**운영체제 03분반 스레드 관리: 자동화 물류 센터 시뮬레이터 개발 보고서**

20205181 박지형

* **목차**

1. (요구사항 1) 시뮬레이터 실행 파라미터 값 파싱 및 스레드 생성 구현
2. (요구사항 2) Block, Unblock 함수 구현
3. (요구사항 3) 스레드 간 데이터 전달 시스템 구현
4. (요구사항 4, 5) 중앙 관제 노드를 통한 N개의 물류 로봇 제어 구현
5. 트러블 슈팅 사항들
6. 결과 화면

**1. (요구사항 1) 시뮬레이터 실행 파라미터 값 파싱 및 스레드 생성 구현**

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

시뮬레이터 실행 파라미터 값에는 다음과 같은 정보가 포함되어 있다.

***물류 로봇의 개수*** -> 위 그림에선 ‘5’에 해당된다.

***각 물류 로봇이 적재해야 할 물건 번호, 하역해야 할 장소 알파벳*** -> 기호 ‘ : ‘을 기준으로 각 로봇에 할당해야 할 정보가 담겨 있다. 2A중 2는 물건의 번호, A는 하역 장소에 해당된다.

프로그램의 동작은 automated\_warehouse.c 파일의 void run\_automated\_warehouse(char \*\*argv)에서 시작된다.

run\_automated\_warehouse 함수가 파라미터로 위 그림과 같은 코드를 시뮬레이터를 통해 제공받는다. 따라서 **물류 로봇의 개수**는 **argv[1]**에, **각 물류 로봇의 정보들**은 **argv[2]**에 저장되게 된다.

argv[1]은 int 형 변수 **robot\_cnt**에 따로 저장한다.

argv[2]는 strtok\_r 함수를 이용하여 기호 ‘ : ‘을 기준으로 파싱, 분리된 문자열을 다시 한 번 숫자와 문자로 분리하여 각각 저장했다.

분리된 문자들은 *전역변수* **char\* docks**에 저장했다. 해당 문자열의 인덱스는 각 로봇의 번호를 의미하게 된다.

robot\_cnt 만큼의 robot를 **struct robot \* robots**에 동적 할당한다.

이후 **setRobot** 함수를 호출, 해당 함수의 인자에 robot 객체 주소, 로봇 이름, 시작 위치(5,5 로 고정), 적재해야 할 물건 번호를 넘겨준다.

로봇의 생성을 마치고, 로봇의 상태 변화를 담당할 **로봇 스레드**와, 중앙 관제 노드 역할을 맡을 **메인 스레드**를 생성한다. (**thread\_create** 함수를 호출한다.)

로봇 스레드 내부 구현은 **test\_thread**에서, 메인 스레드는 **test\_cnt**에서 진행되었다.

test\_cnt에 내부 인자로는 robot\_cnt를, test\_thread에는 로봇의 인덱스를 추가적으로 넘겨주었다.

<구현 요구사항 1 코드 전문>

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명 텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

**2. (요구사항 2) Block, Unblock 함수 구현**

**Block**을 담당하는 함수는 aw\_thread.c에서 찾아볼 수 있었다.

함수 **void block\_thread**는 스레드의 블록 기능이 구현되어 있었으나, **블록된 해당 스레드들을 배열에 저장하는 기능**은 구현되어 있지 않았다.

제공된 자료구조 list에 선언된 **list\_push\_back** 함수를 이용해, **struct list blocked\_thread**에 블록된 스레드를 저장하는 함수를 추가적으로 구현했다.

**Unblock**은 struct list blocked\_thead에 담긴 **모든 스레드의 블록 상태를 해제하는 역할**을 담당한다.

따라서 list에 담긴 모든 스레드를 탐색하여, **thread\_unblock**함수를 통해 해당 스레드를 블록 해제시키면 된다.

**Intr\_disable** 함수와 **intr\_set\_level** 함수는 순서를 따라 동시에 호출되어, 스레드의 선점을 방지한다.

<구현 요구사항 2 코드 전문>

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

**3. (요구사항 3) 스레드 간 데이터 전달 시스템 구현**

Message Passing 방식을 이용했다.

로봇에서 중앙 관제 노드로 향하는 Message Box, **boxes\_from\_robot.**

중앙 관제 노드에서 로봇으로 향하는 Message Box, **boxes\_from\_central\_control\_node**

위 두가지 메시지 박스와, 메시지 초기화와 전달에 사용될 함수를 구현했다.

* 몇 가지 주요 함수들 -

1. void set\_message

* robot의 row, col, required\_payload, current\_payload, cmd 를 struct message에 담는다.

1. set\_message\_box\_from\_robots

* Message box인 boxes\_from\_robot에 각 로봇으로부터 온 메시지(상태 정보)를 담는다.

1. set\_message\_box\_from\_central\_control\_node

* Message box인 boxes\_from\_central\_control\_node에 중앙 관제 노드로부터의 변경된 각 로봇의 상태를 담은 메시지를 담는다

1. get\_message\_box\_from\_robots

* boxes\_from\_robot에서 메시지를 얻어온다.

<구현 요구사항 3 코드 전문>

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진

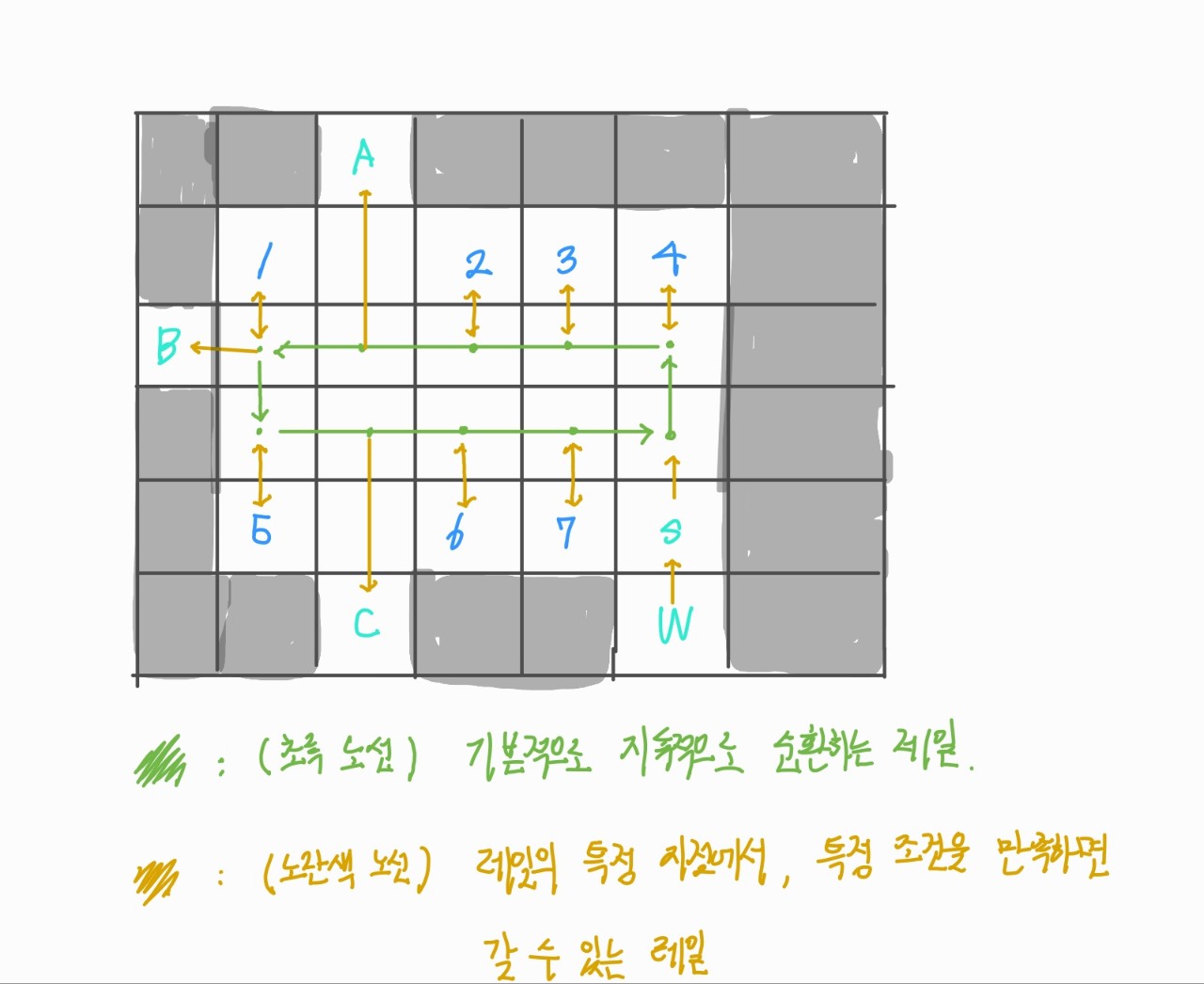
자동 생성된 설명

**4. (요구사항 4, 5) 중앙 관제 노드를 통한 N개의 물류 로봇 제어 구현**

여러 로봇들이 동일한 공간을 공유하며, 각자가 원하는 목적지에 도달해야 한다.

이 과정에서 충돌이 발생하는 것을 막아야 한다.

본인은 위 요구사항을 만족시킬 아이디어를 **지하철 노선도(혹은 버스 노선)에서 착안**했다.



모든 로봇은 **기본적으로 초록색 레일을 끊임없이 순환**한다.

초록색 레일의 **각 지점에서 해당 위치에 존재하는 로봇이 특정 조건을 만족**할 때, **로봇은 노란색 레일을 통해 원하는 물건이 놓인 장소나 하역장에 도달**할 수 있다.

로봇 간의 충돌을 막기 위해, 크게 두 가지 제약을 적용했다

첫째로, **특정 로봇이 노란색 레일 노선을 사용**하게 되면, **뒤 차례의 로봇들의 움직임을 전부 중단**한다.

(R1~R7 중 R3이 3번 물건으로 가는 레일에 탑승 -> R4~R7 전부 stop)

둘째로, **로봇의 다음 예정지에 다른 로봇이 놓여 있는 경우 stop** 한다.

메인 스레드와 로봇 스레드들을 동시에 실행시킨다.

각 로봇 스레드는 자신이 담당한 로봇의 정보를 boxes\_from\_robots 에 저장한다.

이후 해당 로봇 스레드는 Block 된다.

모든 로봇 스레드가 정보를 메시지 박스에 넣고 Block되면, 메인 스레드의 중앙 관제 로직의 작동 조건을 만족하게 된다. (list\_size(&blocked\_threads) == robot\_cnt)

메인 스레드는 메시지 박스로부터 로봇의 정보를 하나하나 차례대로 제공받는다.

이후 로봇의 위치와, 해당 로봇이 가져가야 할 물건, 도달해야 할 하역장, 위 세 가지의 위치 정보를 고려하여 다음 움직임을 정한다.

(여기서 충돌이 일어나지 않게 위에 설명했던 로직을 따른다.)

robot 구조체의 cmd 변수를 통해 모든 로봇에게 다음 이동 명령을 내린다.

(로봇이 이동할 지역에 대한 정보도 전역 변수 location\_robot에 저장한다. -> 충돌 방지를 위한 코드와 시뮬레이션 전체 종료에 사용된다.)

변경된 로봇 정보들은 메시지 박스 boxes\_from\_central\_control\_node에 저장되어 전달된다.

(cmd == 0 : stop / cmd == 1 : up / cmd == 2 : down / cmd == 3 : left / cmd == 4 : right)

(move[5][2] = {{0, 0}, {-1,0}, {1,0}, {0,-1}, {0, 1}})

이후 Block된 스레드를 모두 Unblock한다. (중앙 관제 로직은 조건을 만족하지 않기에 작동하지 않게 된다)

다시 작동을 시작한 로봇 스레드들은 메시지 박스를 통해 전달받은 정보를 로봇에 반영한다.

이후 다시 로봇의 정보를 boxes\_from\_robots에 저장하고 Block 되기를 반복한다.

위 과정을 모든 로봇이 소기의 목적을 달성할 때까지 반복한다.

모든 로봇이 하역장에 도달하면 A, B, C 세 하역장의 위치 정보를 인덱스로 한 robot\_location의 값을 더했을 때의 값이 robot\_cnt와 같아진다. (robot\_location[0][2] + robot\_location[2][0] + robot\_location[5][2] == robot\_cnt)

해당 조건을 만족하면 end\_flag를 1로 바꾸고 종료한다.

<구현 요구사항 4, 5 코드 전문> \* 코드는 따로 첨부하겠습니다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

<test\_thread>

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

<test\_cnt>

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명 텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명 텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명 텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명 텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명 텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명 텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명 텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명 텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명

**5. 트러블 슈팅**

각 로봇에게 이름을 부여할 때 잠깐 애먹었었다. snprintf 함수의 존재를 알게 되어 로봇의 이름을 문자와 숫자가 조합된 문자열로 제공할 수 있었다.

기존 코드에 구현되어 있던 thread 관련 함수들 덕에 block을 어렵지 않게 구현할 수 있었다.

Unblock이 담당해야 할 역할이 무엇인지 고민했고, list에 담긴 모든 스레드를 내뱉도록 하면 된다는 사실을 깨닫는데 조금 시간을 소요했다. 마찬가지로 list 관련 함수가 잘 구현되어 있어 적재적소에 사용할 수 있었다.

메시지 박스는 결국에는 메시지의 배열로 표현할 수 있다고 생각했고, 그에 맞추어 구현했다.

배열을 가장 많이 다뤄보아 익숙하게 구현할 수 있었기 때문에, 해당 자료구조를 택했다.

중간에 바보같이 엉뚱한 변수를 만들고, 그것을 사용해 테스트하는 바람에 많은 시간을 날려먹었다.

가장 처음에는 test\_cnt와 test\_thread가 무엇인지 깨닫지 못했다. 나중에 가서야 저 둘이 중앙 관제노드와 로봇 스레드를 의미한다는 것을 알게 되었다. 이 부분을 인지하는 것이 이번 과제를 함에 있어서 정말 중요했던 순간이었다. 과제에 대한 시야가 확 트이는 기분이었다.

로봇이 목적지까지 최단거리로 이동할 수 있게 구현할 수 있었으면 더할 나위 없을 테이지만, 여러 개의 로봇을 동시에 움직이면서 해당 조건을 만족시키기 어렵다는 사실을 스스로 받아들여야만 했다.

어쩌면 좋을지 지하철에서 열심히 생각하다, 노선도를 딱 발견하게 되었고, 지금의 로직을 떠올리게 되었다. 로봇이 지나가는 길을 고정시키고, 마치 기차의 칸처럼 줄줄이 뒤를 따라오는 형태를 머리속에 떠올렸던 것 같다. 이 방법의 장점은 모든 구역을 순환하기에 도달하지 못하는 구역이 없고, 충돌 처리에 있어서도 수월하게 해결할 수 있다는 것이다.

코드가 길어지게 된 것은 어쩔 수 없는 부분인지, 아니면 내 미흡한 코딩 실력의 결과물인지 판단이 잘 안 선다.

마지막 종료조건은 location 변수의 값의 합을 계산함으로써 해결했다. 조금 시간이 걸렸다.

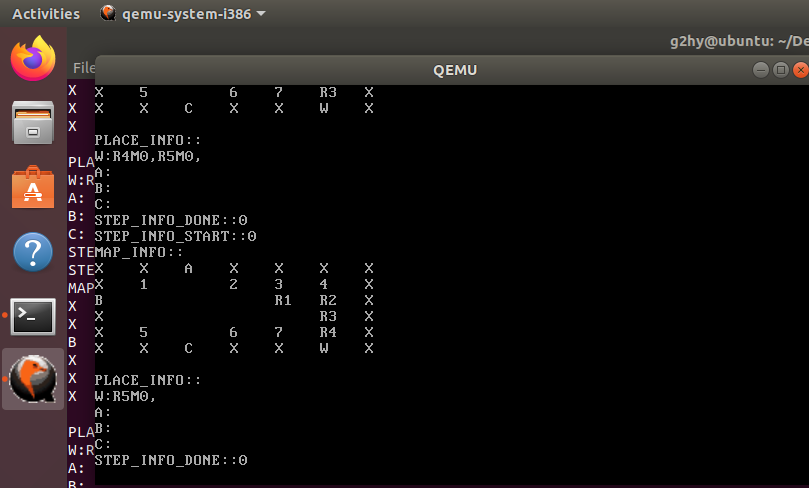
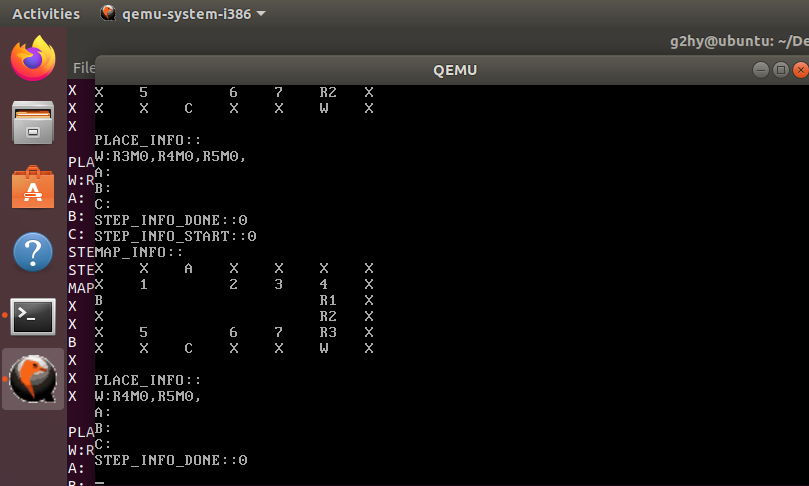
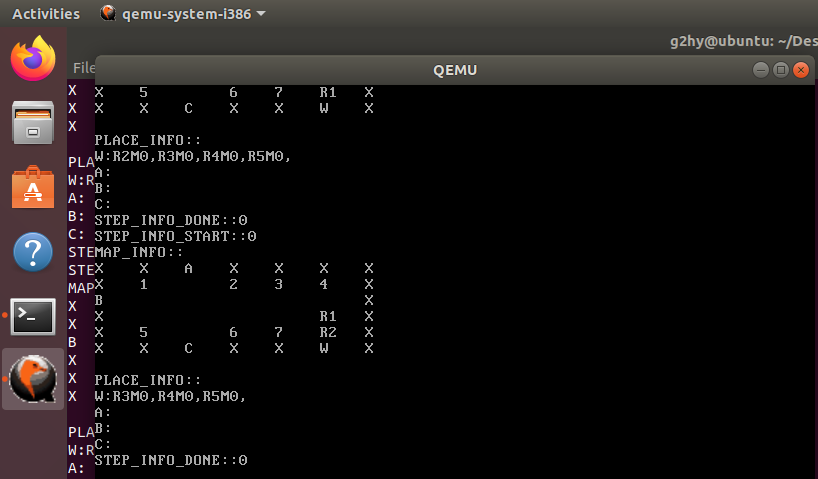
**6. 결과 화면**

**2A:4C:2B:2C:3A**

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명