

---

# DEADLOCK:

## 교차로 문제

---

[2025 Operating System Project#2



- 2025.06.15

학번: 20214234

이름: 김하람

## 1. 프로젝트 요구사항 #0

- Pintos 환경에서 빌드 및 실행을 위해 wsl 환경에서 프로젝트 폴더를 구축하였고, run\_crossroads.sh 스크립트를 제작하여 프로젝트 빌드 및 실행을 원활하게 하였음. 에러가 발생할 경우를 대비하여 make\_result 파일과 make\_clean\_result 파일을 생성하게 하여 디버깅을 하였음.
- 프로그램의 실행 결과는 output.txt에 저장함.
- ./run\_crossroads.sh {vehicle\_parameter} 명령으로 프로그램을 쉽게 실행할 수 있음

```
ram@DESKTOP-6FMR5E6:~/os-pintos-project-2$ ls -al
total 360 --
drwxr-xr-x 14 ram ram 4096 Jun 15 18:53 .
drwxr-xr-x 12 ram ram 4096 Jun 3 17:23 ..
drwxr-xr-x 8 ram ram 4096 Jun 15 18:47 .git
-rw-r--r-- 1 ram ram 50 Jun 3 17:09 .gitignore
drwxr-xr-x 2 ram ram 4096 Jun 3 17:09 .vscode
-rw-r--r-- 1 ram ram 4621 May 28 00:30 LICENSE
-rw-r--r-- 1 ram ram 1653 May 28 00:30 Make.config
-rw-r--r-- 1 ram ram 628 May 28 00:30 Makefile
-rw-r--r-- 1 ram ram 3927 May 28 00:30 Makefile.build
-rw-r--r-- 1 ram ram 336 May 28 00:30 Makefile.kernel
-rw-r--r-- 1 ram ram 1551 May 28 00:30 Makefile.userprog
-rw-r--r-- 1 ram ram 21 Jun 3 17:08 README.md
-rw-r--r-- 1 ram ram 235180 May 28 00:30 cscope.out
drwxr-xr-x 2 ram ram 4096 Jun 3 17:09 devices
drwxr-xr-x 3 ram ram 4096 Jun 3 17:09 examples
drwxr-xr-x 2 ram ram 4096 Jun 3 17:09 filesys
drwxr-xr-x 4 ram ram 4096 Jun 3 17:09 lib
-rw-r--r-- 1 ram ram 1660 Jun 15 19:00 make_clean_result
-rw-r--r-- 1 ram ram 20788 Jun 15 19:00 make_result
drwxr-xr-x 2 ram ram 4096 Jun 3 17:09 misc
-rw-r--r-- 1 ram ram 3621 Jun 15 19:00 output.txt
drwxr-xr-x 3 ram ram 4096 Jun 3 17:09 projects
-rwxr-xr-x 1 ram ram 551 Jun 10 14:06 run_crossroads.sh
drwxr-xr-x 3 ram ram 4096 Jun 3 17:09 threads
drwxr-xr-x 2 ram ram 4096 Jun 3 17:09 userprog
drwxr-xr-x 2 ram ram 4096 Jun 10 14:06 utils
drwxr-xr-x 2 ram ram 4096 Jun 3 17:09 vm
```

- vehicle.h/c에 정의되어있는 parse\_vehicle 함수를 구현  
입력 파라미터로 들어온 값을 일반 차량과 앰불런스 차량으로 나누어 각 값을 vehicle\_info 구조체 배열 각 필드에 저장.

```

// 차량 정보를 파싱하는 함수
void parse_vehicles(struct vehicle_info *vehicle_info, char *input)
{
    // input example: aAA:bBD:cCD:dDB:fAB5.12:gAC6.13
    printf("Parsing vehicles from input: %s\n", input);

    int idx = 0;
    char *save_ptr;
    char *token = strtok_r(input, ":", &save_ptr);
    while (token && idx < 8) {
        char id, start, dest;
        int arrival = -1, golden_time = -1;
        int type = VEHICL_TYPE_NORMAL;

        // 앰بول런스인지 확인 ('.' 포함)
        char *dot = strchr(token, '.');
        if (dot != NULL) {
            // 앰بول런스: id start dest arrival_step.golden_time
            size_t len = dot - token; // '.' 이전까지 길이
            if (len >= 4) {
                id = token[0];
                start = token[1];
                dest = token[2];
                arrival = atoi(&token[3]);
                golden_time = atoi(dot + 1);
                type = VEHICL_TYPE_AMBULANCE;
            }
        } else {
            // 일반 차량: id start dest
            if (strlen(token) >= 3) {
                id = token[0];
                start = token[1];
                dest = token[2];
            }
        }

        vehicle_info[idx].id = id;
        vehicle_info[idx].start = start;
        vehicle_info[idx].dest = dest;
        vehicle_info[idx].type = type;
        vehicle_info[idx].arrival = arrival;
        vehicle_info[idx].golden_time = golden_time;
        vehicle_info[idx].state = VEHICLE_STATUS_READY;

        idx++;
        token = strtok_r(NULL, ":", &save_ptr);
    }

    // 파싱한 차량 정보 출력
    for (int i = 0; i < idx; i++) {
        printf("Vehicle %c: Start %c, Dest %c, Type %d, Arrival %d, Golden Time %d\n",
            vehicle_info[i].id, vehicle_info[i].start, vehicle_info[i].dest,
            vehicle_info[i].type, vehicle_info[i].arrival, vehicle_info[i].golden_time);

        printf("Position: (%d, %d)\n", vehicle_info[i].position.row, vehicle_info[i].position.col);
    }
}

```

## 2. 프로젝트 요구사항 #1: 우선 순위 Semaphore 구현

- 우선 순위를 고려하는 동기화 장치(priority\_semaphore, priority\_lock, priority\_condition)을 구현하였음. 세 장치 모두 thread\_priority\_cmp 함수를 기준으로 대기 리스트(waiters)를 정렬하도록 하였음. 이 비교 함수로 정렬된 리스트는 가장 우선순위가 높은 스레드가 리스트의 앞쪽에 위치하게 됨. 세마포어, 락, 조건변수에서 가장 우선순위가 높은 스레드를 먼저 깨우는 데에 사용되었음.

```
// 우선 순위 비교 함수
bool thread_priority_cmp(const struct list_elem *a, const struct list_elem *b, void *aux) {
    const struct thread *t_a = list_entry(a, struct thread, elem);
    const struct thread *t_b = list_entry(b, struct thread, elem);
    return t_a->priority > t_b->priority;
}
```

- 구현한 구조체와 멤버 변수

priority_semaphore	
unsigned value	struct list waiters
priority_lock	
struct thread *holder	struct priority_semaphore sema
priority_condition	
struct list waiters	

- 구현한 함수

Priority Semaphore	
p_sema_init	p_sema_down
p_sema_up	
Priority Lock	
p_lock_init	p_lock_acquire
p_lock_try_acquire	p_lock_release
p_lock_held_by_current_thread	
Priority Condition	
p_cond_init	p_cond_wait
p_cond_signal	p_cond_broadcast

3. 프로젝트 요구사항 #2: 차량이 출발지에서 시작하여 도착지까지 이동하여야 함
- 각 차량마다 스레드를 생성하고 해당 스레드에 메인 루프 함수인 `vehicle_loop`를 할당하여 주었음. 내부의 동작은 다음과 같음.
    - 1) 앰불런스의 경우에는 출발 시간 전까지 대기
    - 2) 신호등에게 진입 허가 요청(앰불런스의 골든 타임 처리, 충돌 검사)
    - 3) 차량 이동 시도
    - 4) 신호등에게 차량이 이동을 시도하였음을 알림
    - 5) 차량이 도착지에 도달하면 차량의 상태 변경
  - 앞에서 파싱한 각 차량의 출발지, 도착지에 따라 해당 `vehicle_path` 변수에 저장된 경로를 따라 이동하도록 구현.

```

// 차량 스레드 메인 루프 함수
void vehicle_loop(void *_vi)
{
    int res;
    int start, dest, step;
    struct vehicle_info *vi = _vi;

    start = vi->start - 'A';
    dest = vi->dest - 'A';
    step = 0;

    vi->position.row = vi->position.col = -1;
    vi->state = VEHICLE_STATUS_READY;

    while (1) {

        // [1] 앰불런스는 도착(arrival) 전까지 대기
        if (vi->type == VEHICLE_TYPE_AMBULANCE && crossroads_step < vi->arrival) {
            notify_vehicle_moved_to_blinker();
            thread_yield();
            continue;
        }

        // [2] 신호등에 진입 허가 요청
        bool can_enter = false;
        if (vi->state == VEHICLE_STATUS_READY || vi->state == VEHICLE_STATUS_RUNNING) {
            can_enter = request_permission_to_blinker(vi, step);
        }

        // [3] 차량 이동 시도
        int res = -1;
        if (can_enter) {
            res = try_move(start, dest, step, vi);

            // [3-1] 차량 이동 성공
            if (res == 1) {
                step++;
                vi->step++;
            }
            // [3-2] 차량 도착지 도달
            else if (res == 0) {
                notify_vehicle_moved_to_blinker();
                notify_vehicle_finished_to_blinker();
                break;
            }
        }

        // [4] 신호등에 차량이 이동을 '시도'하였음을 알림
        notify_vehicle_moved_to_blinker();
    }

    // [5] 차량이 도착지에 도달했으므로 상태를 변경
    vi->state = VEHICLE_STATUS_FINISHED;
}

```

4. 프로젝트 요구사항 #3: 차량이 모두 대기/이동을 하였다면 단위 스텝을 증가시켜야 함

- 신호등은 모든 차량이 이동 시도를 완료했는지 확인하고, 완료되었다면 단위 스텝인 교차로 단계(crossroads\_step)를 증가시킴. 이때, 현재 도착지에 도착하지 않은 차량의 수를 카운트하는 cur\_vehicle\_count 값만큼 이동을 시도한 차량의 수 값이 되면, 해당 동작을 수행하고 다음 단계로 넘어감. 이때, 각 차량들은 이동을 시도한 뒤에 조건 변수에서 대기하고 있기에, p\_cond\_broadcast 함수로 대기 중인 스레드들을 깨움. unitstep\_changed 함수도 내부에서 호출.

```
// 신호등 스레드 함수: 모든 차량이 이동 시도를 완료했는지 확인하고, 완료되었다면 교차로 단계를 증가시킴
void blinker_thread(void *aux UNUSED) {
    while (true) {
        p_lock_acquire(&blinker_lock);
        if (moved_vehicle_count >= cur_vehicle_count) {
            crossroads_step++;
            unitstep_changed();
            moved_vehicle_count = 0;
            p_cond_broadcast(&cond_all_can_enter, &blinker_lock);
        }
        p_lock_release(&blinker_lock);
        thread_yield();
    }
}
```

- 차량은 이동할 수 있는 상황이거나 이동할 수 없는 상황 모두 이동을 시도했다는 사실을 notify\_vehicle\_moved\_to\_blinker 함수를 통하여 신호등에게 알림. 해당 함수에서는 moved\_vehicle\_count 변수 값을 증가시켜, 이후 blinker\_thread가 이를 확인할 수 있도록 함. 그리고 해당 차량 스레드는 조건 변수에서 대기함.

```
// 차량이 이동을 '시도' 했음을 알리는 함수
void notify_vehicle_moved_to_blinker() {
    p_lock_acquire(&blinker_lock);

    moved_vehicle_count++;
    p_cond_wait(&cond_all_can_enter, &blinker_lock);

    p_lock_release(&blinker_lock);
}
```

5. 프로젝트 요구사항 #4: 차량이 교차로에서 Deadlock이 발생하지 않도록 해야함
- 해당 요구사항은 차량이 움직임을 시도하기 전에 신호등에 허가를 요청하는 request\_permission\_to\_blinker 함수 내에서 is\_conflict 함수를 호출함으로써 구현함.
  - is\_conflict 함수에서는 움직이고자 하는 차량을 기준으로 다른 모든 차량들에 대해서 다음 이동 위치를 비교하여 deadlock이 발생하지 않도록 구현함. 이때, 앰불런스의 경우에는 일반차량이 무조건 양보하도록 하고, 충돌이 발생하는 경우에는 이동하지 않고 대기하도록 bool 값을 반환함.

```
// 차량이 충돌하는지 검사하는 함수
bool is_conflict(struct vehicle_info *vi, int step) {
    // [1] 차량의 현재 위치와 다음 위치 가져오기
    int vi_from = vi->start - 'A';
    int vi_to = vi->dest - 'A';
    struct position vi_cur = vehicle_path[vi_from][vi_to][step];
    struct position vi_next = vehicle_path[vi_from][vi_to][step + 1];

    // [2] 차량이 도착했을 경우 충돌 검사 중지
    if ((vi_next.row == -1 && vi_next.col == -1) ||
        (vi_cur.row == -1 && vi_cur.col == -1)) {
        return false;
    }

    // [3] 다른 모든 차량들과 충돌 검사
    for (int i = 0; i < all_vehicle_count; i++) {
        struct vehicle_info *other = &all_vehicles[i];

        if (other == vi) continue;
        if (other->state != VEHICLE_STATUS_RUNNING) continue;

        int o_from = other->start - 'A';
        int o_to = other->dest - 'A';
        int o_step = other->step;
        struct position o_cur = vehicle_path[o_from][o_to][o_step];
        struct position o_next = vehicle_path[o_from][o_to][o_step + 1];

        // [3-1] 두 차량이 같은 위치로 이동하려는 경우 -> 앰불런스: 양보 / 일반차량: 충돌
        if (vi_next.row == o_next.row && vi_next.col == o_next.col) {
            if (vi->type == VEHICL_TYPE_AMBULANCE && other->type != VEHICL_TYPE_AMBULANCE) continue;

            return true;
        }

        // [3-2] 두 차량이 서로의 위치를 교차하려는 경우 -> 충돌
        if (vi_next.row == o_cur.row && vi_next.col == o_cur.col &&
            o_next.row == vi_cur.row && o_next.col == vi_cur.col) return true;
    }

    // [4] 충돌이 발생하지 않음
    return false;
}
```



6. 프로젝트 요구사항 #5: 앰불런스는 출발 시간 이후에 맵에 등장해야하며, 골든 타임 전 도착하여야 함

- 앰불런스의 골든타임은 request\_permission\_to\_blinker 함수 내에서 제어하며, 골든 타임을 넘겼을 경우에는 반드시 이동하게 함. 또한, 요구사항 4에서 구현한 is\_conflict 함수 내에서 앰불런스는 타 차량이 반드시 양보하도록 구현하여 골든 타임을 지킬 수 있도록 구현함.

```
// [1] 앰불런스 골든타임 처리
if (vi->type == VEHICL_TYPE_AMBULANCE) {
    int remain_time = vi->golden_time - crossroads_step;
    if (remain_time <= 0) {
        p_lock_release(&blinker_lock);
        return true;
    }
}
```

```
// [3-1] 두 차량이 같은 위치로 이동하려는 경우 -> 앰불런스: 양보 / 일반차량: 충돌
if (vi_next.row == o_next.row && vi_next.col == o_next.col) {
    if (vi->type == VEHICL_TYPE_AMBULANCE && other->type != VEHICL_TYPE_AMBULANCE) continue;

    return true;
}
```

- 앰불런스는 while loop 내에서 출발 시간 전까지는 신호등에 이동을 시도하였다는 사실을 알리고, thread\_yield를 호출하여 다른 차량 스레드들에게 우선 순위를 부여함.

```
// [1] 앰불런스는 도착(arrival) 전까지 대기
if (vi->type == VEHICL_TYPE_AMBULANCE && crossroads_step < vi->arrival) {
    notify_vehicle_moved_to_blinker();
    thread_yield();
    continue;
}
```

7. 프로젝트 요구사항 #6: 과제 제출 요구사항

- 단위 스텝별 실행 결과 출력

```

X X d X X X
X X X X X
- - - - -
a -
X X - X X
X X - b X X
unit step: 1

X X X X X
X X d X X
- - - - -
a -
X X - b X X
X X - X X
unit step: 2

X X X X X
X X X X X
d - c
- - - - -
a - b
X X - X X
X X - X X
unit step: 3

X X X c X X
X X X X X
- b
- - - - -
d - a
X X - X X
X X - X X
unit step: 5

X X X X X
X X X b X X
-
- - - - a - -
f -
X X d - X X
X X - X X
unit step: 6

X X X b X X
X X X X X
- a
- - - - -
g f -
X X - X X
X X d - X X
unit step: 7

X X X X X
X X X X X
a
- - - - -
g f -
X X - X X
X X - X X
unit step: 8

```

```

X X   X   X X
X X   X   X X
    a -
- - - - -
    g -
X X f -   X X
X X   -   X X
unit step: 9

```

```

X X   X   X X
X X   X   X X
    a -
- - - - -
    g -
X X   -   X X
X X f -   X X
unit step: 10

```

```

X X   X   X X
X X   X   X X
a     -
- - - - -
    - g
X X   -   X X
X X   -   X X
unit step: 11

```

```

X X   X   X X
X X   X   X X
a     -
- - - - -
    - g
X X   -   X X
X X   -   X X
unit step: 12

```

```

X X   X   X X
X X   X   X X
    -
- - - - -
    - g
X X   -   X X
X X   -   X X
unit step: 13

```

```

X X   X   X X
X X   X   X X
    -
- - - - -
    -
X X   -   X X
X X   -   X X
unit step: 14

```

```

finished. releasing resources ...
good bye.

```