Homework

1) Try below (ex1.c) and explain the result.

```
#include <stdio.h>
unsigned long long sum=0;
void main(){
    int x=fork();
  if (x==0){ // child
    int i, j;
    for(i=0;i<20000;i++)
            for(j=0;j<2000;j++)
                sum++;
    printf("child sum:%llu\n",sum);
  }else{ // parent
        int i, j;
    for(i=0;i<20000;i++)
            for(j=0;j<2000;j++)
                sum++;
    printf("parent sum:%llu\n",sum);
  }
}
```

#gcc -o ex1 ex1.c

#./ex1

```
#include <stdio.h>
void main(){
        unsigned long long sum=0;
        int x=fork();
        if(x==0){
                int i, j;
                for(i=0; i<20000; i++)
                        for(j=0; j<2000; j++)
                                sum++;
                printf("child sum: %llu\n", sum);
        }
        else{
                int i, j;
                for(i=0; i<20000; i++)
                        for(j=0; j<2000; j++)
                                sum++;
                printf("parent sum: %llu\n", sum);
        }
}
~
~
~
~
~
"ex1.c" [converted] 19L, 319C                                 19,1          All
```

강의노트대로 코드를 작성한 모습이다.

fork하여 프로세스를 복제한 후, 각각 부모와 자식 따로 2중 for문을 돌아서 20000 * 2000 만큼 sum++ 연산을 반복하고 이를 출력한다.

fork는 전체를 아예 따로 복사하는 것이므로 부모 프로세스와 자식 프로세스 모두 제대로 계산이 되어 20000 * 2000 = 40000000이 출력된 모습이다.

2) Try below (th.c) and explain the result.

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

pthread_t t1, t2;  // thread 1, thread 2
unsigned long long sum=0;

void * foo1(void *arg){
  int i,j;
  for(i=0;i<20000;i++){
    for(j=0;j<2000;j++)
      sum += 1;
  }
  printf("thread 1 sum:%llu\n", sum);
```

```
   return NULL;
}
void * foo2(void *arg){
   int i,j;
   for(i=0;i<20000;i++){
      for(j=0;j<2000;j++)
         sum += 1;
   }
   printf("thread 2 sum:%llu\n", sum);
   return NULL;
}

int main(void){
     pthread_create(&t1, NULL, &foo1, NULL);
     pthread_create(&t2, NULL, &foo2, NULL);
     pthread_join(t1, NULL);
   pthread_join(t2, NULL);
     return 0;
}

# gcc -o th -pthread th.c
#./th
....
#./th
....
#./th
....
```

```c
#include <stdio.h>
#include <string.h>
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>

pthread_t t1, t2;
unsigned long long sum=0;

void* foo1(void* arg){
        int i, j;
        for(i=0; i<20000; i++)
                for(j=0; j<2000; j++)
                        sum++;
        printf("thread 1 sum: %llu\n", sum);
        return NULL;
}

void* foo2(void* arg){
        int i, j;
        for(i=0; i<20000; i++)
                for(j=0; j<2000; j++)
                        sum++;
        printf("thread 2 sum: %llu\n", sum);
        return NULL;
}

int main(void){
        pthread_create(&t1, NULL, &foo1, NULL);
        pthread_create(&t2, NULL, &foo2, NULL);
        pthread_join(t1, NULL);
        pthread_join(t2, NULL);
        return 0;
}
                                                        34,1          Bot
```

강의노트대로 코드를 작성한 모습이다.

foo1 함수, foo2 함수 모두 전역변수인 sum을 20000 * 2000번 2중 for문을 돌며 증가시키고 이를 출력한다. thread를 이용하여 2개의 프로세스가 같은 resource를 공유하며 실행되는 코드이다.

```
localhost week7 # ./ex2
thread 1 sum: 51056132
thread 2 sum: 56370569
localhost week7 # ./ex2
thread 1 sum: 60403630
thread 2 sum: 66346328
localhost week7 # ./ex2
thread 1 sum: 49989205
thread 2 sum: 61223280
localhost week7 # ./ex2
thread 1 sum: 50797291
thread 2 sum: 58333886
localhost week7 # ./ex2
thread 1 sum: 51881864
thread 2 sum: 58869942
localhost week7 # ./ex2
thread 1 sum: 42441874
thread 2 sum: 56361236
localhost week7 # ./ex2
thread 1 sum: 47581194
thread 2 sum: 61953787
localhost week7 #
```

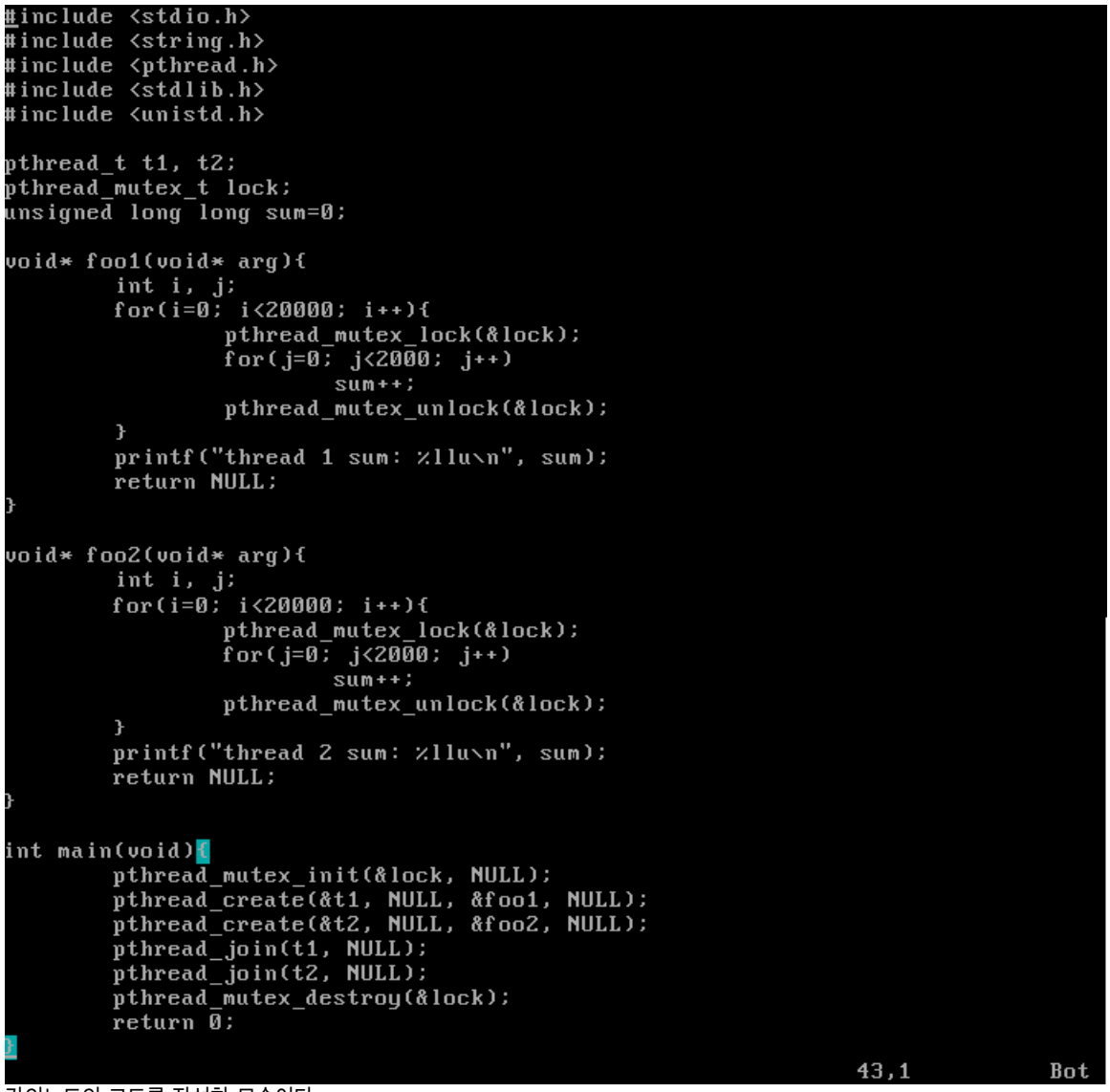
여러 번 실행한 모습이다. thread1, thread2 모두 이상한 값이 나오는 것을 확인할 수 있다.

이것은 C 상에서 sum++은 assembly 상에서는 (mov ax, sum), (inc ax), (mov sum, ax) 이런 식으로 3단계를 거치는데, thread가 mov ax, sum을 실행한 후 scheduler에 의해 stop되고 다른 thread로 넘어갔다가 온 경우 문제가 발생한다. 다른 thread에서 실행한 후 다시 돌아왔을 때 값은 sum에 저장되어있는데, mov ax, sum의 다음 명령인 inc ax 부터 실행이 되므로, 원래 저장되어있던 ax를 증가시키고 이것을 다시 sum에 넣게되어 원래의 sum 값에 overwrite하게 되어 이러한 결과가 나타난 것이다.

3. Try below(th2.c) and explain the result.

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

pthread_t t1, t2;  // thread 1, thread 2
pthread_mutex_t lock;  // semaphore
unsigned long long sum=0;

void * foo1(void *arg){
  int i,j;
  for(i=0;i<20000;i++){
    pthread_mutex_lock(&lock);
    for(j=0;j<2000;j++)
      sum += 1;
    pthread_mutex_unlock(&lock);
  }
  printf("thread 1 sum:%llu\n", sum);
  return NULL;
}
void * foo2(void *arg){
  int i,j;
  for(i=0;i<20000;i++){
      pthread_mutex_lock(&lock);
    for(j=0;j<2000;j++)
      sum += 1;
      pthread_mutex_unlock(&lock);
  }
```

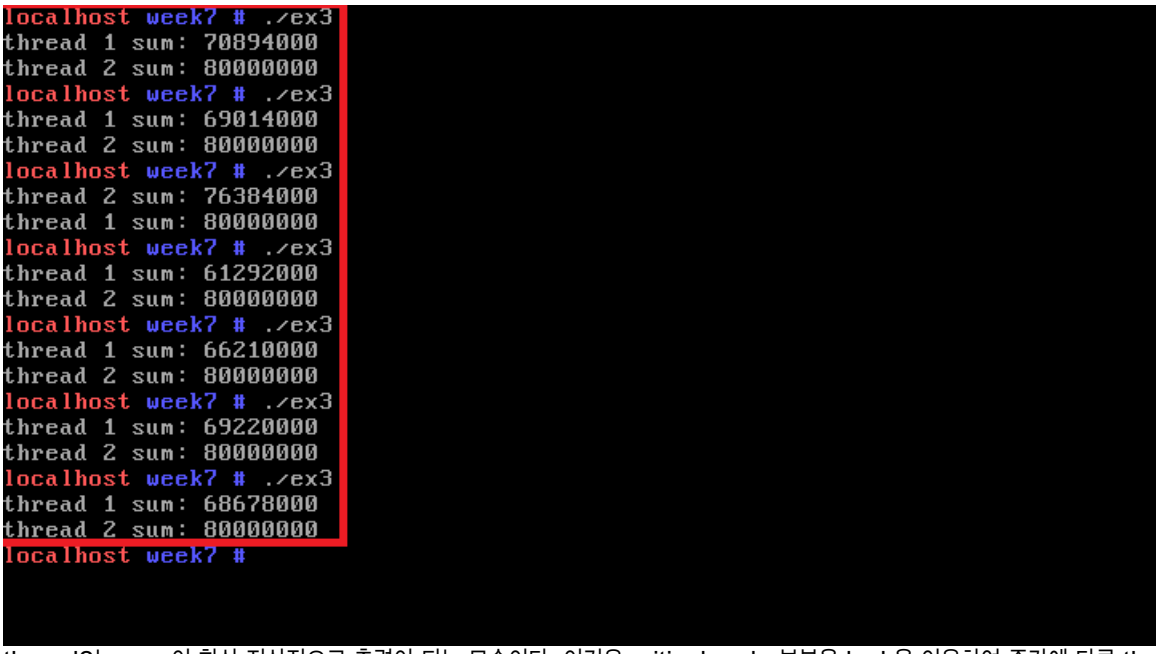
```
    printf("thread 2 sum:%llu\n", sum);
    return NULL;
}

int main(void){
    pthread_mutex_init(&lock, NULL);
      pthread_create(&t1, NULL, &foo1, NULL);
      pthread_create(&t2, NULL, &foo2, NULL);
      pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    pthread_mutex_destroy(&lock);
      return 0;
}

# gcc -o th2 -pthread th2.c
#./th2
...
#./th2
...
#./th2
.....
```



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

pthread_t t1, t2;
pthread_mutex_t lock;
unsigned long long sum=0;

void* foo1(void* arg){
        int i, j;
        for(i=0; i<20000; i++){
                pthread_mutex_lock(&lock);
                for(j=0; j<2000; j++)
                        sum++;
                pthread_mutex_unlock(&lock);
        }
        printf("thread 1 sum: %llu\n", sum);
        return NULL;
}

void* foo2(void* arg){
        int i, j;
        for(i=0; i<20000; i++){
                pthread_mutex_lock(&lock);
                for(j=0; j<2000; j++)
                        sum++;
                pthread_mutex_unlock(&lock);
        }
        printf("thread 2 sum: %llu\n", sum);
        return NULL;
}

int main(void){
        pthread_mutex_init(&lock, NULL);
        pthread_create(&t1, NULL, &foo1, NULL);
        pthread_create(&t2, NULL, &foo2, NULL);
        pthread_join(t1, NULL);
        pthread_join(t2, NULL);
        pthread_mutex_destroy(&lock);
        return 0;
}
                                                        43,1          Bot
```

강의노트의 코드를 작성한 모습이다.

foo1 함수, foo2 함수 각각 두 번째 for문을 lock을 걸어준 모습이다. 이렇게 하면 lock을 한 후, unlock이 되기 전까지는 다른 thread가 진행하려고 해도 불가능하게 된다.



thread2는 sum이 항상 정상적으로 출력이 되는 모습이다. 이것은 critical code 부분을 lock을 이용하여 중간에 다른 thread가 실행되지 못하게 막아주어 값이 overwrite되는 것을 방지하였기 때문에 정상적인 결과값이 나온 것이다. thread1은 80000000 보다 작은 값이 나오는 이유는 모든 계산이 끝나기 직전은 항상 thread2가 끝나게 되어서 thread1은 마지막 loop를 돌기 전에 이미 출력을 해서 그런 것 같다. 이 것은 숫자에 따라 달라질 것 같기도 하고, schedule이 어떻게 되느냐에 따라 달라질 것 같기도 하다.

4. (Deadlock) Try below(th3.c) and explain the result.

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

pthread_t t1, t2;  // thread 1, thread 2
pthread_mutex_t lock1;  // semaphore 1 for sum 1
pthread_mutex_t lock2;  // semaphore 2 for sum 2

unsigned long long sum1=0;
unsigned long long sum2=0;

void * foo1(void *arg){
   int i,j;
   for(i=0;i<20000;i++){
```

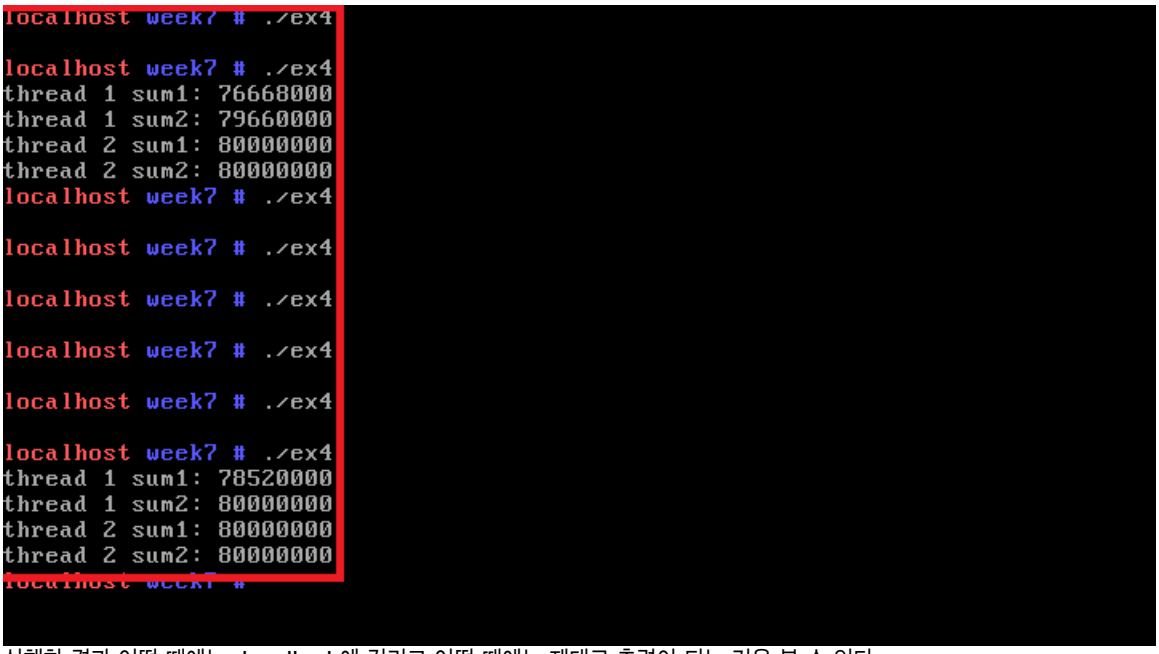
```
        pthread_mutex_lock(&lock1);
        pthread_mutex_lock(&lock2);
    for(j=0;j<2000;j++)
      sum1 += 1;
        pthread_mutex_unlock(&lock1);
        for(j=0;j<2000;j++)
      sum2 += 1;
        pthread_mutex_unlock(&lock2);
  }
  printf("thread 1 sum1:%llu\n", sum1);
    printf("thread 1 sum2:%llu\n", sum2);

  return NULL;
}
void * foo2(void *arg){
  int i,j;
  for(i=0;i<20000;i++){
        pthread_mutex_lock(&lock2);
        pthread_mutex_lock(&lock1);
    for(j=0;j<2000;j++)
      sum1 += 1;
        pthread_mutex_unlock(&lock1);
        for(j=0;j<2000;j++)
      sum2 += 1;
     pthread_mutex_unlock(&lock2);
  }
  printf("thread 2 sum1:%llu\n", sum1);
    printf("thread 2 sum2:%llu\n", sum2);

  return NULL;
}

int main(void){
    pthread_mutex_init(&lock1, NULL);
  pthread_mutex_init(&lock2, NULL);
    pthread_create(&t1, NULL, &foo1, NULL);
    pthread_create(&t2, NULL, &foo2, NULL);
```

```
    pthread_join(t1, NULL);
  pthread_join(t2, NULL);
  pthread_mutex_destroy(&lock1);
    pthread_mutex_destroy(&lock2);

    return 0;
}

# gcc -o th3 -pthread th3.c
#./th3
...
#./th3
...
#./th3
.....
```

```c
#include <stdio.h>
#include <string.h>
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>

pthread_t t1, t2;
pthread_mutex_t lock1;
pthread_mutex_t lock2;
unsigned long long sum1=0;
unsigned long long sum2=0;

void* foo1(void* arg){
        int i, j;
        for(i=0; i<20000; i++){
                pthread_mutex_lock(&lock1);
                pthread_mutex_lock(&lock2);
                for(j=0; j<2000; j++)
                        sum1++;
                pthread_mutex_unlock(&lock1);
                for(j=0; j<2000; j++)
                        sum2++;
                pthread_mutex_unlock(&lock2);
        }
        printf("thread 1 sum1: %llu\n", sum1);
        printf("thread 1 sum2: %llu\n", sum2);
        return NULL;
}

void* foo2(void* arg){
        int i, j;
        for(i=0; i<20000; i++){
                pthread_mutex_lock(&lock2);
                pthread_mutex_lock(&lock1);
                for(j=0; j<2000; j++)
                        sum1++;
                pthread_mutex_unlock(&lock1);
                for(j=0; j<2000; j++)
                        sum2++;
                pthread_mutex_unlock(&lock2);
        }
        printf("thread 2 sum1: %llu\n", sum1);
        printf("thread 2 sum2: %llu\n", sum2);
        return NULL;
}

int main(void){
        pthread_mutex_init(&lock1, NULL);
        pthread_mutex_init(&lock2, NULL);
        pthread_create(&t1, NULL, &foo1, NULL);
        pthread_create(&t2, NULL, &foo2, NULL);
        pthread_join(t1, NULL);
        pthread_join(t2, NULL);
        pthread_mutex_destroy(&lock1);
        pthread_mutex_destroy(&lock2);
        return 0;
}
                                                        57,1          Bot
```

강의노트의 코드대로 작성한 모습이다.

lock을 할 수 있는 것을 2개를 만들어서 foo1 함수는 lock1, lock2 순서대로 lock을 하고, foo2 함수는 lock2, lock1 순서대로 lock을 하게 되어있다.



실행한 결과 어떨 때에는 deadlock에 걸리고 어떨 때에는 제대로 출력이 되는 것을 볼 수 있다.

deadlock이 걸리는 이유는 만약 foo1에서 lock1을 lock한 후에 thread가 넘어갔다고 치면, foo2에서는 lock2를 먼저 lock 할 것이다. 그리고 lock1을 접근하려고 하는데 foo1에서 lock1을 lock하였으므로 실행이 안되므로 다시 foo1로 thread가 넘어갈 것이다. 근데 foo1에서는 lock1을 lock 한 후 넘어갔으므로 다음 명령은 lock2에 접근하는데, 이 것은 foo2에서 lock2를 lock을 해버렸으므로 실행이 안되서 다시 foo2로 넘어가는 무한 loop에 빠지게 된다. 이렇게 thread가 lock되어서 서로 기다리는 상태가 되버리는 것이 deadlock이다. 정상적으로 출력이 된 경우는 foo1 에서 lock1, lock2 사이, foo2에서 lock2, lock1 사이에서 thread가 넘어간 적이 한번도 없는 경우에는 정상적으로 출력이 된다.