

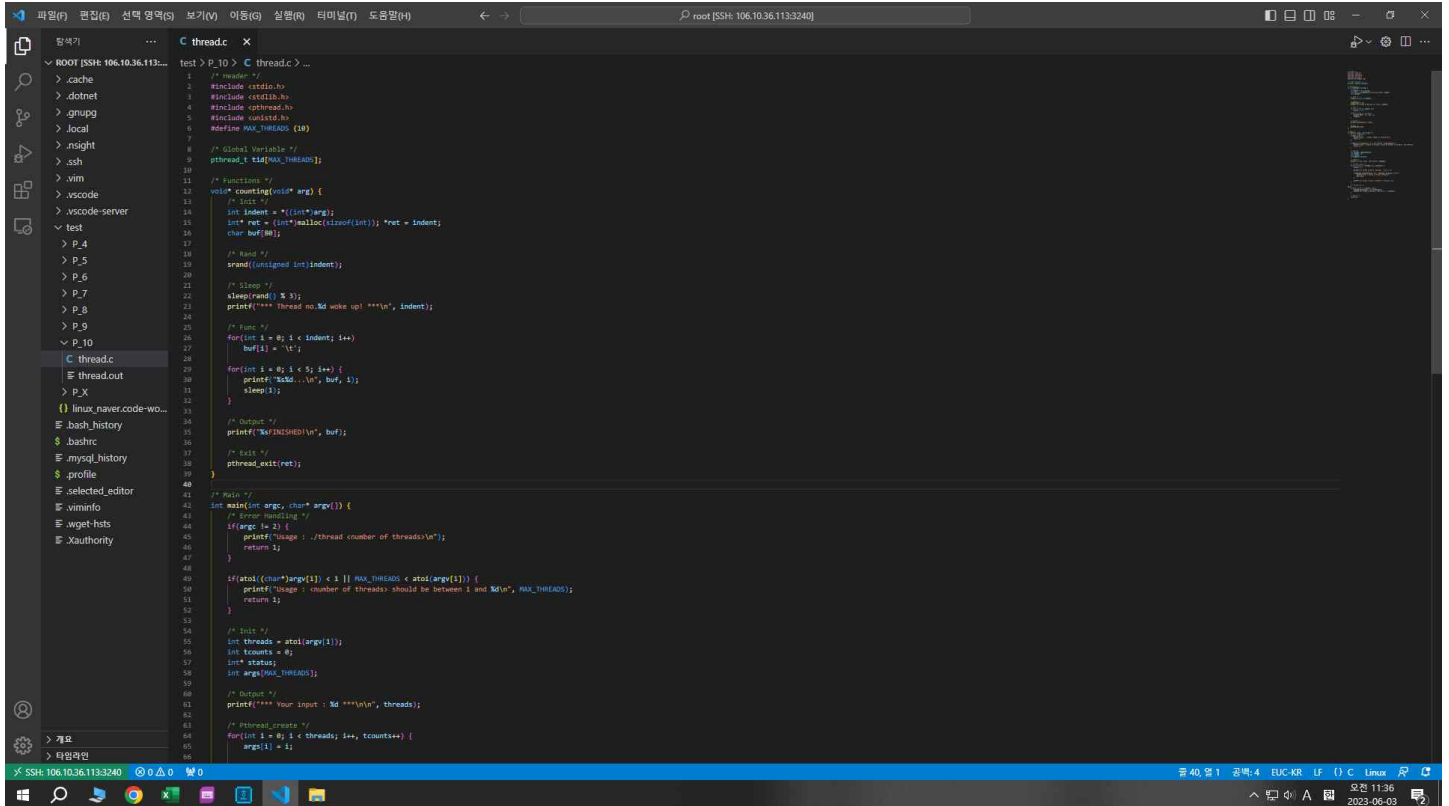
201818716

컴퓨터공학부

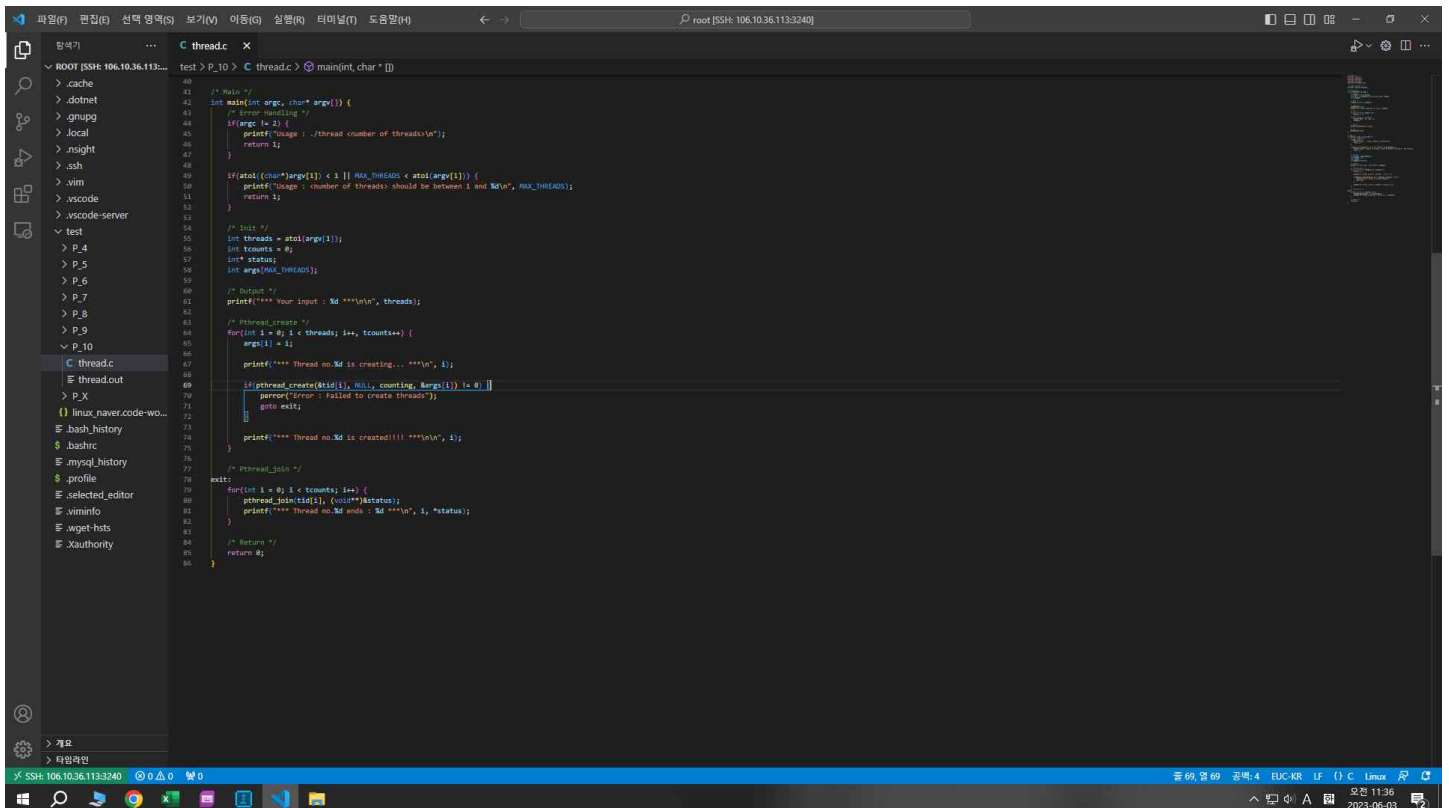
김용현

※ 실습 간 Putty · Xshell 대신, VScode상의 SSH 확장프로그램을 이용하여 실습 서버에 접근하였습니다.

[코드 스크린샷] : PPT상에 올라운 코드와 동일한 코드이며, 코드 정리 및 디버그 용도의 콘솔 출력만 추가 하여 작성하였음.

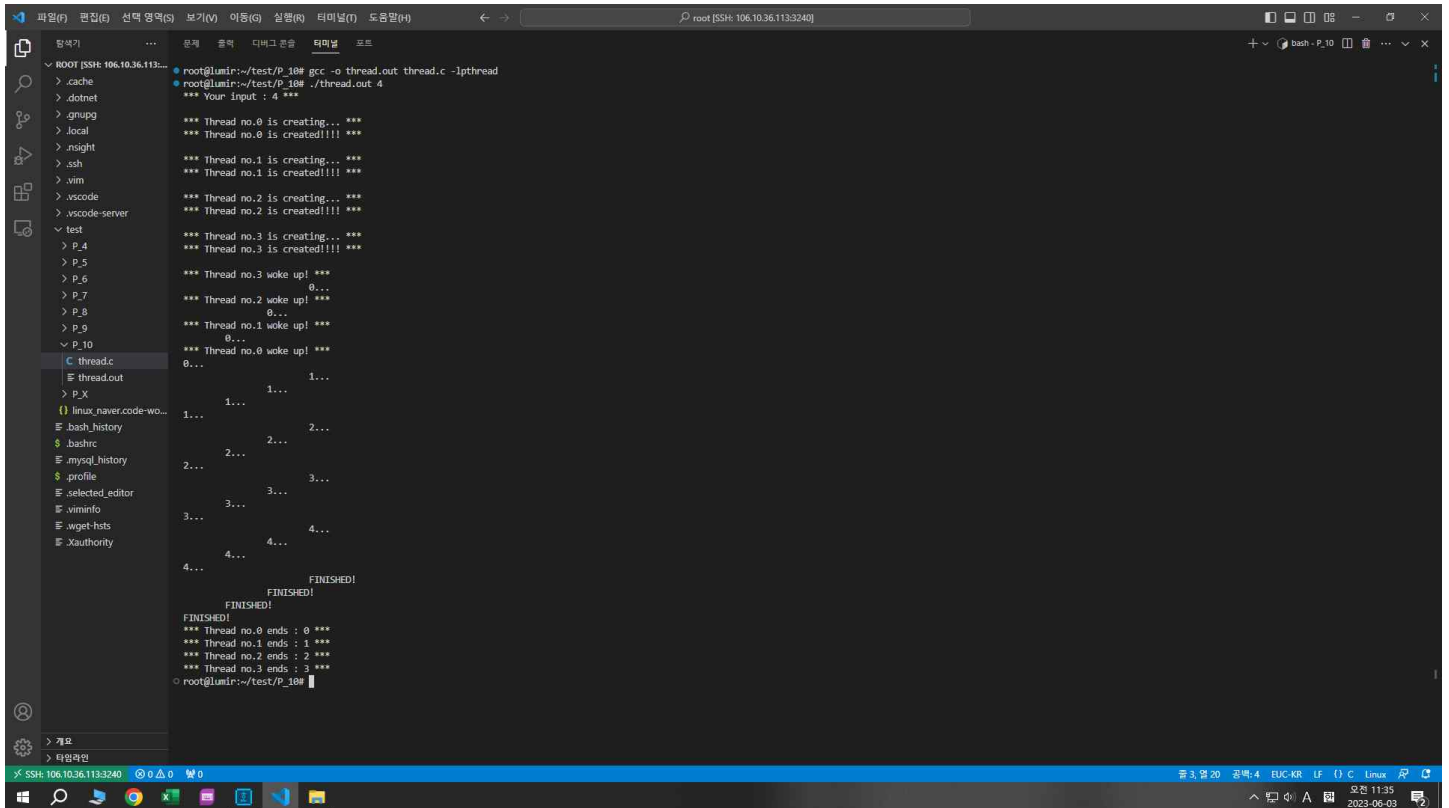


```
1  /* Header */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <unistd.h>
5  #include <sys/stat.h>
6  #define MAX_THREADS (10)
7
8  /* Global Variable */
9  pthread_t tid[MAX_THREADS];
10
11 /* Functions */
12 void* counting(void* arg) {
13     /* Init */
14     int indent = *((int*)arg);
15     int* ret = (int*)malloc(sizeof(int)); *ret = indent;
16     char buf[64];
17
18     /* Loop */
19     srand((unsigned)time(NULL));
20
21     /* Sleep */
22     sleep(rand() % 3);
23     printf("**** Thread no.%d wake up! ****\n", indent);
24
25     /* Func */
26     for(int i = 0; i < indent; i++)
27         buf[i] = 'A';
28
29     for(int i = 0; i < 5; i++) {
30         printf("A...%s\n", buf, i);
31         sleep(1);
32     }
33
34     /* Output */
35     printf("A[SHED]\n", buf);
36
37     /* Exit */
38     pthread_exit(ret);
39 }
40
41 /* Main */
42 int main(int argc, char* argv[]) {
43     /* Error Handling */
44     if(argc < 2) {
45         printf("Usage : ./thread number of threads\n");
46         return 1;
47     }
48
49     if(atoi(char*)argv[1] < 1 || MAX_THREADS < atoi(argv[1])) {
50         printf("Usage : number of threads should be between 1 and %d\n", MAX_THREADS);
51         return 1;
52     }
53
54     /* Init */
55     int threads = atoi(argv[1]);
56     int tcounts = 0;
57     int* status;
58     int args[MAX_THREADS];
59
60     /* Output */
61     printf("**** Your input : %d ****\n\n", threads);
62
63     /* pthread_create */
64     for(int i = 0; i < threads; i++, tcounts++) {
65         args[i] = i;
66
67         printf("**** Thread no.%d is creating... ****\n", i);
68
69         if(pthread_create(&tid[i], NULL, counting, &args[i]) != 0) {
70             perror("Error : failed to create thread");
71             goto exit;
72         }
73
74         printf("**** Thread no.%d is created!!! ****\n\n", i);
75     }
76
77     /* pthread_join */
78     exit:
79     for(int i = 0; i < tcounts; i++) {
80         pthread_join(tid[i], (void**)&status);
81         printf("**** Thread no.%d ends : %d ****\n", i, *status);
82     }
83
84     /* Return */
85     return 0;
86 }
```



```
1  /* Header */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <unistd.h>
5  #include <sys/stat.h>
6  #define MAX_THREADS (10)
7
8  /* Global Variable */
9  pthread_t tid[MAX_THREADS];
10
11 /* Functions */
12 void* counting(void* arg) {
13     /* Init */
14     int indent = *((int*)arg);
15     int* ret = (int*)malloc(sizeof(int)); *ret = indent;
16     char buf[64];
17
18     /* Loop */
19     srand((unsigned)time(NULL));
20
21     /* Sleep */
22     sleep(rand() % 3);
23     printf("**** Thread no.%d wake up! ****\n", indent);
24
25     /* Func */
26     for(int i = 0; i < indent; i++)
27         buf[i] = 'A';
28
29     for(int i = 0; i < 5; i++) {
30         printf("A...%s\n", buf, i);
31         sleep(1);
32     }
33
34     /* Output */
35     printf("A[SHED]\n", buf);
36
37     /* Exit */
38     pthread_exit(ret);
39 }
40
41 /* Main */
42 int main(int argc, char* argv[]) {
43     /* Error Handling */
44     if(argc < 2) {
45         printf("Usage : ./thread number of threads\n");
46         return 1;
47     }
48
49     if(atoi(char*)argv[1] < 1 || MAX_THREADS < atoi(argv[1])) {
50         printf("Usage : number of threads should be between 1 and %d\n", MAX_THREADS);
51         return 1;
52     }
53
54     /* Init */
55     int threads = atoi(argv[1]);
56     int tcounts = 0;
57     int* status;
58     int args[MAX_THREADS];
59
60     /* Output */
61     printf("**** Your input : %d ****\n\n", threads);
62
63     /* pthread_create */
64     for(int i = 0; i < threads; i++, tcounts++) {
65         args[i] = i;
66
67         printf("**** Thread no.%d is creating... ****\n", i);
68
69         if(pthread_create(&tid[i], NULL, counting, &args[i]) != 0) {
70             perror("Error : failed to create thread");
71             goto exit;
72         }
73
74         printf("**** Thread no.%d is created!!! ****\n\n", i);
75     }
76
77     /* pthread_join */
78     exit:
79     for(int i = 0; i < tcounts; i++) {
80         pthread_join(tid[i], (void**)&status);
81         printf("**** Thread no.%d ends : %d ****\n", i, *status);
82     }
83
84     /* Return */
85     return 0;
86 }
```

## [실행 스크린샷]



```
root@lumin:~/test/P_10# gcc -o thread.out thread.c -lpthread
root@lumin:~/test/P_10# ./thread.out 4
*** Your Input : 4 ***

*** Thread no.0 is creating... ***
*** Thread no.0 is created!!! ***

*** Thread no.1 is creating... ***
*** Thread no.1 is created!!! ***

*** Thread no.2 is creating... ***
*** Thread no.2 is created!!! ***

*** Thread no.3 is creating... ***
*** Thread no.3 is created!!! ***

*** Thread no.3 woke up! ***
0...

*** Thread no.2 woke up! ***
0...

*** Thread no.1 woke up! ***
0...

*** Thread no.0 woke up! ***
0...

1...
1...
2...
2...
3...
3...
4...
4...

FINISHED!
FINISHED!
FINISHED!
*** Thread no.0 ends : 0 ***
*** Thread no.1 ends : 1 ***
*** Thread no.2 ends : 2 ***
*** Thread no.3 ends : 3 ***
root@lumin:~/test/P_10#
```

## [1\_변수 설명]

### <전역변수>

#. pthread\_t tid[MAX\_THREADS] : 최대 10개의, pthread\_create를 통해 만들어진 쓰레드들을 관리하기 위한, pthread\_t 타입의 tid 배열을 선언. 이때, 모든 위치에서 해당 변수에 접근할 수 있게 하기 위해 전역변수로 선언하였다. 전역변수로 선언할 경우, Data영역에 배치가 되어, 다른 모든 쓰레드들도 여기에 접근하여 데이터를 공유하며 쓸 수 있게 된다. (단, 일반적인 경우, 동기화 문제가 발생할 수 있음에 유의하자.)

### <지역변수 : main>

#. int threads : argv의 2번째 인자로 들어온 입력값을 저장하기 위한 int형 변수. 즉, threads는 내가 생성하고자 하는 쓰레드의 개수를 의미한다. (입력값)

#. int tcounts : 실제로 생성된 쓰레드의 개수를 의미한다.

#. int\* status : counting 함수에서의 종료 상태를 전달받아 저장하기 위한 포인터 변수. counting 함수에서 pthread\_exit()을 통해 종료 상태를 전달하면, main 함수에서 쓰레드 종료를 대기하는 pthread\_join()을 통해 status값을 전달 받는다. (프로세스에서의 wait()과 유사한 역할)

#. int args[MAX\_THREADS] : 제일 눈여겨 봐야하는 변수. for문에서 args[i] = i 에서의 활용법을 살펴보면, 그 역할을 제대로 알 수 있다. pthread\_create()에서 &i를 인자로 넘겨주지 않고, &args[i]를 인자로 넘겨주는 이유는 다음과 같다. 만약, i의 주소값을 인자로 넘겨주고 해당 i값을 이용하게 된다면, i값은 반복문의 ++연산에 의해 그 값이 계속 변화하는 값이 된다. (나는 i=1 이란 값을 인자로 전달 받아 쓰고 싶었는데, 어느새 보니 i=2, i=3, ... 등이 되어있는 것이다.) 따라서, 이를 방지하여 고정된 값을 사용하기 위해 args[]를 사용하는 것이다. 즉, args[]를 통해 변하지 않는 값을 사용할 수 있게 된다. 이를 counting 함수와 관련해서 살펴보면, i를 넘겨줄 당시에는 i=1이었지만, i값을 indent 변수에 대입하려 하니, i값이 2, 3, 4 등으로 변해있는 상황을 의미한다. 이는, 쓰레드가 병렬적으로 동작하기에, 동작하는 순서가 어떻게 될지 모르기에 이와 같은 일이 발생하는 것이다.

### <지역변수 : counting>

#. void\* arg : pthread\_create()의 4번째 인자로 전달된 값을 의미한다.

#. int indent : indent는 인자로 전달받은 arg값. 즉, &args[]값을 의미한다. 즉, 1을 넘겨받았으면 그 값이 1, 2를 넘겨받았으면 그 값이 2, ... 등이 되는 것이다.

#. int\* ret : 동적 할당을 통해 선언해야만 하며, indent의 값이 그대로 대입되어, pthread\_exit()을 실행할 시 그 값이 반환된다. 즉, 자신이 부여받은 번호인 indent를 다시 반환하는 역할을 한다.

#. char buf[80] : 'wt'문자를 저장하기 위한 임시 버퍼로 사용된다. indent값 만큼 buf에 들어쓰기로 공백을 채워준다.

[2\_코드 구조에 대한 해설] : 위 코드의 주석문을 따라가며 설명.

〈전체 구조 요약〉

0. 헤더파일 include와 전역변수의 초기화가 진행된다.
1. main 함수를 통해 진입한 후, Error Handling 과정을 통해, 입력값이 문제없나 확인받는다.
2. Init 과정을 통해, 변수의 초기화 진행.
3. Output 과정을 통해, threads의 값을 출력한다.
4. Pthread\_create 과정을 통해, 쓰레드를 최대 threads 개수(input값)만큼 반복 생성한다. (오류 발생 시, error를 출력하며, exit 레이블로 이동한다.)
5. 각 쓰레드들은 counting 함수에 쓰여있는 내용들을 수행하며, 모든 수행이 종료될 시, pthread\_exit()을 통해 상태값을 반환하며, pthread\_join() 이후 소멸되게 된다.
6. Pthread\_join 과정을 통해, main 함수에서는 쓰레드들의 종료를 대기한다. 이때, int i = 0 부터 시작하여 pthread\_join()을 시행하므로, 0번 쓰레드의 종료부터 대기하게 된다. 따라서, 아무리 2번, 3번 쓰레드가 먼저 종료되었다 하더라도 이는 우선적으로 처리되지 못하며, 어쩔 수 없이 0번 쓰레드가 종료될 때 까지 기다렸다가, 0번 쓰레드부터 순차적으로 처리가 진행된다.
7. main 함수가 종료된다.

〈전체 구조 상세〉

```
/* Header */
```

```
#. 헤더파일의 include와 매크로 상수를 지정한다.
```

```
/* Global Variable */
```

```
#. 전역변수 pthread_t tid[]를 선언한다. (변수의 용도에 대한 해설은 위에서 언급하였다.)
```

```
/* Functions */
```

```
#. pthread_create()에서 쓰레드로 수행할 코드의 함수 포인터가 될, counting 함수를 만들었다. (쓰레드들이 진행할 일을 start_routine으로 정의한 것.) 이 함수는 초시계 역할을 진행하며, 1초마다 카운트를 진행한다.)
```

```
/* Init */
```

```
#. 변수를 선언하며, 값을 대입해 초기화 해준다.
```

```
/* Rand */
```

```
#. random seed를 통해 랜덤값을 만들 준비를 한다. 이때, srand()를 indent라는 고정값으로 사용했기 때문에, 예제 실행 결과에서 어느정도 유사한 값이 나타날 수 있음에 유의하자.
```

```
/* Sleep */
```

```
#. sleep(rand() % 3)을 통해 0~2초 간격으로 해당 쓰레드를 sleep 시킨다. 이는, 쓰레드의 동작을 확인하기 위해, 쓰레드 별로 동작시간에 차이를 둔 것이다.
```

```
/* Func */
```

```
#. 첫번째 반복문을 통하여, 전달받은 값만큼 buf에 들어쓰기로 공백을 채워준다.
```

```
#. 두번째 반복문을 통하여, 1초마다 해당 쓰레드를 카운트 해준다.
```

```
/* Output */
```

```
#. 쓰레드가 모든 일을 수행하였을 경우, FINIDHSED 라는 문자열을 출력해준다.
```

```
/* Exit */
```

```
#. ret 값을 반환하며, 쓰레드를 종료한다.
```

```
/* Main */
```

```
/* Error Handling */
```

```
#. 잘못된 인자가 전달되었을 경우, 오류메시지를 출력하며 종료한다.
```

```
/* Init */
```

```
#. 변수를 선언하며, 값을 대입해 초기화 해준다.
```

```
/* Output */
```

```
#. 입력값으로 들어온 thread 값을 출력한다.
```

```
/* Pthread_create */
```

```
#. Pthread_create 과정을 통해, 쓰레드를 최대 threads 개수(input값)만큼 반복 생성한다. (오류 발생 시, error를 출력하며, exit 레이블로 이동한다.) 이때, 각 쓰레드들은 counting 함수에 쓰여있는 내용들을 수행하며, 모든 수행이 종료될 시, pthread_exit()을 통해 상태값을 반환하며, pthread_join() 이후 소멸되게 된다.
```

```
/* Pthread_join */
```

```
#. Pthread_join 과정을 통해, main 함수에서는 쓰레드들의 종료를 대기한다. 이때, int i = 0 부터 시작하여 pthread_join()을 시행하므로, 0번 쓰레드의 종료부터 대기하게 된다. 따라서, 아무리 2번, 3번 쓰레드가 먼저 종료되었다 하더라도 이는 우선적으로 처리되지 못하며, 어쩔 수 없이 0번 쓰레드가 종료될 때 까지 기다렸다가, 0번 쓰레드 부터 순차적으로 처리가 진행된다.
```

```
/* Return */
```

```
#. 값을 반환하며, 프로그램을 종료한다.
```

### [3\_실행 결과에 대한 해설]

- #. 쓰레드들의 생성 시점에 대해 우선 알아보자. pthread\_create()을 수행하는 반복문을 int i = 0부터 순차적으로 실행하였으므로, 쓰레드는 0번 쓰레드 부터 순차적으로 생성되어 3번 쓰레드 까지 만들어진다. (콘솔 출력을 보면, Thread no.0 is created!!!! 이후, Thread no.3 is created!!!! 까지 순차적으로 진행된다.)
- #. 쓰레드들의 수행 시점은 제 각기 차이가 있다. sleep(rand() % 3)에 의해 각각의 쓰레드들의 수행 시점이 결정되는데, 이번 실행 결과에서는 3번 쓰레드부터 0번 쓰레드까지 역순으로 수행되었다. (콘솔 출력을 보면, Thread no.3 woke up! 부터 시작하여 Thread no.0 woke up! 까지 역순으로 진행된다.) 이때, random seed를 indent라는 고정값으로 사용했기 때문에, 여러번 실행 결과를 출력해봐도 거의 비슷한 결과만이 출력된다. (가끔 조금 다른 결과가 출력되는데, 이는 CPU 스케줄링 방식에 따라 출력 순서에 조금씩 변동이 생기기 때문이다.)
- #. 쓰레드들의 종료(소멸) 시점을 살펴보자. 우선, 3번 쓰레드 부터 0번 쓰레드 까지 3, 2, 1, 0 의 순서대로 쓰레드의 수행이 종료된다. (FINISHED가 출력되는 순서를 확인하면 된다.) 하지만, 3번 쓰레드가 먼저 종료되었다고 해도 pthread\_join() 에서 먼저 처리해주지는 않는데, 그 이유는 위에서 언급한 이유와 같다. 다시 한번 서술하면, Pthread\_join 과정을 통해, main 함수에서는 쓰레드들의 종료를 대기한다. 이때, int i = 0 부터 시작하여 pthread\_join()을 시행하므로, 0번 쓰레드의 종료부터 대기하게 된다. 따라서, 아무리 2번, 3번 쓰레드가 먼저 종료되었다 하더라도 이는 우선적으로 처리되지 못하며, 어쩔 수 없이 0번 쓰레드가 종료될 때 까지 기다렸다가, 0번 쓰레드부터 순차적으로 처리가 진행된다. 따라서, 종료(소멸)시점에 대한 콘솔 출력을 확인하면, Thread no.0 ends 부터 시작하여, Thread no.3 ends 까지 순차적으로 처리되는 것을 볼 수 있다.
- #. 따라서, 위에서 언급한 생성 · 수행 · 종료 시점에 대한 이유로, 실행결과와 같이 값이 출력되는 것이다.