PintOS : Project 1

MLFQS의 공정한 CPU 분배

정글 10기-34 박도현

MLFQS : Multi-Level Feedback Queue Scheduler

최근에 CPU를 많이 쓴 스레드는 잠시 뒤로, 덜 쓴 스레드는 앞으로 보내 공정하게 CPU를 분배한다

중요한 키워드 변수 4가지

NICE : 양보 성향 - 값이 클수록 스스로 우선순위를 낮추려는 경향

LOAD_AVG : 시스템이 얼마나 바쁜지?

RECENT_CPU : 스레드가 최근에 CPU를 얼마나 썼는지?

PRIORITY : 스케줄 순서를 결정하는 값. 위 3 요소를 반영해 주기적으로 갱신

timer_interrupt()에서 호출 : 1틱마다 벌어지는 일

```
// 매 틱마다 현재 실행중인 스레드의 recent_cpu를 +1
void mlfqs_increment(void) {
    struct thread *cur = thread_current();
    // 아무 일 하지 않는 유휴 스레드의 작업은 계산 x
    if (cur == idle_thread) {
        return;
    }
    // 현재 스레드의 recent_cpu 값에 1 더해주기
    cur->recent_cpu = FP_ADD_INT(cur->recent_cpu, 1);
}
```

timer_interrupt()에서 호출 : 4틱마다 벌어지는 일

```
// 모든 스레드를 순회하며 우선순위를 갱신한다
for (e = list_begin(&all_list); e != list_end(&all_list); e = list_next(e)) {
    struct thread *t = list_entry(e, struct thread, allelem);
    mlfqs_priority(t);
}
```

```
// ready 큐에 있는 스레드들에 대해 스레드 우선순위 내림차순 정렬
list_sort(&ready_list, compare_thread_priority, NULL);
// 재정렬 이후 ready 큐 맨 앞(top)이 현재 스레드보다 우선순위가 높다면 양보하여 top이 선점
struct thread *top = list_entry(list_front(&ready_list), struct thread, elem);
if (top->priority > thread_current()->priority) {
    intr_yield_on_return();
}
```

timer_interrupt()에서 호출 : 1초마다 벌어지는 일

```
struct list_elem *e;
// 시스템 내 모든 스레드를 훑음
for (e = list_begin(&all_list); e != list_end(&all_list); e = list_next(e)) {
    struct thread *t = list_entry(e, struct thread, allelem);
    // TIMER_FREQ = 초당 발생 틱 수 = 100 -> 총 tick % 100 == 0이면 1초 단위 true
    if (timer_ticks() % TIMER_FREQ == 0) {
        // 1초마다 해당 스레드의 recent_cpu 재계산
        mlfqs_recent_cpu(t);
    }
    // 항상 우선순위를 재계산 -> 갱신된 recent_cpu값을 반영하여 재계산
    mlfqs_priority(t);
}
```

결과 : 왜 "공정한 분배"인가

1. 여러 개의 우선순위(레벨)를 두고 주기적 갱신과 정렬

- 여러 레벨(우선순위 값)을 단일 ready_list 정렬로 구현

2. 선점 현황(nice, load_avg, recent_cpu)을 실시간 피드백

- 최근 사용량(recent_cpu)↑ → 우선순위↓, 양보 성향(nice)↑ → 우선순위↓

→ 최근 많이 쓴 스레드는 잠시 쉬고 덜 쓴 스레드가 기회를 얻는 구조 달성