〈레포트〉

자동화 물류 센터 시뮬레이터

강의명: 운영체제
담당교수: 박상오 교수님
작성자: 신성섭
학번: 20226XX2
학과: 소프트웨어학부

1. 코드 구현 방법

1) 시뮬레이터에서 동시에 동작 할 수 있는 로봇의 개수

본 프로그램에서는 1개의 로봇부터 최대 9개의 로봇까지만 시뮬레이터를 돌릴 수 있다. 물류 센터에서는 동시에 여러 물류 로봇들이 제어된다.

2) automated_warehouse.c

(1) 로봇의 이름과 로봇 인덱스 관리

```
16    struct robot* robots;
17    char* robotName[] = { "R1", "R2","R3", "R4", "R5", "R6", "R7", "R8", "R9"}; // 각 로봇별 이름
18    int idxs[11] = { 0, 0, 1, 2, 3, 4, 5, 6, 7 , 8, 9 }; // 0번 인덱스 : 로봇 총개수  1번 인덱스 이후 : 로봇에게 부여되는 고유 ID
```

로봇의 이름과 각 로봇에 부여되는 인덱스는 전역변수로 정적으로 선언하여 관리하였다.

(2) 유저 입력 파라미터 파싱

```
int robotCnt = atoi(argv[1]); // 로봇의 총 개수
char* inputString = argv[2]; // 로봇들의 할 일

// requireLoad 의미 : (A:10, B : 20, C: 30 + 물건 번호(1~7))
int requireLoad[10] = { 0 }; // 각 로봇별 원하는 짐

// strtok_r를 사용하여 문자열을 :을 기준으로 파싱
int robotNum = 0;
char* token;
char* savePtr; // strtok_r 함수에서 사용할 임시 변수

token = strtok_r(inputString, ":", &savePtr);

while (token != NULL) {
    //printf("%s\n", token); // 디버깅 코드

    // 문자열로부터 정수를 얻기 위해 atoi 함수 사용
    requireLoad[robotNum] += atoi(&token[0]);

    if (token[1] == 'A') {
        requireLoad[robotNum] += 10;
    }
    else if (token[1] == 'B') {
        requireLoad[robotNum] += 20;
    }
    else if (token[1] == 'C') {
        requireLoad[robotNum] += 30;
    }

    // 다음 토큰을 가져오기.
    token = strtok_r(NULL, ":", &savePtr);
    //printf("%d : %d \n", robotNum, requireLoad[robotNum]); //디버깅용 코드
    robotNum += 1;
}
```

atoi 함수를 이용하여 로봇의 총 개수를 구하였고, strtok_r 함수를 이용하여 각 로봇이 해야 할 업무를 구해 주었다. 편의성을 위해 requireLoad 값으로 십의 자리는 로봇이 최종

도착해야하는 장소를 나타내는 값(A=10, B=20, C=30 으로 설정)으로 만들어 주었고 일의 자리는 로봇이 필요한 짐의 위치를 나타내는 값 (1~7)을 나타내었다. 이 과정을 통해 파라미터 값으로 3A가 들어오면 requireLoad에는 13이 저장된다.

(3) 메세지 박스 초기화 및 중앙 노드 및 로봇 쓰레드 생성

```
initialize_messageBoxs(robotCnt); // 쓰레드간 메세지 통신을 위한 messageBox 초기화
robots = malloc(sizeof(struct robot) * robotCnt); //로봇 생성

for (int i = 0; i < robotCnt; i++) {
    setRobot(&robots[i], robotName[i], 5, 5, requireLoad[i], 0);
}

tid_t* threads = malloc(sizeof(tid_t) * (robotCnt + 1));

idxs[0] = robotCnt; //0번 인덱스는 로봇의 총 개수
threads[0] = thread_create("CNT", 0, &centralNode, &idxs[0]);

for (int i = 1; i <= robotCnt; i++) {
    threads[i] = thread_create(robotName[i-1], 0, &robotNode, &idxs[i]);
}
```

중앙 노드와 로봇 쓰레드가 통신을 하기 위해 필요한 초기화 작업을 호출 한다. 그 후 중앙노드와 각 로봇을 위한 쓰레드를 생성해준다. 중앙노드 쓰레드는 centralNode(void *aux) 함수를 실행하고 로봇 쓰레드는 robotNode(void * aux)함수를 실행한다.

(4) centralNode(void* aux)

```
// 중앙 관제 노드를 동작시키는 쓰레드가 수행하는 함수
void centralNode(void* aux) {
    int robotCnt = *((int*)aux);
    //printf("%d \n", robotCnt); // 디버깅 코드

    int total = 0;
    while (total != robotCnt) { // 모든 로봇이 짐을 하역 할때까지 반복
        thread_sleep(1000);
        total+=MesaageByCentral(robotCnt);
        thread_sleep(1000);
        print_map(robots, robotCnt);
        increase_step();
        unblock_threads();
    }

    thread_sleep(1000);
    total += MesaageByCentral(robotCnt);
    print_map(robots, robotCnt);
    increase_step();

    printf("simulation success! congratulation! \n");
}
```

모든 로봇이 최종 목적지에 도착한 것을 확인 할 때까지 다음을 반복한다.

로봇들로 부터 온 메세지를 확인하여 명령을 내린다. 현재 턴의 시뮬레이션 상황을 화면에 표시하는 작업을 한다. 블록되어 있는 모든 로봇 쓰레드를 언블락 해준다.
모든 로봇이 최종 목적지에 도착하면 시뮬레이션 성공 메세지를 보여준 후 종료한다.

(5) robotNode(void* aux)

```
// 로봇을 동작시키는 쓰레드가 수행하는 함수
void robotNode(void* aux) {
    int idx = *((int*)aux);
    while (1) {
        writeMessageByRobot(idx, &robots[idx]);
        block_thread();
        readMessageByRobot(idx, &robots[idx]);
    }
}
```

로봇의 현재 상황을 중앙노드에 전달하는 메세지를 쓰고 블록 된다. 블록된 상태가 풀린 후에는 중앙노드로부터 온 메세지를 읽고 행동을 수행한다.

3) aw_thread.c

(1) block_thread()

```
struct list blocked_threads;

/**
 * A function unblocking all blocked threads in "blocked_threads"
 * It must be called by robot threads
 */
void block_thread(){
    // You must implement this

    enum intr_level old_level;

    old_level = intr_disable();

    list_push_back(&blocked_threads, &thread_current()->elem);
    thread_block();

    intr_set_level(old_level);

}
```

블록 대상이 되는 쓰레드를 리스트에 추가 한 후 현재 쓰레드를 블록시킨다.

(2) unblock_threads()

```
void unblock_threads(){
    // you must implement this
    enum intr_level old_level;

    old_level = intr_disable();
    while (!list_empty(&blocked_threads)) {;
        thread_unblock(list_entry(list_pop_front(&blocked_threads),
            struct thread, elem));
    }

    intr_set_level(old_level);
}
```

모든 쓰레드를 언블록 시킨다. 리스트에 모든 블록된 쓰레드들이 들어 있으므로 리스트에서 쓰레드를 꺼내면서 꺼낸 쓰레드를 언블록 시킨다.

4) aw_massage.c

(1) 메세지 박스, 중요 위치들

```
struct messsage_box* boxes_from_central_control_node;
/** message boxes from robots to central control node */
struct messsage_box* boxes_from_robots;

// message의 cmd 정의 : 0:(0,0) wait , 1:(0,-1) 서  , 2:(1,0) 남 , 3:(0,1) 동 , 4:(-1,0) 북

int requirePosition[8][2] = { {0,0},{1,1},{1,3},{1,4},{1,5},{4,1},{4,3},{4,4} }; // 물건이 있는 위치
int destination[4][2] = { {0,0},{0,2},{2,0},{5,2} }; // A,B,C 하역 장소의 위치
```

중앙센터가 보내는 메세지들의 배열을 저장하는 boxes_from_cetral_control_node, 로봇들이 보내는 메세지들의 배열을 저장하는 boxes_from_robots, 물건이 있는 위치를 나타내는 배열 requirePosition, 로봇의 최종 목적지를 나타내는 destination 배열은 전역변수로 관리한다.

(2) 메세지 박스 초기화

```
void initialize_messageBoxs(int robotCnt) {
    boxes_from_central_control_node = malloc(sizeof(struct messsage_box) * robotCnt);
    boxes_from_robots = malloc(sizeof(struct messsage_box) * robotCnt);
    for (int i = 0; i < robotCnt; i++) {
        boxes_from_robots[i].dirtyBit = 0;
        boxes_from_central_control_node[i].dirtyBit = 0;
    }
}
```

각 로봇의 개수만큼 사전에 메세지를 쓸 수 있는 공간을 만들어 준다.

(3) 로봇이 중앙 노드에게 보내는 메세지를 쓰는 함수

```
void writeMessageByRobot(int robotIndex, struct robot* _robot) {
    //printf("%d : writeMessageByRobt 수행₩n", robotIndex);
    if (boxes_from_robots[robotIndex].dirtyBit == 0) { //dirtybit가 1인경우 아직 사용하.
        boxes_from_robots[robotIndex].dirtyBit = 1;
        boxes_from_robots[robotIndex].msg.row = _robot->row;
        boxes_from_robots[robotIndex].msg.col = _robot->col;
        boxes_from_robots[robotIndex].msg.required_payload = _robot->required_payload;
        boxes_from_robots[robotIndex].msg.current_payload = _robot->current_payload;
        boxes_from_robots[robotIndex].msg.cmd = 0;
    }
    //printf("%d: WriteMessageByRobt 종료 ₩n₩n", robotIndex);
}
```

writeMessageByRobot()을 이용하여 각 로봇이 본인의 index에 해당하는 위치에 중앙 노드에 전달할 메세지를 쓴다.

(4) 중앙 노드가 로봇에게 보내는 메세지를 쓰는 함수

```
void writeMessageByCentral(int robotIndex, int nextCommand) {

    if (boxes_from_central_control_node[robotIndex].dirtyBit == 0) {
        boxes_from_central_control_node[robotIndex].msg.cmd = nextCommand;
        boxes_from_central_control_node[robotIndex].dirtyBit = 1;
    }

}
```

중앙노드가 각 로봇에게 로봇이 다음 수행해야 할 메세지를 쓴다. 이때 0은 대기 , 1은 서쪽으로 이동, 2는 남쪽으로 이동, 3은 동쪽으로 이동, 4는 북쪽으로 이동이다.

(5) 로봇이 중앙노드가 보낸 메세지를 읽고 움직임을 수행하는 함수

```
void readMessageByRobot(int robotIndex, struct robot* _robot) {
    //printf("%d : readMessageByRobot 수행₩n", robotIndex);
    //printf("%d ₩n", boxes_from_central_control_node[robotIndex].msg.cmd);

    if (boxes_from_central_control_node[robotIndex].dirtyBit == 1) {
        if (boxes_from_central_control_node[robotIndex].msg.cmd == 1) { // 서
            _robot->col = _robot->col - 1;
        }
        else if (boxes_from_central_control_node[robotIndex].msg.cmd == 2) { // 남
            _robot->row = _robot->row + 1;
        }
        else if (boxes_from_central_control_node[robotIndex].msg.cmd == 3) { // 동
            _robot->col = _robot->col + 1;
        }
        else if (boxes_from_central_control_node[robotIndex].msg.cmd == 4) { // 북
            _robot->row = _robot->row - 1;
        }

        int gotoPosition = _robot->required_payload % 10;
        if (_robot->col == requirePosition[gotoPosition][1] && _robot->row == requirePosition[gotoPosition][0]) { //로봇이 짐을
            _robot->current_payload = gotoPosition;
        }

        boxes_from_central_control_node[robotIndex].dirtyBit = 0; // 메세지를 다 사용했으므로 새로운 메세지가 쓰이는 걸 허용.
    }
}
```

메세지에 들어있는 명령어를 기반으로 로봇이 움직인다. 움직인 위치가 짐이 있는 장소면 로봇은 짐을 든다.

(6) 중앙노드가 로봇이 보낸 메세지를 읽고 로봇들에게 명령을 결정하는 함수

```
int MesaageByCentral(int robotCnt) {
    for (int i = 0; i < robotCnt; i++) {
        while (boxes_from_robots[i].dirtyBit == 0) {}; //중앙 노드는 모든 로봇들이 메세지를 보낼 때까지 대기
    }

    int total = 0; // 이번턴에 하역장소에 도착하는 로봇의 총 수

    // 로봇들이 이동가능한 위치를 표기 하기 위한 맵
    char world[6][7] = {
        {'X', 'X', 'D', 'X', 'X', 'X', 'X' },
        { 'X', ' ', ' ', ' ', ' ', ' ', 'X' },
        { 'D', ' ', ' ', ' ', ' ', ' ', 'X' },
        { 'X', ' ', ' ', ' ', ' ', ' ', 'X' },
        { 'X', ' ', ' ', ' ', ' ', ' ', 'X' },
        { 'X', 'X', 'D', 'X', 'X', 'W', 'X' }
    };

    // S위치를 제외한 나머지 위치의 로봇의 위치를 world에 X로 표시
    for (int i = 0; i < robotCnt; i++) {
        int row = boxes_from_robots[i].msg.row;
        int column = boxes_from_robots[i].msg.col;
        if (!(row == 4 && column == 5)) {
            world[row][column] = 'X';
        }


    }

    for (int i = 0; i < robotCnt; i++) {
```

[각 로봇의 경로를 결정하는 알고리즘] }

```
    for (int i = 0; i < robotCnt; i++) {
        boxes_from_robots[i].dirtyBit = 0; // 메세지를 다 사용했으므로 새로운 메세지가 쓰이는 걸 허용
    }

    return total; // 이번 턴에 하역장소에 도착한 로봇의 수.
}
```

모든 로봇들이 메세지를 보낼때까지 대기한다. 각 로봇의 현재 위치를 파악한 후 로봇의 경로를 결정하는 알고리즘을 통해 각 로봇에게 내릴 명령어를 결정하고 메세지함에 명령을 쓴다. 모든 로봇에게 명령을 내린 후에는 최종목적지에 도착한 로봇의 수를 반환한다. 각 로봇의 경로를 결정하는 알고리즘은 '2. 트러블 슈팅과정 3) 물류 로봇의 경로 설정 문제'에 작성하였다.

2.트러블 슈팅 과정

1) printMap()을 통한 robot 정보 출력 시 로봇이름이 깨지는 문제.

setRobot(&robots[i], robotName[i], 5, 5, requireLoad[i], 0); 에서 처음에는 robotName[i] 부분을 char[] 형의 robotName으로 선언하여 프로그램을 작성하였다. 이렇게 작성하니 로봇이름이 깨지는 문제가 발생하였다. 로봇 이름을 전역 변수로 따로 빼고 char* [] robotName 으로 관리하니 로봇 이름이 정상 출력 되었다.

2) 멀티 쓰레드 환경에서 strtok() 함수 사용 문제

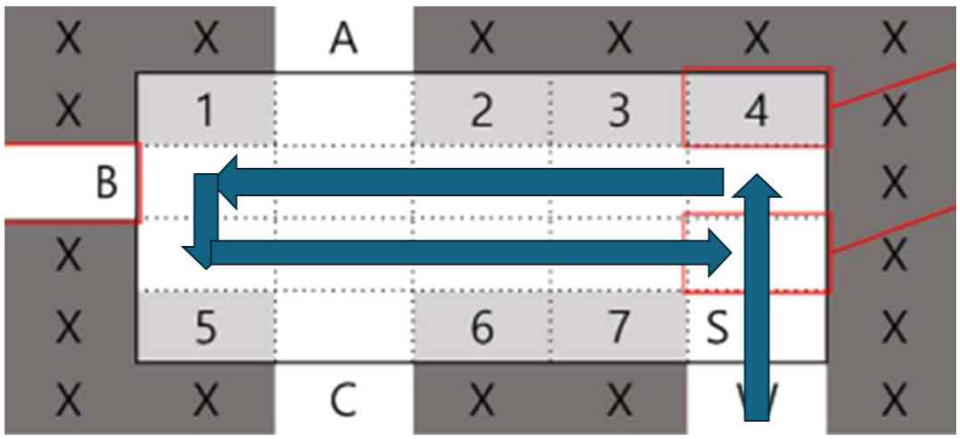
멀티 쓰레드 환경에서 strtok()를 사용할시 문제가 발생 할 수 있어서 make를 수행 할 시, 정상적으로 컴파일이 되지 않았다. 컴파일러가 추천하는 대로 strtok_r()함수로 바꾸어 작성하였더니 정상적으로 컴파일 되었다.

3) 물류 로봇의 경로 설정 문제

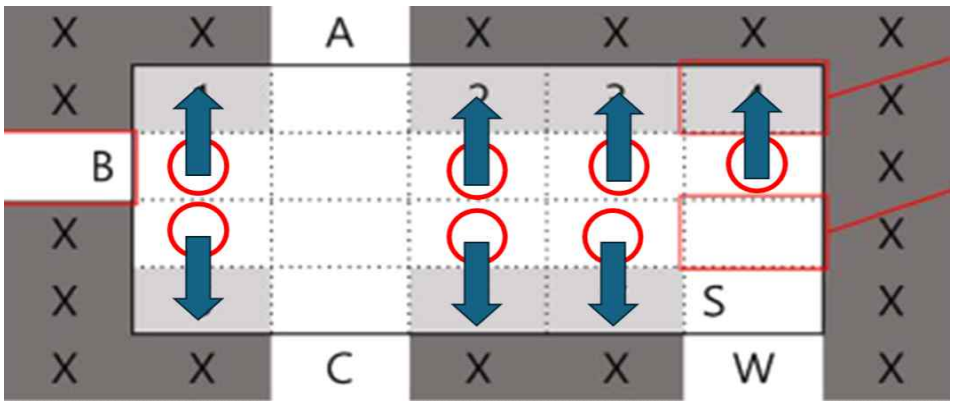
물류센터에서 동시에 여러 대의 로봇을 동작시키 다보니 로봇들의 경로가 겹치는 문제가 발생하였다. 다양한 알고리즘을 시도해 보았지만 각각의 알고리즘들에서 로봇들의 경로가 겹치는 바람에 교착상태의 문제가 발생하였다.

위의 문제를 해결하기 위해 로봇들 간의 서로 겹치는 경로를 최소화하였고, 어쩔 수 없이 경로가 겹치는 경우에는 우선순위를 부여하여 교착상태가 되는 상황을 피하였다.

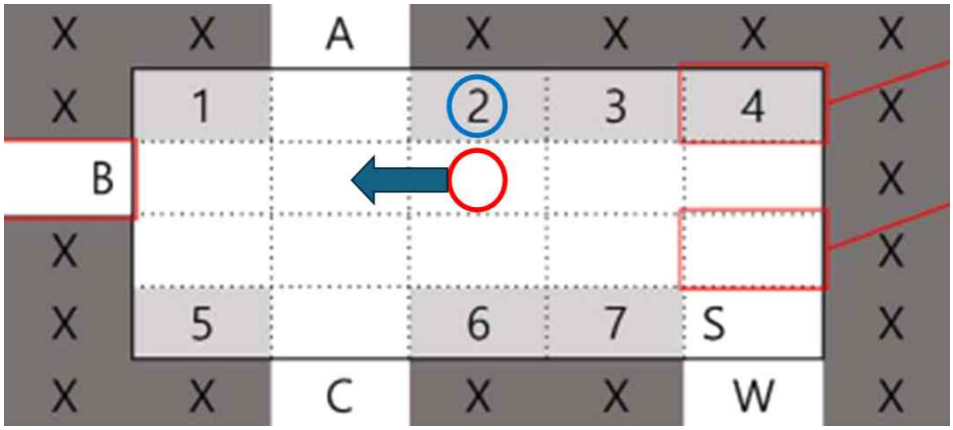
로봇들이 교착상태가 되는 것을 피하기 위해 다음과 같은 알고리즘을 설정하였다.



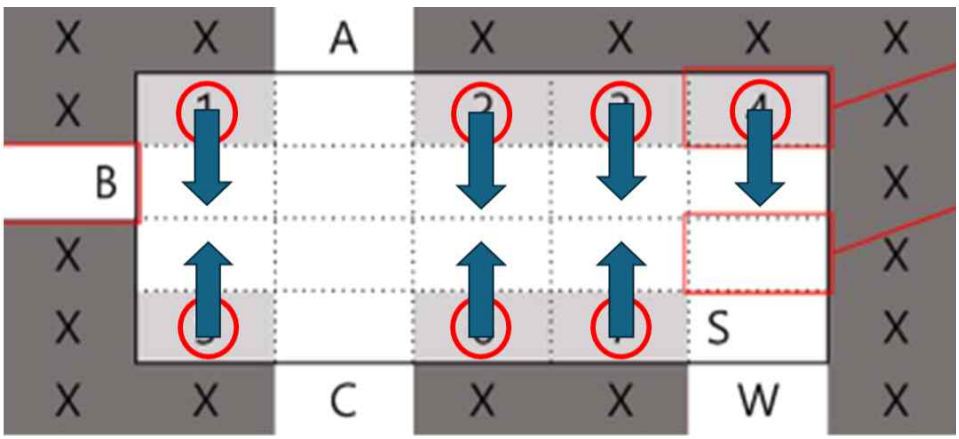
로봇들은 w에서 시작하여 다음 파란 경로로만 움직인다. 일방통행 조건을 부여하여 두 로봇이 서로 반대 방향으로 움직이려고 함으로 인해 발생하는 교착상태를 피하였다.



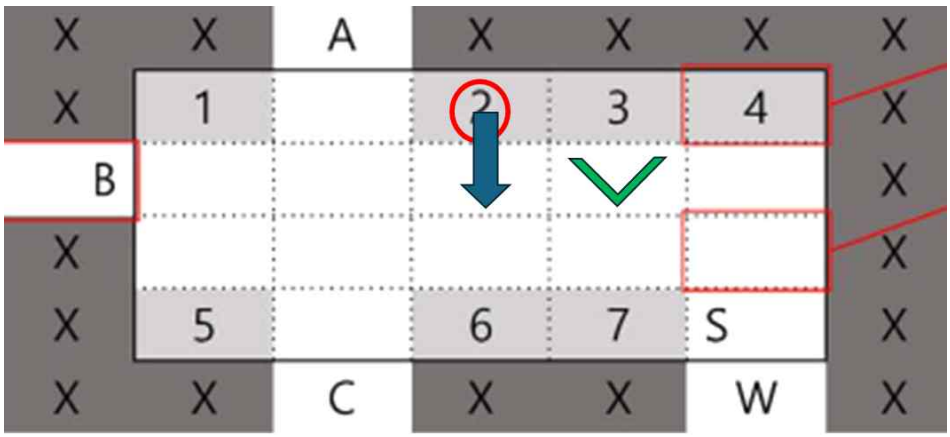
각 빨강 원 위치에서 자신이 필요한 짐이 바로 1칸 위 또는 아래에 있는 경우 물류 창고에 진입한다. 단 이때 물류 창고에는 로봇이 없는 경우에만 진입한다.



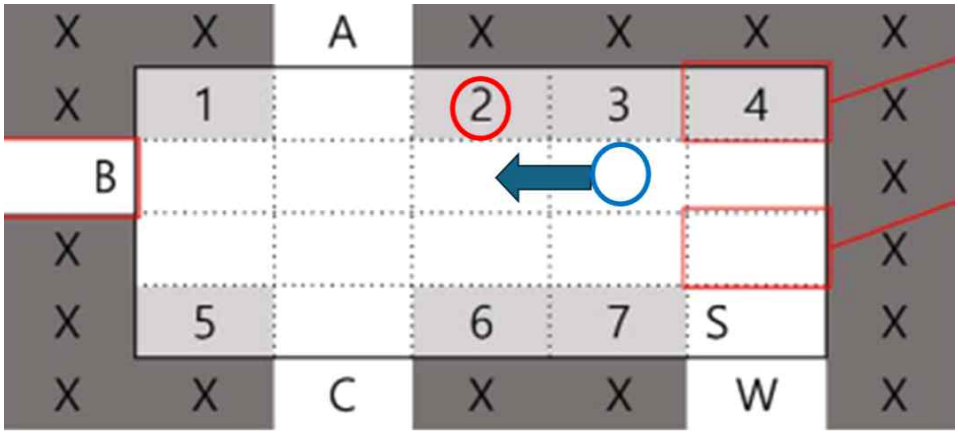
만약 파란색 원의 위치에 물류 로봇이 있는 경우라면 물류창고에 진입 하지 않고 회전로를 따라 한 바퀴 추가로 돈 후에 진입 시도를 한다.



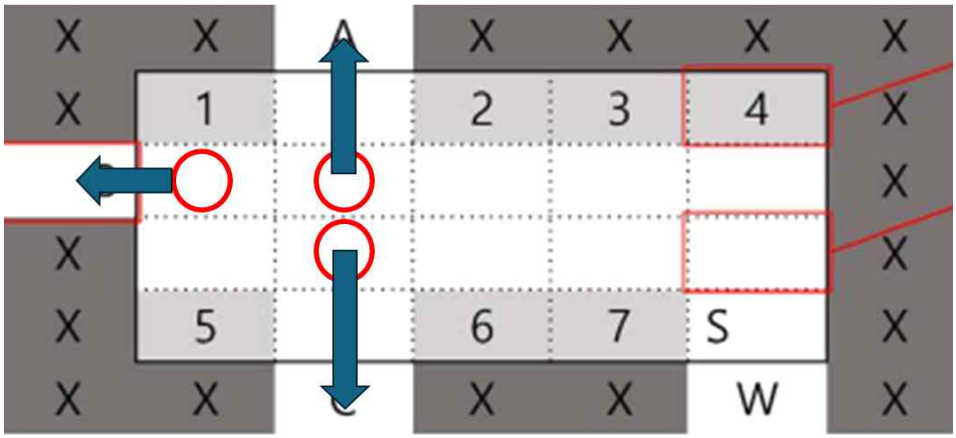
각 물류창고에 진입한 후 로봇은 다음 파란색 경로로만 움직 일 수 있는 데 이때 파란색 경로로 움직일 수 있다면 이동하고 그렇지 않다면 대기한다.



파란색 경로로 움직일 수 있는 경우는 위의 사진처럼 초록색 V 위치에 로봇이 없는 경우다.



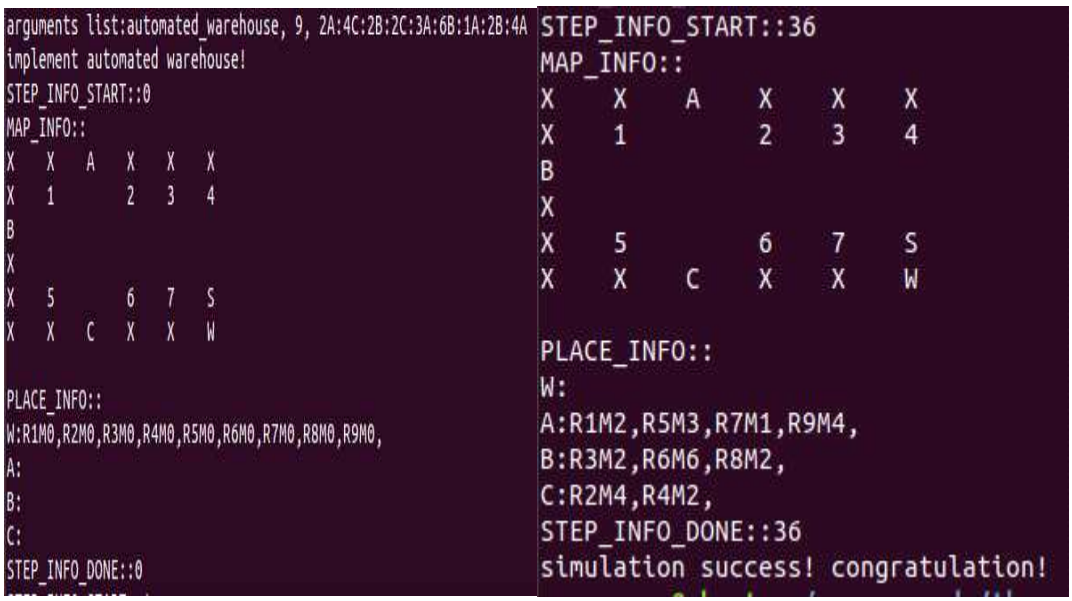
만약 위의 그림처럼 초록색 V위치에 로봇 한대가 있다면 그 위치에 있는 로봇에게 우선순위가 부여되기 때문에 초록색 V위치에 있는 로봇(파란색 원)은 왼쪽으로 이동하고 빨간색 원에 있는 로봇은 대기한다.
위의 설명에서는 로봇이 2번의 위치에 있는 경우 로만 설명하였지만 1~7번의 경우도 위와 동일한 원리로 동작한다.



빨간색 원 위치에서 로봇이 짐을 가지고 있고 자신이 가야하는 최종 목적지 이동 방향과 일치한다면 파란색 경로로 움직인다.

위와 같은 알고리즘은 현실의 주차장 시스템과 매우 유사하게 돌아간다. 위의 알고리즘을 사용하면 구현이 엄청 복잡하지 않고, 많은 로봇이 동시에 동작할 때 교착상태를 피할 수 있는 장점이 있다. 하지만 로봇의 수가 적을 때는 로봇이 원하는 짐을 얻기 위해 한 바퀴 더 돌아야하는 비효율적인 상황이 발생할 수 있다는 단점이 있다.

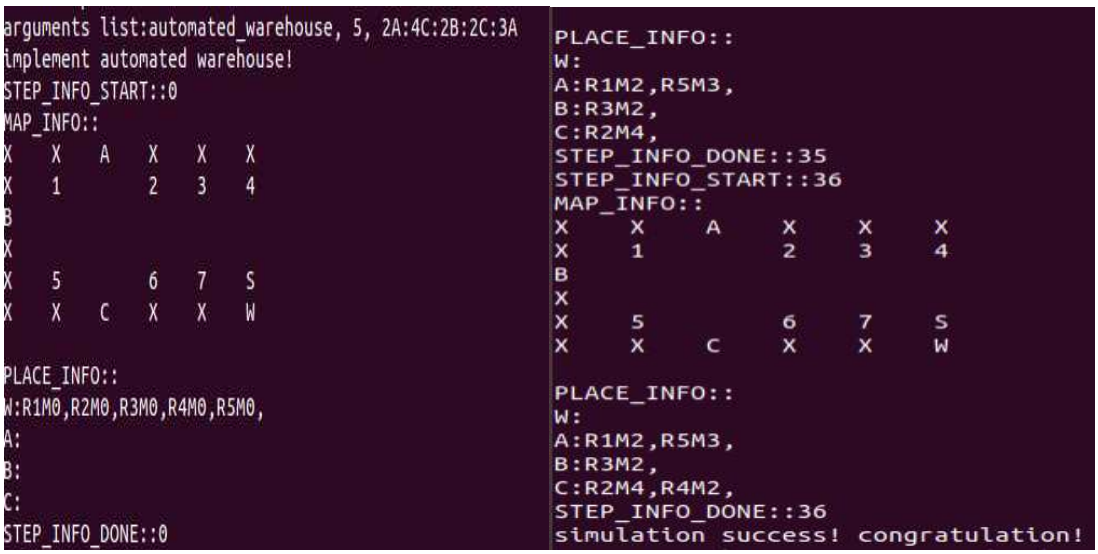
3. 시뮬레이션 결과 설명



<로봇이 9 대 동시에 동작하는 초기 상황(좌)/ 최종 결과(우)>

최대 9대까지의 로봇이 동시에 모두 정상적으로 설계한 알고리즘대로 동작하였다.

설명의 편의를 위해 5대가 동시에 돌아가는 예시인 ../../utils/pintos automated_warehouse 5 2A:4C:2B:2C:3A 로 시뮬레이션한 결과를 보면 다음과 같다.



< 초기 상황(좌)/ 최종 결과(우)>

1번 로봇은 2번에 위치한 짐을 가지고 A에 최종 도착하는 것이고, 2번 로봇은 4번 짐을 가지고 C에 도착, 3번은 2번짐을 가지고 B에 도착, 4번은 2번짐을 가지고 C에 도착, 5번은 3번짐을 가지고 A에 도착하는게 목표이다. 최종결과 사진을 보면 A위치에 R1M2,R5M3가 있는 것을 보아 1번로봇이 2번짐을 가지고 있고, 5번로봇이 3번 짐을 가지고 있는 상황이다. 이는 우리의 시뮬레이션 목표와 일치한다. 마찬가지로 B 위치에 R3M2가 있고 이는 3번로봇이 2번의 짐을 가지고 있는 상황을 의미하며, C번 위치에 R2M4,R4M2가 있는 것은

C번위치에 2번 로봇이 4번의 짐을 가지고 있는 상황, 4번 로봇이 2번의 짐을 가지고 있는 상상황이다. 5개의 로봇 모두 초기 목표 상황과 일치하므로 시뮬레이션 결과는 올바르다.

```
STEP_INFO_START::6
MAP_INFO::
X    X    A    X    X    X
X    1         R1M2 3    R2M4
B                   R3   R4
X                        R5
X    5         6    7    S
X    X    C    X    X    W

PLACE_INFO::
W:
A:
B:
C:
STEP_INFO_DONE::6
```

```
STEP_INFO_START::7
MAP_INFO::
X    X    A    X    X    X
X    1         R1M2 3    R2M4
B              R3   R4   R5
X
X    5         6    7    S
X    X    C    X    X    W

PLACE_INFO::
W:
A:
B:
C:
STEP_INFO_DONE::7
```

< step 6(좌) / step7 모습(우)>

알고리즘이 올바르게 동작함을 확인하기 위해 step 6 과 step 7 상황을 보자.

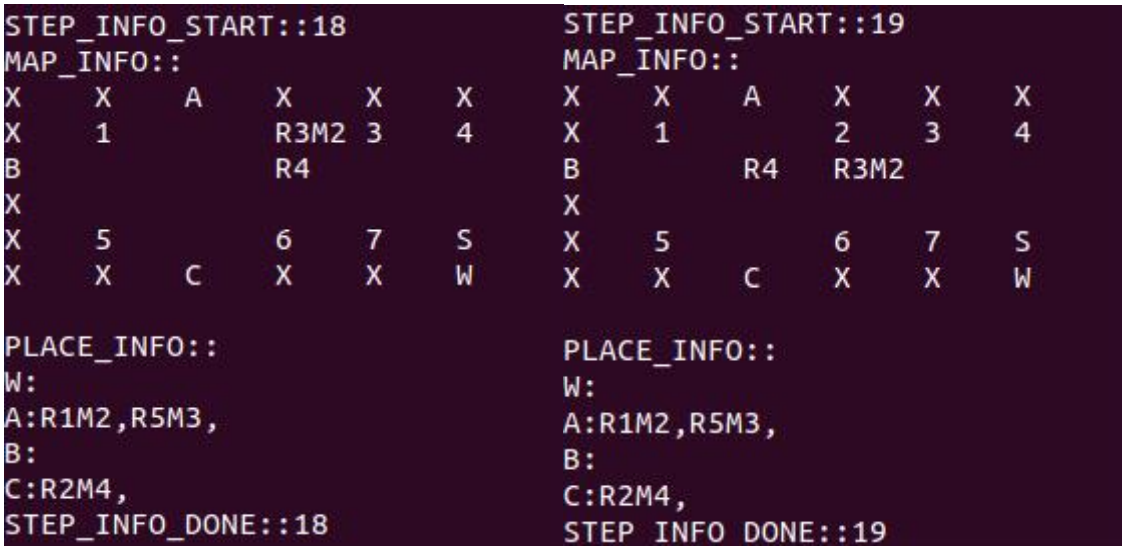
R1과 R3 중에 회전 교차로를 돌고 있는 R3에게 이동의 우선순위가 있다. 따라서 R1은 대기하고 R3은 이동 해야 하므로 step7의 모습은 올바르다.

```
STEP_INFO_START::13
MAP_INFO::
X    X    A    X    X    X
X    1    R5M3 2    3    4
B         R1M2
X    R2M4      R4   R3
X    5         6    7    S
X    X    C    X    X    W

PLACE_INFO::
W:
A:
B:
C:
STEP_INFO_DONE::13
```

```
STEP_INFO_START::14
MAP_INFO::
X    X    A    X    X    X
X    1    R1M2 2    3    4
B
X         R2M4      R4   R3
X    5         6    7    S
X    X    C    X    X    W

PLACE_INFO::
W:
A:R5M3,
B:
C:
STEP_INFO_DONE::14
```

< step 13(좌) / step14 모습(우)>

step 13에서 R1과 R5은 모두 짐을 가지고 있는 상황이고 두 로봇 모두 A가 최종 목적지이다. R5는 A로 들어가고 R1은 위로 이동하는 것이 올바른 상황인데 step 14에서는 이 상황이 올바르게 반영되어 있다. A Place에 R5M3가 있는 것을 통해 A에 올바르게 도착함을 확인 할 수 있다.



< step 18(좌) / step 19 모습(우)>

또 다른 상황으로 step 18 과 step 19를 확인해 보자.

3번과 4번은 모두 2번짐을 필요로 한다. 2번 창고에 3번 로봇이 이미 있는 상황이므로 4번 로봇은 한바퀴 더 돈 후 진입을 시도해야하는 데 step 19에서는 이 상황이 올바르게 반영되어 있다.

../../utils/pintos automated_warehouse 5 2A:4C:2B:2C:3A의 전체 시뮬레이션 과정은 '4. 시뮬레이션 전체 과정 모습'에서 확인 가능하다.

4. 시뮬레이션 전체 과정 모습

```
arguments list:automated_warehouse, 5, 2A:4C:2B:2C:3A
implement automated warehouse!
STEP_INFO_START::0
MAP_INFO::
X   X   A   X   X   X
X   1       2   3   4
B                       
X                       
X   5       6   7   S
X   X   C   X   X   W

PLACE_INFO::
W:R1M0,R2M0,R3M0,R4M0,R5M0,
A:
B:
C:
STEP_INFO_DONE::0
```

```
STEP_INFO_START::1
MAP_INFO::
X    X    A    X    X    X
X    1         2    3    4
B
X
X    5         6    7    R1
X    X    C    X    X    W

PLACE_INFO::
W:R2M0,R3M0,R4M0,R5M0,
A:
B:
C:
STEP_INFO_DONE::1
STEP_INFO_START::2
MAP_INFO::
X    X    A    X    X    X
X    1         2    3    4
B
X                        R1
X    5         6    7    R2
X    X    C    X    X    W

PLACE_INFO::
W:R3M0,R4M0,R5M0,
A:
B:
C:
STEP_INFO_DONE::2
```

```
STEP_INFO_START::3
MAP_INFO::
X    X    A    X    X    X
X    1         2    3    4
B                        R1
X                        R2
X    5         6    7    R3
X    X    C    X    X    W

PLACE_INFO::
W:R4M0,R5M0,
A:
B:
C:
STEP_INFO_DONE::3
STEP_INFO_START::4
MAP_INFO::
X    X    A    X    X    X
X    1         2    3    4
B                   R1   R2
X                        R3
X    5         6    7    R4
X    X    C    X    X    W

PLACE_INFO::
W:R5M0,
A:
B:
C:
STEP_INFO_DONE::4
STEP_INFO_START::5
MAP_INFO::
X    X    A    X    X    X
X    1         2    3    R2M4
B              R1        R3
X                        R4
X    5         6    7    R5
X    X    C    X    X    W

PLACE_INFO::
W:
A:
B:
C:
STEP_INFO_DONE::5
```

```
STEP_INFO_START::6
MAP_INFO::
X    X    A    X    X    X
X    1         R1M2 3    R2M4
B                   R3   R4
X                        R5
X    5         6    7    S
X    X    C    X    X    W

PLACE_INFO::
W:
A:
B:
C:
STEP_INFO_DONE::6
STEP_INFO_START::7
MAP_INFO::
X    X    A    X    X    X
X    1         R1M2 3    R2M4
B              R3   R4   R5
X
X    5         6    7    S
X    X    C    X    X    W

PLACE_INFO::
W:
A:
B:
C:
STEP_INFO_DONE::7
STEP_INFO_START::8
MAP_INFO::
X    X    A    X    X    X
X    1         R1M2 3    4
B         R3   R4   R5   R2M4
X
X    5         6    7    S
X    X    C    X    X    W

PLACE_INFO::
W:
A:
B:
C:
STEP_INFO_DONE::8
```

```
STEP_INFO_START::9
MAP_INFO::
X    X    A    X    X    X
X    1         R1M2 R5M3 4
B    R3   R4        R2M4
X
X    5         6    7    S
X    X    C    X    X    W

PLACE_INFO::
W:
A:
B:
C:
STEP_INFO_DONE::9
STEP_INFO_START::10
MAP_INFO::
X    X    A    X    X    X
X    1         R1M2 3    4
B    R4        R2M4 R5M3
X    R3
X    5         6    7    S
X    X    C    X    X    W

PLACE_INFO::
W:
A:
B:
C:
STEP_INFO_DONE::10
STEP_INFO_START::11
MAP_INFO::
X    X    A    X    X    X
X    1         R1M2 3    4
B         R2M4 R5M3
X    R4   R3
X    5         6    7    S
X    X    C    X    X    W

PLACE_INFO::
W:
A:
B:
C:
STEP_INFO_DONE::11
```

```
STEP_INFO_START::12
MAP_INFO::
X    X    A    X    X    X
X    1         2    3    4
B    R2M4 R5M3 R1M2
X         R4   R3
X    5         6    7    S
X    X    C    X    X    W

PLACE_INFO::
W:
A:
B:
C:
STEP_INFO_DONE::12
STEP_INFO_START::13
MAP_INFO::
X    X    A    X    X    X
X    1    R5M3 2    3    4
B         R1M2
X    R2M4      R4   R3
X    5         6    7    S
X    X    C    X    X    W

PLACE_INFO::
W:
A:
B:
C:
STEP_INFO_DONE::13
STEP_INFO_START::14
MAP_INFO::
X    X    A    X    X    X
X    1    R1M2 2    3    4
B
X         R2M4      R4   R3
X    5         6    7    S
X    X    C    X    X    W

PLACE_INFO::
W:
A:R5M3,
B:
C:
STEP_INFO_DONE::14
```

```
STEP_INFO_START::15
MAP_INFO::
X    X    A    X    X    X
X    1         2    3    4
B                   R3
X                   R4
X    5    R2M4 6    7    S
X    X    C    X    X    W

PLACE_INFO::
W:
A:R1M2,R5M3,
B:
C:
STEP_INFO_DONE::15
STEP_INFO_START::16
MAP_INFO::
X    X    A    X    X    X
X    1         2    3    4
B                   R3   R4
X
X    5         6    7    S
X    X    C    X    X    W

PLACE_INFO::
W:
A:R1M2,R5M3,
B:
C:R2M4,
STEP_INFO_DONE::16
STEP_INFO_START::17
MAP_INFO::
X    X    A    X    X    X
X    1         2    3    4
B              R3   R4
X
X    5         6    7    S
X    X    C    X    X    W

PLACE_INFO::
W:
A:R1M2,R5M3,
B:
C:R2M4,
STEP_INFO_DONE::17
```

```
STEP_INFO_START::18
MAP_INFO::
X    X    A    X    X    X
X    1         R3M2 3    4
B              R4
X
X    5         6    7    S
X    X    C    X    X    W

PLACE_INFO::
W:
A:R1M2,R5M3,
B:
C:R2M4,
STEP_INFO_DONE::18
STEP_INFO_START::19
MAP_INFO::
X    X    A    X    X    X
X    1         2    3    4
B         R4   R3M2
X
X    5         6    7    S
X    X    C    X    X    W

PLACE_INFO::
W:
A:R1M2,R5M3,
B:
C:R2M4,
STEP_INFO_DONE::19
STEP_INFO_START::20
MAP_INFO::
X    X    A    X    X    X
X    1         2    3    4
B    R4   R3M2
X
X    5         6    7    S
X    X    C    X    X    W

PLACE_INFO::
W:
A:R1M2,R5M3,
B:
C:R2M4,
STEP_INFO_DONE::20
```

```
STEP_INFO_START::21
MAP_INFO::
X    X    A    X    X    X
X    1         2    3    4
B    R3M2
X    R4
X    5         6    7    S
X    X    C    X    X    W

PLACE_INFO::
W:
A:R1M2,R5M3,
B:
C:R2M4,
STEP_INFO_DONE::21
STEP_INFO_START::22
MAP_INFO::
X    X    A    X    X    X
X    1         2    3    4
B
X         R4
X    5         6    7    S
X    X    C    X    X    W

PLACE_INFO::
W:
A:R1M2,R5M3,
B:R3M2,
C:R2M4,
STEP_INFO_DONE::22
STEP_INFO_START::23
MAP_INFO::
X    X    A    X    X    X
X    1         2    3    4
B
X              R4
X    5         6    7    S
X    X    C    X    X    W

PLACE_INFO::
W:
A:R1M2,R5M3,
B:R3M2,
C:R2M4,
STEP_INFO_DONE::23
```

```
STEP_INFO_START::24
MAP_INFO::
X    X    A    X    X    X
X    1         2    3    4
B
X                   R4
X    5         6    7    S
X    X    C    X    X    W

PLACE_INFO::
W:
A:R1M2,R5M3,
B:R3M2,
C:R2M4,
STEP_INFO_DONE::24
STEP_INFO_START::25
MAP_INFO::
X    X    A    X    X    X
X    1         2    3    4
B
X                        R4
X    5         6    7    S
X    X    C    X    X    W

PLACE_INFO::
W:
A:R1M2,R5M3,
B:R3M2,
C:R2M4,
STEP_INFO_DONE::25
STEP_INFO_START::26
MAP_INFO::
X    X    A    X    X    X
X    1         2    3    4
B                        R4
X
X    5         6    7    S
X    X    C    X    X    W

PLACE_INFO::
W:
A:R1M2,R5M3,
B:R3M2,
C:R2M4,
STEP_INFO_DONE::26
```

```
STEP_INFO_START::27
MAP_INFO::
X    X    A    X    X    X
X    1         2    3    4
B                   R4
X
X    5         6    7    S
X    X    C    X    X    W

PLACE_INFO::
W:
A:R1M2,R5M3,
B:R3M2,
C:R2M4,
STEP_INFO_DONE::27
STEP_INFO_START::28
MAP_INFO::
X    X    A    X    X    X
X    1         2    3    4
B              R4
X
X    5         6    7    S
X    X    C    X    X    W

PLACE_INFO::
W:
A:R1M2,R5M3,
B:R3M2,
C:R2M4,
STEP_INFO_DONE::28
STEP_INFO_START::29
MAP_INFO::
X    X    A    X    X    X
X    1         R4M2 3    4
B
X
X    5         6    7    S
X    X    C    X    X    W

PLACE_INFO::
W:
A:R1M2,R5M3,
B:R3M2,
C:R2M4,
STEP_INFO_DONE::29
```

```
STEP_INFO_START::30
MAP_INFO::
X     X     A     X     X     X
X     1           2     3     4
B                 R4M2
X
X     5           6     7     S
X     X     C     X     X     W

PLACE_INFO::
W:
A:R1M2,R5M3,
B:R3M2,
C:R2M4,
STEP_INFO_DONE::30
STEP_INFO_START::31
MAP_INFO::
X     X     A     X     X     X
X     1           2     3     4
B           R4M2
X
X     5           6     7     S
X     X     C     X     X     W

PLACE_INFO::
W:
A:R1M2,R5M3,
B:R3M2,
C:R2M4,
STEP_INFO_DONE::31
STEP_INFO_START::32
MAP_INFO::
X     X     A     X     X     X
X     1           2     3     4
B     R4M2
X
X     5           6     7     S
X     X     C     X     X     W

PLACE_INFO::
W:
A:R1M2,R5M3,
B:R3M2,
C:R2M4,
STEP_INFO_DONE::32
```

```
STEP_INFO_START::33
MAP_INFO::
X     X     A     X     X     X
X     1           2     3     4
B
X     R4M2
X     5           6     7     S
X     X     C     X     X     W

PLACE_INFO::
W:
A:R1M2,R5M3,
B:R3M2,
C:R2M4,
STEP_INFO_DONE::33
```

```
STEP_INFO_START::34
MAP_INFO::
X     X     A     X     X     X
X     1           2     3     4
B
X           R4M2
X     5           6     7     S
X     X     C     X     X     W

PLACE_INFO::
W:
A:R1M2,R5M3,
B:R3M2,
C:R2M4,
STEP_INFO_DONE::34
STEP_INFO_START::35
MAP_INFO::
X     X     A     X     X     X
X     1           2     3     4
B
X
X     5     R4M2  6     7     S
X     X     C     X     X     W

PLACE_INFO::
W:
A:R1M2,R5M3,
B:R3M2,
C:R2M4,
STEP_INFO_DONE::35
STEP_INFO_START::36
MAP_INFO::
X     X     A     X     X     X
X     1           2     3     4
B
X
X     5           6     7     S
X     X     C     X     X     W

PLACE_INFO::
W:
A:R1M2,R5M3,
B:R3M2,
C:R2M4,R4M2,
STEP_INFO_DONE::36
simulation success! congratulation!
```