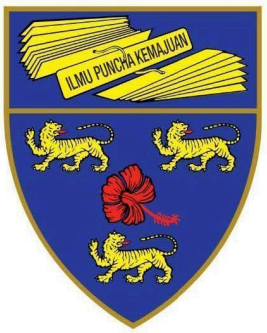


UNIVERSITY OF MALAYA

WIA2004 OPERATING SYSTEMS

LAB REPORT

SEMESTER 2, 2023/2024

Occurrence 3

Lab 9 - The concept of Dining-Philosophers

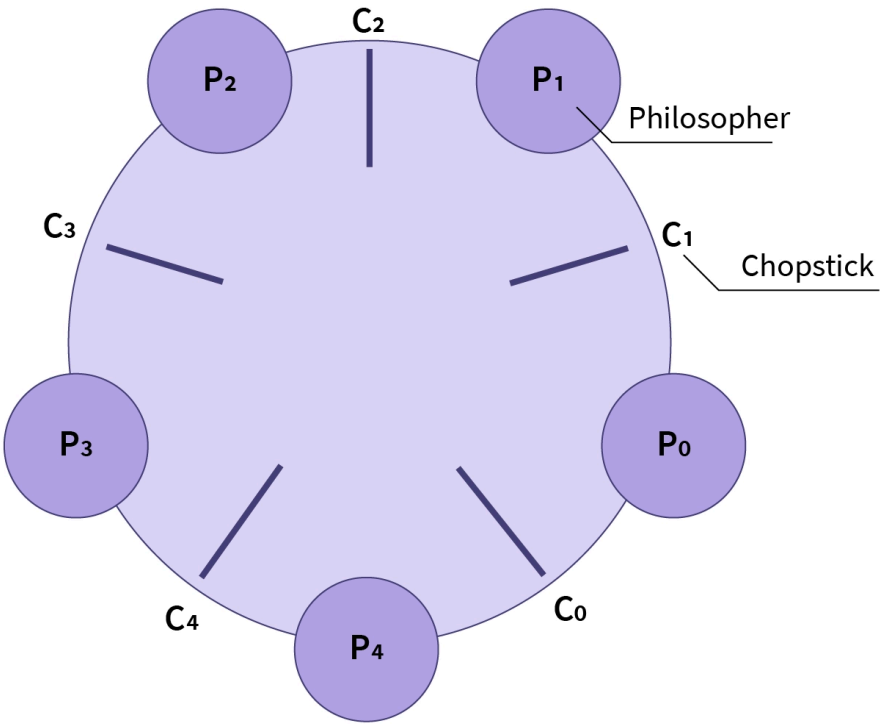
GROUP 2

NAME	MATRIC NUMBER
AINA NUR MALISA BINTI AZIZI	U2101529
MUHAMMAD DANIEL BIN ABD.RAZAK	U2101081
EISRAQ REJAB BIN HASSAN	U2102340
NUR QISTINA IMANI BINTI MOHAMMAD FADLY	U2101068
YOURISHA SOFEA BINTI MOHD YOUZAIMI	U2102065

Table Of Content

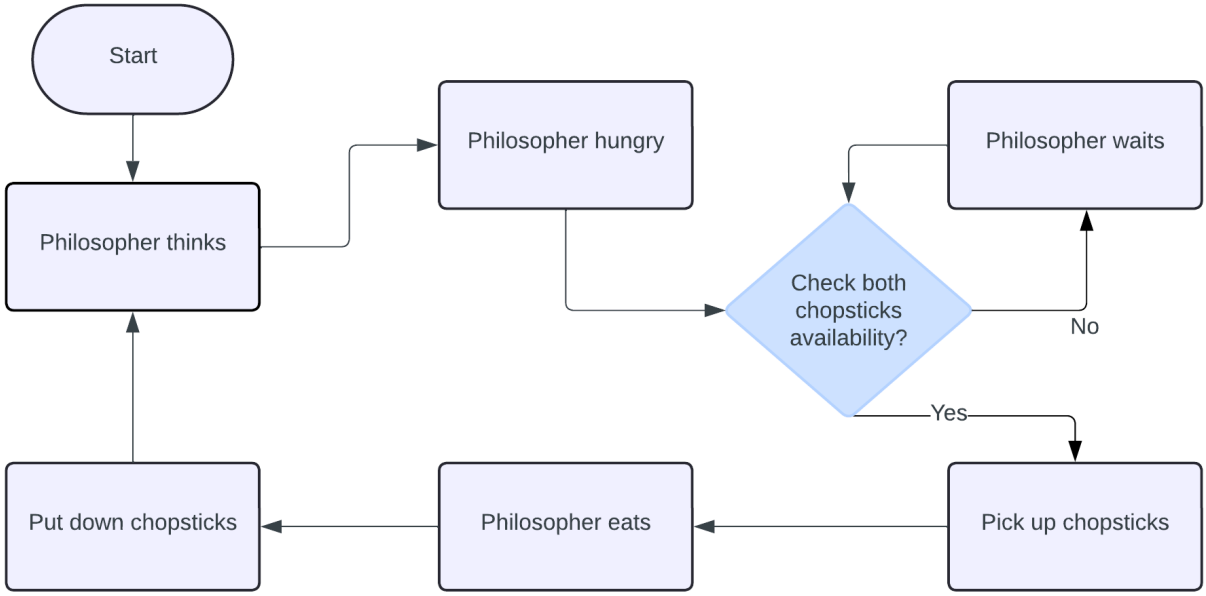
Introduction To Algorithm

- The Dining Philosophers problem is a classic synchronization problem that demonstrates the challenges of allocating resources among concurrent processes in a manner that avoids deadlock and starvation. In this problem, five philosophers sit at a table with a bowl of rice in the center and five chopsticks placed between them. Philosophers alternate between thinking and eating. To eat, a philosopher needs to pick up both the chopsticks adjacent to them, which can lead to a deadlock if all philosophers pick up their left chopstick simultaneously.



- To solve this problem, we need to ensure that the philosophers can eat without leading to a deadlock or starvation. Several strategies can be implemented to achieve this, including the use of semaphores, mutexes, or monitors to control access to the chopsticks.

Flow Chart


Start
Philosopher thinks
Philosopher hungry
Philosopher waits
Check both chopsticks availability?
No
Yes
Put down chopsticks
Philosopher eats
Pick up chopsticks

Pseudo Code

```
initialize chopsticks as array of mutexes

function philosopher(id)
    while true
        think()
        pick_up_chopsticks(id)
        eat()
        put_down_chopsticks(id)

function pick_up_chopsticks(id)
    if id is even
        lock chopstick[id]
        lock chopstick[(id + 1) % 5]
    else
        lock chopstick[(id + 1) % 5]
        lock chopstick[id]

function put_down_chopsticks(id)
    unlock chopstick[id]
    unlock chopstick[(id + 1) % 5]
```

1. Create an array of mutexes, each representing a chopstick, to control access and ensure mutual exclusion.
2. Define a function for the philosopher's actions, taking the philosopher's ID as an argument.
3. Start an infinite loop to simulate the philosopher's continuous cycle of thinking and eating.
4. Simulate the philosopher's thinking.
5. Call the function to pick up the two chopsticks needed for eating.
6. Simulate the philosopher eating once they have both chopsticks.
7. Call the function to put down the chopsticks after finishing eating.

8. Define a function to handle the process of picking up chopsticks.
9. Check if the philosopher's ID is even to determine the order of picking up chopsticks.
10. Lock the chopstick on the philosopher's left side if the ID is even.
11. Lock the chopstick on the philosopher's right side next.
12. Otherwise, if the philosopher's ID is odd, proceed with a different order.
13. Lock the chopstick on the philosopher's right side first if the ID is odd.
14. Lock the chopstick on the philosopher's left side next.
15. Define a function to handle putting down the chopsticks.
16. Unlock the chopstick on the philosopher's left side.
17. Unlock the chopstick on the philosopher's right side.

Coding

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

#define N 5

sem_t chopsticks[N];

void think(int philosopher) {
    printf("Philosopher %d is thinking.\n", philosopher);
    sleep(1);
}

void eat(int philosopher) {
    printf("Philosopher %d is eating.\n", philosopher);
    sleep(2);
}

void pick_up_chopsticks(int philosopher) {
    if (philosopher % 2 == 0) {
        sem_wait(&chopsticks[philosopher]);
        sem_wait(&chopsticks[(philosopher + 1) % N]);
    } else {
        sem_wait(&chopsticks[(philosopher + 1) % N]);
        sem_wait(&chopsticks[philosopher]);
    }
}

void put_down_chopsticks(int philosopher) {
    sem_post(&chopsticks[philosopher]);
    sem_post(&chopsticks[(philosopher + 1) % N]);
}

void* philosopher(void* num) {
    int philosopher = *(int*)num;
    while (1) {
        think(philosopher);
```

```
        pick_up_chopsticks(philosopher);
        eat(philosopher);
        put_down_chopsticks(philosopher);
    }
}

int main() {
    pthread_t thread_id[N];
    int philosophers[N];

    for (int i = 0; i < N; i++) {
        sem_init(&chopsticks[i], 0, 1);
    }

    for (int i = 0; i < N; i++) {
        philosophers[i] = i;
        pthread_create(&thread_id[i], NULL, philosopher, &philosophers[i]);
    }

    for (int i = 0; i < N; i++) {
        pthread_join(thread_id[i], NULL);
    }

    for (int i = 0; i < N; i++) {
        sem_destroy(&chopsticks[i]);
    }

    return 0;
}
```

Sample Output

```
Philosopher 4 is thinking.
Philosopher 3 is thinking.
Philosopher 2 is thinking.
Philosopher 1 is thinking.
Philosopher 0 is thinking.
Philosopher 3 is eating.
Philosopher 0 is eating.
Philosopher 0 is thinking.
Philosopher 3 is thinking.
Philosopher 2 is eating.
Philosopher 4 is eating.
Philosopher 2 is thinking.
Philosopher 1 is eating.
Philosopher 4 is thinking.
Philosopher 3 is eating.
Philosopher 1 is thinking.
Philosopher 0 is eating.
Philosopher 3 is thinking.
Philosopher 2 is eating.
Philosopher 0 is thinking.
Philosopher 2 is thinking.
Philosopher 4 is eating.
Philosopher 1 is eating.
Philosopher 4 is thinking.
Philosopher 3 is eating.
Philosopher 0 is eating.
Philosopher 1 is thinking.
```

Presentation Video

Link:https://drive.google.com/drive/folders/1eSaSOcHBxj_eQ6pU_hfSjiILLbhERlsE?usp=sharing

References

- GeeksforGeeks. (2024, April 21).Dining Philosophers Problem.GeeksforGeeks. https://www.geeksforgeeks.org/dining-philosophers-problem/

- Scaler.com .(2024, May 7). Dining Philosophers Problem is OS. https://www.scaler.com/topics/operating-system/dining-philosophers-problem-in-os/

- Medium. (2023, July 8) . Dining Philosophers Problem .Francesco Franco https://medium.com/@francescofranco_39234/dining-philosophers-problem-36d0030a4459