리눅스 커널 변경 사항 (v6.5 ~ v6.10)

LG전자 이현철
hyc.lee@gmail.com

목차

- core/mm
  - Deprecate the SLAB allocator (v6.5)
  - EEVDF (Eligible Earliest Virtual Deadline First) scheduler (v6.6)
  - Deadline servers for better realtime scheduling (v6.8)
- io uring
  - Support for user allocated memory for rings/sqes (v6.5)
  - Add support Multishot reads (v6.7)
  - Add ftruncate support (v6.9)
- filesystem /block layer
  - API to report block device events properly to holders (v6.5)
  - Faster FUSE I/O (v6.9)
- etc
  - New cachestat(2) system call (v6.5)
  - Scope-based resource management for the kernel (v6.5)
  - Memory allocation profiler for kernel (v6.10)
  - mseal(2) to protect memory from unwanted modifications (v6.10)
- kernel CVE

core/mm

Deprecate the SLAB allocator (v6.5)

- kmalloc 구현체인 SLUB, SLAB 중에서 SLAB을 삭제할 예정이고, deprecated 로 상태를 변경
  - SLOB은 v6.4에서 제거됨
- 이유는
  - 각 할당자 사이에 존재했던 성능상의 차이점이 크게 완화됨
  - kmalloc API에 신규 기능을 추가할 때마다 각 할당자마다 구현이 필요함
  - SLUB을 제외한 할당자는 사용이 드물고, 유지 보수가 제대로 되지 않음
- 그리고 v6.8에서 삭제됨

EEVDF (Earliest Eligible Virtual Deadline First) scheduler (v6.6)

- CFS의 단점
  필요한 때에 빠르게 실행되어야 하는 태스크에 대한 고려가 없음
- CPU 시간을 태스크들 사이에 공정하게 배분하면서, latency에 민감한 태스크를 짧게 자주 실행
- eligible 태스크들 중에서 가장 빠른 virtual deadline을 갖는 태스크를 선택

EEVDF (Earliest Eligible Virtual Deadline First) scheduler (v6.6)

- eligible
  - 실제 할당된 시간 <= 할당 됐어야 했을 시간
  - se->vruntime <= 모든 태스크 vruntime의 weighted average
- virtual deadline
  - eligible time + time slice
  - se->vruntime + time slice
  - 지연 시간에 민감한 태스크는 짧은 time slice (by sched_setattr or nice?)

Deadline servers for better realtime scheduling (v6.8)

- 비정상적인 realtime 태스크가 CPU를 100% 점유해서 시스템을 hang 으로 만들 여지가 있음
- realtime throttling
  - CPU 시간의 5%를 비 realtime 태스크에게 할당

    ```
    # echo 1000000 >  /proc/sys/kernel/sched_rt_period_us
    # echo  950000 > /proc/sys/kernel/sched_rt_runtime_us
    ```

  - 그런 태스크가 없는 경우에도 realtime 태스크의 CPU 시간이 95% 로 제한됨

Deadline servers for better realtime scheduling (v6.8)

- starvation이 발생하면 낮은 우선순위 태스크를 실행할 수 있는 데드라인 태스크(deadline server) 인프라를 도입

```
struct sched_dl_entity
+  dl_server_pick_f              server_pick
+  dl_server_has_tasks_f    server_has_tasks

func pick_task_dl(rq)
  dl_se = pick_next_dl_entity(dl_rq)
+  if (dl_server(dl_se)) p = dl_se->server_pick(dl_se)
```

Deadline servers for better realtime scheduling (v6.8)

- 사용 예) Fair deadline server
  - runnable인 NORMAL / IDLE / BATCH인 태스크가 있으면 CPU 시간의 5%를 할당

```
struct rq
+  struct sched_dl_entity     fair_server

+  func fair_server_init(rq)
+    // rq->fair_server를 20 ticks 주기로 1 tick 실행되도록 정의

+  func fair_server_pick(dl_se)
+    return pick_next_task_fair(dl_se->rq, NULL, NULL);
```

io uring

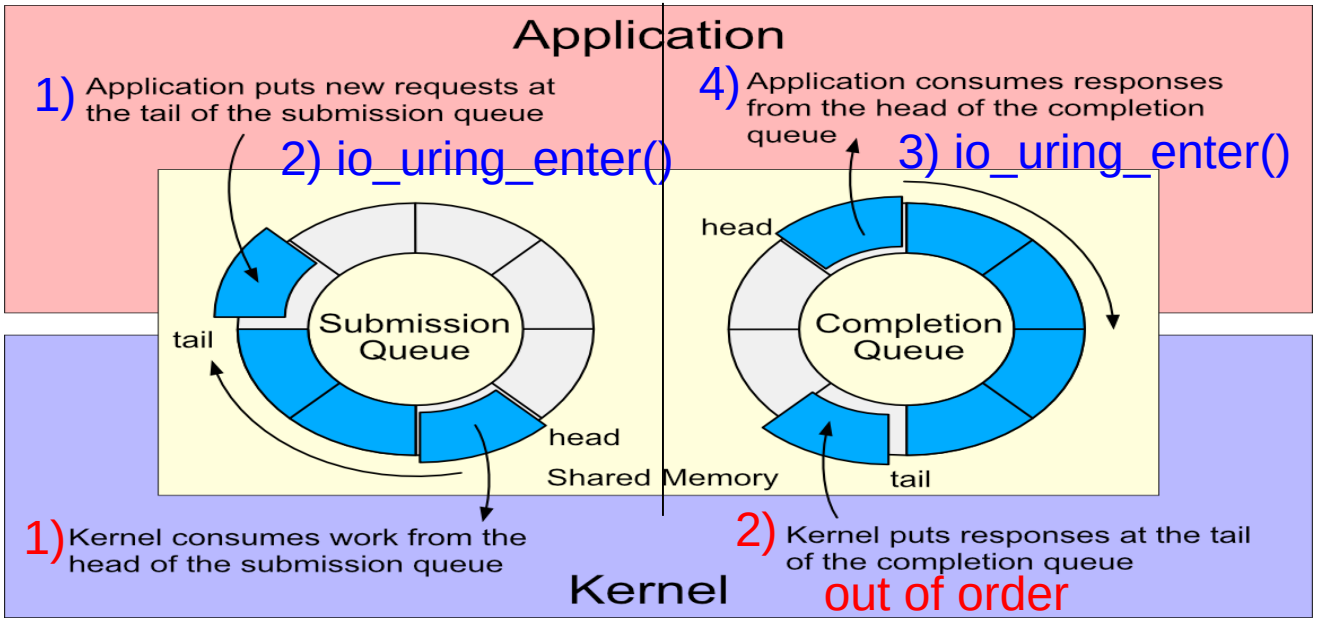
io_uring

- 비동기 I/O를 위한 API로 커널 v5.1에 처음 도입
- 사용자 / 커널 사이에 공유 메모리인 링 버퍼를 통해서 I/O를 수행

0) io_uring_setup() and mmap

사용자)

커널)



https://developers.redhat.com/articles/2023/04/12/why-you-should-use-iouring-network-io

io_uring

- 지원하는 operations
  - 다양한 file I/O operations, socket I/O operations, 미리 등록한 버퍼 사용, chains of IO 요청, time out, multi shot, waitpid, futex, …
- Synchronous I/O API 대비 장점
  - system call 호출과 context switch 횟수가 줄어듬 -> 성능 향상
  - I/O 진행되는 동안에 다른 작업 수행 -> 자원의 utilization 증가
  - 쓰레드, 버퍼 등의 갯수가 줄어듬 -> 적은 자원 사용
- 기존 비동기 I/O API 대비 장점
  - system call의 횟수와 인자의 copy 량이 줄어듬 -> 성능 향상

io_uring: 업데이트

- Support for user allocated memory for rings/sqes (v6.5)
  - 사용자가 할당한 huge pages로 SQ/CQ를 사용할 수 있게 해서 TLB miss를 줄임
- Add support Multishot reads (v6.7)
  - pipe 같은 파일에 하나의 읽기 요청만 등록하고, 데이터가 읽힐 때마다 완료 이벤트를 발생시킬 수 있음
  - accept(2) / recv(2) / poll(2) 등은 multishot 모드를 이미 지원
  - 버퍼 풀 등록이 필요
- Add ftruncate support (v6.9)

file system / block layer

API to report block device events properly to holders (v6.5)

- block device가 제거될 때에 holder에게 event를 전달하는 internal API가 추가됨
- holder가 blk_holder_ops 중 mark_dead를 구현하고 등록
- 파일 시스템은 dcache shrink, inode invalidate, 그리고 메타 데이터 를 정리

API to report block device events properly to holders (v6.5)

filesystem)

(1) 마운트 시에 mark_dead() 등록
func __ext4_fill_super()
bdev_file_open_by_dev(..., &fs_holder_ops)

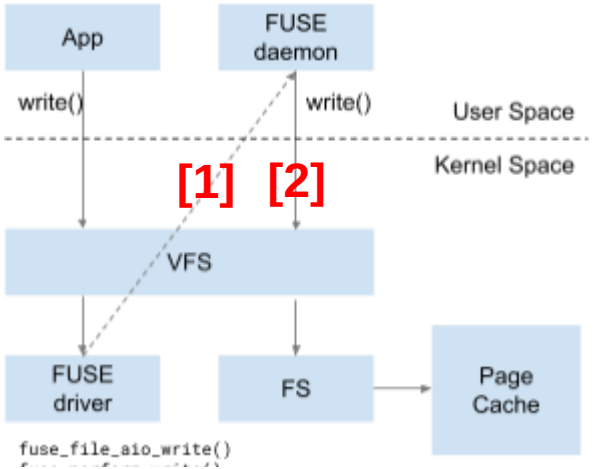
(3) mark_dead()가 호출됨
func fs_bdev_mark_dead
// shrink dcache, invalidate inodes, …
sb->s_ops->shutdown

block device)

(2) 디바이스가 제거될 때에 mark_dead() 호출
func del_gendsik()
bdev->bd_holder_ops->mark_dead()

Faster FUSE I/O (v6.9)

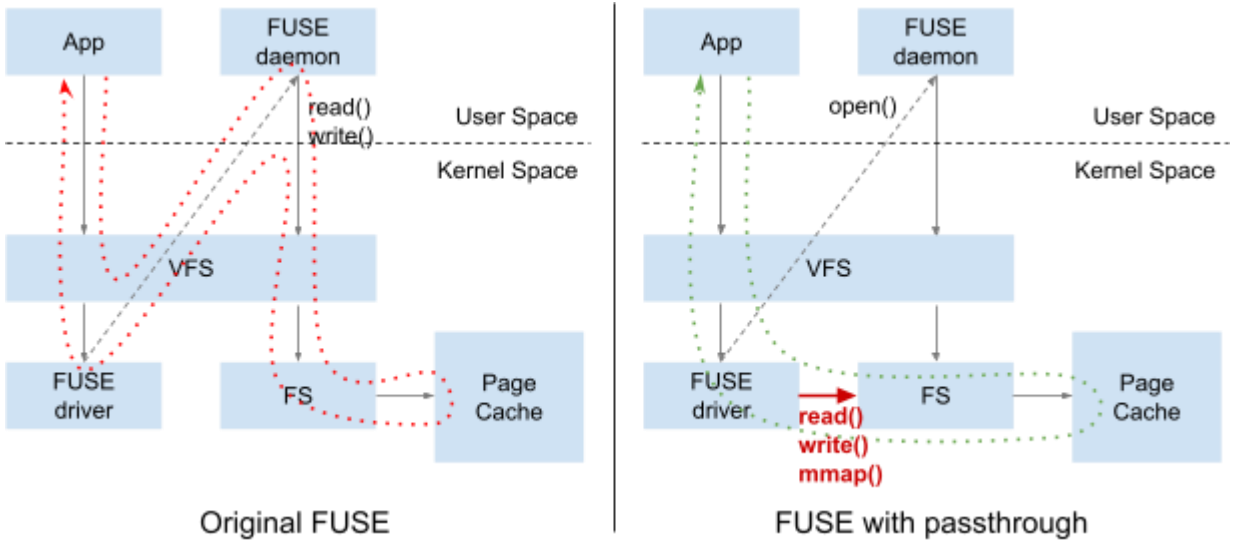
- 사용자 공간에서 파일 시스템을 구현하고 동작할 수 있게 하는 프레임워크
- 컨텍스트 스위칭[1], 사용자/커널 모드 변경[2]으로 인한 낮은 성능
  - NOOP FUSE의 읽기/쓰기 대역폭은 디스크 그것의 8% 수준



[1]: 대기 중인 FUSE 데몬 깨움
[2]: 커널 모드로 변경

Faster FUSE I/O (v6.9)

- 앱이 파일을 열 때에 FUSE 데몬이 권한을 확인하고, 이후 읽기/쓰기는 FUSE 데몬을 bypass



https://source.android.com/docs/core/storage/fuse-passthrough

Etc

New cachestat(2) system call (v6.5)

- 파일의 특정 범위에 대한 page cache 상태를 파악 할 수 있는 API

```
int cachestat(unsigned int fd, loff_t offset, size_t len,
              size_t cstat_size, struct cachestat *cstat)

struct cachestat {
  __u64 nr_cache;
  __u64 nr_dirty;
  __u64 nr_writeback;
  __u64 nr_evicted;
  __u64 nr_recently_evicted
```

New cachestat(2) system call (v6.5)

- posix_fadvise로 파일의 데이터를 미리 cache에 올려놓는 어플이 그 유용성을 판단할 때 활용
  - fsck, sqlite, …

Scope-based resource management for the kernel (v6.5)

- 로컬 변수가 스코프를 벗어날 때에 특정 cleanup 함수가 호출되도록 하는 설비를 도입해서 메모리나 락 누수를 방지하고, 에러 처러 코드를 단순화
- 컴파일러 확장인 cleanup attribute

  void **auto_kfree**(void **p) { kfree(*p); }

  struct foo *ptr *__attribute__((__cleanup__(***auto_kfree***)))* = NULL;

  // **auto_kfree**(&ptr)

- 이를 활용한 3가지 설비를 도입
  - self-freeing pointer, class, guard

Scope-based resource management for the kernel (v6.5)

- self-freeing pointer

```
//                      name,   type,     cleanup의 정의
DEFINE_FREE(kfree,   void *,   if (_T) kfree(_T))

struct foo *ptr __free(kfree) = NULL
```

- class
  - 특정 타입을 캡슐화 해서 constructor와 destructor가 호출

```
//                      name,  type,        exit,          init,        init의 인자
DEFINE_CLASS(fdget,   struct fd,   fdput(_T),   fget(fd),   int fd)

CLASS(fdget, f)(fd)
```

Scope-based resource management for the kernel (v6.5)

- guard
  - lock과 unlock을 자동화

```
//                          name,    type,              lock,              unlock
DEFINE_GUARD(mutex,  struct mutex *,  mutex_lock(_T), mutex_unock(_T))

guard(mutex)(&uclamp_mutex);
```

Memory allocation profiler for kernel (v6.10)

- 커널 내의 메모리 할당 call site 마다 할당량 / 호출 횟수를 추적할 수 있는 프로파일러
  - 제품에 사용할 수 있을 정도의 경량화

```
root@moria-kvm:~# sort -g /proc/allocinfo|tail|numfmt --to=iec
      2.8M    22648 fs/kernfs/dir.c:615 func:__kernfs_new_node
      3.8M      953 mm/memory.c:4214 func:alloc_anon_folio
      4.0M     1010 drivers/staging/ctagmod/ctagmod.c:20 [ctagmod] func:ctagmod_start
      4.1M        4 net/netfilter/nf_conntrack_core.c:2567 func:nf_ct_alloc_hashtable
      6.0M     1532 mm/filemap.c:1919 func:__filemap_get_folio
      8.8M     2785 kernel/fork.c:307 func:alloc_thread_stack_node
       13M      234 block/blk-mq.c:3421 func:blk_mq_alloc_rqs
       14M     3520 mm/mm_init.c:2530 func:alloc_large_system_hash
       15M     3656 mm/readahead.c:247 func:page_cache_ra_unbounded
       55M     4887 mm/slub.c:2259 func:alloc_slab_page
      122M    31168 mm/page_ext.c:270 func:alloc_page_ext
```

Memory allocation profiler for kernel (v6.10)

- [1] 각 call site 마다 코드 태그와 per-cpu counter를 정의(alloc_tags 섹션)
- [2] current->alloc_tag에 태그와 per-cpu counter를 대입
- 각 할당자의 alloc/free hook에서 per-cpu counter를 업데이트

```
#define kmalloc(...)          alloc_hooks(kmalloc_noprof(__VA_ARGS__))    [0]

#define alloc_hooks(_do_alloc) ({
  static struct alloc_tag __alloc_tag  =  { … };                          [1]
  swap(current->alloc_tag, &__alloc_tag)                                  [2]
  _res = __do_alloc
  …
})
```

mseal(2) to protect memory from unwanted modifications

- 메모리의 접근 권한이나 맵핑을 변경하지 못 하도록 하는 API가 추가됨

  int mseal(void addr, size_t len, unsigned long flags)

- 아래 작업을 차단
  - mprotect(2), pkey_mprotect(2)를 통한 접근 권한 변경
  - mremap(2), munmap(2)을 통한 축소, 확장, 그리고 이동
  - madvise with MAV_DONTNEED
- 해당 영역은 프로세스가 종료 되거나 exec(2)가 실행되기 전까지는 유지
- dynamic loader는 .rodata나 .text 섹션을 sealing

Linux kernel CVE(Common Vulnerabilities and Exposures)

- CVE 번호
  - 소프트웨어 패키지의 보안 취약점에 대한 고유 식별자
  - 미국 국토안보부가 펀딩하는 MITRE Corporation이 운영, 유지보수
  - CVE-YYYY-NNNN
- 리눅스 커널 커뮤니티의 CNA 팀
  - CNA(CVE Numbering Authority) 로 인증 받음
  - Greg Kroh-Hartman, Sasha Levin, Lee Johns 3명으로 구성
  - 커널 보안 커밋에 대한 CVE 번호를 부여하고, linux-cve-announce 메일링 리스트에 공개

Linux kernel CVE(Common Vulnerabilities and Exposures)

- CVE 번호 할당 기준
  - 보안과 관련된 모든 버그 수정 커밋이 대상: 버퍼 오버플로우, 데이터 손상, 충돌(커널 패닉), use-after-free(UAF), 이중해제, 서비스 거부(DoS), 데이터 유출, …
  - 보안 팀은 해당 패치만 반영하는 경향 vs. 많은 패치 반영으로 인한 안정성 문제 그리고 보안 팀의 업무 부하
  - v6.7.1 ~ v6.8.9 사이의 16514 개의 커밋 중 863 개(5%)

Linux kernel CVE(Common Vulnerabilities and Exposures)

- CVE 번호 할당 프로세스
  - 모든 커밋을 검토해서 만장 일치를 받은 패치에 CVE 번호를 할당
  - linux-cve-announce 메일링 리스트에 공개
  - 커뮤니티의 피드백을 반영해서 reject도 가능
    - 24년 6월 기준으로 65개의 CVE 번호가 reject