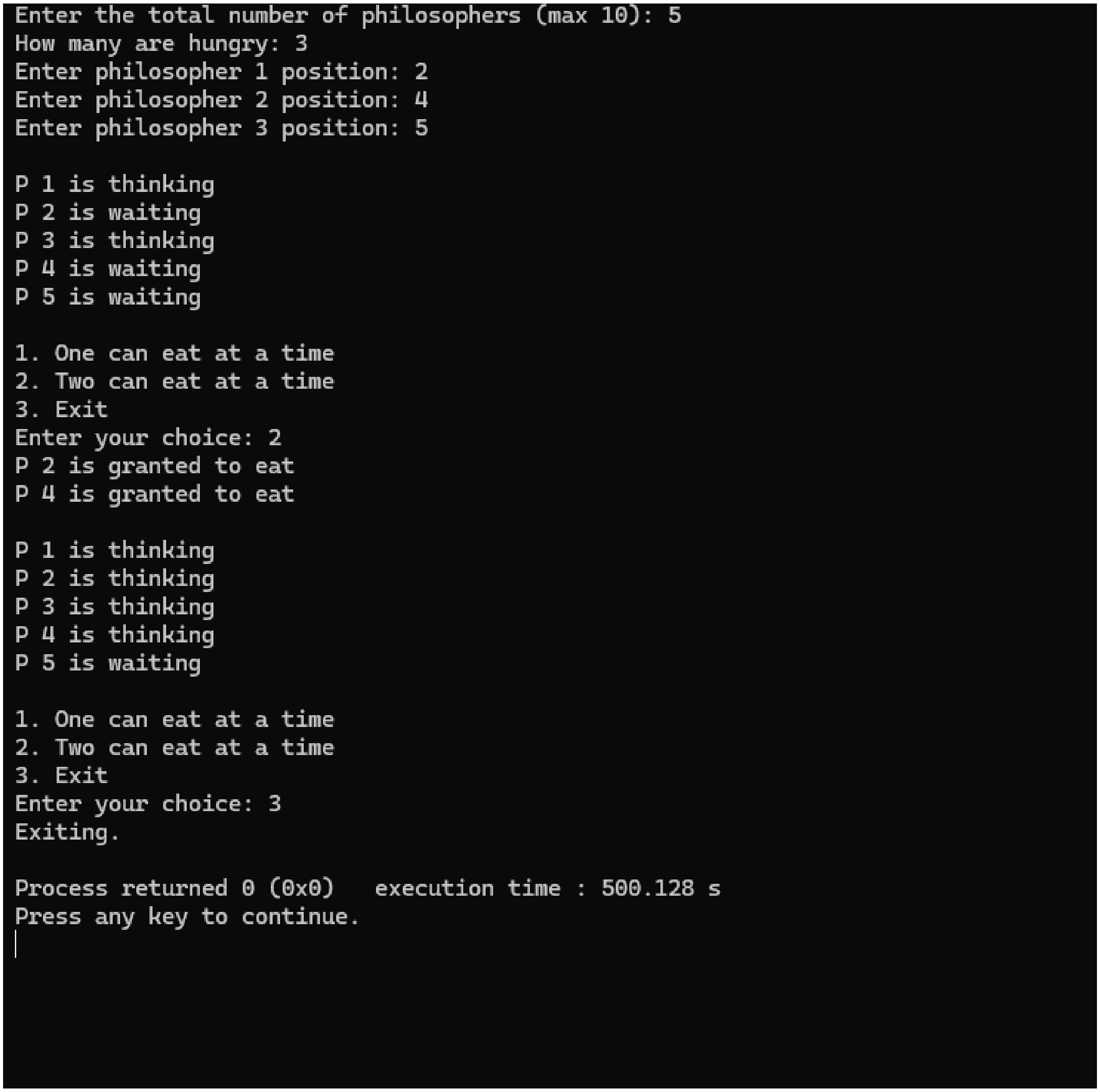
Write a C program to simulate the concept of Dining-Philosophers problem.

```
#include <stdio.h>
#include <stdlib.h>

#define MAX_PHILOSOPHERS 10

typedef enum { THINKING, HUNGRY, EATING } state_t;

state_t states[MAX_PHILOSOPHERS];
int num_philosophers;
int num_hungry;
int hungry_philosophers[MAX_PHILOSOPHERS];
int forks[MAX_PHILOSOPHERS];

void print_state() {
    printf("\n");
    for (int i = 0; i < num_philosophers; ++i) {
        if (states[i] == THINKING) printf("P %d is thinking\n", i + 1);
        else if (states[i] == HUNGRY) printf("P %d is waiting\n", i + 1);
        else if (states[i] == EATING) printf("P %d is eating\n", i + 1);
    }
}

int can_eat(int philosopher_id) {
    int left_fork = philosopher_id;
    int right_fork = (philosopher_id + 1) % num_philosophers;

    if (forks[left_fork] == 0 && forks[right_fork] == 0) {
        forks[left_fork] = forks[right_fork] = 1;
        return 1; // Philosopher can eat
    }
    return 0; // Philosopher cannot eat
}

void simulate(int allow_two) {
    int eating_count = 0;
    for (int i = 0; i < num_hungry; ++i) {
        int philosopher_id = hungry_philosophers[i];

        if (states[philosopher_id] == HUNGRY) {
            if (can_eat(philosopher_id)) {
                states[philosopher_id] = EATING;
                eating_count++;
                printf("P %d is granted to eat\n", philosopher_id + 1);
                if (!allow_two && eating_count == 1) break;
                if (allow_two && eating_count == 2) break;
            }
        }
    }

    // Simulate eating time (optional, as no explicit delay needed)

    // Transition eating philosophers back to thinking
    for (int i = 0; i < num_hungry; ++i) {
        int philosopher_id = hungry_philosophers[i];
        if (states[philosopher_id] == EATING) {
            int left_fork = philosopher_id;
            int right_fork = (philosopher_id + 1) % num_philosophers;
            forks[left_fork] = forks[right_fork] = 0;
            states[philosopher_id] = THINKING;
        }
    }
}

int main() {
    printf("Enter the total number of philosophers (max %d): ", MAX_PHILOSOPHERS);
    scanf("%d", &num_philosophers);

    if (num_philosophers < 2 || num_philosophers > MAX_PHILOSOPHERS) {
        printf("Invalid number of philosophers. Exiting.\n");
        return 1;
    }

    printf("How many are hungry: ");
    scanf("%d", &num_hungry);

    for (int i = 0; i < num_hungry; ++i) {
        printf("Enter philosopher %d position: ", i + 1);
        int position;
        scanf("%d", &position);
        hungry_philosophers[i] = position - 1; // Adjusting to 0-based indexing
        states[hungry_philosophers[i]] = HUNGRY;
    }

    // Initialize forks as available
    for (int i = 0; i < num_philosophers; ++i) {
        forks[i] = 0;
    }

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

        switch (choice) {
            case 1:
                simulate(0); // Allow one philosopher to eat at a time
                break;
            case 2:
                simulate(1); // Allow two philosophers to eat at the same time
                break;
            case 3:
                printf("Exiting.\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
                break;
        }

    } while (choice != 3);

    return 0;
}
```

OUTPUT:



```
Enter the total number of philosophers (max 10): 5
How many are hungry: 3
Enter philosopher 1 position: 2
Enter philosopher 2 position: 4
Enter philosopher 3 position: 5

P 1 is thinking
P 2 is waiting
P 3 is thinking
P 4 is waiting
P 5 is waiting

1. One can eat at a time
2. Two can eat at a time
3. Exit
Enter your choice: 2
P 2 is granted to eat
P 4 is granted to eat

P 1 is thinking
P 2 is thinking
P 3 is thinking
P 4 is thinking
P 5 is waiting

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

Process returned 0 (0x0)   execution time : 500.128 s
Press any key to continue.
```