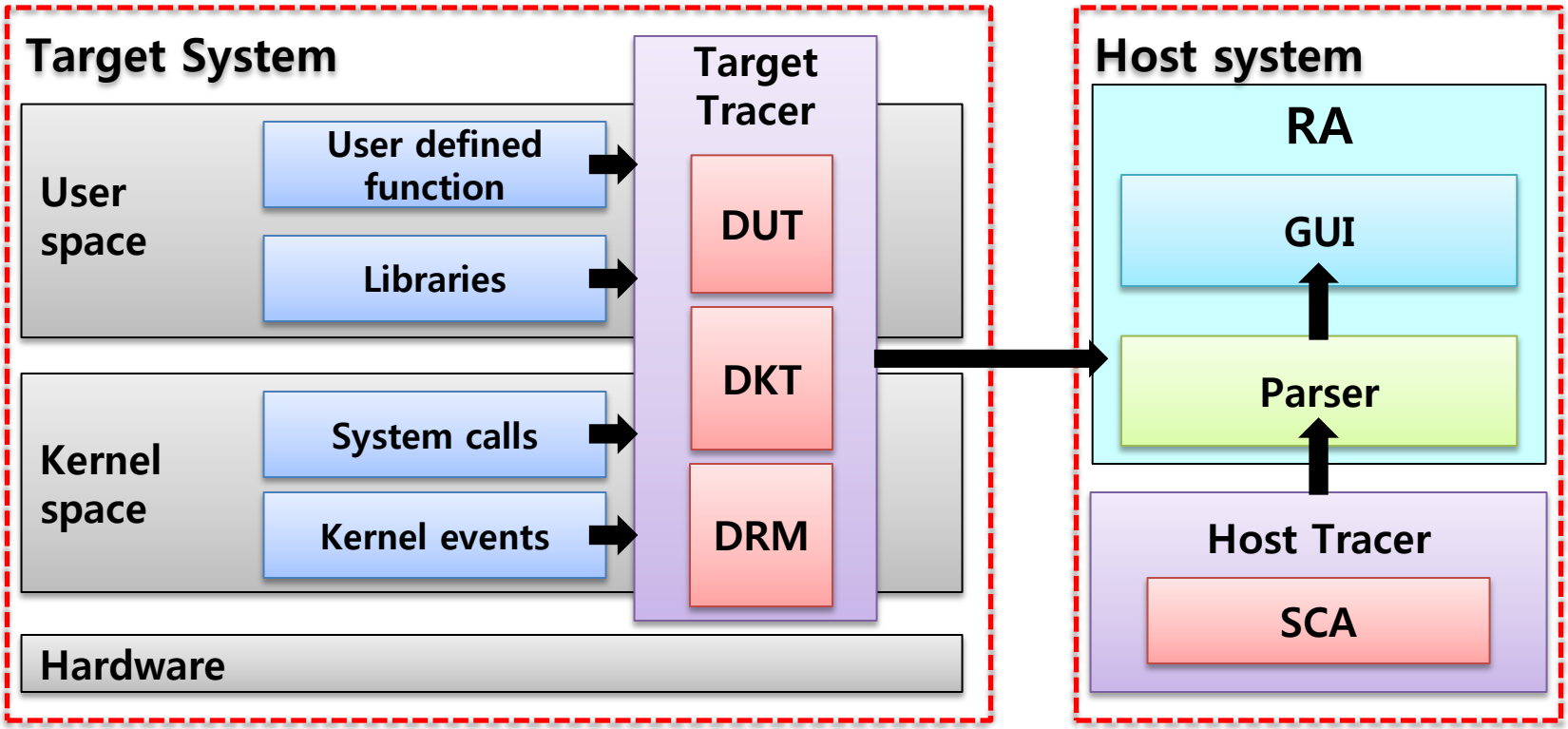


<함수들의 소요시간>

- 체계적인 성능 분석으로 성능상의 병목지점을 직관적으로 도출하는 통합 성능 분석 도구 구현

개발 내용

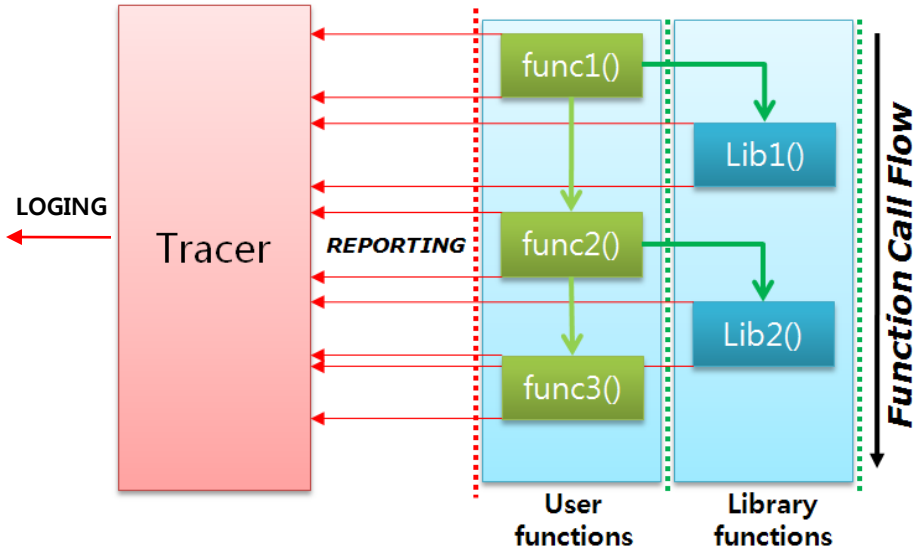
시스템 구조



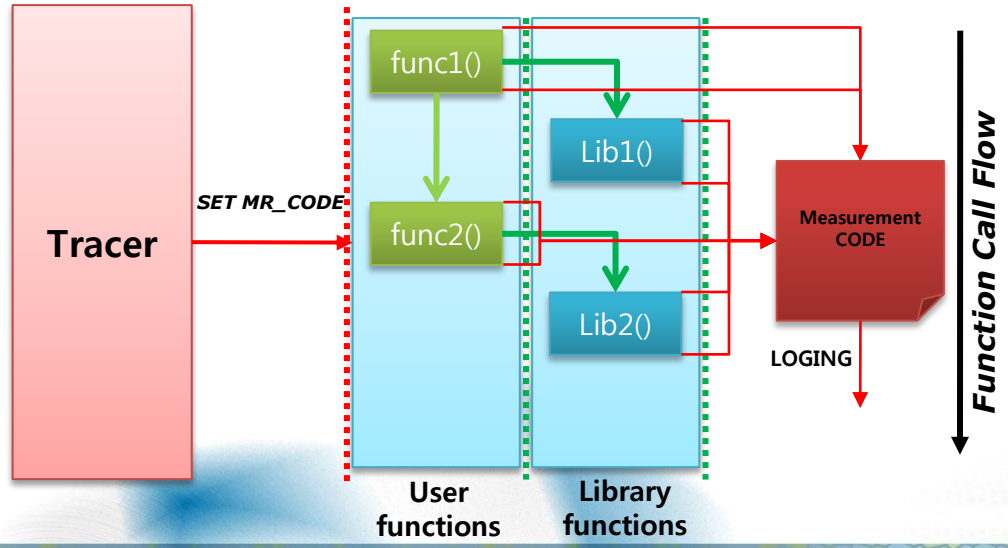
- 호스트 시스템(Host System)
  - **Host Tracer :** 어플리케이션의 심볼 테이블을 이용하여 정적 호출 관계 도출
  - **Parser, GUI :** 분석 결과들을 연동하여 사용자에게 제공
- 타겟 시스템(Target system)
  - DUT(Dynamic Userspace Tracer), DKT(Dynamic Kernelspace Tracer) : 동적으로 사용자, 커널 영역을 추적
  - DRM(Dynamic Resource Monitor) : 동적으로 리소스 모니터링

개발 내용

사용자 영역 성능 추적기 구현



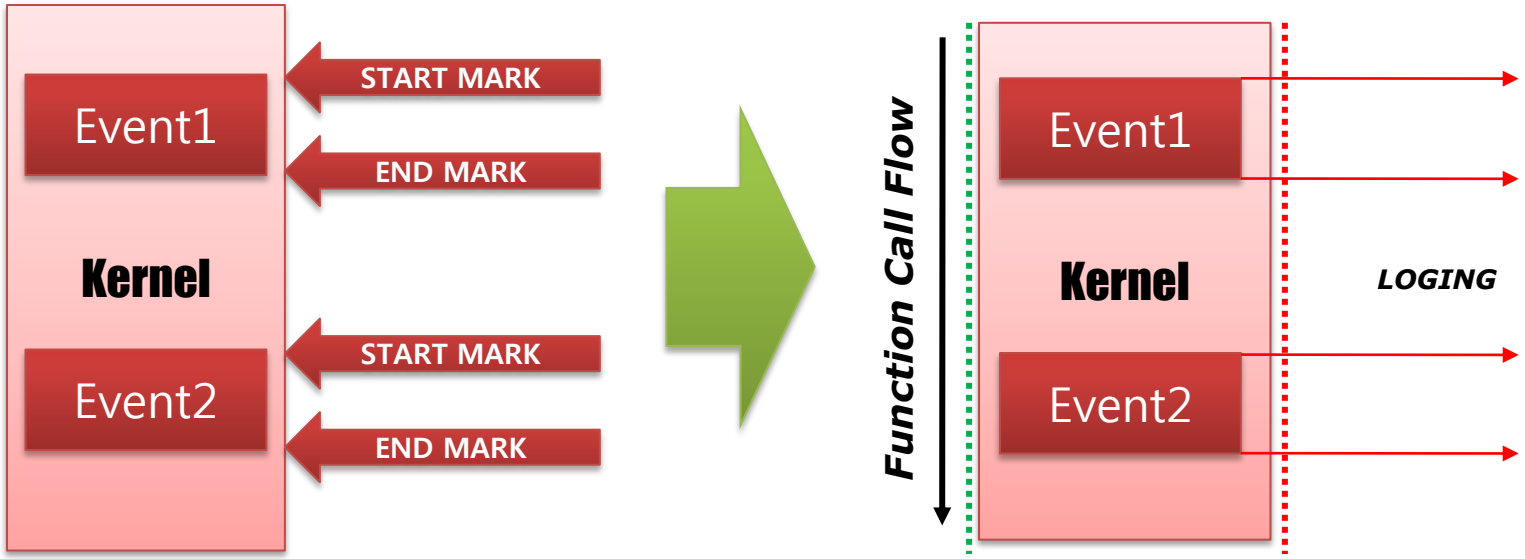
- 기존 연구
  - 소스 수정 없이, 바이너리 분석 후 중단점 삽입으로 응용 추적(Ptrace system call 사용)
    - 함수 진입, 진출 점을 추적
  - 단점
    - 성능 추적 간에 잦은 문맥 교환으로 큰 오버헤드 발생



- 본 연도 연구
  - 소스 수정 없이, 바이너리 분석 후 런타임에 Measurement Code(성능 추적 코드) 삽입(Ptrace system call 사용)
    - 함수 진입, 진출 점을 추적
  - 장점
    - 어플리케이션 수행간 문맥 교환이 발생하지 않음(오버헤드 문제 개선)

개발내용

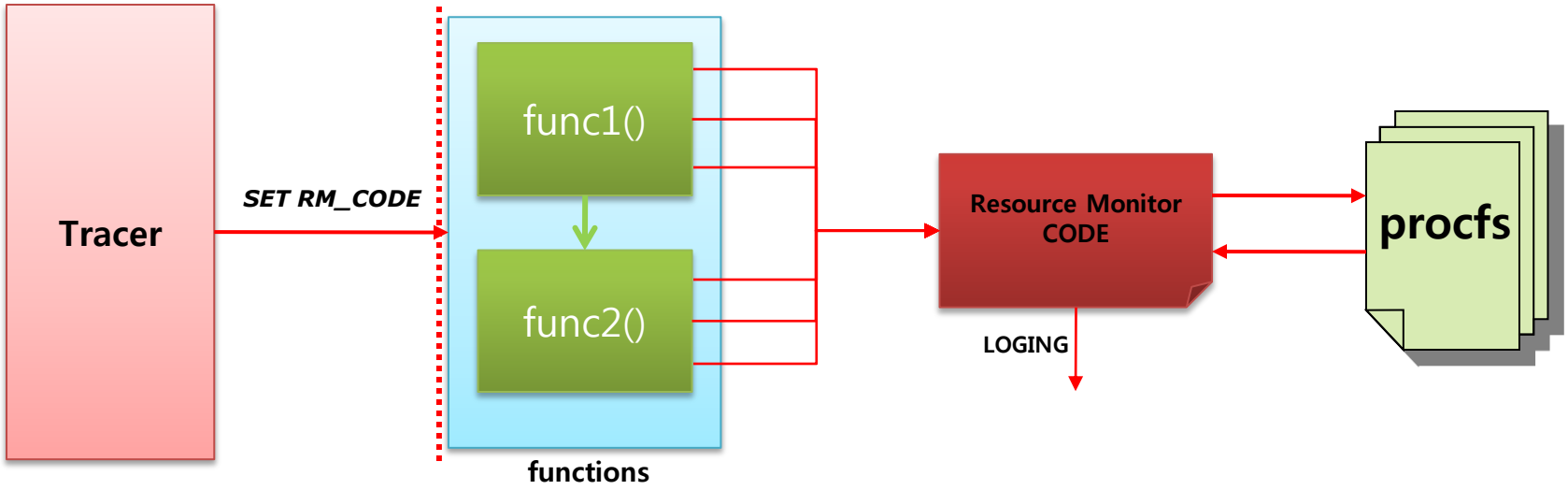
커널 영역 성능 추적기 구현



- 측정 코드 삽입을 통한 시스템 호출, 커널 이벤트를 추적
  - 리눅스 커널 추적기 LTTng 이용
    - 시스템 호출과 커널 이벤트의 핸들러에 측정 코드인 마커 삽입
    - 측정 코드 삽입 후 실행 시 핸들러의 마커가 추적 정보를 기록

개발내용

리소스 모니터 구현



- 런타임에 각 함수에 RM_CODE(Resource Monitor Code) 를 삽입하고, 어플리케이션이 구동되어 각 함수 안의 RM_CODE 가 구동 될 때 RM_CODE 가 procfs 로부터 시스템의 리소스 상태 정보를 읽어와서 로깅 함

개발내용

Graphic User Interface

- 사용의 편의성을 제공하고 성능 병목 지점을 직관적으로 도출 할 수 있는 성능 분석 GUI 개발
  - 그래프 등의 직관적인 형식으로 성능 분석 결과 도출
    - 쉽게 성능 병목 지점을 발견할 수 있도록 도움
  - 성능 분석 결과의 체계적인 저장 관리
    - 성능 개선 전과 성능 개선 이후의 결과 비교 분석을 용이하게 함

활용 방향

- **오픈 소스 커뮤니티와 현업 개발자의 테스트를 통한 개선**
  - 현업 개발자의 요구를 반영하여 필요한 성능 분석 요소 추가
- **다양한 플랫폼으로의 이식**
  - 다양한 리눅스 커널 버전용 패치 개발
  - ARM 기반의 여러 플랫폼 지원
- **리눅스 기반의 플랫폼용 성능 분석 공개 소프트웨어로 활용**
  - Android, LiMo, GPE, Opie 등 리눅스 기반 플랫폼으로의 이식 및 활용