6주차 실습 보고서

실습 주제

- Semaphore, event flag를 활용한 task synchronization

성공 실패 여부

- 성공

원인 분석

1. Semaphore, event flag 생성

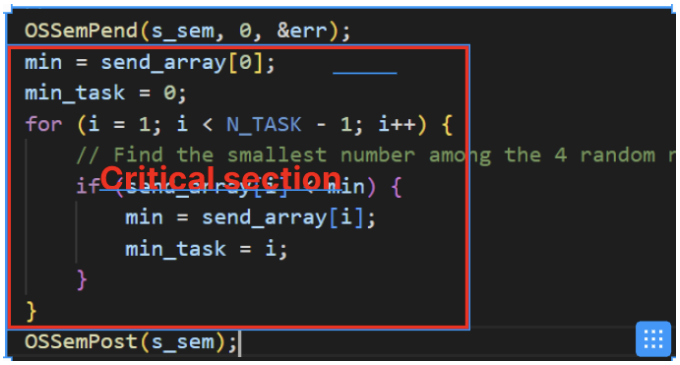
```
        if (select == 1) {
            r_sem = OSSemCreate(1);
            s_sem = OSSemCreate(1);
            s_grp = OSFlagCreate(0x00, &err);
            r_grp = OSFlagCreate(0x00, &err);
        }
```

전역변수 send_array[4], receive_array[4]를 위한 각각의 세마포 s_sem, r_sem을 생성 후

s_grp, r_grp event flag 생성.

2. Semaphore활용

전역변수 send_array, receive_array에 접근하는 모든 구역은 critical section으로, Mutual exclusion을 보장해야한다.
따라서 동기화 도구인 Semaphore을 이용하여 atomic operation을 구현하였다.



Event flag

```
/** RANDOM TASK
 * task 0 - 0x01
 * task 1 - 0x02
 * task 2 - 0x04
 * task 3 - 0x08
 */
OSFlagPost(s_grp, 0x01 << task_number, OS_FLAG_SET, &err);
OSFlagPend(r_grp, 0x01 << task_number, OS_FLAG_WAIT_SET_ALL +
```

1. 각 random task(0 ~ 3) 은 해당 코드에서 (0x01 << task_number) bit를 1로 세팅하고 Pend함수를 통해 wait상태가 된다.

```
/** DECISION TASK
*/
OSFlagPend(s_grp, 0x0F, OS_FLAG_WAIT_SET_ALL + OS_FLAG_CONSUM
```

2. 4개의 task가 모두 Post함수를 호출하면 s_grp 은 0x0F가 되고, 나머지 4개의 random task가 모두 wait state이기 때문에 decision task(4) 가 스케쥴링된다.

```
/** DECISION TASK
*/
OSFlagPost(r_grp, 0x0F, OS_FLAG_SET, &err);
```

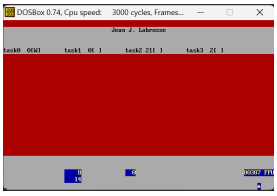
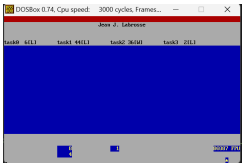
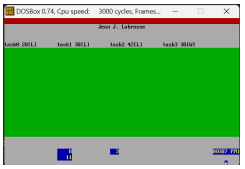
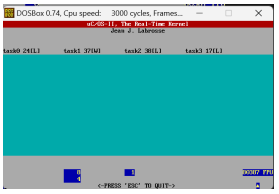
3. 이후 r_grp의 bit가 0x0F가 되어 wait state 인 4개의 random task가 ready state 가 되어

   만약 'W'이면 자신의 색을 화면에 출력한다.

```
OSTimeDlyHMSM(0, 0, 5, 0);
```

4. 수행을 마친 task는 5초동안 대기함으로써, 낮은 우선순위의 task들이 순차적으로 수행된다.

결과

5. 결과 분석

화면에 랜덤으로 4개의 색깔이 번갈아 가며 출력되는 것을 확인할 수 있다.

다만, random()을 적절히 초기화 하지 않아, 매번 같은 랜덤 수가 생성된다.