운영체제론 실습

- 식사하는 철학자 문제

목 차

예제 코드 다운로드 경로

아래 명령어를 linux 환경에서 치면 다운받을 수 있음.

$ wget http://ce.hanyang.ac.kr/week10.zip

$ unzip week10.zip

테스팅 코드

$ wget http://ce.hanyang.ac.kr/testing_solutions.zip

$ unzip testing_solutions.zip

프로세스 동기화

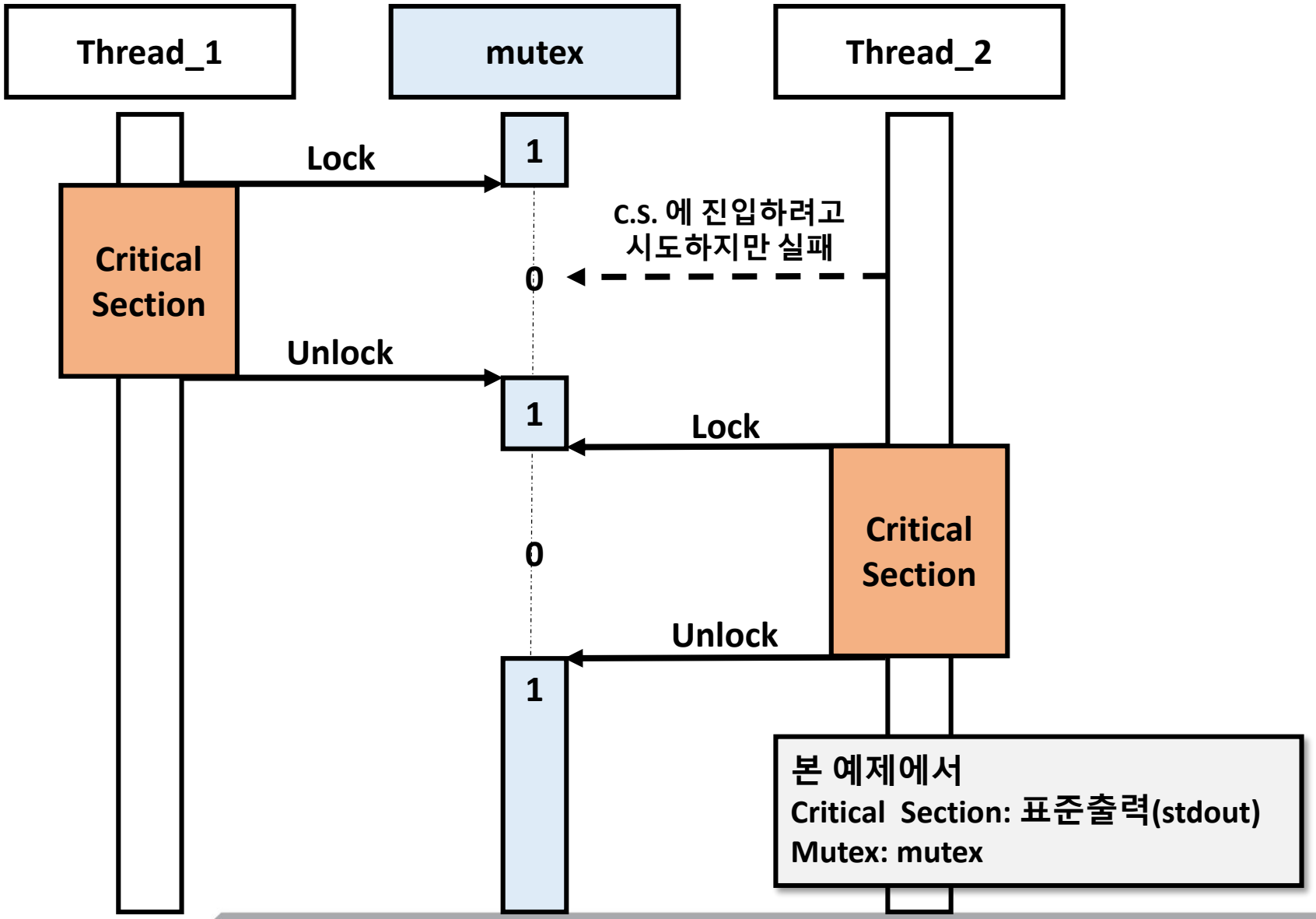
프로세스 동기화

- **상호 배제(mutual exclusion)**
  - 특정 프로세스가 임계구역에서 실행 중이면,
    다른 프로세스들은 자신들의 임계구역에서 실행될 수 없음

- **진행(progress)**
  - 임계구역에 아무 프로세스도 실행되고 있지 않고,
    그들 자신의 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

뮤텍스(Mutex)의 역할: 상호배제(Mutual Exclusion)



pthread_mutex_init() 사용법

```
#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.
  - *mutex* : 초기화 하고자 하는 mutex
  - *attr* : mutex 속성 (NULL 사용하면 기본값)

- **사용 예제**

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

pthread_mutex_lock() 사용법

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

- **mutex 를 lock** 한다.
  - *mutex* : lock 하고자 하는 mutex
- **사용 예제**

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

pthread_mutex_unlock() 사용법

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

- **mutex 를 unlock** 한다.
  - *mutex* : unlock 하고자 하는 mutex
- **사용 예제**

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

pthread_mutex_destroy() 사용법

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

- **mutex를 소멸시킴으로 mutex를 통해 할당할 수 있었던 공유 자원을 해방시킴**
  - *mutex* : 소멸시킬 mutex
- **사용 예제**

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

Example 1) print_mutex.c

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

Example 1) print_mutex.c

```
17 void *print_red(void *data) {
18   while (1) {
19     pthread_mutex_lock(&mutex);
20     printf("%sI am a red sentence.\n", KRED);
21     printf("I love red apples\n");
22     printf("and red strawberries\n");
23     printf("I'am so excited and angry\n");
24     printf("And I am trying to bother you all!!!!!!!!!!\n%s", KNRM
25     pthread_mutex_unlock(&mutex);
26     usleep(rand()%2);
27   }
28 }
29
30 void *print_blue(void *data) {
31   while (1) {
32     pthread_mutex_lock(&mutex);
33     printf("%sI am a blue sentence.\n", KBLU);
34     printf("I am always sad.....\n");
35     printf("That's why people say\n");
36     printf("\"I'm feeling blue\"\n");
37     printf("when they are sad....\n");
38     printf("And please leave me alone....\n%s", KNRM);
39     pthread_mutex_unlock(&mutex);
40     usleep(rand()%2);
41   }
42 }
43
44 void *print_green(void *data) {
45   while (1) {
46     pthread_mutex_lock(&mutex);
47     printf("%sI am a green sentence.\n", KGRN);
48     printf("I a piece of public or common grassy land,\n");
49     printf("especially in the center of a town.\n");
50     printf("Guys, please don't fight\n%s", KNRM);
51     pthread_mutex_unlock(&mutex);
52     usleep(rand()%2);
53   }
54 }
```

Critical Section

red가 Critical Section에 진입한 경우, blue와 green은 진입할 수 없음

경로: week10/mutex/print_no_mutex.c

Example 1) 결과화면 (mutex를 사용하지 않은 경우, 상호배제 위반)

```
I am a green sentence.
I a piece of public or common grassy land,
especially in the center of a town.
Guys, please don't fight
I am a blue sentence.
I am a green sentence.
I a piece of public or common grassy land,
especially in the center of a town.
Guys, please don't fight
I am a red sentence.
I am always sad.....
That's why people say
"I'm feeling blue"
when they are sad....
And please leave me alone....
I love red apples
and red strawberries
I'am so excited and angry
And I am trying to bother you all!!!!!!!!!!
I am a green sentence.
I a piece of public or common grassy land,
especially in the center of a town.
Guys, please don't fight
```

기존 print_blue 함수 출력내용

```
I am a blue sentence.
I am always sad.....
That's why people say
"I'm feeling blue"
when they are sad....
And please leave me alone....
```

가만히 내버려두길 바랬던 Blue 는 Red 와 Green 의 방해를 받음.

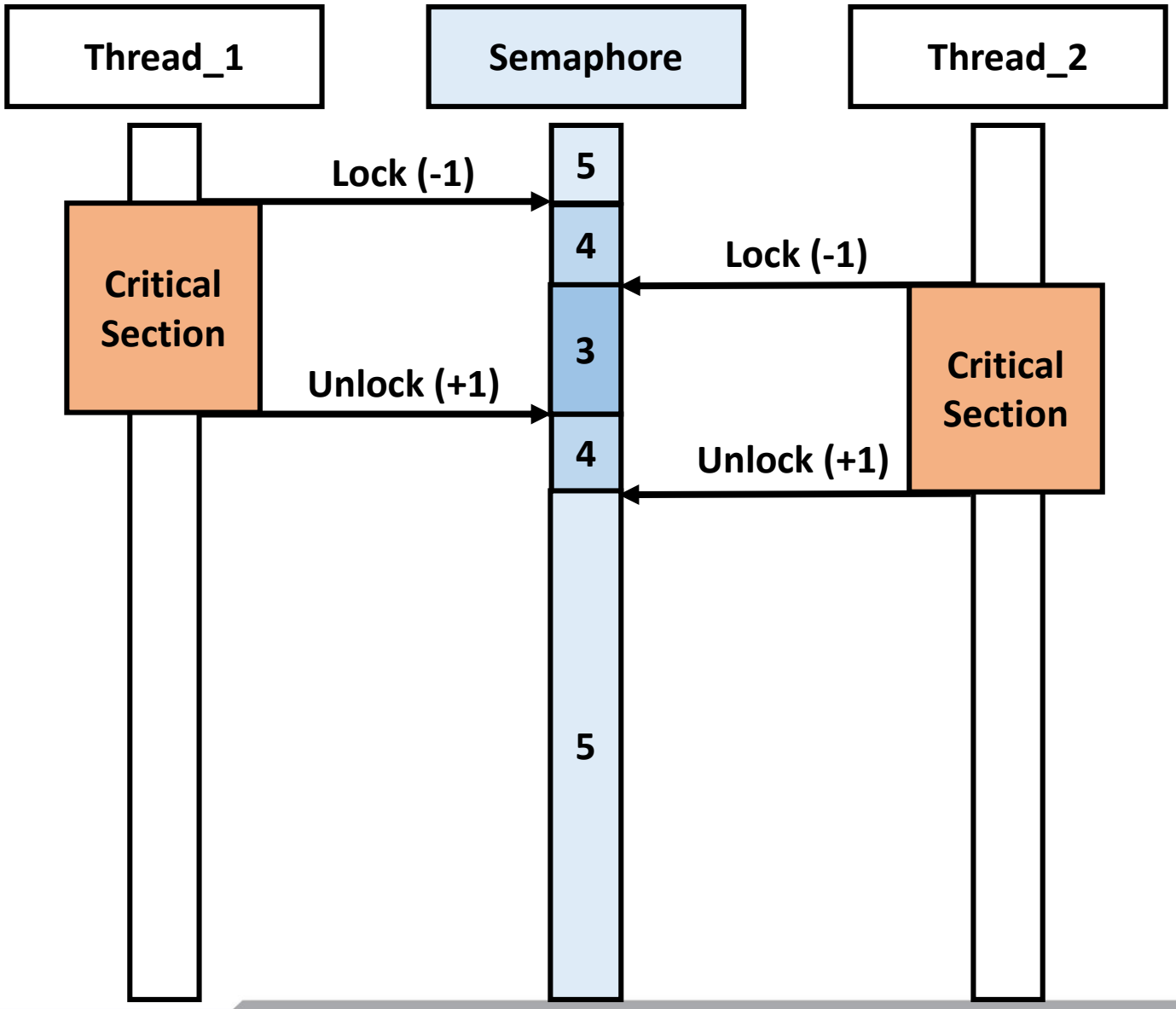
공유자원인 표준출력을 Critical Section으로 지정하고 각 thread가 내용을 전부 출력하기 전에는 다른 thread가 진입하지 않도록 하는 상호배제를 어떻게 구현해야 할까?

경로: week10/mutex/print_mutex.c

Example 1) 결과화면 (mutex를 사용한 경우)

```
I am a red sentence.
I love red apples
and red strawberries
I'am so excited and angry
And I am trying to bother you all!!!!!!!!!!
I am a blue sentence.
I am always sad.....
That's why people say
"I'm feeling blue"
when they are sad....
And please leave me alone....
I am a green sentence.
I a piece of public or common grassy land,
especially in the center of a town.
Guys, please don't fight
I am a red sentence.
I love red apples
and red strawberries
I'am so excited and angry
And I am trying to bother you all!!!!!!!!!!
I am a blue sentence.
I am always sad.....
That's why people say
"I'm feeling blue"
when they are sad....
And please leave me alone....
I am a green sentence.
I a piece of public or common grassy land,
especially in the center of a town.
Guys, please don't fight
```

세마포어(Semaphore)



sem_init() 사용법

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

- **value의 값을 가진 semaphore를 초기화**한다.
  - *sem* : 초기화 하고자 하는 semaphore
  - *pshared* : 0 값이면 스레드 간 공유되고, 아니면 프로세스 간 공유됨
  - *value* : semaphore의 초기값을 지정함

- **사용 예제**

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

sem_wait() 사용법

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

- **semaphore를 값을 1 감소**시킨다. (즉, **lock**을 수행함)
  - *sem* : lock 하고자 하는 semaphore
- **사용 예제**

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

sem_post() 사용법

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

- **semaphore를 값을 1 증가**시킨다. (즉, **unlock**을 수행함)
  - *sem* : unlock 하고자 하는 semaphore
- **사용 예제**

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

sem_destroy() 사용법

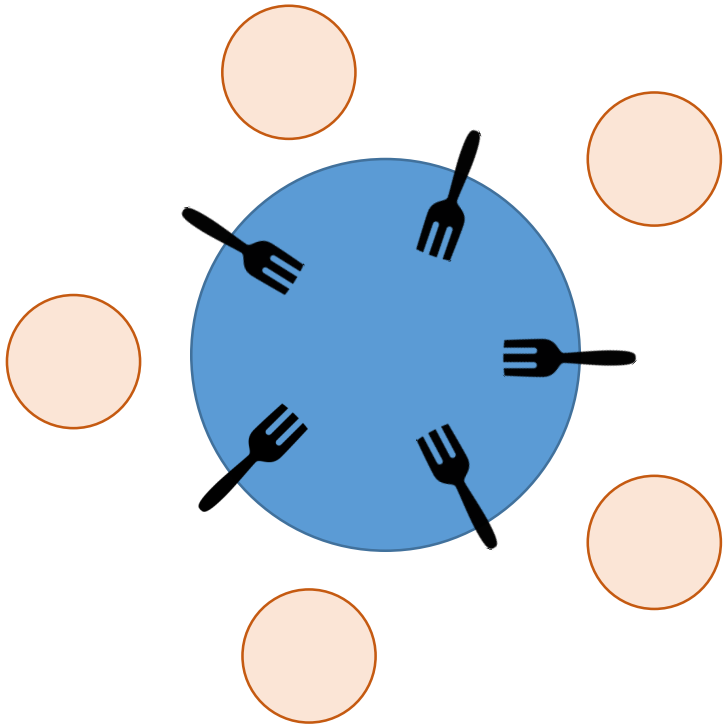
```
#include <semaphore.h>
int sem_destroy(sem_t *sem);
```

- **semaphore를 파괴**한다.
  - *sem* : 파괴하고자 하는 semaphore
- **사용 예제**

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

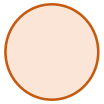
프로젝트:
식사하는 철학자 문제

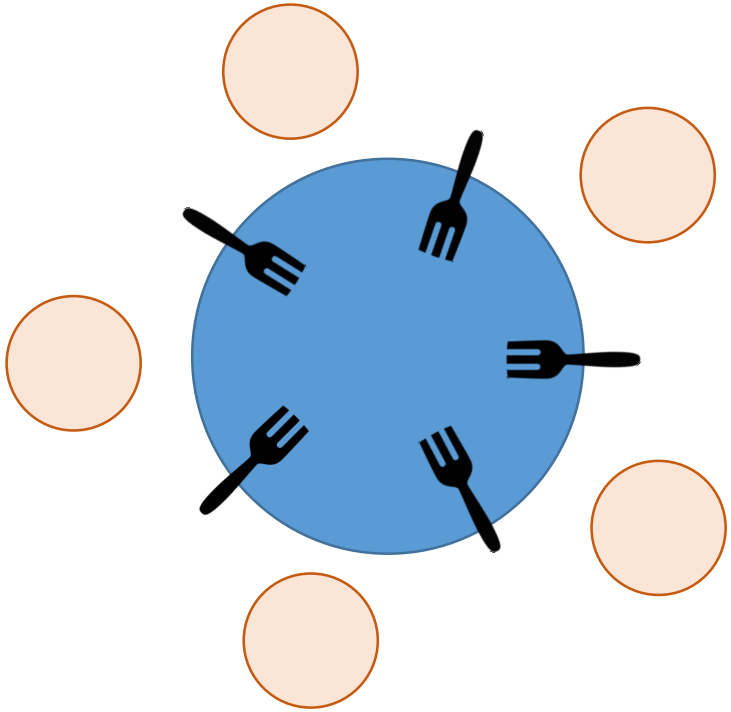
프로젝트: 식사하는 철학자 문제



- 원형 테이블에 5명의 사람과 5개의 포크가 있다. 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 생각한다.

프로젝트: 식사하는 철학자 문제

	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]);              // 오른쪽에 있는 포크를 집음
   lock(fork[(i+1)%N]);   // 왼쪽에 있는 포크를 집음

   // 임계구역(Critical Section) 진입
   Eat();

   unlock(fork[i]);            // 오른쪽에 있는 포크를 내려놓음
   unlock(fork[(i+1)%N]); // 왼쪽에 있는 포크를 내려놓음

   // 다음 임계구역 진입 전까지 대기
   Think();

}while(true);
```

Solution 1: Right chopstick first Solution

```
88 int main() {
89   pthread_t thread_id[N];
90
91   srand(time(NULL));
92
93   pthread_mutex_init(&print_mutex, NULL);
94
95   for (int i = 0; i < N; i++) {
96     sem_init(&forks[i], 0, 1);
97     state[i] = 0;
98   }
99
100  print_table_index();
101
102  for (int i = 0; i < N; i++) {
103    name[i] = i;
104    pthread_create(&thread_id[i], NULL, philosopher, &name[i]);
105  }
106
107  for (int i = 0; i < N; i++) {
108    pthread_join(thread_id[i], NULL);
109  }
110
111  for (int i = 0; i < N; i++) {
112    sem_destroy(&forks[i]);
113  }
114
115  pthread_mutex_destroy(&print_mutex);
116 }
```

세마포어(포크) 값 초기화

각 철학자 Thread 생성

Solution 1: Right chopstick first Solution

```
58 void *philosopher(void *_phil) {
59   int phil = *((int *)_phil);
60
61   do {
62     sem_wait(&forks[(phil + 1) % N]);
63     print_fork(phil, (phil + 1) % N);
64
65     // DEADLOCK: Every philosopher holds
66     // the right fork simultaneously
67     sleep(2);
68
69     sem_wait(&forks[phil]);
70     print_fork(phil, phil);
71
72     state[phil] = EATING;
73     print_phstates();
74
75     usleep(rand() % 1000000);
76
77     sem_post(&forks[(phil + 1) % N]);
78     sem_post(&forks[phil]);
79
80     state[phil] = THINKING;
81     print_phstates();
82     usleep(rand() % 2000000);
83
84   } while (1);
85   pthread_exit(0);
86 }
```

각 철학자가 하는 일

자신의 오른쪽에 있는 포크를 집는다.

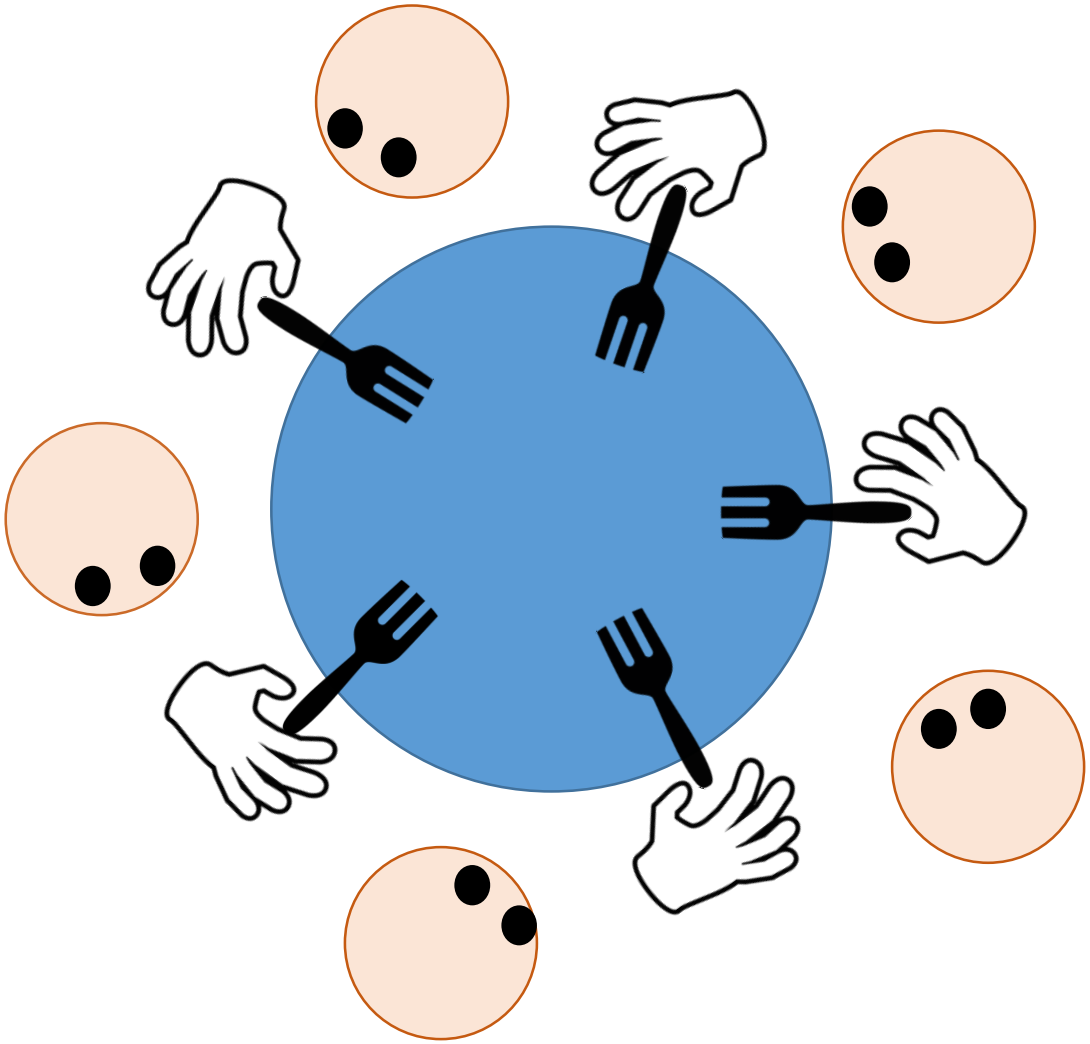
하지만 모든 철학자가 동시에
오른쪽 포크를 집기 때문에
어느 누구도 왼쪽 포크를
사용할 수 없다.
=> DEADLOCK 발생

Solution 1: 결과화면

```
jsbaik@jsbaik:~/OS2019/week10/solution_1$ make
gcc -g -o right_fork_first right_fork_first.c -lpthread
jsbaik@jsbaik:~/OS2019/week10/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



- **Deadlock 발생**

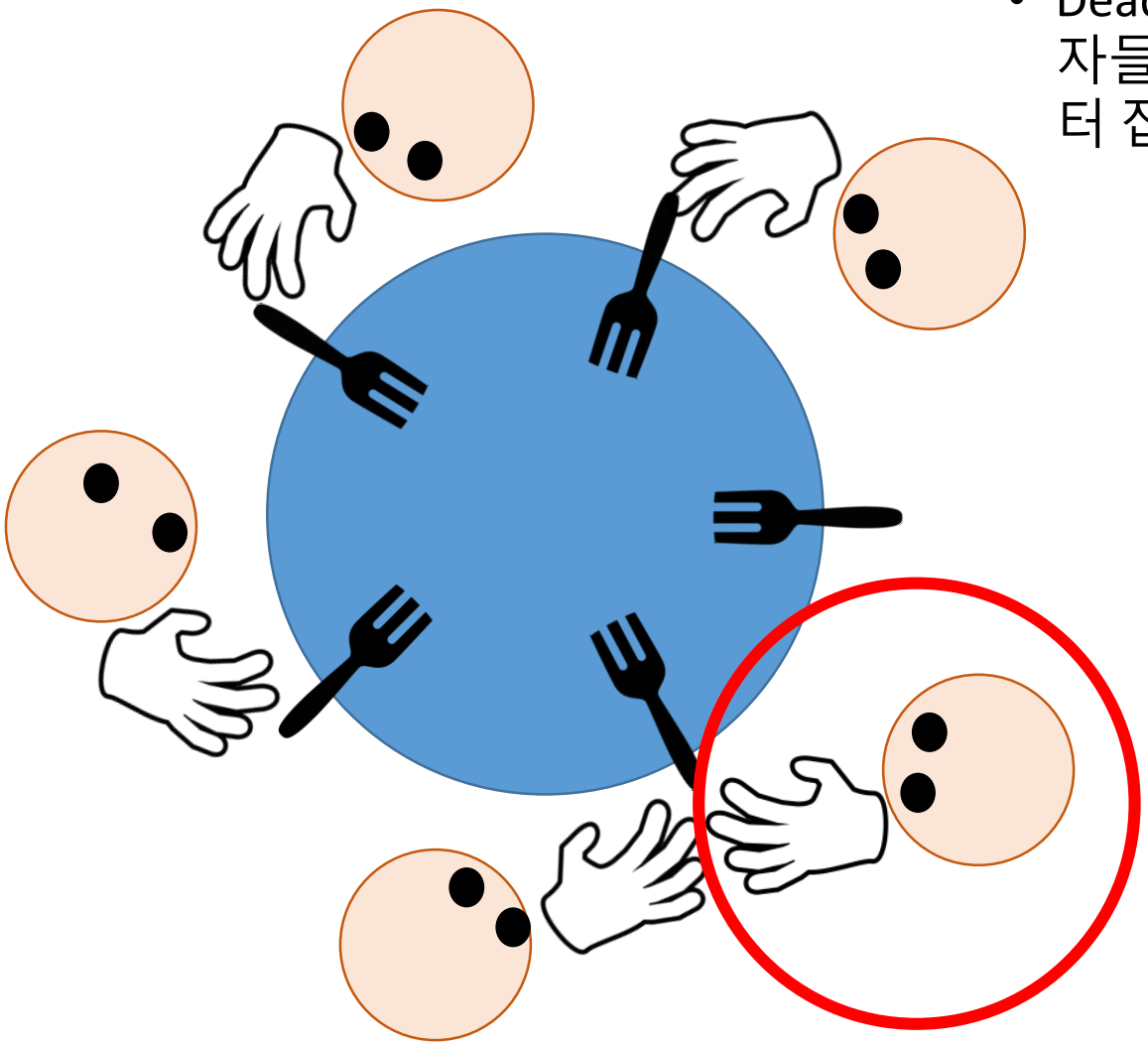
- 모든 철학자가 식사를 하지 못해 굶어 죽음

Deadlock이 발생할 조건

1. **상호배제 (Mutual exclusion)**
   : 한 명의 철학자가 포크를 가지고 있으면 다른 철학자는 가질 수 없음
2. **보유 및 대기 (Hold and wait)**
   : 한쪽 포크는 가지고 있는 상태에서 다른 쪽 포크를 기다리고 있는 경우
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 만족하지 않으면,
Deadlock이 발생하지 않음

Solution 2: Right-Left Solution



- Deadlock을 피하기 위해, 철학자들 중 하나는 포크를 왼쪽부터 잡게 함

Solution 2: Right-Left Solution

각 철학자가 하는 일

왼손 잡이

```
do {
   lock(fork[i]);
   lock(fork[(i+1)%N]);

   // 임계구역(Critical Section)
   Eat();

   unlock(fork[i]);
   unlock(fork[(i+1)%N]);

   Think();

}while(true);
```

오른손 잡이

```
do {
   lock(fork[(i+1)%N]);
   lock(fork[i]);

   // 임계구역(Critical Section)
   Eat();

   unlock(fork[(i+1)%N]);
   unlock(fork[i]);

   Think();

}while(true);
```

Solution 2: Right-Left Solution

```
115 int main() {
116   pthread_t thread_id[N];
117
118   srand(time(NULL));
119
120   /* Only one philosopher can use print_state at a time. */
121   pthread_mutex_init(&print_mutex, NULL);
122
123   for (int i = 0; i < N; i++) {
124     sem_init(&forks[i], 0, 1);
125     state[i] = 0;
126   }
127
128   print_table_index();
129
130   name[0] = 0;
131   pthread_create(&thread_id[0], NULL, left_handed_philosopher, &name[0]);
132   for (int i = 1; i < N; i++) {
133     name[i] = i;
134     pthread_create(&thread_id[i], NULL, right_handed_philosopher, &name[i]);
135   }
136
137   for (int i = 0; i < N; i++) {
138     pthread_join(thread_id[i], NULL);
139   }
140
141   for (int i = 0; i < N; i++) {
142     sem_destroy(&forks[i]);
143   }
144
145   pthread_mutex_destroy(&print_mutex);
146 }
```

Solution 2: Right-Left Solution

왼손 잡이

```
58 void *left_handed_philosopher(void *_phil) {
59   int phil = *((int *)_phil);
60
61   do {
62     sem_wait(&forks[phil]);
63     print_fork(phil, phil, "take");
64     sem_wait(&forks[(phil + 1) % N]);
65     print_fork(phil, (phil + 1) % N, "take");
66
67     state[phil] = EATING;
68     print_phstates();
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     state[phil] = THINKING;
77     print_phstates();
78
79     usleep(rand() % 2000000);
80
81   } while (1);
82
83   pthread_exit(0);
84 }
```

오른손 잡이

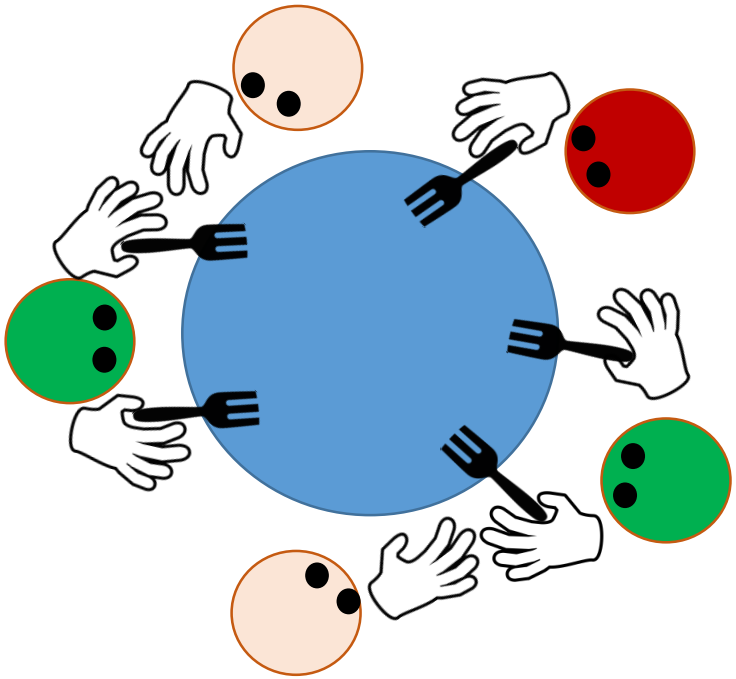
```
86 void *right_handed_philosopher(void *_phil) {
87   int phil = *((int *)_phil);
88
89   do {
90     sem_wait(&forks[(phil + 1) % N]);
91     print_fork(phil, (phil + 1) % N, "take");
92     sem_wait(&forks[phil]);
93     print_fork(phil, phil, "take");
94
95     state[phil] = EATING;
96     print_phstates();
97
98     usleep(rand() % 1000000);
99
100    sem_post(&forks[(phil + 1) % N]);
101    print_fork(phil, (phil + 1) % N, "put down");
102    sem_post(&forks[phil]);
103    print_fork(phil, phil, "put down");
104    state[phil] = THINKING;
105    print_phstates();
106
107    usleep(rand() % 2000000);
108
109  } while (1);
110
111  pthread_exit(0);
112 }
```

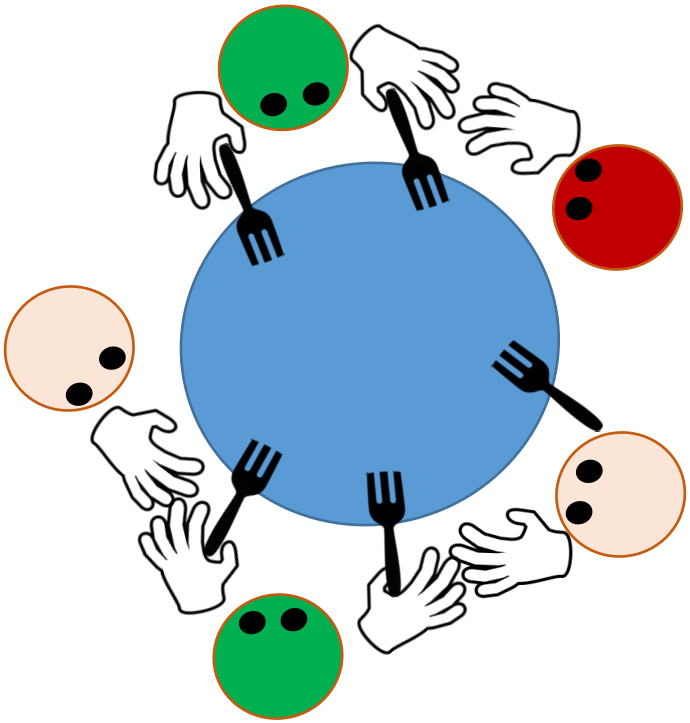
Solution 2: 결과화면

```
jsbaik@jsbaik:~/OS2019/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[2] take 3 th fork.
PHIL[1] take 2 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[3] take 3 th fork.
|       THINKING        ||      THINKING        ||      THINKING        ||      EATING          ||      INIT            |
PHIL[3] put down 4 th fork.
PHIL[3] put down 3 th fork.
|       THINKING        ||      THINKING        ||      THINKING        ||      THINKING        ||      INIT            |
PHIL[4] take 4 th fork.
|       THINKING        ||      THINKING        ||      THINKING        ||      THINKING        ||      EATING          |
PHIL[2] take 3 th fork.
```

Solution 2의 결과

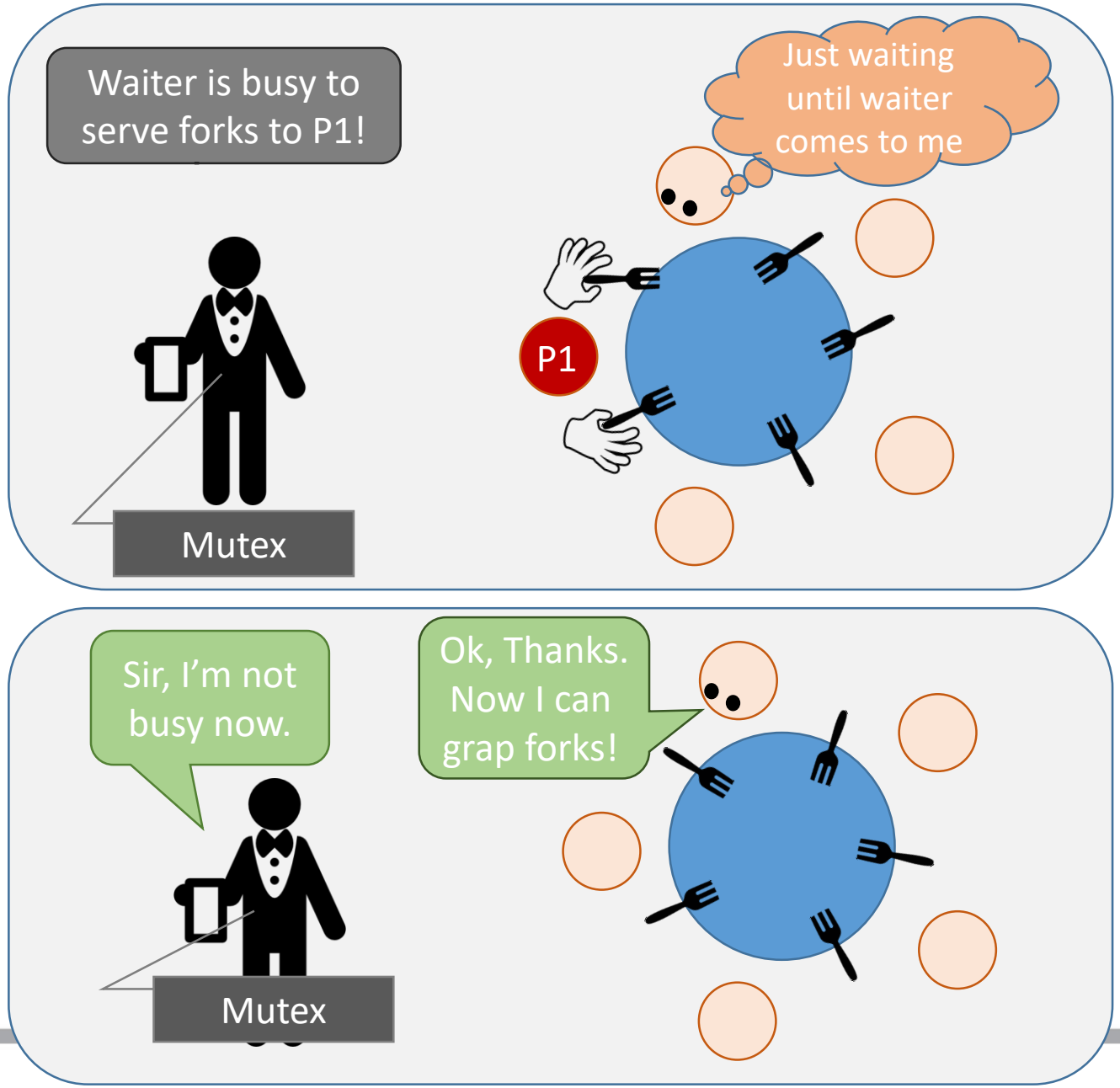
- **Starvation 발생 가능**
  - 어떤 철학자는 계속 기다릴 수도 있음





Solution 3: Use of Arbitrator

- 철학자는 웨이터가 도착할 때까지 기다림
- 웨이터가 철학자에게 와야지만 철학자는 포크를 집을 수 있음



Solution 3: Use of Arbitrator

각 철학자가 하는 일

```
do {
   wait(mutex);              // 웨이터가 서빙해주러 오길 기다림
   wait(fork[i]);            // 오른쪽에 있는 포크 집음
   wait(fork[(i+1)%N]);      // 왼쪽에 있는 포크 집음
   signal(mutex);            // 웨이터가 포크를 다 대접하고 떠남

   // 임계구역(Critical Section) 진입
   Eat();

   Signal(fork[i]);          // 오른쪽에 있는 포크 내려놓음
   Signal(fork[(i+1)%N]);    // 왼쪽에 있는 포크 내려놓음

   // 다음 임계구역 진입 전까지 대기
   Think();
} while(true);
```

Solution 3: Use of Arbitrator

```c
int main() {
  pthread_t thread_id[N];

  srand(time(NULL));

  pthread_mutex_init(&print_mutex, NULL);
  pthread_mutex_init(&mutex, NULL);

  for (int i = 0; i < N; i++) {
    sem_init(&forks[i], 0, 1);
    state[i] = 0;
  }

  print_table_index();

  for (int i = 0; i < N; i++) {
    name[i] = i;
    pthread_create(&thread_id[i], NULL, philosopher, &name[i]);
  }

  for (int i = 0; i < N; i++) {
    pthread_join(thread_id[i], NULL);
  }

  for (int i = 0; i < N; i++) {
    sem_destroy(&forks[i]);
  }

  pthread_mutex_destroy(&print_mutex);
  pthread_mutex_destroy(&mutex);
  pthread_mutex_destroy(&mutex);
}
```

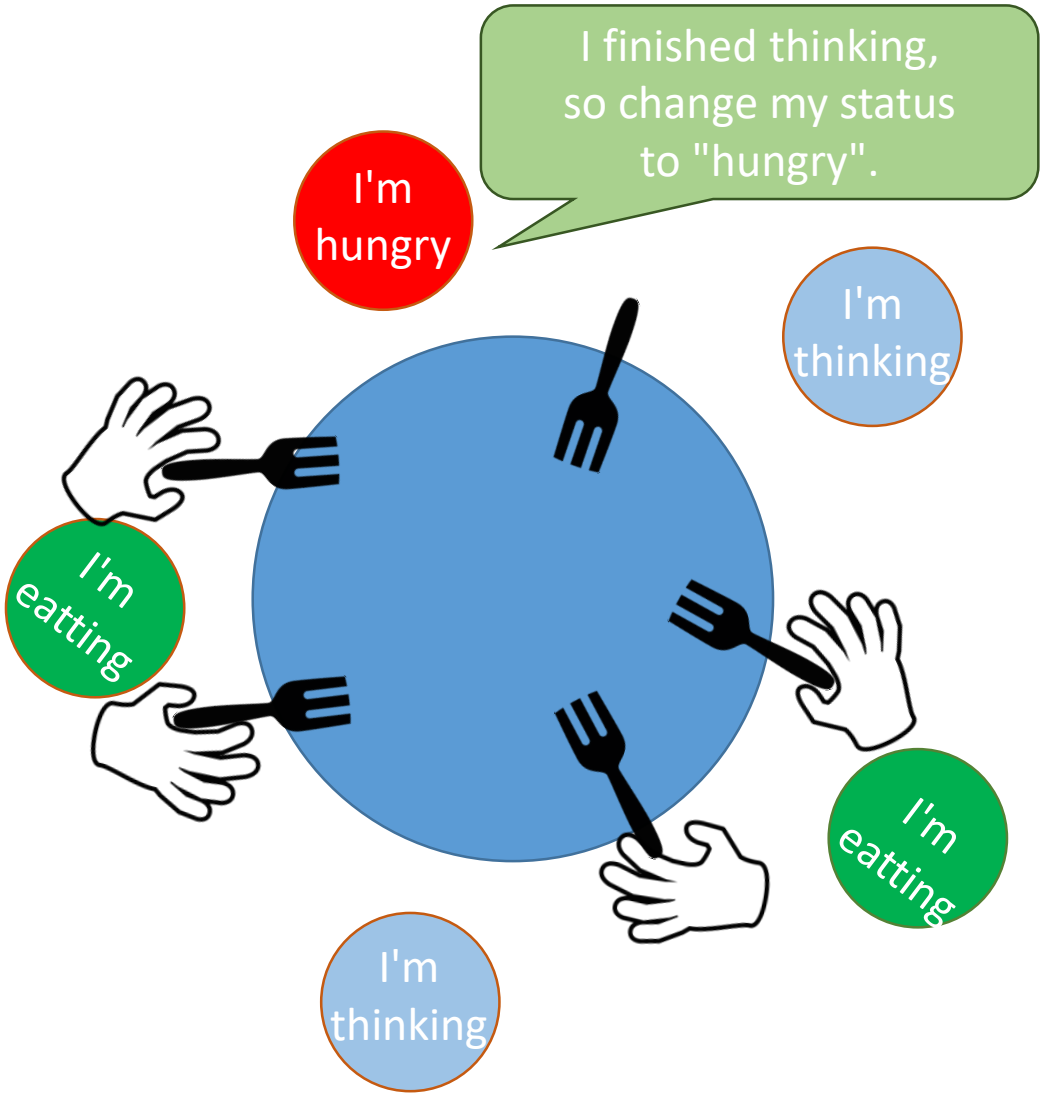
```c
void *philosopher(void *_name) {
  int phil = *((int*)_name);

  usleep(rand()%3000000);

  do {
    pthread_mutex_lock(&mutex);
    print_arbitrator(phil, "serves");

    sem_wait(&forks[phil]);
    print_fork(phil, phil, "take");
    sem_wait(&forks[(phil + 1) % N]);
    print_fork(phil, (phil + 1) % N, "take");

    pthread_mutex_unlock(&mutex);
    print_arbitrator(phil, "leaves");

    state[phil] = EATING;
    print_phstates();

    usleep(rand()%1000000);

    state[phil] = THINKING;
    print_phstates();

    sem_post(&forks[phil]);
    print_fork(phil, phil, "put");
    sem_post(&forks[(phil + 1) % N]);
    print_fork(phil, (phil + 1) % N, "put");

    usleep(rand()%2000000);

  } while (1);
}
```

Solution 3: 결과화면

동시에 최대 2명이 식사할 수 있는 반면,
웨이터를 기다리느라 그렇게 하지 못함.

```
jsbaik@jsbaik:~/OS2019/week10/solution_3$ ./use_of_arbitrator
================================================================================================================================
|       PHIL[0]         ||      PHIL[1]         ||      PHIL[2]         ||      PHIL[3]         ||      PHIL[4]         |
================================================================================================================================
The waiter serves PHIL[3]
PHIL[3] take 3 th fork.
PHIL[3] take 4 th fork.
The waiter leaves PHIL[3]
|       INIT            ||      INIT            ||      INIT            ||      EATING          ||      INIT            |
The waiter serves PHIL[4]
|       INIT            ||      INIT            ||      INIT            ||      THINKING        ||      INIT            |
PHIL[3] put 3 th fork.
PHIL[3] put 4 th fork.
PHIL[4] take 4 th fork.
PHIL[4] take 0 th fork.
The waiter leaves PHIL[4]
|       INIT            ||      INIT            ||      INIT            ||      THINKING        ||      EATING          |
The waiter serves PHIL[2]
PHIL[2] take 2 th fork.
PHIL[2] take 3 th fork.
The waiter leaves PHIL[2]
|       INIT            ||      INIT            ||      EATING          ||      THINKING        ||      THINKING        |
The waiter serves PHIL[0]
|       INIT            ||      INIT            ||      EATING          ||      THINKING        ||      THINKING        |
PHIL[4] put 4 th fork.
PHIL[4] put 0 th fork.
|       INIT            ||      INIT            ||      THINKING        ||      THINKING        ||      THINKING        |
PHIL[2] put 2 th fork.
PHIL[2] put 3 th fork.
PHIL[0] take 0 th fork.
PHIL[0] take 1 th fork.
The waiter leaves PHIL[0]
|       EATING          ||      INIT            ||      THINKING        ||      THINKING        ||      THINKING        |
The waiter serves PHIL[1]
|       THINKING        ||      INIT            ||      THINKING        ||      THINKING        ||      THINKING        |
PHIL[0] put 0 th fork.
PHIL[0] put 1 th fork.
PHIL[1] take 1 th fork.
PHIL[1] take 2 th fork.
The waiter leaves PHIL[1]
|       THINKING        ||      EATING          ||      THINKING        ||      THINKING        ||      THINKING        |
The waiter serves PHIL[3]
PHIL[3] take 3 th fork.
PHIL[3] take 4 th fork.
The waiter leaves PHIL[3]
```

Solution 4: Tanenbaum's Solution



- **<u>Hungry 상태 추가</u>**

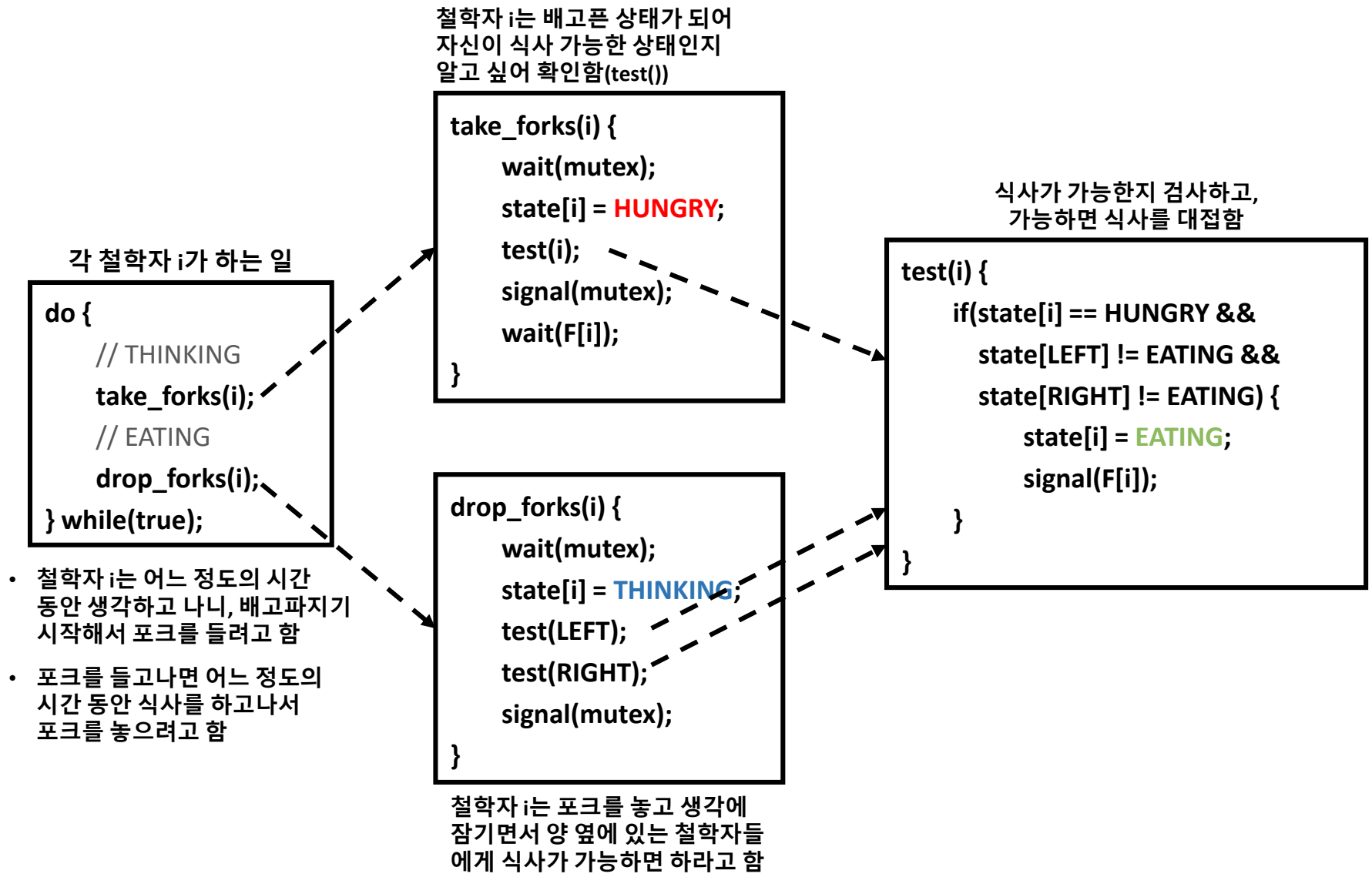
- hungry 상태를 추가해, 양 옆 포크가 사용 가능해야지만 포크를 집음

Solution 4: Tanenbaum's Solution

	상태			
포크	사용		lock	공유자원
	미사용		unlock	
철학자	먹는다	<-- 대응 -->	running	프로세스 / 스레드
	생각한다		waiting	
	배고파한다		**ready**	

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

Solution 4: Tanenbaum's Solution



- 철학자 i는 어느 정도의 시간 동안 생각하고 나니, 배고파지기 시작해서 포크를 들려고 함
- 포크를 들고나면 어느 정도의 시간 동안 식사를 하고나서 포크를 놓으려고 함

Solution 4: 결과화면

```
jsbaik@jsbaik:~/OS2019/week10/solution_4_answer$ make
gcc -g -o tanenbaum tanenbaum.c -lpthread
jsbaik@jsbaik:~/OS2019/week10/solution_4_answer$ ./tanenbaum
```

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

Testing Solutions

측정 단위

스케줄링 기준(Scheduling Criteria) 운영체제론 서적: 5.2절

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

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 최대한의 포크 개수는?

F : 포크 한 쌍의 개수

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

f : 총 포크 개수

동시에 최대한 많은 철학자들이 식사할 수 있을 때, 걸리는 시간은?

N : 철학자의 수

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

측정 단위

이 기간 안에 반드시 1회 이상 식사를 제공받아야 한다고 가정

- *Context switch overhead*
- *Release time (offset)*
- *…*

측정 단위 함수

```
int evaluate_time_unit() {
  int i_f = N / 2;  // 동시에 사용할 수 있는 포크 한쌍의 수
  float n = N;      // 철학자의 수(소수점 자리 계산이 가능하도록 타입 변환)
  float f = i_f;    // 포크 한쌍의 수(소수점 자리 계산이 가능하도록 타입 변환)
  return ceil(n / f) + 2; // 시간 단위 계산
}
```

반환 값: 측정 시간 단위

Solution 3: 시간 단위 테스팅 (N = 10, time_unit: 4 sec)

```
jsbaik@jsbaik:~/OS2019/week10/testing_solutions$ ./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[1]
        PHIL[1] take 2 th fork.
        PHIL[1] take 1 th fork.
The waiter leaves PHIL[1]
|       INIT            ||      EATING          ||      INIT            ||      INIT            ||      INIT            ||      INIT            ||      INIT            ||      INIT            ||      INIT            ||      INIT            |
The waiter serves PHIL[0]
|       INIT            ||      THINKING        ||      INIT            ||      INIT            ||      INIT            ||      INIT            ||      INIT            ||      INIT            ||      INIT            ||      INIT            |
        PHIL[1] put 2 th fork.
        PHIL[1] put 1 th fork.
        PHIL[0] take 1 th fork.
        PHIL[0] take 0 th fork.
The waiter leaves PHIL[0]
|       EATING          ||      THINKING        ||      INIT            ||      INIT            ||      INIT            ||      INIT            ||      INIT            ||      INIT            ||      INIT            ||      INIT            |
The waiter serves PHIL[2]
        PHIL[2] take 3 th fork.
        PHIL[2] take 2 th fork.
The waiter leaves PHIL[2]
|       EATING          ||      THINKING        ||      EATING          ||      INIT            ||      INIT            ||      INIT            ||      INIT            ||      INIT            ||      INIT            ||      INIT            |
The waiter serves PHIL[3]
        PHIL[3] take 4 th fork.
|       THINKING        ||      THINKING        ||      EATING          ||      INIT            ||      INIT            ||      INIT            ||      INIT            ||      INIT            ||      INIT            ||      INIT            |
        PHIL[0] put 1 th fork.
        PHIL[0] put 0 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[5]
        PHIL[5] take 6 th fork.
        PHIL[5] take 5 th fork.
The waiter leaves PHIL[5]
|       THINKING        ||      THINKING        ||      THINKING        ||      EATING          ||      INIT            ||      EATING          ||      INIT            ||      INIT            ||      INIT            ||      INIT            |
The waiter serves PHIL[4]
|       THINKING        ||      THINKING        ||      THINKING        ||      THINKING        ||      INIT            ||      EATING          ||      INIT            ||      INIT            ||      INIT            ||      INIT            |
        PHIL[3] put 4 th fork.
        PHIL[3] put 3 th fork.
|       THINKING        ||      THINKING        ||      THINKING        ||      THINKING        ||      INIT            ||      THINKING        ||      INIT            ||      INIT            ||      INIT            ||      INIT            |
        PHIL[5] put 6 th fork.
        PHIL[5] put 5 th fork.
        PHIL[4] take 5 th fork.
        PHIL[4] take 4 th fork.
The waiter leaves PHIL[4]
|       THINKING        ||      THINKING        ||      THINKING        ||      THINKING        ||      EATING          ||      THINKING        ||      INIT            ||      INIT            ||      INIT            ||      INIT            |
The waiter serves PHIL[6]
        PHIL[6] take 7 th fork.
        PHIL[6] take 6 th fork.
The waiter leaves PHIL[6]
|       THINKING        ||      THINKING        ||      THINKING        ||      THINKING        ||      EATING          ||      THINKING        ||      EATING          ||      INIT            ||      INIT            ||      INIT            |
The waiter serves PHIL[7]
        PHIL[7] take 8 th fork.

**** Failed to Satisfy the Time Unit ****
jsbaik@jsbaik:~/OS2019/week10/testing_solutions$
```

시간 단위 테스트를 통과하지 못하고 종료함

Solution 4: 시간 단위 테스팅 (N = 10, time_unit: 4 sec)

```
jsbaik@jsbaik:~/OS2019/week10/testing_solutions$ ./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*	INIT	INIT	*HUNGRY*	INIT	INIT	INIT	INIT	INIT
EATING	*HUNGRY*	INIT	INIT	EATING	INIT	INIT	INIT	INIT	INIT
EATING	*HUNGRY*	*HUNGRY*	INIT	EATING	INIT	INIT	INIT	INIT	INIT
EATING	*HUNGRY*	EATING	INIT	EATING	INIT	INIT	INIT	INIT	INIT
EATING	*HUNGRY*	EATING	INIT	EATING	*HUNGRY*	INIT	INIT	INIT	INIT
EATING	*HUNGRY*	EATING	*HUNGRY*	EATING	*HUNGRY*	INIT	INIT	INIT	INIT
EATING	*HUNGRY*	EATING	*HUNGRY*	EATING	*HUNGRY*	INIT	*HUNGRY*	INIT	INIT
EATING	*HUNGRY*	EATING	*HUNGRY*	EATING	*HUNGRY*	INIT	EATING	INIT	INIT
EATING	*HUNGRY*	EATING	*HUNGRY*	EATING	*HUNGRY*	*HUNGRY*	EATING	INIT	INIT
EATING	*HUNGRY*	EATING	*HUNGRY*	EATING	*HUNGRY*	*HUNGRY*	EATING	*HUNGRY*	INIT
THINKING	*HUNGRY*	EATING	*HUNGRY*	EATING	*HUNGRY*	*HUNGRY*	EATING	*HUNGRY*	*HUNGRY*
THINKING	*HUNGRY*	EATING	*HUNGRY*	EATING	*HUNGRY*	*HUNGRY*	EATING	*HUNGRY*	EATING
THINKING	*HUNGRY*	EATING	*HUNGRY*	THINKING	*HUNGRY*	*HUNGRY*	EATING	*HUNGRY*	EATING
THINKING	*HUNGRY*	EATING	*HUNGRY*	THINKING	EATING	*HUNGRY*	EATING	*HUNGRY*	EATING
THINKING	*HUNGRY*	THINKING	*HUNGRY*	THINKING	EATING	*HUNGRY*	EATING	*HUNGRY*	EATING
THINKING	EATING	THINKING	*HUNGRY*	THINKING	EATING	*HUNGRY*	EATING	*HUNGRY*	EATING
THINKING	EATING	THINKING	EATING	THINKING	EATING	*HUNGRY*	EATING	*HUNGRY*	EATING
THINKING	EATING	THINKING	EATING	THINKING	THINKING	*HUNGRY*	EATING	*HUNGRY*	EATING
THINKING	EATING	THINKING	THINKING	THINKING	THINKING	*HUNGRY*	EATING	*HUNGRY*	EATING
THINKING	EATING	THINKING	THINKING	THINKING	THINKING	*HUNGRY*	THINKING	*HUNGRY*	EATING
THINKING	EATING	THINKING	THINKING	THINKING	THINKING	EATING	THINKING	*HUNGRY*	EATING
THINKING	EATING	THINKING	THINKING	THINKING	THINKING	THINKING	THINKING	*HUNGRY*	EATING
THINKING	EATING	THINKING	THINKING	THINKING	THINKING	THINKING	THINKING	*HUNGRY*	THINKING
HUNGRY	EATING	THINKING	THINKING	THINKING	THINKING	THINKING	THINKING	EATING	THINKING
HUNGRY	EATING	THINKING	THINKING	*HUNGRY*	THINKING	THINKING	THINKING	EATING	THINKING
HUNGRY	EATING	THINKING	THINKING	EATING	THINKING	THINKING	THINKING	EATING	THINKING
HUNGRY	EATING	*HUNGRY*	THINKING	EATING	THINKING	THINKING	THINKING	EATING	THINKING
HUNGRY	*HUNGRY*	*HUNGRY*	THINKING	EATING	THINKING	THINKING	THINKING	EATING	THINKING
HUNGRY	EATING	*HUNGRY*	THINKING	EATING	THINKING	THINKING	THINKING	EATING	THINKING
HUNGRY	EATING	*HUNGRY*	THINKING	EATING	*HUNGRY*	THINKING	THINKING	EATING	THINKING
HUNGRY	EATING	*HUNGRY*	THINKING	EATING	*HUNGRY*	THINKING	*HUNGRY*	EATING	THINKING
HUNGRY	EATING	*HUNGRY*	THINKING	EATING	*HUNGRY*	THINKING	*HUNGRY*	*HUNGRY*	THINKING
HUNGRY	EATING	*HUNGRY*	THINKING	EATING	*HUNGRY*	THINKING	*HUNGRY*	EATING	THINKING
HUNGRY	EATING	*HUNGRY*	THINKING	EATING	*HUNGRY*	*HUNGRY*	*HUNGRY*	EATING	THINKING
HUNGRY	EATING	*HUNGRY*	THINKING	EATING	*HUNGRY*	EATING	*HUNGRY*	EATING	THINKING
HUNGRY	EATING	*HUNGRY*	*HUNGRY*	EATING	*HUNGRY*	EATING	*HUNGRY*	EATING	*HUNGRY*

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

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

시간 단위 테스트를 통과함

수고하셨습니다.

- 다음 시간:

없음. 한 학기 동안 수고하셨습니다.