|  |
| --- |
| Operating Systems  Spring, 2022  School of Software, CAU  **Project #1**  **- A Thread Systems and Synchronization –**  **[프로젝트 보고서]**  학번: 20180500  이름: 장 훈 석 |
|  |

**1. 서론**

본 보고서에서는 교차로 문제의 해결 방안과 결과에 대해 기술한다.

* 1. **단위 스텝 문제**

이번 과제에서 한 턴(turn)은 모든 이동가능한 차량이 최대 한칸 이동하고, 교차로에서의 path가 겹쳐 이동하지 못하는 차량이 대기하는 행위가 이루어진다. 이러한 한 턴이 끝나고 난 뒤 단위스텝을 하나씩 올려주는 것이 단위스텝 문제이다.

**1.2 교차로 동기화 문제**

차량들이 교차로에 진입하게 되면 먼저 진입한 차량이 교차로에서 이동가능한 path들이 있고 다른 차량이 이 교차로에 진입하려고 한다면 아무런 handling을 해주지 않으면 deadlock의 가능성이 존재한다. 이러한 deadlock을 막는 것이 교차로 동기화 문제이다.

**2. 본론**

**1.1 단위 스텝 문제**

|  |  |
| --- | --- |
| 함수 | 쓰임새 |
| cond\_wait | lock을 가지고 있어야 이 함수를 호출 할 수 있음. condition에 대해 대기하고 lock을 release한 뒤 signal을 기다림. |
| cond\_signal | lock을 가진 상태에서 호출 가능하며 cond\_signal을 호출하면 cond\_wait에서 대기하고 있던 waiter들 중 하나를 wake up 시킨다. |
| cond\_broadcast | cond\_wait되어 있는 모든 WAITER들에게 SIGNAL을 날려 대기상태를 풀어준다. |

모든 전진 가능한 차량스레드가 한 칸 전진하고 이동하지 못하는 차량이 대기하는 행위를 한번의 turn이라고합니다. 이러한 하나의 turn이 지나고 나서 crossroads\_step을 1씩 증가시키는 것이 단위 스텝 문제입니다. 저는 이 문제를 해결하기 위해 프로젝트에서 제시된 condition 을 활용하였습니다. 먼저 condition은 OS에서 배웠던 monitor와 같이 동작합니다. condition 내부에는 lock, condition 이 존재합니다. lock은 monitor에서 한 쓰레드만 접근 가능하도록 하는 역할을 합니다. 모든 쓰레드는 cond\_wait, cond\_signal을 호출하기 전 lock 을 가지고 있어야합니다. 그래서 condition\_in\_while()이라는 함수를 만들어 condition과 관련된 함수 처리를 하였습니다. condition\_in\_while()함수 제일 첫 줄에 lock\_aquire(&lock)을 호출하여 모든 쓰레드가 loop함수를 돌때 lock을 갖도록 하였습니다. 단위스텝 문제를 해결하기 위해 try\_move함수를 돌고 나온 쓰레드들이 다른 쓰레드들의 동작을 기다리기 위해 cond\_wait함수를 호출하여 condition에 대해 block되도록 하였습니다. 그리고 cond\_wait함수를 호출하기 전 block된 쓰레드들의 개수를 저장하는 count변수를 ++해주었습니다. 이런식으로 count와 vehicles\_num을 비교하여 언제 한 turn이 끝나는지에 대해 측정하였습니다. 전체 running하고 있는 쓰레드들의 개수보다 count가 하나 작다면 마지막에 running하는 쓰레드임을 알 수 있고 이 조건을 만족한다면 이 쓰레드는 cond\_broadcast를 호출합니다. cond\_broadcast는 condition에 대해 block하고 있는 모든 쓰레드들을 깨우는 역할을 합니다. 이런식으로 모든 쓰레드들이 block됨을 확인하여 한 turn이 끝남을 알 수 있었습니다. 이렇게 cond\_wait함수에서 condition을 기다리고 있던 함수들이 깨어나면서 count—를 하나씩 해주며 count를 한 턴이 끝나면 0으로 초기화하였습니다. 그리고 단위스텝에 대한 처리는 cond\_broadcast를 하기 전 crossroads\_step을 하나씩 올려주고 unitstep\_changed()를 호출하여 단위 스텝을 올려 주었습니다.

* 1. **교차로 동기화 문제**

|  |  |
| --- | --- |
| 함수명 | 인자 |
| at\_crossroad | position |
| exit\_cs | position, start, dest, step, vehicle\_info |
| enter\_cs | position, start, dest, step, vehicle\_info |
| try\_move | start, dest, vehicle\_info |

차량 스레드들이 교차로에 동시에 진입할 때 발생하는 동기화 문제들로는 크게 두가지가 있습니다. 첫째로 교차로(Critical Section)을 선점하는 쓰레드(VEHICLE)가 진행하는 경로(교차로에서의 경로)와 겹치는 쓰레드(VEHICLE)들은 교차로에 진입하게 해서는 안됩니다. 둘째로 만약 교차로에 들어간 쓰레드(VEHICLE)가 교차로에서 진행하는 경로에 대해 겹치지 않는 쓰레드(VEHICLE)들은 교차로에 진입 할 수 있게끔 해야합니다.

그래서 저는 먼저 교차로에 쓰레드가 존재하는지 존재하지 않는지에 대해 판단하는 **at\_crossroad**라는 함수를 구현하였습니다. 현재 쓰레드(VEHICLE) POSITION((4,2)와 같은 raw, col로 이루어진 MAP에서의 좌표)값들과 교차로의 좌표를 비교하여 교차로에 있는지 없는지에 대한 여부를 판단하였습니다.

다음으로 저는 **enter\_cs**라는 함수를 구현하였습니다. 이 함수는 교차로에 차량이 진입할 때 lock을 거는 함수입니다. 이 함수에서 사용한 lock\_try\_acquire라는 함수는 만약 lock을 걸 수 있다면 true를 반환하고 lock도 거는 함수입니다(과제에서 주어진 함수). 먼저 다음에 진행할 position에 대해 lock을 lock\_try\_acquire함수를 사용하여 lock을 걸었습니다. 만약 다음 position을 lock할 수 없다면 return 0을 하였습니다. 그리고 만약 lock을 할 수 있다면 이 차량은 잠재적으로 교차로에 진입 할 수 있는 차량입니다. 이 차량의 다음 step을 enter\_cs함수를 재귀호출하여 교차로를 빠져나갈 때까지의 모든 path를 lock할 수 있도록 하였습니다. 그리고 enter\_cs함수를 재귀 호출 하다가 어떤 path를 lock할 수 없다면 재귀호출 되어 lock을 해놓았던 모든 path들의 lock을 순차적으로 release한 뒤 return 0를 해줬습니다. 이로써 이미 교차로에 진입한 다른 차량의 교차로에서의 path와 겹침이 있기 때문에 이렇게 release하고 진입하지 않도록 한 것입니다. 그리고 확인한 position이 교차로에 있는지에 대해 at\_crossroad 함수를 호출하여 확인하였습니다. 만약 다음 position이 교차로에 존재하지 않는다면 (!at\_crossroad가 참이라면) 1을 return함으로써 그 차량의 교차로에서의 진행 path들을 모두 lock시키고 1을 반환하게 하였습니다.

다음으로 **exit\_cs 함수를** 구현하였습니다. 이 함수는 차량이 교차로에서 빠져나갈 때 자신이 enter\_cs함수를 통해 교차로에서의 모든 path를 lock했던 것을 release해주는 함수입니다. 쓰레드가 lock을 가지는지 가지지 않는지 확인해주는 lock\_held\_by\_current\_thread함수를 통해 lock을 가지는지 가지지 않는지 확인 후 재귀를 통하여 release하게 해주었습니다.

그리고 저는 주어진 **try\_move**함수와 위에서 설명했던 at\_crossroad, enter\_cs, exit\_cs 함수를 이용하여 교차로 동기화 문제를 해결하였습니다. try\_move함수는 차량의 움직임을 통제하는 함수입니다. 이 함수를 통해 차량이 이동할지, block되어 다른 차량을 기다릴지 모든 것이 결정됩니다. 크게 두가지 경우의 수로 나눠서 진행하였습니다. 첫째로 다음 위치가 교차로에 있을경우, 다음 위치가 교차로가 아닐 경우입니다. 제일 중요한 부분은 다음 위치가 교차로에 있는경우 입니다. 이 때 교차로에 진입하기 전, 자신이 교차로에 진입 할 수 있는지 없는지, 진입한다면 어떤 경로를 lock할 것인지에 대한 것들이 모두 결정되어야 합니다. 제일 먼저 **현재는 교차로에 위치하지 않고 다음 step이 교차로에 진입하는 경우**를 살펴 보겠습니다. 이 경우엔 위에서 구현한 enter\_cs함수를 이용하였습니다. 먼저 교차로에 진입하기 전 enter\_cs함수를 호출하여 자신이 교차로의 모든 path를 lock할 수 없는지 있는지에 대해 검사하여 만약 lock할 수 있다면 모든 path를 lock 시킨 뒤 다음 position으로 이동하였습니다.이렇게 되면 enter\_cs함수 때문에 이 차량이 교차로에서 이동하는 모든 position이 lock되어 있기 때문에 동시에 진입하려는 다른 차량들 역시 enter\_cs 함수를 통해 자신이 교차로에 진입 할 수 있는지 없는지에 대한 여부를 확인 후 진입 할 수 없다면 block되어 기다리게 됩니다.

다음으로 **현재 교차로에 있고 다음 step도 교차로에 있다면** 아직 교차로에서 빠져나가지 않았기 때문에 release 해줄 것은 없습니다.

다음으로 **현재 교차로에 있고 다음 step에 교차로를 빠져나가는 경우**를 알아보겠습니다. 이때는 자신이 교차로에 진입하면서 lock 해놓았던 모든 path들의 lock을 풀어주어야 합니다. 고로 다음에 자신이 다음으로 갈 곳을 map lock을 통해 lock을 시켜준 뒤 exit\_cs 함수를 호출하여 자신이 지나온 교차로에서의 path들의 lock을 풀어주었습니다. 이로써 교차로에 진입하지 못하고 block되어 있던 다른 차량들은 교차로에 진입 할 수 있는지 없는지에 대해 다시 검사하여 진입 할 수 있는 차량은 교차로에 진입하게 됩니다.

그리고 교차로가 아닌 지점에서 교차로가 아닌 지점으로 이동하는 코드는 이미 주어져 있었습니다.하지만 저는 lock\_aquire로 되어있었던 코드를 lock\_try\_aquire로 함수를 바꾸었습니다. test case를 돌리던 중 3개 이상의 차량이 A에서 출발을 하게 되면 deadlock이 발생됨을 확인하였고 이는 두대 이상의 차량이 첫 출발지점을 점유하려고 lock\_aquire을 하게 되어서 deadlock이 발생함을 발견하게 되었습니다. 그래서 저는 lock\_try\_aquire로 바꾸어 진행하였고 이 deadlock문제를 해결 할 수 있었습니 다. 이 코드를 간단히 설명해 보면 다음 이동 할 곳을 lock하고 차량의 state를 확인해줍니다. 만약 VEHICLE\_STATUS\_READY이면 VEHICLE\_STATUS\_RUNNING으로 바꿔주어 MAP에 진입 할 수 있도록 해줍니다. 그리고 RUNNING상태의 차량들은 이동합니다.

그리고 try\_move함수에서는 모든 이동을 끝마치고 난 차량은 기존의 주어진 is\_position\_outside함수를 통해 판별 후 0을 return합니다.

이로써 map을 빠져나간 차량만 0을 return하고 map을 빠져나가지 않은 차량 중 이동을 끝마친 차량은 1을 return하고 이동하지 않은 차량들은 block됨으로써 아무것도 return하지 않고 기다립니다.

이런식으로 저는 차량 스레드들이 교차로에 동시에 진입할 때 발생하는 동기화 문제들을 교차로 자체를 critical section으로 지정하여 진입하기 전 모든 path를 lock할 수 있는지에 대해 확인한 뒤 모든 path의 lock이 불가하다면 이미 그 path를 쓸 차량이 교차로에 있음을 확인 후 그 차량이 빠져나갈 때까지 기다리는 방식으로 동기화 문제들을 해결하였습니다.

**3. 결론**

먼저 이론으로만 배웠던 세마포어와 쓰레드들 간에 공유되는 자원을 보호하기 위한 critical section같은 개념들을 직접 사용을 해보니 더욱 몸에 와닿는것 같았다. 이 프로젝트를 진행하면서 제일 어려웠던 부분은 아무래도 처음 코드를 이해하는 것이였던것 같다. lock에 관한 함수들을 어떻게 사용을 해야하는지, lock안에는 어떤식으로 세마포어가 작동하는지, 등등 초기 코드를 이해하는데 시간을 정말 많이 들였던 것 같다. 그리고 교차로를 Critical Section으로 설정하면서 나만의 세마포어를 만들어 보았는데 작동되는 결과를 보며 희열을 느꼈던 것 같다. 정말 어려웠던 프로젝트였고 시간도 많이 들인 프로젝트였기 때문에 기억에 오래남는 프로젝트가 될 것 같다.