QUESTION

The dining philosophers problem contains five philosophers sitting on a round table can perform only one among two actions – eat and think. For eating, each of them requires two forks, one kept beside each person. Typically, allowing unrestricted access to the forks may result in a deadlock. (a) Write a program to simulate the philosophers using threads, and the forks using global variables. Resolve the deadlock using the following techniques: 1. Strict ordering of resource requests, and 2. Utilization of semaphores to access the resources. (b) Repeat the above system only using semaphores now with a system that also has two sauce bowls. The user would require access to one of the two sauce bowls to eat, and can access any one of them at any point of time.

ANSWER

STEP 0

Q:-The dining philosophers problem contains five philosophers sitting on a round table can perform only one among two actions – eat and think. For eating, each of them requires two forks, one kept beside each person. Typically, allowing unrestricted access to the forks may result in a deadlock. (a) Write a program to simulate the philosophers using threads, and the forks using global variables. Resolve the deadlock using the following techniques: 1. Strict ordering of resource requests, and 2. Utilization of semaphores to access the resources.

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

sem_t forks[5];
sem_t sauceBowls[2];

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

        while(1) {
                // Strict ordering of resource requests
                if (id % 2 == 0) {
                        //Philosopher with even id first requests for two forks
                        sem_wait(&forks[id]);
                        sem_wait(&forks[(id + 1) % 5]);
                }
                else {
                        //Philosopher with odd id first requests for two forks
                        sem_wait(&forks[(id + 1) % 5]);
                        sem_wait(&forks[id]);
                }

                //Utilization of semaphores to access the resources
                sem_wait(&sauceBowls[id%2]);

                printf("Philosopher %d is eating\n", id);
                //Do the required eating operations

                //Release the resources
                sem_post(&sauceBowls[id%2]);
                sem_post(&forks[id]);
                sem_post(&forks[(id + 1) % 5]);
        }
}

int main() {
        int i;
        pthread_t phil[5];
        int args[5];

        //Initializing the semaphores
        for (i = 0; i < 5; i++) {
                sem_init(&forks[i], 0, 1);
        }
        for (i = 0; i < 2; i++) {
                sem_init(&sauceBowls[i], 0, 1);
        }

        //Creating the threads
        for (i = 0; i < 5; i++) {
                args[i] = i;
                pthread_create(&phil[i], NULL, philosopher, &args[i]);
        }

        //Waiting for the threads to complete
        for (i = 0; i < 5; i++) {
                pthread_join(phil[i], NULL);
        }
        return 0;
}
```

Please refer to solution in this step.
STEP 1

(b) Repeat the above system only using semaphores now with a system that also has two sauce bowls. The user would require access to one of the two sauce bowls to eat, and can access any one of them at any point of time.

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

sem_t forks[5];
sem_t sauceBowls[2];

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

        while(1) {
                //Utilization of semaphores to access the resources
                sem_wait(&forks[id]);
                sem_wait(&forks[(id + 1) % 5]);
                sem_wait(&sauceBowls[id%2]);

                printf("Philosopher %d is eating\n", id);
                //Do the required eating operations

                //Release the resources
                sem_post(&sauceBowls[id%2]);
                sem_post(&forks[id]);
                sem_post(&forks[(id + 1) % 5]);
        }
}

int main() {
        int i;
        pthread_t phil[5];
        int args[5];

        //Initializing the semaphores
        for (i = 0; i < 5; i++) {
                sem_init(&forks[i], 0, 1);
        }
        for (i = 0; i < 2; i++) {
                sem_init(&sauceBowls[i], 0, 1);
        }

        //Creating the threads
        for (i = 0; i < 5; i++) {
                args[i] = i;
                pthread_create(&phil[i], NULL, philosopher, &args[i]);
        }

        //Waiting for the threads to complete
        for (i = 0; i < 5; i++) {
                pthread_join(phil[i], NULL);
        }
        return 0;
}
```

Please refer to solution in this step.
STEP 0

The dining philosopher's problem is the classical problem of synchronization which says that Five philosophers are sitting around a circular table and their job is to think and eat alternatively. A bowl of noodles is placed at the center of the table along with five chopsticks for each of the philosophers.