

## A. 과제 A

### 1. 프로그램 설명

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

#define ARGUMENT_NUMBER 20

//array for containing computational values for each thread
long long result[ARGUMENT_NUMBER];

//put the thread number 25000000times to each thread result array
void *ThreadFunc(void *n)
{
    long long i;
    long long number = *((long long *)n);
    printf("number = %lld\n",number);
    for(i=0; i<25000000;i++)
    {
        result[number]+=number;
    }
}

int main(void)
{
    pthread_t threadID[ARGUMENT_NUMBER];

    long long argument[ARGUMENT_NUMBER];
    long long i;

    for(i=0;i<ARGUMENT_NUMBER;i++)
    {
        argument[i] = i;
    }
}
```

조건에 맞게 항상 결과로 나오는 값이 똑같게 하기 위해서 각 스레드 별로 더해 주는 변수를 따로 설정해 줘야한다. 따라서 스레드가 ARGUMENT\_NUMBER 만큼 생성하므로 result도 array로 만들어서 ARGUMENT\_NUMBER만큼의 원소를 가지도록 해주었다. 그리고 global로 선언했기 때문에 모든 배열의 값은 0을 가진다. ThreadFunc 함수는 함수의 인자를 25000000더해주고 그 값을 result에 더해주는 함수이다. main문에서 pthread\_t threadID[ARGUMENT\_NUMBER]를 통해 ARGUMENT\_NUMBER만큼의 스레드를 선언해준다. argument 배열은 함수에 인자로 넣어질 배열로 for문을 통해 argument의 각각의 위치에 따라 같은 값을 가지

도록 해준다. (0번째에는 0을 갖도록, 1번째에는 1을 갖도록)

```
//20 threads create and run ThreadFunc function. function's argument is argument[i]

for(i=0;i<ARGUMENT_NUMBER;i++)
{
    pthread_create(&threadID[i],NULL,ThreadFunc,(void*)&argument[i]);
}

printf("Main Thread is waiting for Child Thread!\n");

//wait for the end of the 20 threads
for(i=0;i<ARGUMENT_NUMBER;i++)
{
    pthread_join(threadID[i],NULL);
}

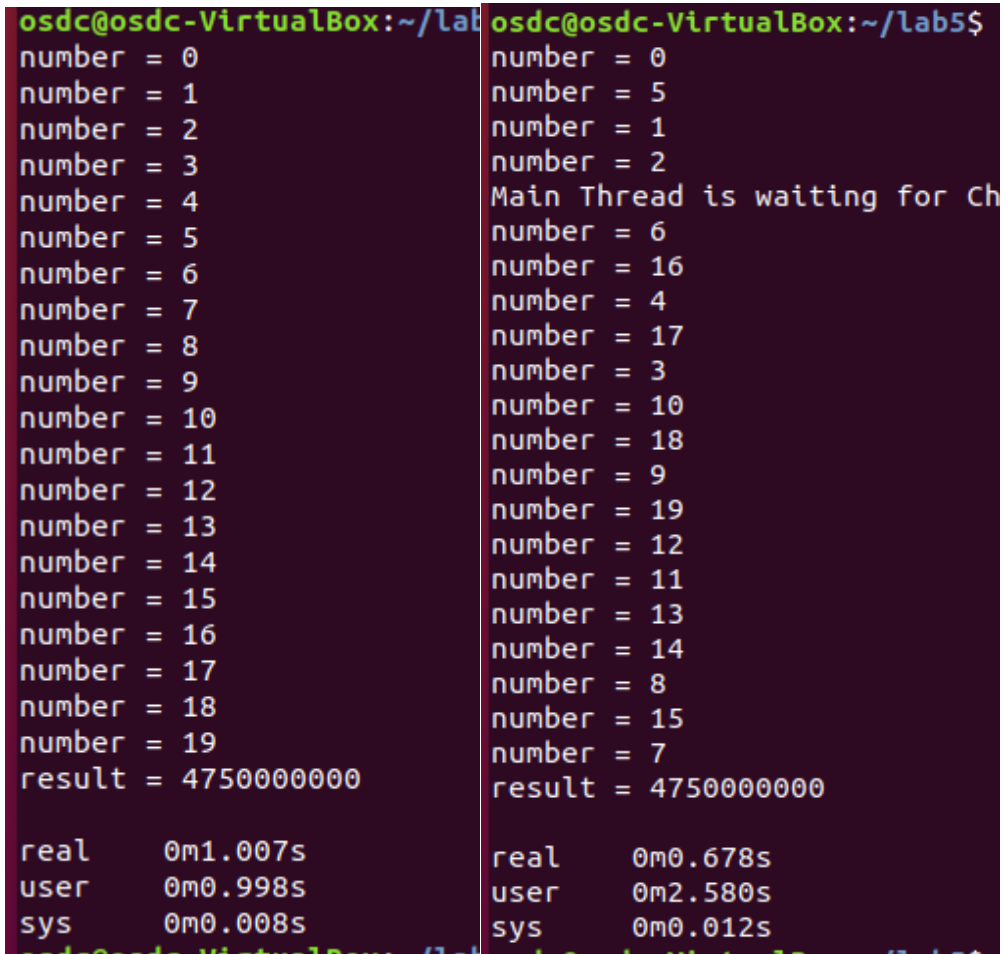
//variable to add result of each thread
long long total=0;

//add the result of each thread to the total variable
for(i=0;i<ARGUMENT_NUMBER;i++)
{
    total+=result[i];
}

printf("result = %lld\n",total);
return 0;
```

Pthread\_create함수를 통해 스레드를 생성해준다. for문을 이용해서 20개의 스레드를 생성한다. 이 함수의 첫번째 인수에 i번째의 스레드 식별자 threadID[i]를 넣어준다. 기본 스레드로 사용할 것 이기 때문에 NULL을 두번째 인자로 넣어준다. 그리고 스레드별로 분기 시켜서 실행할 함수인 ThreadFunc를 세번째 인자로 넣어준 후, ThreadFunc의 매개변수로 넘겨질 argument를 4번째 인자로 넣어준다. 스레드가 모두 생성되면 "Main thread is waiting for Child Thread!"를 출력시킨다. 그다음 pthread\_join함수를 통해서 스레드의 종료를 기다린다. 첫번째 인자인 스레드 식별자를 통해 해당 스레드의 종료를 기다리고 return value로 null을 넣어준다. 종료가 되었다면, 각 스레드가 연산한 result값을 모두 더할 변수인 total 변수를 선언해준다. 그 다음 for 문을 통해서 각 result의 값을 더해준 뒤, printf를 통해서 total 값을 출력한다.

## 2. Time 결과값 설명



```
osdc@osdc-VirtualBox:~/lab5$ ./lab5_1
number = 0
number = 1
number = 2
number = 3
number = 4
number = 5
number = 6
number = 7
number = 8
number = 9
number = 10
number = 11
number = 12
number = 13
number = 14
number = 15
number = 16
number = 17
number = 18
number = 19
result = 4750000000

real    0m1.007s
user    0m0.998s
sys     0m0.008s

osdc@osdc-VirtualBox:~/lab5$ ./lab5_2
number = 0
number = 5
number = 1
number = 2
Main Thread is waiting for Ch
number = 6
number = 16
number = 4
number = 17
number = 3
number = 10
number = 18
number = 9
number = 19
number = 12
number = 11
number = 13
number = 14
number = 8
number = 15
number = 7
result = 4750000000

real    0m0.678s
user    0m2.580s
sys     0m0.012s
```

Real time은 0.6~0.8s user time은 2.5~2.9s, sys time은 0.00~0.01이다. Real time은 프로그램의 시작부터 종료까지의 시간이다. 즉 이 프로그램은 시작부터 끝날 때까지 약 0.6~0.8s의 시간이 소요된다. User time은 user mode에서 소요된 CPU 시간이다. CPU는 이 프로그램을 실행하는데 user mode에서 약 2.5~2.9 시간동안 일을 했다. Sys는 프로세스 내에서 커널에 소요된 CPU 시간이다. 이는 커널 내의 시스템 호출에 소요된 CPU 시간을 말한다. 이 프로그램에서 CPU는 시스템 콜에 의해 kernel mode에서 0.00~0.01시간동안 일을 했다.

## 3. 시간 차이가 나는 이유 설명

사용자 관점에서는 완료하는 시간은 real time은 멀티 스레드를 사용했을 때 줄어든 것을 확인할 수 있다. 시간이 준 이유는 한 프로세스를 여러 스레드로 나누

어 병렬 처리했기 때문에, 속도가 상승한 것이다. 자세하게 설명하자면 250000000 씩 더하는 연산이 과제 코드에서는 20번 동안 겹침없이 진행되었지만, 스레드를 사용해서는 250000000씩 더하는 연산이 병렬적으로 진행되어 속도가 상승하였다. 따라서 프로그램의 시작부터 종료시까지의 시간은 줄어든다. 반면, User time은 늘어났는데 User mode의 코드는 여러 개의 스레드에 의해 실행되었고 user time은 모든 스레드 동안의 CPU 소요 시간이 합산되어 나오기 때문에 user time이 커진 것이다. 또한, 이러한 영향으로 과제 코드의 real time과 user time은 거의 비슷한 반면, thread를 이용한 코드는 real time과 user time이 많이 차이 나는 것을 알 수 있다. Sys 에는 별 차이가 없는데 이는 두 코드의 시스템 콜에 의한 kernel mode에서의 실행시간 차이가 적다는 것을 알 수 있다.

## B. 과제 B

### 1. Mutex 예비레포트

mutex란 Critical Section을 가진 thread들의 running time이 서로 겹치지 않게 각각 단독으로 실행되게 하는 기술이다. Critical Section은 프로그램 상에서 동시에 실행될 경우 문제가 일어날 수 있어 thread의 동시 접근을 막아야 하는 부분이다. 따라서, 만약 어느 thread에서 Critical Section을 실행하고 있으면 다른 thread들은 그 Critical Section에 접근할 수 없고 해당 thread가 Critical Section을 벗어나기를 기다려야 한다. 이를 가능하게 하는 것이 mutex이다. Mutex는 다수의 스레드(혹은 프로세스)들의 공유 리소스에 대한 접근을 locking과 unlocking을 통해 조율한다. Mutex는 key에 해당하는 어떤 object가 있고 이 object를 소유한 스레드만이 공유 리소스에 접근할 수 있다. Mutex는 한 번에 하나의 스레드만이 실행할 수 있도록 한다. 즉, mutex는 한 스레드, 프로세스에 의해 소유될 수 있는 key를 기반으로 상호배제기법이라고 할 수 있다. Mutex는 0과 1의 값을 가지는데 해당 자원을 사용 중이면 1, 미사용 중이면 0으로 표시한다. 프로그래밍할 때 mutex는 한 프로세스의 내부에서 여러 스레드의 Critical Section 제어를 위해 사용하는 객체를 뜻한다. 특별히 다른 프로세스와 공유해 사용하도록 옵션을 가하지 않는 이상 프로세스 외부에는 공개되지 않는다.