과목명	운영체제	분반	X	담당교수	김화성 교수님
학과	전자통신공학과	학번	2016707079	이름	하상천
과제명: H/W 9 - 쓰레드 과제					

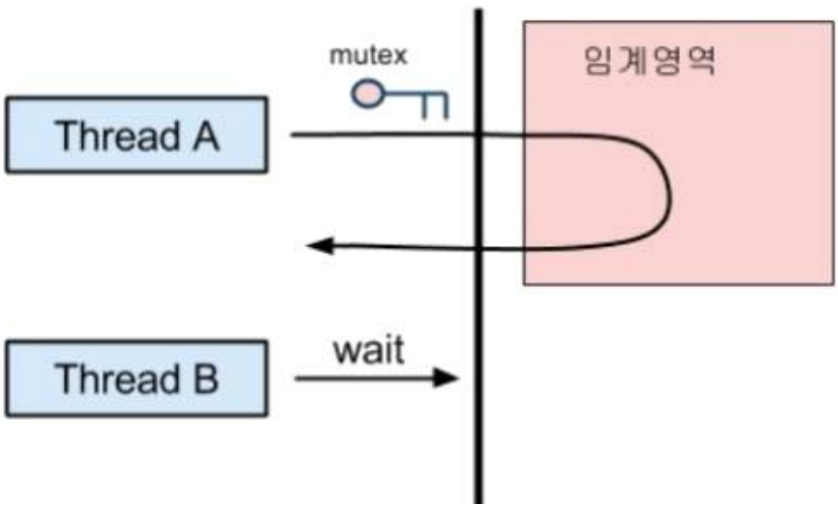
1. 과제설명 (사용자 요구사항 기술: 과제에 대한 설명 및 목표)

Factorial의 합을 병렬로 계산하는 fact 프로그램을 작성하시오.

- fact는 argument로 숫자 하나를 받고, 그 숫자까지의 factorial의 합을 구한다.
  - 예) ./fact 4
  - 1! + 2! + 3! + 4! 의 총 합을 구한다.
- pthread를 이용하여 각 thread가 각각 1!, 2!, 3!, 4!를 계산하고 출력한다.
- 계산이 모두 종료된 이후, 각 계산결과를 합산하고 출력한다.
- 각 thread가 공유된 변수에 접근할 때는 반드시 mutex를 이용하여 critical section을 보호하여야 한다.

주의사항 1
출력되는 순서는 상관없음. 단, total은 마지막에 출력
total값이 항상 일정하도록 thread 동기화 필요

2. 알고리즘 및 자료구조 설계 내용



thread가 공유된 변수에 접근할 때 mutex를 통해 thread를 동기화 시킨다.

3. 소스코드 설명 (직접 작성한 소스코드중에 핵심 부분을 발췌하여 설명)

```
pthread_mutex_t m_lock = PTHREAD_MUTEX_INITIALIZER;

int total = 0;

void *factorial(void *number) {
int factorial_cal = 1;
int *num = (int *)number
for (int i = 1; i <= *(num); i++) {
factorial_cal *= i;
}
printf("%d! : %d₩n", *num, factorial_cal);
pthread_mutex_lock(&m_lock);
total += factorial_cal;
pthread_mutex_unlock(&m_lock);
}
```

Thread 동기화에 사용할 변수 m_lock을 초기화하고, total 값을 계산할 전역변수를 선언해준다. factorial을 구하는 함수로 반복문을 통해 계산하였고, 계산한 값을 바로 출력하였다. 전역변수 total은 thread간의 공유를 하기 때문에 lock과 unlock을 통해 thread 동기화를 해주었다.

```
int main(int argc, char *argv[]) {
if (argc != 2) {
printf("Please put one argument.₩n");
return 0;
}
int num;
num = atoi(argv[1]);
```

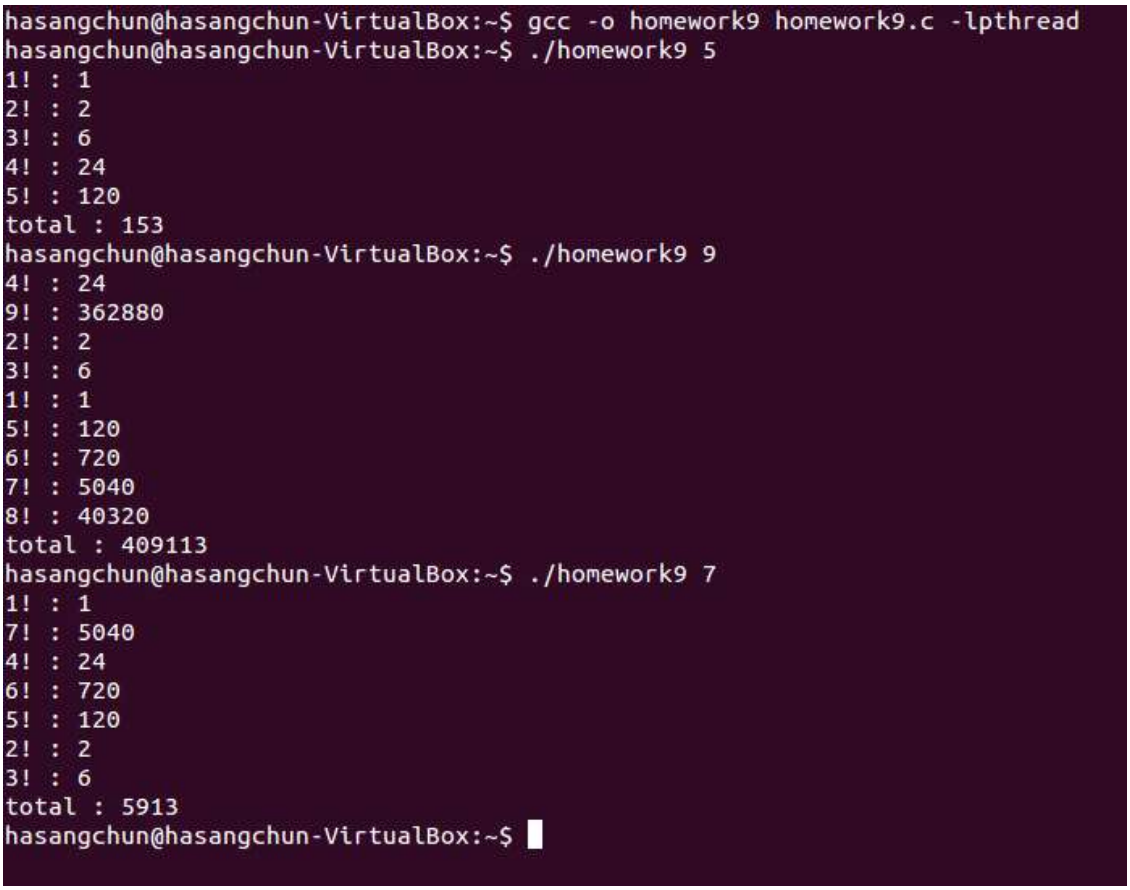
argc는 argv에 저장된 문자열의 수이고, argv[0]은 프로그램 경로가 저장되어있다. 따라서 argument로 하나의 input이 들어오면 argc는 2이기 때문에, 2가 아니라면 프로그램을 종료시켰다. 또한 num = atoi(argv[1])에서는 argument로 들어온 input을 문자열에서 정수타입으로 바꾸어주었다.

```
pthread_t thread_id[num];
int *array = (int *)malloc(num * sizeof(int));
for (int i = 0; i < num; i++) {
array[i] = i + 1;
}
for (int i = 0; i < num; i++) {
pthread_create(&thread_id[i], NULL, factorial, (void *)&array[i]);
}
for (int i = 0; i < num; i++) {
pthread_join(thread_id[i], NULL);
}
free(array);
pthread_mutex_destroy(&m_lock);

printf("total : %d₩n", total);
return 0;
}
```

num개 만큼의 동적 배열을 할당하여 1부터 num까지 값을 넣어주었다. num개 만큼 thread를 생성하여, 미리 정의해둔 factorial 함수와 동적 배열의 주소값을 파라미터로 넣어주었다. join함수를 통해 main thread에서 thread를 모두 기다리게 하여 total 값을 마지막에 출력하게 해주었고, free함수를 통해 동적 메모리를 반납하고 mutex를 파괴하였다. 그리고 모든 factorial 값이 더해진 전역변수 total 값을 출력하였다.

4. 실행결과 및 설명 (실행 결과를 캡쳐하여 첨부한 후 설명)



(그림을 문서에 포함, 글자처럼 취급 옵션, 잉크 절약과 잘 보이게 하기위해 그림 반전)

5. 고찰 (과제를 진행하면서 배운점 이나, 시행 착오 내용, 기타 느낀점)

이번 과제는 thread 동기화를 이용하여 argument로 입력받은 값까지의 factorial값을 계산하여 출력하고 모두 더한 값을 출력하는 것이었다. 객체지향프로그래밍 수업을 들을 때, 똑같은 과제를 진행해보아서 쉽게 접근해볼 수 있었다. 그때는 자바로 과제를 했기 때문에 static 변수를 통해서 total 값을 구하였다. 이번에도 언어만 c언어로 바뀔 뿐 똑같이 전역변수 total을 이용하였다. thread가 factorial 계산을 마치면 mutex lock과 unlock을 통해 thread를 동기화시키면서 전역변수 total에 값을 더하였다. pthread_join 함수를 통해 main thread가 thread를 모두 기다린 후 total 값을 출력하게 하였다. 그래서 total 값은 마지막에 정확한 값으로 나오지만, factorial을 계산하는 것은 thread를 동기화 시키지 않았기 때문에 factorial 값이 1부터 순서대로 나오는 것이 아니라 순서 없이 나오는 것을 확인할 수 있었다.

6. 전체 소스코드 (글자크기 9에 줄간격을 120%로 유지하고 한 줄이 너무 길지 않게 작성)

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

pthread_mutex_t m_lock = PTHREAD_MUTEX_INITIALIZER;

int total = 0;

void *factorial(void *number) {
```

```
int factorial_cal = 1;
int *num = (int *)number
for (int i = 1; i <= *(num); i++) {
factorial_cal *= i;
}
printf("%d! : %d₩n", *num, factorial_cal);
pthread_mutex_lock(&m_lock);
total += factorial_cal;
pthread_mutex_unlock(&m_lock);
}

int main(int argc, char *argv[]) {
if (argc != 2) {
printf("Please put one argument.₩n");
return 0;
}
int num;
num = atoi(argv[1]);

pthread_t thread_id[num];
int *array = (int *)malloc(num * sizeof(int));
for (int i = 0; i < num; i++) {
array[i] = i + 1;
}
for (int i = 0; i < num; i++) {
pthread_create(&thread_id[i], NULL, factorial, (void *)&array[i]);
}
for (int i = 0; i < num; i++) {
pthread_join(thread_id[i], NULL);
}
free(array);
pthread_mutex_destroy(&m_lock);

printf("total : %d₩n", total);
return 0;
}
```

(글자크기는 10으로 유지하고 줄간격도 160%를 유지할 것)