Semaphore 보충 정리

(1) 자판기 문제 (생산자 소비자 문제의 응용)

- 20개의 슬롯이 있는 자판기가 있다고 가정하자. 생산자는 콜라를 슬롯에 넣고, 소비자는 자판기에게 콜라를 소비한다고 가정하자. 다음과 같이 세마포어 변수가 사용되고, produce(), consume()이라는 함수가 있을 때, 각 세마포어 변수의 초기 값을 쓰고, 상호배제를 보장하고 deadlock이 발생하지 않은 pseudo-code를 완성하시오.

```
/*initialize*/
#define NUM_SLOTS 20
semapore mutex =   1;   //이진. 자판기를 한 프로세스가 한순간에 한번씩 접근. 화장실 문제
semaphore fullBuffer = 0;  //범용. 빈 슬롯이 있는지 확인 세마포어
semaphore emptyBuffer = NUM_SLOTS;  // 범용. 자판기에 콜라가 있는지 확인하는 세마포어. 채울 수 있는 개수
```

```
Produce()
{
      wait(emptybuffer); //20->19->18
      wait(mutex); //1->0  본인이 들어가야 하므로
      put_1_coke in machine();
      signal(mutex); //0->1  다 했다면
      signal(fullbuffer); //0->1  full buffer 증가시켜서 빈 슬롯 증가
 }
```

```
Consume()
{
   wait(fullbuffer); //0 -> -1 대기
   wait(mutex); 본인이 들어가야 하므로
   take_1_coke_from_mahcine();
   signal(mutex);
   signal(emptybuffer); // 채울 수 있는 콜라 수 증가
 }
```

(2) 한강대교 문제 (교차로 문제)

- 한강대교가 보수 공사 중이라 여러 개의 차선 중에서 단지 하나의 차선만 자동차가 달릴 수 있다. 자동차의 충돌 사고를 막기 위해 다리 양쪽 끝에 신호등을 설치해 양 방면에서 오는 자동차들의 진입을 동기화한다. 반대 방면에서 자동차가 오지 않으면 다른 쪽 반대 방면의 자동차는 한강대교를 건널 수 있다. 또한 한강대교 다른 양쪽에 센서를 달아 자동차가 한강대교에 진입하려는지 확인할 수 있고, 자동차가 한강대교를 완전히 건넜는지 알 수 있다. 신호등은 다리 양쪽에 부착된 센서를 통해 제어된다. 보수 공사 중인 한강대교에서 자동차가 달릴 수 있도록 arrive()와 depart()를 세마포어로 슈도 코드를 작성하시오. 단, 자동차 1대는 1개의 프로세스로 표현할 수 있고, 각 프로세스는 자동차가 다리에 이르면 arrive()을 호출하고 다리를 지나가면 depart()를 호출한다

```
semaphore bridge = 1;  // 다리 세마포어, 한번에 다리에 진입할 수 있는 자동차 수를 관리. 초기값을 1. 한번에 하나의 자동차만 건널 수 있다. 비효율적이지만 기본 세마포어
semaphore mutex = 1;   // 방향 정보를 보호하는 세마포어 북쪽-> 남쪽 / 남쪽 -> 북쪽 . 방향을 위한. 양쪽 방향에서 자동차를 동시에 한강대교에 진입하는 것을 막아야 함. // 화장실 문제
int northCount = 0;    // 북쪽 대기 자동차 수 서울역 -> 우리학교. 동부이촌동에서 대기하는 자동차 수
int southCount = 0;    // 남쪽 대기 자동차 수 우리학교 -> 서울역. 상도터널 지나 대교를 건너려는 자동차 수
```

```
arriveNorth() { // 서울역 -> 우리학교
 wait(mutex);
 northcount ++;
 if (northcount ==1){
   wait(brige);
 }
 signal(mutex);
}
```

```
arriveSouth() { //우리학교 -> 서울역
 wait(mutex);
 southcount ++;
 if (southcount ==1){
   wait(brige); //다리 접근을 막는다
 }
 signal(mutex);
}
```

```
departNorth() {
 wait(mutex);
 northcount --;
 if(northcount ==0){
  signal(bridge);
 }
 signal(mutex);
}
```

```
departSouth() {
 wait(mutex);
 southcount --;
 if(southcount ==0){
  signal(bridge); //다리 접근 해지
 }
 signal(mutex); //방향 정보 해지
}
```

한쪽방향의 자동차가 다리를 무한히 기다리는가? 반대방향 차만 달린다.

대기중인 자동차만 다리를 점유. 현재 방향에서 자동차 없으면 bridge를 점유하지 않는다. 반대 방향의 차가 자유롭게 다리 건너면 된다.

(3) 한강대교 문제 응용

- 한 방향으로 최대 3대의 자동차가 동시에 다리를 건널 수 있도록 수정하려면, 즉, 다리 위에서 동시에 존재할 수 있는 자동차 수를 제한하는 새로운 세마포어와 카운터를 추가

semaphore bridge = 1; // 다리 세마포어, 한번에 다리에 진입할 수 있는 자동차 수를 관리. 초기값을 1. 한번에 하나의 자동차만 건널 수 있다. 비효율적이지만 기본 세마포어_

semaphore mutex = 1;

carcount =0 // 현재 다리에 있는 자동차 수

maxcar = 3

카운트 기반으로 접근을 접근?

자동차가 다리에 접근하려면 carcount 〈= maxcar

수정된 알고리즘

```
semaphore bridge = 1;  // 다리의 방향 보호
semaphore mutex = 1;   // carCount 보호
int carCount = 0;      // 현재 다리 위 자동차 수
#define maxCars 4      // 최대 자동차 수
```

```
arriveNorth() { // 서울역 -〉 우리학교
 wait(bridge);
 wait(mutex);
 while(carcount 〉 = maxcar) {
  signal(mutex);
  wait(mutex);
}
 carcount++;
 signal(mutex); //carcount  보호를 해제
}
```

```
arriveSouth() { //우리학교 -〉 서울역
wait(bridge); // 방향 확보. 다른 방향 접근 막고
 wait(mutex);  // carcount 보호
 while(carcount 〉 = maxcar) {
  signal(mutex);
  wait(mutex);
}
 carcount++;
 signal(mutex); //carcount  보호를 해제
}
```

```
departNorth() {
 wait(mutex);
 carcount --;
 if(carcount ==0){
  signal(bridge);
 }
 signal(mutex);
}
```

```
departSouth() {
wait(mutex);
 carcount --;
 if(carcount ==0){
  signal(bridge);
 }
 signal(mutex);
}
```

(4) H2O 문제

- 다음은 물을 생성하기 위한 범용 세마포어 변수를 이용한 2가지 구현 방법(프로그램 1와 2) 보여준다. 물(H20)을 만들기 위해 두 개의 H 원자와 하나의 O 원자가 필요하다. 각 원자는 쓰레드로 구현되고 각 H 원자는 반응할 준비가 되면 hReady 함수를 호출한다. 각 O 원자가 준비가 되면 oReady 함수를 호출한다. 최소한 2개 H 원자와 1개 O 원자가 나타날때까지 두 함수는 지연(delay)된다. 2개의 H 원자와 하나의 O 원자가 나타나면 두 쓰레드 중 하나가 makeWater 함수(물이 만들어졌음을 메시지로 프린트하는)를 호출한다. makeWater 함수가 호출되면, hReady 함수의 2개 인스턴스와 oReady 함수의 인스턴스를 리턴한다. 기아나, 바쁜 대기(Busy Waiting)이 없어야 하며, FIFO 방식(SemWait() 함수에서 가장 오래 기다린 쓰레스는 항상 SemSignal() 함수 호출로 깨어 난 다음 쓰레드이어야 함)으로 작동해야 한다. 다음 프로그램 1과 2가 제대로 동작하는지 확인하고 동작하지 않으면 이유를 설명하시오

```
// 프로그램 1
semaphore h_wait=0;
semaphore o_wait=0;
```

```
hReady()
{
    count ++; //H의 개수를 추적. 전역변수 ???
    if(count %2 ==1) { // 홀수번째 H원자는
            SemWait(h_wait);
 }
  else { //짝수번째 h원자
            SemSignal(o_wait);
            SemWait(h_wait);
  }
 return;
}
```

```
oReady()
{
 SemWait(o_wait); //
 makeWater();
 SemSignal(h_wait);
 SemSignal(h_wait);
  return;
}
```

H2O를 생성하는 조건 : 2h, 1o
지연관리 : 필요한 수의 h와 0의 원자가 준비되지 않으면 hready, oready()가 대기
FIFO : 대기중인 쓰레드가 FIFO 순서대로 처리
두 개의 H원자가 동시에 count 변수를 접근하면???

```
// 프로그램 2
semaphore h_wait=0;
semaphore o_wait=0;
```

```
hReady()
{
      SemSignal(o_wait); //O 원자를 깨움 1-〉 2
      SemWait(h_wait); // H 원자가 다시 생성할 수 있도록 만들어준다.

 return;
}
```

```
oReady()
{
 SemWait(o_wait);
 SemWait(o_wait);
 makeWater();
 SemSignal(h_wait);
 SemSignal(h_wait);
 return;
}
```

```
// 프로그램 3
semaphore h_wait = 0;       // H 원자 대기 세마포어
semaphore o_wait = 0;       // O 원자 대기 세마포어
semaphore mutex = 1;        // 공유 변수 보호

int h_count = 0;            // 현재 대기 중인 H 원자 수
int o_count = 0;            // 현재 대기 중인 O 원자 수
```

```
hReady() {
    SemWait(mutex);          // 공유 변수 보호 시작
    h_count++;               // H 원자 수 증가

    if (h_count >= 2 && o_count >= 1) {
        // 물을 만들 수 있는 조건이 만족되면 O와 H 원자 대기 해제
        SemSignal(o_wait);  // O 원자 대기 해제
        SemSignal(h_wait);  // 첫 번째 H 원자 대기 해제
        SemSignal(h_wait);  // 두 번째 H 원자 대기 해제

        h_count -= 2;        // 사용된 H 원자 수 감소
        o_count--;           // 사용된 O 원자 수 감소
    }
    SemSignal(mutex);        // 공유 변수 보호 해제

    SemWait(h_wait);         // 물 생성 전 대기
    return;
}
```

```
oReady() {
    SemWait(mutex);          // 공유 변수 보호 시작
    o_count++;               // O 원자 수 증가

    if (h_count >= 2 && o_count >= 1) {
        // 물을 만들 수 있는 조건이 만족되면 O와 H 원자 대기 해제
        SemSignal(o_wait);  // O 원자 대기 해제
        SemSignal(h_wait);  // 첫 번째 H 원자 대기 해제
        SemSignal(h_wait);  // 두 번째 H 원자 대기 해제

        h_count -= 2;        // 사용된 H 원자 수 감소
        o_count--;           // 사용된 O 원자 수 감소
    }
    SemSignal(mutex);        // 공유 변수 보호 해제

    SemWait(o_wait);         // 물 생성 전 대기
    makeWater();             // 물 생성
    return;
}
```