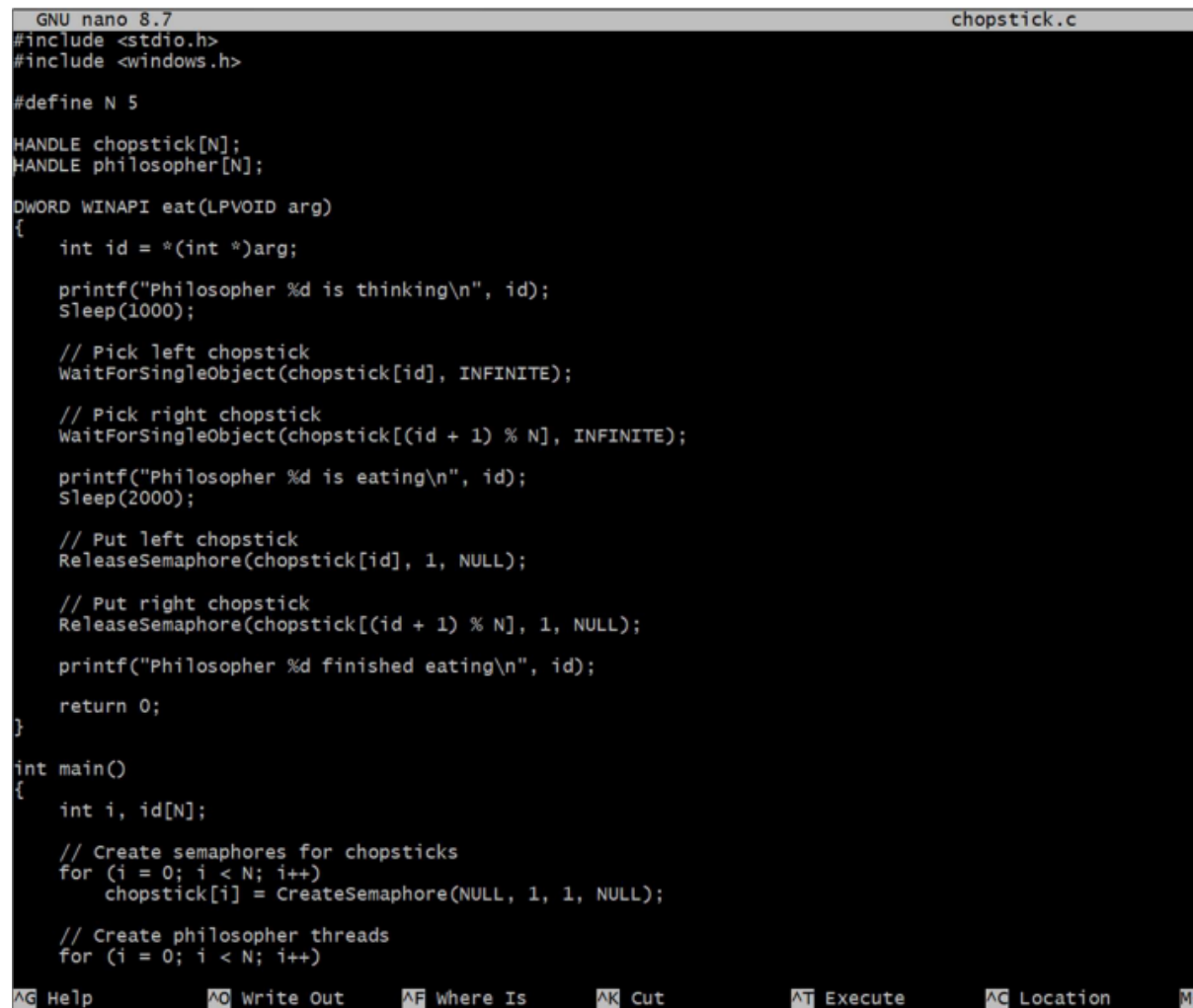


Practical 6

Aim: Considered there are N philosophers seated around a circular table with one chopstick between each pair of philosophers. There is one chopstick between each philosopher. A philosopher may eat if he can pick up the two chopsticks adjacent to him. One chopstick may be picked up by any one of its adjacent followers but not both. Write a program to solve the problem using process synchronization technique.



```
GNU nano 8.7 chopstick.c
#include <stdio.h>
#include <windows.h>

#define N 5

HANDLE chopstick[N];
HANDLE philosopher[N];

DWORD WINAPI eat(LPVOID arg)
{
    int id = *(int *)arg;

    printf("Philosopher %d is thinking\n", id);
    Sleep(1000);

    // Pick left chopstick
    WaitForSingleObject(chopstick[id], INFINITE);

    // Pick right chopstick
    WaitForSingleObject(chopstick[(id + 1) % N], INFINITE);

    printf("Philosopher %d is eating\n", id);
    Sleep(2000);

    // Put left chopstick
    ReleaseSemaphore(chopstick[id], 1, NULL);

    // Put right chopstick
    ReleaseSemaphore(chopstick[(id + 1) % N], 1, NULL);

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

    return 0;
}

int main()
{
    int i, id[N];

    // Create semaphores for chopsticks
    for (i = 0; i < N; i++)
        chopstick[i] = CreateSemaphore(NULL, 1, 1, NULL);

    // Create philosopher threads
    for (i = 0; i < N; i++)
```



```
Barkha@DESKTOP-35GK287 MSYS ~  
$ nano dinning.c  
  
Barkha@DESKTOP-35GK287 MSYS ~  
$ gcc dinning.c -lpthread -o dinning  
./dinning  
Philosopher 0 is thinking  
Philosopher 1 is thinking  
Philosopher 2 is thinking  
Philosopher 3 is thinking  
Philosopher 4 is thinking  
Philosopher 0 is eating  
Philosopher 2 is eating  
Philosopher 4 is eating  
Philosopher 0 finished eating  
Philosopher 2 finished eating  
Philosopher 1 is eating  
Philosopher 3 is eating  
Philosopher 4 finished eating  
Philosopher 1 finished eating  
Philosopher 3 finished eating  
  
Barkha@DESKTOP-35GK287 MSYS ~  
$ |
```