Mainak Chattopadhyay
21BAI1217
OS LAB 12

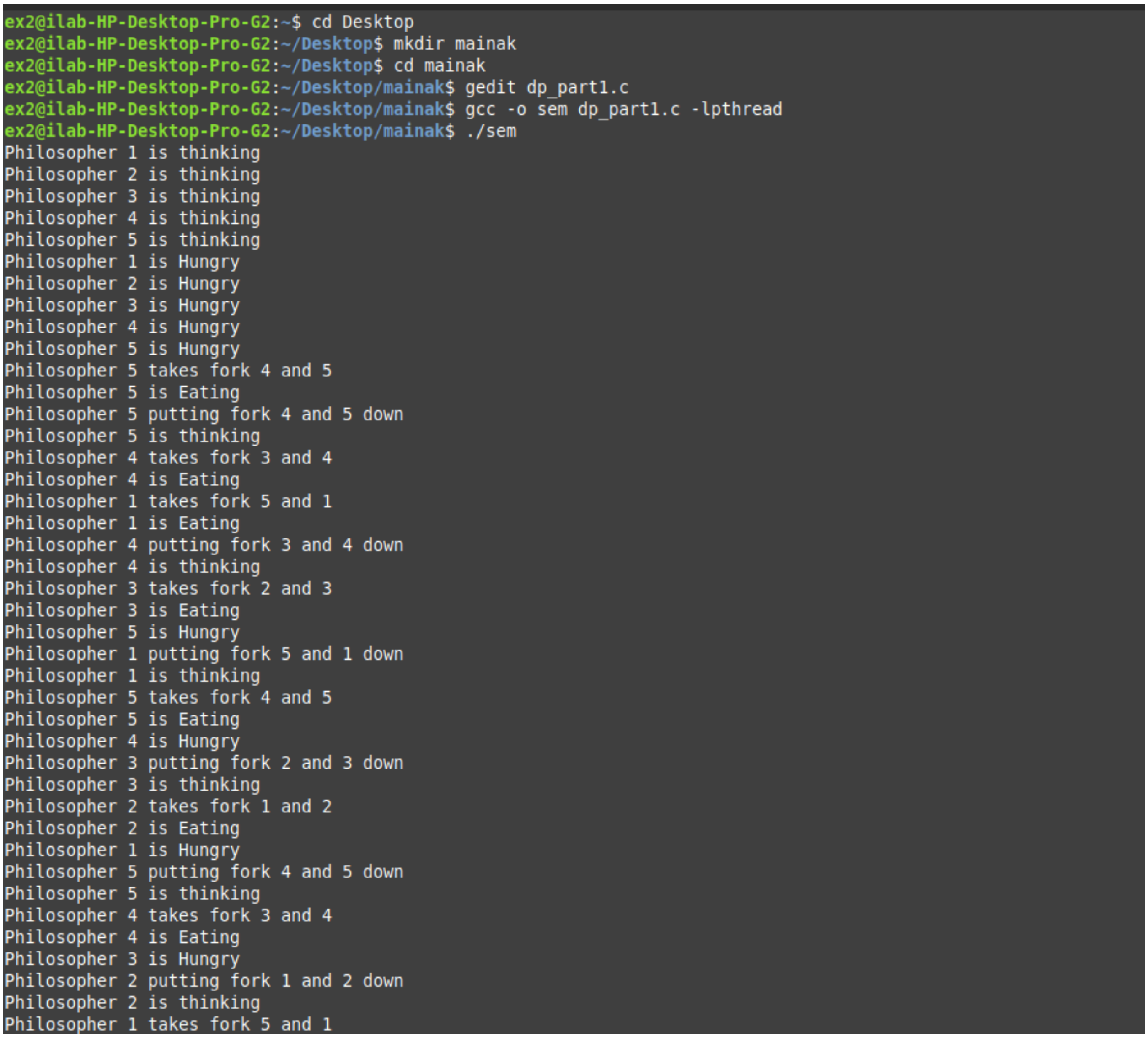
Dining Philosophers Problem using Semaphores

CODE

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



```
ex2@ilab-HP-Desktop-Pro-G2:~$ cd Desktop
ex2@ilab-HP-Desktop-Pro-G2:~/Desktop$ mkdir mainak
ex2@ilab-HP-Desktop-Pro-G2:~/Desktop$ cd mainak
ex2@ilab-HP-Desktop-Pro-G2:~/Desktop/mainak$ gedit dp_part1.c
ex2@ilab-HP-Desktop-Pro-G2:~/Desktop/mainak$ gcc -o sem dp_part1.c -lpthread
ex2@ilab-HP-Desktop-Pro-G2:~/Desktop/mainak$ ./sem
Philosopher 1 is thinking
Philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 4 is thinking
Philosopher 5 is thinking
Philosopher 1 is Hungry
Philosopher 2 is Hungry
Philosopher 3 is Hungry
Philosopher 4 is Hungry
Philosopher 5 is Hungry
Philosopher 5 takes fork 4 and 5
Philosopher 5 is Eating
Philosopher 5 putting fork 4 and 5 down
Philosopher 5 is thinking
Philosopher 4 takes fork 3 and 4
Philosopher 4 is Eating
Philosopher 1 takes fork 5 and 1
Philosopher 1 is Eating
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 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 3 is Hungry
Philosopher 2 putting fork 1 and 2 down
Philosopher 2 is thinking
Philosopher 1 takes fork 5 and 1
```

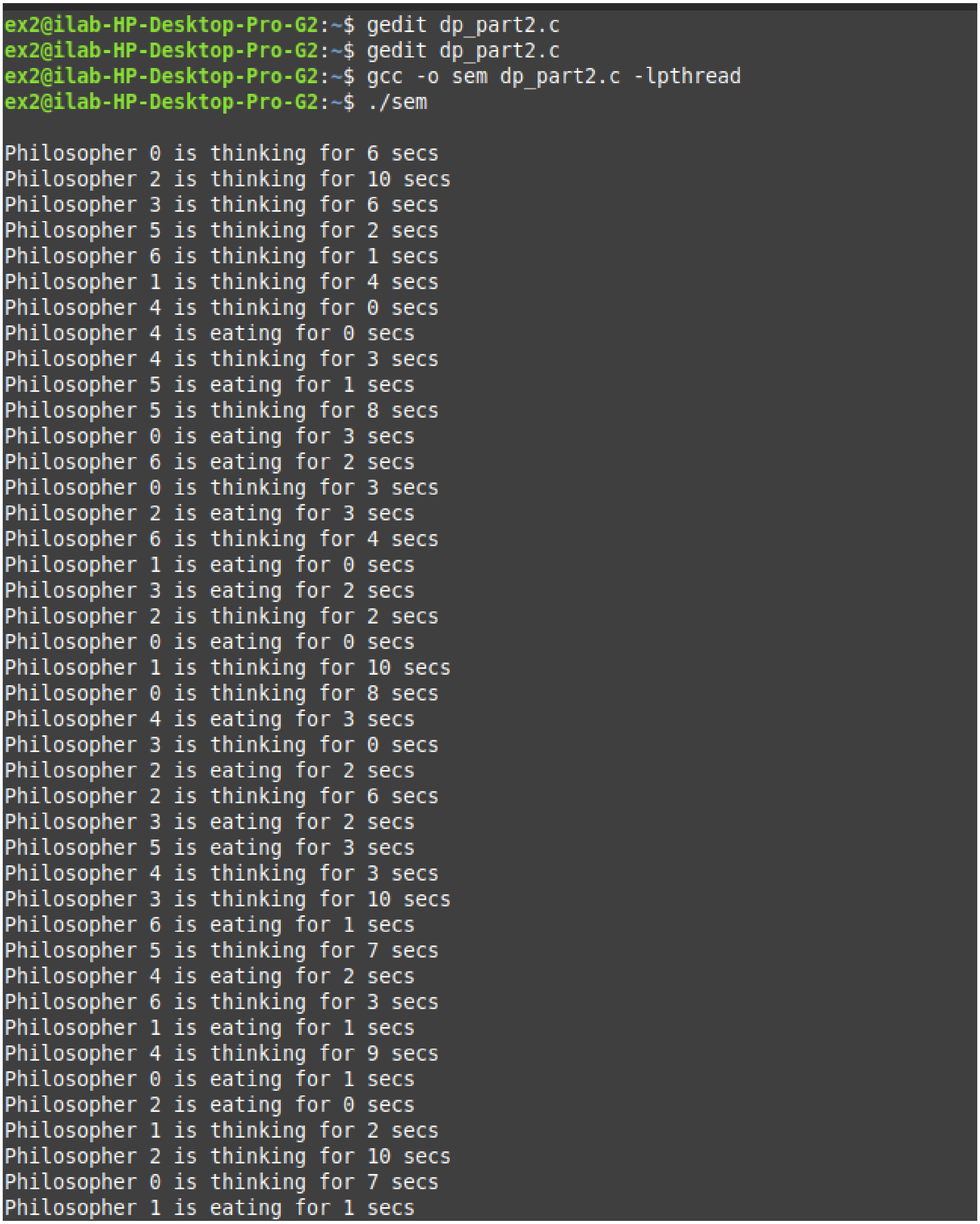
Dining Philosophers Problem using Monitors

CODE

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

#define N 7
#define THINKING 0
#define HUNGRY 1
#define EATING 2
#define LEFT (i+N-1)%N
#define RIGHT (i+1)%N

void initialization();
void test(int i);
void take_chopsticks(int i);
void put_chopsticks(int i);

//semaphores to implement monitor
sem_t mutex;
sem_t next;
//count varaible for philosophers waiting on semaphore next
int next_count = 0;

//implementing condition variable using semaphore
//semaphore and integer variable replacing condition variable
typedef struct
{
        sem_t sem;
        //count variable for philosophers waiting on condition semaphore sem
        int count;
}condition;
condition x[N];

//state of each philosopher(THINKING, HUNGRY or EATING)
int state[N];

//turn variable corresonding to each chopstick
//if philosopher i wants to each the turn[i] and turn[LEFT]must be set to i
int turn[N];

//wait on condition
void wait(int i)
```

```
{
    x[i].count++;
    if(next_count > 0)
    {
        //signal semaphore next
        sem_post(&next);
    }
    else
    {
        //signal semaphore mutex
        sem_post(&mutex);
    }
    sem_wait(&x[i].sem);
    x[i].count--;
    //  printf("\nX.count -> %d",x.count);
}

//signal on condition
void signal(int i)
{
    if(x[i].count > 0)
    {
        next_count++;
        //signal semaphore x[i].sem
        sem_post(&x[i].sem);
        //wait semeaphore next
        sem_wait(&next);
        next_count--;
    }
}
void test(int i)
{
    if(state[i] == HUNGRY && state[LEFT] != EATING && state[RIGHT] != EATING && turn[i] == i &&
turn[LEFT] == i)
    {
        state[i] = EATING;

        //signal on condition
        signal(i);

        /*  printf("\nNext Count -> %d, X_count -> %d,state[%d] -> %d,state[%d] ->
%d,state[%d] -> %d",
            next_count,x[i].count,i,state[i],LEFT,state[LEFT],RIGHT,state[RIGHT]);*/

    }
}

void take_chopsticks(int i)
{
    //wait semaphore mutex
    sem_wait(&mutex);
    state[i] = HUNGRY;
    test(i);
    while(state[i] == HUNGRY)
    {
        //printf("\nThread %d is waiting on condition",i);

        //wait on condition
        wait(i);
    }
    if(next_count > 0)
    {
        //signal semaphore next
        sem_post(&next);
    }
    else
    {
        //signal semaphore mutex
        sem_post(&mutex);
    }
}

void put_chopsticks(int i)
{
    //wait semaphore mutex
    sem_wait(&mutex);
    state[i] = THINKING;
    //set turn variable pointing to LEFT and RIGHT philosophers
    turn[i] = RIGHT;
    turn[LEFT] = LEFT;

    test(LEFT);
    test(RIGHT);

    if(next_count > 0)
    {
        //signal semaphore next
        sem_post(&next);
    }
    else
    {
        //signal semaphore mutex
        sem_post(&mutex);
    }
}

void initialization()
{
    int i;
    sem_init(&mutex,0,1);
    sem_init(&next,0,0);
    for(i = 0;i < N;i++)
    {
        state[i] = THINKING;
        sem_init(&x[i].sem,0,0);
        x[i].count = 0;
        turn[i] = i;
    }
    //setting turn variables such that Philosophers 0,2 or 4 can grab both chopsticks initially
    turn[1] = 2;
    turn[3] = 4;
    turn[6] = 0;

}

//pthread_mutex_t lock;

void *philosopher(void *i)
{
    while(1)
    {
        //variable representing philosopher
        int self = *(int *) i;
        int j,k;
        j = rand();
        j = j % 11;
        printf("\nPhilosopher %d is thinking for %d secs",self,j);
        sleep(j);
        //philosopher take chopsticks
        take_chopsticks(self);
        k = rand();
        k = k % 4;
        printf("\nPhilosopher %d is eating for %d secs",self,k);
        sleep(k);
        //philosopher release chopsticks
        put_chopsticks(self);
    }

}

int main()
{
    int i, pos[N];
    //one thread corresponding to each philosopher
    pthread_t thread[N];
    pthread_attr_t attr;
    //initilize semaphore and other variables
    initialization();

    pthread_attr_init(&attr);

    for (i = 0; i < N; i++)
    {
        pos[i] = i;
        //create thread corresponding to each philosopher
        pthread_create(&thread[i], NULL,philosopher, (int *) &pos[i]);
    }
    for (i = 0; i < N; i++)
    {
        pthread_join(thread[i], NULL);
    }

    return 0;
}
```

OUTPUT



```
ex2@ilab-HP-Desktop-Pro-G2:~$ gedit dp_part2.c
ex2@ilab-HP-Desktop-Pro-G2:~$ gedit dp_part2.c
ex2@ilab-HP-Desktop-Pro-G2:~$ gcc -o sem dp_part2.c -lpthread
ex2@ilab-HP-Desktop-Pro-G2:~$ ./sem

Philosopher 0 is thinking for 6 secs
Philosopher 2 is thinking for 10 secs
Philosopher 3 is thinking for 6 secs
Philosopher 5 is thinking for 2 secs
Philosopher 6 is thinking for 1 secs
Philosopher 1 is thinking for 4 secs
Philosopher 4 is thinking for 0 secs
Philosopher 4 is eating for 0 secs
Philosopher 4 is thinking for 3 secs
Philosopher 5 is eating for 1 secs
Philosopher 5 is thinking for 8 secs
Philosopher 0 is eating for 3 secs
Philosopher 6 is eating for 2 secs
Philosopher 0 is thinking for 3 secs
Philosopher 2 is eating for 3 secs
Philosopher 6 is thinking for 4 secs
Philosopher 1 is eating for 0 secs
Philosopher 3 is eating for 2 secs
Philosopher 2 is thinking for 2 secs
Philosopher 0 is eating for 0 secs
Philosopher 1 is thinking for 10 secs
Philosopher 0 is thinking for 8 secs
Philosopher 4 is eating for 3 secs
Philosopher 3 is thinking for 0 secs
Philosopher 2 is eating for 2 secs
Philosopher 2 is thinking for 6 secs
Philosopher 3 is eating for 2 secs
Philosopher 5 is eating for 3 secs
Philosopher 4 is thinking for 3 secs
Philosopher 3 is thinking for 10 secs
Philosopher 6 is eating for 1 secs
Philosopher 5 is thinking for 7 secs
Philosopher 4 is eating for 2 secs
Philosopher 6 is thinking for 3 secs
Philosopher 1 is eating for 1 secs
Philosopher 4 is thinking for 9 secs
Philosopher 0 is eating for 1 secs
Philosopher 2 is eating for 0 secs
Philosopher 1 is thinking for 2 secs
Philosopher 2 is thinking for 10 secs
Philosopher 0 is thinking for 7 secs
Philosopher 1 is eating for 1 secs
```