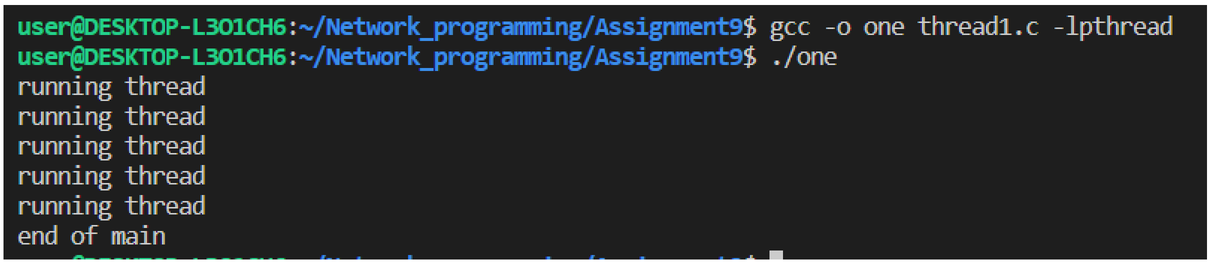
Assignment9

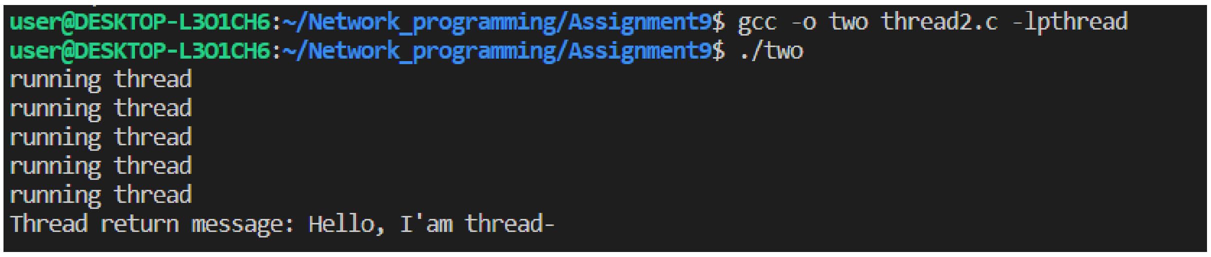
2016116783 김성록

Example 1



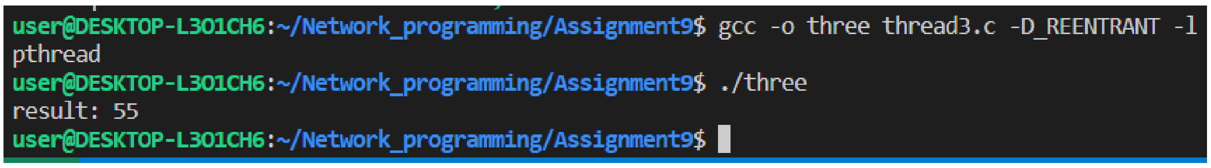
```
user@DESKTOP-L3O1CH6:~/Network_programming/Assignment9$ gcc -o one thread1.c -lpthread
user@DESKTOP-L3O1CH6:~/Network_programming/Assignment9$ ./one
running thread
running thread
running thread
running thread
running thread
end of main
```

Example 2



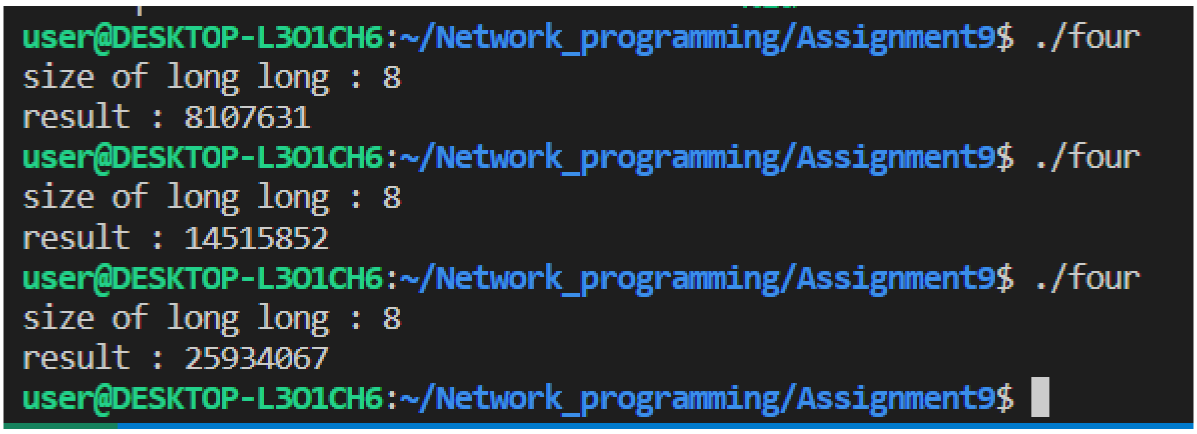
```
user@DESKTOP-L3O1CH6:~/Network_programming/Assignment9$ gcc -o two thread2.c -lpthread
user@DESKTOP-L3O1CH6:~/Network_programming/Assignment9$ ./two
running thread
running thread
running thread
running thread
running thread
Thread return message: Hello, I'am thread-
```

Example 3



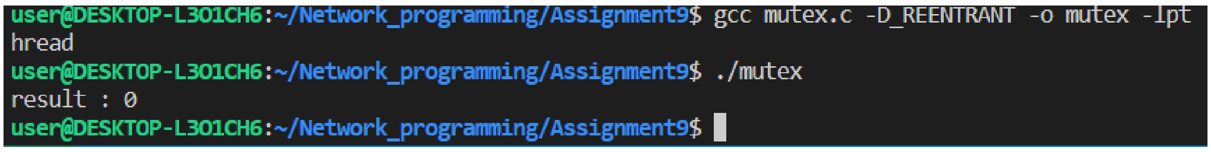
```
user@DESKTOP-L3O1CH6:~/Network_programming/Assignment9$ gcc -o three thread3.c -D_REENTRANT -l
pthread
user@DESKTOP-L3O1CH6:~/Network_programming/Assignment9$ ./three
result: 55
user@DESKTOP-L3O1CH6:~/Network_programming/Assignment9$
```

Example 4



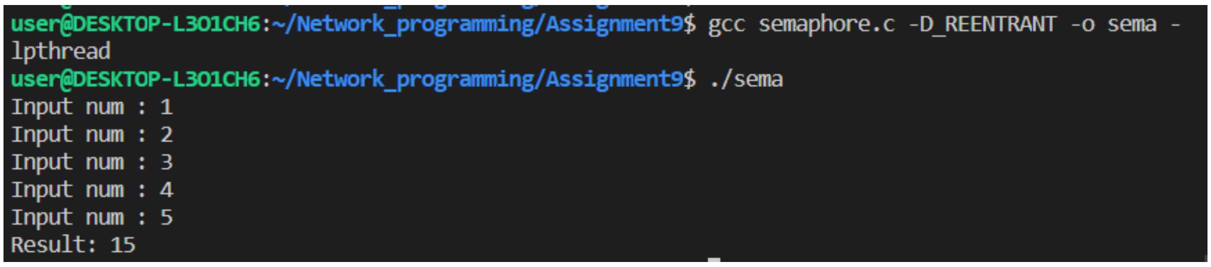
```
user@DESKTOP-L301CH6:~/Network_programming/Assignment9$ ./four
size of long long : 8
result : 8107631
user@DESKTOP-L301CH6:~/Network_programming/Assignment9$ ./four
size of long long : 8
result : 14515852
user@DESKTOP-L301CH6:~/Network_programming/Assignment9$ ./four
size of long long : 8
result : 25934067
user@DESKTOP-L301CH6:~/Network_programming/Assignment9$
```

Example 5



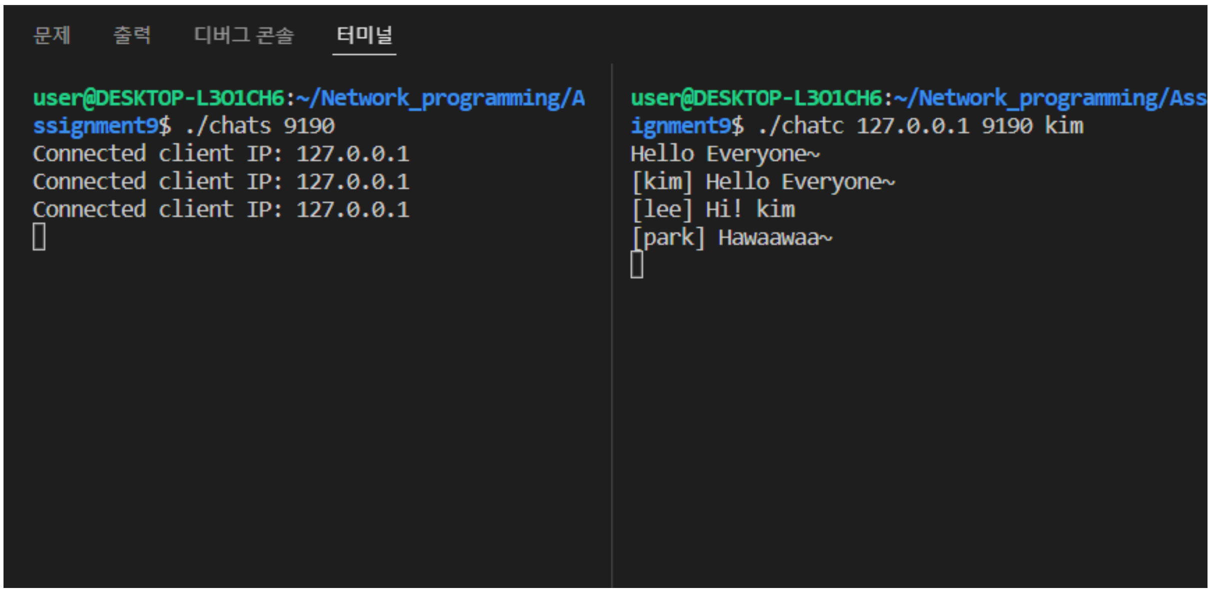
```
user@DESKTOP-L301CH6:~/Network_programming/Assignment9$ gcc mutex.c -D_REENTRANT -o mutex -lpt
hread
user@DESKTOP-L301CH6:~/Network_programming/Assignment9$ ./mutex
result : 0
user@DESKTOP-L301CH6:~/Network_programming/Assignment9$
```

Example 6



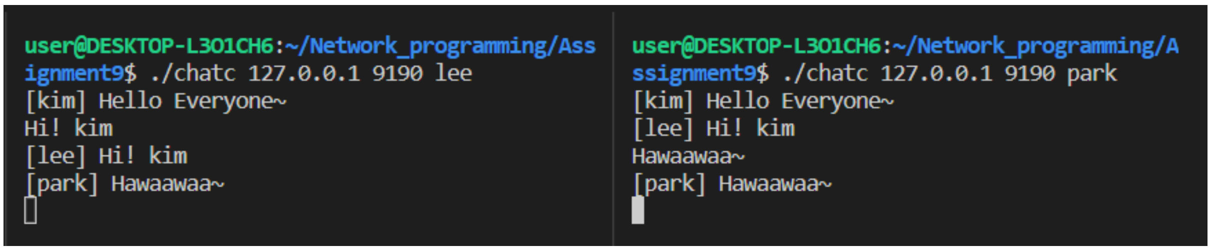
```
user@DESKTOP-L301CH6:~/Network_programming/Assignment9$ gcc semaphore.c -D_REENTRANT -o sema -
lpthread
user@DESKTOP-L301CH6:~/Network_programming/Assignment9$ ./sema
Input num : 1
Input num : 2
Input num : 3
Input num : 4
Input num : 5
Result: 15
```

Example 7



```
user@DESKTOP-L301CH6:~/Network_programming/Assignment9$ ./chats 9190
Connected client IP: 127.0.0.1
Connected client IP: 127.0.0.1
Connected client IP: 127.0.0.1
```

```
user@DESKTOP-L301CH6:~/Network_programming/Assignment9$ ./chatc 127.0.0.1 9190 kim
Hello Everyone~
[kim] Hello Everyone~
[lee] Hi! kim
[park] Hawaawaa~
```



```
user@DESKTOP-L301CH6:~/Network_programming/Assignment9$ ./chatc 127.0.0.1 9190 lee
[kim] Hello Everyone~
Hi! kim
[lee] Hi! kim
[park] Hawaawaa~
```

```
user@DESKTOP-L301CH6:~/Network_programming/Assignment9$ ./chatc 127.0.0.1 9190 park
[kim] Hello Everyone~
[lee] Hi! kim
Hawaawaa~
[park] Hawaawaa~
```

Program

objective

trying parallel merge sorting using multi-threading to reduce sorting time.

motivation

I think divide-and-conquer problem would be fit with parallel programming and Merge short is basic algorithm for divide-and-conquer problem, so I selec this one.

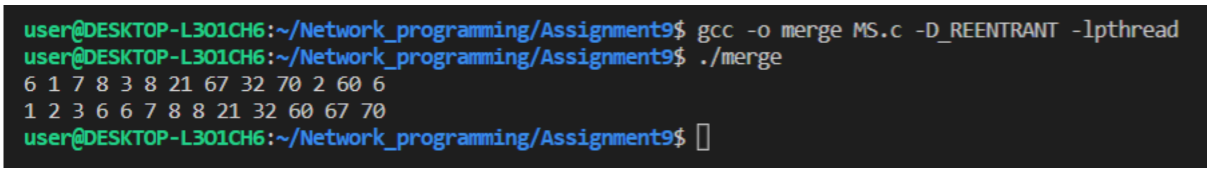
solution

Divide nums into half, allocate threads the half each. And when the threads' works done, merge once again and return the sorted result. The most difficult part was send and receive parameter, I solved with struct. I use **common buffer** and when a thread try to access the buffer, it blocks with mutex when the other is use it.

results

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

#define BUF_SIZE 50

void merge(int nums[], int l, int m, int r);
void *mergeSort(void *arg);

struct threadArgs{int* nums; int l; int r;};

int buffer[BUF_SIZE];
pthread_mutex_t mutex;

int main(){
    pthread_t thread_id[2];
    int nums[] = {6,1,7,8,3,8,21,67,32,70,2,60,6};
    int num_size = sizeof(nums)/sizeof(int);
    int i;
    int l=0;
    int r=num_size-1;
    int m = (l+r)/2;

    struct threadArgs *targs1,*targs2;
    targs1 = (struct threadArgs *)malloc(sizeof(struct threadArgs));
    targs2 = (struct threadArgs *)malloc(sizeof(struct threadArgs));

    targs1->nums=nums;
    targs1->l=l;
    targs1->r=m;
    targs2->nums=nums;
    targs2->l=m+1;
    targs2->r=r;

    for(i=0;i<num_size;i++){
        printf("%d ",nums[i]);
    }
    printf("\n");

    pthread_mutex_init(&mutex,NULL);

    pthread_create(&(thread_id[0]), NULL, mergeSort, (void *)targs1);
    pthread_create(&(thread_id[1]), NULL, mergeSort, (void *)targs2);
    pthread_join(thread_id[0],NULL);
    pthread_join(thread_id[1],NULL);

    merge(nums, l, m, r);

    for(i=0;i<num_size;i++){
        printf("%d ",nums[i]);
    }
    printf("\n");
```

```c
    pthread_mutex_destroy(&mutex);
    free(targs1);
    free(targs2);
    return 0;
}

void *mergeSort(void *arg){
    struct threadArgs * targs = (struct threadArgs *)arg;
    struct threadArgs * temp1 = (struct threadArgs *)malloc(sizeof(struct threadArgs));
    struct threadArgs * temp2 = (struct threadArgs *)malloc(sizeof(struct threadArgs));
    int *nums = targs->nums;
    int l=targs->l;
    int r=targs->r;
    int m = (l+r)/2;
    if (l<r){
        temp1->nums=nums;
        temp1->l=l;
        temp1->r=m;
        temp2->nums=nums;
        temp2->l=m+1;
        temp2->r=r;

        mergeSort((void*)temp1);
        mergeSort((void*)temp2);
        merge(nums,l,m,r);
    }
    free(temp1);
    free(temp2);
    return NULL;
}

void merge(int nums[], int l, int m, int r){
    int i,j,k;
    i=l; j=m+1; k=0;
    pthread_mutex_lock(&mutex);
    while(i<=m && j<=r){
        if(nums[i]<=nums[j])
          buffer[k++] = nums[i++];
        else
          buffer[k++] = nums[j++];
    }

    while(i<=m){
        buffer[k++]=nums[i++];
    }
    while(j<=r){
        buffer[k++]=nums[j++];
    }

    for(i=l,k=0;i<=r;i++,k++){
        nums[i]=buffer[k];
    }
    pthread_mutex_unlock(&mutex);
}
```



```
user@DESKTOP-L3O1CH6:~/Network_programming/Assignment9$ gcc -o merge MS.c -D_REENTRANT -lpthread
user@DESKTOP-L3O1CH6:~/Network_programming/Assignment9$ ./merge
6 1 7 8 3 8 21 67 32 70 2 60 6
1 2 3 6 6 7 8 8 21 32 60 67 70
user@DESKTOP-L3O1CH6:~/Network_programming/Assignment9$ 
```