OS 세마포어 공략

- 세마포어 문제들은 대부분 순서(Order) 가 있다.
- **P : wait 와 동일 (Pause 라고 기억하라!)**
- **V : signal과 동일**

Producer-Cosumer 계열 Sem 문제들 + 식사하는 철학자

⇒ 생산하는 비율/횟수 에 따라 순서가 부여되는 문제

즉, Pipeline처럼 A =⇒ B 과정이 명확하게 이루어지며 서로의 쓰레드의 역할이 명확하게 구분되어있는 문제

▼ H2O 생성 문제

- **접근 핵심 : hydroProvider가 2개를 생산한 시점을 count로 확인 후 oxy(최종 결정자)를 호출한다. ⇒ 그리고 호출당한 OxyProvider 가 bond하고 hydrogen 2개를 재호출한다.**
- 전역변수 hydro 값에 대한 락이 필요함.
  - 실제 교수님 답에선 oxLock과 hydrolock을 반대로 쓰고있음을 주의

```
int hydro;
Semaphore lock=1;
Semaphore hydroLock=0;
Semaphore oxLock=0;

void bond() {
  static int val;
  printf("Bonding *** : %d\n", 1 + val++);
}

void OxyProvider()
{
  while(1) {
    printf("make oxygen\n");
    wait(lock);
    hydro = 0;
    signal(lock);
    wait(oxLock);
    bond();
    signal(hydrolock);
    signal(hydrolock);
  }
}
void hydroProvider()
{
```

```
  while(1) {
    wait(lock);
    printf("make hydrogen\n");
    hydro++;
    if (hydro >= 2) {
      signal(lock);
      signal(oxLock);
    }
    else
      signal(lock);
    wait(hydroLock);
  }
}
```

▼ Smoker-Provider Problem ⇒ 일단 교수님답으로..

cigSmoker.c

cigSmokerII.c

- 본인은 아래처럼 주장 중..
  - flag[?] % 3
  - lock = 0;

```
Semaphore lock=1;
Semaphore more=0;
Semaphore tmp=0;
int count = 0;
flag[0~2] = False;

void Smoker (int i)
{
  while(1) {
    wait(lock);
    if (flag[(i-1) % 2] and flag[(i+1) % 2]) {
      flag[(i-1) % 2] = flag[(i+1) % 2] = False;
      Smoke(); //담배피러감

      while(count > 0) {
        signal(lock);
        count--; //사실 이렇게하면 문제가 발생함.. (Queue에서 빠져나오기도 전에 count 가 감소되버리기때문)
      }
      signal(more);
    }
    else { //아이템이 매칭이 안되면
```

```
      count++;
      signal(lock);
      wait(tmp); //대기함.
    }
  }
}

void Provider ()
{
  while(1) {
    flag[2개의 아이템] = True;
    signal(lock);
    wait(more);
  }
}
```

Read-Write Lock 계열 Sem 문제들

⇒ 이중 교차로 문제들이 이에 속한다.

락을 한 번 획득하면 최대 N개까지 동시에 진행할 수 있고 **임계조건에서 (한 방향으로의) 과부하가 걸리지 않도록 설계해야되는 문제들**

▼ Read-Write Lock 문제

```
Semaphore lock=1, read_write=1;
Semaphore order=1;
int reader;

void reader_acqure()
{
  wait(order);
  wait(lock);
  reader++;
  if (reader == 1) {
    wait(read_write);
  }
  signal(lock);
  signal(order);
}

void reader_release()
{
  wait(lock);
  reader--;
  if (reader == 0) {
    signal(read_write);
  }
  signal(lock);
}


void writer_acquire()
```

```
{
  wait(order);  //순서독점 방지를 위한 락
  wait(read_write);
  signal(order);
}

void writer_release()
{
  signal(read_write);
}
```

▼ Bridge Monkey 문제

- 교차로문제I

```
Semaphore dest_mutex[2] = {1,1}; //1로 초기화
Semaphore rope=1, order=1;
Semaphore max_monkeys = 5; //최대 5마리원숭이 (FullBuffer느낌)
int dest_monkeys[2] = {0,0}; //각 각 현재 프로세스가 진행중인 개수

void start_wait_queue (int dest)
{
  wait (order); //A 방향으로 오다가 갑자기 B방향으로 올 때 주도권을 주기위한 세마포어
  wait (dest_mutex[dest]);
  dest_monkeys[dest]++;
  if (dest_monkeys[dest] == 1) {
    wait(rope); //Rope 획득 대기 시 oreder, 과 자신의 mutex 먹고 기다려버림!
  }
  signal(dest_mutex[dest]);
  wait(max_monkeys);
  signal(order);
}

void finsihing_Roping(int dest)
{
  wait(dest_mutex[dest]);
  dest_monkeys[dest]--;
  signal(max_monkeys);  //max_monkeys 놔줌
  if (dest_monkeys[dest] == 0) {
    signal(rope);
  }
  signal(dest_mutex[dest]);
}
```

▼ Bridge Man 문제

- 교차로에 양방향으로 사람이 왔다갔다 피해가면서 갈 수 있지만 최대 5명까지 움직일 수 있다면?

```
Semaphore dest_mutex[2] = {1,1}; //1로 초기화
Semaphore max_men = 5; //최대 5마리원숭이 (FullBuffer느낌)

void start_wait_queue (int dest)
{
  wait(dest_mutex[dest]);
  wait(max_men);
}

void finsihing_Roping(int dest)
{
  signal(max_men);
  signal(dest_mutex[dest]);
{
```

▼ East-West Bridge Problem

```
void proceeed_to_wait()
{
    P(lock); // lock is a mutex and is initializes to 1.
    if ((nw_active + nw_waiting == 0) && (nr_active < 3))
    {
        nr_active++; // Notify we are active
        V(r_sem);    // Allow ourself to get through
    }
    else
        nr_waiting++; // We are waiting
    V(lock);
    P(r_sem); // Readers will wait here, if they must wait.
}

void passing_complete() {
    P(lock);
    nr_active--;
    if ((nr_active == 0) && (nw_waiting > 0))
    {              // If we are the last reader
        count = 0; // Makes sure we only at most 3 writers in
        while ((nw_waiting > 0) && (count < 3))
        {
            V(w_sem);    // wake a writer;
            w_active++;  // one more active writer
            w_waiting--; // one less waiting writer.
            count++;
        }
    }
    else if ((nw_waiting == 0) && (nr_waiting > 0)) {
        // allow another waiting reader to go in, if no waiting writers
        V(r_sem);
        w_active++;  // one more active reader
        w_waiting--; // one less waiting reader.
    }
    V(lock);
}
```