20172621 컴퓨터학부 노정환

0. 소스코드 목록

task.c 파일 하나를 -lpthread 옵션을 넣고 컴파일 하면 됩니다.

1. 구현 설명

이번 과제의 목표는 pthread를 이용해서 교차로 진입 차량을 통제하는 프로그램을 구현하는 것이다. 사용자가 10 ~ 15 사이의 정수를 입력하면, 해당 숫자만큼 무작위로 1 ~ 4 중에 숫자를 골라 목록을 만들고, 그에 맞는 출입구 번호에서 차량이 대기 상태에 놓이게 된다.

1초 간격으로 지나간 차량과 대기 중인 차량의 목록을 출력해야 하며, 차량은 출발할 수 있다면 한 번에 한 대씩만 출발해야 하고, 서로 충돌이 발생하면 안 된다.

여기서 차량의 총 수는 number에 저장하고, 이번에 지나간 차량의 목록은 passed 배열, 이번에 대기한 차량의 목록은 waited 배열에 저장한다.

과제의 핵심인 '멀티스레드로 차량 통제'는 어떻게 구현할까? 처음에는 메인 스레드에서 sleep(1)을 호출해서 1초씩 쉬고, 각 자식 스레드에서는 sleep(2)를 호출해서 2초씩 쉬는 방식을 고안했다. 그러나 다른 스레드와 시간을 맞추기 어렵고, 차량이 지나가는데 2초가 걸린다는 것을 굳이 자식 스레드에서 2초 쉬는 것으로 표현할 필요는 없으므로 메인 스레드에서만 1초씩 쉬도록 하였다.

그 다음으로 메인 스레드가 자식 스레드에게 해당 도로가 비어있다는 신호를 주고, 그에 맞게 대기 중인 자식 스레드가 동작하도록 구현하였다. 그러나 이렇게 구현한다면 마주 보는 두 대의 차량(1번과 3번, 또는 2번과 4번)이 대기 중일 때 두 대가 동시에 출발하게 된다.

따라서 메인 스레드는 가로나 세로의 도로별로 신호를 주는 대신, 각 출입구별로 해당 차선이 비어있다고 신호를 보내는데, 자식 스레드는 대기 중인 차량이 있는지 확인하고 움직인다.

2초 동안 발생하는 일들을 6단계로 정리하면, 1 -> 2 -> 3 -> (여기까지 1초) -> 4 -> 5 -> 6에서 2단계와 5단계는 자식 스레드에서, 나머지는 메인 스레드에서 일어나는 일이다.

자식 스레드를 먼저 자세히 설명하자면, 우선 4개의 자식 스레드에서는 담당하는 차선의 번호만 다르지 하는 일은 같다. 차선이 이용 가능한지 semaphore rsem으로 신호를 받고 ready 배열 안에 있는 값을 확인해서 자기 차선에 대기 중인 차량이 있는지 확인한다.
없다면 반복문을 다시 시작해서 차선에 맞는 rsem을 얻을 때까지 다시 대기 상태에 들어가고, 있다면 대기 차량 수를 감소시킨 다음 waited 배열에서 앞에 있는 차량 한 대를 지운다.(2단계) 그리고 한 번 더 rsem을 얻기까지 대기하는데, 메인 스레드에서 1초가 지나고 waited와 passed를 출력하고 갱신할 때까지 기다리는 것이다. rsem을 얻으면 이번엔 passed에 번호를 추가하고 반복문의 처음으로 돌아간다.(5단계)

메인 스레드에서는 1단계에서 passed 배열 초기화, waited 배열에 대기하는 차량 번호를 추가, ready 배열에서 번호에 맞는 값을 증가시킨 다음에 자식 스레드에 semaphore를 통해 신호를 보낸다. 자식 스레드가 2단계를 수행하는 동안 1초 동안 쉬다가 3단계에서 passed와 waited를 출력하고 처음으로 돌아간다. 4단계와 6단계는 각각 1단계, 3단계와 같다.
어느 자식 스레드에게 신호를 주는 지는 대기 중인 차선과 진행 중인 차선의 개수를 보고 정한다. garo_wait과 sero_wait에는 2번과 4번에서 대기 중인 차량의 수와 1번과 3번에서 대기 중인 차량의 수가 저장된다. garo_cnt와 sero_cnt에는 2번과 4번에서 출발해서 지나가는 차량의 수와 1번과 3번에서 출발해서 지나가는 차량의 수가 저장된다.
garo_cnt와 sero_cnt가 모두 0이면, 모든 차선이 비어있는 상태이므로 garo_wait과 sero_wait을 보고 ready 값이 양수인 번호를 찾아 여러 개일 경우 무작위로 하나를 선택한다. garo_cnt, sero_cnt 둘 중 하나가 양수일 경우, 양수인 쪽의 도로 번호들에 모두 semaphore로 신호를 보낸다. 첫 번째 1초를 보내고 두 번째 1초를 보내고 있는 차량에 대해서는 처음 1초에서 ready 값과 garo_wait 또는 sero_wait을 감소시켰으므로 중간 지점을 지나는 차량은 garo_cnt 또는 sero_cnt로만 확인할 수 있기 때문이다. 신호를 받은 도로에 대기 중인 차량이 없더라도 ready 값이 0이므로 처음으로 되돌아오기

때문에 문제없다.

garo_cnt, sero_cnt, garo_wait, sero_wait이 모두 0인 경우 number만큼의 차량이 모두 통과한 상태이므로 메인 스레드와 자식 스레드에서 반복문의 조건으로 사용한 전역 변수 def에 0을 넣어 끝날 수 있게 한다. 이때, semaphore를 기다리는 자식 스레드들을 위해 pthread_join 호출 전에 sem_post를 호출한다.

이 4개의 garo, sero 값들은 1단계와 4단계에서는 메인 스레드가 혼자 확인만 하므로 상관없지만, 2단계와 5단계에서 다수의 자식 스레드가 늘리거나 줄인다. 따라서 waited 배열과 passed 배열처럼 보호하기 위해 자식 스레드에서 다룰 때에는 semaphore wsem과 psem으로 보호해서 동시에 접근하지 못하도록 방지한다.

2. 실행 결과

```
root@ubuntu:/home/potato/task# ./task
Total number of vehicles : 10
Start point : 4 4 3 4 1 2 2 3 3 3
tick : 1
==============================
Passed Vehicle
Car
Waiting Vehicle
Car
==============================
tick : 2
==============================
Passed Vehicle
Car 4
Waiting Vehicle
Car 4
==============================
tick : 3
==============================
Passed Vehicle
Car
Waiting Vehicle
Car 4
==============================
tick : 4
==============================
Passed Vehicle
Car 3
Waiting Vehicle
Car 4 4
==============================
tick : 5
==============================
Passed Vehicle
Car
Waiting Vehicle
Car 4 1
==============================
tick : 6
==============================
Passed Vehicle
Car 4
Waiting Vehicle
Car 4 1
==============================
```

```
tick : 7
==============================
Passed Vehicle
Car 2
Waiting Vehicle
Car 1 2
==============================
tick : 8
==============================
Passed Vehicle
Car 4
Waiting Vehicle
Car 1 3
==============================
tick : 9
==============================
Passed Vehicle
Car 2
Waiting Vehicle
Car 1 3 3
==============================
tick : 10
==============================
Passed Vehicle
Car
Waiting Vehicle
Car 3 3 3
==============================
tick : 11
==============================
Passed Vehicle
Car 1
Waiting Vehicle
Car 3 3
==============================
tick : 12
==============================
Passed Vehicle
Car 3
Waiting Vehicle
Car 3 3
==============================
```

```
tick : 13
==============================
Passed Vehicle
Car
Waiting Vehicle
Car 3
==============================
tick : 14
==============================
Passed Vehicle
Car 3
Waiting Vehicle
Car 3
==============================
tick : 15
==============================
Passed Vehicle
Car
Waiting Vehicle
Car
==============================
tick : 16
==============================
Passed Vehicle
Car 3
Waiting Vehicle
Car
==============================
tick : 17
==============================
Passed Vehicle
Car
Waiting Vehicle
Car
==============================
Number of vehicles passed from each start point
P1 : 1 times
P2 : 2 times
P3 : 4 times
P4 : 3 times
Total time : 17 ticks
```

```
root@ubuntu:/home/potato/task# gcc task.c -o task -lpthread
root@ubuntu:/home/potato/task# ./task
Total number of vehicles : 15
Start point : 3 1 1 2 4 4 2 2 4 4 3 3 4 3 2
tick : 1
==============================
Passed Vehicle
Car
Waiting Vehicle
Car
==============================
tick : 2
==============================
Passed Vehicle
Car 3
Waiting Vehicle
Car
==============================
tick : 3
==============================
Passed Vehicle
Car 1
Waiting Vehicle
Car 1
==============================
tick : 4
==============================
Passed Vehicle
Car
Waiting Vehicle
Car 2
==============================
tick : 5
==============================
Passed Vehicle
Car 1
Waiting Vehicle
Car 2 4
==============================
tick : 6
==============================
Passed Vehicle
Car
Waiting Vehicle
Car 2 4
==============================
```

```
tick : 7
==============================
Passed Vehicle
Car 4
Waiting Vehicle
Car 4 2
==============================
tick : 8
==============================
Passed Vehicle
Car 2
Waiting Vehicle
Car 2 2
==============================
tick : 9
==============================
Passed Vehicle
Car 4
Waiting Vehicle
Car 2 4
==============================
tick : 10
==============================
Passed Vehicle
Car 2
Waiting Vehicle
Car 2 4
==============================
tick : 11
==============================
Passed Vehicle
Car 4
Waiting Vehicle
Car 4 3
==============================
tick : 12
==============================
Passed Vehicle
Car 2
Waiting Vehicle
Car 3 3
==============================
tick : 13
==============================
Passed Vehicle
Car 4
Waiting Vehicle
Car 3 3 4
==============================
```

```
tick : 14
==============================
Passed Vehicle
Car
Waiting Vehicle
Car 3 4 3
==============================
tick : 15
==============================
Passed Vehicle
Car 3
Waiting Vehicle
Car 3 4 3 2
==============================
tick : 16
==============================
Passed Vehicle
Car
Waiting Vehicle
Car 3 4 3
==============================
tick : 17
==============================
Passed Vehicle
Car 2
Waiting Vehicle
Car 3 3
==============================
tick : 18
==============================
Passed Vehicle
Car 4
Waiting Vehicle
Car 3 3
==============================
tick : 19
==============================
Passed Vehicle
Car
Waiting Vehicle
Car 3
==============================
tick : 20
==============================
Passed Vehicle
Car 3
Waiting Vehicle
Car 3
==============================
```

```
tick : 21
==============================
Passed Vehicle
Car
Waiting Vehicle
Car
==============================
tick : 22
==============================
Passed Vehicle
Car 3
Waiting Vehicle
Car
==============================
tick : 23
==============================
Passed Vehicle
Car
Waiting Vehicle
Car
==============================
Number of vehicles passed from each start point
P1 : 2 times
P2 : 4 times
P3 : 4 times
P4 : 5 times
Total time : 23 ticks
root@ubuntu:/home/potato/task#
```