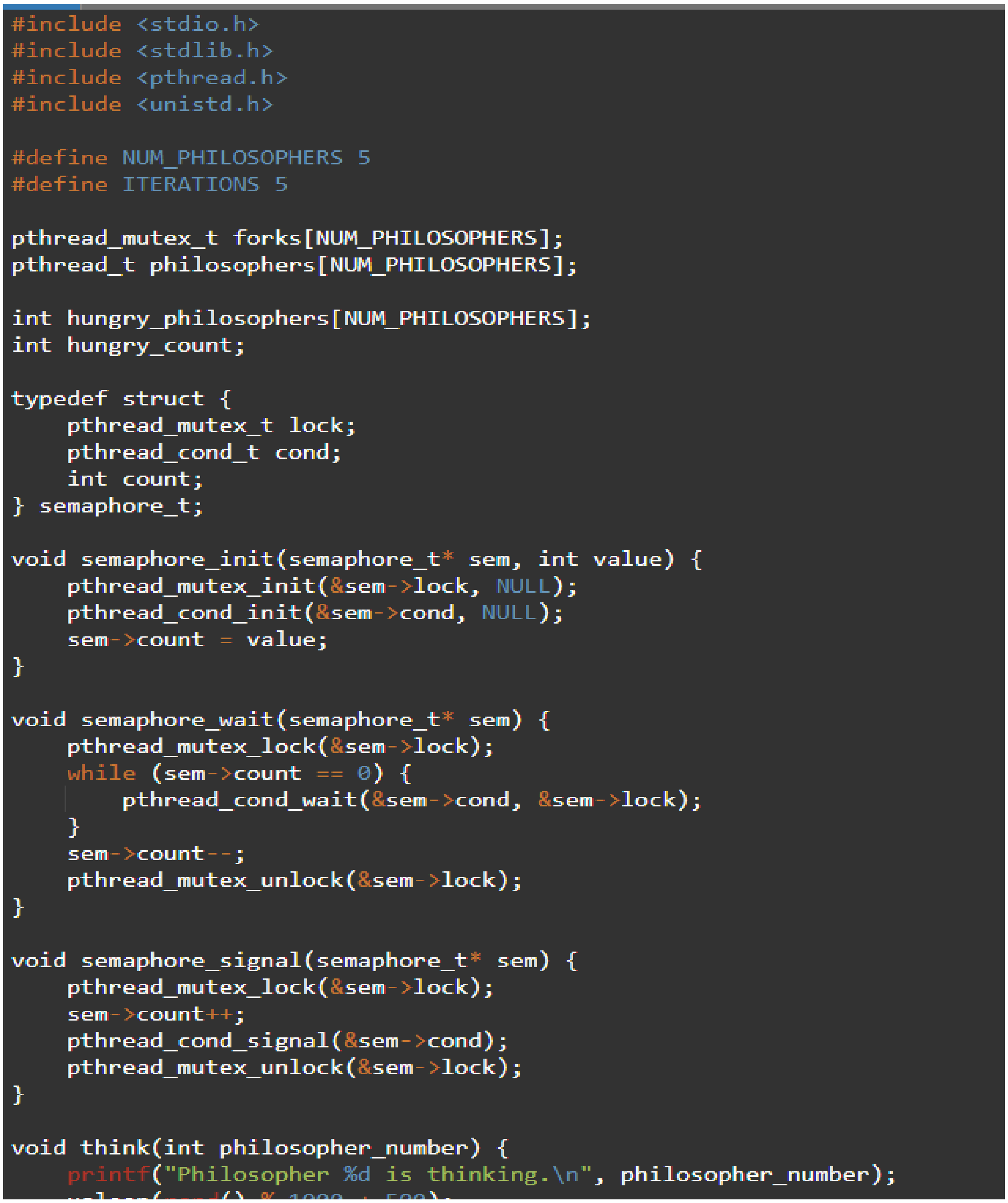
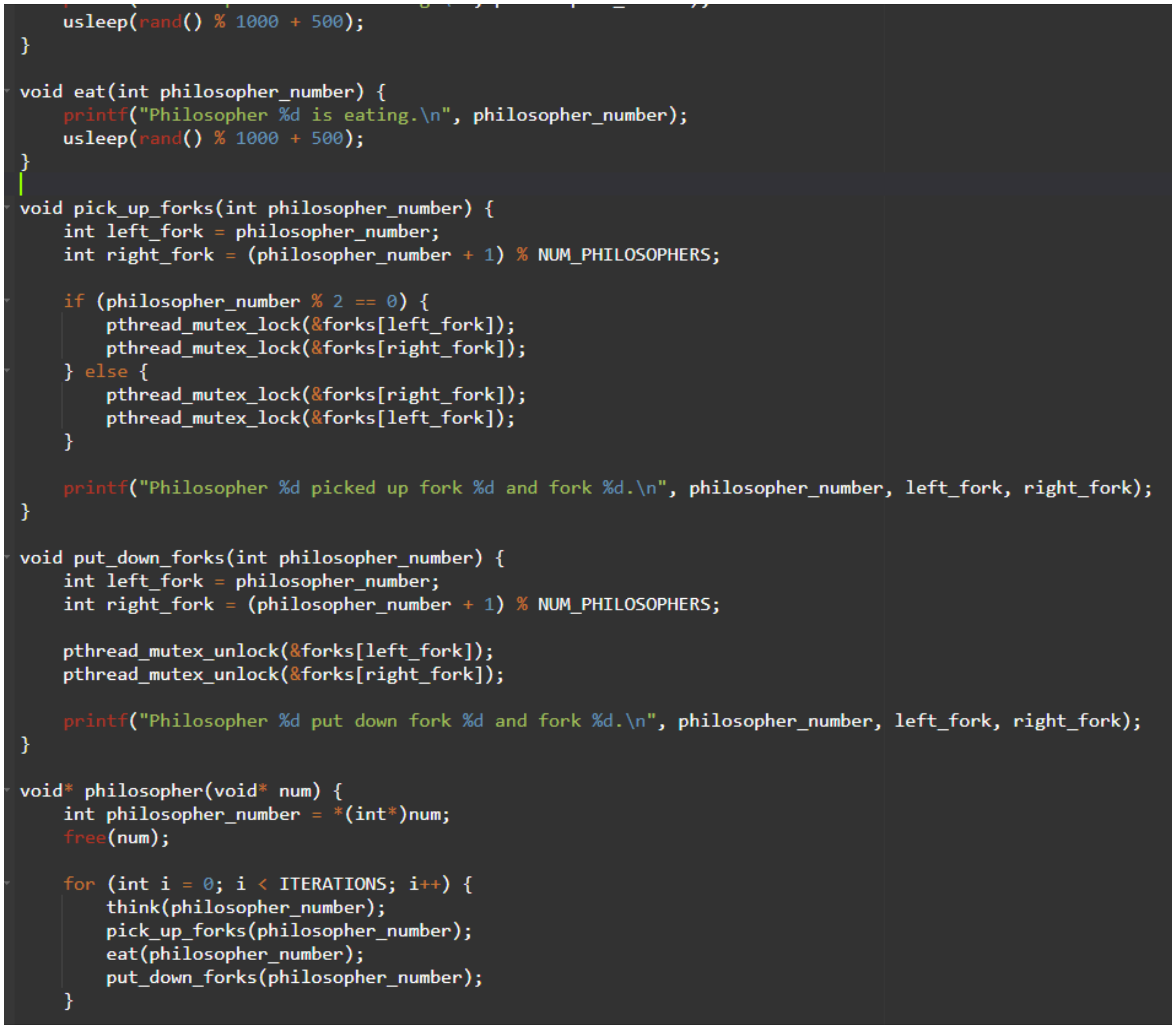
LAB PROGRAM 6
Write a C program to simulate the concept of Dining-Philosophers problem.

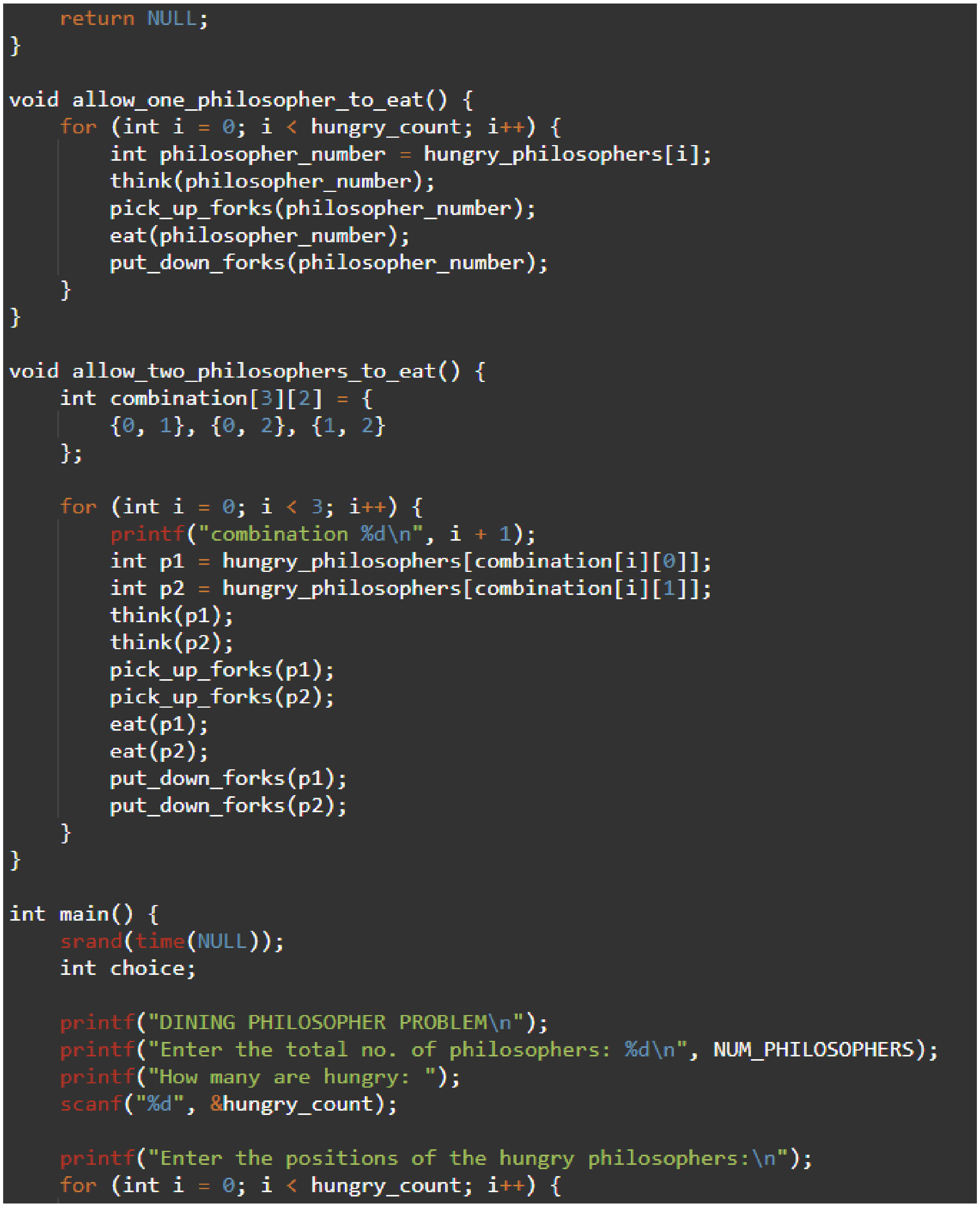
INPUT

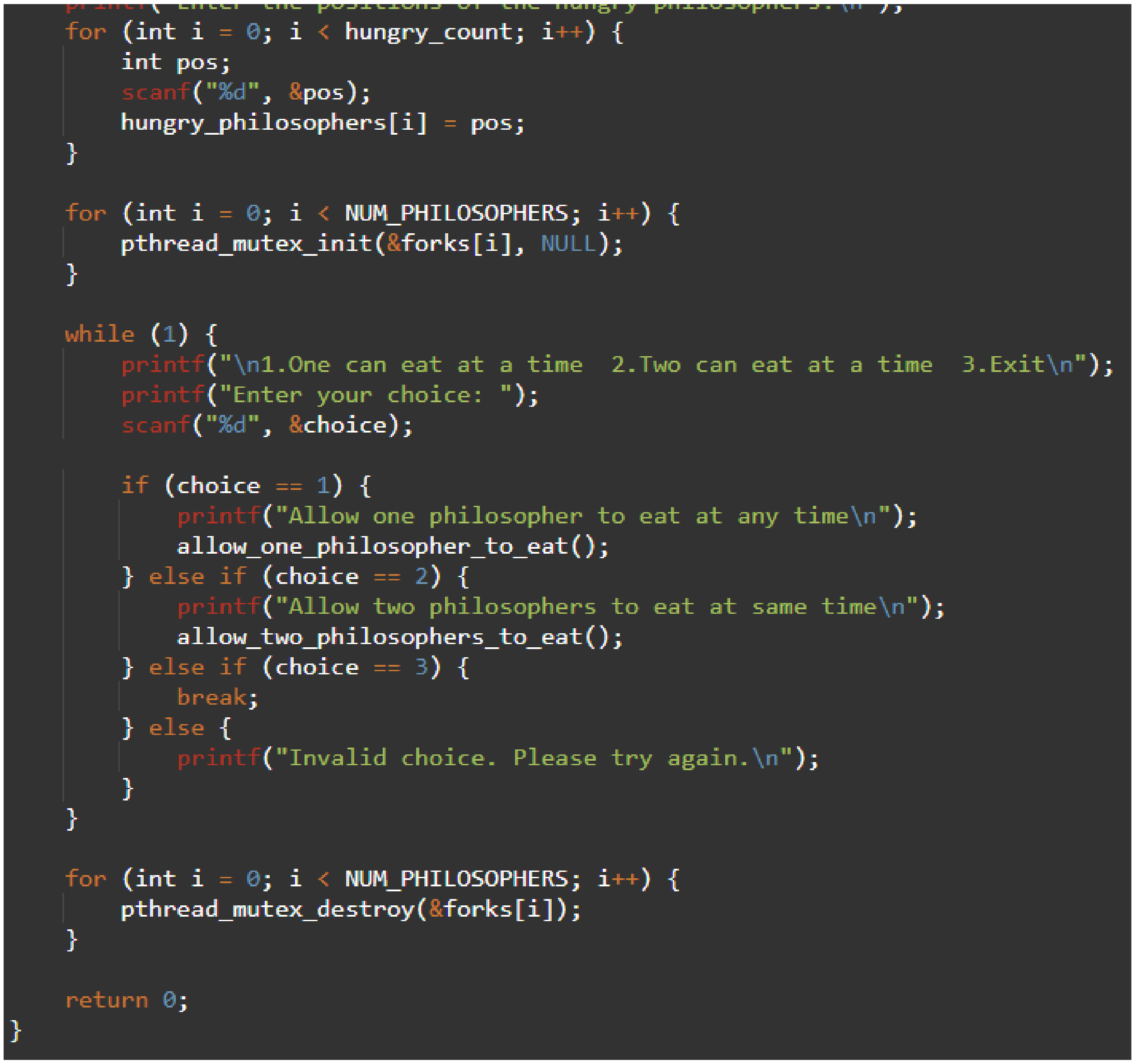


```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>

#define NUM_PHILOSOPHERS 5
#define ITERATIONS 5

pthread_mutex_t forks[NUM_PHILOSOPHERS];
pthread_t philosophers[NUM_PHILOSOPHERS];

int hungry_philosophers[NUM_PHILOSOPHERS];
int hungry_count;

typedef struct {
    pthread_mutex_t lock;
    pthread_cond_t cond;
    int count;
} semaphore_t;

void semaphore_init(semaphore_t* sem, int value) {
    pthread_mutex_init(&sem->lock, NULL);
    pthread_cond_init(&sem->cond, NULL);
    sem->count = value;
}

void semaphore_wait(semaphore_t* sem) {
    pthread_mutex_lock(&sem->lock);
    while (sem->count == 0) {
        pthread_cond_wait(&sem->cond, &sem->lock);
    }
    sem->count--;
    pthread_mutex_unlock(&sem->lock);
}

void semaphore_signal(semaphore_t* sem) {
    pthread_mutex_lock(&sem->lock);
    sem->count++;
    pthread_cond_signal(&sem->cond);
    pthread_mutex_unlock(&sem->lock);
}

void think(int philosopher_number) {
    printf("Philosopher %d is thinking.\n", philosopher_number);
    usleep(rand() % 1000 + 500);
```



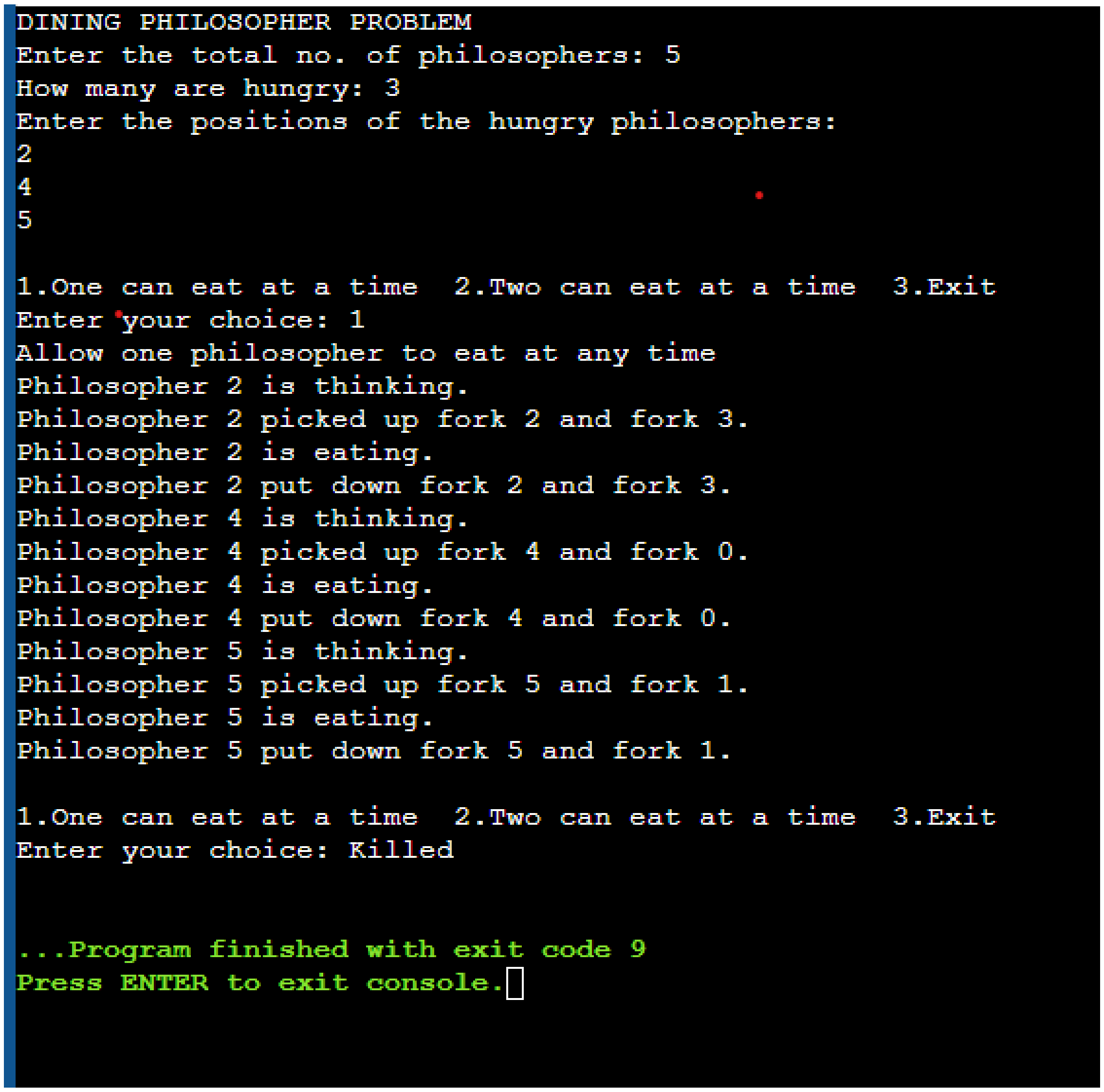
```c
    usleep(rand() % 1000 + 500);
}

void eat(int philosopher_number) {
    printf("Philosopher %d is eating.\n", philosopher_number);
    usleep(rand() % 1000 + 500);
}

void pick_up_forks(int philosopher_number) {
    int left_fork = philosopher_number;
    int right_fork = (philosopher_number + 1) % NUM_PHILOSOPHERS;

    if (philosopher_number % 2 == 0) {
        pthread_mutex_lock(&forks[left_fork]);
        pthread_mutex_lock(&forks[right_fork]);
    } else {
        pthread_mutex_lock(&forks[right_fork]);
        pthread_mutex_lock(&forks[left_fork]);
    }

    printf("Philosopher %d picked up fork %d and fork %d.\n", philosopher_number, left_fork, right_fork);
}

void put_down_forks(int philosopher_number) {
    int left_fork = philosopher_number;
    int right_fork = (philosopher_number + 1) % NUM_PHILOSOPHERS;

    pthread_mutex_unlock(&forks[left_fork]);
    pthread_mutex_unlock(&forks[right_fork]);

    printf("Philosopher %d put down fork %d and fork %d.\n", philosopher_number, left_fork, right_fork);
}

void* philosopher(void* num) {
    int philosopher_number = *(int*)num;
    free(num);

    for (int i = 0; i < ITERATIONS; i++) {
        think(philosopher_number);
        pick_up_forks(philosopher_number);
        eat(philosopher_number);
        put_down_forks(philosopher_number);
    }
```


```c
    return NULL;
}

void allow_one_philosopher_to_eat() {
    for (int i = 0; i < hungry_count; i++) {
        int philosopher_number = hungry_philosophers[i];
        think(philosopher_number);
        pick_up_forks(philosopher_number);
        eat(philosopher_number);
        put_down_forks(philosopher_number);
    }
}

void allow_two_philosophers_to_eat() {
    int combination[3][2] = {
        {0, 1}, {0, 2}, {1, 2}
    };

    for (int i = 0; i < 3; i++) {
        printf("combination %d\n", i + 1);
        int p1 = hungry_philosophers[combination[i][0]];
        int p2 = hungry_philosophers[combination[i][1]];
        think(p1);
        think(p2);
        pick_up_forks(p1);
        pick_up_forks(p2);
        eat(p1);
        eat(p2);
        put_down_forks(p1);
        put_down_forks(p2);
    }
}

int main() {
    srand(time(NULL));
    int choice;

    printf("DINING PHILOSOPHER PROBLEM\n");
    printf("Enter the total no. of philosophers: %d\n", NUM_PHILOSOPHERS);
    printf("How many are hungry: ");
    scanf("%d", &hungry_count);

    printf("Enter the positions of the hungry philosophers:\n");
    for (int i = 0; i < hungry_count; i++) {
```



```c
    printf("Enter the positions of the hungry philosophers:\n");
    for (int i = 0; i < hungry_count; i++) {
        int pos;
        scanf("%d", &pos);
        hungry_philosophers[i] = pos;
    }

    for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
        pthread_mutex_init(&forks[i], NULL);
    }

    while (1) {
        printf("\n1.One can eat at a time  2.Two can eat at a time  3.Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        if (choice == 1) {
            printf("Allow one philosopher to eat at any time\n");
            allow_one_philosopher_to_eat();
        } else if (choice == 2) {
            printf("Allow two philosophers to eat at same time\n");
            allow_two_philosophers_to_eat();
        } else if (choice == 3) {
            break;
        } else {
            printf("Invalid choice. Please try again.\n");
        }
    }

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

    return 0;
}
```

(OUPUT



```
DINING PHILOSOPHER PROBLEM
Enter the total no. of philosophers: 5
How many are hungry: 3
Enter the positions of the hungry philosophers:
2
4
5

1.One can eat at a time  2.Two can eat at a time  3.Exit
Enter your choice: 1
Allow one philosopher to eat at any time
Philosopher 2 is thinking.
Philosopher 2 picked up fork 2 and fork 3.
Philosopher 2 is eating.
Philosopher 2 put down fork 2 and fork 3.
Philosopher 4 is thinking.
Philosopher 4 picked up fork 4 and fork 0.
Philosopher 4 is eating.
Philosopher 4 put down fork 4 and fork 0.
Philosopher 5 is thinking.
Philosopher 5 picked up fork 5 and fork 1.
Philosopher 5 is eating.
Philosopher 5 put down fork 5 and fork 1.

1.One can eat at a time  2.Two can eat at a time  3.Exit
Enter your choice: Killed


...Program finished with exit code 9
Press ENTER to exit console.
```