OPERATING SYSTEMS

Dining Philosophers Problem

Nitin V Pujari
Faculty, Computer Science
Dean - IQAC, PES University

OPERATING SYSTEMS

Course Syllabus - Unit 2

12 Hours

Unit 2: Threads & Concurrency

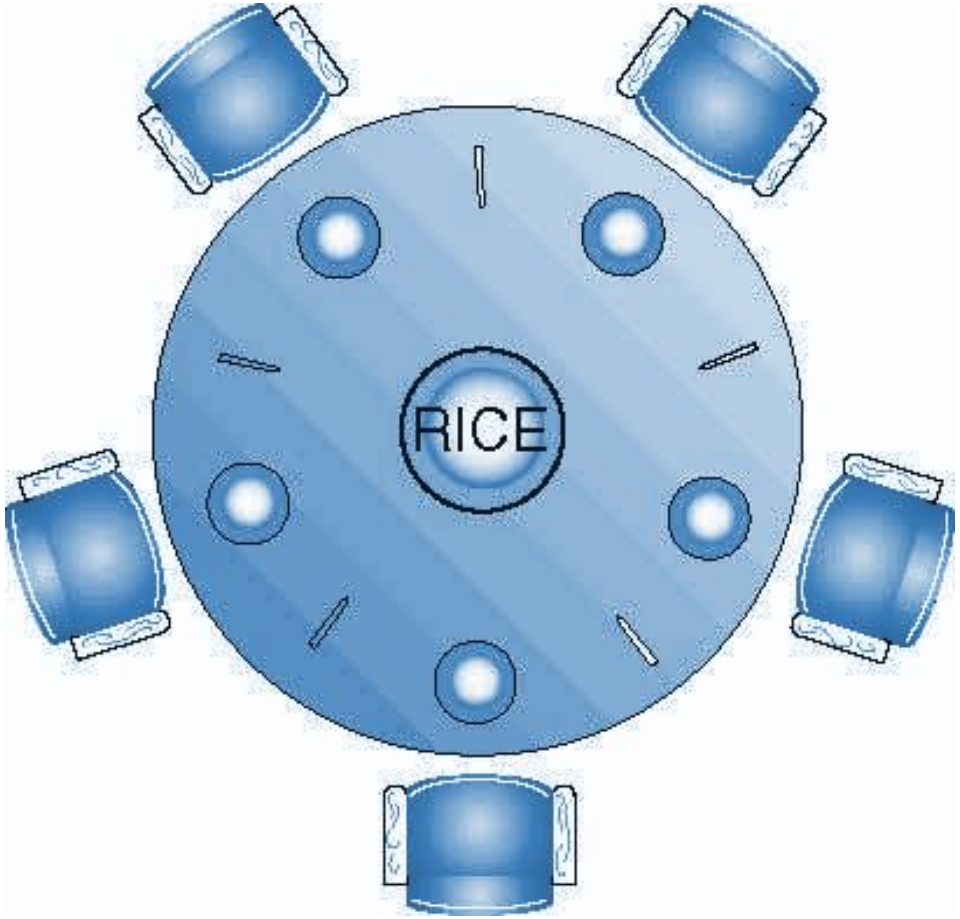
Introduction to Threads, types of threads, Multicore Programming, Multithreading Models, Thread creation, Thread Scheduling, PThreads and Windows Threads, Mutual Exclusion and Synchronization: software approaches, principles of concurrency, hardware support, Mutex Locks, Semaphores. Classic problems of Synchronization: Bounded-Buffer Problem, Readers -Writers problem, Dining Philosophers Problem concepts. Synchronization Examples - Synchronisation mechanisms provided by Linux/Windows/Pthreads. Deadlocks: principles of deadlock, tools for detection and Prevention.

OPERATING SYSTEMS

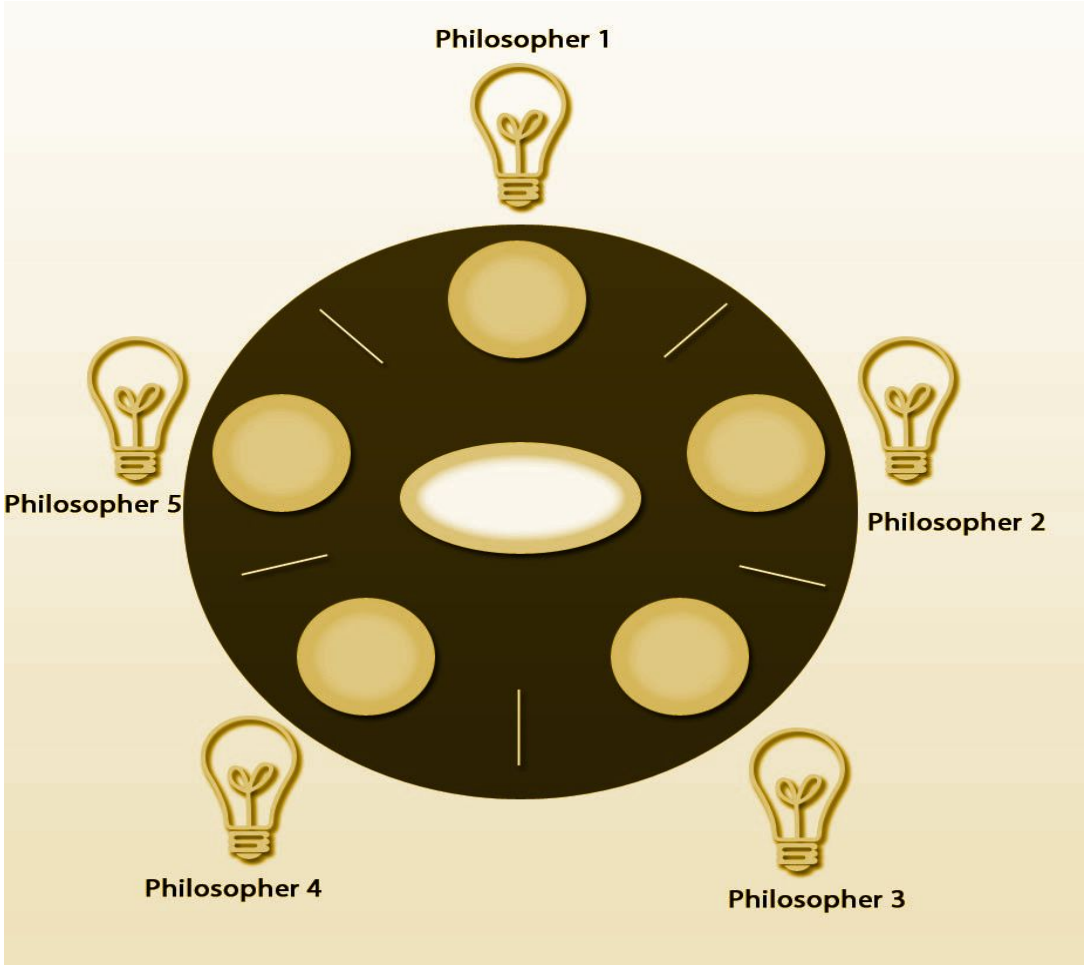
Course Outline

13	Introduction to Threads, types of threads, Multicore Programming, Multithreading Models	4.1 - 4.3	42.8
14	Thread creation, Thread Scheduling	5.4	
15	Pthreads and Windows Threads	4.4	
16	Mutual Exclusion and Synchronization: software approaches,	6.1-6.2	
17	principles of concurrency, hardware support	6.3-6.4	
18	Mutex Locks, Semaphores	6.5, 6.6	
19	Classic problems of Synchronization: Bounded-Buffer Problem, Readers-Writers problem	6.7-6.8	
20	Dining-Philosophers Problem	6.8	
21	Synchronization Examples: Synchronisation mechanisms provided by Linux/Windows/Pthreads.	6.9	
22	Deadlocks: principles of deadlock, Deadlock Characterization	7.1-7.3	
23	Deadlock Prevention, Deadlock example	7.4-7.5	
24	Deadlock Detection, Algorithm	7.6	

- The Dining Philosopher Problem

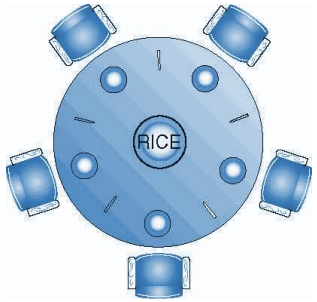


- The Dining Philosopher Problem



Classical Problems of Synchronization: Dining philosophers Problem

- The dining philosophers problem is another classic synchronization problem which is used to evaluate situations where there is a need of allocating multiple resources to multiple processes.
- Philosophers spend their lives alternating thinking and eating
- Don’t interact with their neighbors, occasionally try to pick up 2 Forks (Chopsticks) (one at a time) to eat from bowl
- Need both to eat, then release both when done

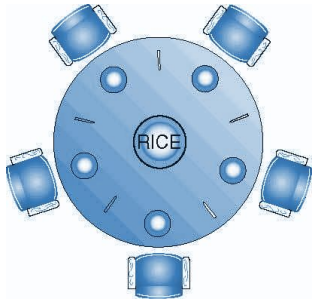


Classical Problems of Synchronization: Dining philosophers Problem

- In the case of 5 philosophers
- Shared data
  - Bowl of rice (data set)
  - Semaphore Fork [5] or Chopstick[5] initialized to 1

The structure of Philosopher i:

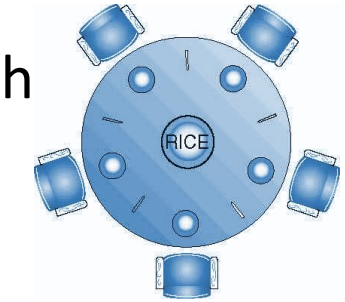
```
do {
        wait (chopstick[i] );
        wait (chopStick[ (i + 1) % 5] );
        // eat

        signal (chopstick[i] );
        signal (chopstick[ (i + 1) % 5] );
        //Think
} while (TRUE);
```



Classical Problems of Synchronization: Dining philosophers Problem

Deadlock handling

- Allow at most 4 philosophers to be sitting simultaneously at the table.
- Allow a philosopher to pick up the forks only if both are available (picking must be done in a critical section.
- Use an asymmetric solution => an odd-numbered philosopher picks up first the left chopstick and then the right chopstick.
- Even-numbered philosopher picks up first the righ chopstick and then the left chopstick.



Incorrect use of Semaphore Operation

- Incorrect use of semaphore operations:
  - signal (mutex) .... wait (mutex)
  - wait (mutex) ... wait (mutex)
  - Omitting of wait (mutex) or signal (mutex) (or both)
- Deadlock and starvation are possible.

THANK YOU

Nitin V Pujari
Faculty, Computer Science
Dean - IQAC, PES University

nitin.pujari@pes.edu

For Course Deliverables by the Anchor Faculty click on www.pesuacademy.com