Name: Spandan Mukherjee
Registration Number: 21BCE1132
Subject: OS Lab

Experiment: Dining Philosopher's

1.Maths teacher gives homework to the students. There are twenty students in the class. She has a box with five divisions in it. In that division she will write a integer number. Any student who is ready, may get the number and display the number name of it. At the maximum, teacher can give 5 numbers in 5 divisions. The same way student can consume 5 integer numbers which in turn will be displayed as number names. Develop a C program for this scenario and justify the output generated by your code.

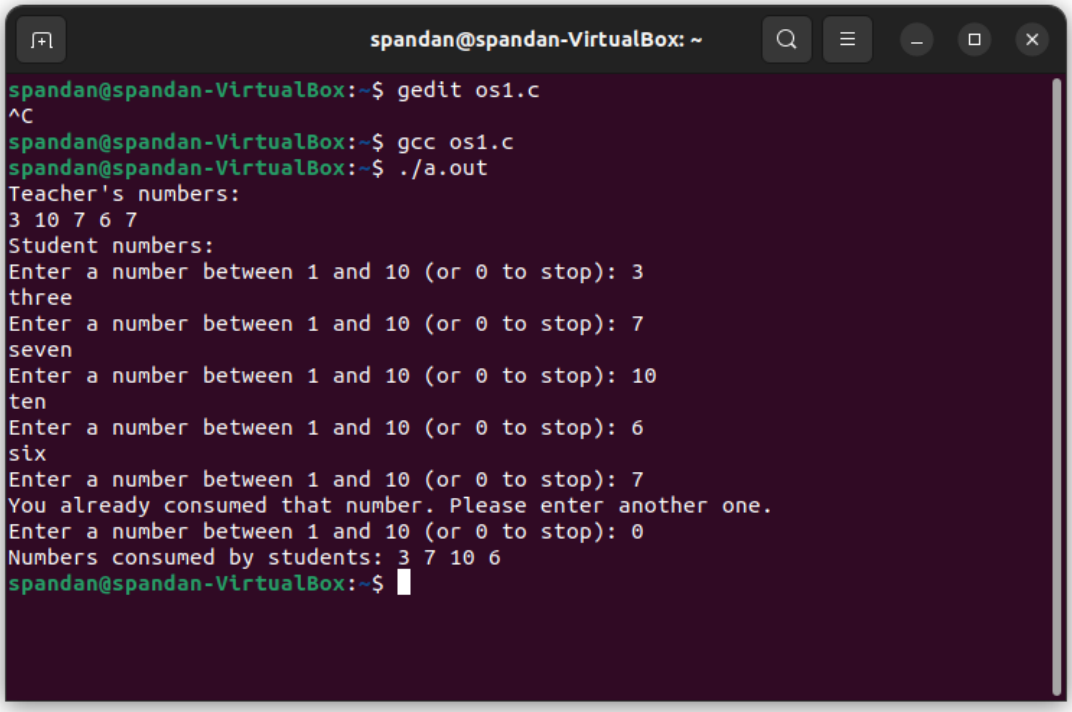
Code:

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

#define MAX_STUDENTS 20
#define MAX_NUMBERS 5

char* number_name(int n) {
   // function to return the name of the given number
   // assuming the numbers are between 1 and 10
   static char* names[] = {"one", "two", "three", "four", "five", "six", "seven", "eight", "nine",
"ten"};
   if (n >= 1 && n <= 10) {
      return names[n-1];
   }
   return "unknown";
}

int main() {
   srand(time(NULL)); // initialize random number generator

   int numbers[MAX_NUMBERS]; // array to store the numbers
   int num_count = 0; // number of numbers generated
   int consumed_count = 0; // number of numbers consumed by students
   int consumed[MAX_NUMBERS]; // array to store the consumed numbers
   memset(consumed, 0, sizeof(consumed)); // initialize consumed array to all zeros

   // generate 5 numbers and display them
   printf("Teacher's numbers:\n");
   while (num_count < MAX_NUMBERS) {
      int n = rand() % 10 + 1; // generate a random number between 1 and 10
      numbers[num_count] = n;
      printf("%d ", n);
      num_count++;
   }
```

```
    printf("\n");

    // allow students to consume up to 5 numbers
    printf("Student numbers:\n");
    while (consumed_count < MAX_NUMBERS) {
        int n;
        printf("Enter a number between 1 and 10 (or 0 to stop): ");
        scanf("%d", &n);
        if (n == 0) {
            break;
        }
        // check if the number has already been consumed
        int i;
        for (i = 0; i < consumed_count; i++) {
            if (consumed[i] == n) {
                printf("You already consumed that number. Please enter another one.\n");
                break;
            }
        }
        if (i < consumed_count) {
            continue;
        }
        // check if the number is in the teacher's list
        for (i = 0; i < num_count; i++) {
            if (numbers[i] == n) {
                printf("%s\n", number_name(n));
                consumed[consumed_count] = n;
                consumed_count++;
                break;
            }
        }
        if (i >= num_count) {
            printf("That number is not in the teacher's list. Please enter another one.\n");
        }
    }

    // print the list of consumed numbers
    printf("Numbers consumed by students:");
    int i;
    for (i = 0; i < consumed_count; i++) {
        printf(" %d", consumed[i]);
    }
    printf("\n");

    return 0;
}
```

Output:

2.



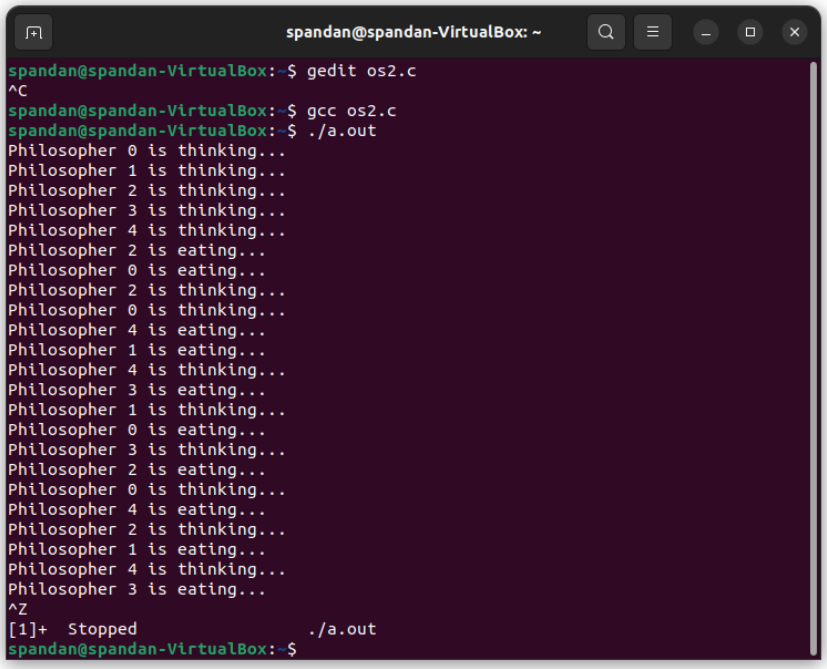
Develop a dining philosopher problem using C without synchronization.

CODE:

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

#define N 5 // number of philosophers
#define THINKING 0
#define HUNGRY 1
#define EATING 2

int state[N]; // array to store the state of each philosopher
pthread_t thread_ids[N]; // array to store the thread IDs of each philosopher
pthread_mutex_t forks[N]; // array to store the mutex locks for each fork

void *philosopher(void *arg) {
    int id = *((int*)arg);
    int left_fork = id;
    int right_fork = (id + 1) % N;

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

        // get forks
        pthread_mutex_lock(&forks[left_fork]);
        pthread_mutex_lock(&forks[right_fork]);
```

```
        // start eating
        state[id] = EATING;
        printf("Philosopher %d is eating...\n", id);
        sleep(rand() % 3 + 1);

        // release forks
        pthread_mutex_unlock(&forks[right_fork]);
        pthread_mutex_unlock(&forks[left_fork]);
        state[id] = THINKING;
    }
}

int main() {
    int i;
    for (i = 0; i < N; i++) {
        state[i] = THINKING;
        pthread_mutex_init(&forks[i], NULL);
    }

    for (i = 0; i < N; i++) {
        int *id = malloc(sizeof(int));
        *id = i;
        pthread_create(&thread_ids[i], NULL, philosopher, id);
    }

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

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

    return 0;
}
```

OUTPUT:



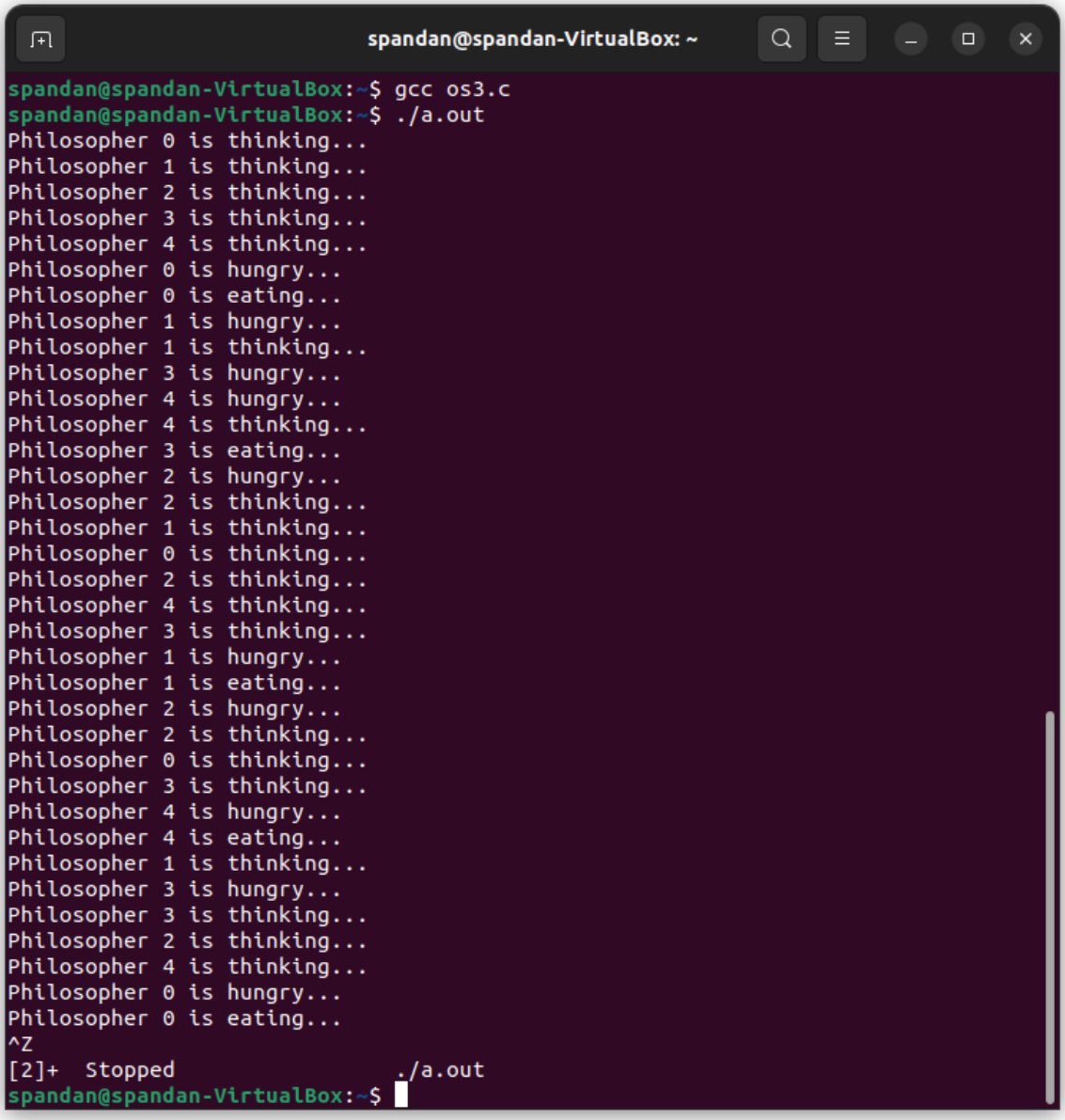
3. Develop a dining philosopher problem with if condition and constant sleep time using C without synchronization construct.

CODE:

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

#define N 5 // number of philosophers
#define THINKING 0
#define HUNGRY 1
#define EATING 2

int state[N]; // array to store the state of each philosopher
pthread_t thread_ids[N]; // array to store the thread IDs of each philosopher

void pickup_forks(int id) {
    state[id] = HUNGRY;
    printf("Philosopher %d is hungry...\n", id);
}

void putdown_forks(int id) {
    state[id] = THINKING;
    printf("Philosopher %d is thinking...\n", id);
}

void test(int id) {
    int left_fork = (id + N - 1) % N;
    int right_fork = (id + 1) % N;
```

```
    if (state[id] == HUNGRY && state[left_fork] != EATING && state[right_fork] != EATING) {
        state[id] = EATING;
        printf("Philosopher %d is eating...\n", id);
        sleep(1);
    }
}

void *philosopher(void *arg) {
    int id = *((int*)arg);

    while (1) {
        // thinking
        putdown_forks(id);
        sleep(1);

        // get forks
        pickup_forks(id);
        test(id);

        // release forks
        putdown_forks(id);
        sleep(1);
    }
}

int main() {
    int i;
    for (i = 0; i < N; i++) {
        state[i] = THINKING;
    }

    for (i = 0; i < N; i++) {
        int *id = malloc(sizeof(int));
        *id = i;
        pthread_create(&thread_ids[i], NULL, philosopher, id);
    }

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

    return 0;
}
```

OUTPUT:



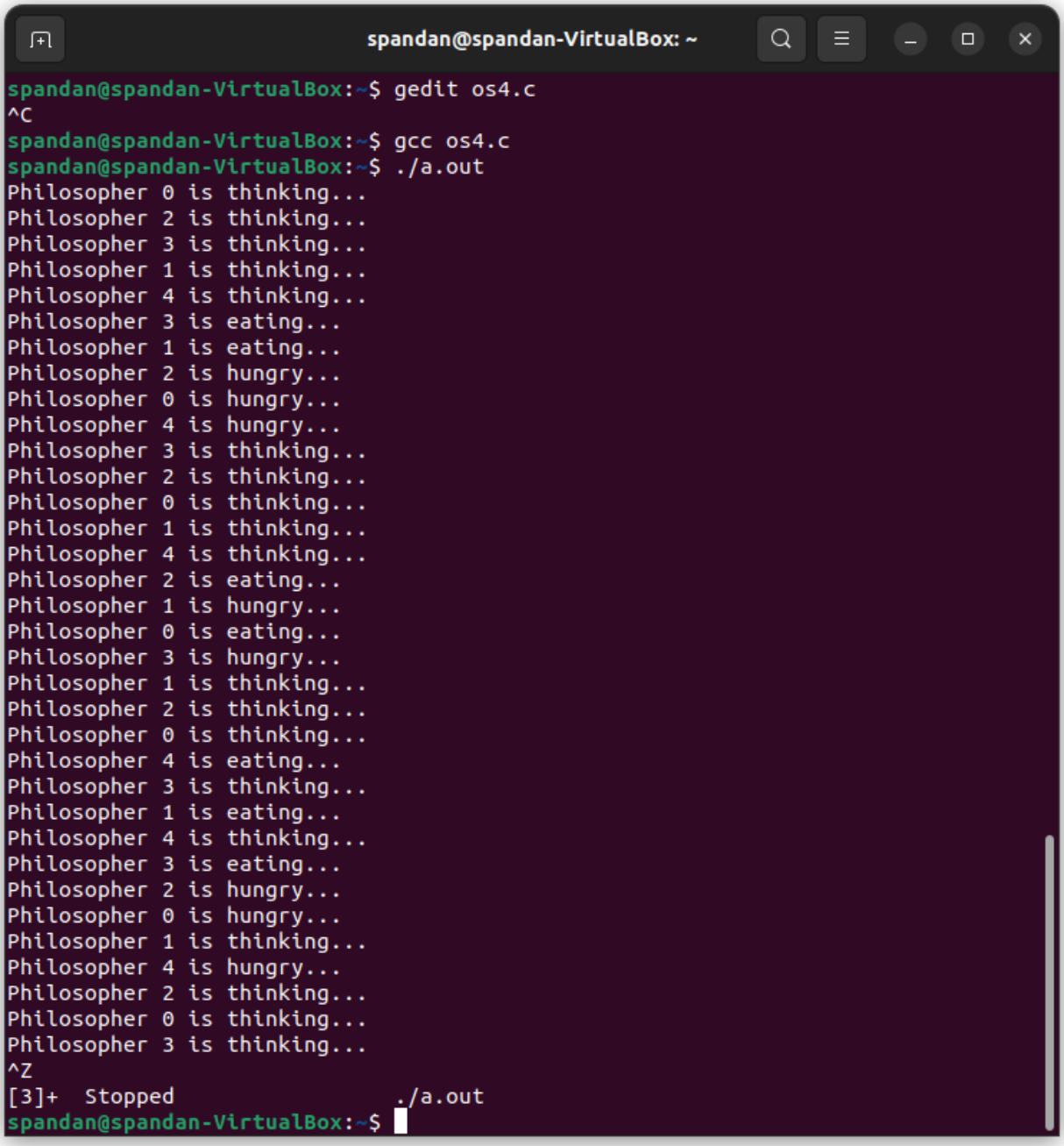
4. Develop a dining philosopher problem with if condition and random sleep time using C without synchronization construct.

CODE:

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

#define N 5 // number of philosophers
#define THINKING 0
#define HUNGRY 1
#define EATING 2

int state[N]; // array to store the state of each philosopher
pthread_t thread_ids[N]; // array to store the thread IDs of each philosopher
int forks[N]; // array to store the availability of each fork
```

```
void *philosopher(void *arg) {
    int id = *((int*)arg);
    int left_fork = id;
    int right_fork = (id + 1) % N;

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

        // try to get forks
        if (forks[left_fork] == 1 && forks[right_fork] == 1) {
            forks[left_fork] = 0;
            forks[right_fork] = 0;
            state[id] = EATING;
            printf("Philosopher %d is eating...\n", id);
            sleep(rand() % 3 + 1);
            forks[left_fork] = 1;
            forks[right_fork] = 1;
            state[id] = THINKING;
        } else {
            state[id] = HUNGRY;
            printf("Philosopher %d is hungry...\n", id);
            sleep(rand() % 3 + 1);
        }
    }
}

int main() {
    int i;
    for (i = 0; i < N; i++) {
        state[i] = THINKING;
        forks[i] = 1; // all forks are initially available
    }

    for (i = 0; i < N; i++) {
        int *id = malloc(sizeof(int));
        *id = i;
        pthread_create(&thread_ids[i], NULL, philosopher, id);
    }

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

    return 0;
}
```

OUTPUT:



5.Develop a Dining philosopher problem using Mutex

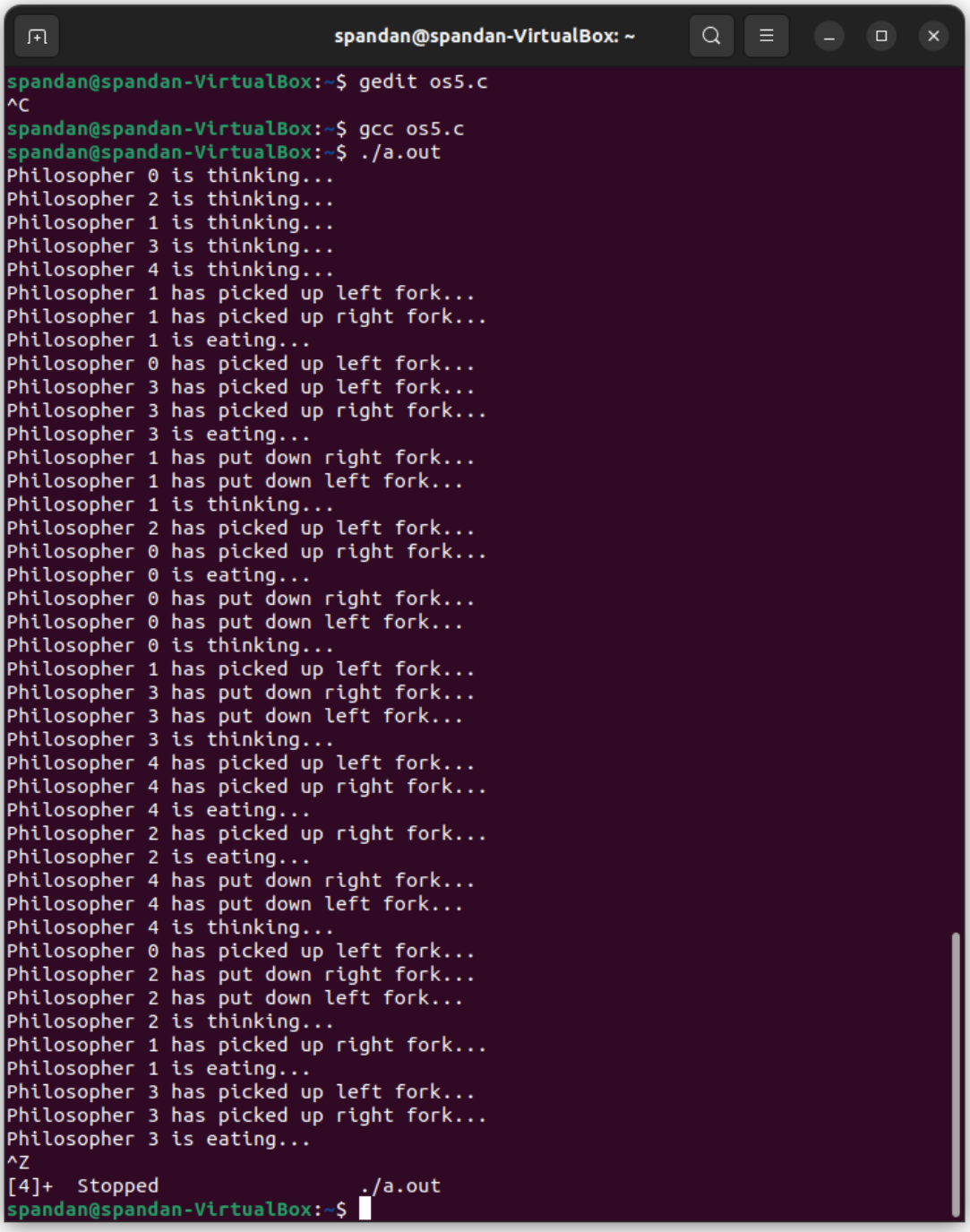
CODE:

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

#define N 5 // number of philosophers
#define THINKING 0
```

```
#define HUNGRY 1
#define EATING 2

int state[N]; // array to store the state of each philosopher
pthread_t thread_ids[N]; // array to store the thread IDs of each philosopher
pthread_mutex_t forks[N]; // array of mutex locks, one for each fork

void *philosopher(void *arg) {
    int id = *((int*)arg);
    int left_fork = id;
    int right_fork = (id + 1) % N;

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

        // try to get forks
        pthread_mutex_lock(&forks[left_fork]);
        printf("Philosopher %d has picked up left fork...\n", id);
        pthread_mutex_lock(&forks[right_fork]);
        printf("Philosopher %d has picked up right fork...\n", id);
        state[id] = EATING;
        printf("Philosopher %d is eating...\n", id);
        sleep(rand() % 3 + 1);
        pthread_mutex_unlock(&forks[right_fork]);
        printf("Philosopher %d has put down right fork...\n", id);
        pthread_mutex_unlock(&forks[left_fork]);
        printf("Philosopher %d has put down left fork...\n", id);
        state[id] = THINKING;
    }
}

int main() {
    int i;
    for (i = 0; i < N; i++) {
        state[i] = THINKING;
        pthread_mutex_init(&forks[i], NULL); // initialize mutex lock for each fork
    }

    for (i = 0; i < N; i++) {
        int *id = malloc(sizeof(int));
        *id = i;
        pthread_create(&thread_ids[i], NULL, philosopher, id);
    }

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

    for (i = 0; i < N; i++) {
        pthread_mutex_destroy(&forks[i]); // destroy mutex locks
```

```
    }

    return 0;
}
```

OUTPUT:



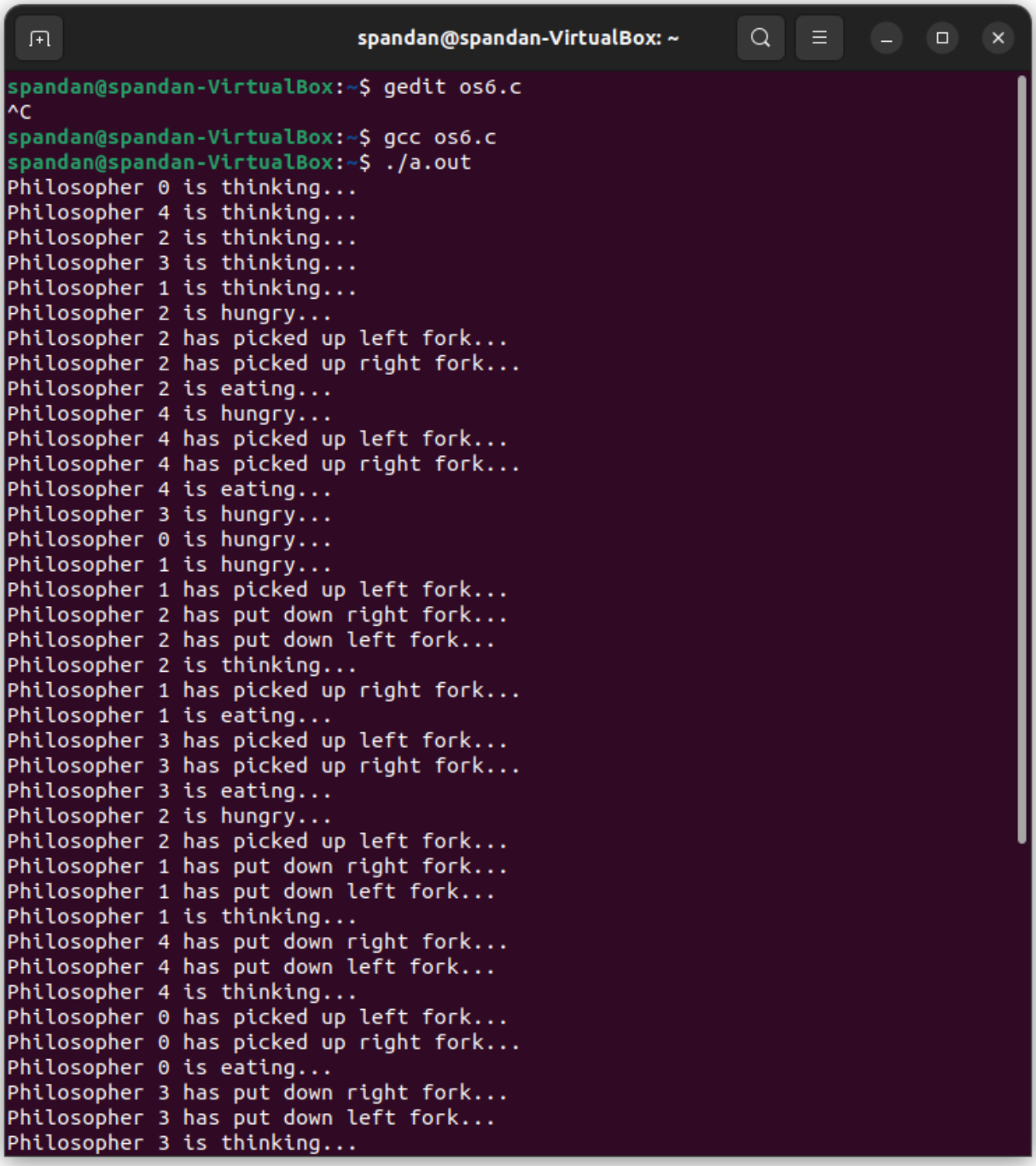
6.Develop a Dining problem using semaphore with test(), takefork() and putfork() functions for better synchronization of Dining_philosopher problem

CODE:

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

#define N 5 // number of philosophers
#define THINKING 0
#define HUNGRY 1
#define EATING 2

int state[N]; // array to store the state of each philosopher
pthread_t thread_ids[N]; // array to store the thread IDs of each philosopher
sem_t forks[N]; // array of semaphores, one for each fork

void test(int id) {
   if (state[id] == HUNGRY && state[(id + N - 1) % N] != EATING && state[(id + 1) % N] !=
EATING) {
       state[id] = EATING;
       sem_post(&forks[id]);
   }
}

void takefork(int id) {
   sem_wait(&forks[id]);
}

void putfork(int id) {
   sem_post(&forks[id]);
}

void *philosopher(void *arg) {
   int id = *((int*)arg);
   int left_fork = id;
   int right_fork = (id + 1) % N;

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

      // try to get forks
      state[id] = HUNGRY;
      printf("Philosopher %d is hungry...\n", id);
      test(id);
      takefork(id);
      printf("Philosopher %d has picked up left fork...\n", id);
```

```
        takefork((id + 1) % N);
        printf("Philosopher %d has picked up right fork...\n", id);
        printf("Philosopher %d is eating...\n", id);
        sleep(rand() % 3 + 1);

        // release forks
        putfork((id + 1) % N);
        printf("Philosopher %d has put down right fork...\n", id);
        putfork(id);
        printf("Philosopher %d has put down left fork...\n", id);
        state[id] = THINKING;
    }
}

int main() {
    int i;
    for (i = 0; i < N; i++) {
        state[i] = THINKING;
        sem_init(&forks[i], 0, 1); // initialize semaphores for each fork
    }

    for (i = 0; i < N; i++) {
        int *id = malloc(sizeof(int));
        *id = i;
        pthread_create(&thread_ids[i], NULL, philosopher, id);
    }

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

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

    return 0;
}
```

OUTPUT:


spandan@spandan-VirtualBox: ~
spandan@spandan-VirtualBox:~$ gedit os6.c
^C
spandan@spandan-VirtualBox:~$ gcc os6.c
spandan@spandan-VirtualBox:~$ ./a.out
Philosopher 0 is thinking...
Philosopher 4 is thinking...
Philosopher 2 is thinking...
Philosopher 3 is thinking...
Philosopher 1 is thinking...
Philosopher 2 is hungry...
Philosopher 2 has picked up left fork...
Philosopher 2 has picked up right fork...
Philosopher 2 is eating...
Philosopher 4 is hungry...
Philosopher 4 has picked up left fork...
Philosopher 4 has picked up right fork...
Philosopher 4 is eating...
Philosopher 3 is hungry...
Philosopher 0 is hungry...
Philosopher 1 is hungry...
Philosopher 1 has picked up left fork...
Philosopher 2 has put down right fork...
Philosopher 2 has put down left fork...
Philosopher 2 is thinking...
Philosopher 1 has picked up right fork...
Philosopher 1 is eating...
Philosopher 3 has picked up left fork...
Philosopher 3 has picked up right fork...
Philosopher 3 is eating...
Philosopher 2 is hungry...
Philosopher 2 has picked up left fork...
Philosopher 1 has put down right fork...
Philosopher 1 has put down left fork...
Philosopher 1 is thinking...
Philosopher 4 has put down right fork...
Philosopher 4 has put down left fork...
Philosopher 4 is thinking...
Philosopher 0 has picked up left fork...
Philosopher 0 has picked up right fork...
Philosopher 0 is eating...
Philosopher 3 has put down right fork...
Philosopher 3 has put down left fork...
Philosopher 3 is thinking...