|  |
| --- |
| Operating Systems  Spring, 2022  School of Software, CAU  **Project #1**  **- A Thread Systems and Synchronization –**  **[프로젝트 보고서]**  **Template**  학번: 20180436  이름: 김 찬 수 |

**1. 서론**

본 보고서에서는 교차로 문제의 해결 방안과 결과에 대해 기술한다.

**1.1 단위 스텝 문제**

모든 전진 가능한 차량 스레드가 한 칸 전진하고 나서 단위스텝이 1씩 증가하도록 구현하기 위해서 condition을 사용하여 모니터를 구현하여 진행을 완료한 스레드는 wait하고 있고, 각 단위 스텝의 마지막에 완료되는 스레드가 broadcast를 해서 모든 스레드를 깨우는 방법으로 진행을 할 예정입니다.

**1.2 교차로 동기화 문제**

각 차량들이 교착상태에 빠지는 것을 방지하는 것이 핵심입니다. 그러기 위해서 교차로 내부의 3x3 칸에 대해서 하나의 스레드가 진입을 하면 교차로 내부의 칸에서 사용하는 칸들을 선점하고 스레드가 교차로 외부로 나갈 때 선점했던 칸들을 풀어주는 것이 요지입니다.

**2. 본론**

본론에서는 문제 해결 방법에 대해 상세히 기술한다.

* 1. **단위 스텝 문제**

**주석 보고 설명 추가**

모든 전진 가능할 차량 스레드가 한 칸 전진하고 나서 단위 스텝이 1씩 증가하도록 구현하기 위해 condition을 사용하여 모니터를 구현하여 사용했다.어떤 임계 영역의 Mutex를 보장하기 위해 임계 영역과 관련되어 있는 차량 스레드들을 모두 재우고 오직 하나의 스레드만 임계영역에서 작업하도록 하고, 그 스레드가 작업이 끝나면 자고있는 스레드들을 broadcast로 깨워주도록 했다.

또한, 한 단위 스텝이 끝나고 다음 단위 스텝에서 몇 개의 차량 스레드를 관리해야하는지를 알기 위해 threadcnt 변수와 waitcnt 변수를 선언하여 사용했다.

Lock을 이용하여 lock\_acquire와 lock\_release를 통해 Critical Section을 vehivle\_loop 함수의 while문 내부에 구현을 하고 다른 스레드의 간섭을 방지하지 위해 Critical Section 내부에서 모든 차량 스레드가 끝났는지를 확인하였다.

try\_move 함수 동작을 마치고 마지막 남은 차량 스레드가 아닌 다른 스레드들은 waitcnt를 1 올려준 우 cond\_wait(&waitlist, &waitlock)을 통해 wait하도록 구현했고, 만약 wait가 풀린다면 waitcnt를 다시 1 내려주도록 구현했다.

마지막 차량 스레드가 try\_move함수 동작을 마치면 waitcnt가 threadcnt보다 1 큰 상황이 되는데, 이때 한 단위 스텝이 끝났다는 것을 인지할 수 있고, unitstep\_changed()를 실행해서 단위 스텝을 올려주고 cond\_broadcast로 wait하고 있던 모든 스레드들을 깨워준다.

만약 이후 1.2에서 설명하는 바와 같이 try\_move 함수의 return 값이 0이라면 동작할 수 없는 차량 스레드이므로 threadcnt를 1 줄여준다.

마지막으로 반복문을 반복하기 전에(새로운 단위 스텝을 진행하기 전에) 현재 점유 중인 스레드가 점유를 양보다고 ready\_list에 스레드를 추가할 때 사용하는 함수인 thread\_yield()함수를 호출하여 모든 스레드들이 완전히 단위 스텝을 끝낼 수 있도록 유도한다.

**1.2 교차로 동기화 문제**

차량 동기화의 문제들로는 교차로에 진입한 차들이 서로가 선점한 칸을 기다리는 교착상태, 하나의 출발지점에서 시작한 차들이 계속 줄지어서 들어와 다른 시작점을 가진 차들이 계속 기다리는 기아 상태등이 있다.

이러한 동기화 문제를 해결하기 위해서 3x3 교차로 내부의 각각의 칸을 Critical Section으로 지정하고 교차로에 진입하기 전에 각 차의 예상 경로가 선점되었는지를 확인하고 모두 이용 가능하다면 교차로로 진입하고, 선점된 칸이 하나라도 있을 경우 block 처리한다.

코드에 적용시키기 위해서

is\_position\_crossroad 함수: position이 교차로 내부인지 확인하는 함수이다. Row와 column이 둘 다 2,3,4 중 하나라면 교차로 내부에 있는 함수이므로 1을 반환하고 아니라면 0을 반환한다

try\_lock\_crossroad 함수: 교차로에 진입할 때 차량 스레드가 지나가는 칸을 선점하고, 만약 지나가고자 하는 칸이 선점되어 있다면 block되도록 구현.다음 스텝이 교차로 내부인지 외부인지 판별하고 교차로를 벗어날 때까지 재귀적으로 구현하였다.if문 내부에 재귀적으로 함수를 넣어서 depth가 깊은 곳의 함수 하나가 0을 반환하면 나머지도 0을 반환하게 해서 차량 스레드가 선점해야할 교차로 내부의 좌표를 전부 선점하거나, 전부 그대로 두도록 하였다.

release\_lock\_crossroad 함수: 교차로에서 빠져나올 때 차가 선점했던 교차로 내부의 칸을 release 해준다.

현재 교차로 내부라면 현재 좌표의 선점이 해당 차량 스레드의 것인지를 확인하고 맞다면 현재 좌표의 선점을 푼다. 그리고 step을 1을 줄여 교차로를 벗어날 때까지 전까지 재귀적으로 그 전의 지나온 교차로 내부의 좌표 또한 release한다.

try\_move 함수: vehicle의 state가 running이라면 다음 갈 칸이 끝인지를 판별하고 그렇가면 lock을 풀어준다. 교차로에 진입하려고 할 때는 사전에 만들어 둔 try\_lock\_crossroad 함수를 호출하여 차량 스레드가 교차로에서 지나가야할 길이 모두 선점당하지 않았다면, 현재 좌표의 map\_lock을 풀고 교차로로 진입한다. 교차로에서 지나가야할 칸 중 하나라도 선점당해있다면 기다린다.

교차로 내부에서 움직이는 경우 이미 칸을 lock시켜놨기 때문에 다음 좌표로 position을 옮겨주고 step을 올려준다

교차로에서 나오려고 하는 경우 다음 가야할 좌표를 선점하고, 현재 좌표를 풀어준 후 좌표를 다음 가야할 좌표로 바꿔준다.

교차로가 아닌 곳에서 아닌 곳으로 갈 경우 다음에 진행할 좌표가 선점되었는지 확인하고 선점되지 않았다면 선점하고 현재 좌표를 풀어준다. 이때 status가 ready라면 running으로 변경시켜준다.

정상적으로 진행됐다면 return 값으로 1을, 선점하지 못하거나, 끝나거나, ready 상태인 경우 0을 return해준다.

**3. 결론**

**내용 추가**

수업에서 해운 것들로 운영체제의 이론적인 부분들을 이해를 했다면, 이번 프로젝트를 수행하면서 실제로 세마포어, 모니터와 같은 것들이 어떻게 동작하고, 어떻게 사용되는지 경험을 함으로써 조금 더 깊게 운영체제에 대해서 알아볼 수 있었다.