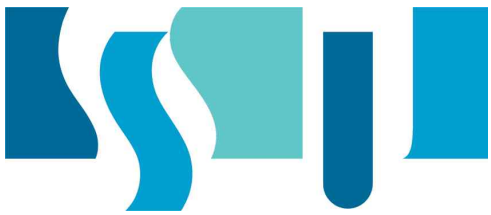


운영체제 과제 6

강의명	운영체제
교수명	김철홍 교수님
학 과	컴퓨터학부
학 번	20172655
이 름	이강산
제출일	2021.12.5.

< 목차 >

1. 개요

1-0. 요약

제출 파일 목록 : 20172655-56.c, 보고서
컴파일 및 실행 : gcc 20172655-6.c -lpthread

1-1. 동기화 기법이란

스레드는 스택 영역을 제외한 메모리를 공유하므로 적은 메모리 사용으로 병행 작업 처리에 적합하지만, 공유하는 자원에 여러 스레드가 동시에 접근하게 되면 경쟁조건에 의해 데이터의 일관성이 깨지는 문제가 발생한다. 이런 문제를 방지하게 위해 공유 자원 접근 시 한 번에 하나의 스레드만 가능하도록 순서를 조정해주는 것을 동기화 기법이라 한다.

임계구역 (Critical Section)
임계 구역은 둘 이상의 스레드가 동시에 접근해서는 안 되는 공유 자원들을 말한다. 스레드가 공유자원의 배타적인 사용을 위해 임계 구역에 들어가거나 나올 때는 뮤텍스, 세마포어 같은 동기화 매커니즘이 필요하다.

상호배제 (Mutual Exclusion)
한 스레드가 공유 자원을 접근하는 임계영역 코드를 수행중이면 다른 스레드들은 공유 자원을 접근하는 임계영역 코드를 수행할 수 없다.
- 임계영역을 보호하기 위한 개념
- 둘 이상의 프로세스가 공유자원에 대해 동시에 읽거나 쓰는 것을 방지하기 위한 기법
- 상호배제 기법에는 뮤텍스, 세마포어 등의 기법을 사용한다.

1-2. Pthread

Pthread는 Posix thread의 약자로 POSIX시스템에서 병렬적으로 작동하는 소프트웨어를 작성하기 위하여 제공하는 API이다. 사용자 수준에서의 동기화를 지원하며 Mutex lock, semaphore, 조건변수를 제공한다.

2. 구현

2-1. 소스코드

20172655-6.c

함수 프로토타입 선언

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

#define true 1



//function of threads
void *path_go1(void *null);
void *path_go2(void *null);
void *path_go3(void *null);
void *path_go4(void *null);

//tick
int tick=0;

//mutex, condition var
pthread_mutex_t mutex;
pthread_cond_t cond0, cond1, cond2, cond3, cond4;
pthread_t tid[5];
```

크리티컬 섹션, 제어 관련 변수

```
//critical section
int passing_queue[100]; // FIFO queue
int pass_front=0, pass_end=0, passed=0; //pop from front, push at end
int passed_cnt[5]; //passed cnt poped from passing queue, [0] is amount

int waiting_queue[100]; //bag
int wait_front=0, wait_end=0; //push at end, front<end
int wait_cnt[5]; //wait cnt in waiting queue, [0] is amount
```

main 함수(메인 스레드)

```
int main() {
    //INIT MUTEX, INIT COND VAR, CREATE THREADS,
    pthread_mutex_init(&mutex, NULL);
    pthread_cond_init(&cond0, NULL);
```

```
    pthread_cond_init(&cond1, NULL);
    pthread_cond_init(&cond2, NULL);
    pthread_cond_init(&cond3, NULL);
    pthread_cond_init(&cond4, NULL);
    pthread_create(&tid[1], NULL, path_go1, NULL);
    pthread_create(&tid[2], NULL, path_go2, NULL);
    pthread_create(&tid[3], NULL, path_go3, NULL);
    pthread_create(&tid[4], NULL, path_go4, NULL);
////////////////////////////////////////////////////////////////////////////////
    //CREATE STARTING POINT LIST
    int num_of_vehicle,    //user input
    *start_list,        //array, 1~4 random value
    index=0;        //cursor for start_list[]
    srand(time(NULL));
    printf("Total number of vehicles : ");
    if(1!=scanf("%d", &num_of_vehicle) ||
        !(10<=num_of_vehicle && num_of_vehicle<=15)) {
        printf("Out of Range.\n");
        exit(1);
    }
    start_list=(int*)malloc(sizeof(int)*num_of_vehicle);
    printf("Start point : ");
    for(int i=0; i<num_of_vehicle; i++) {
        start_list[i]=1+rand()%4;
        printf("%d ", start_list[i]);
    }
    printf("\n");
////////////////////////////////////////////////////////////////////////////////
    //main thread -> insert car into waiting queue, select one
    int choice;
    while(true) {
        pthread_mutex_lock(&mutex);
        if(passed_cnt[0]==num_of_vehicle) break; //모든 차량 pass 완료

        choice=0;
        tick++;
```

```
        printf("tick : %d\n", tick);
        printf("==============================\n");
////////////////////////////////////////////////////////////////////////////////////////
        if(index<num_of_vehicle) { //매 틱마다 대기큐에 하나씩 삽입
            waiting_queue[wait_end]=start_list[index];
            wait_cnt[0]++; //대기큐 내 총 개수
            wait_cnt[start_list[index]]++; //특정도로 위 대기수
            wait_end++; //대기큐 삽입커서이동
            index++; //start_list 커서이동
        }
////////////////////////////////////////////////////////////////////////////////////////

        switch(passing_queue[pass_front]) {
            case 1:
                pthread_cond_signal(&cond1);
                pthread_cond_wait(&cond0, &mutex);
                break;
            case 2:
                pthread_cond_signal(&cond2);
                pthread_cond_wait(&cond0, &mutex);
                break;
            case 3:
                pthread_cond_signal(&cond3);
                pthread_cond_wait(&cond0, &mutex);
                break;
            case 4:
                pthread_cond_signal(&cond4);
                pthread_cond_wait(&cond0, &mutex);
                break;
        }
        if(passing_queue[pass_front]==0) {//도로 위에 차 없는 경우
      if(wait_end-wait_front!=0)
                choice=waiting_queue[wait_front+rand()%(wait_end-wait_front)];//
대기큐 내에서 아무 원소나 선택, 패싱큐에 넣는다
            switch(choice) {
                case 1: //thread1 출발
                    pthread_cond_signal(&cond1);
```

```
                pthread_cond_wait(&cond0, &mutex);
                break;
            case 2: //thread2 출발
                pthread_cond_signal(&cond2);
                pthread_cond_wait(&cond0, &mutex);
                break;
            case 3: //thread3 출발
                pthread_cond_signal(&cond3);
                pthread_cond_wait(&cond0, &mutex);
                break;
            case 4: //thread4 출발
                pthread_cond_signal(&cond4);
                pthread_cond_wait(&cond0, &mutex);
                break;
        }

    }

    else { //도로 위에 차가 있는 경우 - 대응하는 길에서만 출발 가능
        if(passing_queue[pass_front]==1) { //1번도로 주행 중
            if(wait_cnt[3]>0) choice=3; //3번만 출발 가능
            else choice=0; //3번 대기차량 없으면 0
        }
        else if(passing_queue[pass_front]==2) {
            if(wait_cnt[4]>0) choice=4;
            else choice=0;
        }
        else if(passing_queue[pass_front]==3) {
            if(wait_cnt[1]>0) choice=1;
            else choice=0;
        }
        else if(passing_queue[pass_front]==4) {
            if(wait_cnt[2]>0) choice=2;
            else choice=0;
        }
        switch(choice) {
            case 1: pthread_cond_signal(&cond1);
```

```
                pthread_cond_wait(&cond0, &mutex);
                break;
            case 2: pthread_cond_signal(&cond2);
                pthread_cond_wait(&cond0, &mutex);
                break;
            case 3: pthread_cond_signal(&cond3);
                pthread_cond_wait(&cond0, &mutex);
                break;
            case 4: pthread_cond_signal(&cond4);
                pthread_cond_wait(&cond0, &mutex);
                break;
            case 0: break;
        }
    }
    //현재 달리는 중인 차 빼기(목적지도착)

    printf("Waiting Vehicle\n");
    printf("Car ");
    for(int i=wait_front; i<wait_end; i++)
        printf("%d ", waiting_queue[i]);
    printf("\n");
    printf("==============================\n");

    pthread_mutex_unlock(&mutex);
  }
  printf("Number of vehicles passed from each start point\n");
  printf("P1 : %d times\n", passed_cnt[1]);
  printf("P2 : %d times\n", passed_cnt[2]);
  printf("P3 : %d times\n", passed_cnt[3]);
  printf("P4 : %d times\n", passed_cnt[4]);
  printf("Total time : %d ticks\n", tick);

  return 0;
}
```

스레드함수

```
void *path_go1(void *null) {
  while(true) {
```

```
        pthread_mutex_lock(&mutex);
        pthread_cond_wait(&cond1, &mutex); //wait
        //대기 -> 출발 연산
        for(int i=wait_front; i<wait_end; i++) {
            if(waiting_queue[i]==1) {
                passing_queue[pass_end]=waiting_queue[i];
                wait_cnt[0]--;
                wait_cnt[waiting_queue[i]]--;
                pass_end++;
                for(int j=i; j>wait_front; j--)
                    waiting_queue[j]=waiting_queue[j-1];
                wait_front++;
                break;
            }
        }
        pthread_cond_signal(&cond0);
        pthread_cond_wait(&cond1, &mutex);
        printf("Passed   Vehicle\n");
        printf("Car 1\n");
        pthread_cond_signal(&cond0);

        pthread_cond_wait(&cond1, &mutex);
        passed=passing_queue[pass_front];
        passed_cnt[0]++;
        passed_cnt[passed]++;
        pass_front++;
        passed=0;
        pthread_cond_signal(&cond0);
        pthread_mutex_unlock(&mutex);
    }
}

void *path_go2(void *null) {
    while(true) {
        pthread_mutex_lock(&mutex);
        pthread_cond_wait(&cond2, &mutex); //wait
```

```
        //대기 -> 출발 연산
        for(int i=wait_front; i<wait_end; i++) {
            if(waiting_queue[i]==2) {
                passing_queue[pass_end]=waiting_queue[i];
                wait_cnt[0]--;
                wait_cnt[waiting_queue[i]]--;
                pass_end++;
                for(int j=i; j>wait_front; j--)
                    waiting_queue[j]=waiting_queue[j-1];
                wait_front++;
                break;
            }
        }
        pthread_cond_signal(&cond0);
        pthread_cond_wait(&cond2, &mutex);
        printf("Passed   Vehicle\n");
        printf("Car 2\n");

        pthread_cond_signal(&cond0);
        pthread_cond_wait(&cond2, &mutex);
        passed=passing_queue[pass_front];
        passed_cnt[0]++;
        passed_cnt[passed]++;
        pass_front++;
        passed=0;
        pthread_cond_signal(&cond0);
        pthread_mutex_unlock(&mutex);
    }
}

void *path_go3(void *null) {
    while(true) {
        pthread_mutex_lock(&mutex);
        pthread_cond_wait(&cond3, &mutex); //wait

        //대기 -> 출발 연산
        for(int i=wait_front; i<wait_end; i++) {
```

```
            if(waiting_queue[i]==3) {
                passing_queue[pass_end]=waiting_queue[i];
                wait_cnt[0]--;
                wait_cnt[waiting_queue[i]]--;
                pass_end++;
                for(int j=i; j>wait_front; j--)
                    waiting_queue[j]=waiting_queue[j-1];
                wait_front++;
                break;
            }
        }
        pthread_cond_signal(&cond0);
        pthread_cond_wait(&cond3, &mutex);
        printf("Passed   Vehicle\n");
        printf("Car 3\n");
        pthread_cond_signal(&cond0);

        pthread_cond_wait(&cond3, &mutex);
        passed=passing_queue[pass_front];
        passed_cnt[0]++;
        passed_cnt[passed]++;
        pass_front++;
        passed=0;
        pthread_cond_signal(&cond0);
        pthread_mutex_unlock(&mutex);
    }
}

void *path_go4(void *null) {
    while(true) {
        pthread_mutex_lock(&mutex);
        pthread_cond_wait(&cond4, &mutex); //wait

        //대기 -> 출발 연산
        for(int i=wait_front; i<wait_end; i++) {
            if(waiting_queue[i]==4) {
                passing_queue[pass_end]=waiting_queue[i];
```

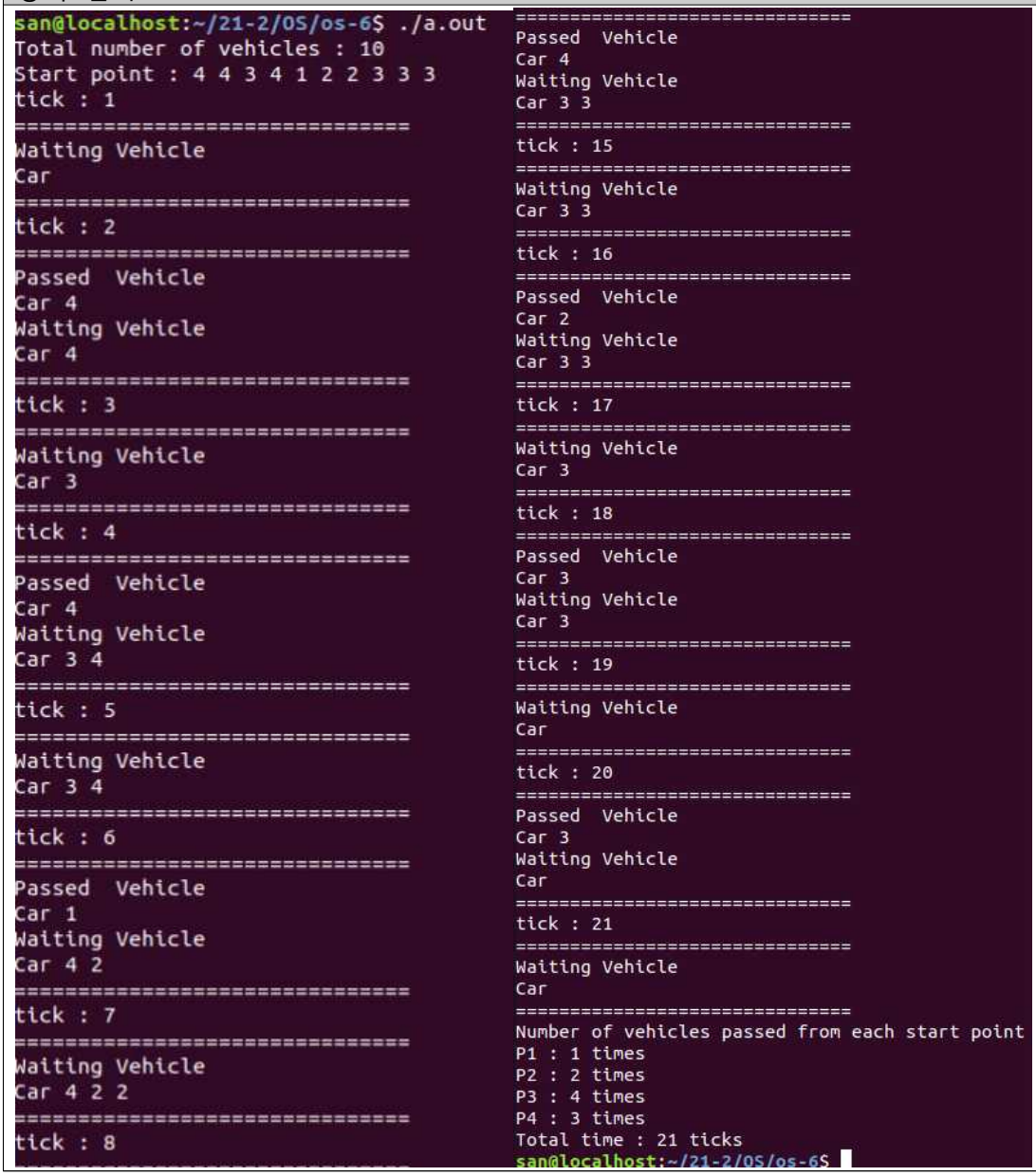
```
                wait_cnt[0]--;
                wait_cnt[waiting_queue[i]]--;
                pass_end++;
                for(int j=i; j>wait_front; j--)
                    waiting_queue[j]=waiting_queue[j-1];
                wait_front++;
                break;
            }
        }
        pthread_cond_signal(&cond0);
        pthread_cond_wait(&cond4, &mutex);
        printf("Passed   Vehicle\n");
        printf("Car 4\n");
        pthread_cond_signal(&cond0);

        pthread_cond_wait(&cond4, &mutex);
        passed=passing_queue[pass_front];
        passed_cnt[0]++;
        passed_cnt[passed]++;
        pass_front++;
        passed=0;
        pthread_cond_signal(&cond0);
        pthread_mutex_unlock(&mutex);
    }
}
```

4개의 스레드를 생성하였으며 각 스레드들은 path_go1, 2, 3, 4 함수를 수행하게 된다. 각 스레드들은 조건변수를 사용하여 waiting 상태로 signal을 기다리며, 메인함수는 필요 시 각 스레드별로 부여한 조건변수를 이용하여 작업한다. 스레드들은 waiting_queue에 들어간 대기중인 차량들 중 출발할 수 있는 것들 중에서 랜덤하게 선별하여 passing_queue에 집어넣는다.

2-2. 동작 결과

동작 결과


san@localhost:~/21-2/OS/os-6$ ./a.out
Total number of vehicles : 10
Start point : 4 4 3 4 1 2 2 3 3 3
tick : 1
Waiting Vehicle
Car
tick : 2
Passed Vehicle
Car 4
Waiting Vehicle
Car 4
tick : 3
Waiting Vehicle
Car 3
tick : 4
Passed Vehicle
Car 4
Waiting Vehicle
Car 3 4
tick : 5
Waiting Vehicle
Car 3 4
tick : 6
Passed Vehicle
Car 1
Waiting Vehicle
Car 4 2
tick : 7
Waiting Vehicle
Car 4 2 2
tick : 8
Passed Vehicle
Car 4
Waiting Vehicle
Car 3 3
tick : 15
Waiting Vehicle
Car 3 3
tick : 16
Passed Vehicle
Car 2
Waiting Vehicle
Car 3 3
tick : 17
Waiting Vehicle
Car 3
tick : 18
Passed Vehicle
Car 3
Waiting Vehicle
Car 3
tick : 19
Waiting Vehicle
Car
tick : 20
Passed Vehicle
Car 3
Waiting Vehicle
Car
tick : 21
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 : 21 ticks
san@localhost:~/21-2/OS/os-6$

3. 과제를 통해 배운 점

프로세스, 스레드들은 병행적으로 동작하며 성능을 향상시킬 수 있지만, 적절한 제어를 취해주지 않으면 데이터의 일관성을 해칠 수 있다.

스레드를 이용하여 작성한 프로그램은 경쟁 조건에 의해 불확실한 상태에 도달할 수 있으므로 공유된 자원의 접근은 배타적으로 이루어지도록 보장해야 한다. 사용자 프로그램 뿐만 아니라 커널 내부적으로도 동일한 자료구조에 접근하기 위해 수많은 경쟁조건이 발생한다. 이를 해결하기 위해서는 mutex lock, semaphore 등의 기법을 사용함으로써 mutual exclusion, progress, boundary waiting을 만족하며 공통된 자원에 접근할 수 있게 된다.