1번 문제 수행화면

```
hunterspin@ubuntu:~/Desktop/practice$ gcc -o thread thread.c -lpthread
hunterspin@ubuntu:~/Desktop/practice$ ./thread
Thread 1
Thread 2
Thread 1 returns: 0
Thread 2 returns: 0
```

2번 문제 수행화면

```
hunterspin@ubuntu:~/Desktop/practice$ gcc -o multiThread multiThread.c -lpthread
hunterspin@ubuntu:~/Desktop/practice$ ./multiThread
Thread number 139855447648000
Thread number 139855456040704
Thread number 139855464433408
Thread number 139855333943040
Thread number 139855439255296
Thread number 139855430862592
Thread number 139855279859456
Thread number 139855422469888
Thread number 139855472826112
Thread number 139855342335744
Final counter value: 10
```

3번 문제 수행화면

```
hunterspin@ubuntu:~/Desktop/practice$ ./producerConsumer
Producer 3: Insert Item 1804289383 at 0
Producer 4: Insert Item 846930886 at 1
Producer 5: Insert Item 1681692777 at 2
Consumer 3: Remove Item 1804289383 from 0
Producer 3: Insert Item 424238335 at 3
Consumer 1: Remove Item 846930886 from 1
Consumer 2: Remove Item 1681692777 from 2
Producer 5: Insert Item 1649760492 at 4
Consumer 5: Remove Item 424238335 from 3
Consumer 4: Remove Item 1649760492 from 4
Producer 4: Insert Item 719885386 at 0
Consumer 3: Remove Item 719885386 from 0
Producer 4: Insert Item 1025202362 at 1
Consumer 1: Remove Item 1025202362 from 1
Producer 5: Insert Item 1189641421 at 2
Consumer 2: Remove Item 1189641421 from 2
Producer 5: Insert Item 783368690 at 3
Producer 3: Insert Item 596516649 at 4
Producer 4: Insert Item 1350490027 at 0
Consumer 3: Remove Item 783368690 from 3
Producer 5: Insert Item 1102520059 at 1
Consumer 1: Remove Item 596516649 from 4
Producer 3: Insert Item 2044897763 at 2
Consumer 2: Remove Item 1350490027 from 0
Producer 4: Insert Item 1967513926 at 3
Consumer 3: Remove Item 1102520059 from 1
Producer 3: Insert Item 1365180540 at 4
Producer 2: Insert Item 1714636915 at 0
Producer 1: Insert Item 1957747793 at 1
Consumer 3: Remove Item 2044897763 from 2
Producer 2: Insert Item 1540383426 at 2
Consumer 1: Remove Item 1967513926 from 3
Consumer 1: Remove Item 1365180540 from 4
Producer 2: Insert Item 1303455736 at 3
Consumer 4: Remove Item 1714636915 from 0
Consumer 2: Remove Item 1957747793 from 1
Producer 1: Insert Item 304089172 at 4
Producer 1: Insert Item 521595368 at 0
Consumer 4: Remove Item 1540383426 from 2
Consumer 4: Remove Item 1303455736 from 3
Consumer 5: Remove Item 304089172 from 4
```

운영체제 보고서

빅데이터 학과

20175119 김영식

1번 프로그램 코드

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

void *print_message_function( void *ptr );

int main()
{
	pthread_t thread1, thread2;
	char *message1 = "Thread 1";
	char *message2 = "Thread 2";
	int  iret1, iret2;

	iret1 = pthread_create( &thread1, NULL, print_message_function, (void*) message1);
	iret2 = pthread_create( &thread2, NULL, print_message_function, (void*) message2);

	pthread_join( thread1, NULL);
	pthread_join( thread2, NULL);

	printf("Thread 1 returns: %d₩n",iret1);
	printf("Thread 2 returns: %d₩n",iret2);
	exit(0);
}
```

```
void *print_message_function( void *ptr )
{
        char *message;
        message = (char *) ptr;
        printf("%s ₩n", message);
}
```

프로그램 설명

이 프로그램은 void *print_message_function(void *ptr)라는 스레드가 사용하기 위한 함수가 있고, 메인함수인 main()함수로 이루어져있다.

이 프로그램의 목적은 스레드를 두개를 생성했을 때, 두 스레드가 동시에 실행되어 처리되느냐는 것을 보기 위한 프로그램이라고 판단된다.

코드 설명

char *message1 = "Thread 1";

char *message2 = "Thread 2";

이 코드는 스레드 각각마다 보내는 메시지의 값을 다르게 처리하려고 하여 선언한 것이다.

iret1 = pthread_create(&thread1, NULL, print_message_function, (void*) message1);

iret2 = pthread_create(&thread2, NULL, print_message_function, (void*) message2);

이 코드는 스레드를 생성하는 코드로 각각의 매개변수가 뜻하는 것은

스레드 식별자는 thread1(2)

특성은 NULL이므로 기본 특성

print_message_function은 실행할 스레드 함수

(void*) message1(2)을 스레드 함수의 매개변수로 넘긴다.

pthread_join(thread1, NULL);

pthread_join(thread2, NULL);

이 함수는 매개변수로 받은 특정 스레드가 종료하기를 기다렸다가, 종료된 이후 다음을 진행하는 함수이다.

첫번째 매개변수는 함수에 전달할 스레드

두번째 매개변수는 스레드의 리턴 값으로 NULL이 아닐 경우 해당 포인터로 쓰레드의 리턴 값을 받아올 수 있다.

```
printf("Thread 1 returns: %d₩n",iret1);

printf("Thread 2 returns: %d₩n",iret2);

exit(0);
```

이 코드는 각각의 쓰레드가 종료될 경우 반환한 리턴 값을 출력해주는 것이다. 성공적으로 스레드가 생성되었다면 iret1(2)는 0을 반환한다.

exit(0)는 함수를 종료 시키는 것으로 메인문의 종료를 뜻한다.

```
void *print_message_function( void *ptr )

{

        char *message;

        message = (char *) ptr;

        printf("%s ₩n", message);

}
```

마지막으로 이 함수는 메세지를 받아서 그 메시지를 출력해주는 함수이다.

2번 프로그램 코드

```
#include <stdio.h>
#include <pthread.h>
#define NTHREADS 10

void *thread_function(void *);

pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
int   counter = 0;

int main(){
    pthread_t thread_id[NTHREADS];
    int i, j;
    for(i=0; i < NTHREADS; i++) {
        pthread_create( &thread_id[i], NULL, thread_function, NULL );
    }
    for(j=0; j < NTHREADS; j++) {
        pthread_join( thread_id[j], NULL);
    }
    printf("Final counter value: %d₩n", counter);
}
void *thread_function(void *dummyPtr){
    printf("Thread number %ld₩n", pthread_self());
    pthread_mutex_lock( &mutex1 );
    counter++;
    pthread_mutex_unlock( &mutex1 );}
```

프로그램 설명

이 프로그램은 다수의 스레드를 생성한 뒤, 전역변수인 counter를 각각의 스레드가 동시에 접근하지 못하도록 mutex를 사용하여 총 만들어진 스레드의 개수를 출력하는 프로그램이다.

코드 설명

```
void *thread_function(void *dummyPtr)
{
    printf("Thread number %ld₩n", pthread_self());
    pthread_mutex_lock( &mutex1 );
    counter++
    pthread_mutex_unlock( &mutex1 );
}
```

이 함수는 스레드가 실행시킬 함수로 먼저 pthread_self() 사용하여 현재 스레드의 식별자를 출력하고,

pthread_mutex_lock(&mutex1) 를 이용하여 임계구역의 시작을 알리고 pthread_mutex_unlock(&mutex1) 를 이용해서 임계구역의 끝을 알린다.

이 사이에 있는 코드인 counter++에서 접근하고있는 변수인 counter는 여러 스레드가 동시에 접근할 경우 치명적인 에러가 발생할 수 있기 때문에 mutex함수를 이용하여 임계구역을 설정한다.

Counter++코드로 인해 스레드가 실행될 때마다 1씩 증가하여 총 실행된 스레드의 개수를 알 수 있게 된다.

pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;

mutex변수를 초기화할 때 사용하는

PTHREAD_MUTEX_INITIALIZER를 사용하여 mutex를 default속성으로 초기화 시켜준다.

int counter = 0;

전역변수를 설정한다.

pthread_t thread_id[NTHREADS];

스레드 배열을 선언한다.(크기10)

```
for(i=0; i < NTHREADS; i++) {
        pthread_create( &thread_id[i], NULL, thread_function, NULL );
    }
for(j=0; j < NTHREADS; j++){
        pthread_join( thread_id[j], NULL);
  }
```

스레드를 배열의 개수만큼 생성하고 그 스레드들은 thread_function함수를 실행시킨다.

join함수는 매개변수로 받은 특정 스레드가 종료하기를 기다렸다가, 종료된 이후 다음을 진행하는 함수이다.

printf("Final counter value: %d₩n", counter);

마지막으로 총 실행된 스레드의 개수를 출력해준다.

3번 프로그램 코드

```
#include <pthread.h>
#include <semaphore.h
#include <stdlib.h>
#include <stdio.h>
#define MaxItems 5
#define BufferSize 5

sem_t empty;
sem_t full;
int in = 0;
int out = 0;
int buffer[BufferSize];
pthread_mutex_t mutex;
void *producer(void *pno){
      int item;
      for(int i = 0; i < MaxItems; i++) {
            item = rand(); // Produce an random item
            sem_wait(&empty);
            pthread_mutex_lock(&mutex);
            buffer[in] = item;
           usleep(500000);
            printf("Producer %d: Insert Item %d at %d₩n", *((int *)pno),buffer[in],in);
            in = (in+1)%BufferSize;
```

```
            pthread_mutex_unlock(&mutex);


            sem_post(&full);
        }
}
void *consumer(void *cno){
        for(int i = 0; i < MaxItems; i++)
            sem_wait(&full);
            pthread_mutex_lock(&mutex);
            int item = buffer[out];
           usleep(500000);
            printf("Consumer %d: Remove Item %d from %d₩n",*((int *)cno),item, out)
            out = (out+1)%BufferSize;
            pthread_mutex_unlock(&mutex);
            sem_post(&empty);
        }
}
int main(){
        pthread_t pro[5],con[5];
        pthread_mutex_init(&mutex, NULL)
        sem_init(&empty,0,BufferSize);
        sem_init(&full,0,0)
        int a[5] = {1,2,3,4,5};
        for(int i = 0; i < 5; i++) {
            pthread_create(&pro[i], NULL, (void *)producer, (void *)&a[i]);
        }
```

```
    for(int i = 0; i < 5; i++) {
        pthread_create(&con[i], NULL, (void *)consumer, (void *)&a[i]);
    }
    for(int i = 0; i < 5; i++) {
        pthread_join(pro[i], NULL);
    }
    for(int i = 0; i < 5; i++) {
        pthread_join(con[i], NULL);
    }
    pthread_mutex_destroy(&mutex)
    sem_destroy(&empty);
    sem_destroy(&full);
    return 0;
}
```

프로그램 설명

세마포어와 뮤텍스를 이용하여 생산자와 소비자가 각각 생산하고 소비하는 것을 출력하는 프로그램이다.

뮤텍스는 임계영역에 한 개의 스레드밖에 존재하지 못하지만,세마포어는 설정한 만큼 스레드의 개수가 존재 할 수 있는 특징을 사용하여 작성되어있다.

코드 설명

sem_t empty;

sem_t full;

empty - full만큼의 공간이 세마포어에 존재하는 크기이다.

int in = 0;

int out = 0;

변수로 in은 생산자가 배열에 값을 넣을 때,

Out은 소비자가 배열에 있는 값을 빼올 때 사용하는 인덱스이다.

int buffer[BufferSize];

데이터값을 저장하고 있는 배열이다.

pthread_mutex_t mutex;

뮤텍스 변수를 사용하기 위해 선언한 것이다.

```
void *producer(void *pno){
    int item;
    for(int i = 0; i < MaxItems; i++) {
        item = rand(); // Produce an random item
        sem_wait(&empty);
        pthread_mutex_lock(&mutex);
        buffer[in] = item;
          usleep(500000);
        printf("Producer %d: Insert Item %d at %d₩n", *((int
          *)pno),buffer[in],in);
        in = (in+1)%BufferSize;
        pthread_mutex_unlock(&mutex);

        sem_post(&full);
    }
}
```

sem_wait(&empty);로 인해 empty의 값이 1 감소한다.

sem_post(&full);로 인해 full의 값이 1증가한다.

생산자 함수로 스레드를 받아서 뮤텍스와 세마포어를 이용하여 배열에 랜덤한 값을 넣고 그 값을 출력해주는 함수이다.

생산자 하나당 5개의 제품을 만든다.

```
void *consumer(void *cno){
      for(int i = 0; i < MaxItems; i++)
            sem_wait(&full);
            pthread_mutex_lock(&mutex);
            int item = buffer[out];
        usleep(500000);
            printf("Consumer %d: Remove Item %d from %d₩n",*((int *)cno),item, out)
            out = (out+1)%BufferSize;
            pthread_mutex_unlock(&mutex);
            sem_post(&empty);
      }
}
```

sem_wait(&full);로 인해 full의 값이 1 감소한다.

sem_post(&empty);로 인해 empty의 값이 1증가한다.

생산자 함수로 스레드를 받아서 뮤텍스와 세마포어를 이용하여 값을 꺼내온 뒤 out의 값을 1 증가시키고(배열사이즈와 같을 경우 0으로 돌아감) 꺼내온 값을 출력하는 함수이다.

소비자 하나당 5개의 제품을 소비한다.

pthread_t pro[5],con[5];

pthread_mutex_init(&mutex, NULL)

스레드 배열을 선언하고, 뮤텍스를 사용하기위해 초기화 해준다.

sem_init(&empty,0,BufferSize);

sem_init(&full,0,0)

세마포어를 사용하기 위해 변수 두개를 선언하는데 empty는 세마포어의 최대크기를 지정하고 full에는 0을 지정한다.

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

for문을 사용하여 생산자와 소비자 스레드를 각각 5개씩 생성한다.(producer와 consumer는 각각 5회씩 돌기 때문에 총 50회 작동한다)

```
for(int i = 0; i < 5; i++) {
        pthread_join(pro[i], NULL);
    }
for(int i = 0; i < 5; i++) {
        pthread_join(con[i], NULL);
    }
```

이 함수는 매개변수로 받은 특정 스레드가 종료하기를 기다렸다가, 종료된 이후 다음을 진행하는 함수이다.

```
pthread_mutex_destroy(&mutex)
sem_destroy(&empty);
sem_destroy(&full);
```

뮤텍스,세마포어와 관련된 리소스들을 소멸시킨다.