|  |
| --- |
| Operating Systems  Spring, 2022  School of Software, CAU  **Project #1**  **- A Thread Systems and Synchronization –**  **[프로젝트 보고서]**  **Template**  학번: 20182345  이름: 이 건 호 |

**1. 서론**

본 보고서에서는 교차로 문제의 해결 방안과 결과에 대해 기술한다.

**1.1 단위 스텝 문제**

*교차로 동기화와 단위스텝 부분을 따로 분리하여 구현하였다. 모든 쓰레드의 교차로 동기화 작업이 끝나면 전역변수로 선언한 count를 쓰레드당 한번씩 -1을 해주다가 count가 0이되면 단위스텝을 1 증가시키고 unitstep\_changed()를 호출하였다. 여기서 실제 구현할 때 문제가 발생했는데, 각 쓰레드가 교차로 동기화를 끝낸 후 count를 처리해 주는 부분을 critical section으로 설정해 주지 않았을 때 count==0인 상태를 가진 쓰레드가 다수 발생할 수 있어서 단위 스텝을 여러 번 증가시키게 되는 문제가 발생하였다. 추가로 적절히 block상태를 만들어 주지 않는다면 단위스텝이 증가하기도 전에 어떤 쓰레드는 다음 칸(도로 상에서 다음 경로)으로 이동해버릴 수도 있기 때문에 이 또한 critical section에서 적절히 처리해주어야 했다. 이를 해결한 방법은 본론 부분에 서술하겠다.*

**1.2 교차로 동기화 문제**

*교차로 동기화와 단위스텝 부분을 따로 분리하여 구현하였다. 교차로 동기화 부분이 먼저 작동하게 되는데 이는 기존에 구현되어 있는 map\_lock을 이용하였다. 특정 쓰레드가 교차로에 진입하기 직전에 자신이 갈 경로에 있는 교차로 도로에 대하여 lock을 시도한다. 이때 lock을 얻을 수 있으면 자신이 갈 경로에 해당되는 교차로도로에 대해 lock을 획득하고 한칸 이동한다. 그리고 교차로를 빠져나왔을 때 자신이 지나온 교차로도로를 한꺼번에 release해준다. 이번엔 교차로 진입에 실패한 경우를 설명하겠다. 만약에 이미 다른 쓰레드가 자신이 가려고 하는 경로에 있는 교차로 도로중 하나라도 lock을 보유하고있었다면 이동하지 않고 제자리에 있게 구현하였다. 이를 모든 쓰레드에 일관되게 적용되게 코딩하여서 교차로 동기화 문제를 해결하였다. 이렇게 단위 스텝부분과 교차로동기화 부분을 구분하여서 문제를 간단히 만들고자 하였다.*

**2. 본론**

본론에서는 문제 해결 방법에 대해 상세히 기술한다.

* 1. **단위 스텝 문제**

*서론부분에서 말했듯이 교차로 동기화와 단위 스텝처리를 아예 분리하였다. 어떤식으로 구현하였냐면, 모든 쓰레드는 필연적으로 vehicle\_loop함수를 구동한다. 특히 그 중에서도 주목해야 할 부분은 while문이다. 중요한 것은 이 while문이 각 쓰레드당 모두 한번씩 돌았다면 단위스텝을 증가시켜주는 처리를 해주는 것이다. (필자의 코드에서는 이렇게 해 줘야함.) 나는 이 부분의 구현을 위해 vehicle.c에 전역변수로 int num\_of\_T, int count, struct lock lock2, struct condition condition2 이렇게 4개를 선언하였다.*

*먼저 init\_on\_mainthread부분에서 전역변수를 초기화해준다.*

*텍스트이(가) 표시된 사진

자동 생성된 설명*

*num\_of\_T는 종료되지 않은 thread의 개수를 저장하기 위한 변수이고, count는 단위 스텝을 증가 시키기 위한 조건으로 사용될 변수이다.(본 코드에서는 count==0일 때 단위스텝이 증가된다.)*

*교차로 동기화가 끝나고(try\_move부분) 단위스텝 처리부분이 나타난다.*

*텍스트이(가) 표시된 사진

자동 생성된 설명*

*코드는 다음과 같다.*

*크게 보면, 본 코드는 count가 현시점에 종료되지 않은 thread의 갯수에서 시작하여 각 쓰레드가 이 부분을 통과할 때마다 1씩 줄어들면서 최종적으로 count가 0이된 쓰레드가 단위 스텝을 증가시켜 주는 방식이다.*

*이때 여러 쓰레드가 동시에 count를 감소시키면서 count==0인 부분(단위 스텝 증가)을 여러 개의 쓰레드가 접근하면 절대 안되기 때문에 전역변수로 선언했던 lock2를 사용해서 이 부분을 critical section화 시켰다. 따라서 이 부분에 접근하는 쓰레드는 순서대로 처리하게 되는데, 마지막 쓰레드가 아닌 경우에는 cond\_wait함수를 이용하여 마지막 쓰레드가 critical section에 접근할때까지 sleep상태로 대기한다. 그러다가 마지막 쓰레드가 이 영역에 접근하면, 즉 count==0이 된 상황인데, 그러면 crossroads\_step을 증가시키고 unistep\_changed()를 호출하여 단위스텝 증가 처리를 해준다. 즉, 모든 쓰레드가 자신이 할 수 있는 행동을 한 싸이클을 해줬기에 단위스텝을 증가시켜준 것이다. 그리고 count는 이제 다음 싸이클에서 다시 사용해야 하기 때문에 현재 끝나지 않고 살아있는 thread의 갯수인 num\_of\_T의 값을 count에 대입하여준다.*

*이제 단위스텝을 증가시켰으므로 앞에서 sleep상태로 들어간 thread들을 깨워줘야한다. 이는 cond\_broadcast함수를 통해서 모두 깨워주는 것을 구현하였으며, 마지막 쓰레드 자기자신이 lock을 가지고 있는 상태이므로 lock\_release를 해주면 처리가 끝나게된다. 그러면 이제 sleep에서 깨어난 쓰레드들은 이전에 대기하기 위하여 사용했던 cond\_wait의 영향에서 벗어나기 위하여 lock\_release 처리를 해준다.*

*추가로 여기에는 res==0인 경우, 즉 쓰레드가 자신이 갈 경로를 모두 지나가서 종료된 경우가 포함이 되어있다. 이때는 만약 그 종료된 쓰레드가 마지막 쓰레드일 경우에는 실행중인 쓰레드의 갯수인 num\_of\_T를 1 감소시키고 단위스텝 처리 부분을 동일하게 해준 뒤 break를 통해 while문을 빠져나간다. 종료된 쓰레드가 마지막 쓰레드가 아닐 경우에는 이미 종료된 쓰레드이기 때문에 cond\_wait에 들어가서 sleep되면 안되기 때문에 실행중인 쓰레드의 개수인 num\_of\_T만 1감소시키고 다음 쓰레드의 critical section진입을 위해 lock\_release를 해준 뒤 break를 통해 while문을 빠져나가게 된다.*

* 1. **교차로 동기화 문제**

*차량 스레드들이 교차로에 동시 진입할 때,먼저 교차로에 도착한 차량 스레드순으로 경로에 해당하는 교차로 부분들을 한꺼번에 lock해줌으로써 경로가 한칸이라도 겹치는 차량 스레드는 움직이지 못하고 return -1을 해주며 가만히 있게 처리를 해주었다. 물론 교차로 경로에 대한 lock을 획득한 차량 스레드는 정상적으로 한칸 이동한다.*

*굉장히 직관적인 방법으로 코딩하였고, 반복적인 방식이기 때문에 일부분을 예시로 들겠다. 이 예시는 교차로의 9시 방향을 A라고 두고 반시계 방향으로 B,C,D라고 가정한다.*

**

*가장 먼저, 차량 스레드의 출발지로 if,else if문을 사용하여 나눈다. 여기선 출발지가 A라고 가정하겠다.*

*텍스트이(가) 표시된 사진

자동 생성된 설명*

*그 다음 목적지로 또 나눈다. 여기선 목적지가 C라고 하겠다. 즉 A->C인 차량스레드이다.*

*위 코드를 보면 만약 해당 차량 스레드가 (4,1)위 위치 , 즉*

*텍스트, 낱말맞추기게임, 쇼지이(가) 표시된 사진

자동 생성된 설명*

*A가 교차로에 진입하기 직전의 위치에 위치한다면 자신이 갈 교차로 경로인 (4,2),(4,3),(4,4)에 대하여 lock\_try\_acquire를 통해 순서대로 lock을 얻을수 있으면 얻고 없다면 else문을 실행하게된다. 만약에 해당 교차로 경로에 대한 모든 lock을 획득하였다면 (4,2)로 이동하게 된다.(이 과정에서 자신이 지금 있는 칸을 release 해 준 후 이동하는 것이다. 그래야지 다른 쓰레드가 이동할 기회를 줄 수 있다.) else문은 왜 존재하냐면 lock\_try\_acquire를 중첩으로 교차로 경로 순서대로 시도하게 되는데, 이때 (4,2),(4,3)까지는 lock을 획득하였지만 (4,4)가 다른 차량쓰레드가 소유하고 있어서 결국엔 교차로에 진입하지 못하는 상황이라면 앞에서 lock을 획득한 (4,2),(4,3)을 release해주어서 다른 차량 쓰레드에게 이동할 수 있는 기회를 주어야 하기 때문이다. 그리고 else문 마지막을 부분을 보면 return -1 처리를 해줌으로써 이동하지 않고 대기하는 것을 구현하였다.*

*그러면 자연스럽게 어떤 교차로에 진입하기 직전의 다른 쓰레드가 자신의 경로에 (4,2),(4,3),(4,4)중에 하나라도 포함되어 있다면 lock을 획득하지 못하고 가만히 있게 된다.*

*텍스트이(가) 표시된 사진

자동 생성된 설명*

*그리고 정상적으로 교차로에 진입하였다면 위 코드처럼 while문 한번당 try\_move가 한번이기 때문에 한칸씩 자신의 다음 경로로 이동하게 된다. lock은 교차로 진입전에 한번에 획득한 상태이므로 이 과정에서 lock을 얻을 이유는 없고, 교차로 탈출 직전에 자신이 lock을 보유했던 모든 교차로에 대해 lock\_release를 해줌으로써 다른 차량 스레드가 교차로 진입을 시도할 수 있게 처리해주었다. 그리고 자신이 이동할 블록을 lock해줌으로서 다른 스레드가 못 들어오게 한다.(lock\_release를 해주는 순서도 중요하다. 순서대로 release를 해주지 않으면 동기화 과정이 꼬일 수 있다. 교차로 모양의 특성 때문이다)*

*텍스트, 스크린샷, 화면이(가) 표시된 사진

자동 생성된 설명*

*위는 교차로란 관련이 없을때의 처리이다. 즉 이미 교차로를 나왔거나 진입 직전이 아닐때의 경우인데, 자신이 다음에 갈 경로에 누군가가 lock을 보유중이라면 return-1을 통해 움직임을 제한하고, 다음 경로에 lock을 획들할 수 있고 동시에 ready상태라면 running상태로 바꿔준 후 위치를 옮겨준다(처음 시작 위치를 의미함). 이미 러닝상태인 경우 다음 경로를 lock 할 수 있는 경우에는 자신의 현재 위치의 lock을 풀어주고 다음 칸을 lock을 한다. 그리고 다음 위치로 옮겨간다.*

*이런식으로 교차로 동기화와 단위스텝을 구분하여서 해결하였다.*

**3. 결론**

*critical section설정을 조금이라도 잘못 해주면 page fault오류가 뜨는 등 프로그램 자체가 붕괴된다는 것을 경험하였다. 모든 경우의 수를 고려하여야 하기 때문에 막 코딩하기보다는 사전에 치밀하게 계획한 로직을 바탕으로 코딩해야함의 중요성을 느꼈다. 지금까지 했던 과제중에 가장 많은 시간을 쏟은 것 같다. 그만큼 얻은 것도 많은 중요한 과제인 것 같다. 특히 그냥 if문을 사용하는 것이 아닌 세마포어를 사용함으로써 atomic하게 작동하는 것의 중요성을 다시금 깨달았다.*