스레드 관리: 자동화 물류 센터 시뮬레이터 개발

[2025 Operating System Project#1]

시계, 폰트, 상징, 그래픽이(가) 표시된 사진

자동 생성된 설명

- 2025.05.07 -

학번: 20214234

이름: 김하람

1. 프로젝트 요구사항 #0

* Pintos 환경에서 빌드 및 실행을 위해 wsl 환경에서 프로젝트 폴더를 구축하였고, run\_warehouse.sh 쉘 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\_warehouse.sh {robot\_count} {물건 번호와 하역 장소 쌍의 리스트}  
  명령으로 프로그램을 쉽게 실행할 수 있음

텍스트, 스크린샷, 폰트, 메뉴이(가) 표시된 사진

자동 생성된 설명

1. 프로젝트 요구사항 #1
2. 시뮬레이터 실행 파라미터 입력 값을 파싱하는 동작은 parse\_parametes 함수에서 처리함.

첫 번째 인자는 robot\_count 전역 변수에 저장, 두 번째 인자는 strtok\_r 함수를 주로 이용하여 colon 문자를 기준으로 파싱하여 robot\_tasks 배열에 저장함.

텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명

1. 파싱된 파라미터 값을 기준으로 중앙 관제 노드 스레드(cnn\_thread)는 run\_automated\_warehouse 함수 내에서 생성하였음.

텍스트, 스크린샷, 폰트, 디스플레이이(가) 표시된 사진

자동 생성된 설명

물류 로봇들에 대한 스레드(robot\_thread)는 중앙 관제 노드 내에서 robot 구조체 배열 생성(init\_robots) 이후 init\_robots\_threads 함수를 통해 스레드 생성함.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

1. 프로젝트 요구사항 #2

* 스레드 간 데이터 전송을 위한 Block, Unblock 함수 구현
* 이전 인터럽트 값을 old\_level 변수에 저장 후 block\_threads 리스트에 push 수행. 이후 thread\_block 함수를 호출하여 스레드를 block 상태로 변경. 스레드를 block 후에 intr\_set\_level(old\_level) 함수를 호출하여 이전 인터럽트 값으로 설정하여 선점을 방지함.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

* block\_thread와 마찬가지로 이전 인터럽트 값을 old\_level 변수에 저장 후 block\_threads 리스트에 있는 모든 값들을 꺼내어 unblock 상태로 변경. 모든 스레드에 대해 unblock을 수행한 후에 이전 인터럽트 값으로 설정함.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

1. 프로젝트 요구사항 #3

* block\_thread 및 unblock\_threads를 통한 스레드 간 데이터 전달 시스템 구현

1. message 구조체를 이용하여 스레드 간 데이터 전달 내용을 저장함. 각 변수는 구현에 필요한 역할로 해석하여 사용하였으며, is\_stopped 변수를 새로 추가함.

in aw\_message.h

|  |  |
| --- | --- |
| 변수 | 역할 |
| **로봇 -> 중앙 관제 노드** | |
| int row | 현재 로봇이 위치한 row 값 |
| int col | 현재 로봇이 위치한 col 값 |
| int current\_payload | 현재 로봇이 물건을 들고 있는지를  나타내는 값  (0: 들고 있지 않음 / 1: 들고 있음) |
| int required\_payload | 현재 로봇이 적재와 하역해야 하는 물건이  존재하는지를 나타내는 값  (0: 존재하지 않음 / 1: 존재함) |
| int is\_stopped | 현재 로봇이 멈춰 있는지를 나타내는 값  (0: 멈춰 있지 않음 / 1: 멈춰 있음) |
| **중앙 관제 노드 -> 로봇** | |
| int cmd | 로봇이 다음으로 수행해야 할 명령 |
| int target\_row | 로봇이 다음으로 움직여야 할 row 값 |
| int target\_col | 로봇이 다음으로 움직여야 할 col 값 |

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

1. 중앙 관제 노드가 각 로봇에게 전달해야 할 메시지를 전달하는 메시지 함 message\_box는 boxes\_from\_central\_node 구조체 배열에 인덱스로 구분하여 저장하게 함.

각 로봇이 중앙 관제 노드에게 전달해야 할 메시지를 전달하는 메시지 함 message\_box는 boxes\_from\_robots 구조체 배열에 인덱스로 구분하여 저장하게 함.

각 message\_box 구조체 배열의 동적 할당은 init\_message\_boexs 함수에서 수행함.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

message\_box는 수신자가 메시지를 읽었는지를 나타내기 위해 dirtyBit를 사용하였고, 0일 때 읽었음을 나타내고 1일 때 읽지 않았음을 나타냄.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

1. 물류 로봇은 while-loop 내에서 [1]맨 처음 block 이후 unblock이 되면 [2]중앙 관제 노드에게 데이터를 전달한 후에 block 됨.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

[9-1]중앙 관제 노드는 모든 물류 로봇에게 데이터를 전달 받은 후에 데이터를 기반으로 각 로봇에게 명령을 전달한 다음, [9-5] block 상태의 로봇 스레드들을 unblock 하여 주었음.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

1. 프로젝트 요구사항 #4
2. 우선, 필요한 역할에 따라 robot 구조체의 멤버 변수를 추가함.

|  |  |
| --- | --- |
| 변수 | 역할 |
| const char\* name | 로봇의 이름 R1부터 시작 |
| int index | 로봇의 인덱스 값 |
| int row | 로봇이 위치한 현재 row 값 |
| int col | 로봇이 위치한 현재 col 값 |
| int current\_payload | 로봇이 |
| int current\_payload | 현재 로봇이 물건을 들고 있는지를  나타내는 값  (0: 들고 있지 않음 / 1: 들고 있음) |
| int required\_payload | 현재 로봇이 적재와 하역해야 하는 물건이  존재하는지를 나타내는 값  (0: 존재하지 않음 / 1: 존재함) |
| char load\_location | 적재 물건이 존재하는 장소 |
| char unload\_location | 하역장 장소 |
| int load\_location\_row | 적재 물건이 존재하는 장소의 row 값 |
| int load\_location\_col | 적재 물건이 존재하는 장소의 col 값 |
| int unload\_location\_row | 하역장 장소의 row 값 |
| int unload\_location\_col | 하역장 장소의 col 값 |
| int priority | 로봇 간의 우선 순위를 나타내는 값  낮은 숫자일수록 큰 우선 순위  인덱스를 바탕으로 부여함 |
| int is\_stopped | 현재 로봇이 멈춰 있는지를 나타내는 값  (0: 멈춰 있지 않음 / 1: 멈춰 있음) |

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

1. 중앙 관제 노드는 물류 로봇 배열을 동적으로 할당해주고, setRobot 함수를 통해 초기 위치인 W 위치에 위치시킴.

setRobot 함수의 마지막 인자는 추후에 로봇끼리 충돌 검사를 수행할 때 사용할 로봇의 우선 순위를 로봇의 인덱스 값으로 설정하였음.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

앞서 파싱 하였던 파라미터 값을 바탕으로 물류 로봇이 요구하는 물건 적재 지역과 하역 장소를 robot의 멤버 변수인 load\_location과 unload\_location을 참조하여 해당 장소의 row, col 값을 저장함.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

각 물류 로봇은 boxes\_from\_central\_control\_node로부터 전달받은 명령(msg.cmd)을 바탕으로 switch-case문 내에서 {이동, 적재, 하역, 대기}의 4가지 명령 중 한 가지를 수행함.

텍스트, 스크린샷, 소프트웨어, 운영 체제이(가) 표시된 사진

자동 생성된 설명

이후에는 boxes\_from\_central\_control\_node의 dirtyBit를 0으로 설정하여 메시지를 읽었음을 표시하고, boxes\_from\_robots를 통해 상태 정보를 전달 후 dirtyBit를 1로 설정하여 메세지를 전달하였음을 나타냄.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

1. 중앙 관제 노드는 물류 로봇이 이동하기 위한 조건을 고려하여 모든 물류 로봇들이 각자의 적재 장소로 이동하여 물건 적재 이후 하역 장소로 이동하여 물건을 하역 하도록 제어함.

물류 로봇이 이동하기 위해서는 요구 사항을 고려하여 (시작점 -> 적재 장소)와 (적재 장소 -> 하역 장소)로의 이동 경로 탐색을 수행한 후에 경로에 따라 로봇이 움직이게 함.

이동 경로 탐색은 너비 우선 탐색(Breadth First Search) 알고리즘을 이용하여 최단 경로를 계산하였음. find\_shortest\_path\_by\_bfs 함수를 통하여 각 로봇이 할당 받은 적재 장소와 하역 장소에 따라 bfs 함수를 통하여 탐색한 최단 경로를 전역으로 선언된 shortest\_path\_by\_bfs 구조체 배열에 저장하였음. bfs는 Queue 자료 구조를 활용하여 구현함.

|  |  |
| --- | --- |
| 변수 | 역할 |
| shortest\_path\_by\_bfs[i] | i번째 인덱스를 가진 로봇의 최단 경로를 저장 (i < robot\_count) |
| shortest\_path\_by\_bfs[i][j] | j 값에 따라 경로의 타입을 달리 저장  j=0 : (시작점 -> 적재 장소) 경로  j=1 : (적재 장소 -> 하역 장소) 경로 |
| shortest\_path\_by\_bfs[i][j][k] | 각 경로에 따라 순서대로 location 값이 저장  (k < MAP\_HEIGHT \* MAP\_WIDTH) |



텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

shortest\_path\_by\_bfs 구조체 배열이 저장되면, 물류 로봇이 움직여야 할 경우에 assign\_next\_move 함수를 호출하여 path\_type(0: 시작 장소 -> 적재 장소 / 1: 적재 장소 -> 하역 장소)에 따라 다음 경로로 로봇이 이동하도록 row 값과 col 값을 전달해줌. 이때, step 배열을 사용하여 각 로봇별로 경로를 따라 몇 번째 위치까지 이동하였는지를 추적함.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

물류 로봇이 [9-4-c]적재 장소로 이동해야 할 경우와 [9-4-d]하역 장소로 이동해야 할 경우는 로봇 구조체 변수가 가지고 있는 current\_payload 값과 required\_payload 값에 따라 올바른 경로로 움직이도록 제어하였음.

|  |  |  |
| --- | --- | --- |
| current\_payload | required\_payload | 상태 |
| 0 | 0 | 모든 작업 종료 |
| 0 | 1 | 적재 장소로 이동 |
| 1 | 1 | 하역 장소로 이동 |

물류 로봇은 next\_location\_row 값과 next\_location\_col 값이 모두 -1일 경우에는 적재 장소 혹은 하역 장소에 도착한 상태이므로 적재 및 하역 작업을 수행하도록 명령함.

물류 로봇이 다음 경로로 움직이기 전에, check\_collision 함수를 통해 다른 로봇과 충돌하지 않는지를 확인함. check\_collision 함수는 다음의 일련 과정을 통해 충돌 검사를 수행함.

1. 현재 이동하려는 로봇이 적재 혹은 하역 중이라면 다음 이동 위치는 현재 위치로 지정하고, 적재 혹은 하역 중이 아니라면 shortest\_path\_by\_bfs 배열에서 다음 경로를 꺼내어 지정
2. 다른 모든 로봇에 대해서 다음 이동 위치를 계산. 이때, 하역 장소에는 여러 로봇이 존재할 수 있기에 해당 로봇의 current\_payload 값과 required\_payload 값이 모두 0이면 충돌 없음 반환.

다른 로봇이 어떠한 이유이든지 멈춰 있는 상태거나 다른 로봇이 적재 혹은 하역 중일 때에는 다음 이동 위치는 현재 위치로 지정. 그 경우가 아니라면 shortest\_path\_by\_bfs 배열에서 다음 경로를 꺼내어 지정.

1. 앞서 계산한 현재 이동하려는 로봇과 다른 모든 로봇과의 다음 이동 위치가 동일할 때, 현재 이동 로봇의 우선 순위가 더 작다면 충돌 발생 반환. 그 경우가 아니라면 충돌 없음 반환.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 디스플레이, 폰트이(가) 표시된 사진

자동 생성된 설명

상태 정보를 기반으로 각 물류 로봇에게 명령을 지시한 후에 모든 물류 로봇 스레드를 unblock하여 작업을 수행하도록 하고, increase\_step() 호출. 이후 중앙 관제 노드 스레드가 1초 동안 sleep하는 타이머를 작동하여 시뮬레이터처럼 작동하도록 함.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

[10]모든 로봇이 작업 수행을 마치면 while loop를 빠져나와 시뮬레이션을 종료함.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

1. 프로젝트 요구사항 #5

* 물류 로봇 스레드는 unblock 되기 전까지 대기하고 있고, 중앙 관제 노드는 모든 물류 로봇 스레드에게 작업을 지정해주고, unblock\_threads 함수를 호출함으로써 모든 물류 로봇 스레드를 unblock 상태로 지정함. 이후 물류 로봇 스레드는 지정 받은 작업을 동시에 수행함. 중앙 관제 노드는 동시에 물류 로봇 스레드를 제어하는 효과를 볼 수 있음.

(앞선 요구 사항에서 해당 동작 과정을 설명하였음)

1. 프로젝트 요구사항 #6
2. Trouble Shooting

step 배열 -> temp\_step 배열 복사