

운영체제론 실습 10주차

CPS LAB

식사하는 철학자 문제

지난 주 실습 프로젝트의 문제점

```
os@os-virtual-machine:~/os-week/week9/mthread_sort_ans$ ./mthread_sort_ans
[thread_m] pid: 4876, tid: f3837740
[thread_2] pid: 4876, tid: f2814700
[thread_1] pid: 4876, tid: f3015700
[thread_3] pid: 4876, tid: f2013700
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
os@os-virtual-machine:~/os-week/week9/mthread_sort_ans$ ./mthread_sort_ans
[thread_m] pid: 4880, tid: dbf6740
[thread_1] pid: 4880, tid: d3d4700
[thread_3] pid: 4880, tid: c3d2700
[thread_2] pid: 4880, tid: cbd3700
1 2 3 4 5 6 7 8 13 9 10 11 12 14 15 16
os@os-virtual-machine:~/os-week/week9/mthread_sort_ans$ ./mthread_sort_ans
[thread_m] pid: 4884, tid: c560b740
[thread_1] pid: 4884, tid: c4de9700
[thread_2] pid: 4884, tid: c45e8700
[thread_3] pid: 4884, tid: c3de7700
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
os@os-virtual-machine:~/os-week/week9/mthread_sort_ans$ ./mthread_sort_ans
[thread_m] pid: 4888, tid: 4b096740
[thread_1] pid: 4888, tid: 4a874700
[thread_2] pid: 4888, tid: 4a073700
[thread_3] pid: 4888, tid: 49872700
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
os@os-virtual-machine:~/os-week/week9/mthread_sort_ans$ ./mthread_sort_ans
[thread_m] pid: 4892, tid: 39a56740
[thread_1] pid: 4892, tid: 39234700
[thread_2] pid: 4892, tid: 38a33700
[thread_3] pid: 4892, tid: 38232700
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
os@os-virtual-machine:~/os-week/week9/mthread_sort_ans$ ./mthread_sort_ans
[thread_m] pid: 4896, tid: 7caaa740
[thread_1] pid: 4896, tid: 7c288700
[thread_2] pid: 4896, tid: 7ba87700
[thread_3] pid: 4896, tid: 7b286700
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
```

Thread 1, 2 : Sorting
Thread 3 : Merge

thread_3이 thread_2 보다 먼저 종료

정렬이 완료되기 전에 병합

프로세스 동기화 문제 제기

간단한 해결 방안

```
/* 2.SORT */
/* TODO 1: 1st Thread / Sort first half of data */
thr_id = pthread_create(&tid[0], NULL, sort_thread, &first_half);
if (thr_id < 0)
{
    perror("thread create error : ");
    exit(0);
}

/* TODO 2: 2nd Thread / Sort second half of data */
thr_id = pthread_create(&tid[1], NULL, sort_thread, &second_half);
if (thr_id < 0)
{
    perror("thread create error : ");
    exit(0);
}

/* 3.MERGE */
/* TODO 3: 3rd Thread / Merge the result of two halves */
thr_id = pthread_create(&tid[2], NULL, merge_thread, &merge_range);
if (thr_id < 0)
{
    perror("thread create error : ");
    exit(0);
}

/* TODO 4: waits for the first thread */
pthread_join(tid[0], (void **)&status);
/* TODO 5: waits for the second thread */
pthread_join(tid[1], (void **)&status);
/* TODO 6: waits for the third thread */
pthread_join(tid[2], (void **)&status);
```



```
/* 2.SORT */
/* TODO 1: 1st Thread / Sort first half of data */
thr_id = pthread_create(&tid[0], NULL, sort_thread, &first_half);
if (thr_id < 0)
{
    perror("thread create error : ");
    exit(0);
}

/* TODO 2: 2nd Thread / Sort second half of data */
thr_id = pthread_create(&tid[1], NULL, sort_thread, &second_half);
if (thr_id < 0)
{
    perror("thread create error : ");
    exit(0);
}

/* TODO 4: waits for the first thread */
pthread_join(tid[0], (void **)&status);
/* TODO 5: waits for the second thread */
pthread_join(tid[1], (void **)&status);

/* 3.MERGE */
/* TODO 3: 3rd Thread / Merge the result of two halves */
thr_id = pthread_create(&tid[2], NULL, merge_thread, &merge_range);
if (thr_id < 0)
{
    perror("thread create error : ");
    exit(0);
}

/* TODO 6: waits for the third thread */
pthread_join(tid[2], (void **)&status);
```

thread_3 생성 전에
thread_1, thread_2 종료 시키기

해결 방안 적용 결과

쓰레드가 순서대로 잘 종료됨



정렬이 끝난 후 병합됨

```
os@os-virtual-machine:~/os-week/week9/mthread_sort_ans$ ./mthread_sort_ans_2
[thread_m] pid: 10300, tid: b8d89740
[thread_1] pid: 10300, tid: b8566700
[thread_2] pid: 10300, tid: b7d65700
[thread_3] pid: 10300, tid: b7d65700
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
os@os-virtual-machine:~/os-week/week9/mthread_sort_ans$ ./mthread_sort_ans_2
[thread_m] pid: 10304, tid: 2382c740
[thread_1] pid: 10304, tid: 23009700
[thread_2] pid: 10304, tid: 22808700
[thread_3] pid: 10304, tid: 22808700
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
os@os-virtual-machine:~/os-week/week9/mthread_sort_ans$ ./mthread_sort_ans_2
[thread_m] pid: 10308, tid: d9140740
[thread_1] pid: 10308, tid: d891d700
[thread_2] pid: 10308, tid: d811c700
[thread_3] pid: 10308, tid: d811c700
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
os@os-virtual-machine:~/os-week/week9/mthread_sort_ans$ ./mthread_sort_ans_2
[thread_m] pid: 10312, tid: c224b740
[thread_1] pid: 10312, tid: c1a28700
[thread_2] pid: 10312, tid: c1227700
[thread_3] pid: 10312, tid: c1227700
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
os@os-virtual-machine:~/os-week/week9/mthread_sort_ans$ ./mthread_sort_ans_2
[thread_m] pid: 10316, tid: 1ec15740
[thread_1] pid: 10316, tid: 1e3f2700
[thread_2] pid: 10316, tid: 1dbf1700
[thread_3] pid: 10316, tid: 1dbf1700
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
os@os-virtual-machine:~/os-week/week9/mthread_sort_ans$ ./mthread_sort_ans_2
[thread_m] pid: 10320, tid: de791740
[thread_1] pid: 10320, tid: ddf6e700
[thread_2] pid: 10320, tid: dd76d700
[thread_3] pid: 10320, tid: dd76d700
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
os@os-virtual-machine:~/os-week/week9/mthread_sort_ans$ ./mthread_sort_ans_2
[thread_m] pid: 10324, tid: 84c20740
[thread_1] pid: 10324, tid: 843fd700
[thread_2] pid: 10324, tid: 83bfc700
[thread_3] pid: 10324, tid: 83bfc700
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
```


목차

1. 프로세스 동기화

- Critical Section의 문제 해결 요구 조건
- Mutex (뮤텍스)
- Semaphore (세마포어)

2. 식사하는 철학자 문제

- Solution 1: Right first solution
- Solution 2: Right-Left solution
- Solution 3: Use of Arbitrator
- Solution 4: Tanenbaum's solution

3. Solution Testing

- 측정 기준
- Solution 3 testing 결과
- Solution 4 testing 결과

CHAPTER 1.

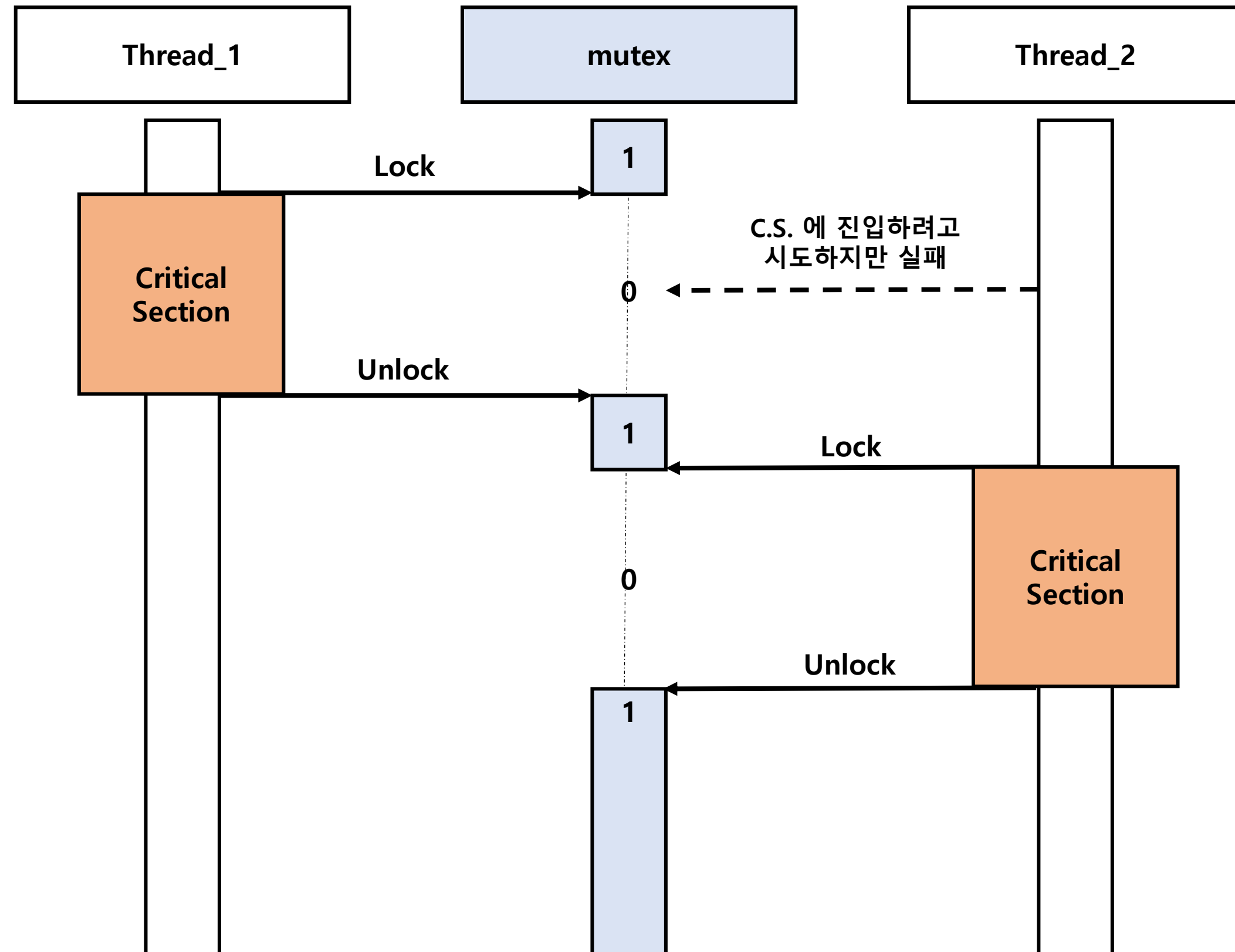
프로세스 동기화

- Critical Section의 문제 해결 요구 조건
- Mutex (뮤텍스)
- Semaphore (세마포어)

임계구역(critical section) 문제 해결 요구 조건

- 상호 배제(mutual exclusion)
 - 특정 프로세스가 임계구역(critical section)에서 실행 중이면,
다른 프로세스들은 자신들의 임계구역에서 실행될 수 없음
- 진행(progress)
 - 임계구역에 아무 프로세스도 실행되고 있지 않은데, 임계구역으로 진입하려고 하는 프로세스들이 있다면,
어느 프로세스가 진입할 수 있는지를 결정해야 되며, 이 결정은 무한정 연기될 수 없음
- 한정된 대기(bounded waiting)
 - 프로세스가 자기의 임계 구역에 진입하려는 요청을 한 후부터, 그 요청이 허용될 때까지
다른 프로세스들이 그들 자신의 임계구역에 진입하도록 허용되는 횟수에 제한이 있어야 함

Mutex (뮤텍스)



pthread_mutex_init()

```
#include <pthread.h>
int pthread_mutex_init(pthread_mutex_t *restrict mutex,
                       const pthread_mutexattr_t *restrict attr);
```

- **attr**로 지정하는 속성을 가지고 **mutex**를 초기화 한다.

- 1) **mutex**: 초기화 하고자 하는 mutex
- 2) **attr**: mutex 속성 (NULL : 기본 값)

- 사용 예제

```
pthread_mutex_t mutex;
pthread_mutex_init(&mutex, NULL);
```

pthread_mutex_lock()

```
#include <pthread.h>
int pthread_mutex_lock(pthread_mutex_t *mutex);
```

- mutex를 lock 한다.
 - 1) *mutex*: lock 하고자 하는 mutex

- 사용 예제

```
pthread_mutex_t mutex;
pthread_mutex_lock(&mutex);
```

pthread_mutex_unlock()

```
#include <pthread.h>
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

- mutex를 unlock 한다.
 - 1) *mutex*: unlock 하고자 하는 mutex

- 사용 예제

```
pthread_mutex_t mutex;
pthread_mutex_unlock(&mutex);
```

pthread_mutex_destroy()

```
#include <pthread.h>
int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

- mutex를 소멸시킴으로, mutex를 통해 할당할 수 있었던 공유 자원을 해방시킴

1) *mutex*: 소멸시킬 mutex

- 사용 예제

```
pthread_mutex_t mutex;
pthread_mutex_destroy(&mutex);
```

Mutex 사용 예제

week10/mutex/print_mutex.c

```
15 pthread_mutex_t mutex;
16
17 void *print_red(void *data) {
18     while (1) {
19         pthread_mutex_lock(&mutex);
20         printf("%sZERG UNITS\n", KRED);
21         printf("-----\n");
22         printf("Hydralisk\n");
23         printf("Mutalisk\n");
24         printf("Ultralisk\n");
25         printf("-----\n%s", KNRM);
26         pthread_mutex_unlock(&mutex);
27         usleep(rand()%2);
28     }
29 }
30
31 void *print_blue(void *data) {
32     while (1) {
33         pthread_mutex_lock(&mutex);
34         printf("%sTERRAN UNITS\n", KBLU);
35         printf("-----\n");
36         printf("Marine\n");
37         printf("Siege Tank\n");
38         printf("Wraith\n");
39         printf("-----\n%s", KNRM);
40         pthread_mutex_unlock(&mutex);
41         usleep(rand()%2);
42     }
43 }
44
45 void *print_green(void *data) {
46     while (1) {
47         pthread_mutex_lock(&mutex);
48         printf("%sPROTOSS UNITS\n", KGRN);
49         printf("-----\n");
50         printf("Zealot\n");
51         printf("Dragoon\n");
52         printf("Observer\n");
53         printf("-----\n%s", KNRM);
54         pthread_mutex_unlock(&mutex);
55         usleep(rand()%2);
56     }
57 }
```

```
59 int main() {
60     pthread_t thread_id[N];
61     srand(time(NULL));
62
63     pthread_mutex_init(&mutex, NULL);
64
65     pthread_create(&thread_id[0], NULL, print_red, NULL);
66     pthread_create(&thread_id[1], NULL, print_blue, NULL);
67     pthread_create(&thread_id[2], NULL, print_green, NULL);
68
69     for (int i = 0; i < N; i++) {
70         pthread_join(thread_id[i], NULL);
71     }
72
73     pthread_mutex_destroy(&mutex);
74 }
```

모든 thread가
종료된 후, mutex 소멸

**Critical
Section**

green이 Critical Section에 진입한 경우,
red와 blue는 진입할 수 없음

Mutex 사용 예제 실행 결과

1. Mutex를 사용하지 않았을 경우 (상호배제 위반)

```
$ ./print_no_mutex
```

```
TERRAN UNITS
-----
Marine
Siege Tank
Wraith
-----
TERRAN UNITS
-----
Marine
Siege Tank
Wraith
-----
ZERG UNITS
-----
PROTOSS UNITS
-----
Zealot
Dragoon
Observer
-----
```

red thread가 blue, green thread로부터 방해가 받아 작업을 제대로 수행하지 못함

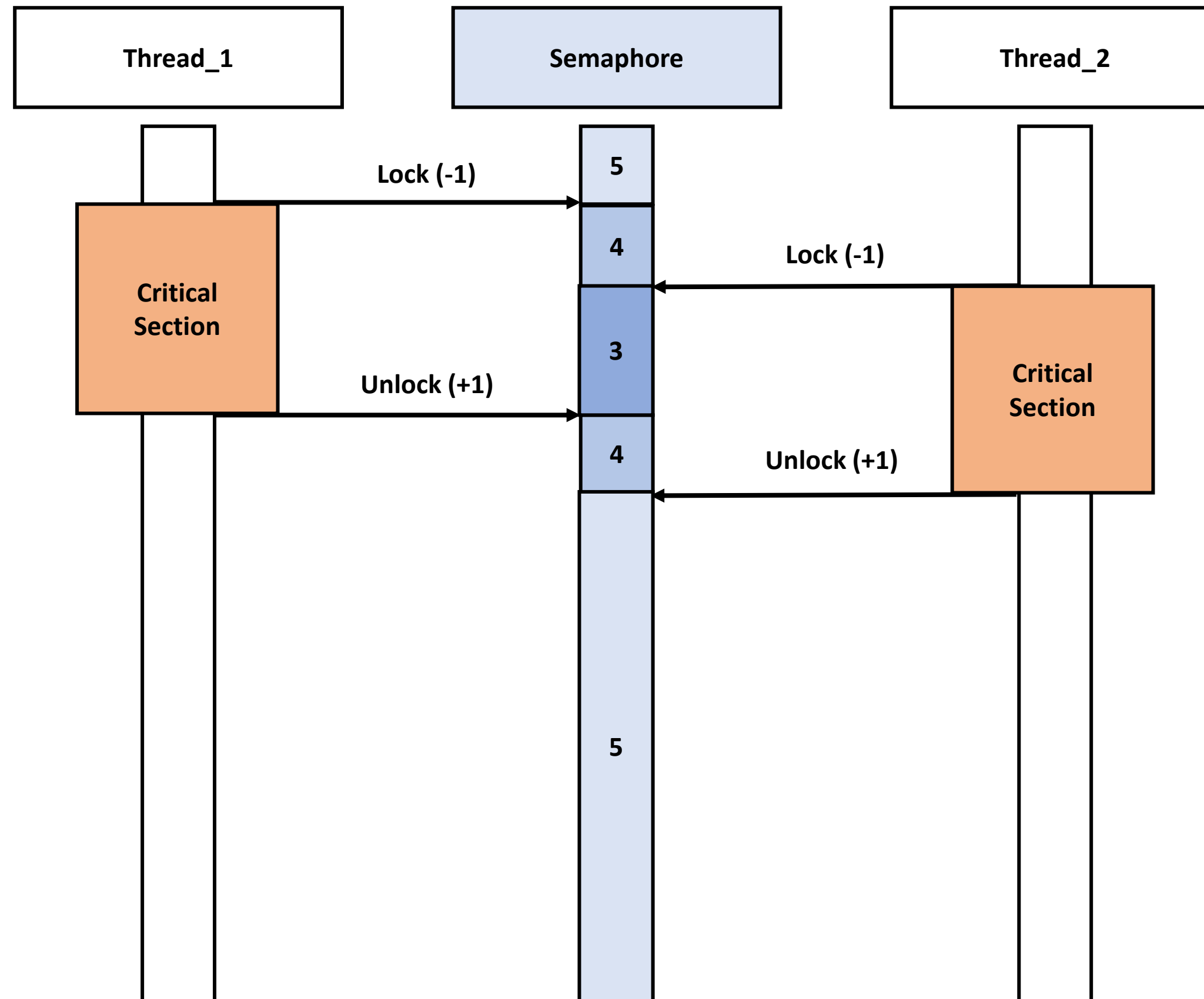
2. Mutex를 사용한 경우

```
$ ./print_mutex
```

```
PROTOSS UNITS
-----
Zealot
Dragoon
Observer
-----
ZERG UNITS
-----
Hydralisk
Mutalisk
Ultralisk
-----
TERRAN UNITS
-----
Marine
Siege Tank
Wraith
-----
PROTOSS UNITS
-----
Zealot
Dragoon
Observer
-----
```

하나의 thread가 모든 작업을 완료한 후, 다음 thread가 작업을 수행함

Semaphore (세마포어)



sem_init()

```
#include <semaphore.h>
int sem_init(sem_t *sem, int pshared, unsigned int value);
```

- value 값을 가진 semaphore를 초기화한다.
 - 1) *sem*: 초기화 하고자 하는 semaphore
 - 2) *pshared*: 0 값이면 스레드 간 공유되고, 아니면 프로세스 간 공유됨
 - 3) *value*: semaphore의 초기값을 지정함
- 사용 예제

```
sem_t sem;
sem_init(&sem, 0, 1);
```

sem_wait()

```
#include <semaphore.h>
int sem_wait(sem_t *sem);
```

- semaphore 값을 1 감소시킨다. (=lock 수행)

1) *sem* : lock 하고자 하는 semaphore

- 사용 예제

```
sem_t sem;
sem_wait(&sem);
```

sem_post()

```
#include <semaphore.h>
int sem_post(sem_t *sem);
```

- semaphore 값을 1 증가시킨다. (=unlock 수행)

1) *sem* : unlock 하고자 하는 semaphore

- 사용 예제

```
sem_t sem;
sem_post(&sem);
```

sem_destroy()

```
#include <semaphore.h>
int sem_destroy(sem_t *sem);
```

- semaphore를 파괴한다.
 - 1) *sem* : 파괴하고자 하는 semaphore

- 사용 예제

```
sem_t sem;
sem_destroy(&sem);
```

Semaphore 사용 예제

week10/semaphore/print_semaphore.c

```
9 void* threadFunc1(void *arg)
10 {
11     sem_wait(&sem);
12     printf("Thread 1 go to the toilet...\n");
13     sleep(5);
14     sem_post(&sem);
15     printf("Thread 1 exits\n");
16     return NULL;
17 }
18 void* threadFunc2(void *arg)
19 {
20     sem_wait(&sem);
21     printf("Thread 2 go to the toilet...\n");
22     sleep(3);
23     sem_post(&sem);
24     printf("Thread 2 exits\n");
25     return NULL;
26 }
27 void* threadFunc3(void *arg)
28 {
29     sem_wait(&sem);
30     printf("Thread 3 go to the toilet...\n");
31     sleep(3);
32     sem_post(&sem);
33     printf("Thread 3 exits\n");
34     return NULL;
35 }
36 }
```

```
51 int main(void)
52 {
53     pthread_t tid1, tid2, tid3;
54     // Semaphore init value = 2
55     sem_init(&sem, 0, 2);
56     pthread_create(&tid1, NULL, threadFunc1, NULL);
57     pthread_create(&tid2, NULL, threadFunc2, NULL);
58     pthread_create(&tid3, NULL, threadFunc3, NULL);
59     pthread_join(tid1, NULL);
60     pthread_join(tid2, NULL);
61     pthread_join(tid3, NULL);
62     sleep(3);
63     sem_destroy(&sem);
64     printf("FINISHED\n");
65     return 0;
66 }
```

화장실 칸 수 : 2칸 ←

Semaphore 사용 예제 실행 결과

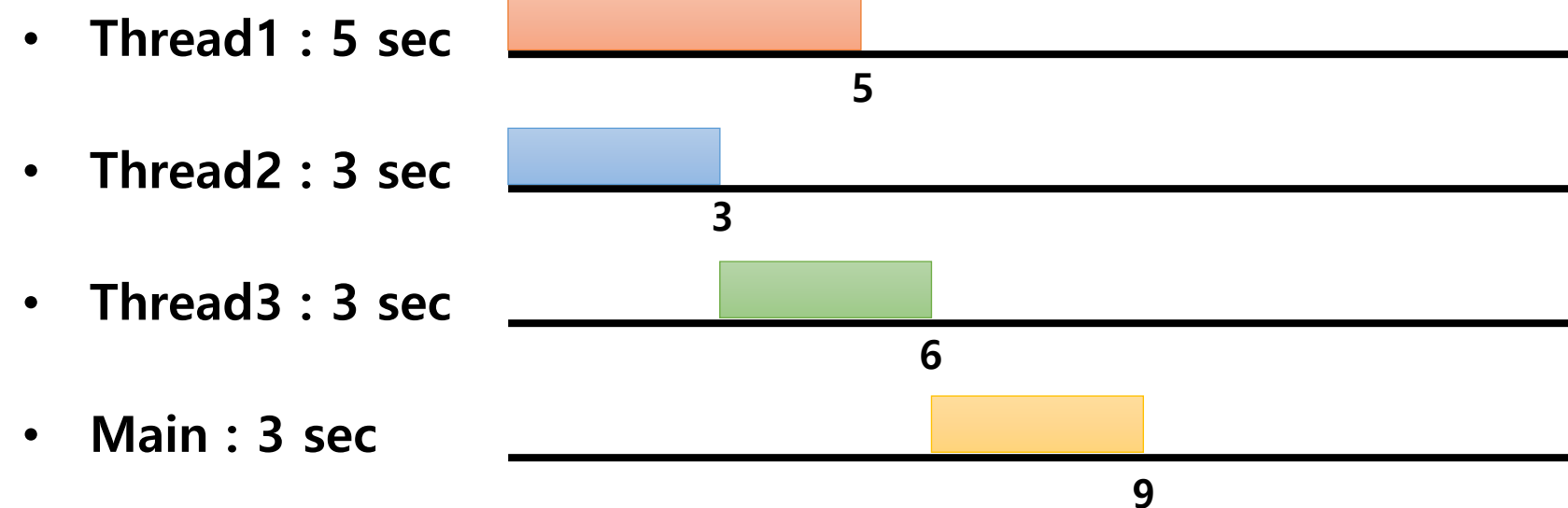
```
$ time ./print_semaphore
```

```
os@os-virtual-machine:~/Downloads/semaphore$ time ./print_semaphore
Thread 1 go to the toilet...
Thread 2 go to the toilet...
Thread 2 exits
Thread 3 go to the toilet...
Thread 1 exits
Thread 3 exits
FINISHED

real    0m9.003s
user    0m0.000s
sys     0m0.002s
```

→ Thread2가 종료된 후, Thread3가 바로 진입

• Code Timeline

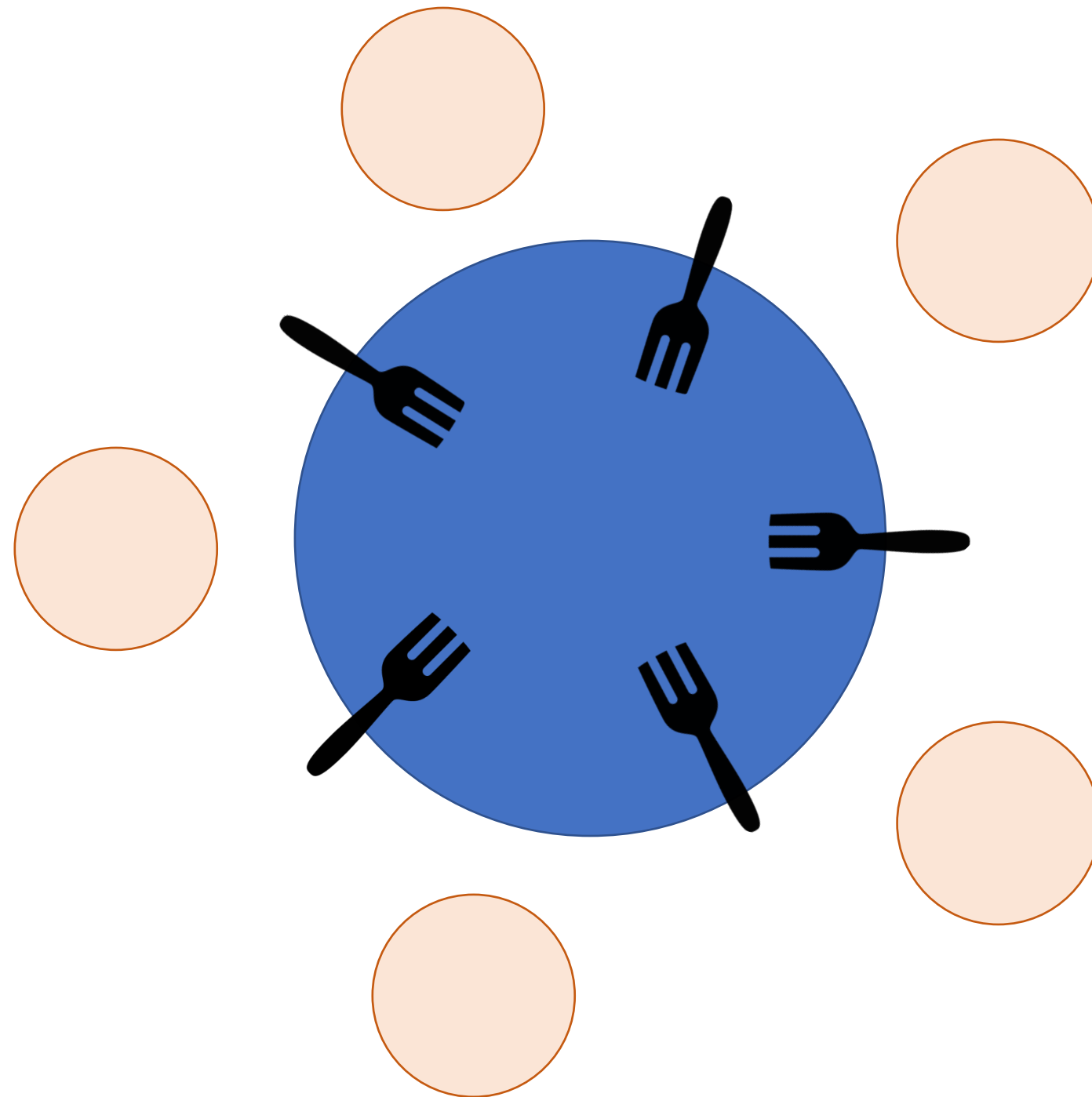


CHAPTER 2.

식사하는 철학자 문제

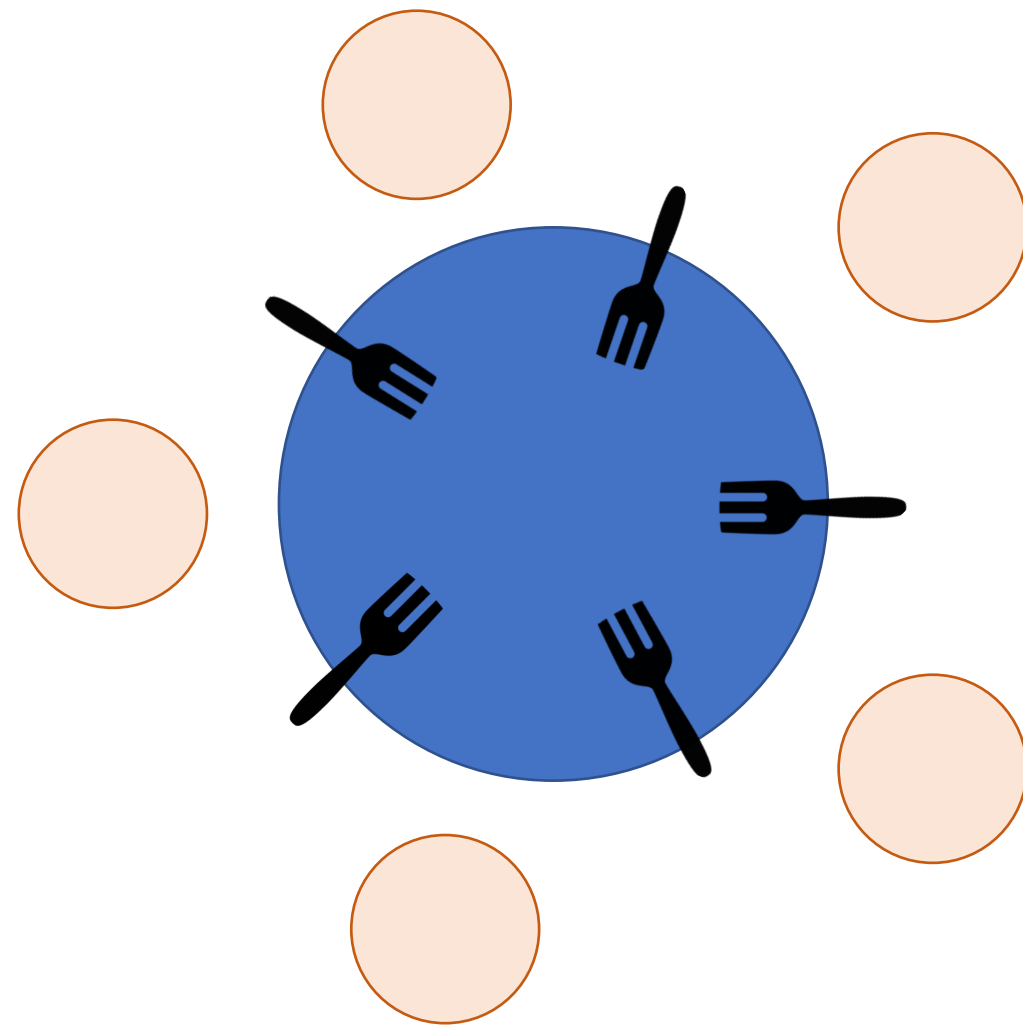
- Solution 1: Right first solution
- Solution 2: Right-Left solution
- Solution 3: Use of Arbitrator
- Solution 4: Tanenbaum's solution


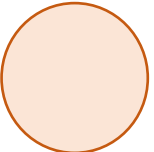
식사하는 철학자 문제 (1)



- 원형 테이블에 5명의 사람과 5개의 포크가 있다.
- 왼쪽 그림과 같이 각 포크는 두 철학자 사이에 존재한다.
- 철학자는 한 번에 하나의 포크만 집을 수 있다.
- 철학자는 어떤 음식을 먹기 위해 본인 위치에서 가장 가까운 양쪽 2개의 포크를 필요로 한다.
 - 2개의 포크를 집게 되면, 철학자는 식사를 한다.
 - 그 외의 경우, 철학자는 포크 집기를 시도하거나 생각에 빠진다.

식사하는 철학자 문제 (2)



상태				
포크 	사용	← 대응 →	Locked	공유자원
	미사용		Unlocked	
철학자 	먹는다		Running	프로세스, 스레드
	생각한다		Waiting	

Solution 1 : Right fork first Solution

- 각 철학자가 하는 일

```
do {  
    lock(fork[i]);           // 왼쪽 포크를 집음  
    lock(fork[(i+1)%N]);     // 오른쪽 포크를 집음  
  
    // Critical Section 진입  
    Eat();  
  
    unlock(fork[i]);         // 왼쪽 포크를 내려놓음  
    unlock(fork[(i+1)%N]);   // 오른쪽 포크를 내려놓음  
  
    // 다음 Critical Section 진입까지 대기  
    Think();  
} while(true);
```

PSEUDO CODE

Solution 1 : Right fork first Solution

week10/solution_1/right_fork_first.c

```
87 int main() {
88     pthread_t thread_id[N];
89
90     srand(time(NULL));
91
92     pthread_mutex_init(&print_mutex, NULL);
93
94     for (int i = 0; i < N; i++) {
95         sem_init(&forks[i], 0, 1);
96         state[i] = 0;
97     }
98
99     print_table_index();
100
101     for (int i = 0; i < N; i++) {
102         name[i] = i;
103         pthread_create(&thread_id[i], NULL, philosopher, &name[i]);
104     }
105
106     for (int i = 0; i < N; i++) {
107         pthread_join(thread_id[i], NULL);
108     }
109
110     for (int i = 0; i < N; i++) {
111         sem_destroy(&forks[i]);
112     }
113
114     pthread_mutex_destroy(&print_mutex);
115 }
```

Semaphore(포크) 값 초기화

각 Thread(철학자) 생성

Solution 1 : Right fork first Solution

week10/solution_1/right_fork_first.c

- 각 철학자가 하는 일

```
59 void *philosopher(void *_phil) {
60     int phil = *((int *)_phil);
61
62     do {
63         sem_wait(&forks[(phil + 1) % N]);
64         print_fork(phil, (phil + 1) % N);
65
66         // DEADLOCK: Every philosopher holds
67         // the right fork simultaneously
68         sleep(2);
69
70         sem_wait(&forks[phil]);
71         print_fork(phil, phil);
72
73         update_state(phil, EATING);
74
75         usleep(rand() % 1000000);
76
77         sem_post(&forks[(phil + 1) % N]);
78         sem_post(&forks[phil]);
79
80         update_state(phil, THINKING);
81         usleep(rand() % 2000000);
82
83     } while (1);
84     pthread_exit(0);
85 }
```

자신의 오른쪽에 있는 포크를 집음

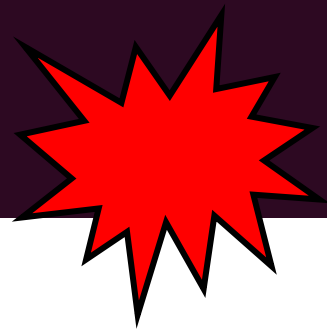
모든 철학자가 동시에 오른쪽 포크를 집기 때문에,
어느 누구도 왼쪽 포크를 사용할 수 없다.

→ **DEADLOCK(교착상태) 발생**

Solution 1 : 실행 결과

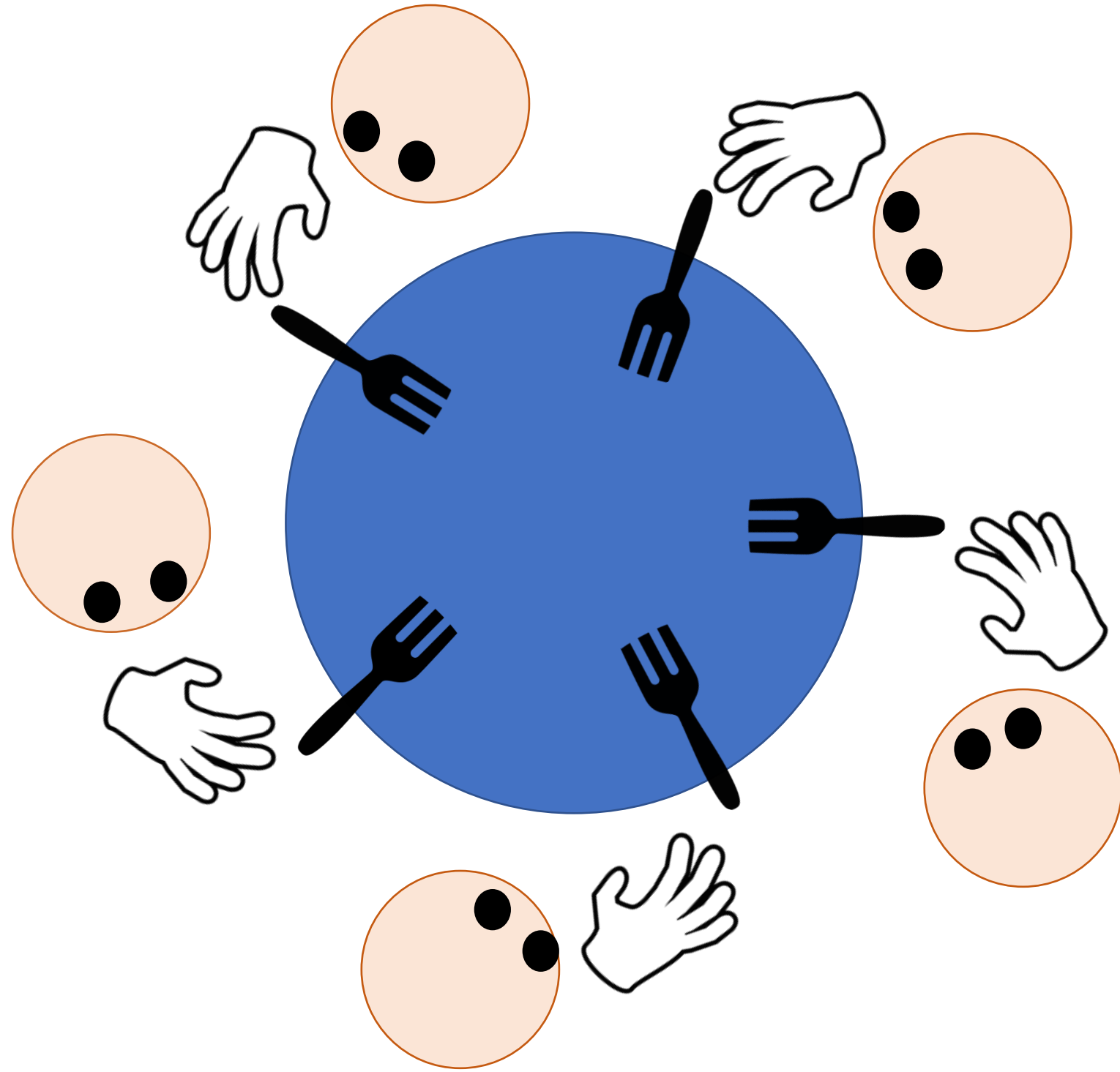
```
os@os-virtual-machine:~/Downloads/solution_1$ ./right_fork_first
```

```
=====
====
|      PHIL[0]      ||      PHIL[1]      ||      PHIL[2]      ||      PHIL[3]      ||      PHIL[4]      |
=====
====
PHIL[0] has taken 1 th fork.
PHIL[1] has taken 2 th fork.
PHIL[2] has taken 3 th fork.
PHIL[3] has taken 4 th fork.
PHIL[4] has taken 0 th fork.
```



DEADLOCK 발생

Deadlock (교착상태)



- Deadlock(교착상태) 발생
 - 모든 철학자가 식사를 하지 못해 굶어 죽음

Deadlock이 발생할 조건

1. 상호배제 (Mutual Exclusion)

- 한 명의 철학자가 포크를 잡고 있으면, 다른 철학자는 집을 수 없음

2. 보유 및 대기 (Hold and Wait)

- 한 쪽 포크는 가지고 있는 상태에서 다른 쪽 포크를 기다리고 있음

3. 비선점 (Non-Preemptive)

- 철학자가 포크를 가지고 있으면, 다른 철학자가 포크를 뺏을 수 없음

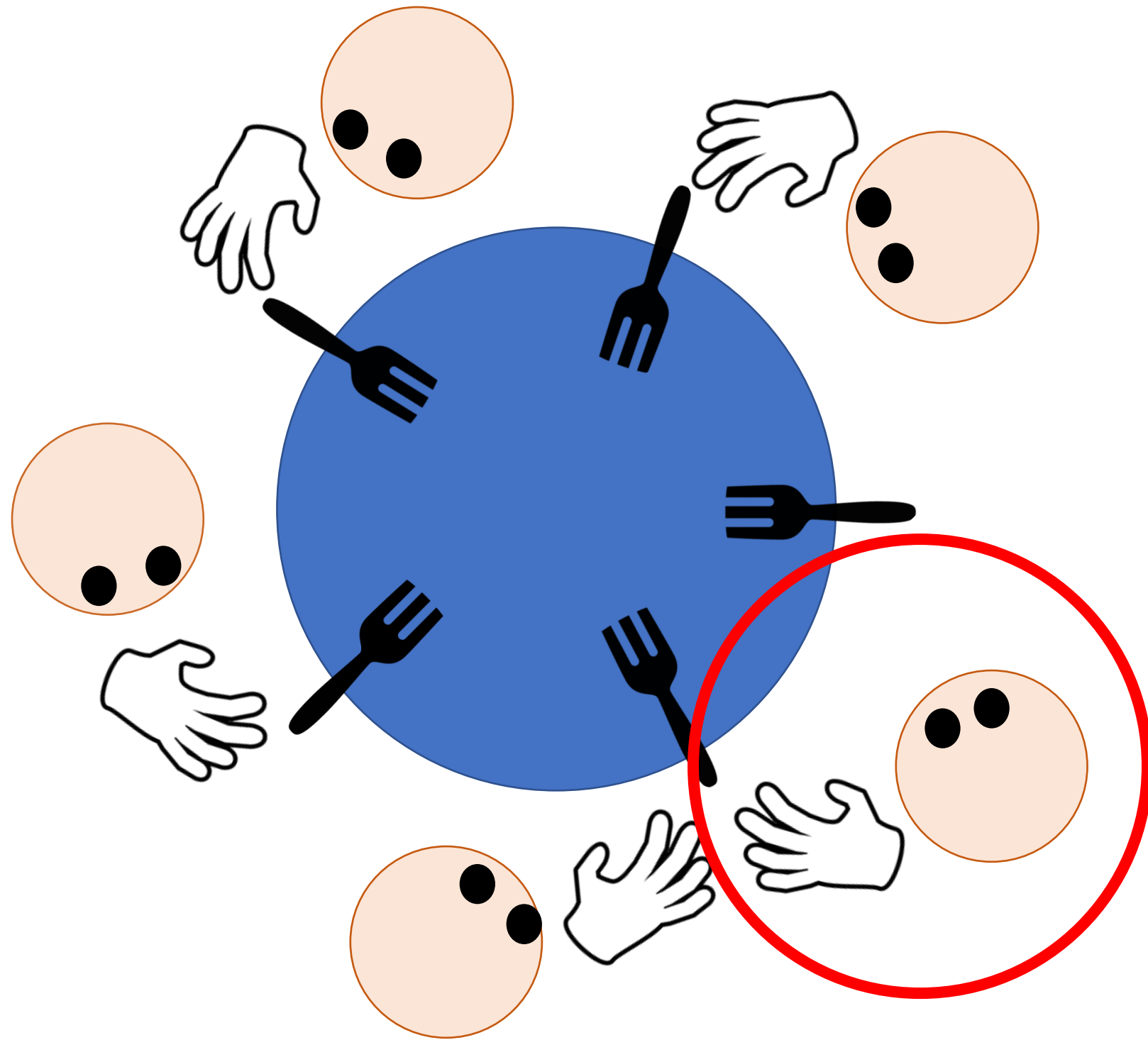
4. 환형 대기 (Circular Wait)

- 원형 테이블에서 이루어지는 형태

이 중 하나라도 해당되지 않으면,
Deadlock이 발생하지 않음

Solution 2 : Right-Left Solution

- Deadlock(교착상태)을 피하기 위해,
철학자들 중 한 명은 포크를 왼쪽부터 잡게 함



Solution 2 : Right-Left Solution

- 각 철학자가 하는 일

왼손 잡이

```
do {  
    lock(fork[i]);  
    lock(fork[(i+1)%N]);  
  
    // Critical Section 진입  
    Eat();  
  
    unlock(fork[i]);  
    unlock(fork[(i+1)%N]);  
  
    // 다음 Critical Section 진입까지 대기  
    Think();  
} while(true);
```

오른손 잡이

```
do {  
    lock(fork[(i+1)%N]);  
    lock(fork[i]);  
  
    // Critical Section 진입  
    Eat();  
  
    unlock(fork[(i+1)%N]);  
    unlock(fork[i]);  
  
    // 다음 Critical Section 진입까지 대기  
    Think();  
} while(true);
```

PSEUDO CODE

Solution 2 : Right-Left Solution

week10/solution_2/right_left_forks.c

```
111 int main() {
112     pthread_t thread_id[N];
113
114     srand(time(NULL));
115
116     /* Only one philosopher can use print_state at a time. */
117     pthread_mutex_init(&print_mutex, NULL);
118
119     for (int i = 0; i < N; i++) {
120         sem_init(&forks[i], 0, 1);
121         state[i] = 0;
122     }
123
124     print_table_index();
125
126     name[0] = 0;
127     pthread_create(&thread_id[0], NULL, left_handed_philosopher, &name[0]);
128     for (int i = 1; i < N; i++) {
129         name[i] = i;
130         pthread_create(&thread_id[i], NULL, right_handed_philosopher, &name[i]);
131     }
132
133     for (int i = 0; i < N; i++) {
134         pthread_join(thread_id[i], NULL);
135     }
136
137     for (int i = 0; i < N; i++) {
138         sem_destroy(&forks[i]);
139     }
140
141     pthread_mutex_destroy(&print_mutex);
142 }
```

→ 왼손잡이 철학자(Thread) 생성

Solution 2 : Right-Left Solution

week10/solution_2/right_left_forks.c

왼손 잡이

```
59 void *left_handed_philosopher(void *_phil) {
60     int phil = *((int *)_phil);
61
62     do {
63         sem_wait(&forks[phil]);
64         print_fork(phil, phil, "take");
65         sem_wait(&forks[(phil + 1) % N]);
66         print_fork(phil, (phil + 1) % N, "take");
67
68         update_state(phil, EATING);
69
70         usleep(rand() % 1000000);
71
72         sem_post(&forks[phil]);
73         print_fork(phil, phil, "put down");
74         sem_post(&forks[(phil + 1) % N]);
75         print_fork(phil, (phil + 1) % N, "put down");
76         update_state(phil, THINKING);
77
78         usleep(rand() % 2000000);
79     } while (1);
80
81     pthread_exit(0);
82 }
83 }
```

왼쪽 포크를 먼저 집음

오른손 잡이

```
85 void *right_handed_philosopher(void *_phil) {
86     int phil = *((int *)_phil);
87
88     do {
89         sem_wait(&forks[(phil + 1) % N]);
90         print_fork(phil, (phil + 1) % N, "take");
91         sem_wait(&forks[phil]);
92         print_fork(phil, phil, "take");
93
94         update_state(phil, EATING);
95
96         usleep(rand() % 1000000);
97
98         sem_post(&forks[(phil + 1) % N]);
99         print_fork(phil, (phil + 1) % N, "put down");
100        sem_post(&forks[phil]);
101        print_fork(phil, phil, "put down");
102        update_state(phil, THINKING);
103
104        usleep(rand() % 2000000);
105    } while (1);
106
107    pthread_exit(0);
108 }
109 }
```

오른쪽 포크를 먼저 집음

Solution 2 : 실행 결과

```
pro@pro-virtual-machine:~/os-week/week10/solution_2$ ./right_left_forks
=====
| PHIL[0] || PHIL[1] || PHIL[2] || PHIL[3] || PHIL[4] |
=====
PHIL[0] take 0 th fork.
PHIL[0] take 1 th fork.
| EATING || INIT || INIT || INIT || INIT |
PHIL[1] take 2 th fork.
PHIL[2] take 3 th fork.
PHIL[3] take 4 th fork.
PHIL[0] put down 0 th fork.
PHIL[0] put down 1 th fork.
| THINKING || INIT || INIT || INIT || INIT |
PHIL[4] take 0 th fork.
PHIL[1] take 1 th fork.
| THINKING || EATING || INIT || INIT || INIT |
PHIL[1] put down 2 th fork.
PHIL[1] put down 1 th fork.
| THINKING || THINKING || INIT || INIT || INIT |
PHIL[2] take 2 th fork.
| THINKING || THINKING || EATING || INIT || INIT |
PHIL[2] put down 3 th fork.
PHIL[2] put down 2 th fork.
| THINKING || THINKING || THINKING || INIT || INIT |
PHIL[1] take 2 th fork.
PHIL[1] take 1 th fork.
| THINKING || EATING || THINKING || INIT || INIT |
PHIL[3] take 3 th fork.
| THINKING || EATING || THINKING || EATING || INIT |
PHIL[3] put down 4 th fork.
PHIL[3] put down 3 th fork.
| THINKING || EATING || THINKING || THINKING || INIT |
PHIL[4] take 4 th fork.
| THINKING || EATING || THINKING || THINKING || EATING |
PHIL[4] put down 0 th fork.
PHIL[4] put down 4 th fork.
| THINKING || EATING || THINKING || THINKING || THINKING |
PHIL[0] take 0 th fork.
PHIL[2] take 3 th fork.
PHIL[1] put down 2 th fork.
PHIL[1] put down 1 th fork.
| THINKING || THINKING || THINKING || THINKING || THINKING |
PHIL[0] take 1 th fork.
| EATING || THINKING || THINKING || THINKING || THINKING |
PHIL[2] take 2 th fork.
| EATING || THINKING || EATING || THINKING || THINKING |
=====
```

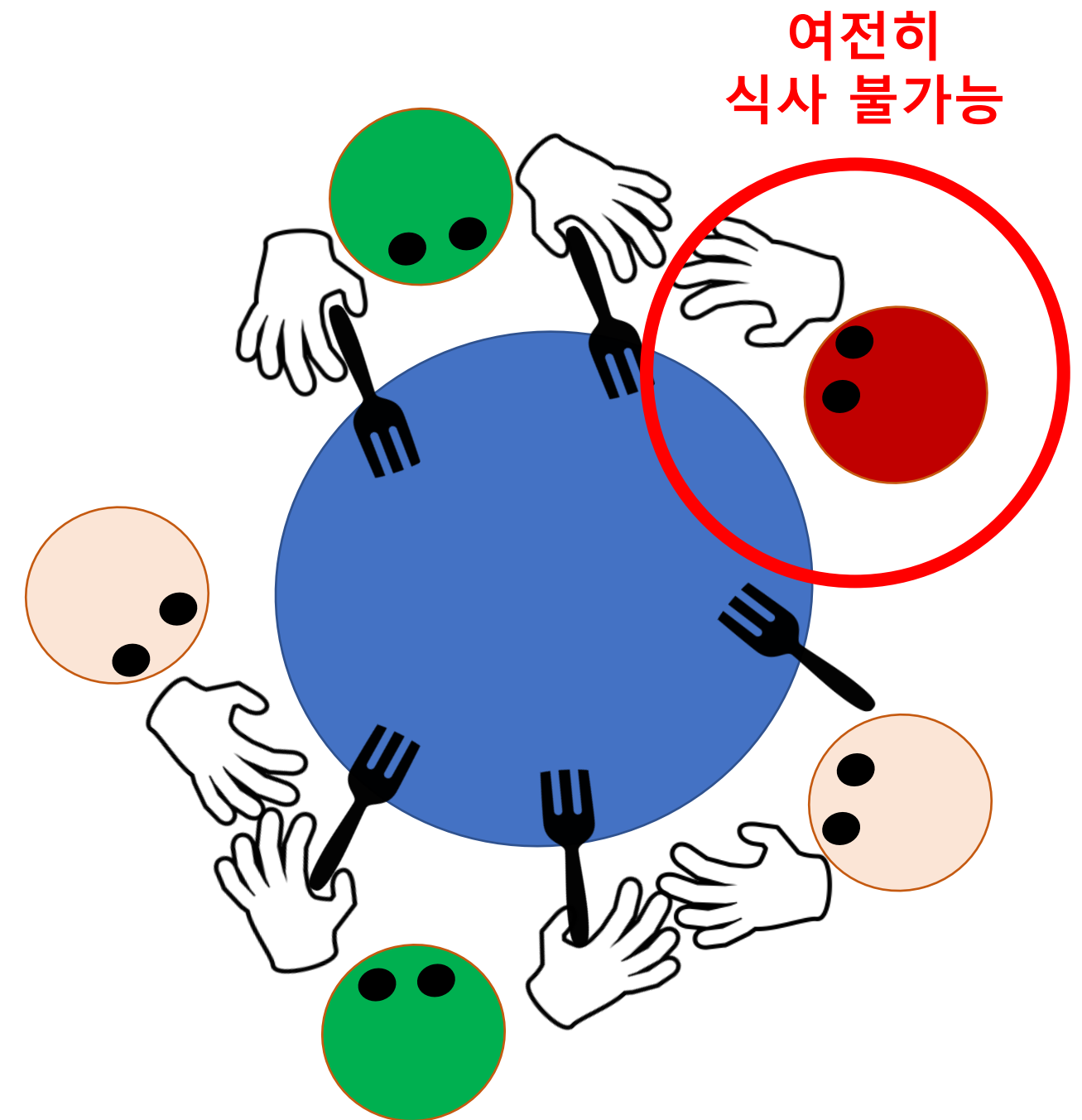
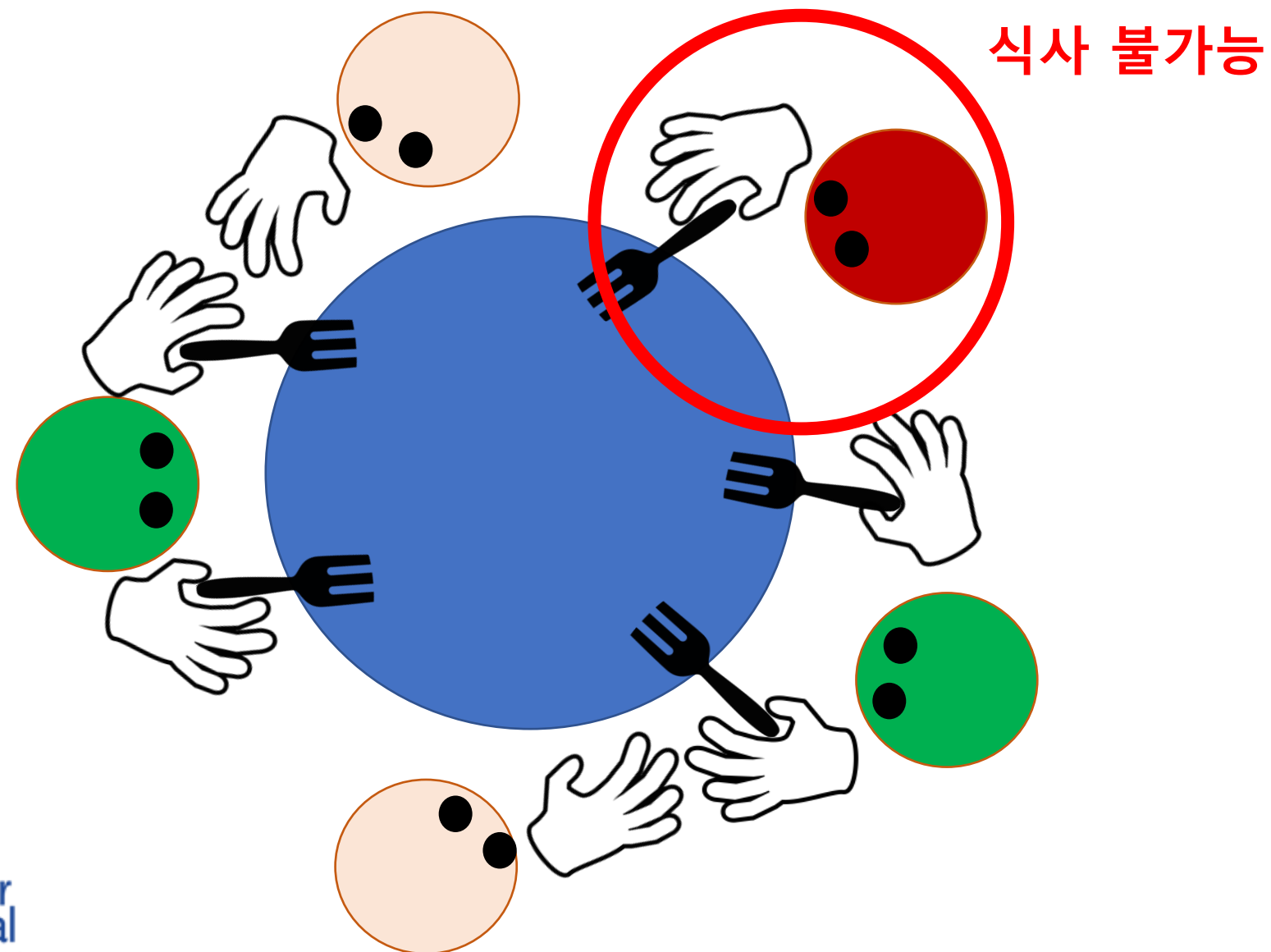
운이 나쁘면 계속 식사를 못할 수도 있음



Starvation이 발생할 수 있음

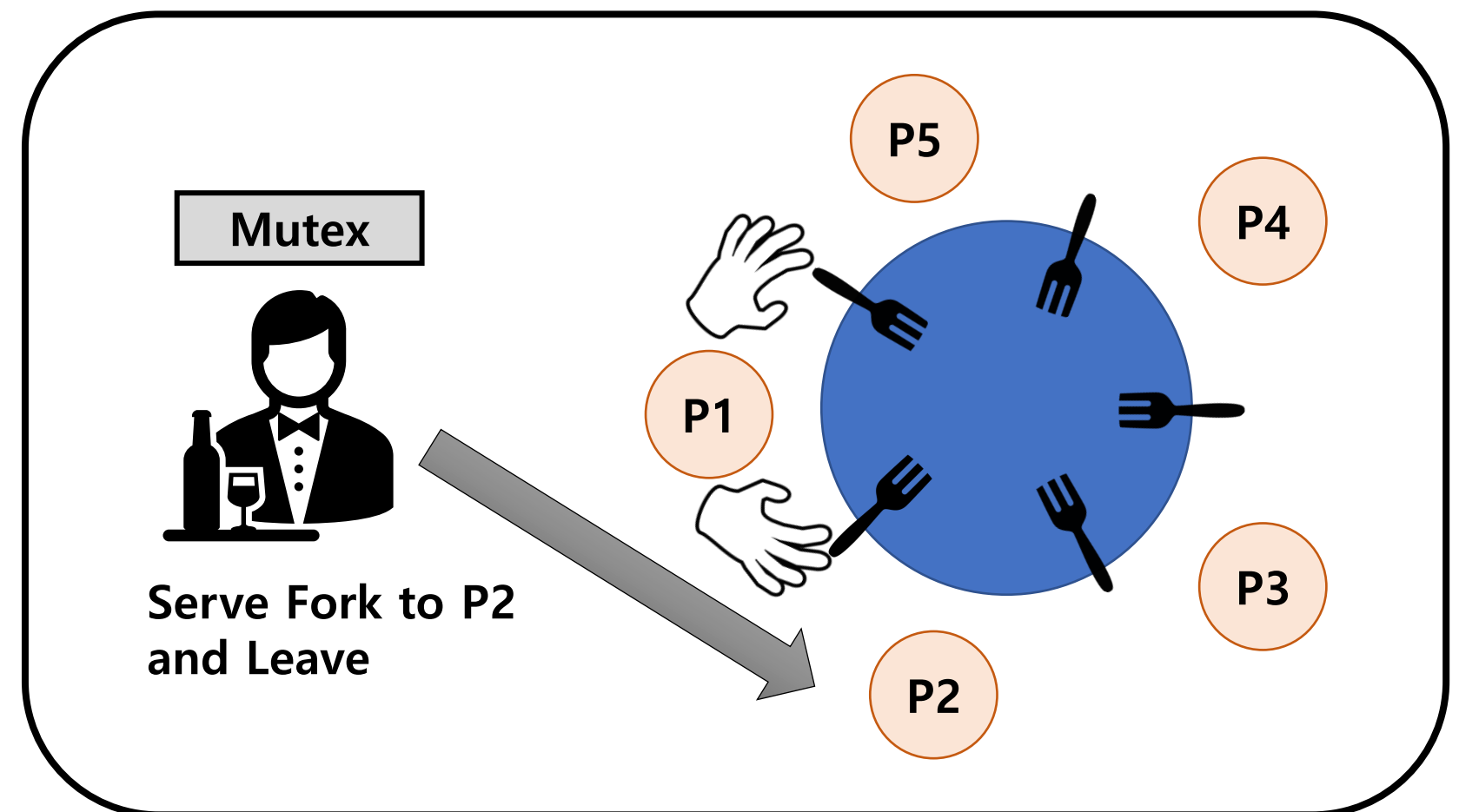
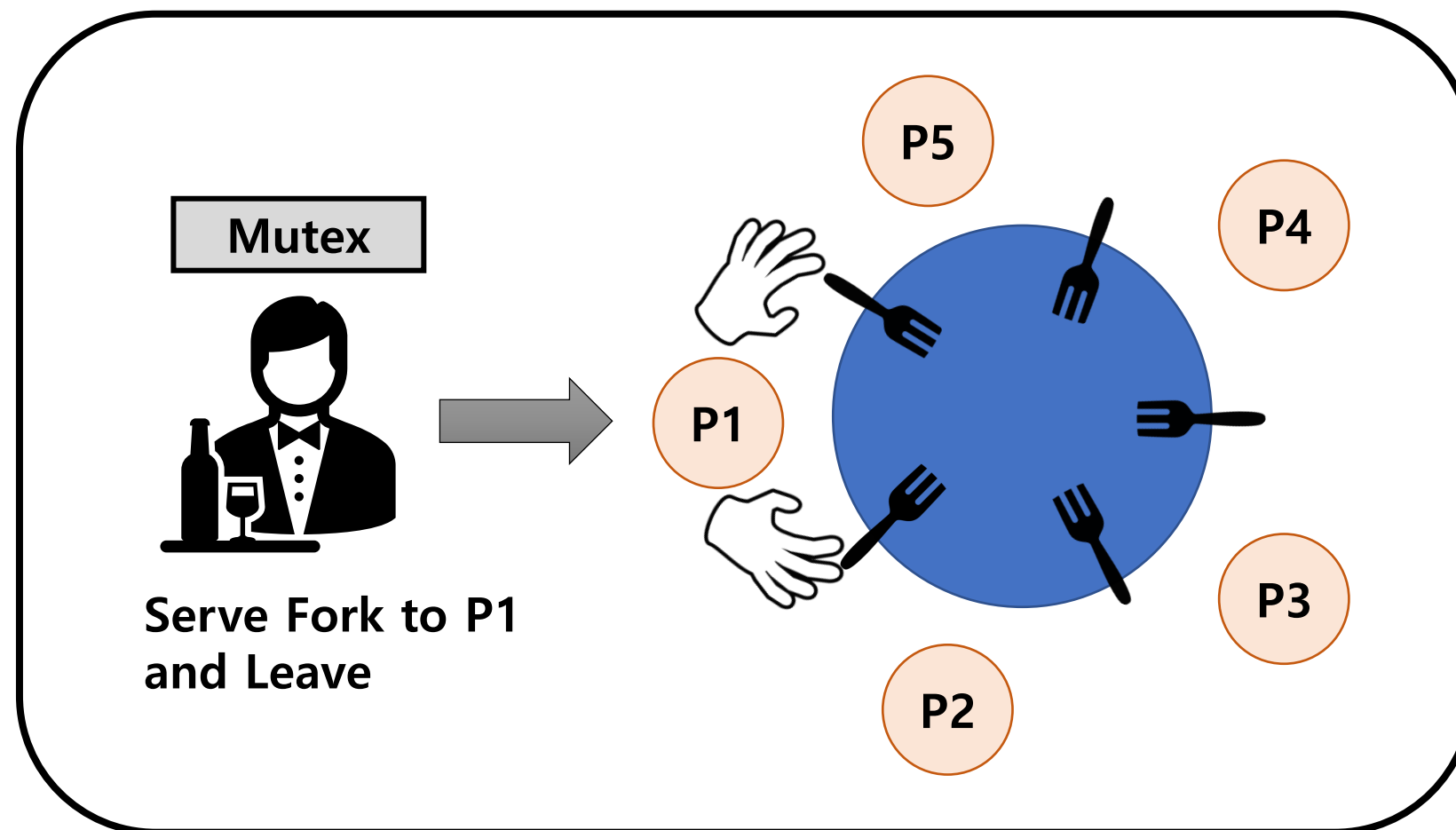
Starvation (기아상태)

- Starvation이 발생할 수도 있음
 - 어떤 철학자는 식사하지 못하고, 계속 기다릴 수도 있음



Solution 3 : Use of Arbitrator

- 철학자는 웨이터가 도착할 때까지 기다림
- 웨이터가 철학자에게 왔을 때만 철학자는 포크를 집을 수 있음



Solution 3 : Use of Arbitrator

- 각 철학자가 하는 일

```
do {  
    mutex_lock(mutex);           // 웨이터 서빙을 기다림  
    sem_lock(fork[i]);           // 왼쪽 포크를 집음  
    sem_lock(fork[(i+1)%N]);      // 오른쪽 포크를 집음  
    mutex_unlock(mutex);         // 웨이터가 서빙을 마치고 떠남  
  
    // Critical Section 진입  
    Eat();  
  
    sem_unlock(fork[i]);         // 왼쪽 포크를 내려놓음  
    sem_unlock(fork[(i+1)%N]);   // 오른쪽 포크를 내려놓음  
  
    // 다음 Critical Section 진입까지 대기  
    Think();  
} while(true);
```

PSEUDO CODE

Solution 3 : Use of Arbitrator

week10/solution_3/use_of_arbitrator.c

```
103 int main() {
104     pthread_t thread_id[N];
105
106     pthread_mutex_init(&print_mutex, NULL);
107
108     /* Initialize the waiter */
109     pthread_mutex_init(&mutex, NULL);
110
111     for (int i = 0; i < N; i++) {
112         /* Initialize each forks with 1 */
113         sem_init(&forks[i], 0,
114                 1); // Assign semaphore as a binary value (0 or 1) (== mutex)
115         state[i] = 0;
116     }
117
118     print_table_index();
119
120     for (int i = 0; i < N; i++) {
121         name[i] = i;
122         pthread_create(&thread_id[i], NULL, philosopher, &name[i]);
123     }
124
125     for (int i = 0; i < N; i++) {
126         pthread_join(thread_id[i], NULL);
127     }
128
129     for (int i = 0; i < N; i++) {
130         /* Destroy each forks */
131         sem_destroy(&forks[i]);
132     }
133
134     pthread_mutex_destroy(&print_mutex);
135     /* Destroy the waiter */
136     pthread_mutex_destroy(&mutex);
137 }
```

```
71 void *philosopher(void *_name) {
72     int phil = *((int *)_name);
73
74     do {
75         /* Wait for the waiter */
76         pthread_mutex_lock(&mutex);
77         print_arbitrator(phil, "serves");
78
79         sem_wait(&forks[(phil + 1) % N]);
80         print_fork(phil, (phil + 1) % N, "take");
81         sem_wait(&forks[phil]);
82         print_fork(phil, phil, "take");
83
84         /* The waiter leaves */
85         print_arbitrator(phil, "leaves");
86         pthread_mutex_unlock(&mutex);
87
88         update_state(phil, EATING);
89
90         sleep(1);
91
92         update_state(phil, THINKING);
93
94         print_fork(phil, (phil + 1) % N, "put");
95         sem_post(&forks[(phil + 1) % N]);
96         print_fork(phil, phil, "put");
97         sem_post(&forks[phil]);
98
99         sleep(1);
100     } while (1);
101 }
```

Solution 3 : 실행 결과

```
pro@pro-virtual-machine:~/os-week/week10/solution_3$ ./use_of_arbitrator
=====
| PHIL[0] || PHIL[1] || PHIL[2] || PHIL[3] || PHIL[4] |
=====
The waiter serves PHIL[0]
  PHIL[0] take 1 th fork.
  PHIL[0] take 0 th fork.
The waiter leaves PHIL[0]
  EATING | INIT || INIT || INIT || INIT |
The waiter serves PHIL[1]
  PHIL[1] take 2 th fork.
  THINKING || INIT || INIT || INIT || INIT |
  PHIL[0] put 1 th fork.
  PHIL[0] put 0 th fork.
  PHIL[1] take 1 th fork.
The waiter leaves PHIL[1]
  THINKING | EATING | INIT || INIT || INIT |
The waiter serves PHIL[2]
  PHIL[2] take 3 th fork.
  THINKING || THINKING || INIT || INIT || INIT |
  PHIL[1] put 2 th fork.
  PHIL[1] put 1 th fork.
  PHIL[2] take 2 th fork.
The waiter leaves PHIL[2]
  THINKING || THINKING | EATING | INIT || INIT |
The waiter serves PHIL[3]
  PHIL[3] take 4 th fork.
  THINKING || THINKING || THINKING || INIT || INIT |
  PHIL[2] put 3 th fork.
  PHIL[2] put 2 th fork.
  PHIL[3] take 3 th fork.
The waiter leaves PHIL[3]
  THINKING || THINKING || THINKING | EATING | INIT |
The waiter serves PHIL[4]
  PHIL[4] take 0 th fork.
  THINKING || THINKING || THINKING || THINKING || INIT |
  PHIL[3] put 4 th fork.
  PHIL[3] put 3 th fork.
  PHIL[4] take 4 th fork.
The waiter leaves PHIL[4]
  THINKING || THINKING || THINKING || THINKING || EATING |
```

동시에 최대 2명이 식사할 수 있으나,
웨이터를 기다리느라 그렇게 하지 못함

Solution 4 : Tanenbaum's Solution



- Hungry 상태 추가

- Hungry 상태를 추가해, 양 옆 포크가 사용 가능할 때만 포크를 집음

Solution 4 : Tanenbaum's Solution

상태				
포크 	사용	← 대응 →	Locked	공유자원
	미사용		Unlocked	
철학자 	먹는다		Running	프로세스, 스레드
	생각한다		Waiting	
	배고파한다		Ready	

Status	Description
Eating	When philosopher has got both the forks, i.e., he/she has entered the section.
Thinking	When philosopher doesn't want to gain access to either fork.
Hungry	When philosopher wants to enter the critical section.

Solution 4 : Tanenbaum's Solution

- 철학자 i가 배고픈 상태가 되어,
자신이 식사 가능한 상태인지
알고 싶어서 확인함 : test(i)

각 철학자 i가 하는 일

```
do {  
    // THINKING  
    take_forks(i);  
    // EATING  
    drop_forks(i);  
} while(true);
```

- 일정 시간 생각 후에,
배가고파 포크를 들려고 함
- 일정 시간 식사를 한 후,
포크를 놓으려고 함

```
take_forks(i) {  
    mutex_lock(mutex);  
    state[i] = HUNGRY;  
    test(i);  
    mutex_unlock(mutex);  
    sem_lock(F[i]);  
}
```

```
drop_forks(i) {  
    mutex_lock(mutex);  
    state[i] = THINKING;  
    test(LEFT);  
    test(RIGHT);  
    mutex_unlock(mutex);  
}
```

- 철학자 i는 포크를 놓고 생각에
잠기면서 양 옆의 철학자 들에게
식사가 가능하면 하라고 함

- 식사가 가능한지 검사하고,
가능하면 식사를 하도록 함

```
test(i) {  
    if (state[i] == HUNGRY  
        && state[LEFT] != EATING  
        && state[RIGHT] != EATING) {  
        state[i] = EATING;  
        sem_unlock(F[i]);  
    }  
}
```

PSEUDO CODE

Solution 4 : Tanenbaum's Solution

week10/solution_4/tanenbaum.c

```
94 void *philosopher(void *_name) {
95     int phil = *((int *)_name);
96
97     do {
98         sleep(1); /* Thinking */
99         take_forks(phil);
100        sleep(1); /* Eating */
101        drop_forks(phil);
102    } while (1);
103
104    pthread_exit(0);
105 }
106
```

실습 : TODO 채우기

```
64 /* Test if the philosopher can eat */
65 void test(int phil) {
66     if (state[phil] == HUNGRY && state[LEFT] != EATING &&
67         state[RIGHT] != EATING) {
68         update_state(phil, EATING);
69         /* TODO 3: Drop the forks */
70     }
71 }
72
73 void take_forks(int phil) {
74     /* TODO 4: Wait for the waiter */
75
76     update_state(phil, HUNGRY);
77     test(phil);
78
79     /* TODO 5: The waiter leaves */
80
81     /* TODO 6: Take the forks */
82 }
83
84 void drop_forks(int phil) {
85     /* TODO 7: Wait for the waiter */
86
87     update_state(phil, THINKING);
88     test(LEFT);
89     test(RIGHT);
90
91     /* TODO 8: The waiter leaves */
92 }
```

Solution 4 : 실행 결과

pro@pro-virtual-machine:~/os-week/week10/solution_4\$./tanenbaum_ans

PHIL[0]	PHIL[1]	PHIL[2]	PHIL[3]		PHIL[4]
HUNGRY	INIT	INIT	INIT		INIT
EATING	INIT	INIT	INIT		INIT
EATING	INIT	*HUNGRY*	INIT		INIT
EATING	INIT	EATING	INIT		INIT
EATING	*HUNGRY*	EATING	INIT		INIT
EATING	*HUNGRY*	EATING	INIT		INIT
EATING	*HUNGRY*	EATING	*HUNGRY*		INIT
THINKING	*HUNGRY*	EATING			
THINKING	*HUNGRY*	THINKING			
THINKING	EATING	THINKING	*HUNGRY*		EATING
THINKING	THINKING	THINKING	*HUNGRY*		EATING
THINKING	THINKING	*HUNGRY*	*HUNGRY*		EATING
THINKING	THINKING	EATING	*HUNGRY*		EATING
HUNGRY	THINKING	EATING	*HUNGRY*		EATING
HUNGRY	THINKING	EATING	*HUNGRY*		THINKING
EATING	THINKING	EATING	*HUNGRY*		THINKING
THINKING	THINKING	EATING	*HUNGRY*		THINKING
THINKING	THINKING	THINKING	*HUNGRY*		THINKING
THINKING	THINKING	THINKING	EATING		THINKING
THINKING	THINKING	THINKING	EATING		*HUNGRY*
THINKING	*HUNGRY*	THINKING	EATING		*HUNGRY*
THINKING	EATING	THINKING	EATING		*HUNGRY*
HUNGRY	EATING	THINKING	EATING		*HUNGRY*
HUNGRY	EATING	*HUNGRY*	EATING		*HUNGRY*
HUNGRY	THINKING	*HUNGRY*	EATING		*HUNGRY*
EATING	THINKING	*HUNGRY*	EATING		*HUNGRY*
EATING	THINKING	*HUNGRY*	THINKING		*HUNGRY*
EATING	THINKING	EATING	THINKING		*HUNGRY*
THINKING	THINKING	EATING	THINKING		EATING
THINKING	THINKING	THINKING	THINKING		EATING
THINKING	THINKING	THINKING	THINKING		EATING
THINKING	*HUNGRY*	THINKING	THINKING		EATING
THINKING	EATING	THINKING	*HUNGRY*		EATING
THINKING	EATING	THINKING	*HUNGRY*		EATING
HUNGRY	EATING	THINKING	*HUNGRY*		EATING
HUNGRY	THINKING	THINKING	*HUNGRY*		EATING
HUNGRY	THINKING	THINKING	*HUNGRY*		EATING
HUNGRY	THINKING	EATING	*HUNGRY*		EATING
HUNGRY	THINKING	EATING	*HUNGRY*		THINKING
EATING	THINKING	EATING	*HUNGRY*		THINKING

최대 2명의 철학자가 동시에 식사하는 모습



CHAPTER 3. Solution Testing

- 측정 기준
- Solution 3 testing 결과
- Solution 4 testing 결과

측정 단위

Operating System Concepts 9th p.205 5.2, 10th p.204 5.2

• 스케줄링 기준(Scheduling Criteria)

1. CPU 이용률(utilization) 극대화
2. 처리량(throughput) 극대화
3. 총 처리 시간(turnaround time) 최소화
4. 대기 시간(waiting time) 최소화
5. 응답 시간(response time) 최소화

• 본 솔루션을 평가하는 기준

1. 최대한 많은 철학자가 동시에 식사할 수 있다.
2. 최대한 많은 포크를 사용할 수 있다.
3. 모두에게 한번씩 대접하는 시간을 최소화 한다.

- 한번에 사용할 수 있는 최대 포크 개수

$$F = \lfloor \frac{f}{2} \rfloor$$

F : 포크 한 쌍의 수
 f : 포크의 수
 N : 철학자의 수

- 동시에 최대한 많은 철학자들이 식사할 수 있을 때, 걸리는 시간

$$t_{unit} = \lceil \frac{N}{F} \rceil + 2$$

측정 단위

이 기간 안에 반드시 1회 이상 식사를 제공받아야 한다고 가정

Context switching overhead
(문맥교환 오버헤드)

측정 단위 함수

```
60 int evaluate_time_unit() {  
61     int f_i = N / 2; // 동시에 사용할 수 있는 포크 한쌍의 수 (N 만큼 포크가 제공됨)  
62     float n = N; // 철학자의 수 (소수점 자리 계산이 가능하도록 타입 변환)  
63     float f = f_i; // 포크 한쌍의 수 (소수점 자리 계산이 가능하도록 타입 변환)  
64     return ceil(n / f) + 2; // 시간 단위 계산하여 반환  
65 }
```

RETURN VALUE : 측정 시간 단위 (t_{unit})

- 철학자가 10명이라고 가정 ($N = f = 10$)

$$F = \lfloor \frac{f}{2} \rfloor \longrightarrow \text{포크 한 쌍의 수 } (F) = \underline{5}$$

$$t_{unit} = \lceil \frac{N}{F} \rceil + 2 \longrightarrow \text{측정 시간 단위 } (t_{unit}) = 2+2 = \underline{4}$$

측정 방법

```
68 void *testing_solution(void *data) {
69     struct timeval start_time, curr_time;
70     int time_unit = evaluate_time_unit();
71     while (1) {
72         // eat_count_unit 초기화
73         for (int i = 0; i < N; i++) {
74             eat_count_unit[i] = 0;
75         }
76
77         // 시간 단위만큼 기다림
78         gettimeofday(&start_time, NULL);
79         while (curr_time.tv_sec - start_time.tv_sec < time_unit) {
80             gettimeofday(&curr_time, NULL);
81         }
82
83         // 시간 단위 동안 철학자 모두가 식사를 했는지 검사하고 아니면 프로그램 종료
84         for (int i = 0; i < N; i++) {
85             if (eat_count_unit[i] == 0) {
86                 printf("%s\n\n**** Failed to Satisfy the Time Unit ****\n\n%s", KRED,
87                     KNRM);
88                 exit(0);
89             }
90         }
91         print_eat_count();
92     }
93     pthread_exit(0);
94 }
```

→ 시간 단위(t_{unit}) 마다의 식사 횟수 초기화

→ 시간 단위(t_{unit}) 만큼 대기

→ 시간 단위(t_{unit}) 안에
모든 철학자가 식사를 하였는지 검사

↓
한 명이라도 식사를 못했을 경우,
시간 단위를 만족하지 못했으므로
프로그램 종료

측정 방법

week10/solution_testing/use_of_arbitrator.c

```
135 void *philosopher(void *_name) {
136     int phil = *((int *)_name);
137
138     do {
139         /* Wait for the waiter */
140         pthread_mutex_lock(&mutex);
141         print_arbitrator(phil, "serves");
142
143         sem_wait(&forks[(phil + 1) % N]);
144         print_fork(phil, (phil + 1) % N, "take");
145         sem_wait(&forks[phil]);
146         print_fork(phil, phil, "take");
147
148         /* The waiter leaves */
149         pthread_mutex_unlock(&mutex);
150         print_arbitrator(phil, "leaves");
151
152         update_state(phil, EATING);
153
154         /* FIX 6 : 식사를 할 때마다 카운터 증가 */
155         eat_count[phil]++;
156         eat_count_unit[phil]++;
157
158         sleep(1);
159
160         update_state(phil, THINKING);
161
162         print_fork(phil, (phil + 1) % N, "put");
163         sem_post(&forks[(phil + 1) % N]);
164         print_fork(phil, phil, "put");
165         sem_post(&forks[phil]);
166
167         sleep(1);
168     } while (1);
169 }
```

week10/solution_testing/tanenbaum.c

```
126 /* Test if the philosopher can eat */
127 void test(int phil) {
128     if (state[phil] == HUNGRY && state[LEFT] != EATING &&
129         state[RIGHT] != EATING) {
130
131         update_state(phil, EATING);
132
133         /* FIX 5 : 식사를 할 때마다 카운터 증가 */
134         eat_count[phil]++;
135         eat_count_unit[phil]++;
136
137         /* TODO 3: Drop the forks */
138         sem_post(&F[phil]);
139     }
140 }
```

- 철학자가 식사를 할 때마다 카운터 증가
 - eat_count[] : Total Count
 - eat_count_unit[] : Count per Time Unit

Solution 3 Testing 결과

$N = 10$, $t_{unit} = 4$

```
pro@pro-virtual-machine:~/os-week/week10/solution_testing$ ./use_of_arbitrator
=====
| PHIL[0] || PHIL[1] || PHIL[2] || PHIL[3] || PHIL[4] || PHIL[5] || PHIL[6] || PHIL[7] || PHIL[8] || PHIL[9] |
=====
The waiter serves PHIL[0]
  PHIL[0] take 1 th fork.
  PHIL[0] take 0 th fork.
The waiter leaves PHIL[0]
| EATING || INIT || INIT || INIT || INIT || INIT || INIT || INIT || INIT || INIT |
The waiter serves PHIL[1]
  PHIL[1] take 2 th fork.
| THINKING || INIT || INIT || INIT || INIT || INIT || INIT || INIT || INIT || INIT |
  PHIL[0] put 1 th fork.
  PHIL[0] put 0 th fork.
  PHIL[1] take 1 th fork.
The waiter leaves PHIL[1]
| THINKING || EATING || INIT || INIT || INIT || INIT || INIT || INIT || INIT || INIT |
The waiter serves PHIL[2]
  PHIL[2] take 3 th fork.
| THINKING || THINKING || INIT || INIT || INIT || INIT || INIT || INIT || INIT || INIT |
  PHIL[1] put 2 th fork.
  PHIL[1] put 1 th fork.
  PHIL[2] take 2 th fork.
The waiter leaves PHIL[2]
| THINKING || THINKING || EATING || INIT || INIT || INIT || INIT || INIT || INIT || INIT |
The waiter serves PHIL[3]
  PHIL[3] take 4 th fork.
| THINKING || THINKING || THINKING || INIT || INIT || INIT || INIT || INIT || INIT || INIT |
  PHIL[2] put 3 th fork.
  PHIL[2] put 2 th fork.
  PHIL[3] take 3 th fork.
The waiter leaves PHIL[3]
| THINKING || THINKING || THINKING || EATING || INIT || INIT || INIT || INIT || INIT || INIT |
The waiter serves PHIL[4]
  PHIL[4] take 5 th fork.
**** Failed to Satisfy the Time Unit ****
```

**** Failed to Satisfy the Time Unit ****

시간 단위 테스트를 통과하지 못하고 종료함

Solution 4 Testing 결과

$N = 10$, $t_{unit} = 4$

```
pro@pro-virtual-machine:~/os-week/week10/solution_testing$ ./tanenbaum
```

PHIL[0]	PHIL[1]	PHIL[2]	PHIL[3]	PHIL[4]	PHIL[5]	PHIL[6]	PHIL[7]	PHIL[8]	PHIL[9]
HUNGRY	INIT	INIT	INIT	INIT	INIT	INIT	INIT	INIT	INIT
EATING	INIT	INIT	INIT	INIT	INIT	INIT	INIT	INIT	INIT
EATING	*HUNGRY*	INIT	INIT	INIT	INIT	INIT	INIT	INIT	INIT
EATING	*HUNGRY*	*HUNGRY*	INIT	INIT	INIT	INIT	INIT	INIT	INIT
EATING	*HUNGRY*	EATING	INIT	INIT	INIT	INIT	INIT	INIT	INIT
EATING	*HUNGRY*	EATING	*HUNGRY*	INIT	INIT	INIT	INIT	INIT	INIT
EATING	*HUNGRY*	EATING	*HUNGRY*	*HUNGRY*	INIT	INIT	INIT	INIT	INIT
EATING	*HUNGRY*	EATING	*HUNGRY*	EATING	INIT	INIT	INIT	INIT	INIT
EATING	*HUNGRY*	EATING	*HUNGRY*	EATING	INIT	INIT	*HUNGRY*	INIT	INIT
EATING	*HUNGRY*	EATING	*HUNGRY*	EATING	INIT	INIT	EATING	INIT	INIT
EATING	*HUNGRY*	EATING	*HUNGRY*	EATING	INIT	INIT	EATING	*HUNGRY*	INIT
EATING	*HUNGRY*	EATING	*HUNGRY*	EATING	*HUNGRY*	INIT	EATING	*HUNGRY*	*HUNGRY*
THINKING	*HUNGRY*	EATING	*HUNGRY*	EATING	*HUNGRY*	*HUNGRY*	EATING	*HUNGRY*	*HUNGRY*
THINKING	*HUNGRY*	EATING	*HUNGRY*	EATING	*HUNGRY*	*HUNGRY*	EATING	*HUNGRY*	*HUNGRY*
THINKING	*HUNGRY*	THINKING	*HUNGRY*	EATING	*HUNGRY*	*HUNGRY*	EATING	*HUNGRY*	*HUNGRY*
THINKING	EATING	THINKING	*HUNGRY*	EATING	*HUNGRY*	*HUNGRY*	EATING	*HUNGRY*	EATING
THINKING	EATING	THINKING	*HUNGRY*	THINKING	*HUNGRY*	*HUNGRY*	EATING	*HUNGRY*	EATING
THINKING	EATING	THINKING	EATING	THINKING	*HUNGRY*	*HUNGRY*	EATING	*HUNGRY*	EATING
THINKING	EATING	THINKING	EATING	THINKING	EATING	*HUNGRY*	EATING	*HUNGRY*	EATING
THINKING	EATING	THINKING	EATING	THINKING	EATING	*HUNGRY*	THINKING	*HUNGRY*	EATING
HUNGRY	EATING	THINKING	EATING	THINKING	EATING	*HUNGRY*	THINKING	*HUNGRY*	THINKING
HUNGRY	EATING	THINKING	EATING	THINKING	EATING	*HUNGRY*	THINKING	*HUNGRY*	THINKING
HUNGRY	EATING	THINKING	EATING	THINKING	EATING	*HUNGRY*	THINKING	*HUNGRY*	THINKING
HUNGRY	THINKING	*HUNGRY*	EATING	THINKING	EATING	*HUNGRY*	THINKING	EATING	THINKING
EATING	THINKING	*HUNGRY*	EATING	THINKING	EATING	*HUNGRY*	THINKING	EATING	THINKING
EATING	THINKING	*HUNGRY*	EATING	*HUNGRY*	THINKING	*HUNGRY*	THINKING	EATING	THINKING
EATING	THINKING	*HUNGRY*	EATING	*HUNGRY*	THINKING	*HUNGRY*	THINKING	EATING	THINKING
EATING	THINKING	*HUNGRY*	EATING	*HUNGRY*	THINKING	EATING	THINKING	EATING	THINKING
EATING	THINKING	*HUNGRY*	THINKING	*HUNGRY*	THINKING	EATING	THINKING	EATING	THINKING
EATING	THINKING	EATING	THINKING	*HUNGRY*	THINKING	EATING	THINKING	EATING	THINKING
EATING	THINKING	EATING	THINKING	EATING	THINKING	EATING	THINKING	EATING	THINKING
EATING	THINKING	EATING	THINKING	EATING	THINKING	EATING	*HUNGRY*	EATING	THINKING

```

== Unit Eat Count ==
|PHIL[2]|PHIL[1]|PHIL[2]|PHIL[1]|PHIL[2]|PHIL[1]|PHIL[1]|PHIL[1]|PHIL[1]|PHIL[1]|
== Total Eat Count ==
|PHIL[2]|PHIL[1]|PHIL[2]|PHIL[1]|PHIL[2]|PHIL[1]|PHIL[1]|PHIL[1]|PHIL[1]|PHIL[1]|

```

EATING	*HUNGRY*	EATING	THINKING	EATING	THINKING	EATING	*HUNGRY*	EATING	THINKING
EATING	*HUNGRY*	EATING	THINKING	THINKING	THINKING	EATING	*HUNGRY*	EATING	THINKING
EATING	*HUNGRY*	EATING	THINKING	THINKING	*HUNGRY*	EATING	*HUNGRY*	EATING	THINKING

시간 단위 테스트를 통과함

감사합니다.

CPS LAB

Lim Jiseoup

jseoup@hanyang.ac.kr