Philosophical Dining: Implementing using mutual exclusion

A MINI PROJECT REPORT FOR OS

Submitted by

Sivaram Vinod [Reg No:RA2211003011275]

Manish Kumar [Reg No: RA2211003011283]

Gagan Chethan [Reg No:RA2211003011335]

Under the Guidance of

Dr. S. POORNIMA

Assistant Professor, Department of Computing Technologies

in partial fulfillment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTING TECHNOLOGIES

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR– 603 203

November 2023

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR–603 203

BONAFIDE CERTIFICATE

Certified that 21CSC202J/Operating systems mini project report titled "**Philosophical Dining:Implementing using mutual exclusion"** is the bonafide work of **Sivaram Vinod [RegNo:RA2211003011275] , Manish Kumar [RegNo:RA2211003011283]** and **Gagan Chethan [RA2211003011335]** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported here in does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion for this or any other candidate.

Dr. S. POORNIMA
SUPERVISOR
Assistant Professor
Department of Computing Technologies

Dr. M. PUSHPALATHA
HEAD OF THE DEPARTMENT
Department of Computing Technologies

PROBLEM STATEMENT

Create 5 threads each representing a philosopher so totally 5 philosophers. Store the philosophy of any 5 famous philosophers along with their name. Design a common function that displays the philosophy of that particular philosopher who enter into thinking state after eating. Implement mutual exclusion using semaphores.

DESCRPITION ABOUT THE METHODOLOGY USED

1. Philosopher Representation:

The philosophers are represented as threads in the pthread_t philosophers[NUM_PHILOSOPHERS] array.Each philosopher is identified by a unique ID stored in the philosopher_ids array.

2. Semaphore Initialization:

Semaphores are used to represent forks and ensure that the philosophers acquire the necessary forks without conflicting with each other.There is an array of semaphores (sem_t forks[NUM_PHILOSOPHERS]) to represent the forks, and another semaphore (sem_t mutex) to control access to the critical section where philosophers grab forks.

3. Philosopher Function:

The philosopher function is the entry point for each philosopher thread.It uses a loop to simulate the philosopher's actions of grabbing forks, eating, releasing forks, and thinking.The sem_wait and sem_post functions are used to perform semaphore operations, ensuring mutual exclusion when accessing shared resources (forks).

4. Main Function:

The main function initializes the semaphores and creates philosopher threads.Each philosopher thread executes the philosopher function.The main thread waits for all philosopher threads to finish using pthread_join.

5. Randomized Sleeping:

The sleep(rand() % 3) statements simulate the time it takes for a philosopher to eat and think.The randomization adds variability to the execution, making the problem more interesting and highlighting potential synchronization issues.

6. Resource Cleanup:

After all philosopher threads finish, the code destroys the semaphores using sem_destroy to release system resources.

7. Constants and Philosophical Information:

Constants like NUM_PHILOSOPHERS and MAX_MEALS define the number of philosophers and the maximum number of meals each philosopher will have.Information about each philosopher, such as their names and philosophies, is stored in arrays.

The code demonstrates the use of semaphores to address the critical section problem in a multithreaded context, ensuring that philosophers can eat without conflicting over shared resources (forks). The random sleep times add realism to the simulation, making it more representative of a real-world scenario where execution times are unpredictable.

ALGORITHM

1. Initialize Semaphores:

   - Create an array of semaphores (forks) to represent each fork, initializing each semaphore with a value of 1.
   - Create a semaphore (mutex) to control access to the critical section.

2. Create Philosopher Threads:

   - Create an array of philosopher threads (pthread_t philosophers[NUM_PHILOSOPHERS]).
   - For each philosopher, create a unique ID and pass it to the corresponding thread. Each thread runs the philosopher function.

3. Philosopher Function:

   - Define a function philosopher that takes a philosopher's ID as an argument.
   - Inside the function, use a loop to represent the philosopher's activities for a certain number of meals (MAX_MEALS).
   - Inside the loop:
   - Acquire the mutex semaphore (sem_wait(&mutex)) to enter the critical section.
   - Acquire the left and right forks by waiting on the corresponding semaphores (sem_wait(&forks[left_fork]) and sem_wait(&forks[right_fork])).
   - Release the mutex semaphore to allow other philosophers to enter the critical section (sem_post(&mutex)).
   - Simulate eating by printing a message and sleeping for a random time (sleep(rand() % 3)).
   - Release the forks by posting to the corresponding semaphores (sem_post(&forks[left_fork]) and sem_post(&forks[right_fork])).
   - Simulate thinking by printing a message and sleeping for a random time (sleep(rand() % 3)).

4. Main Function:

   - Initialize semaphores and create philosopher threads as described above.
   - Wait for all philosopher threads to finish using pthread_join.

5.Clean Up Resources:

   - Destroy the semaphores to release system resources (sem_destroy(&forks[i]) and sem_destroy(&mutex)).

PROGRAM

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

#define NUM_PHILOSOPHERS 5
#define MAX_MEALS 1

// Names and philosophies of the philosophers
const char* philosopher_names[NUM_PHILOSOPHERS] = {
    "Plato",
    "Aristotle",
    "Kant",
    "Descartes",
    "Sartre"
};

const char* philosopher_philosophies[NUM_PHILOSOPHERS] = {
    "The highest form of knowledge is self-knowledge.",
    "The roots of education are bitter, but the fruit is sweet.",
    "Act only according to that maxim by which you can at the same time will that it should become a universal law.",
    "I think, therefore I am.",
    "Man is condemned to be free; because once thrown into the world, he is responsible for everything he does."
};

// Define semaphores
sem_t forks[NUM_PHILOSOPHERS];
sem_t mutex;

// Meal counters for each philosopher
int meals_eaten[NUM_PHILOSOPHERS] = {0};
```

```
void *philosopher(void *arg) {
    int philosopher_id = *(int *)arg;
    int left_fork = philosopher_id;
    int right_fork = (philosopher_id + 1) % NUM_PHILOSOPHERS;

    for (int meal_count = 0; meal_count < MAX_MEALS; meal_count++) {
        // Grab forks
        sem_wait(&mutex);
        sem_wait(&forks[left_fork]);
        sem_wait(&forks[right_fork]);
        sem_post(&mutex);

        // Eating
        printf("%s is eating. Meal count: %d\n", philosopher_names[philosopher_id], meal_count + 1);
        sleep(rand() % 3);

        // Release forks
        sem_post(&forks[left_fork]);
        sem_post(&forks[right_fork]);

        // Thinking
        printf("%s is thinking: %s\n", philosopher_names[philosopher_id], philosopher_philosophies[philosopher_id]);
        sleep(rand() % 3);
    }
}

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

    // Initialize semaphores
    for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
        sem_init(&forks[i], 0, 1);
    }
    sem_init(&mutex, 0, 1);
```

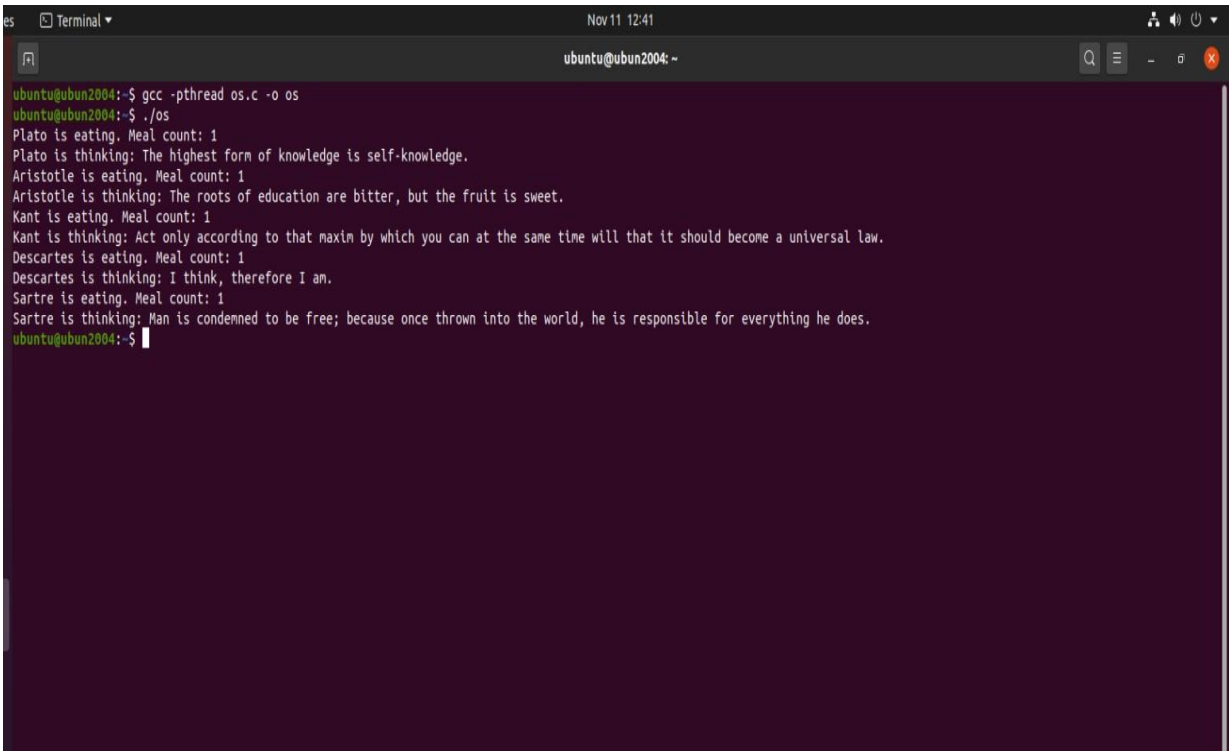
```
    // Create philosopher threads
    for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
        philosopher_ids[i] = i;
        pthread_create(&philosophers[i], NULL, philosopher,
&philosopher_ids[i]);
    }

    // Join philosopher threads
    for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
        pthread_join(philosophers[i], NULL);
    }

    // Destroy semaphores
    for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
        sem_destroy(&forks[i]);
    }
    sem_destroy(&mutex);

    return 0;
}
```

OUTPUT


Terminal
Nov 11 12:41
ubuntu@ubun2004: ~
ubuntu@ubun2004:~$ gcc -pthread os.c -o os
ubuntu@ubun2004:~$ ./os
Plato is eating. Meal count: 1
Plato is thinking: The highest form of knowledge is self-knowledge.
Aristotle is eating. Meal count: 1
Aristotle is thinking: The roots of education are bitter, but the fruit is sweet.
Kant is eating. Meal count: 1
Kant is thinking: Act only according to that maxim by which you can at the same time will that it should become a universal law.
Descartes is eating. Meal count: 1
Descartes is thinking: I think, therefore I am.
Sartre is eating. Meal count: 1
Sartre is thinking: Man is condemned to be free; because once thrown into the world, he is responsible for everything he does.
ubuntu@ubun2004:~$

CONCLUSION

In conclusion, the provided C code offers a solution to the classic dining philosophers problem, a scenario commonly used to illustrate challenges in concurrent programming and synchronization. The algorithm employs pthreads for thread creation and semaphores for synchronization, ensuring that philosophers can acquire and release forks without conflicting with each other.Key elements of the implementation include the use of semaphores to control access to shared resources (forks) and a mutex to manage the critical section where philosophers grab forks. The inclusion of random sleep times adds realism to the simulation, introducing variability in execution and emphasizing potential synchronization issues.