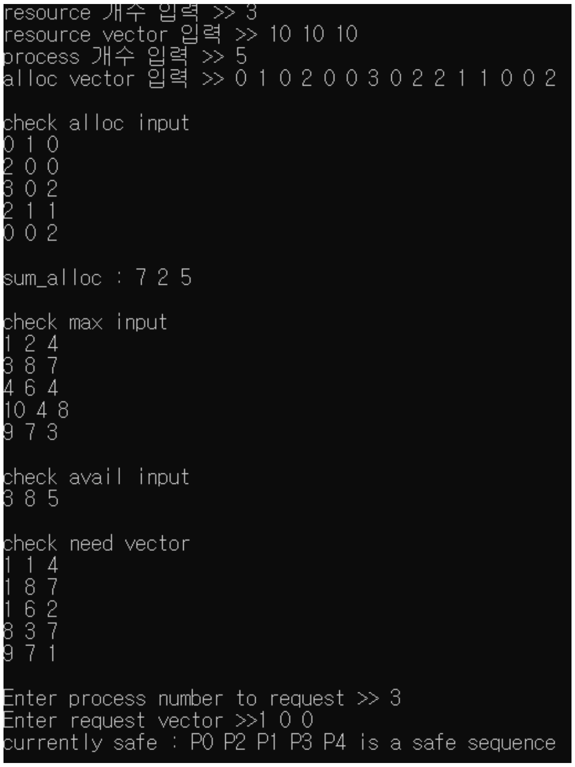
실행결과

1) case 1 : find sequence



```
resource 개수 입력 >> 3
resource vector 입력 >> 10 10 10
process 개수 입력 >> 5
alloc vector 입력 >> 0 1 0 2 0 0 3 0 2 2 1 1 0 0 2

check alloc input
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2

sum_alloc : 7 2 5

check max input
1 2 4
3 8 7
4 6 4
10 4 8
9 7 3

check avail input
3 8 5

check need vector
1 1 4
1 8 7
1 6 2
8 3 7
9 7 1

Enter process number to request >> 3
Enter request vector >>1 0 0
currently safe : P0 P2 P1 P3 P4 is a safe sequence
```

(1) Check request$_i$ <= need$_i$
(1,0,0) <= (8,3,7) -> true -> go to next step

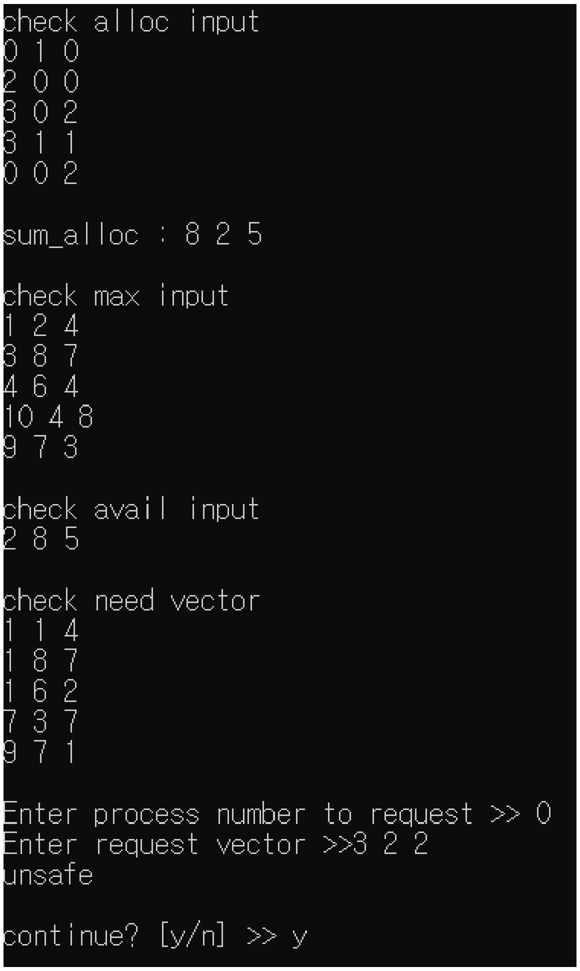
(2) Check request$_i$ <= avail
(1,0,0) (3,8,5) -> true -> go to next step

(3) pretend need$_i$ as (7,3,7), avail as (2,8,5)

(4) safety check (work := avail)
i) Need0 : (1,1,4) <= work : (2,8,5) -> true -> update work to (2,9,5)
ii) Need1 : (1,8,7) <= work : (2,9,5) -> false
iii) Need2 : (1,6,2) <= work : (2,9,5) -> true -> update work to (5,9,7)
iv) Need3 : (7,3,7) <= work : (5,9,7) -> false
v) Need4 : (9,7,1) <= work : (5,9,7) -> false
vi) Need1 : (1,8,7) <= work : (5,8,7) -> true -> update work to (7,9,7)
vii) Need3 : (7,3,7) <= work : (7,9,7) -> true -> update work to (9,10,8)
viii) Need4 : (9,7,1) <= work : (9,10,8) -> true -> update work to (9,10,10)

(5) So, safety sequence is “P0 -> P2 -> P1 -> P3 -> P4”

2) **case 2 : unsafe**

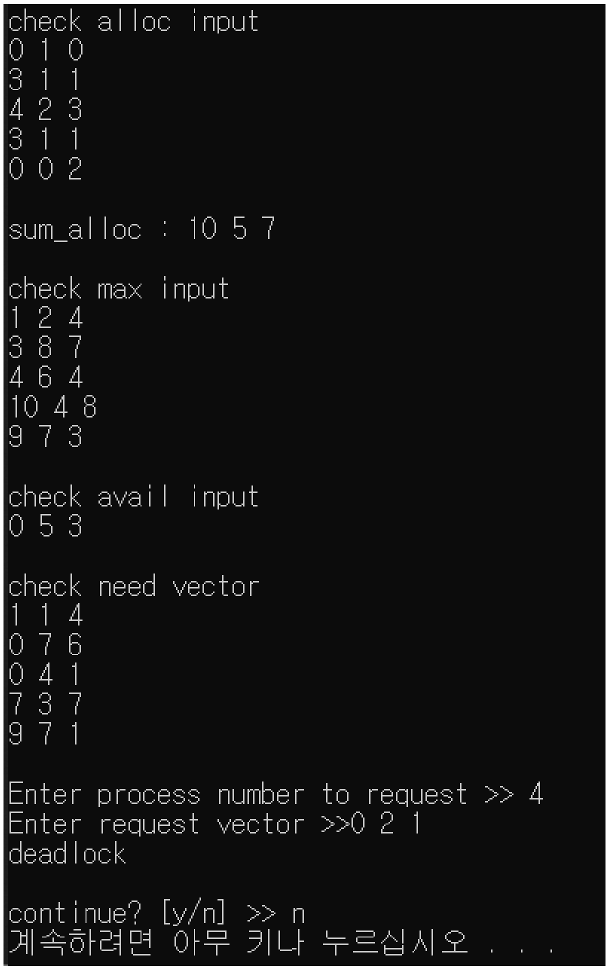


```
check alloc input
0 1 0
2 0 0
3 0 2
3 1 1
0 0 2

sum_alloc : 8 2 5

check max input
1 2 4
3 8 7
4 6 4
10 4 8
9 7 3

check avail input
2 8 5

check need vector
1 1 4
1 8 7
1 6 2
7 3 7
9 7 1

Enter process number to request >> 0
Enter request vector >>3 2 2
unsafe

continue? [y/n] >> y
```

(1) Check request $_i$ <= need $_i$
(3,2,2) <= (1,1,4) -> false -> unsafe

(2) So, request vector (3,2,2) cannot allocate to P0.

3) case 3 : deadlock



```
check alloc input
0 1 0
3 1 1
4 2 3
3 1 1
0 0 2

sum_alloc : 10 5 7

check max input
1 2 4
3 8 7
4 6 4
10 4 8
9 7 3

check avail input
0 5 3

check need vector
1 1 4
0 7 6
0 4 1
7 3 7
9 7 1

Enter process number to request >> 4
Enter request vector >>0 2 1
deadlock

continue? [y/n] >> n
계속하려면 아무 키나 누르십시오 . . .
```

(1) Check request $_i$ <= need $_i$

 (0,2,1) <= (9,7,1) -> true -> go to next step

(2) Check request $_i$ <= avail

 (0,2,1) (0,5,3) -> true -> go to next step

(3) pretend need $_i$ as (9,5,0), avail as (0,3,2)

(4) safety check (work := avail)

 ix) Need0 : (1,1,4) <= work : (0,3,2) -> false

 x) Need1 : (0,7,6) <= work : (0,3,2) -> false

 xi) Need2 : (0,4,1) <= work : (0,3,2) -> false

 xii) Need3 : (7,3,7) <= work : (0,3,2) -> false

 xiii) Need4 : (9,7,1) <= work : (0,3,2) -> false

(5) So, deadlock occurs.

코드 설명

1) init vector, sum_alloc vector, avail vector, pretend_avail vector, req vector 는 1 차원 벡터를 이용하여 생성한다.
2) alloc vector, max vector, need vector 는 2 차원 벡터를 이용하여 생성한다.
3) 이때, max vector 는 랜덤으로 받되 init vector 의 원소보다 작게, alloc vector 의 원소 크게 생성한다.
4) avail vector 에 직접 pretend 하면 deadlock 이 걸렸을시에 Safety Check 을 하면서 할당했던 값을 돌려주기 다소 복잡하다. 따라서 pretend_avail vector 를 하나 더 생성하고 모든 계산은 pretend_avail vector 에 한 후,
   (1) deadlock 일 경우 alloc, sum_alloc, need vector 의 값을 원래대로 되돌리고,
   (2) deadlock 이 아닐 경우 pretend_avail vector 의 값들을 avail vector 에 옮겨준다.