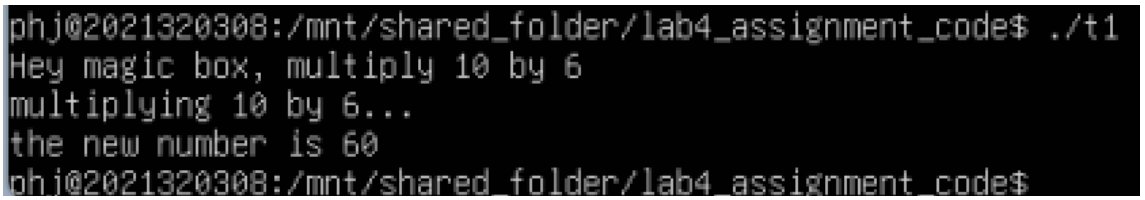
OS 실습 과제 4

thread_01.c

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

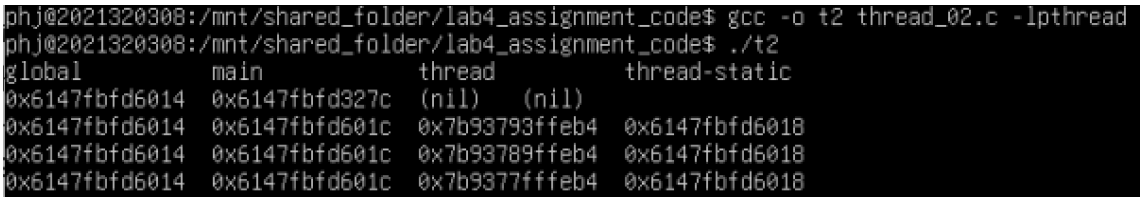
결과 스크린샷:



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:



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>	(int)progress

결과 스크린샷:

```
phj@2021320308:/mnt/shared_folder/lab4_assignment_code$ ./t3
970875
expected: 1000000
result: 980876
```

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>	(int)progress

결과 스크린샷:

```
phj@2021320308:/mnt/shared_folder/lab4_assignment_code$ ./t4
999999
expected: 1000000
result: 1000000
```

thread_05.c

thread_05.c > spread_words(char *)

```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 < 3; i++)
    {
        spread_words(sub);
    }
    sleep(1);
    pthread_exit(NULL);
}
```

thread_05.c > ...

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

    for(int i = 0; i < NUM_SUBS; i++){
        status = pthread_create(&tid, NULL, subordinate, (void*)i);
        if (status != 0){
            printf("error");
            return NULL;
        }
        pthread_detach(tid);
    }

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

thread_05.c > ...

```c
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;
    }

    pthread_join(tid, NULL);

    while(cnt_task > 0) sleep(1);

    printf("The words have been spread...\n");
    return 0;
}
```

설명:

- **line 1~12**: 실행에 필요한 헤더파일, 매크로, 전역 변수들의 정의.

  `NUM_SUBS` : 부하의 수. **subordinate**을 몇 번 수행할 것인지를 정한다.

  `NUM_TASKS` : 각 부하가 **spread_words**를 호출하는 횟수를 정한다(본 코드에는 3으로 명시하고 있으나 이 매크로를 써도 의미상 같다).

  `NUM_TOTAL_TASK` : 각 부하가 수행한 작업 수를 다 더한다. 즉, [부하 수] x [각 부하의 작업 수]

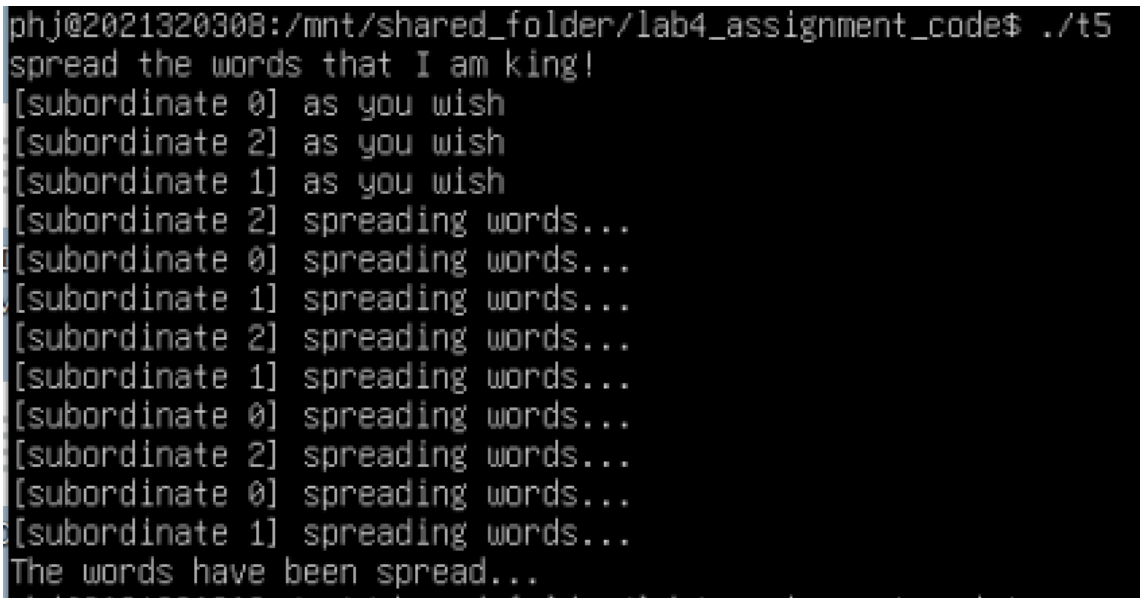
  `SPREADING` : sleep 시간용 매크로.

  `cnt_task` : 원자적 변수로 선언되어 자동으로 동기화된다.

- **line 14~18**: 각 부하가 수행하는 함수로, **"[subordinate i] spreadig words..."**를 출력하고 원자변수 cnt_task를 1 낮춘다. (한 부하당 최대 3번 낮출 것이다)
- **line 22~25**: sub라는 char 배열에 **"subordinate"** + **" "** + 매개변수 arg에 해당하는 값 **"i"** 를 저장하고, 시작 메시지 **"[subordinate i] as you wish"**를 출력한다.
- **line 27~30**: 부하가 spread_words를 수행한다. 세 번 수행된다.
- **line 32**: 부하 스레드가 pthread_exit을 호출하며 종료된다. 값을 넘겨줄 필요 없으므로 NULL을 매개변수로 한다.
- **line 37**: 부하 스레드의 스레드 id가 저장되는 변수. 곧바로 detach되기 때문에 배열이 아니라 한 개의 변수로 선언되었다.

- **line 42~47**: king이, subodinate의 매개변수 arg로 i를 갖는 부하 스레드를 생성한다. 생성된 부하 스레드 각각은 detach되어, join 없이 리소스가 회수된다.
- **line 51**: king 역시, 반환값 없이 NULL로 exit 한다.
- **line 59**: 메인 함수에서 king 스레드를 생성한다. king 함수 내에서 arg를 사용하고 있지 않으므로 매개인자를 넘겨줄 필요가 없다. 따라서 4번째 매개변수로 NULL을 넘겨준다.
- **line 61~65**: create이 실패하여 0이 아닌 값이 status에 저장되면 에러처리.
- **line 67**: king 스레드는 detach되어 있지 않으므로 join이 필요하다. 그런데 return 값이 없으므로, 매개인자로 NULL을 넣어준다.
- **line 69**: 모든 부하들이 작업을 끝내어 전역 원자 변수 cnt_task가 0이될 때까지 메인이 끝나면 안되므로 대기한다.
- **line 71~72**: cnt_task가 되면 대기에서 벗어나 메인 함수까지 종료.

결과 스크린샷:



thread_06.c

C thread_06.c > main(int, char * [])

```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);
}
```

C thread_06.c > main(int, char * [])

```c

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++){
        //start workers
        status = pthread_create(&tid[i], NULL, subordinate, (void*)i);
    }
    pthread_mutex_lock(&lock);
    for(int i=0; i < NUM_SUBS; i++){
        pthread_join(tid[i], NULL);
    }
    pthread_mutex_unlock(&lock);

    pthread_exit(NULL);
}

```

```c
C thread_06.c > king(void *)
int main(int argc, char* argv[])
{
    pthread_t tid;
    int status;
    pthread_mutex_init(&lock, NULL);

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

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

    pthread_detach(tid);

    //added
    sleep(2);

    pthread_mutex_lock(&lock);
    pthread_mutex_unlock(&lock);
    pthread_mutex_destroy(&lock);

    printf("The words have been spread...\n");\

    return 0;
}
```

설명:

- **line 1~13**: 실행에 필요한 헤더파일, 매크로, 전역 변수들의 정의.

  `NUM_SUBS` : 부하의 수. **subordinate**을 몇 번 수행할 것인지를 정한다.

  `NUM_TASKS` : 각 부하가 **spread_words**를 호출하는 횟수를 정한다(본 코드에는 3으로 명시하고 있으나 이 매크로를 써도 의미상 같다).

  `NUM_TOTAL_TASK` : 각 부하가 수행한 작업 수를 다 더한다. 즉, [부하 수] x [각 부하의 작업 수]

  `SPREADING` : sleep 시간용 매크로.

  `cnt_task` : 원자적 변수로 선언되어 자동으로 동기화된다.

  `lock` : 전역 락. main 및 생성된 스레드 모두가 공유한다. 상호배제(mutual-exclusive)를 보장한다.

- **line 15~19**: 각 부하가 수행하는 함수로, **"[subordinate i] spreadig words..."**를 출력하고 원자변수 cnt_task를 1 낮춘다. (한 부하당 최대 3번 낮출 것이다)
- **line 23~25**: sub라는 char 배열에 **"subordinate"** + **" "** + 매개변수 arg에 해당하는 값 **"i"** 를 저장하고, 시작 메시지 **"[subordinate i] as you wish"**를 출력한다.
- **line 27~31**: 부하가 spread_words를 수행한다. 세 번 수행된다. 세 번의 spread_words 수행 후 완료를 나타내기 위해 **"I am done!"**을 출력한다.
- **line 33**: 부하 스레드가 pthread_exit을 호출하며 종료된다. 값을 넘겨줄 필요 없으므로 NULL을 매개변수로 한다.
- **line 38**: 부하 스레드의 스레드 id가 저장되는 변수. thread_05.c와 달리 detach되지 않기 때문에 배열로 선언하여 덮어쓰여지지 않도록 한다.
- **line 42~45**: king이, subodinate의 매개변수 arg로 i를 갖는 부하 스레드를 생성한다.
- **line 46**: 전역 lock을 획득하여 king 스레드에서 join이 완료되기까지 다른 스레드(main 포함)들은 lock을 얻지 못한다.

- **line 47~49**: join으로 부하 스레드 각각의 exit을 받아 리소스를 회수한다. exit의 반환값이 없으므로 매개인자로 NULL을 넣는다.
- **line 50**: join을 모두 완료했으니, 전역 lock을 해제한다.
- **line 52**: king 역시, 반환값 없이 NULL로 exit 한다.
- **line 59**: 뮤텍스를 초기화한다. 전역 변수 lock을 사용한다.
- **line 61**: 메인 함수에서 king 스레드를 생성한다. king 함수 내에서 arg를 사용하고 있지 않으므로 매개인자를 넘겨줄 필요가 없다. 따라서 4번째 매개변수로 NULL을 넘겨준다.
- **line 63~67**: create이 실패하여 0이 아닌 값이 status에 저장되면 에러처리.
- **line 69**: king 스레드를 detach한다. king 스레드는 join할 수 없게 되고, join 없이 리소스가 회수된다.
- **line 72**: 잠깐 대기해서 king 스레드가 main이 끝나기 전에 안전하게 lock을 획득할 수 있도록 한다.
- **line 74**: main이 lock을 얻으려고 시도하지만 king 스레드에서, 부하 스레드들에 대해 join이 끝나서, unlock 하기 전까지 전역 lock을 얻지 못하게 된다(blocking). 시간이 지나 king 스레드에서 unlock 하고나면, lock을 획득한다.

  while문을 쓰지 않고 잠들어 있다가 OS에 의해 깨어나는 방식이다. main은 king 스레드가 거의 종료될 때까지, thread_05.c와 달리 busy waiting하지 않고도 대기할 수 있다.
- **line 75~76**: 안전하게 unlock 하고 lock에 할당된 뮤텍스를 해제한다.

결과 스크린샷: