0. 목표

- N개의 process와 M개의 resource type을 갖는 시스템에서 Deadlock detection 기법 구현

(Multiple Resource types & Multiple Resource units)

1. 입력과 출력

- 입력은 input.txt 파일로 주어진다.

- 프로세스의 개수 N, Resource type의 개수 M, 각 Resource type의 unit 개수 그리고 allocation matrix와 request matrix가 입력으로 주어진다.

- 출력은 Deadlock 상태인지 아닌지, Deadlock 상태라면 Deadlocked process list를 출력한다.

- 출력 양식은 자유이다.

2. 구현에 앞선 가정

- 입력으로 주어지는 Process의 개수, Resource type의 개수는 최대 10000개로 제한한다.

3. 구현

```
FILE* f;
if ((f = fopen("input.txt", "r")) == NULL) {
    printf("FILE READ ERROR₩n");
    return 0;
}

fscanf(f, "%d", &ProcessNum); // Process 개수
fscanf(f, "%d", &Rtype); // Resource 개수
for (i = 0;i < Rtype;i++) {
    fscanf(f, "%d", &Runit[i]); // Resource i의 unit 개수
}

for (i = 0;i < ProcessNum;i++) {
    p[i].ok = 0;
    for (j = 0;j < Rtype;j++) {
        fscanf(f, "%d", &p[i].allocated[j]);
        Runit[j] = Runit[j] - p[i].allocated[j]; // 남은 Resource unit을 구함
    }
}

for (i = 0;i < ProcessNum;i++) {
    for (j = 0;j < Rtype;j++) {
        fscanf(f, "%d", &p[i].request[j]);
    }
}
```

Input.txt 파일을 받아와 해당 내용을 입력으로 받아 process의 정보를 입력한다.

Process 구조체의 allocated와 request 배열의 인덱스는 resource type의 번호를 의미한다.

Runit 배열은 현재 남아있는, 즉 사용가능한 unit의 개수를 나타내고 인덱스가 resource type의 번호를 의미한다.

```
void check() {
    int i, j;
    int flag = 1;
    while (flag) {
        flag = 0;
        for (i = 0;i < ProcessNum;i++) {
            if (!p[i].ok) { // reduction 되었는지 아닌지 판단
                int safe = 1; // 기본값으로 safe 하다고 가정
                for (j = 0;j < Rtype;j++) {
                    if (p[i].request[j] > Runit[j]) { // 사용가능한 Resource unit 보다 적게 request 해야함
                        safe = 0; break; // 해당 process는 reduction 될 수 없음
                    }
                }
                if (safe) { // request하는 resource unit이 모두 사용가능한 상태
                    for (j = 0;j < Rtype;j++) {
                        Runit[j] = Runit[j] + p[i].allocated[j]; //Reduction하면서 allocated 했던 unit을 반납
                        flag = 1; p[i].ok = 1; // reduction 처리
                    }
                }
            }
        }
    }
}
```

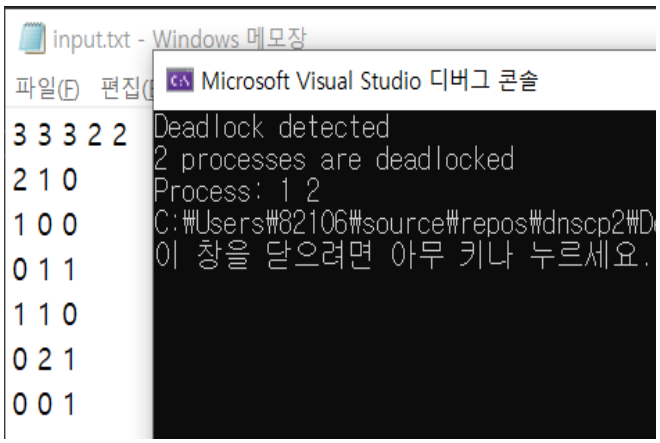
모든 process를 탐색해서 요구하는 resource들이 사용가능한 상태, safe한 상태라면 할당되었던 resource들을 반납하고 reduction 처리를 해준다. Reduction 처리를 할 수 없는 상태까지 while loop를 돌고 모든 process가 reduction 되었거나, 더 이상 reduction할 process가 없다면 check 함수를 나오게 된다.

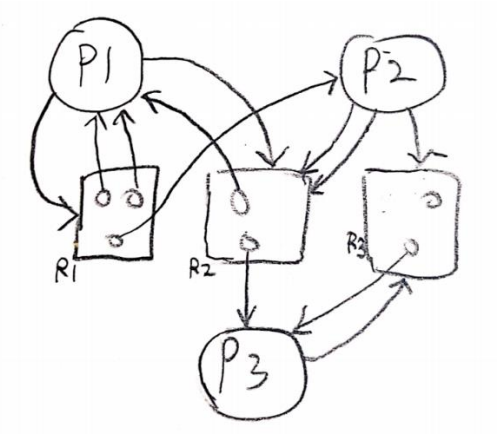
```
check();

int target[10000], k=0; // target: deadlocked process list, k: num of target
for (i = 0;i < ProcessNum;i++) {
    if (p[i].ok == 0) target[k++] = i + 1; // ok 값은 deadlock인지 아닌지 확인
}
if (k == 0) printf("No Deadlock₩n");
else {
    printf("Deadlock detected₩n");
    printf("%d processes are deadlocked₩n", k);
    printf("Process: ");
    for (i = 0;i < k;i++) printf("%d ", target[i]);
}
return 0;
```

Process들의 ok값으로 해당 process가 reduction되었는지 아닌지 판단하여 reduction 되지 않은 process들, 즉 deadlock 상태에 놓여있는 process들을 출력해준다.

4. 예제 테스트

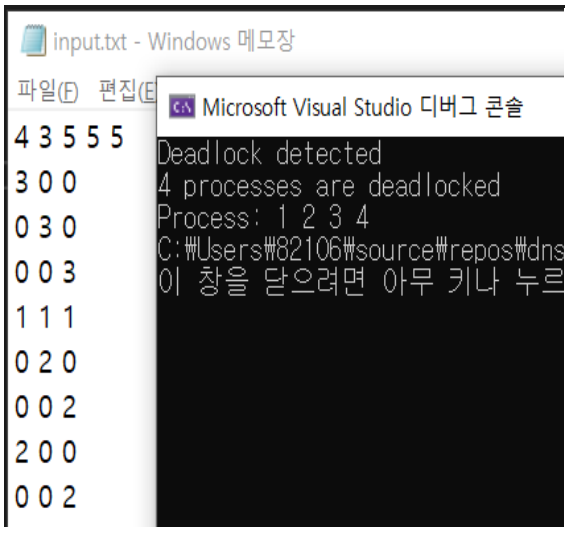
1) 일부 process가 deadlock 상태에 있는 경우

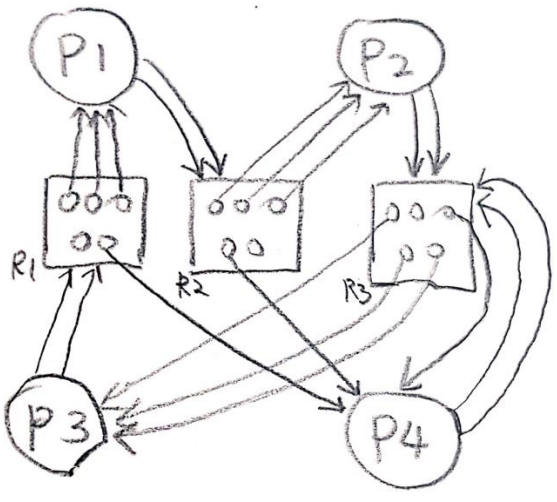




P3이 R3에서 unit 하나를 할당 받아 reduction 되고 나면 resource unit은 (0, 1, 2)만큼 남게 되고, P1은 (1, 1, 0)을 요구하고, P2는 (0, 2, 1)을 요구하기 때문에 현재 상태에서 더 이상 reduction이 불가능하다. 따라서 P1과 P2는 deadlock 상태에 놓여있다.

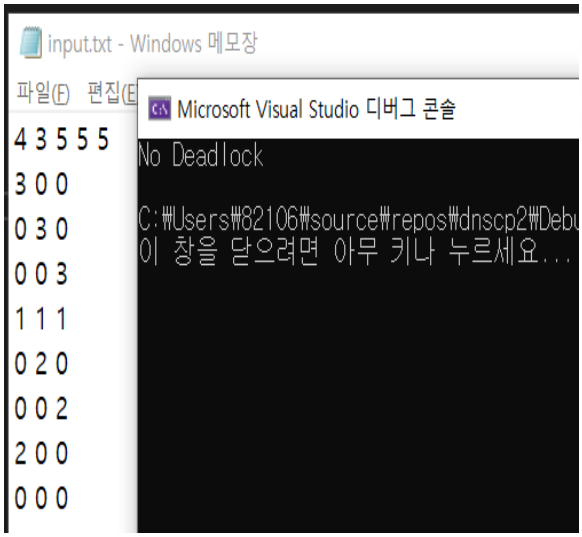
2) 모든 process가 deadlock 상태에 있는 경우

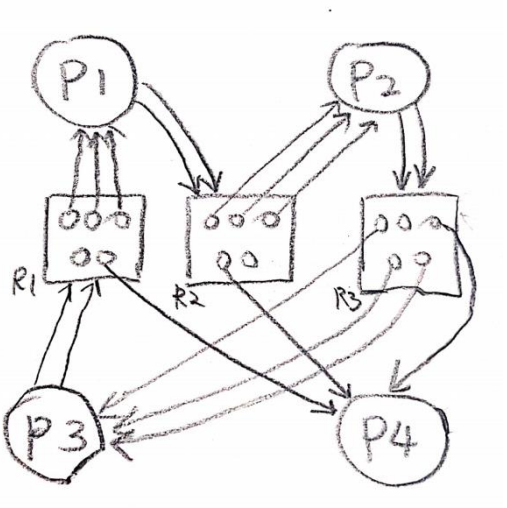




이 상태에서 남아 있는 unit은 (1, 1, 1)이다. 모든 process가 요구하는 unit이 2이상 이기 때문에 Reduction 가능한 process가 없다. 즉 P1, P2, P3, P4 모든 process가 deadlock 상태에 놓여있다.

3) Deadlock이 없는 경우





이 상태에서 남아 있는 unit은 (1, 1, 1)이다. P4가 요구하는 unit이 없기 때문에 즉시 reduction 가능하고 할당 받았던 unit을 반납함으로써 사용가능한 unit은 (2, 2, 2)가 된다. 이 후 P1이 R2에서 unit 2개를 할당 받아 reduction되면 사용가능한 unit은 (5, 2, 2)가 되고, P2가 R3에서 unit 2개를 할당 받아 reduction되면 사용가능한 unit은 (5, 5, 2)가 된다. 마지막으로 P3이 R1에서 unit 2개를 할당 받아 reduction되면 사용가능한 unit은 (5, 5, 5)가 되고, 모든 process가 reduction되었기 때문에 위 상태는 Deadlock이 없다.