

6. [Process Synchronization]

Considered there are N philosophers seated around a circular table CO3 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

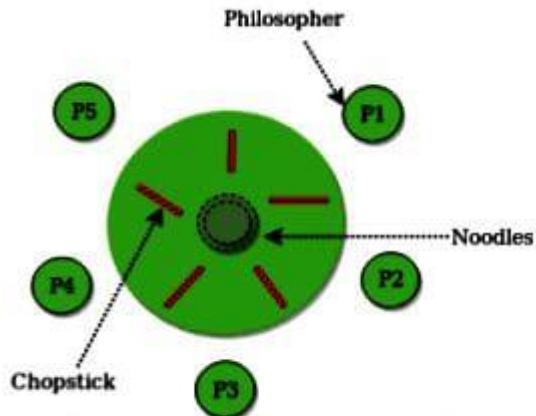


Fig: 1. Dining Philosopher Problem

CODE:

```
M ~
GNU nano 8.7
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>

#define N 5

sem_t chopstick[N];
pthread_t philosopher[N];

void* eat(void* arg) {
    int id = *(int*)arg;
    printf("Philosopher %d is thinking\n", id);
    sleep(1);

    sem_wait(&chopstick[id]);
    sem_wait(&chopstick[(id + 1) % N]);
    printf("Philosopher %d is eating\n", id);
    sleep(1);

    sem_post(&chopstick[id]);
    sem_post(&chopstick[(id + 1) % N]);
    printf("Philosopher %d finished eating\n", id);
    return NULL;
}

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

    for (i = 0; i < N; i++)
        sem_init(&chopstick[i], 0, 1);

    for (i = 0; i < N; i++) {
        id[i] = i;
        pthread_create(&philosopher[i], NULL, eat, &id[i]);
    }

    for (i = 0; i < N; i++)
        pthread_join(philosopher[i], NULL);

    return 0;
}
```

The screenshot shows a terminal window titled "chopstick.c" containing the C code for a philosopher problem. The code uses semaphores and threads to simulate five philosophers eating chopsticks. The file is saved as "chopstick.c".

```
GNU nano 8.7
#include <unistd.h>
#define N 5
sem_t chopstick[N];
pthread_t philosopher[N];
void* eat(void* arg) {
    int id = *(int*)arg;
    printf("Philosopher %d is thinking\n", id);
    sleep(1);
    sem_wait(&chopstick[id]);
    sem_wait(&chopstick[(id + 1) % N]);
    printf("Philosopher %d is eating\n", id);
    sleep(1);
    sem_post(&chopstick[id]);
    sem_post(&chopstick[(id + 1) % N]);
    printf("Philosopher %d finished eating\n", id);
    return NULL;
}
int main() {
    int i, id[N];
    for (i = 0; i < N; i++)
        sem_init(&chopstick[i], 0, 1);
    for (i = 0; i < N; i++) {
        id[i] = i;
        pthread_create(&philosopher[i], NULL, eat, &id[i]);
    }
    for (i = 0; i < N; i++)
        pthread_join(philosopher[i], NULL);
    return 0;
}
```

At the bottom of the terminal window, there is a menu bar with various options like Help, Write Out, Read File, etc.

OUTPUT:

The screenshot shows a terminal window on a Mingw64 system. It starts by running "nano chopstick.c" to edit the source code. Then, it compiles the program with "gcc chopstick.c -o chopstick -pthread". Finally, it runs the compiled program with "./chopstick", which outputs the sequence of thoughts and meals for five philosophers.

```
ASUS@Kalash-Laptop MINGW64 ~
$ nano chopstick.c

ASUS@Kalash-Laptop MINGW64 ~
$ gcc chopstick.c -o chopstick -pthread

ASUS@Kalash-Laptop MINGW64 ~
$ ./chopstick
Philosopher 0 is thinking
Philosopher 1 is thinking
Philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 4 is thinking
Philosopher 4 is eating
Philosopher 3 is eating
Philosopher 4 finished eating
Philosopher 2 is eating
Philosopher 3 finished eating
Philosopher 1 is eating
Philosopher 2 finished eating
Philosopher 0 is eating
Philosopher 1 finished eating
Philosopher 0 finished eating

ASUS@Kalash-Laptop MINGW64 ~
$ |
```