**REPORT**

**| Embedded System**

**| Event flag and Semaphore**

**| 컴퓨터정보공학과**

**| 12141540 박영창**

# 내용

5주차 실습 강의 시간에는 Event flag, Semaphore 에 관해 배웠습니다.

* Semaphore
  + Semaphore는 여러 task에서 공유 자원( ex 전역변수 )를 사용할 때, 동시에 같은 자원이 사용되어 충돌이나 문제가 생기는 것을 방지하기 위해 사용하는 OS\_EVENT \* 형 변수로 uC/OS-II에서 이를 사용할 수 있는 함수를 제공한다.
  + 선언
    - OS\_EVENT \* Semaphore\_name

<설명>

OS\_EVENT \* 형으로 선언

<인자>

Semaphore\_name : 사용할 Semaphore 이름

* + 초기화
    - OS\_EVENT \* OSSemCreate(INT16U value)

<설명>

Semaphore를 만든다

<인자>

value : Semaphore의 초기값

* + Wait
    - Void OSSemPend(OS\_EVENT \* ptr, INT16U timeout, INT8U \*err)

<설명>

Semaphore ptr로부터 신호를 받는다

<인자>

ptr : OSSemCreate()에 의해 만들어진 Semaphore 정보

timeout : 대기시간

err : 에러 코드

* + Signal
    - INT8U OSSemPost(OS\_EVENT \* ptr)

<설명>

Semaphore ptr에 신호를 전달한다

<인자>

ptr : OSSemCreate()에 의해 만들어진 Semaphore 정보

* Event flag
  + Task 간의 동기화(synchronization) 중에 Mailbox, Message Queue처럼 직접적으로 값을 전달하지는 못하지만, Semaphore처럼 신호를 전달할 수 있는 .서비스이다. Semaphore,와 달리 bit의 조합에 의해 동기화를 진행한다.
  + 선언
    - OS\_FLAG\_GRP \*n\_grp

<설명>

OS\_FLAG\_GRP \* 형으로 선언

<인자>

n\_grp : 사용할 flag 이름

* + 초기화
    - OS\_FLAG\_GRP \* OSFlagCreate(OS\_FLAGS f, INT8U \*err)

<설명>

Event Flag를 만든다

<인자>

f : event flag의 초기값

err : 에러 값

* + Wait
    - Void\* OSFlagPend(OS\_FLAG\_GRP \*p, OS\_FLAGS f, INT8U opt, INT8U t, INT8U \*err)

<설명>

Event Flag인 p에서 인자로 주어진 bit 의 조건(f, opt)이 만족될 때까지 대기한다

<인자>

p : OSFlagCreate()에 의해 만들어진 event flag 정보

f : 비교하고자 하는 bit 값

opt : 조건

OS\_FLAG\_WAIT\_CLR\_ALL – f에 해당하는 모든 bit clear

OS\_FLAG\_WAIT\_CLR\_ANY – f에 해당하는 어떤 bit clear

OS\_FLAG\_WAIT\_SET\_ALL – f에 해당하는 모든 bit set

OS\_FLAT\_WAIT\_SET\_ANY – f에 해당하는 어떤 bit set

OS\_FLAG\_CONSUME – 조건이 만족되면 flag의 모든 bit clear

t : 대기시간

err : 에러 코드

* + Signal
    - OS\_FLAGS OSFlagPost(OS\_FLAG\_GRP \*p, OS\_FLAGS f, INT8U opt, INT8U \*err)

<설명>

Event Flag인 p에 주어진 bit 신호를 준다.

<인자>

p : OSFlagCreate()에 의해 만들어진 event flag 정보

f : Set/Clear 시키고자 하는 bit

opt : 실행할 작업

OS\_FLAG\_SET : f에 해당하는 bit set

OS\_FLAG\_CLEAR : f에 해당하는 bit clear

err : 에러 코드

# 과제 목표

* 5개의 태스크 생성 ( 1 : decision , 4 : random )
* 4 개의 Random task에서 0에서 63까지 임의의 숫자를 1개씩 생성한다
* 4 개의 Random task는 Semaphore, Event Flag를 이용해 생성된 숫자를 충돌 없이 전역변수에 저장면서 신호를 Decision Task에 전달한다.
* Decision Task는 Semaphore, Event Flag를 이용해 충돌 없이 전역변수에 저장된 4 개의 숫자를 받아 가장 작은 숫자를 저장한 task의 인덱스 위치에 ‘W’라는 문자를 저장하고, 나머지에는 ‘L’이라는 문자를 저장한다. ( 숫자가 같을 경우는 높은 우선순위 task의 인덱스 위치에 저장한다. )
* ‘W’가 저장된 위치에 해당하는 task는 자신의 우선순위에 맞는 색깔로 화면을 칠한다

1 – Red 2 – Blue 3 – Green 4 – Brown

* 모든 task들은 4초간 쉬고 난 후에 위 과정을 반복한다.

# 문제 해결

**변수**

* **전역변수**
  + Int send\_array[]

생성된 난수를 저장할 전역변수 배열

* + char receive\_array[]

최소값 결정 후 결과를 저장할 전역변수 배열

* **Semaphore 선언**
  + OS\_EVENT \*Sem

전역변수 보호를 위한 semaphore

* **Event Flag 선언**
  + OS\_EVENT \*s\_grp

난수 생성, 저장 후 신호를 나타내는 flag

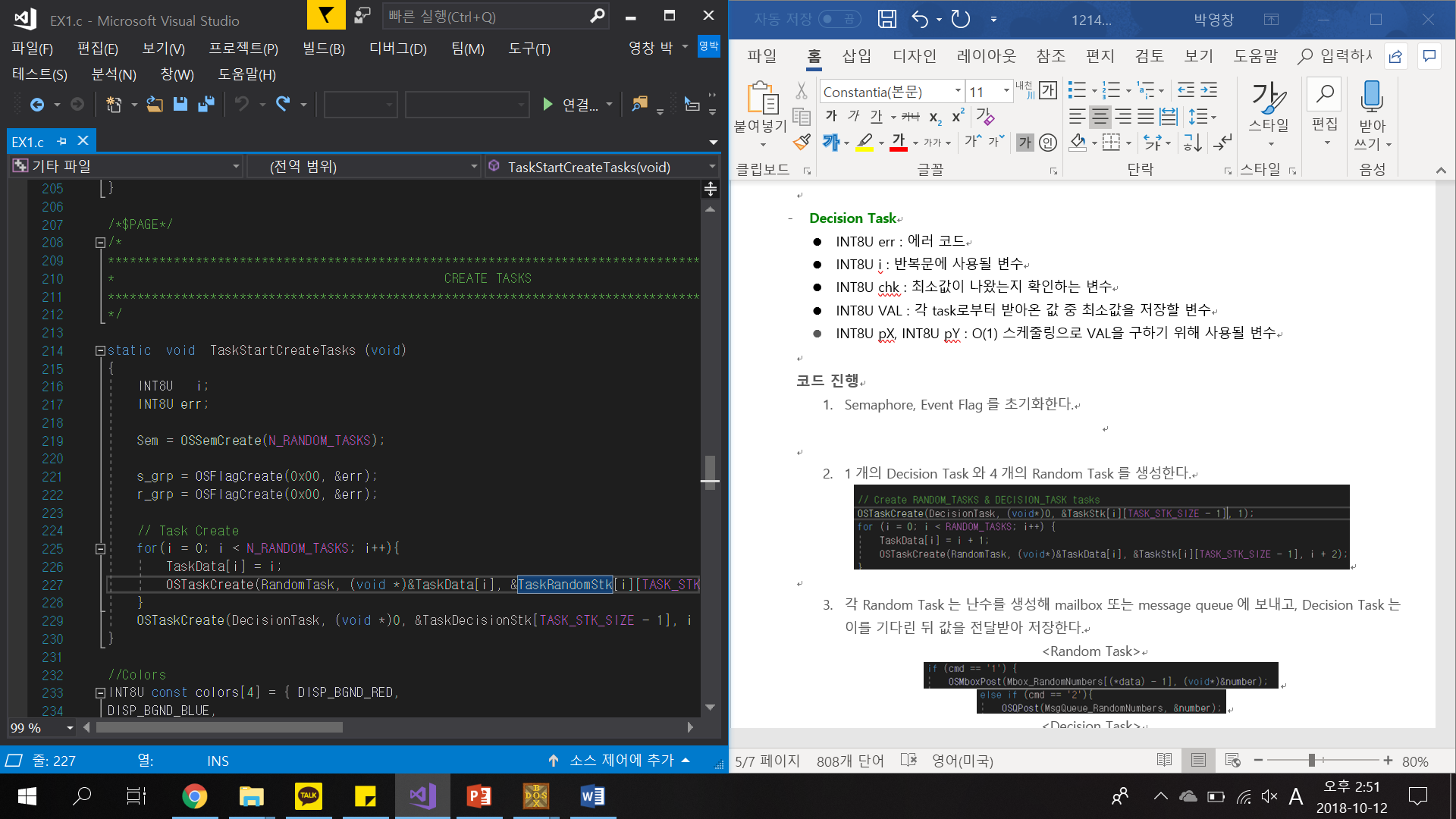
* + OS\_EVENT \*r\_grp

최소값 결정, 결과값 저장 후 신호를 나타내는 flag

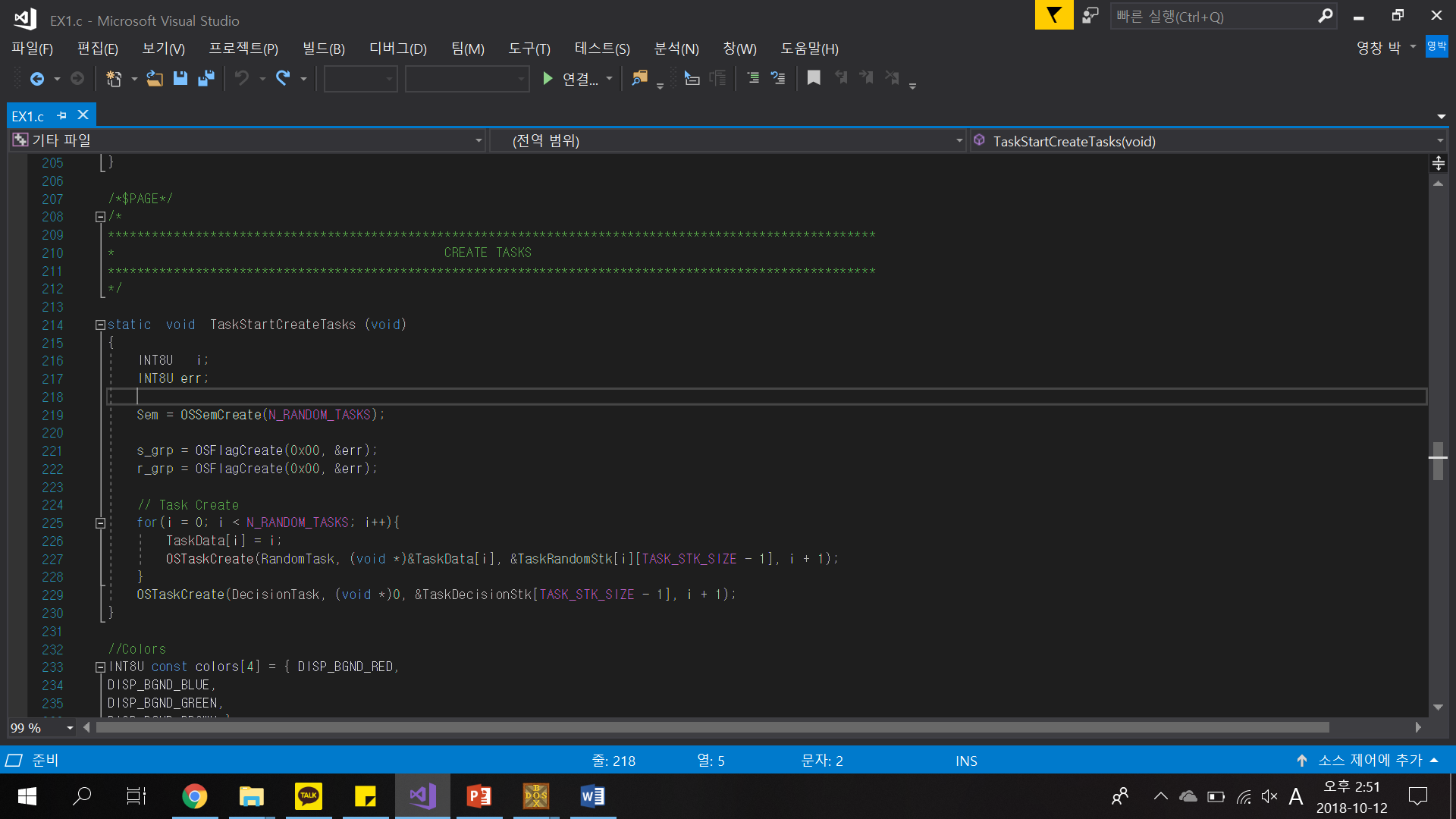
* **상수 define** 
  + #define N\_RANDOM\_TASKS 4 : Random Task의 개수
  + #define N\_DECISION\_TASKS 1 : Decision task의 개수
* INT8U const colors[4] : 우선순위에 맞는 색깔을 저장한 배열
* **스케줄링을 위한 변수**
  + INT8U const myMapTbl[]
  + INT8U const myUnMapTbl[]
  + INT8U myRdyTbl[8];
  + INT8U myRdyGrp;
* **Random Task**
  + INT8U i : 반복문을 위해 사용하는 변수
  + INT8U err; : 에러 코드
  + INT8U number; : 난수 생성
  + INT8U number\_of\_randomtask : task의 우선순위 번호
  + INT8U flagP : task 우선순위별로 OSFlagPost, OSFlagPend의 인자로 넘겨줄 bit
  + INT8U dispX; : 출력할 값의 x좌표
  + INT8U dispY; : 출력할 값의 y좌표
* **Decision Task**
  + INT8U err : 에러 코드
  + INT8U i : 반복문에 사용될 변수
  + INT8U chk : 최소값이 나왔는지 확인하는 변수
  + INT8U VAL : 각 task로부터 받아온 값 중 최소값을 저장할 변수
  + INT8U pX, INT8U pY : O(1) 스케줄링으로 VAL을 구하기 위해 사용될 변수

**코드 진행**

1. Semaphore, Event Flag를 초기화한다.



1. 1개의 Decision Task와 4개의 Random Task를 생성한다.

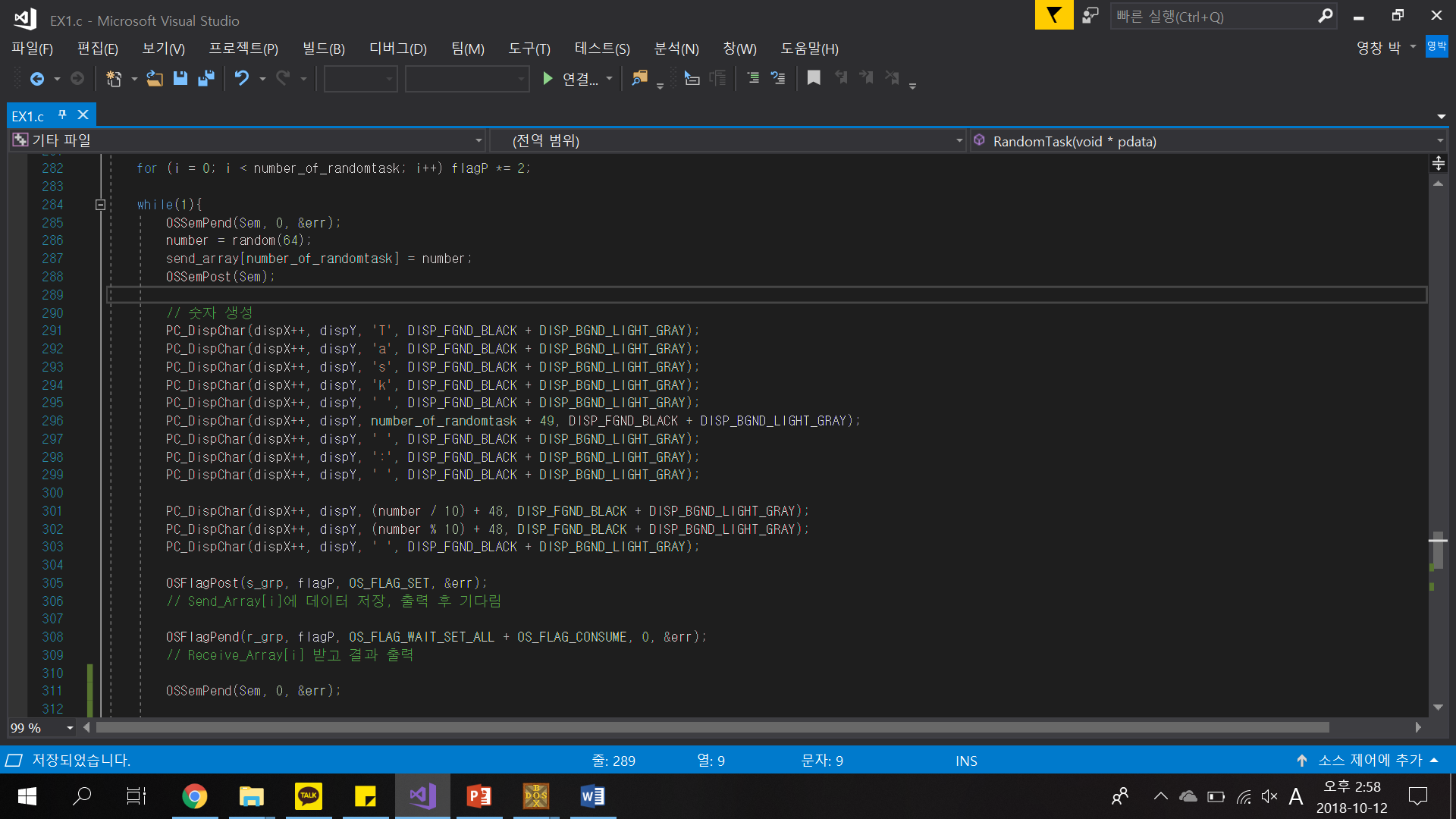


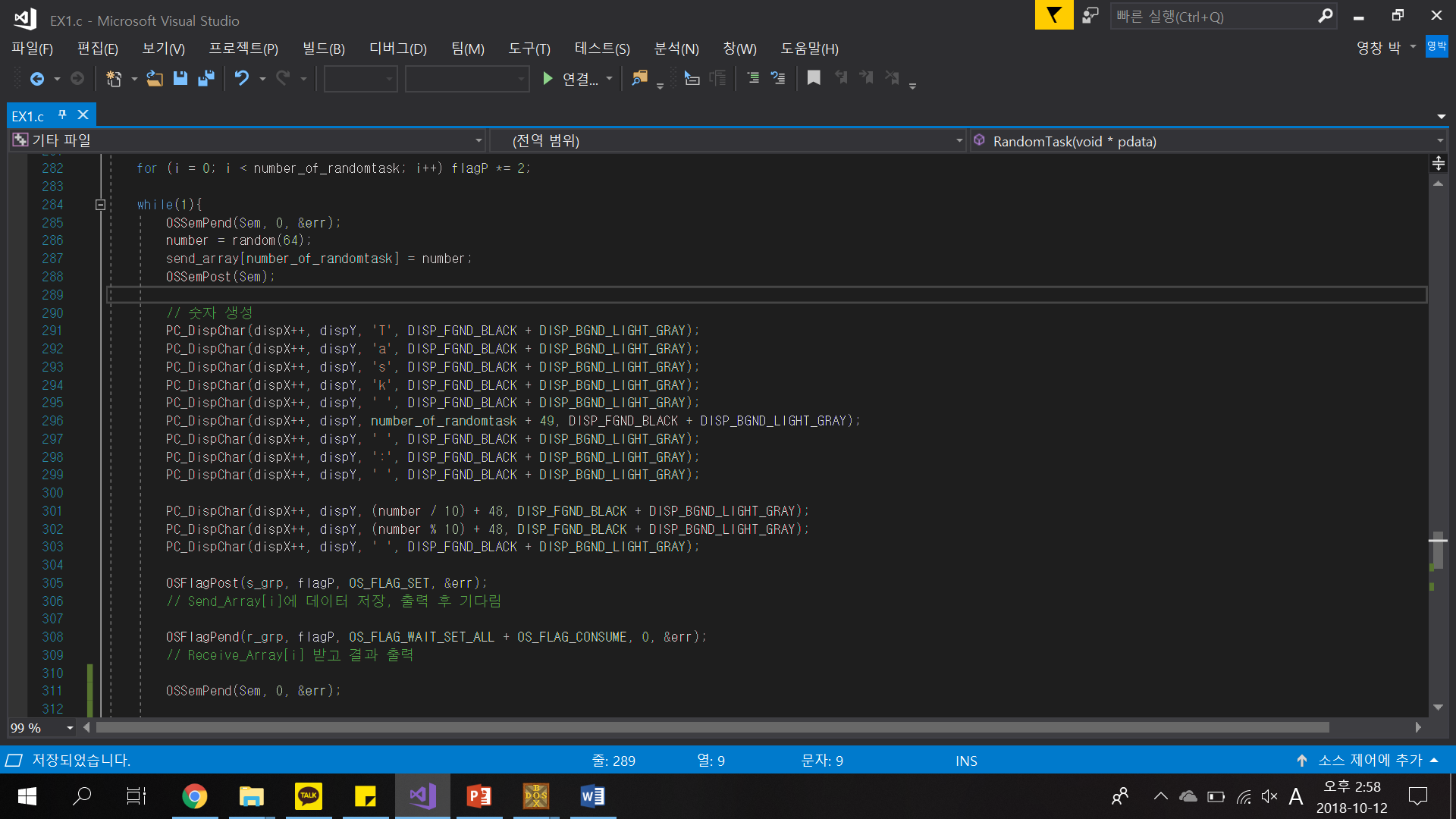
1. 각 Random Task는 난수를 생성해 semaphore의 보호를 받으며 전역변수 배열에 저장하고, 그 다음에 flag에 각 task에 해당하는 bit를 set시키는 신호를 보낸 뒤 Decision Task에서 신호가 오기를 기다린다. 이 때, Decision Task는 모든 task에서 신호를 보내기를 기다린다.

< task 1 : 0x01 (1) task 2 : 0x02 (2) task 3 : 0x04 (4) task 4 : 0x08 (8) >

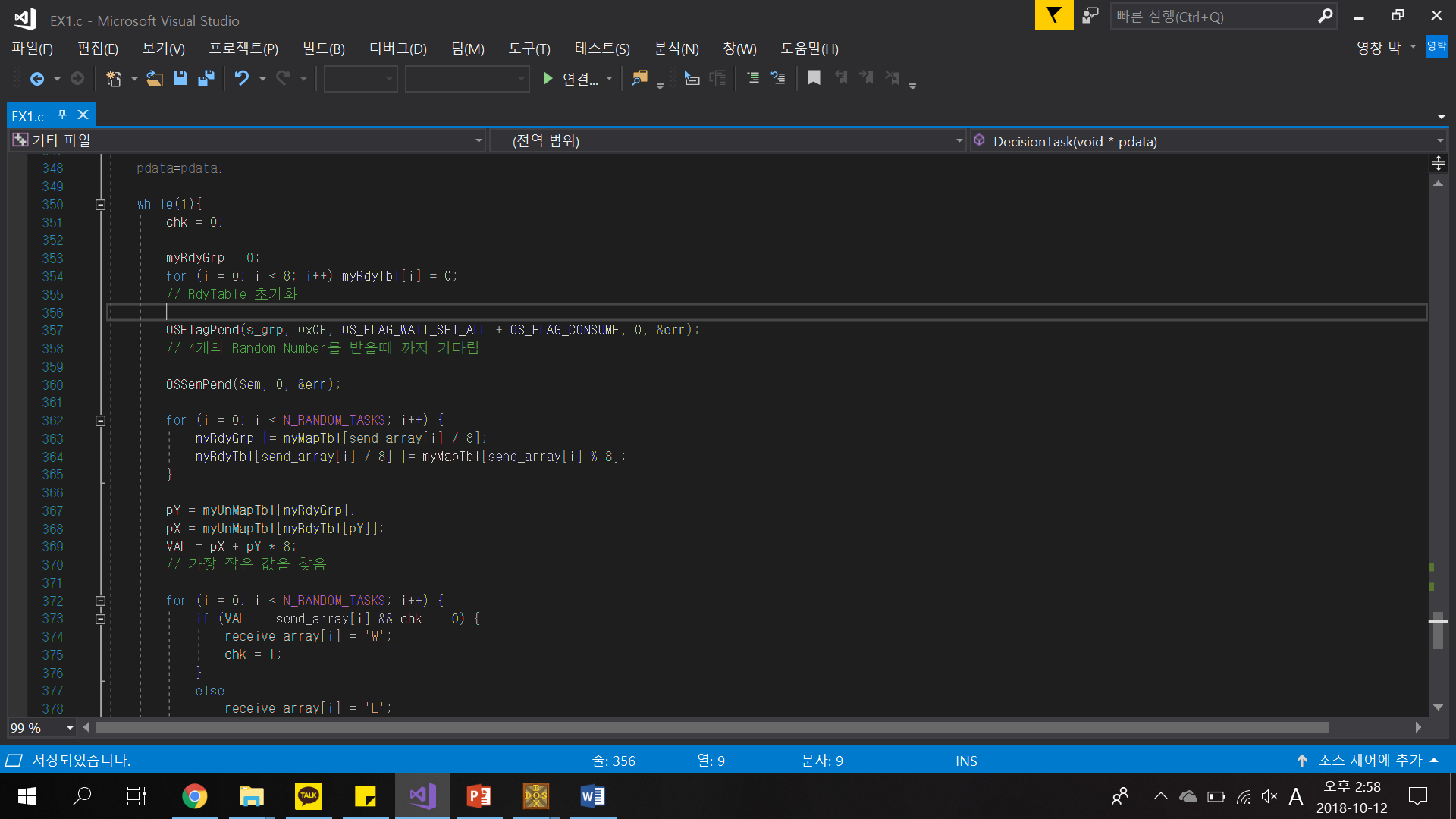
* + - * 모든 bit가 Set되는 상태 : 0x0F

<Random Task>

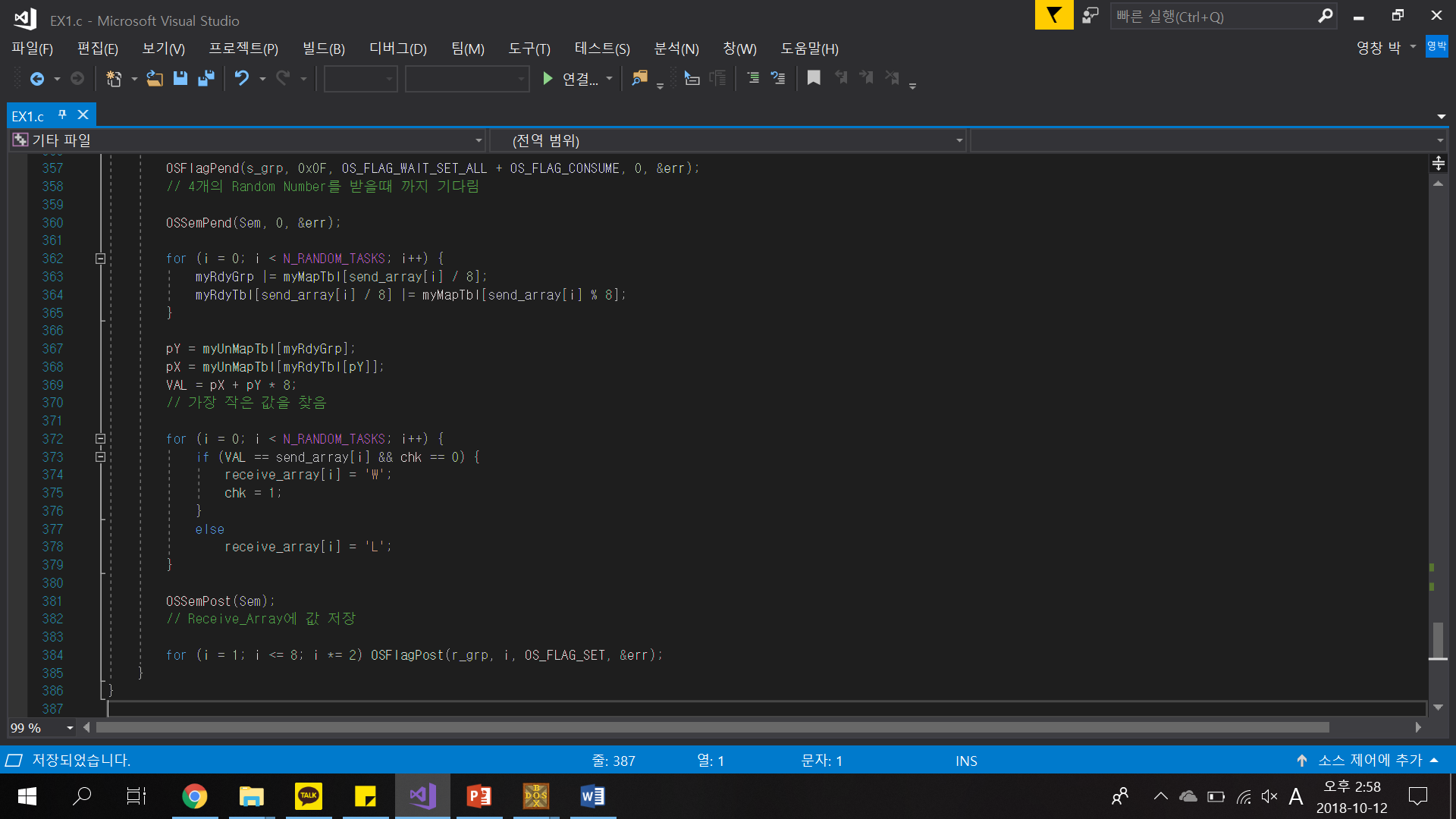




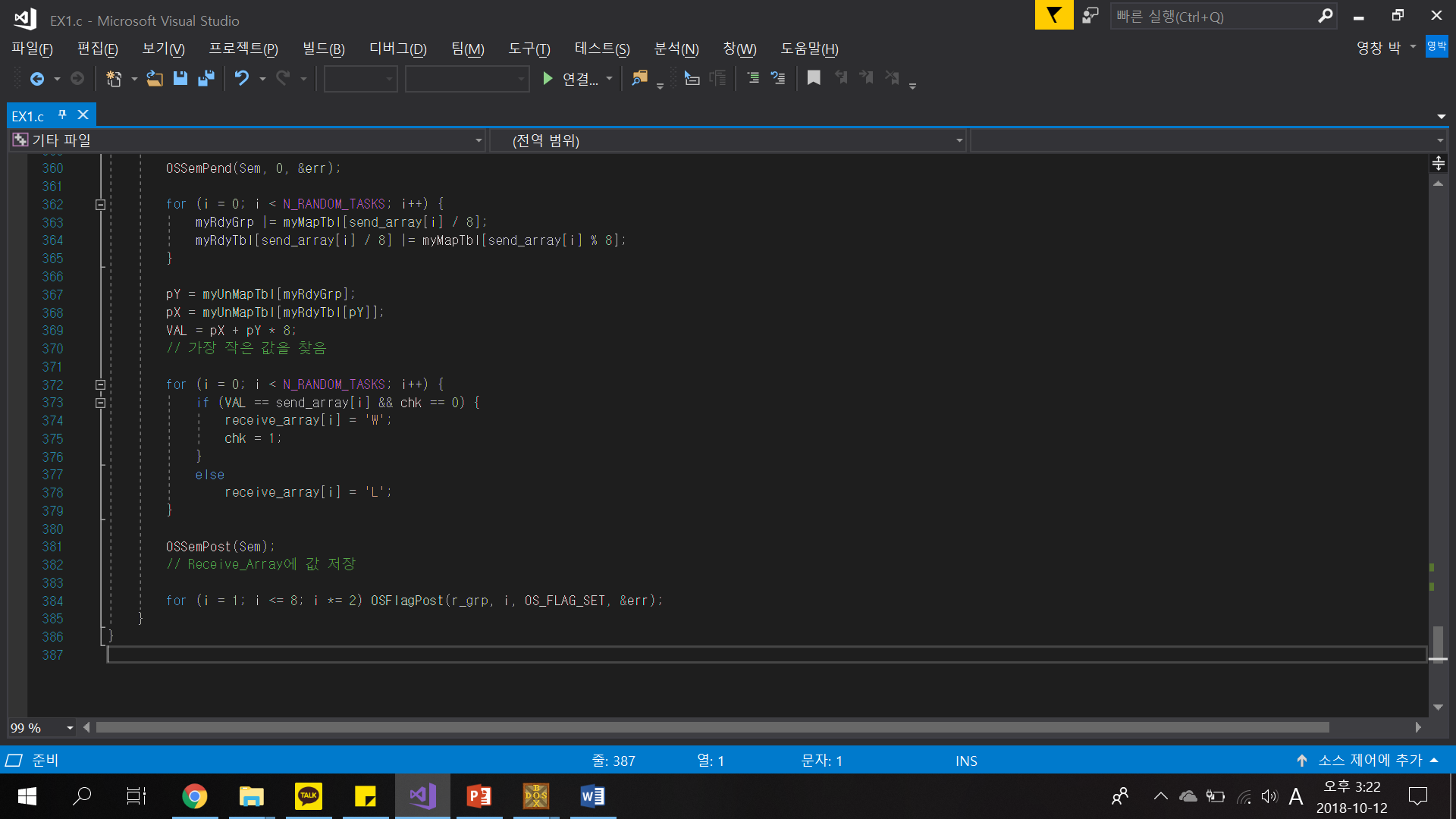
<Decision Task>



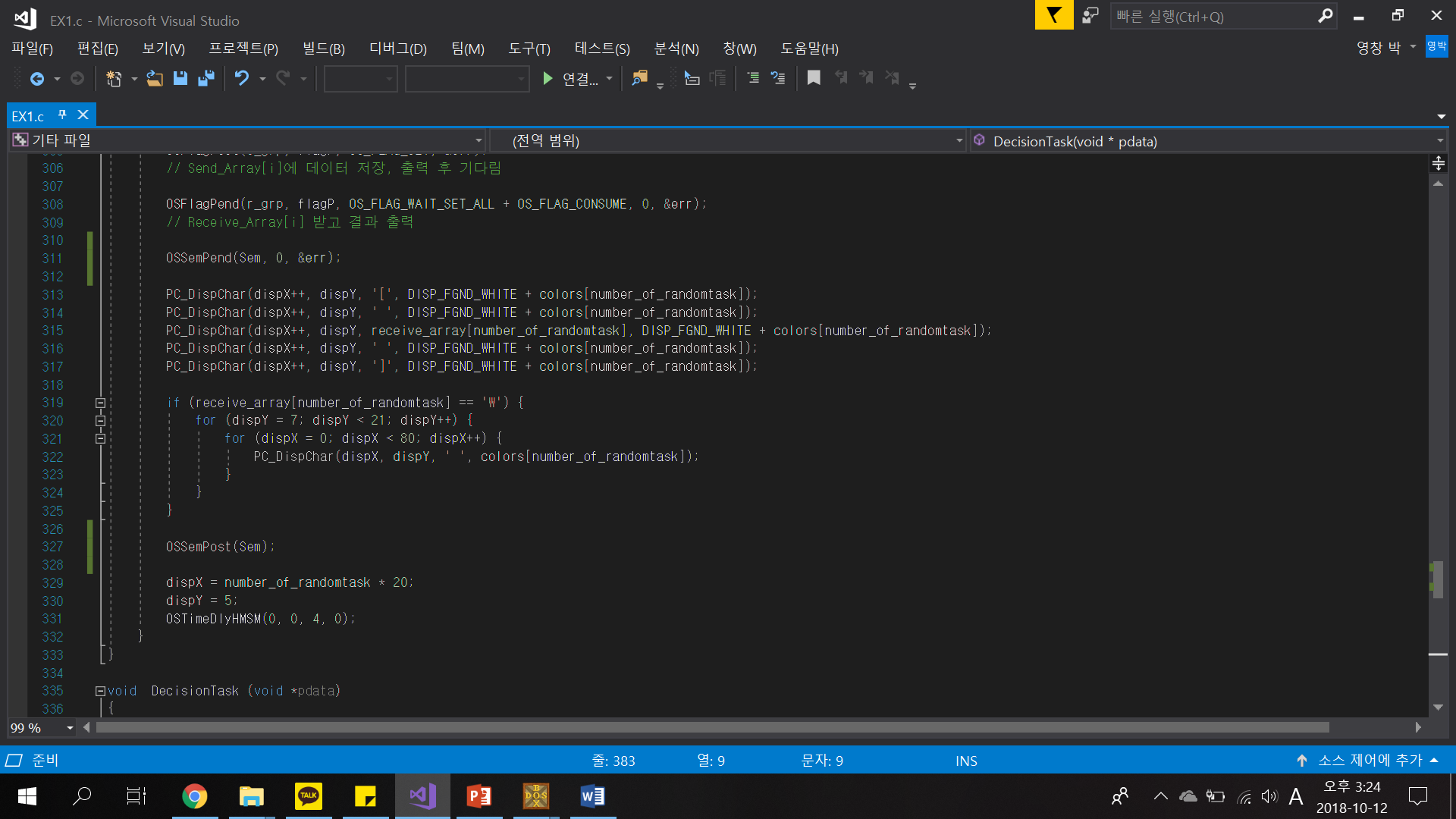
1. Decision Task에서 Semaphore의 보호를 받으며 전역변수에 접근해 O(1) 스케줄러를 이용해 최소값을 구한다.



1. Semaphore의 보호를 받으며 전역변수를 확인해 각 task에서 저장한 값과 최소값을 비교해 같은 값을 가지는 가장 적은 인덱스(가장 높은 우선순위)에게만 W를, 나머지에게는 L를 저장한다. 그 다음, 각 Random Task에 신호를 보낸다.



1. Random Task에서 Semaphore의 보호를 받으며 결과값이 저장된 배열에 접근해 W 인지 L인지 확인한다. 만약 W라면 해당 Task Number에 맞는 색으로 배경을 칠한다.



1. 위 과정을 4초에 한번씩 반복한다.

# 결과

# 