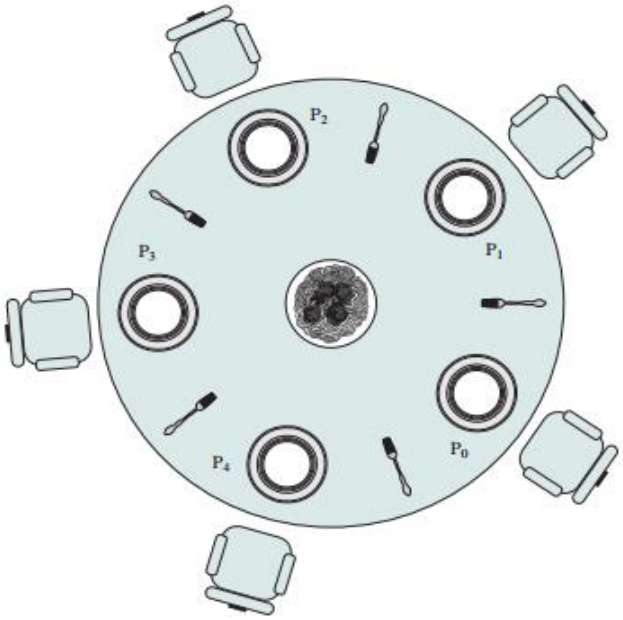
CEN 354 OPERATING SYSTEM COURSE ASSIGNMENT #3

- PROBLEM DESCRIPTION

The dining philosophers problem is an example problem often used in concurrent algorithm design to illustrate synchronization issues and techniques for resolving them. It's about philosophers who are ready to have their meals while thinking but not both at a same time either thinking or eating.

There are 5 philosophers sharing a circular table and they eat and think alternatively. There is a bowl of rice for each of the philosophers and 5 forks. A philosopher needs both their right and a left fork to eat. A hungry philosopher may only eat if there are both forks available. Otherwise, a philosopher puts down their fork and begin thinking again.

Problems That May Occur: In this problem, we encounter certain problems that can be deadlock. If we examine the problems that may occur separately;



1) With each philosopher taking fork at the same time that in the same direction (left or right) according to them, none of the philosophers at the table can eat. In this case, every philosopher on the table has 1 fork in his hand and none of them can carry out eating. With this event taking place, 4 possible deadlock (Mutex, Hold and Wait, No-Preeemption, Circular Wait) conditions may have occurred. And the system may eventually drop into deadlock.

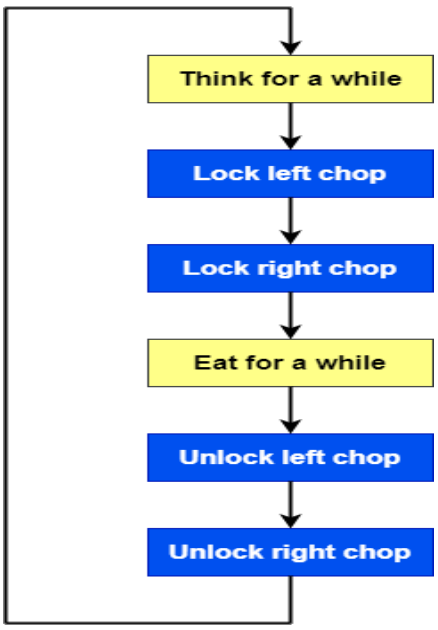
2) Considering the situation where 2 of the philosophers can eat; If the eating philosophers do not transfer forks to the philosopher waiting to eat, after the eating process is finished or after a certain period of eating, these philosophers may experience starvation and cannot progress. At the end of this situation, the event can also take the deadlock dimension.

3) Another problem we may encounter in this problem is the synchronization problem. In order for a philosopher to eat, he must have 2 forks simultaneously.While the philosopher gets a fork, the other must hold it synchronously. If he fails to do this, he will not be able to eat anyway. For this reason, the process of handling forks synchronously can be accomplished by an indivisible (atomic) structure or by leaving the fork in the hands of the neighboring philosopher on the table for use.

- ABSTRACTION FOR SOLUTION

Each philosopher (thread) proceeds through the following cycle.

- THINKING (out of critical section) may remain indefinitely.
- HUNGRY (requesting to enter critical section)
- EATING (executing the critical section) has to finish within finite time.



- SOLUTION PROPERTIES

- SAFETY: No two neighbor eat at the same time.
- LIVENESS: Each hungry philosopher eventually eats.

- MY IMPLEMENTATION

  - LIBRARIES (HEADER FILES) THAT INCLUDED :

    <pthread.h> : The POSIX thread libraries are a standards based thread API for C/C++. It allows one to spawn a new concurrent process flow. It is most effective on multi-processor or multi-core systems where the process flow can be scheduled to run on another processor thus gaining speed through parallel or distributed processing.
    I included this library to use POSIX thread functions and pthread library also includes support for the mutex and conditional variables required for synchronization operations.

    <stdio.h> :Standard input output file contains functions related to the input/output operations like “printf”,”scanf” ,”getc”,”getchar” etc. It defines three variable types, several macros, and various functions for performing input and output.

    <stdlib.h> :It is the header of the general purpose standard library of C programming language which includes functions involving memory allocation, process control, conversions and others.I used for random fuctions.

    <unistd.h> : In the C and C++ programming languages, unistd.h is the name of the header file that provides access to the POSIX operating system API. It is defined by the POSIX.1 standard, the base of the Single Unix Specification.I used for sleep function.

  - METHODS (FUNCTIONS) THAT I USED :

    void *philosopher(void *prm) :Actions of philosophers
    void test(int i) : Check for other :philosophers
    void pickup_forks(int philo_id): Waiting in order to eat
    void putdown_forks(int philo_id): Release the eating order
    extern inline int check_left(int philo_id): Returns left neighbor's id
    extern inline int check_right(int philo_id):Returns right neighbor's id

- OPERATION :

First, I have identified that the initial states of philosophers as THINKING by for loop in the main function. I also stored the ids of the philosophers as an array in the same for loop. The purpose of holding them as an array was to prevent mutual exclusions.And i initialize the condition variables with using pthread_cond_init function.

After initialize the mutex locks with using pthread_mutex_init function, different threads are created for each philosopher and also these are started with using pthread_create method.In this method ,it calls the void *philosopher(void *prm) function and sends the philosopher's id to this function by reference.

In void *philosopher(void *prm) function , it taken thread's id (incoming id) as a parameter.Initialize the random function with using srandom to generate the numbers more efficently.And the loop where the actual actions of the philosophers begin is the while loop.

We have already pointed philosopher's initial states as THINKING in main function. When this infinite loop starts, it 'prints that philosopher is thinking' on the screen. The random time required for the philosopher to think is provided by the sleep function.

So ,for thephilosopher can eat after thinking for a while void pickup_forks(int philo_id) function must be called.In this function,firstly locks that function with using pthread_mutex_lock method to prevent function is accessed by multiple threads at the same time.Since the philosopher wants to eat, we are updating his state as HUNGRY.

It is necessary to check the neighbors before philospher starts eating so void test(int i) function is called. The philosopher's id is sent to the function as a parameter.In this function,it checks whether the current philosopher is HUNGRY. The states of the neighbors that left and right of the philosopher is also checked.Are they EATING or NOT ? If not, the philosopher can eating.So updated the philosopher's state as EATING.

Thus,we broke the while loop in void pickup_forks(int philo_id) function.It prints the 'philosopher is eating' on the screen.After that released the lock of function with using pthread_mutex_unlock method.Again philosopher sleeps during a random time for EATING.

To finish eating void putdown_forks(int philo_id) fuction is called.Firstly,again we locked the function .And updated the philopher's state as THINKING. The philosopher controls his neighbors when he finishes eating and goes into thinking.If they are HUNGRY, he gives the eating order to them.I provide this operation by using inline int check_left(int philo_id) and int check_right(int philo_id) functions. After transferring the philosopher's order this while loop makes all processes infinitely.

MERVE PARLAK

ID:2015556055