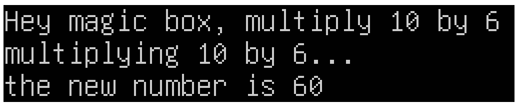
thread_01.c

</1>	create
</2>	&tid, NULL, magic_box, (void*)(intptr_t)10
</3.1>	join
</3.2>	tid, (void**)&new_number

결과 스크린샷:

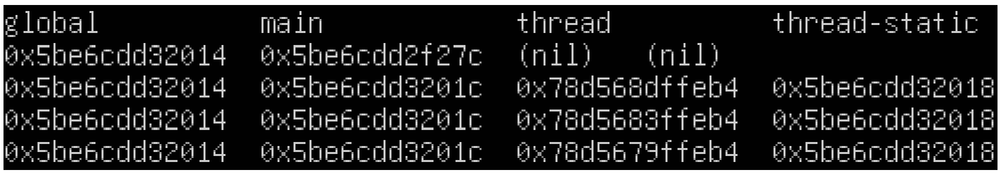


```
Hey magic box, multiply 10 by 6
multiplying 10 by 6...
the new number is 60
```

thread_02.c

</1.1>	exit
</1.2>	NULL
</2>	create
</3>	&tids[i], NULL, worker, &main_static
</4.1>	join
</4.2>	tids[i], NULL

결과 스크린샷:

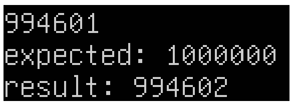


```
global          main            thread          thread-static
0x5be6cdd32014  0x5be6cdd2f27c  (nil)   (nil)
0x5be6cdd32014  0x5be6cdd3201c  0x78d568dffeb4  0x5be6cdd32018
0x5be6cdd32014  0x5be6cdd3201c  0x78d5683ffeb4  0x5be6cdd32018
0x5be6cdd32014  0x5be6cdd3201c  0x78d5679ffeb4  0x5be6cdd32018
```

thread_03.c

</1>	create
</2>	&tids[i], NULL, worker, NULL
</3.1>	join
</3.2>	tids[i], (void**)&progress
</4.1>	exit
</4.2>	(void*)(intptr_t)progress

결과 스크린샷:

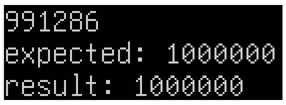


```
994601
expected: 1000000
result: 994602
```

thread_04.c

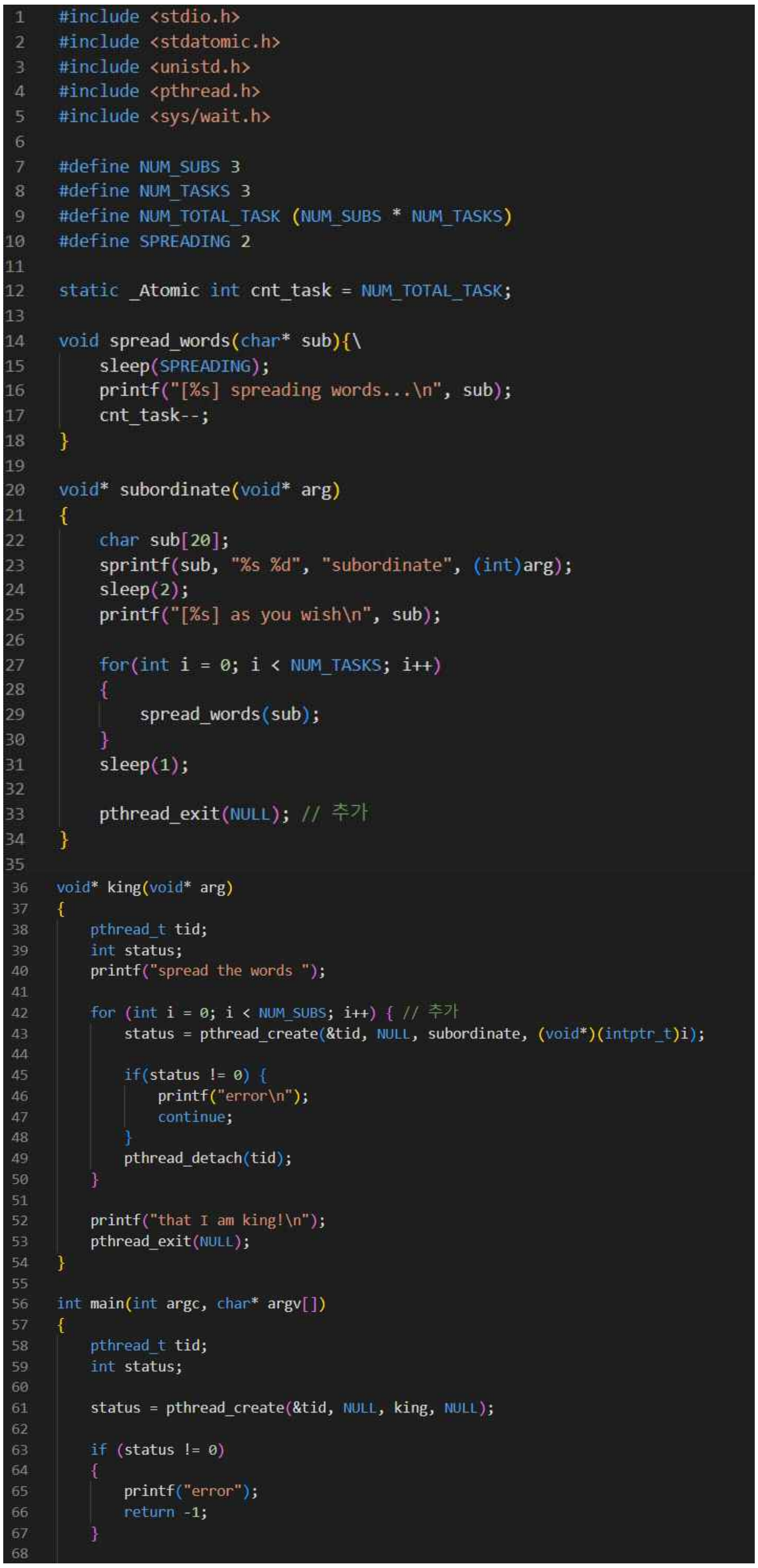
</1>	create
</2>	&tids[i], NULL, worker, NULL
</3.1>	join
</3.2>	tids[i], (void**)&progress
</4.1>	mutex_lock
</4.2>	&lock
</5.1>	mutex_unlock
</5.2>	lock
</7.1>	exit
</7.2>	(void*)(intptr_t)progress

결과 스크린샷:

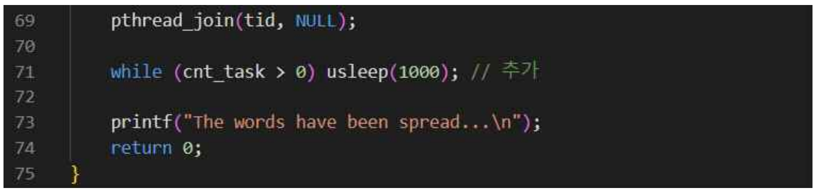


```
991286
expected: 1000000
result: 1000000
```

thread_05.c

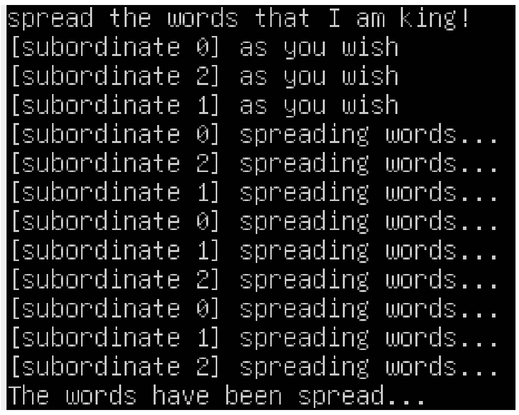


```c
#include <stdio.h>
#include <stdatomic.h>
#include <unistd.h>
#include <pthread.h>
#include <sys/wait.h>

#define NUM_SUBS 3
#define NUM_TASKS 3
#define NUM_TOTAL_TASK (NUM_SUBS * NUM_TASKS)
#define SPREADING 2

static _Atomic int cnt_task = NUM_TOTAL_TASK;

void spread_words(char* sub){\
    sleep(SPREADING);
    printf("[%s] spreading words...\n", sub);
    cnt_task--;
}

void* subordinate(void* arg)
{
    char sub[20];
    sprintf(sub, "%s %d", "subordinate", (int)arg);
    sleep(2);
    printf("[%s] as you wish\n", sub);

    for(int i = 0; i < NUM_TASKS; i++)
    {
        spread_words(sub);
    }
    sleep(1);

    pthread_exit(NULL); // 추가
}

void* king(void* arg)
{
    pthread_t tid;
    int status;
    printf("spread the words ");

    for (int i = 0; i < NUM_SUBS; i++) { // 추가
        status = pthread_create(&tid, NULL, subordinate, (void*)(intptr_t)i);

        if(status != 0) {
            printf("error\n");
            continue;
        }
        pthread_detach(tid);
    }

    printf("that I am king!\n");
    pthread_exit(NULL);
}

int main(int argc, char* argv[])
{
    pthread_t tid;
    int status;

    status = pthread_create(&tid, NULL, king, NULL);

    if (status != 0)
    {
        printf("error");
        return -1;
    }

```



```
69      pthread_join(tid, NULL);
70
71      while (cnt_task > 0) usleep(1000); // 추가
72
73      printf("The words have been spread...\n");
74      return 0;
75  }
```

결과 스크린샷:



```
spread the words that I am king!
[subordinate 0] as you wish
[subordinate 2] as you wish
[subordinate 1] as you wish
[subordinate 0] spreading words...
[subordinate 2] spreading words...
[subordinate 1] spreading words...
[subordinate 0] spreading words...
[subordinate 1] spreading words...
[subordinate 2] spreading words...
[subordinate 0] spreading words...
[subordinate 1] spreading words...
[subordinate 2] spreading words...
The words have been spread...
```

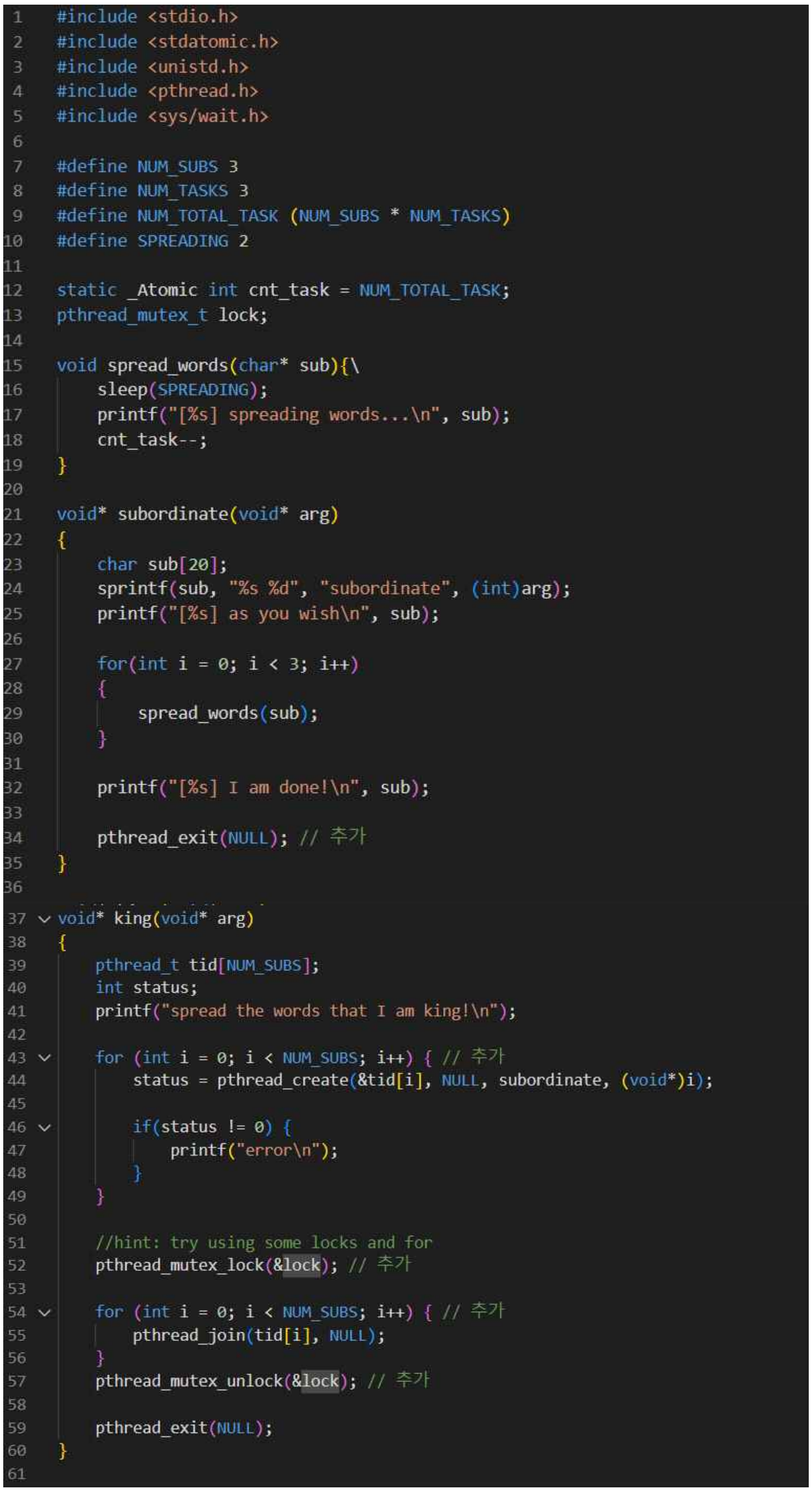
설명:
line 33: pthread_exit()을 통해 subordinate thread를 종료시켜준다.
line 42~50: for문을 사용해 NUM_SUBS만큼 subordinate thread를 생성한다.
생성된 각 thread를 detach시켜주어 종료 후 자동으로 clean up되도록 한다.
line 71: cnt_task가 양수인동안 종료되지 않도록 while문을 사용해 기다린다.

=> 전체 코드는 1개의 king thread가 subordinate thread들을 생성하고, 각각의 subordinate thread가 spread_word함수를 이용해 word를 spreading하는 구조이다.

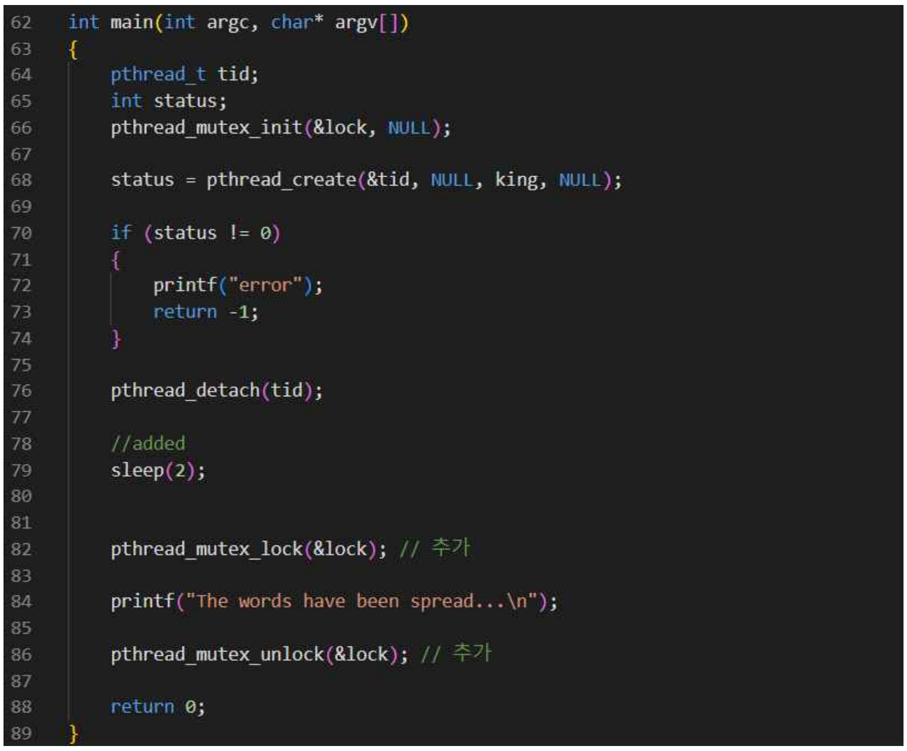
=> main thread는 king thread가 끝날때까지 join을 이용해 기다리고,
subordinate thread들은 종료 시 자동으로 cleanup 되도록 detach처리 해준다.

=> subordinate thread들이 spread_word함수를 모두 실행하기 전까지
main thread가 종료되지 않도록 while문을 활용해 기다린다.

thread_06.c

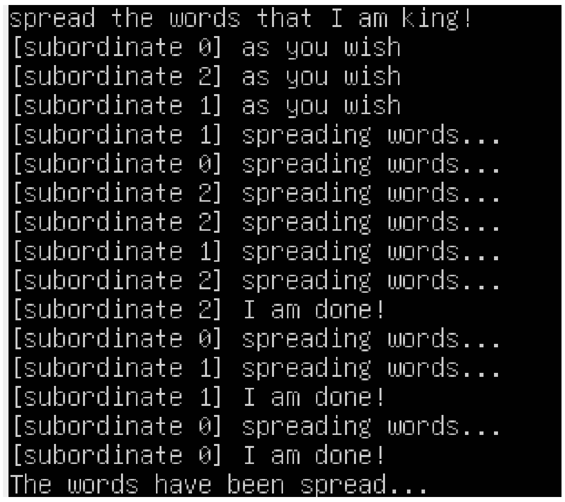


```c
#include <stdio.h>
#include <stdatomic.h>
#include <unistd.h>
#include <pthread.h>
#include <sys/wait.h>

#define NUM_SUBS 3
#define NUM_TASKS 3
#define NUM_TOTAL_TASK (NUM_SUBS * NUM_TASKS)
#define SPREADING 2

static _Atomic int cnt_task = NUM_TOTAL_TASK;
pthread_mutex_t lock;

void spread_words(char* sub){\
    sleep(SPREADING);
    printf("[%s] spreading words...\n", sub);
    cnt_task--;
}

void* subordinate(void* arg)
{
    char sub[20];
    sprintf(sub, "%s %d", "subordinate", (int)arg);
    printf("[%s] as you wish\n", sub);

    for(int i = 0; i < 3; i++)
    {
        spread_words(sub);
    }

    printf("[%s] I am done!\n", sub);

    pthread_exit(NULL); // 추가
}

void* king(void* arg)
{
    pthread_t tid[NUM_SUBS];
    int status;
    printf("spread the words that I am king!\n");

    for (int i = 0; i < NUM_SUBS; i++) { // 추가
        status = pthread_create(&tid[i], NULL, subordinate, (void*)i);

        if(status != 0) {
            printf("error\n");
        }
    }

    //hint: try using some locks and for
    pthread_mutex_lock(&lock); // 추가

    for (int i = 0; i < NUM_SUBS; i++) { // 추가
        pthread_join(tid[i], NULL);
    }
    pthread_mutex_unlock(&lock); // 추가

    pthread_exit(NULL);
}

```



```c
62  int main(int argc, char* argv[])
63  {
64      pthread_t tid;
65      int status;
66      pthread_mutex_init(&lock, NULL);
67
68      status = pthread_create(&tid, NULL, king, NULL);
69
70      if (status != 0)
71      {
72          printf("error");
73          return -1;
74      }
75
76      pthread_detach(tid);
77
78      //added
79      sleep(2);
80
81
82      pthread_mutex_lock(&lock); // 추가
83
84      printf("The words have been spread...\n");
85
86      pthread_mutex_unlock(&lock); // 추가
87
88      return 0;
89  }
```

결과 스크린샷:



```
spread the words that I am king!
[subordinate 0] as you wish
[subordinate 2] as you wish
[subordinate 1] as you wish
[subordinate 1] spreading words...
[subordinate 0] spreading words...
[subordinate 2] spreading words...
[subordinate 2] spreading words...
[subordinate 1] spreading words...
[subordinate 2] spreading words...
[subordinate 2] I am done!
[subordinate 0] spreading words...
[subordinate 1] spreading words...
[subordinate 1] I am done!
[subordinate 0] spreading words...
[subordinate 0] I am done!
The words have been spread...
```

설명:

line 34: pthread_exit()을 통해 subordinate thread를 종료시켜준다.

line 43~49: for문을 사용해 NUM_SUBS만큼 subordinate thread를 생성한다.

line 52~57: 생성된 subordinate thread들이 끝날때까지 join을 이용해 기다린다.
이때 join으로 기다리는 영역을 mutex lock으로 보호한다.

line 82, 86: mutex_lock으로 잠근다. line 79에서 sleep을 사용해 시간이 지연되는 동안 king thread에서 먼저 lock을 획득하므로, king이 모든 subordinate thread의 종료를 기다릴 때까지 lock이 해제되지 않아 main thread가 대기하게 된다.