쓰레드 수행 과제

20185309
황명원

1.문제1번

1 번 소스코드 : 쓰레드를 2 개 생성해서 화면에 메시지를 출력하는 코드입니다.

수행화면:

```
hwangmyeong-won@hwangmyeong-won-ui-MacBookAir desktop % vi week6_1.c
hwangmyeong-won@hwangmyeong-won-ui-MacBookAir desktop % gcc -o week6_1 week6_1.c
 -lpthread
week6_1.c:7:1: warning: type specifier missing, defaults to 'int' [-Wimplicit-in
t]
main()
^
week6_1.c:30:1: warning: non-void function does not return a value [-Wreturn-typ
e]
}
^
2 warnings generated.
```

동작 설명:

우선 예제코드를 보기전에 알아야 할것은 pthread_create 와 pthread_join 코드입니다.
pthread_create 는 쓰레드를 생성하는 코드이며 세부적인 내용을 설명하기 위해
예시코드로 보면 이렇습니다.

(ex 코드)

int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
void *(*start_routine)(void *), void *arg);

첫번째 매개변수인 thread 는 쓰레드가 성공적으로 생성되었을때 생성된 쓰레드를
식별하기 위해서 사용되는 쓰레드 식별자입니다.
두번째 매개변수인 attr 은 쓰레드 특성을 지정하기 위해서 사용하며, 기본 쓰레드 특성을
이용하고자 할경우에 NULL 을 사용합니다.
세번째 매개변수인 start_routine 는 분기시켜서 실행할 쓰레드 함수이며,
네번째 매개변수인 arg 는 위 start_routine 쓰레드 함수의 매개변수로 넘겨집니다.
(성공적으로 리턴 할경우 0 을 리턴합니다.)

그리고 pthread_join 은 특정 쓰레드가 종료하기를 기다렸다가, 쓰레드가 종료된
이후 다음 진행하는 코드 입니다. 코드의 세부적인 내용을 설명하기 위해 예시 코드로
보면 이렇습니다.

```
int pthread_join(pthread_t th, void **thread_return);
```

첫번째 매개변수 **th** 는 기다릴 쓰레드의 식별자 입니다.

두번째 매개변수 **thread_return** 은 쓰레드의 리턴값이며 hread_return 이 NULL 이 아닌 경우 해당 포인터로 쓰레드의 리턴 값을 받아올수 있습니다.

즉 실행 결과를 보면

```
[hwangmyeong-won@hwangmyeong-won-ui-MacBookAir desktop % ./week6_1
Thread 1
Thread 2
Thread 1 returns: 0
Thread 2 returns: 0
```

pthread_create 로 쓰레드를 생성하고 pthread_join 이 쓰레드가 종료될때까지 기다렸다가 실행되는 것을 볼수있습니다.(결과화면에 첫번째줄,두번째줄)
그러고 성공적으로 pthread_create 를 반환 했으므로 둘다 반환 값은 0 이 나왔습니다.(결과화면에 세번째줄,네번째줄)

즉 이 코드를 실행할경우 쓰레드를 생성하고 종료될때까지 기다리고 실행하는 모습을 볼수 있습니다.

2.문제 2 번

2 번 코드 : 다수의 쓰레드를 생성해서 counter 값을 증가시키는 코드입니다.

수행화면:

```
[hwangmyeong-won@hwangmyeong-won-ui-MacBookAir desktop % vi week6_2.c
[hwangmyeong-won@hwangmyeong-won-ui-MacBookAir desktop % gcc -o week6_2 week6_2.c
 -Ipthread
week6_2.c:9:1: warning: type specifier missing, defaults to 'int' [-Wimplicit-in
t]
main()
^
week6_2.c:32:35: warning: format specifies type 'long' but the argument has type
 'pthread_t _Nonnull' (aka 'struct _opaque_pthread_t *') [-Wformat]
    printf("Thread number %ld\n", pthread_self());
                          ~~~    ^~~~~~~~~~~~~~
week6_2.c:36:1: warning: non-void function does not return a value [-Wreturn-typ
e]
}
^
3 warnings generated.
```

동작설명:
우선 각 코드들이 무슨 기능을 하는지(1 번문제에서 설명한 코드는 설명 제외) 알기위해 하나씩 보자면 pthread_self() 는 실행하면 현재 쓰레드의 식별자를 확인할수 있습니다.
그리고 pthread_mutex_t 라는 코드가 나오는데 mutex 는 여러개의 쓰레드가 공유하는 데이타를 보호하기 위해서 사용되는 도구로써, 보호하고자 하는 데이터를 다루는 코드영역을 한번에 하나의 쓰레드만 실행가능 하도록 하는 방법으로 공유되는 데이타를 보호합니다. 즉 수업 시간에 배운 임계 영역의

내용입니다. 코드에서 mutex 있는 부분을 없애고 실행을 해보았더니 counter 가 예측하기 힘들게 출력이 되었습니다. pthread_mutex_lock,pthread_mutex_unlock 코드를 사용하므로써 임계영역을 지정해주고 돌리니 counter 가 순서대로 잘 나오는것을 확인했습니다.

즉 여러개의 쓰레드를 생성하고 공통으로쓰는 전역변수인 counter 를 1 씩 더해줘도 서로 부딪히지않고 1 씩 올라가는 걸 코드를 실행할때 확인할수 있습니다.

수행결과:
(counter 의 증가를 계속해서 확인하기 위해 *thread_function 안에 printf("counter value: %d₩n",counter); 를 넣어 확인했습니다.)

(*pthread_mutex_lock,pthread_mutex_unlock 을 없앨경우)

```
hwangmyeong-won@hwangmyeong-won-ui-MacBookAir desktop % ./week6_2
Thread number 6172356608
Thread number 6174650368
counter value: 1
Thread number 6172930048
counter value: 2
Thread number 6175223808
counter value: 3
Thread number 6173503488
counter value: 4
Thread number 6174076928
counter value: 5
counter value: 1
Thread number 6175797248
counter value: 6
Thread number 6176370688
counter value: 7
Thread number 6176944128
counter value: 8
Thread number 6177517568
counter value: 9
Final counter value: 9
```

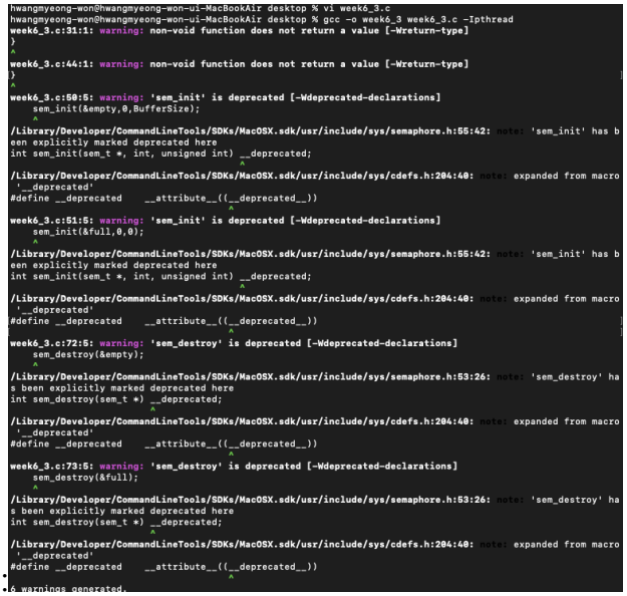
(*pthread_mutex_lock,pthread_mutex_unlock 을 넣어 실행할경우)

```
hwangmyeong-won@hwangmyeong-won-ui-MacBookAir desktop % ./week6_2
Thread number 6130348032
Thread number 6130921472
counter value: 1
Thread number 6132641792
counter value: 2
Thread number 6131494912
counter value: 3
Thread number 6133215232
counter value: 4
Thread number 6133788672
counter value: 5
counter value: 6
Thread number 6134362112
counter value: 7
Thread number 6134935552
counter value: 8
Thread number 6132068352
counter value: 9
Thread number 6135508992
counter value: 10
Final counter value: 10
```

3.문제 3 번

3 번 코드 : 생산자-소비자 동작원리를 쓰레드로 구현했습니다.

수행화면



동작설명:

우선 각 코드들이 무슨 기능을 하는지(1 번,2 번문제에서 설명한 코드는 설명 제외) 알기위해 하나씩 보자면

set_t empty, set_t full 은 세마포어 생성을 의미합니다.

sem_init(sem_t *sem, int pshared, unsigned int value); 여기서 하나씩 살펴보자면

-첫번째 인자: 세마포어 객체를 초기화 할 세마포어를 받습니다.
-두번째 인자: 여기에 0 을 주지 않을경우 sem_init 는 항상 ENOSYS 에러코드를 반환합니다.(0 을 쓰도록 합니다.)
- 세번째 인자 : 세마포어를 몇으로 초기화 할지 의미합니다.

sem_wait(sem_t *sem) 은 세마포어의 P 역할을 합니다. 즉, 세마포어를 하나 감소시키는 역할을 하고 세마포어가 0 일 경우에는 1 이상이 될 때까지 스레드는 대기

상태에 있고 0 이 아닐 경우에는 대기상태에서 빠져나와 세마포어를 또 1 감소시킵니다.

sem_post(sem_t *sem) 은 세마포어의 V 역할을 하고 세마포어 값을 1 증가 시킵니다.

sem_destroy(sem_t *sem) 은 세마포어와 관련된 리소스들을 소멸시킵니다. (객체 소멸)

usleep(값) 은 값 만큼 잠시 멈추는 코드입니다.(값/1000000 초 로 계산합니다.)

즉 main 함수에서 세마포어 empty 를 5 로 초기화 했으므로

```
for(int i = 0; i < 5; i++) {
	pthread_create(&pro[i], NULL, (void *)producer, (void *)&a[i]);
}
```

부분에서 5 번 이 수행될때

```
sem_wait(&empty);
```

이 코드로 인해 empty 라는 세마포어는 한개씩 줄어들고

```
sem_post(&full);
```

이 코드로 인해 full 이라는 세마포어는 한개씩 증가할것입니다.
그렇게 empty 는 5 번이 다돌면 0 이 되고 쓰레드는 대기 상태로 들어갑니다.
그렇게 cousumer 함수가 5 번 실행되며 이번엔 반대로 full 이 한개씩 감소하고 empty 가 한개씩 증가합니다. 이렇게 실행되는 과정은 sem_wait 와 sem_post 안에 있는 mutex 때문에 임계영역이 생겨 한개씩만 실행되며 usleep 으로 인해 0.5 초씩 쉬면서 실행되는것을 커맨드 창을 통해 확인할수 있습니다.

실행결과:

```
hwangmyeong-won@hwangmyeong-won-ui-MacBookAir desktop % ./week6_3
Producer 1: Insert Item 16887 at 0
Producer 1: Insert Item 470211272 at 1
Producer 1: Insert Item 101827544 at 2
Producer 1: Insert Item 1457850878 at 3
Producer 1: Insert Item 1458777923 at 4
Consumer 1: Remove Item 16887 from 0
Consumer 1: Remove Item 470211272 from 1
Consumer 1: Remove Item 101827544 from 2
Consumer 1: Remove Item 1457850878 from 3
Consumer 1: Remove Item 1458777923 from 4
Producer 2: Insert Item 282475249 at 0
Producer 2: Insert Item 2007237709 at 1
Producer 2: Insert Item 823564440 at 2
Producer 2: Insert Item 1115438165 at 3
Producer 2: Insert Item 1784484492 at 4
Consumer 3: Remove Item 282475249 from 0
Consumer 3: Remove Item 2007237709 from 1
Consumer 3: Remove Item 823564440 from 2
Consumer 3: Remove Item 1115438165 from 3
Consumer 3: Remove Item 1784484492 from 4
Producer 5: Insert Item 1144108930 at 0
Producer 5: Insert Item 74243042 at 1
Producer 5: Insert Item 114807987 at 2
Producer 5: Insert Item 1137522503 at 3
Producer 5: Insert Item 1441282327 at 4
Producer 4: Insert Item 984943658 at 0
Producer 4: Insert Item 16531729 at 1
Producer 4: Insert Item 823378840 at 2
Producer 4: Insert Item 143542612 at 3
Producer 4: Insert Item 896544303 at 4
Consumer 2: Remove Item 984943658 from 0
Consumer 2: Remove Item 16531729 from 1
Consumer 2: Remove Item 823378840 from 2
Consumer 2: Remove Item 143542612 from 3
Consumer 4: Remove Item 896544303 from 4
Consumer 4: Remove Item 984943658 from 0
Consumer 4: Remove Item 16531729 from 1
Consumer 4: Remove Item 823378840 from 2
Consumer 4: Remove Item 143542612 from 3
Producer 3: Insert Item 1622650073 at 0
Producer 3: Insert Item 1474833169 at 1
Producer 3: Insert Item 1264817709 at 2
Producer 3: Insert Item 1998097157 at 3
Producer 3: Insert Item 1817129560 at 4
Consumer 2: Remove Item 1817129560 from 4
Consumer 5: Remove Item 1622650073 from 0
Consumer 5: Remove Item 1474833169 from 1
Consumer 5: Remove Item 1264817709 from 2
Consumer 5: Remove Item 1998097157 from 3
Consumer 5: Remove Item 1817129560 from 4
```