# pthread를 이용하여 교차로에서 차량 진입 문제 해결 프로그램 구현하기

2021년 12월

김상현

## 차 례

# 1장　프로젝트 동기/목적

쓰레드를 다루는 방법을 학습하고 이를 직접 이용할 줄 알아야한다.병렬처리가 중점이 아닌 각각의 스레드의 주어진 일을 각각 실행하여야하고 critical area에 어떤 식으로 접근하여 실시간에 근접하여 동작하는 여러 스레드끼리의 동기화에 초점을 두어 진행한다.

- 이번 과제는 linux에서 pthread를 이용하기 때문에 gcc 할 때 -pthread를 추가해주어야한다.
- ex) gcc (소스파일 이름) -o (실행파일 이름) -pthread

# 2장　설계/구현

*(1) 주요 함수 설명*
*(2) 설계 아이디어 설명*
*(3) 코드의 간단 한 분석을 통한 설계 아이디어 보충*

**(1). 주요 함수 및 로직 설명 설명**

여러 스레드들을 다룰 때 제일 중요한 것은 동기화이다. 각기 다른 시점에 memory 에 접근 하는 스레드들은 변경 되지 않는 값을 사용하게될 수 도 있으며 이는 결과 및 모든 프로세스에 영향을 미친다. 즉 각 스레드에서 주요 변수에 접근하게 되었을 때는 해당 접근된 스레드를 제외한 모든 스레드는 해당 스레드가 변수를 다루고 저장이 완료될 때 까지 wait 상태가 되어 기다리게 된다. 이 때 사용되는 것이
pthread_mutex_t mutex　/ pthread_mutex_lock(&mutex) 함수 이다.

해당 함수가 호출 된 스레드는 lock 이라는 일종의 권한을 가지며 critical area 에 접근 할 수 있게 된다.

본 과제에서는 각 스레드 별로 주어진 출발경로만 다룰수 있게 하기 위해 조건 변수를 사용했는데 이 때 사용되는 것이
pthread_cond_t cond / pthread_cond_wait(&cond,&mutex) / pthread_cond_signal(&cond) 함수 이다.
lock 권한을 가진 스레드는 pthread_cond_wait() 함수를 통해 signal을 기다리고 pthread_cond_signal() 함수를 통해 다른 스레드로 signal을 보내게 된다.

스레드를 create하는 main 에서는 pthread_create() 함수를 통해서 수행할 함수, 인자, 등을 전달하며 스레드를 생성한다.

또 main에서는 input string으로 들어오는 첫 번째 차량에 해당하는 스레드로 signal을 보내줌으로써 시작하게 된다. 그 이후 main 은 나머지 모든 스레드가 끝날 때까지 다른 스레드들을 pthread_join() 함수로 기다려준다.

각 스레드들은 종료될 때 exit()나 return로 종료할수 없다. 이는 스레드의 종료가 아닌 전체 프로세스가 종료이기 때문에 의도한 바와 다르다. 이 때 사용되는 스레드 종료 함수는 pthread_exit() 함수이다.

**(2) 설계 아이디어 설명**

모든 스레드는 생성되자마자 해당하는 함수가 실행 되게 된다. 해당 함수들은 모두 mutex_lock을 가지고 곧바로 cond_wait()을 통해 waiting 상태에 빠진다.
이때 모든 스레드를 create 완료한 main 은 sleep(1)을 통해 모두 wait() 상태에 들어간 것을 확실히 하고 input string 에서의 첫 번 째 값에 해당하는 번호의 스레드로 signal을 보내게 된다.

signal을 받은 스레드는 곧바로  현재 passed와 waiting을 출력하고 tick( 초 )를 증가시킨다. input string이 비지 않았다면 해당 값을 waiting에 넣어주고 차로의 반대편에 대기중인 차량이 있는 지 확인한다.
( * waiting[4] 은 전역변수로 waiting[0] 은 1번차로에서 대기중인 차량의 수를 나타낸다 )

차량이 있다면 -> waiting에서 해당 차량을 빼고 그 차로를 담당하는 스레드로 signal을 보낸다.

차량이 없다면 -> 현재 passed와 waiting을 출력 , input string 이 비지 않았다면 waiting에 넣기, tick 증가한 후 waiting 중인 차량들중 랜덤으로 하나를 선택하여 해당 스레드로 signal을 보낸다.

이 프로그램이 정상적으로 종료되기 위해서는 create 한 모든 스레드를 정상적으로 종료해야하고 이를 main 에서 join 하여 정상적으로 종료할 수 있다.

각 스레드에서는 passed count를 증가시키며 이를 전체 input string 의 길이와 비교하여 종료 조건을 가진다. 한 스레드에서 종료조건에 참인 상황이라면 해당 스레드를 종료시키고 나머지 모든 3개의 스레드에 signal을 보낸다. 남은 모든 스레드는 passed_count 와 전체 길이를 비교한 후 스레드를 종료 시키게 된다.

### (3) 코드 설명을 통한 설계 아이디어 보충

```
void *t_function1(void *data){
        srand(time(NULL));
        while(1){
                //printf("passed_count = %d size =%d\n",passed_count,size);
                pthread_mutex_lock(&mutex1);
                pthread_cond_wait(&cond1,&mutex1);
                if(flag == 1 ){
                        pthread_exit(NULL);
                }
```

1번 스레드를 담당하는 function 이다. 스레드가 실행되면 lock이 되고 waiting 하게 되는 부분이다. 또한 종료조건을 만족하는 지에 대한 조건문도 보인다.

```
89              if(passed_count == size){
90                      flag =1;
91                      print_format();
92                      tick++;
93                      passed = 0;
94                      print_format();
95                      pthread_cond_signal(&cond2);
96                      pthread_cond_signal(&cond3);
97                      pthread_cond_signal(&cond4);
98                      pthread_exit(NULL);
99              }
```

size 와 passed_count 가 같아지는 시점은 마지막 waiting에서 대기 중이던 마지막 차량이 점거중인 상태와 같다. 주어진 바 끝난 뒤에도 한번 더 출력해야하기 때문에 이를 위해 마지막 차량이 차로를 점거중일 때의 상황을 위해 print를 먼저하고 tick을 증가시킨 뒤 passed를 초기화 함으로 써 모든 차량이 차로에 없는 상태를 print 하고 자기 자신을 종료하면서 그전에 모든 스레드에 시그널을 보내는 부분이

다.

```
if(waiting[2] !=0){

        //waiting[start_list[tick]-1]++;
        waiting[2]--;
        pthread_cond_signal(&cond3);
}
```

1번 차로에서 출발한 차량이 있을 때 다음 초에서는 3번 차로에서만 출발할 수 있다. waiting[2] 는 3번 차로에서 대기중인 차량이 있는지 파악하고 해당 차로에 대기중인 차량이 있다면 해당 차로 스레드로 시그널을 보내는 부분이다.

```
else{

        int temp;
        while(waiting[(temp = rand()%4)] == 0);
        print_format();
/*      if(passed_count >= size){
                pthread_cond_signal(&cond2);
                pthread_cond_signal(&cond3);
                pthread_cond_signal(&cond4);
                pthread_exit(NULL);
        }*/
        passed = 0;
        sleep(1);
        tick++;
        if(tick<size){
                waiting[start_list[tick]-1]++;
        }
        waiting[temp]--;
```

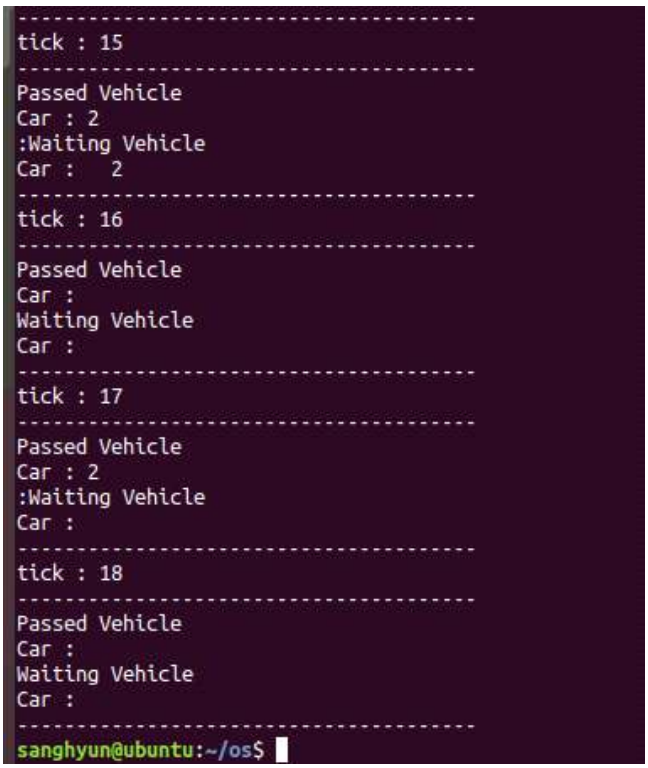
다음 출발할 차로를 고르기위해 랜덤으로 waiting 중인 차로들 중에 뽑는 과정이다.

```
if(temp ==  0 ){
        goto loop;
}
else if(temp == 1){
        pthread_cond_signal(&cond2);
}
else if(temp == 2){
        pthread_cond_signal(&cond3);
}
else if(temp == 3){
        pthread_cond_signal(&cond4);
}
```

만약 랜덤으로 선택된 값이 자기 자신이라면 굳이 signal을 보내거나 할 필요없다. 이미 자기자신이 signal을 받아 지금까지 다루고 있었기 때문이다. 그래서 goto 절을 이용하여 이전 과정을 반복하게 했다. 나머지는 해당하는 번호의 스레드로 signal을 보내는 부분이다.

# 3장 수행 결과


생성된 난수 { 1,4,3,3,1,4,2,2,2,2 }




모두 정리된 시점에서의 tick 은 18이다.

```
sanghyun@ubuntu:~/os$ ./a.out
total number of vehicles : 13
Start point :   3  4  1  3  3  2  1  2  1  2  2  2  1
tick : 1
--------------------------------------
Passed Vehicle
Car :
Waiting Vehicle
Car :
--------------------------------------
tick : 2
--------------------------------------
Passed Vehicle
Car : 3
:Waiting Vehicle
Car :   4
--------------------------------------
tick : 3
--------------------------------------
Passed Vehicle
Car :
Waiting Vehicle
Car :   1
--------------------------------------
tick : 4
--------------------------------------
Passed Vehicle
Car : 4
:Waiting Vehicle
Car :   1  3
--------------------------------------
tick : 5
--------------------------------------
Passed Vehicle
Car :
Waiting Vehicle
Car :   3  3
--------------------------------------
tick : 6
--------------------------------------
Passed Vehicle
Car : 1
:Waiting Vehicle
Car :   2  3
--------------------------------------
```

생성된 난수 { 3,4,1,3,3,2,1,2,1,2,2,2,1 }

```
tick : 18
--------------------------------------
Passed Vehicle
Car : 2
:Waiting Vehicle
Car :   2  2
--------------------------------------
tick : 19
--------------------------------------
Passed Vehicle
Car :
Waiting Vehicle
Car :   2
--------------------------------------
tick : 20
--------------------------------------
Passed Vehicle
Car : 2
:Waiting Vehicle
Car :   2
--------------------------------------
tick : 21
--------------------------------------
Passed Vehicle
Car :
Waiting Vehicle
Car :
--------------------------------------
tick : 22
--------------------------------------
Passed Vehicle
Car : 2
:Waiting Vehicle
Car :
--------------------------------------
tick : 23
--------------------------------------
Passed Vehicle
Car :
Waiting Vehicle
Car :
--------------------------------------
sanghyun@ubuntu:~/os$
```

모두 정리된 후 tick = 23;

# 4장 결론 및 보충 할 점

문제를 이해하기 위해 처음 사용했던 방법은 waiting / passed / using 으로 나누는 것이였다. passed 와 using 은 점거중임을 나타내고 waiting은 대기중은 차량들의 집합을 나타내고자 했다. 지나가기 위해 2초가 필요하기 때문에 차량이 출발하면 using으로 이동하고 tick 마다 passed 로 using을 옮긴다. 차량의 출발을 정하는 것은 현재 점거중인 passed 와 using을 보면 되는 것이다.

이런 식으로 구현했을 때의 문제는 제일 먼저 waiting table 관리가 어려웠다 새로운 값이 tick 마다 들어왔을 때 array에 추가하려고 하니 링크드 리스트 형태로 구현해보려고 했고 이를 랜덤으로 뽑는 과정이 너무 복잡해 졌다.

그래서 waiting 배열을 각 index 는 차로를 가리키고 값이 곧 대기중인 차량의 숫자를 나타내도록 했다. 이렇게 구현함으로써 using을 굳이 확인 할 필요 없었고 using을 확인 하지않고 반대편 (1 -> 3 , 2 -> 4)에 대기중이 차량이 0이 아닌지만 파악하면 되었고 이를 통해 using 변수를 제거하고 waiting 배열과 passed 만을 가지고 구현할 수 있었다.

각 조건 변수들을 모든 스레드에 하나씩 부여하도록 함으로써 특정 스레드를 찾아서 일을 배정 할 수 있었는데 메모리적인 관점에서 봤을 때 모든 스레드에 각기 다른 함수를 배정하고 mutex , cond를 부여한다는 것은 매우 비효율적일 것이라고 예측 했다.

**함수를 하나로 압축하고 해당 함수에 조건문을 추가하는 것이 메모리 적인 관점에서는 조금더 효율적이 아닐까 싶다.**