Linux Kernel Changes from v4.12 to v4.15

max.byungchul.park@gmail.com 박병철

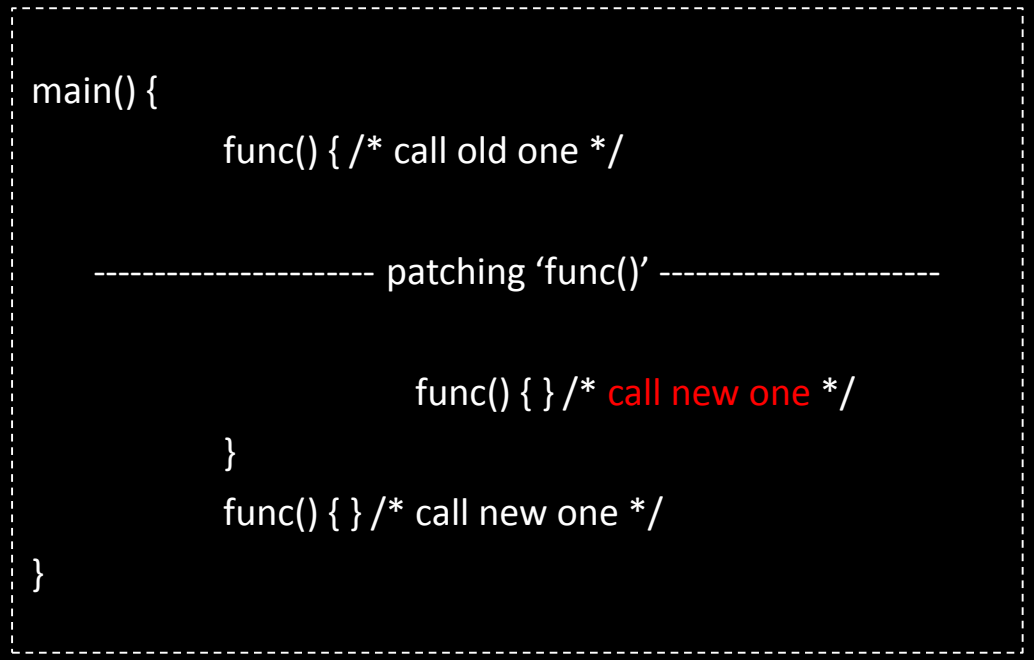
hyc.lee@gmail.com 이현철


v4.12

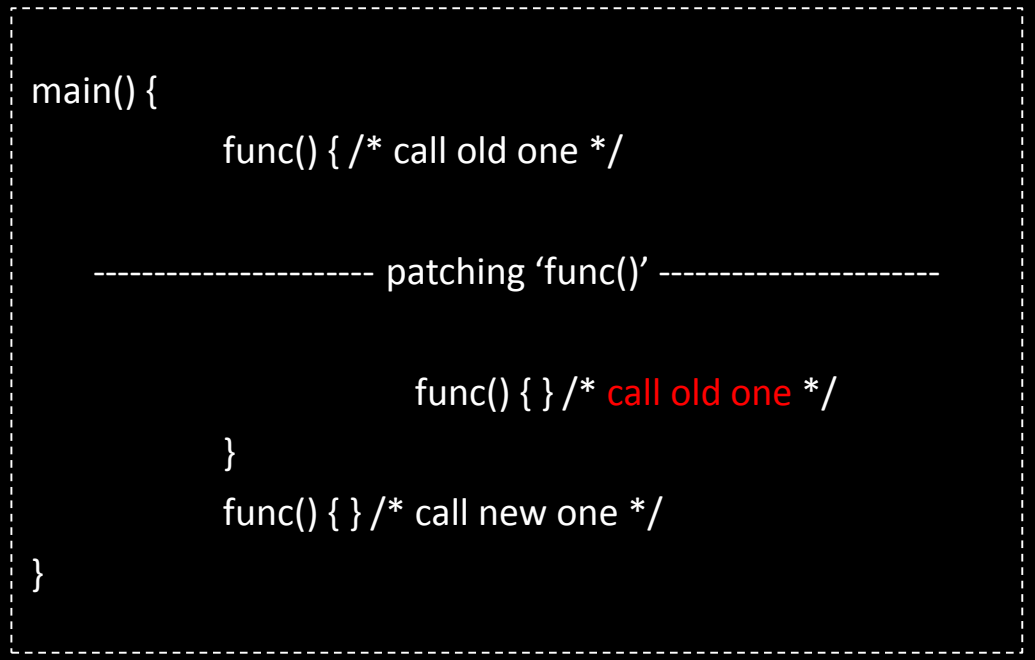
(v4.12) Live patch – consistency model

- Live patch?
  - 실행 도중에 커널의 중요한 변경 사항(주로 보안 패치)들을 적용하여 재부팅 없이도 계속 서비스가 가능하도록 지원하려는 기능
- Consistency model?
  - 패치 대상 함수를 실행 하고 있는 프로세스가 있다면 프로세스 별로 해당 함수 실행이 종료될 때까지 기다렸다가 패치를 반영하는 방식

(v4.12) Live patch – consistency model



Before



After

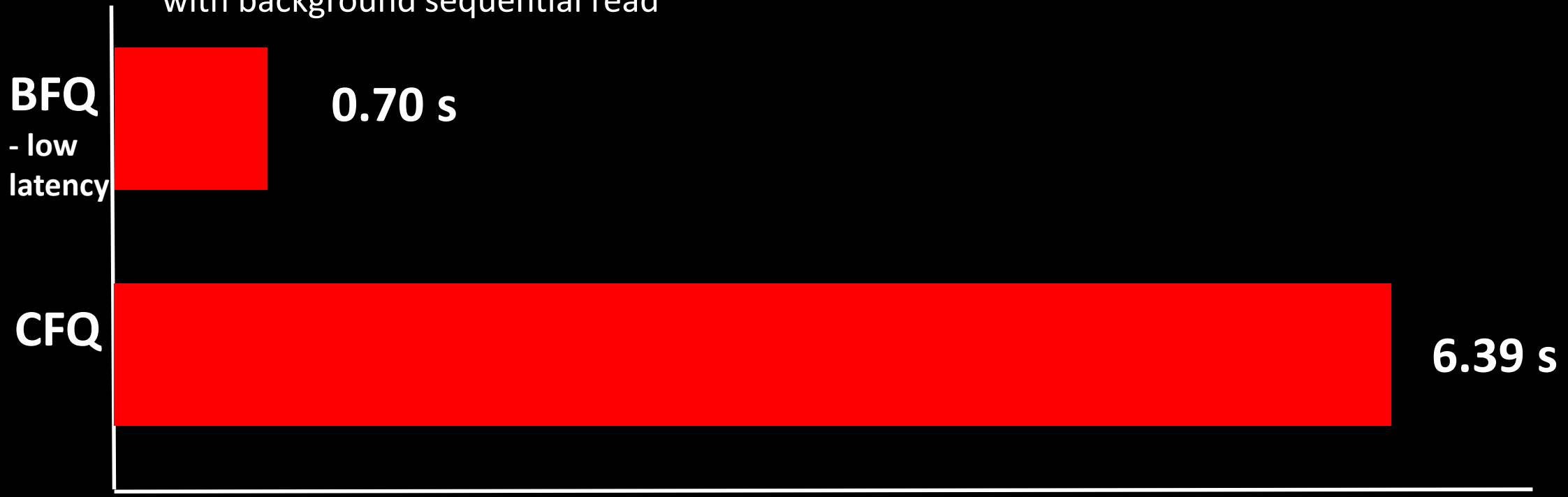
(v4.12) BFQ (Budget Fair Queuing)

- 새로운 I/O 스케쥴러
- 응용 프로그램에게 가중치 만큼의 throughput을 보장하고, bounded per-request delay를 보장
- Interactive 응용 프로그램이나 soft real-time 응용 프로그램에게 최소한의 delay를 보장
- 전체 I/O throughput을 최대화

(v4.12) BFQ (Budget Fair Queuing)

* LibreOffice Writer Start-up Time

with background sequential read

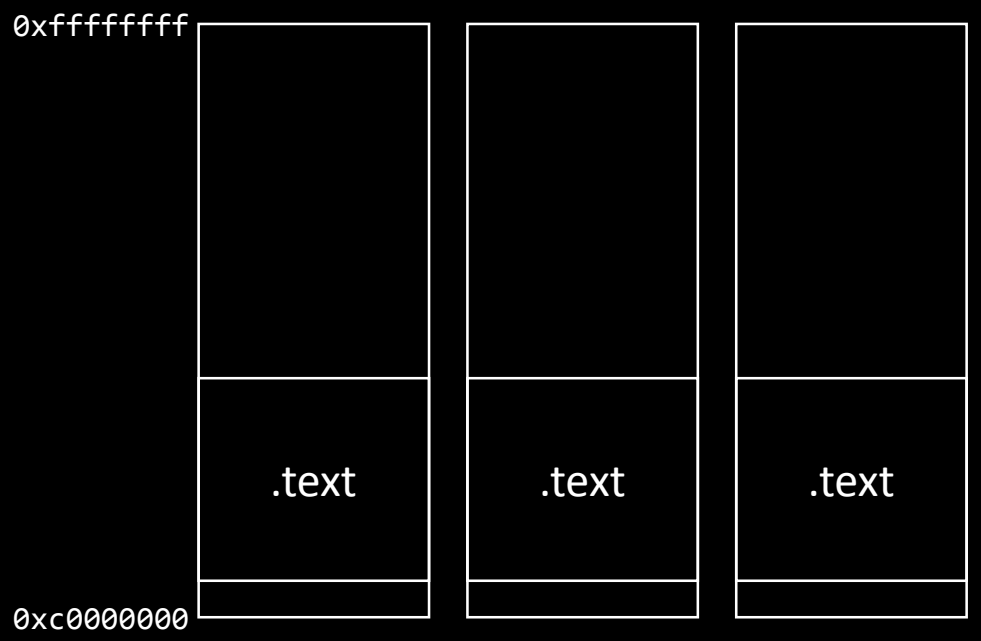


* 출처 https://www.phoronix.com/scan.php?page=article&item=linux-415-iosched

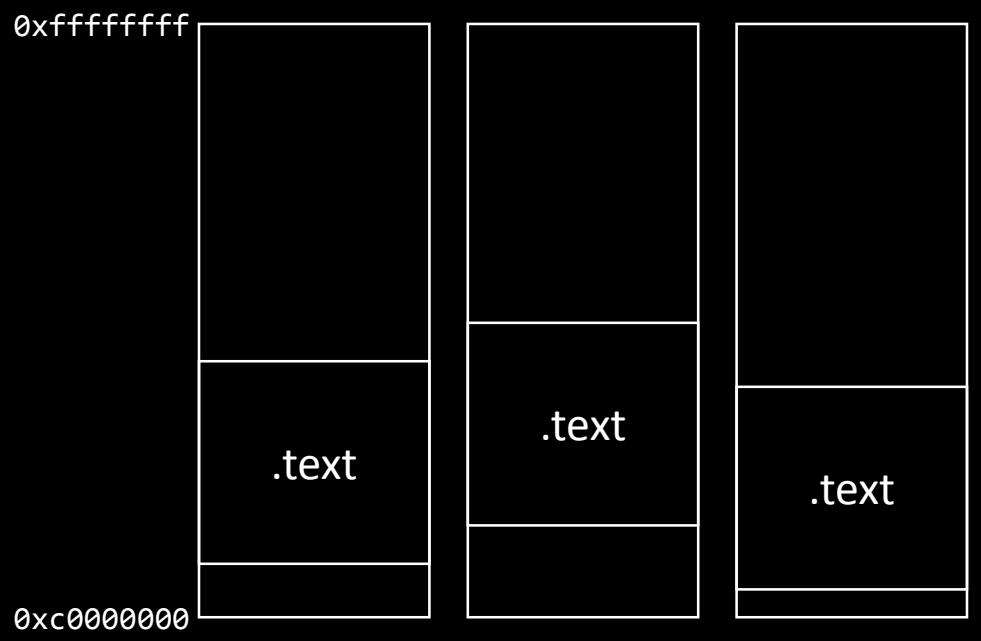
(v4.12) kASLR – by default on x86

- ASLR?
  - 프로그램이 올라가는 주소를 매번 변경하여, 설사 악의적인 사용자가 한 프로세스의 메모리 주소를 알아냈다 하더라도 다른 프로세스에 대해서는 동일한 방법으로 공격하지 못하도록 하기 위한 기법
- kASLR?
  - 부팅될 때마다 커널 기본 주소를 변경하여, 한 시스템이 공격받더라도 다른 시스템은 동일한 방법으로 공격받지 못하도록 보호하기 위한 기법

(v4.12) kASLR – by default on x86



Before



After


v4.13

(v4.13) Nowait Asynchronous I/O

- IOPS를 높이기 위해서, Direct 쓰기는 아래와 같은 블록킹 상황에서 EAGAIN 에러를 반환
  - 락을 잡을 수 없는 경우
  - 블록 할당이 필요한 경우
  - Write-back이 필요한 경우
  - 블록 디바이스가 congested 한 경우
- (v4.14) nowait buffered read
  - 요청한 데이터가 캐시에 없는 경우에 EAGAIN 에러를 반환

(v4.13) Enhanced Write-back Error Reporting

- Write-back 중에 에러가 발생하면, 첫 번째로 호출된 fsync만 에러 코드를 반환

- 에러 발생 전에 그 파일을 오픈한 모든 프로세스에게 fsync 에러를 반환하도록 변경

(v4.13) Ext4 Updates

- 10M 개 이상의 파일을 저장할 수 있는 Large directory feature
    - 디렉토리의 크기가 2GB 이상
    - Htree의 깊이가 3
- 4KB 이상을 저장할 수 있는 Large xattr feature


v4.14

(v4.14) Cross-release – enhancing lockdep

- Lockdep?
  - 스핀락, 뮤텍스와 같은 동기화 메커니즘을 잘못 사용하여 시스템이 데드락에 빠지기 전에 데드락 가능성을 사전에 감지해서 개발자 또는 관리자에게 해당 정보를 제공하는 기능
- Cross-release?
  - Lockdep은 제한적인 락(예, 스핀락, 뮤텍스)에 대해서만 데드락을 감지하는 한계가 있었는데, 새로운 알고리즘으로 이를 극복하고 모든 동기화 메커니즘에 대해서 데드락이 감지될 수 있도록 구현한 기능

(v4.14) Cross-release – enhancing lockdep

Thread 1	Thread 2
mutex_lock A	mutex_lock A
wait_for_completion B	mutex_unlock A
mutex_unlock A	complete B

Impossible to detect this!

Before

Thread 1	Thread 2
mutex_lock A	mutex_lock A
wait_for_completion B	mutex_unlock A
mutex_unlock A	complete B

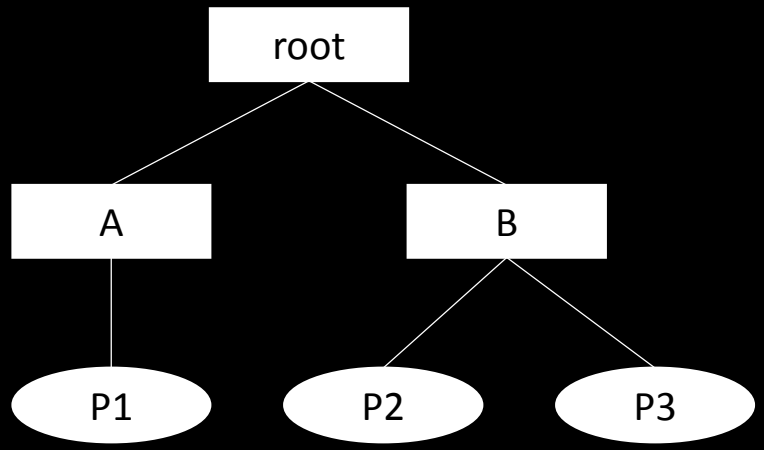
Possible to detect this!

After

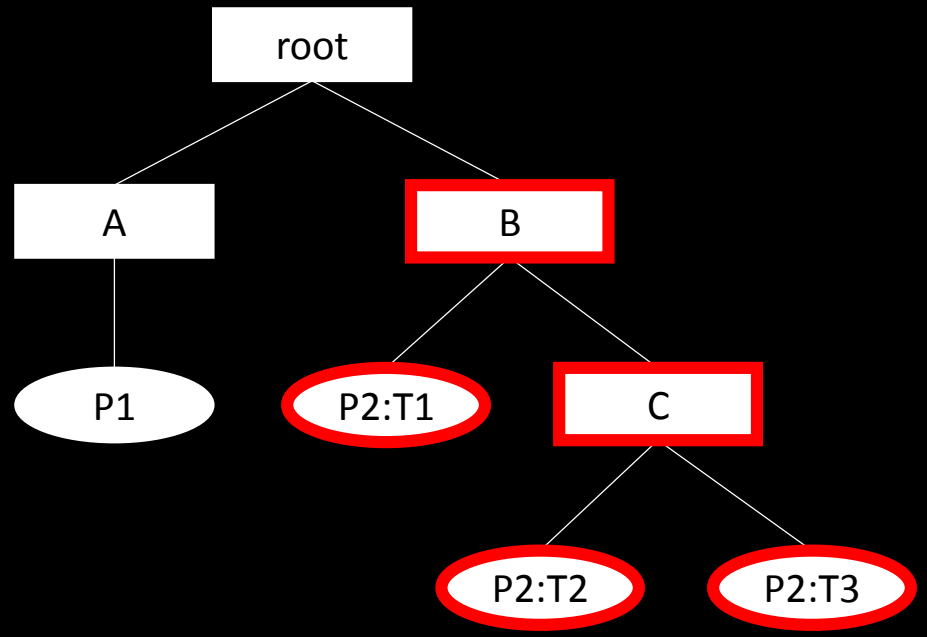
(v4.14) Cgroup v2 – thread mode

- Cgroup v2?
  - 프로세스를 트리 구조로 묶어서 시스템 자원을 효과적으로 관리하기 위해 cgroup을 사용했었으나, 자원 관리자 별로 별도의 트리를 구성하는 문제점 때문에 v2가 제안되어 시스템에 하나의 트리만 유지하고 모든 자원 관리자가 이를 공유하는 형태로 재설계
- Thread mode?
  - Cgroup v2는 스레드 단위로는 묶을 수 없고 (프로세스 단위만 허용) 자식으로 프로세스와 다른 그룹을 동시에 가질 수 없다는 제약을 가지고 있었는데, 위 제약을 따를 수 없는 cpu 자원 관리자를 위해 위 제약을 따르지 않아도 되는 thread mode 도입

(v4.14) Cgroup v2 – thread mode



Before



After

(v4.14) Zstandard compression

- Zlib 만큼 압축률이 높고 속도는 zlib 보다 훨씬 빠르며, 해제 속도는 lzo에 근접하는 알고리즘

- Zram에서는 deflate를 대체하고, Squashfs와 Btrfs가 지원

(v4.14) Zstandard compression

* Core i7-6700 @ 4.0GHZ on Silesia curpus; 4KB unit

Name	Ratio	Compression	Decompression
Zstd 1.3.4 -1	2.877	470 MB/s	1380 MB/s
Zlib 1.2.11 -1	2.743	110 MB/s	400 MB/s
Lzo1x 2.09 -1	2.108	650 MB/s	830 MB/s
Lz4 1.8.1	2.101	750 MB/s	3700 MB/s

* 출처 https://github.com/facebook/zstd/blob/dev/README.md

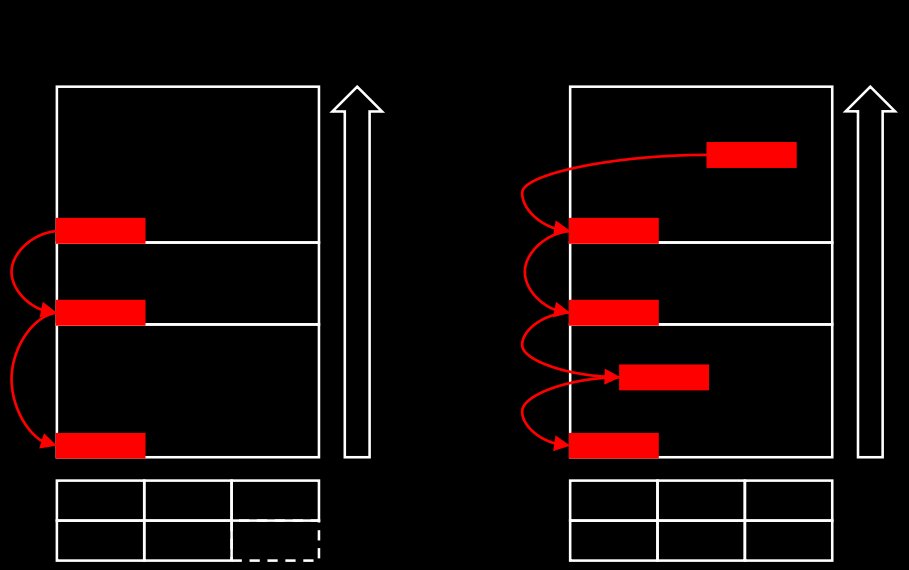
* Core i7 @ 3.1GHZ; Squashfs; 128KB unit

Name	Ratio	Compression	Decompression
Gzip	2.92	15 MB/s	128 MB/s
Lz4	2.12	94 MB/s	218 MB/s
Xz	3.43	5.5 MB/s	35 MB/s
Zstd 15	3.13	11.4 MB/s	224 MB/s

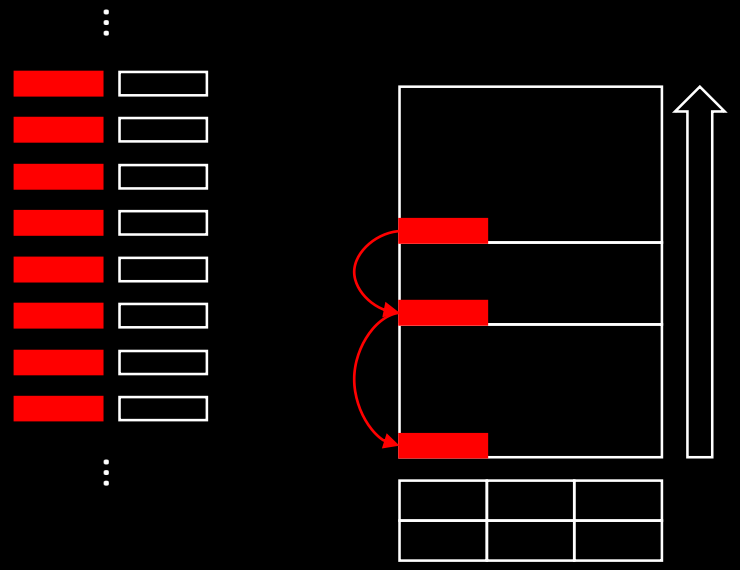
(v4.14) ORC unwinder

- Unwinder?
  - 시스템이 죽는 상황(oops)에서 또는 디버깅 목적으로 백트레이스를 추출하는 기능을 unwinder라고 하고, 지금까지는 스택 전체를 뒤져서 코드영역 주소 값이 있으면 무조건 추출하는 방식을 사용하거나 프레임 포인터를 사용해서 추출
- ORC unwinder?
  - DWARF를 사용한 백트레이스 추출 방법을 커널에서 사용하기에는 너무 무겁고 커널의 특수 상황을 반영하지 못할 뿐만 아니라 오류를 가지고 있어서 누군가에 의해 제안되었지만 메인라인에 반영되지 못했고, 이를 경량화하고 커널의 특수 상황을 반영한 ORC 방식이 채택

(v4.14) ORC unwinder



Before



After


v4.15

(v4.15) Security – meltdown/spectre

- 미리 읽기 (Instruction level)
  - 성능향상을 위해 추후에 사용할 메모리 값을 미리 읽어 캐시와 임시 레지스터에 저장해 놓는 방식을 많이 사용한다. 실제 그 값이 사용되면 그대로 사용하면 되고 어떠한 이유로 사용되지 않으면 버리면 그만이다.
  - 여기서 문제는, 미리 읽은 데이터를 버릴 때에 임시 레지스터에 있는 내용은 버리지만 캐시에 올라온 데이터는 그대로 둔다는 점이다. 캐시에 올라온 데이터를 분석하면 미리 읽기한 데이터 값을 알아 낼 수 있다.

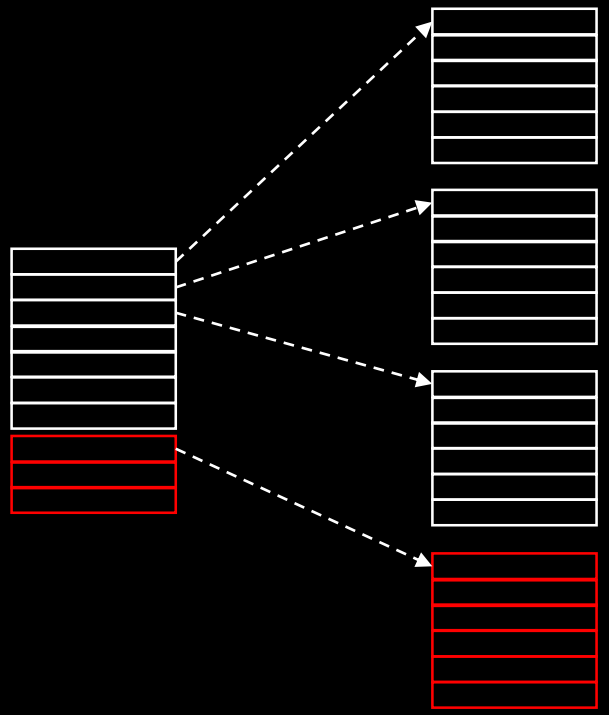
(v4.15) Security – meltdown/spectre

- Meltdown?
  - 미리 읽기보다 permission 체크를 늦게 하는 아키텍쳐에서, 사용자 코드가 커널 영역 미리 읽기를 유도한 후 (결국엔 permission 검사에서 실패하여 버려짐) 캐시를 분석하여 미리 읽은 값을 찾아내는 해킹 방법
- Spectre(v2)?
  - 인접한 cpu의 indirect 분기를 일시적으로 제어할 수 있는 하드웨어 버그(?)를 이용해서 해커가 원하는 코드(미리 읽기 코드)로 분기하게 하여 미리 읽기를 유도한 후 캐시를 분석하여 미리 읽은 값을 찾아내는 해킹 방법

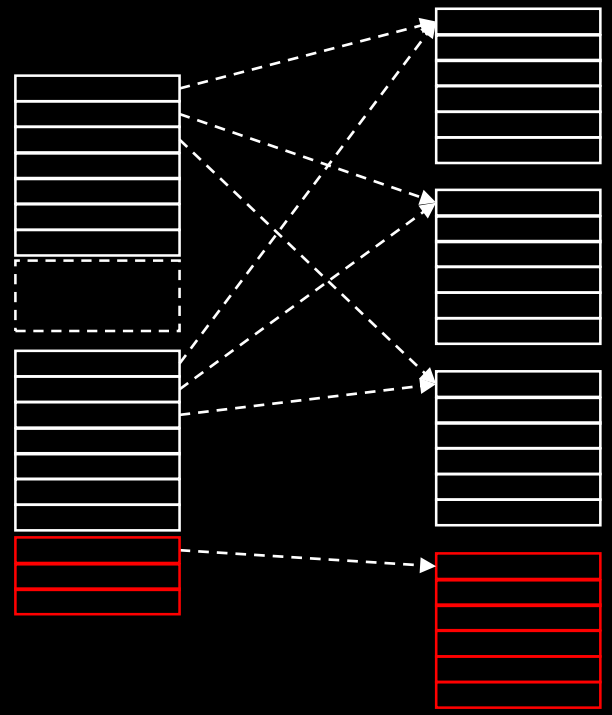
(v4.15) Security – meltdown/spectre

- Meltdown mitigation
  - 사용자 코드에서 커널 영역 미리 읽기를 수행하지 못하도록 사용자 영역 페이지 테이블과 커널 영역 페이지 테이블을 분리
- Spectre(v2) mitigation
  - 컴파일러를 수정하거나 어셈블리 코드를 수정해서 indirect 분기를 사용하지 않고 대신 retpoline을 사용하도록 수정

(v4.15) Security – meltdown



Before



After

(v4.15) Security – spectre

```
jmp     *%rax
```

Before

```
        call    2f
1:      pause
        jmp     1b
2:      mov     %rax, (%rsp)
        ret
```

After

(v4.15) Cramfs updates

- Obsoleted에서 maintained 상태로 변경
- Squashfs 보다 코드 크기가 작고 메모리 사용량도 적어, 메모리가 부족한 임베디드 시스템에 적합
- XIP 지원을 위해서 블록 장치나 MTD를 건너 뛰고 물리 주소를 접근 하는 기능, 압축/비압축 블록 기능을 추가

(v4.15) Cgroup v2 – cpu controller

- Cpu controller?
  - 프로세스들에게 cpu 자원 할당을 제어하는 자원 관리자
- What this patch does?
  - Cpu 자원 제어를 위한 cgroup v2 인터페이스 구현 완료

(v4.15) Cgroup v2 – cpu controller

Cgroup v2를 이용한 cpu 자원 제어

방법 없음

Before

Cgroup v2를 이용한 cpu 자원 제어

가능

After

Q ?