YCCE

Vision

"To become the most preferred institution providing innovative, research and value based, professional education for the society at large".

Mission

YCCE is committed to

- Attract best talent and create learning ambience
- Practice Innovative teaching-learning & research
- Integrate Industry-Institute Collaborations
- Nurture students towards holistic development and choicest career

Department

Vision of the Department

To be a well-known center for pursuing computer education through innovative pedagogy, value-based education and industry collaboration.

Mission of the Department

To establish learning ambience for ushering in computer engineering professionals in core and multidisciplinary arena by developing problem-solving skills through emerging technologies.

Nagar Yuwak Shikshan Sanstha's

Yeshwantrao Chavan College of Engineering

(An Autonomous Institution affiliated to Rashtrasant Tukadoji Maharaj Nagpur University)

Hingna Road, Wanadongri, Nagpur - 441 110

Ph.: 07104-237919, 234623, 329249, 329250 Fax: 07104-232376, Website: www.ycce.edu

Department of Computer Science and Engineering (IOT)

23CT1402	Lab: Operating Systems
Name of the Student: Puneet Raut	Semester/ Section: 5 A
Roll No: 58	Enrollment Number: 23070647

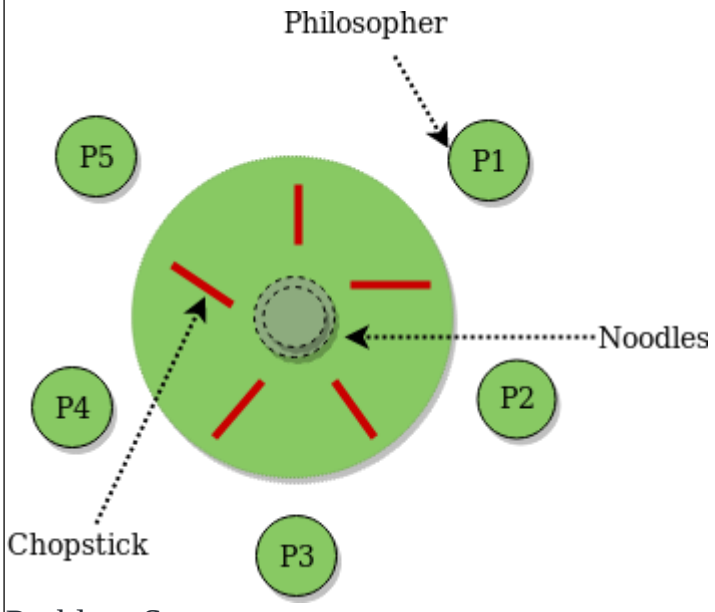
Sr. No.	COs	POs											PSOs	
		Course Outcomes	1	2	3	4	5	6	7	8	9	10	11	PSO 1	PSO2
1	CO1	Demonstrate the ability to execute Linux process management, memory management, and shell commands to manage system resources efficiently.	3	3	3	-		-	-	-	-	-	3	3	
2	CO2	Develop programs utilizing system calls, thread programming, and page replacement algorithms to simulate and analyze operating system functionalities.	3	3	3	-	-	-	-	-	-	-	3	3	
3	CO3	Design and implement process scheduling, memory allocation, and deadlock detection algorithms to address real-world operating system challenges.	3	3	3	-		-	-	-	-	-	3	3	
		Avg		3	3	-		-	-	-	-	-	3	3	

Practical No. 8

Aim: Develop a program to provide synchronization among the 5 philosophers in Dining Philosophers problem using semaphore

Theory:

Dining Philosopher Solution using Semaphores



Problem Statement

- K philosophers sit around a circular table.
- Each philosopher alternates between thinking and eating.
- There is one chopstick between each philosopher (total K chopsticks).
- A philosopher must pick up two chopsticks (left and right) to eat.
- Only one philosopher can use a chopstick at a time.

The challenge: Design a synchronization mechanism so that philosophers can eat without causing **deadlock** (all waiting forever) or **starvation** (some never get a chance to eat).

Issues in the Problem

1. **Deadlock:** If every philosopher picks up their left chopstick first, no one can pick up the right one circular wait.
2. **Starvation:** Some philosophers may never get a chance to eat if others keep eating.
3. **Concurrency Control:** Must ensure no two adjacent philosophers eat simultaneously.

Semaphore Solution to Dining Philosopher

We use **semaphores** to manage chopsticks and avoid deadlock.

Algorithm

- Each chopstick is represented as a binary semaphore (mutex).
- Philosopher must acquire both left and right semaphores before eating.
- After eating, the philosopher releases both semaphores.

Code:

```
#include <iostream>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
using namespace std;

#define NUM_PHILOSOPHERS 5

sem_t chopstick[NUM_PHILOSOPHERS];

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

    while (true) {
```

```
        cout << "Philosopher " << id << " is thinking..." << endl;
        sleep(1);

        if (id % 2 == 0) {
            sem_wait(&chopstick[id]);
            sem_wait(&chopstick[(id + 1) % NUM_PHILOSOPHERS]);
        } else {
            sem_wait(&chopstick[(id + 1) % NUM_PHILOSOPHERS]);
            sem_wait(&chopstick[id]);
        }

        cout << "Philosopher " << id << " is eating " << endl; sleep(2);

        sem_post(&chopstick[id]);
        sem_post(&chopstick[(id + 1) % NUM_PHILOSOPHERS]);

        cout << "Philosopher " << id << " has finished eating and put down
chopsticks.\n" << endl;
        sleep(1);
    }
    return nullptr;
```

```
}

int main() {
    pthread_t philosophers[NUM_PHILOSOPHERS];
    int ids[NUM_PHILOSOPHERS];


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


    for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
        ids[i] = i;
        pthread_create(&philosophers[i], nullptr, philosopher, &ids[i]);
    }


    for (int i = 0; i < NUM_PHILOSOPHERS; i++)
        pthread_join(philosophers[i], nullptr);


    for (int i = 0; i < NUM_PHILOSOPHERS; i++)
```

```
        sem_destroy(&chopstick[i]);


    return 0;
}
```

OUTPUT (SCREEN SHOT) IF ANY:

main.cpp

```cpp
#include <iostream>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
using namespace std;

#define NUM_PHILOSOPHERS 5

sem_t chopstick[NUM_PHILOSOPHERS];

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

    while (true) {
        cout << "Philosopher " << id << " is thinking..." << endl;
        sleep(1);

        if (id % 2 == 0) {
            sem_wait(&chopstick[id]);
            sem_wait(&chopstick[(id + 1) % NUM_PHILOSOPHERS]);
        } else {
            sem_wait(&chopstick[(id + 1) % NUM_PHILOSOPHERS]);
            sem_wait(&chopstick[id]);
        }

        cout << "Philosopher " << id << " is eating " << endl;
        sleep(2);

        sem_post(&chopstick[id]);
        sem_post(&chopstick[(id + 1) % NUM_PHILOSOPHERS]);

        cout << "Philosopher " << id << " has finished eating and put down chopsticks.\n" << endl;
        sleep(1);
    }
    return nullptr;
```

```
Philosopher 1 is thinking...
Philosopher Philosopher 2 is thinking...
Philosopher 3 is thinking...
Philosopher 4 is thinking...
0 is thinking...
Philosopher 1 is eating
Philosopher 4 is eating
Philosopher 1 has finished eating and put down chopsticks.

Philosopher 2 is eating
Philosopher 4 has finished eating and put down chopsticks.

Philosopher 0 is eating
Philosopher 1 is thinking...
Philosopher 4 is thinking...
```

GitHub- https://github.com/Puneet4382/Operating-System-Practicals

Conclusion: Develop a program to provide synchronization among the 5 philosophers in Dining Philosophers problem using semaphore is done successfully