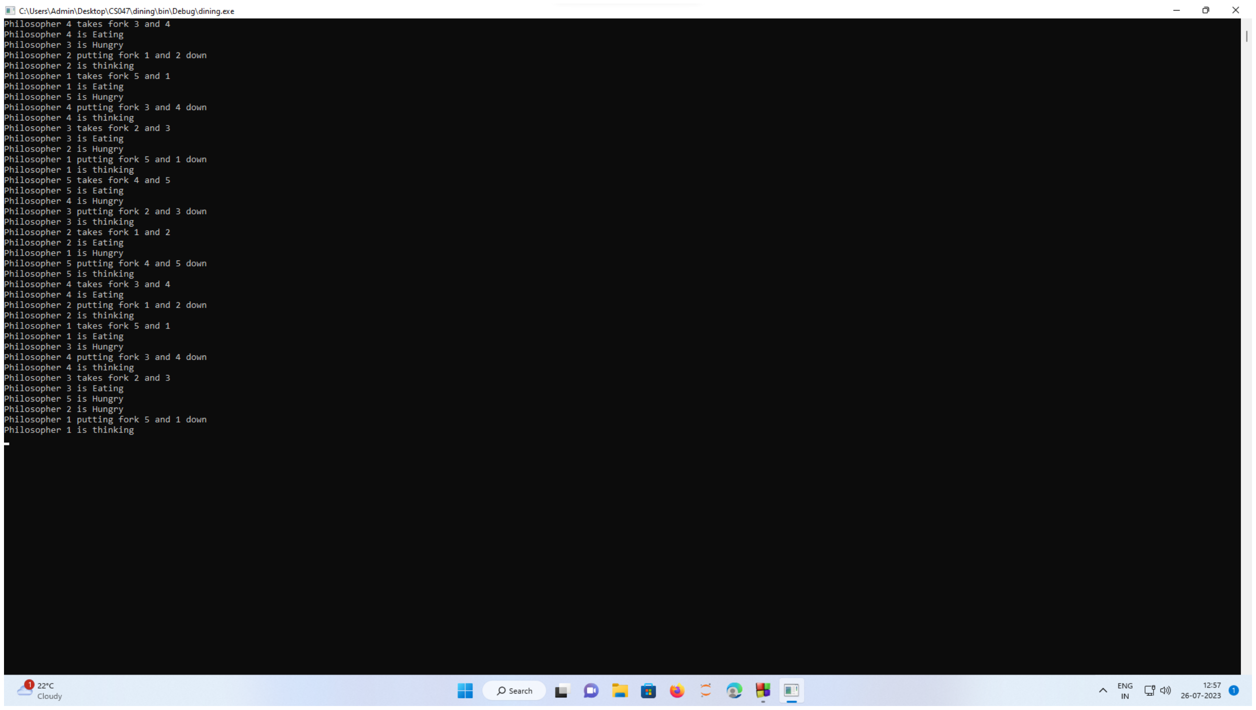
6. Write a C program to simulate the concept of Dining-Philosophers problem.

Code:

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

#define N 5
#define THINKING 2
#define HUNGRY 1
#define EATING 0
#define LEFT (phnum + 4) % N
#define RIGHT (phnum + 1) % N

int state[N];
int phil[N] = { 0, 1, 2, 3, 4 };

sem_t mutex;
sem_t S[N];

void test(int phnum)
{
    if (state[phnum] == HUNGRY
        && state[LEFT] != EATING
        && state[RIGHT] != EATING) {
        // state that eating
        state[phnum] = EATING;

        sleep(2);

        printf("Philosopher %d takes fork %d and %d\n",
                      phnum + 1, LEFT + 1, phnum + 1);

        printf("Philosopher %d is Eating\n", phnum + 1);

        // sem_post(&S[phnum]) has no effect
        // during takefork
        // used to wake up hungry philosophers
        // during putfork
        sem_post(&S[phnum]);
    }
}
```

```
// take up chopsticks
void take_fork(int phnum)
{

   sem_wait(&mutex);

   // state that hungry
   state[phnum] = HUNGRY;

   printf("Philosopher %d is Hungry\n", phnum + 1);

   // eat if neighbours are not eating
   test(phnum);

   sem_post(&mutex);

   // if unable to eat wait to be signalled
   sem_wait(&S[phnum]);

   sleep(1);
}

// put down chopsticks
void put_fork(int phnum)
{

   sem_wait(&mutex);

   // state that thinking
   state[phnum] = THINKING;

   printf("Philosopher %d putting fork %d and %d down\n",
         phnum + 1, LEFT + 1, phnum + 1);
   printf("Philosopher %d is thinking\n", phnum + 1);

   test(LEFT);
   test(RIGHT);

   sem_post(&mutex);
}

void* philosopher(void* num)
{
   while (1) {

       int* i = num;

       sleep(1);

       take_fork(*i);

       sleep(0);

       put_fork(*i);
   }
}

int main()
{

   int i;
   pthread_t thread_id[N];

   // initialize the semaphores
   sem_init(&mutex, 0, 1);

   for (i = 0; i < N; i++)

       sem_init(&S[i], 0, 0);

   for (i = 0; i < N; i++) {

       // create philosopher processes
       pthread_create(&thread_id[i], NULL,
                      philosopher, &phil[i]);

       printf("Philosopher %d is thinking\n", i + 1);
   }

   for (i = 0; i < N; i++)

       pthread_join(thread_id[i], NULL);
}
```

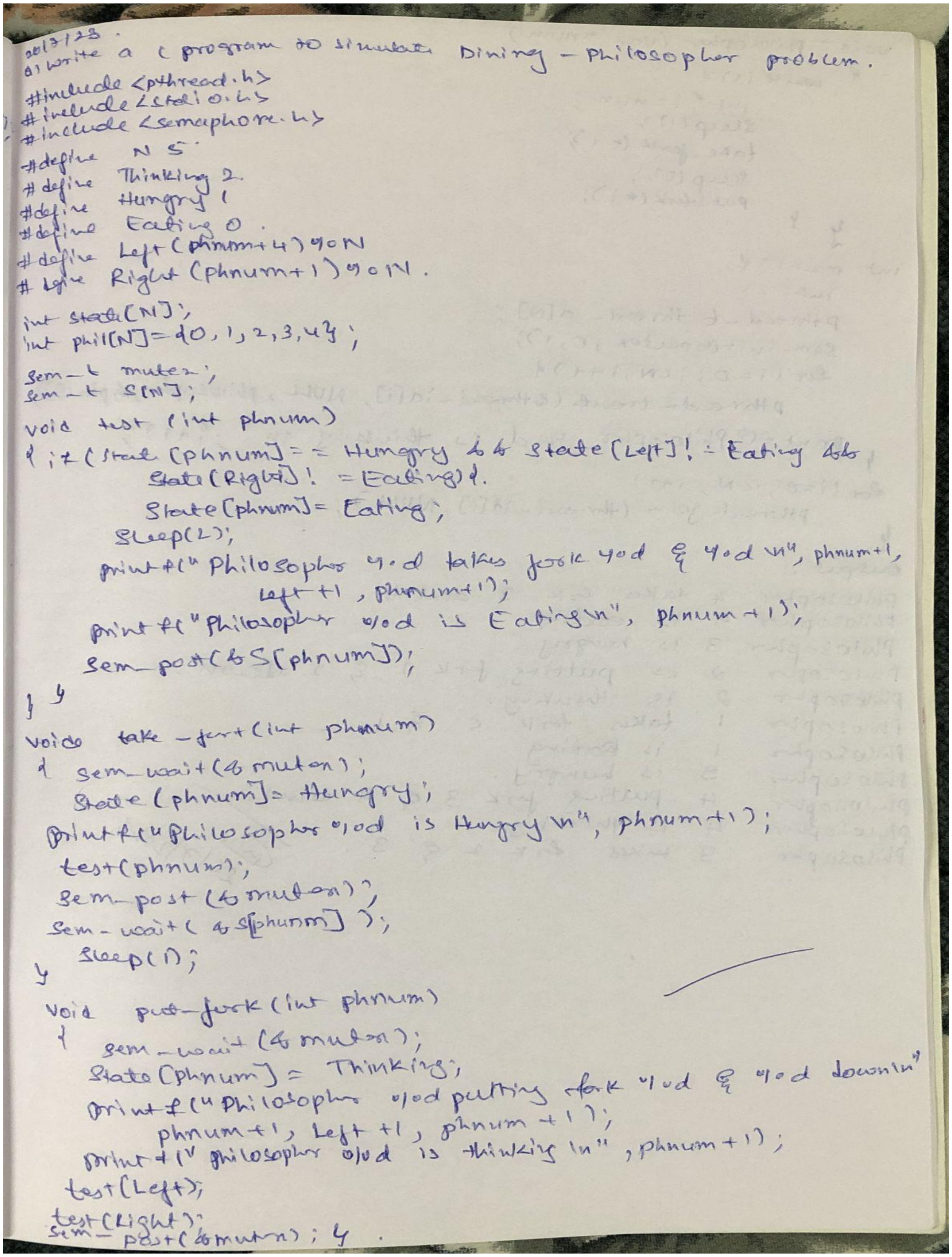
Output:



C:\Users\Admin\Desktop\CS047\dining\bin\Debug\dining.exe

```
Philosopher 4 takes fork 3 and 4
Philosopher 4 is Eating
Philosopher 3 is Hungry
Philosopher 2 putting fork 1 and 2 down
Philosopher 2 is thinking
Philosopher 1 takes fork 5 and 1
Philosopher 1 is Eating
Philosopher 5 is Hungry
Philosopher 4 putting fork 3 and 4 down
Philosopher 4 is thinking
Philosopher 3 takes fork 2 and 3
Philosopher 3 is Eating
Philosopher 2 is Hungry
Philosopher 1 putting fork 5 and 1 down
Philosopher 1 is thinking
Philosopher 5 takes fork 4 and 5
Philosopher 5 is Eating
Philosopher 4 is Hungry
Philosopher 3 putting fork 2 and 3 down
Philosopher 3 is thinking
Philosopher 2 takes fork 1 and 2
Philosopher 2 is Eating
Philosopher 1 is Hungry
Philosopher 5 putting fork 4 and 5 down
Philosopher 5 is thinking
Philosopher 4 takes fork 3 and 4
Philosopher 4 is Eating
Philosopher 2 putting fork 1 and 2 down
Philosopher 2 is thinking
Philosopher 1 takes fork 5 and 1
Philosopher 1 is Eating
Philosopher 3 is Hungry
Philosopher 4 putting fork 3 and 4 down
Philosopher 4 is thinking
Philosopher 3 takes fork 2 and 3
Philosopher 3 is Eating
Philosopher 5 is Hungry
Philosopher 2 is Hungry
Philosopher 1 putting fork 5 and 1 down
Philosopher 1 is thinking
```

Observation:



20/7/23.
Q) Write a C program to simulate Dining - Philosopher problem.

```
#include <pthread.h>
#include <stdio.h>
#include <semaphore.h>
#define  N 5.
#define  Thinking 2.
#define  Hungry 1
#define  Eating 0.
#define  Left (phnum+4)%N
#define  Right (phnum+1)%N.

int state[N];
int phil[N]={0,1,2,3,4};

sem_t mutex;
sem_t S[N];

void test (int phnum)
{ if (state [phnum]== Hungry && state [Left]!= Eating &&
        State (Right]!= Eating){.
        State[phnum]= Eating;
      Sleep(2);
     printf(" Philosopher %d takes fork %d & %d \n", phnum+1,
               Left +1 , phnum+1);
    printf(" Philosopher %d is Eating\n", phnum+1);
   sem_post(&S[phnum]);
} }

void take-fork(int phnum)
{ sem_wait(& mutex);
  State (phnum]= Hungry;
 printf("Philosopher %d is Hungry \n", phnum+1);
  test(phnum);
  sem_post (& mutex);
 sem - wait ( & S[phunm] );
   sleep(1);
}

void   put_fork (int phnum)
 {   sem_wait (& mutex);
   State (phnum] = Thinking;
   printf(" Philosopher %d putting fork %d & %d down\n"
          phnum+1, Left +1, phnum + 1);
  printf(" philosopher %d is thinking \n" , phnum + 1);
 test(Left);
 test(Right);
sem_ post(&mutex); }.
```

```
void * philosopher (void *num)
  { while (1) {
        int * i=num;
        sleep(1);
        take_fork (*i);
        sleep (0);
        put_fork (*i);
  } }
int main() {
    int i;
    pthread_t thread_id[N];
    sem_init(&mutex, 0, 1);
    for (i=0; i<N; i++) {
        pthread_create (&thread_id[i], NULL, philosopher, &phil[i]);
    printf("Philosopher %d is thinking \n"; i+1);
 }
 for (i=0; i<N; i++)
      pthread_join (thread_id[i], NULL);
}.
```

Output:

```
Philosopher 4 takes fork 3 and 4
Philosopher 4 is eating.
Philosopher 3 is hungry.
Philosopher 2 is putting fork 1 & 2 down.
Philosopher 2 is thinking.
Philosopher 1 takes fork 5 & 1
Philosopher 1 is eating.
Philosopher 3 is hungry.
Philosopher 4 putting fork 3 & 4 down.
Philosopher 4 is thinking.
Philosopher. 3 takes fork 2 & 3.
```

[illegible] 26/7/2023