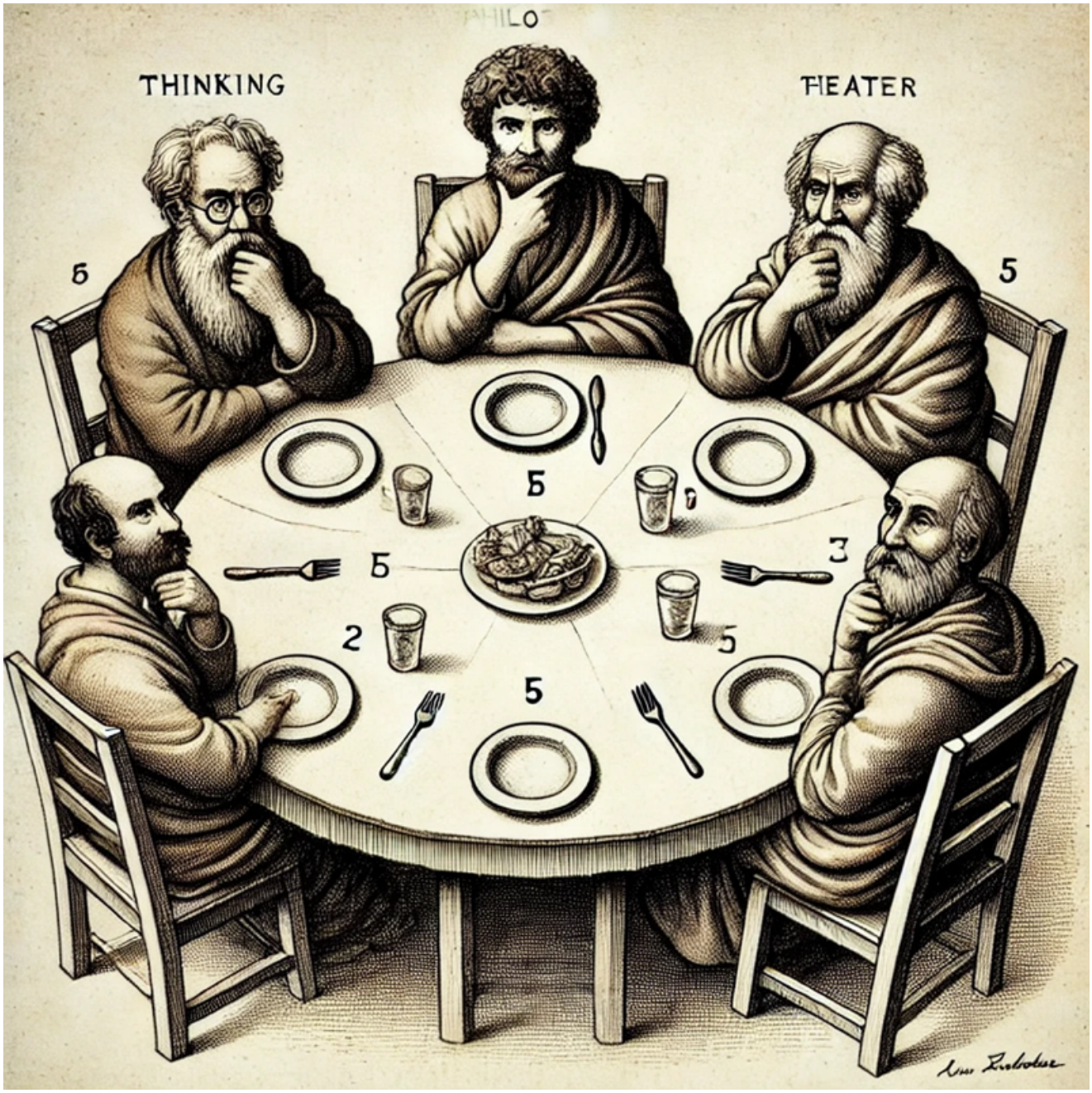
Dinning Philosphers

written by Nik Tsonev

Intro

there are 5 philosophers and 5 forks on a table. Each philosopher needs two forks to eat. They are polite and will not steal from a philosopher that is already using the fork. Figure out an algorithm so that everyone can eat and prevent `starvation` (a thread getting zero resources)



Solution

convert each fork as a binary semaphore, creating an array of 5 semaphores

Philosophers alternate between thinking and eating.

Each philosopher tries to pick up their left and right forks. To prevent deadlock, odd-numbered philosophers pick up the right fork first, while even-numbered philosophers pick up the left fork first.

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#define NUM_PHILOSOPHERS 5

sem_t forks[NUM_PHILOSOPHERS];

void* philosopher(void *num) {

    int id = *(int*)num;
    int left_fork = id;
    int right_fork = (id + 1) % NUM_PHILOSOPHERS;

    while (1) {

        printf("Philosopher %d is thinking.\n", id);
        sleep(rand() % 3 + 1);

        if (id % 2 == 0) {
            sem_wait(&forks[left_fork]);

        } else {

            sem_wait(&forks[right_fork]);

            printf("Philosopher %d picked up right fork
```

```
%d.\n", id, right_fork);

            sem_wait(&forks[left_fork]);

            printf("Philosopher %d picked up left fork
%d.\n", id, left_fork);

        }

        printf("Philosopher %d is eating.\n", id);

        sleep(rand() % 3 + 1);

        sem_post(&forks[left_fork]);

        printf("Philosopher %d put down left fork %d.\n",
id, left_fork);

        sem_post(&forks[right_fork]);

        printf("Philosopher %d put down right fork %d.\n",
id, right_fork);

    }

    return NULL;

}
```