

Contents

1. Introduction	3
A. Purpose	3
B. Problem Statement	3
C. Solving Strategy	3
2. Body	5
A. Implementation	5
B. Input Test	8
RANDOM_INPUT = 0일 때	8
RANDOM_INPUT = 1일 때	12
C. Contents of Files	15
3. Conclusion	15

Figure Contents

Figure 1 input.txt 양식	3
Figure 2 unblocked process의 조건.....	4
Figure 3 입력받은 data가 리소스 종류와 일치하지 않을 때	6
Figure 4 Figure 3 결과(에러 메시지 출력하며 종료).....	6
Figure 5 할당받은 리소스 개수가 총 유닛 개수보다 많을 때	6
Figure 6 Figure 5 결과(에러 메시지와 함께 종료).....	7
Figure 7 input1.txt	9
Figure 8 input1.txt result.....	9
Figure 9 input2.txt	9
Figure 10 input2.txt result	10
Figure 11 input3.txt	11
Figure 12 input3.txt result	11
Figure 13 random input 1	12
Figure 14 random input1 result.....	12
Figure 15 random input 2	13
Figure 16 random input 2 result.....	13
Figure 17 random input 3	14
Figure 18 random input 3result.....	14
Figure 19 random input 4	14
Figure 20 random input 4 result.....	14

1. Introduction

A. Purpose

Deadlock detection 기법을 C 언어를 이용하여 구현한다.

B. Problem Statement

1. 프로세스 개수(N), 리소스 개수(M)와 리소스 유닛 개수는 input.txt로 주어진다. input.txt의 형태는 Figure 1과 같다.

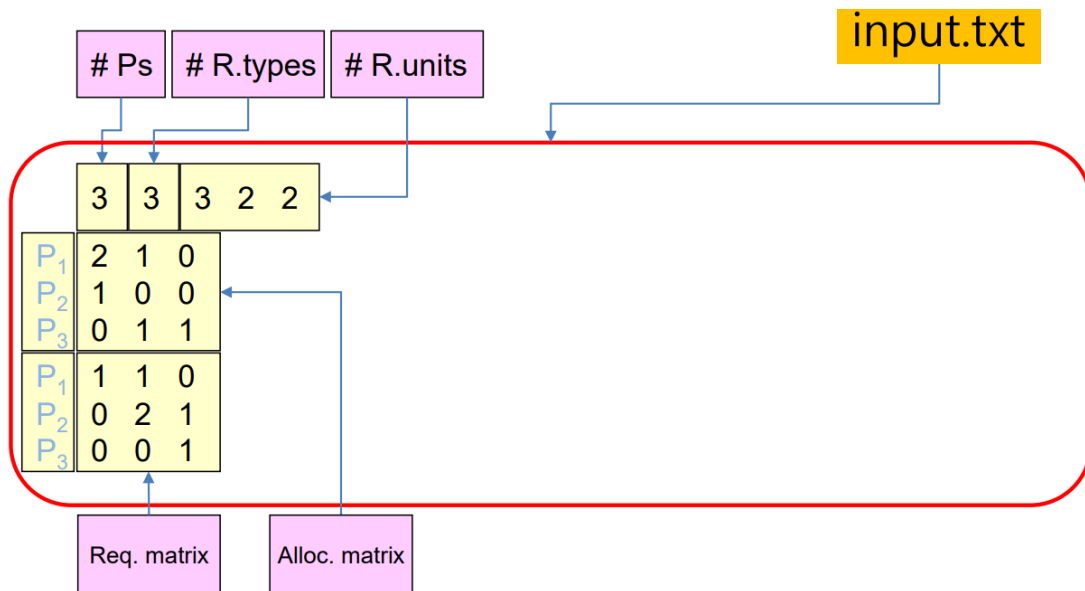


Figure 1 input.txt 양식

2. 프로세스가 할당 받은 리소스 개수, 요구하는 리소스 개수 또한 N x M 행렬로 주어진다.
3. 주어진 정보를 통해 Deadlock 상태인 프로세스의 list를 출력한다.
4. 또한 현재 상태가 Deadlock 상태인지 출력한다.

C. Solving Strategy

1. input.txt에서 프로세스 개수, 리소스 개수와 리소스 유닛 개수를 읽어온다.
2. 프로세스 개수 x 리소스 개수 크기의 matrix – allocate, request-를 만든다.
3. 만들어진 2개의 행렬에 알맞은 데이터를 각각 input.txt 에서 읽어와 저장한다.

Definition: unblocked process

□ The process P_i is unblocked if it satisfies

$$\forall j (|(P_i, R_j)| \leq t_j - \sum_{\text{all } k} |(R_j, P_k)|)$$

Figure 2 unblocked process의 조건ⁱⁱ

4. 프로세스가 Figure 2의 조건, 즉 프로세스가 요구하는 모든 리소스의 리소스 유닛의 개수가 남아있는 리소스 유닛보다 작거나 같아야 그 프로세스는 unblocked이고 deadlock이 아니라고 말할 수 있다.
5. finish라는 어레이를 이용하여 리소스를 할당 받고 끝이 난, 즉 unblocked 프로세스를 표시한다.
6. 아직 끝나지 않은 프로세스(finish에 표시가 되어 있지 않은 프로세스)에 한해 4번의 조건을 부합하는지 확인한다.
7. 만약 조건에 부합하면 finish에 표시를 하고 할당 받았던 리소스를 반납한다.
8. 만약 조건에 부합하는 프로세스가 있었다면 change를 1로 변경한다.
9. 조건에 부합하는 프로세스가 발견되면 다시 finish가 표시되지 않은 첫 프로세스부터 조건에 부합하는지 확인한다.
10. 만약 모든 프로세스를 확인했는데 change가 1이 아니라면, 더 이상 unblocked process가 없이 남아 있는 프로세스는 모두 deadlock 프로세스이거나 모든 프로세스가 끝이 났다는 뜻이므로 detection을 마친다.
11. detection을 마치고 finish에 1이 표시되지 않은 프로세스는 deadlock process list에 출력한다.
12. 만약 모든 프로세스의 finish가 1이라면 blocked 프로세스가 없다는 뜻이므로 시스템은 deadlock 상태가 아니다.
13. 만약 모든 프로세스의 finish가 1이진 않다면 blocked 프로세스가 1개 이상 있다는 뜻이므로 시스템은 deadlock 상태이다.

2. Body

A. Implementation

1. 두 가지 입력 파일 모드가 있다.

default는 RANDOM_INPUT 1이고 RANDOM_INPUT 0일 때는 input1.txt가 입력 파일로 사용된다. 이는 사용자가 언제든지 바꿀 수 있다.

- A. 직접 작성한 입력 파일을 사용하는 모드 (RANDOM_INPUT 0)
- B. 난수를 생성하여 입력 파일을 만들어 사용하는 모드 (RANDOM_INPUT 1)

RAND_RANGE: 프로세스와 리소스 개수의 범위 (0 ~ RAND_RANGE)

UNIT_RANGE: 리소스 유닛의 개수의 범위(0 ~ UNIT_RANGE)

임의로 정한 할당된 유닛 개수의 총 합은 리소스 유닛의 총 개수를 넘지 않는다.

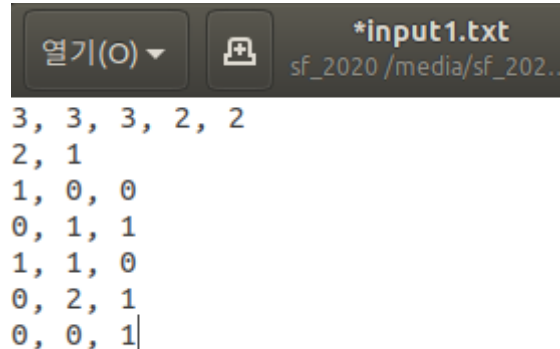
request의 범위 자체는 리소스 유닛의 총 개수(total: 리소스 j의 시스템 내 총 유닛 개수)를 넘을 수는 없으나 request(r_{ij} 프로세스 i의 리소스 j request)의 총 합이 리소스 유닛의 총 개수를 넘을 수는 있다. 그러나 한 프로세스의 각 유닛 당 request와 allocate는(a_{ij} 프로세스 i의 리소스 j allocate 받은 개수) 총 리소스 유닛의 개수를 넘지 않는다. 즉,

$$r_{ij} \leq total, \quad a_{ij} \leq total, \quad \sum_{i=1}^n a_{ij} \leq total, \quad (r_{ij} + a_{ij}) \leq total$$

를 모두 만족한다.

2. 입력 모드에 상관없이 R_info를 통해 입력 파일의 첫 줄을 읽어오고 리소스 유닛의 개수를 저장한 resource pointer array을 반환 받는다.
 - A. 만약 resource 유닛의 개수 종류가 명시된 리소스 유닛의 종류보다 작으면 에러 메시지를 출력하고 -3을 반환하며 종료한다.
 - B. R_info에서 resource라는 pointer array에 동적 메모리 할당을 진행하고 이때 메모리 할당을 실패하면 에러 메시지를 출력하고 -1을 반환하며 종료한다.
3. R_info에서 읽은 프로세스 개수 x 리소스 유닛 종류의 크기를 가지는 행렬을 create_matrix를 이용하여 만든다. (allocate, request 총 2개의 행렬 만듦) 이 때 matrix는 동적 메모리 할당을 받고 실패 시 -1을 반환하며 종료한다. 만들어진 matrix는 0으로 초기화된다.

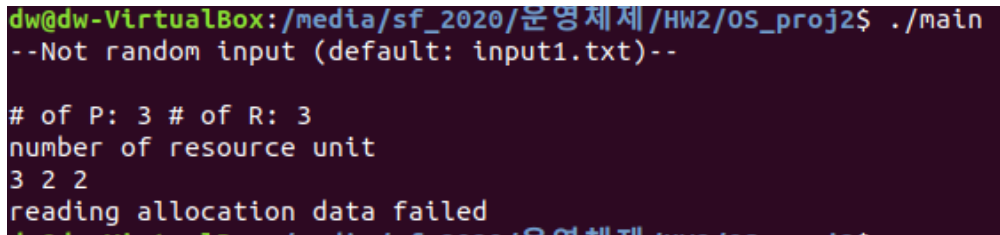
4. get_input을 통해 allocate, request 정보를 각각의 행렬에 저장한다.
- A. 만약 읽어 들인 정보가 명시된 유닛 종류보다 작으면 에러 메시지를 출력하고 -3을 반환하며 프로그램이 종료된다.



```
*input1.txt
sf_2020 /media/sf_202...

3, 3, 3, 2, 2
2, 1
1, 0, 0
0, 1, 1
1, 1, 0
0, 2, 1
0, 0, 1
```

Figure 3 입력받은 data가 리소스 종류와 일치하지 않을 때

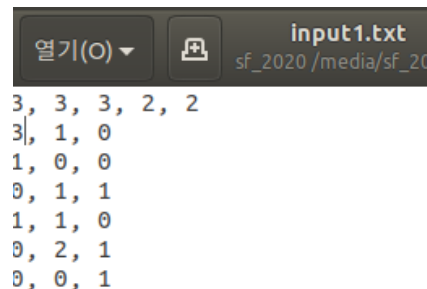


```
dw@dw-VirtualBox:/media/sf_2020/운영체제/HW2/OS_proj2$ ./main
--Not random input (default: input1.txt)--

# of P: 3 # of R: 3
number of resource unit
3 2 2
reading allocation data failed
-3
```

Figure 4 Figure 3 결과(에러 메시지 출력하며 종료)

5. get_R_remain을 이용하여 남은 리소스 유닛 개수를 계산하고 만약 입력 받은 할당된 리소스 유닛의 개수가 총 리소스 유닛의 개수보다 크면 에러 메시지를 출력하고 -4를 return하며 프로그램이 종료된다.



```
input1.txt
sf_2020 /media/sf_20...

3, 3, 3, 2, 2
3, 1, 0
1, 0, 0
0, 1, 1
1, 1, 0
0, 2, 1
0, 0, 1
```

Figure 5 할당받은 리소스 개수가 총 유닛 개수보다 많을 때

```

dw@dw-VirtualBox:/media/sf_2020/운영체제/HW2/OS_proj2$ ./main
--Not random input (default: input1.txt)--

# of P: 3 # of R: 3
number of resource unit
3 2 2
request
1 1 0
0 2 1
0 0 1

allocate
3 1 0
1 0 0
0 1 1

Available resource
cannot allocate more than total unit number

```

Figure 6 Figure 5 결과(에러 메시지와 함께 종료)

6. 입력 받은 행렬을 출력한다.
7. detection에서 실제적인 deadlock detection이 진행된다.
 - A. 모든 프로세스를 상대로 Figure 2의 조건을 만족하는지 확인한다. 만약 만족하지 않는 resource가 하나라도 있다면 impossible을 1로 변경하고 모든 resource에 대해 만족한다면 impossible을 0으로 유지한다.
 - B. 모든 리소스 종류에 대해 검사가 끝나고 impossible이 0이면 이 프로세스는 unblocked state라는 뜻이므로 이 프로세스의 finish를 1로 변경하고 이 프로세스의 request와 allocate는 0으로 변경하고, allocate 되어 있던 리소스 유닛을 반납하므로 리소스 유닛을 r_unit에 더해준다. 이렇게 finish가 1이 된 프로세스는 이미 terminate된 프로세스로 간주하며 더 이상 검사의 대상이 아니다.
 - C. 만약 finish가 1로 변한 프로세스가 있으면 change를 1로 변경한다.
 - D. 만약 change가 1이면 프로세스 0부터 다시 검사를 시작한다. 이 때 검사의 대상은 finish가 0인 프로세스이다.
 - E. 이 행위를 계속 반복하되 만약 change가 0이면 (모든 프로세스의 finish가 1이기 때문이거나 남아있는 프로세스가 모두 blocked 상태이면 더 이상 검사를 진행할 필요가 없기 때문에) 그만한다.
8. 검사가 끝나고 allocate과 request 행렬을 다시 출력한다. 이 때 finish된 프로세스는 request와 allocate이 모두 0이다.
9. find_deadlock을 통해 deadlock detection의 결과를 출력한다.
 - A. finish가 0인 프로세스는 blocked 상태 즉 deadlock 되어 있는 상태이므로 deadlock 프로세스 리스트에 출력한다.
 - B. finish가 0인 프로세스가 1개 이상 있다는 것은 시스템에 deadlock 프

로세스가 1개 이상 있다는 뜻이므로 시스템은 현재 deadlock state이다.

C. 만약 모든 프로세스의 finish가 1이면 프로세스가 모두 unblocked 이므로 deadlock state가 아니다.

10. free_matrix, free를 이용하여 동적 할당된 메모리를 반납한다.

11. 파일을 닫는다.

12. return 값과 상태

Return	상태
0	정상 종료
-1	동적할당 실패
-2	파일 열기 실패
-3	리소스 유닛 가짓수와 파일에 명시된 리소스 유닛 가짓수가 일치하지 않음 (명시된 리소스 유닛 종류보다 적은 data를 제공함)
-4	존재하는 리소스 유닛 개수보다 많은 유닛 할당

Table 1 return값과 상태

B. Input Test

RANDOM_INPUT = 0일 때

A. input1.txt

input1.txt

```

3, 3, 3, 2, 2
2, 1, 0
1, 0, 0
0, 1, 1
1, 1, 0
0, 2, 1
0, 0, 1

```

Figure 7 input1.txt

```

# of P: 3 # of R: 3
number of resource unit
3 2 2
request
1 1 0
0 2 1
0 0 1

allocate
2 1 0
1 0 0
0 1 1

Available resource
0 0 1

result: final allocation
2 1 0
1 0 0
0 0 0

result: remain request
1 1 0
0 2 1
0 0 0

Deadlocked process:
P0 P1
The system is in Deadlock state

```

Figure 8 input1.txt result

과제 파일에서 주어진 input이다.

B. input2.txt

input2.txt

```

2, 2, 3, 2
2, 0
1, 1
0, 1
1, 1

```

```
# of P: 2 # of R: 2
number of resource unit
3 2
request
0 1
1 1

allocate
2 0
1 1

Available resource
0 1


result: final allocation
0 0
1 1

result: remain request
0 0
1 1

Deadlocked process:
P1
The system is in Deadlock state
```

Figure 10 input2.txt result

C. input3.txt

열기(O) ▾  input3.txt
sf_2020 /media/sf_2020

```
5, 3, 10, 5, 7
0, 1, 0
2, 0, 0
3, 0, 2
2, 1, 1
0, 0, 2
7, 4, 3
1, 2, 2
6, 0, 0
0, 1, 1
4, 3, 1
```

Figure 11 input3.txt

```
# of P: 5 # of R: 3
number of resource unit
10 5 7
request
7 4 3
1 2 2
6 0 0
0 1 1
4 3 1

allocate
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2

Available resource
3 3 2

result: final allocation
0 0 0
0 0 0
0 0 0
0 0 0
0 0 0

result: remain request
0 0 0
0 0 0
0 0 0
0 0 0
0 0 0

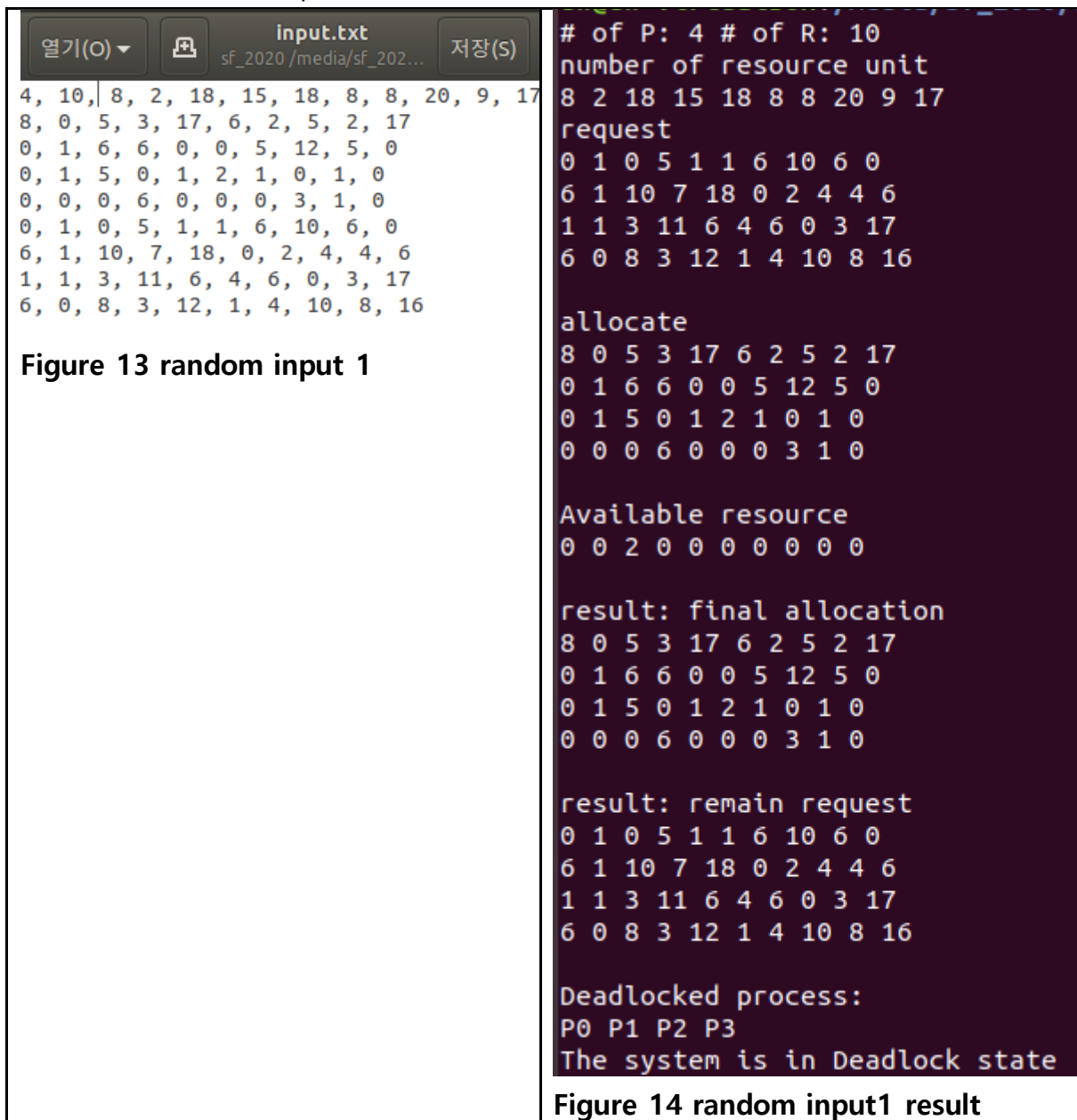
Deadlocked process:
none
Not in Deadlock state
```

Figure 12 input3.txt result

수업시간에 deadlock avoidance 기법에서 다룬 예제를 deadlock detection 기법을 이용하여 교착상태를 검출해봤다. worst case를 다루는 avoidance 기법에서도 safe state였기 때문에 best case를 다루는 deadlock detection에서는 deadlock state가 아닌 것이 합당하며 결과 또한 그렇게 나왔다.

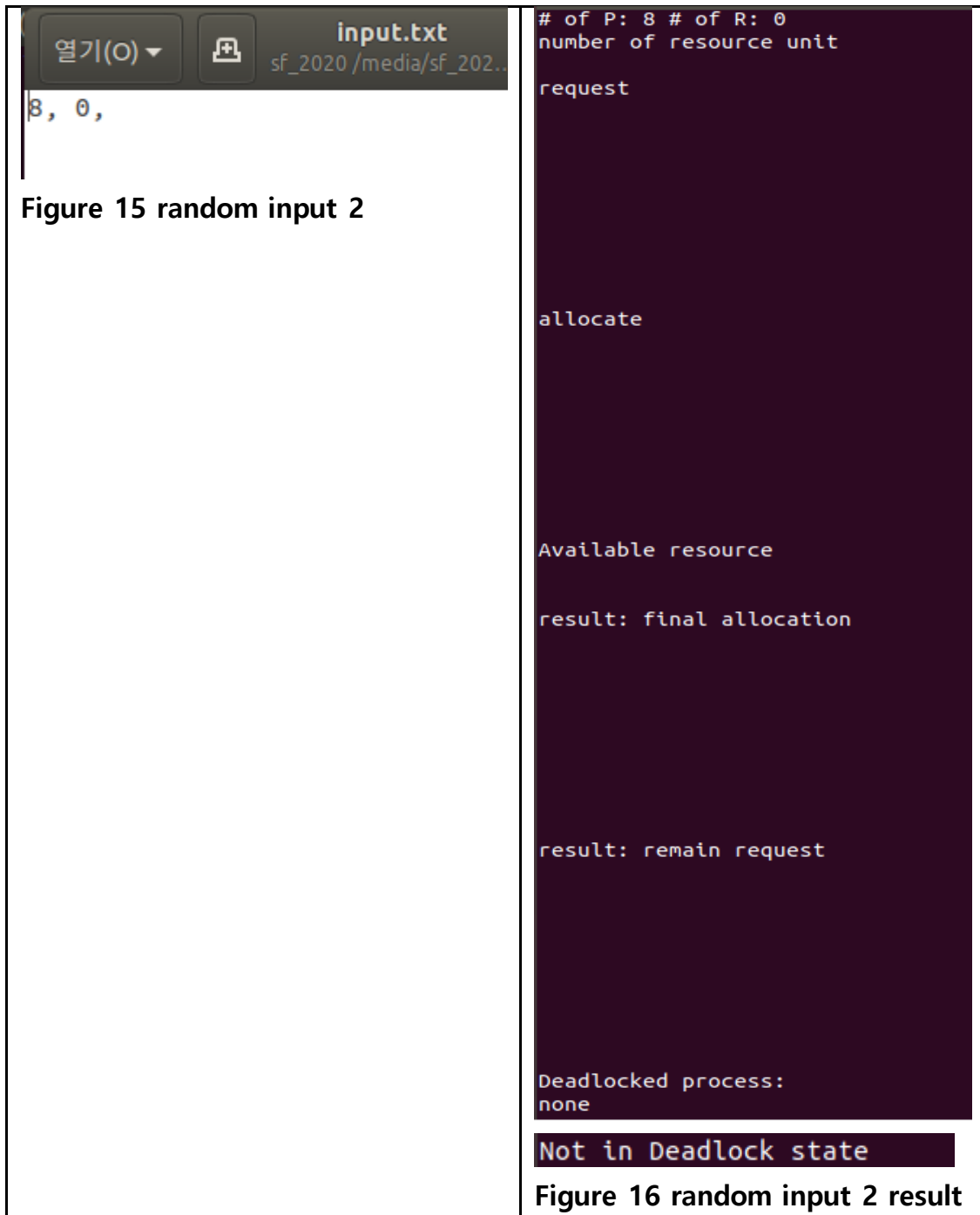
RANDOM_INPUT = 1일 때

A. random input



앞서 언급된 대로 allocate된 resource의 개수의 합은 총 resource 개수를 넘지 않으며 할당된 resource 유닛 개수 + 요구하는 resource 유닛 개수는 총 resource 유닛 개수를 넘지 않는다. 컴퓨터 시스템 내의 총 resource 개수보다 더 많은 자원을 요구하는 것은 합당하지 않다고 판단하였기 때문에 이와 같은 제약을 두었다.

B. resource 종류 0개



총 리소스 유닛 개수를 초과하는 요구나 할당을 받을 수 없기 때문에 리소스가 0개일 때 deadlock state가 아니다. 이는 시스템 내에 존재하지 않는 리소스를 요구하는 상황으로 해석될 수 있고 이 때문에 deadlock state가 되는 것은 합당하지 않다고 생각하였기 때문에 리소스 유닛이 0개일 때는 교착상태가 아니라는 결과가 나오는 것이 적절하다고 판단하였다.

C. 프로세스 개수 0개

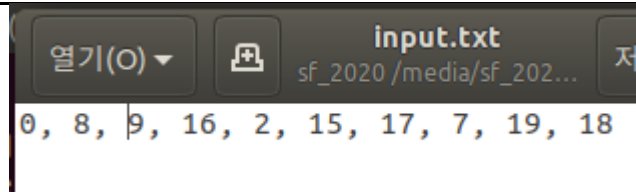


Figure 17 random input 3

```
# of P: 0 # of R: 8
number of resource unit
9 16 2 15 17 7 19 18
request

allocate

Available resource
9 16 2 15 17 7 19 18

result: final allocation

result: remain request

Deadlocked process:
none
Not in Deadlock state
```

Figure 18 random input 3 result

프로세스 개수가 0개일 때는 당연히 deadlock 프로세스가 있을 수 없으므로 deadlock state가 아니다.

D. 프로세스 개수 1개

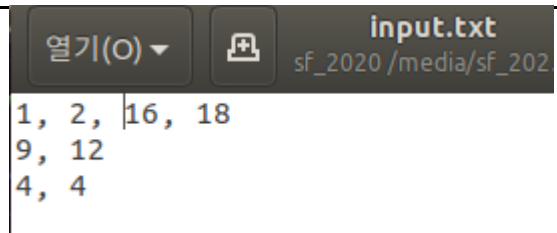


Figure 19 random input 4

```
# of P: 1 # of R: 2
number of resource unit
16 18
request
4 4

allocate
9 12

Available resource
7 6

result: final allocation
0 0

result: remain request
0 0

Deadlocked process:
none
Not in Deadlock state
```

Figure 20 random input 4 result

프로세스가 1개라면 모든 리소스를 혼자 사용할 것이기 때문에 deadlock state가

될 수 없다.

여러 번의 test 결과 수업 시간에 배운 대로 한 프로세스가 교착 상태가 되었다면 다른 프로세스 또한 교착상태가 될 확률이 매우 높다는 것을 알 수 있었다. 20번의 random input으로 테스트를 해본 결과 12번이 교착 상태였고 모든 프로세스가 deadlocked 상태였다. 나머지 7번의 경우 resource unit의 개수가 0이거나 resource 종류의 개수가 0 혹은 프로세스 개수가 0인 경우였다. (test 결과는 test_log.txt 파일에 저장되어 있다.)

C. Contents of Files

1. OS2020-2_2018312292_김동원_P2.c
실제 프로그램 코드이다
2. input.txt
random input을 사용할 때의 input이다
3. input1.txt
random input을 사용하지 않을 때 사용되는 input으로 RANDOM_INPUT = 0일 때 사용자가 따로 입력파일을 지정하지 않는다면 입력파일로 이용된다.
4. input2.txt
random input을 사용하지 않을 때 사용되는 input이다.
5. input3.txt
random input을 사용하지 않을 때 사용되는 input이다.
6. test_log.txt
20개의 random input을 이용하여 test한 결과이며 console의 text를 그대로 옮겨온 것이다.

3. Conclusion

Deadlock detection 기법을 통해 주어진 혹은 random으로 생성된 입력 파일의 상황에서 deadlock 상태인지 아닌지를 파악할 수 있게 되었고 random으로 생성된 다양한 입력 파일을 통해 확인해본 결과 한 프로세스가 교착상태이면 시스템 내 모든 프로세스가 교착상태일 가능성이 매우 높다는 것을 확인할 수 있었다.

i 엄영익, "Deadlock detection", Lecture notes: 운영체제[2020-2학기] Project-2 [Deadlock Detection], pp. 6, Nov 2020.

ii 엄영익, "Deadlock Handling", Lecture notes: Deadlock Handling, pp. 49