Department of Computer Science & Engineering
Guru Tegh Bahadur Institute of Technology
Guru Gobind Singh Indraprastha University
Kashmere Gate,New Delhi

GAGANDEEP SINGH

PRESENTS

The Dining Philosopher Problem

(A JAVA TECHNOLOGY PRODUCT)

Submitted By :

Gagandeep Singh

Enrollment No : 00413202709

CSE - 5th Semester

Year 2011

Theme of Project

Process Synchronization And Deadlock

Process Synchronization

- **Process synchronization** refers to the idea that multiple processes are to join up or handshake at a certain point, so as to reach an agreement or commit to a certain sequence of action.

- **Need for Process Synchronization**

  Process synchronization or serialization, strictly defined, is the application of particular mechanisms to ensure that two concurrently-executing threads or processes do not execute specific portions of a program at the same time. If one process has begun to execute a serialized portion of the program, any other process trying to execute this portion must wait until the first process finishes.

Process Synchronization

- **Application**

  Synchronization is used to control access to state both in small-scale multiprocessing systems -- in multithreaded and multiprocessor computers -- and in distributed computers consisting of thousands of units -- in banking and database systems, in web servers, and so on.

- **The Critical-Section Problem**

  Each process has a segment of code, called **critical section**, in which the process may be changing common variables, updating a table, writing a file and so on.

  So, when one process is executing in its critical section, no other process is to be allowed to execute in its critical section. This is known as critical-section problem.

Process Synchronization

Thus, the execution of critical sections by the process is **mutually exclusive** in time.

The critical-section problem is to design a protocol that the process can use to cooperate. Each process must request permission to enter its critical section.

- **Race Condition**

  A situation where several processes access and manipulate the same data concurrently, and the outcome of the execution depends on the particular order in which the access takes place.

Deadlock

- **Deadlock** is a situation where two or more processes are waiting indefinitely for an event that can be caused by only one of the waiting process.

- Deadlock is a common problem in multiprocessing where many processes share a specific type of mutually exclusive resource known as a *software lock* or *soft lock.*

- It is often seen in a paradox like the "chicken or the egg". Perhaps the best illustration of a deadlock can be drawn from a law passed by the Kansas Legislature, it said –

  "When two trains approach each other at a crossing, both shall come to a full stop and neither shall start up again until the other has gone."

Deadlock

Necessary Conditions for Deadlock

A deadlock situation can arise if the following four conditions hold simultaneously in a system :

- **Mutual exclusion :** Atleast one resource must be held in a non-sharable mode i.e. only one process at a time can use the resource while other processes wait for the resource to be released.

- **Hold and wait :** There must exist a process that is holding at least one resource and is waiting to acquire additional resource that are currently being held by other process.

- **No preemption :** Resources cannot be preempted i.e. a resource can be released only voluntarily by the process holding it, after that process has completed its task.

Deadlock

- **Circular wait :** There must exist a set $\{P_0 , P_1 , \ldots P_n\}$ of waiting processes such that P_0 is waiting for a resource that is held by P_1 , P_1 is waiting for a resource held by P2 …, P_n is waiting for a resource that is held by P_0 .

Methods for handling Deadlock

Principally, there are three different methods for dealing with deadlock problem :

- **Deadlock prevention :** is a set of methods for ensuring that at least one of the necessary conditions cannot hold.

- We can allow the system to enter a deadlock state and then recover i.e. **Deadlock detection** and **Deadlock Recovery**

Deadlock

- We can ignore the problem all together, and pretend that deadlock never occur in the system.
  In such case, the operating system need to be given in advance additional information concerning which resources a process will request and use during its lifetime – **Deadlock avoidance.**

Safe State

A sate is safe if the system can allocate resources to each process in some order and still avoid a deadlock.

More formally, a system is in a safe state only if there exist a safe sequence i.e. a sequence of process such that for each process , the resources that it request can be satisfied by the currently available resources plus the resources held by all other processes.

Welcome To

The Dining Philosopher Problem

Introduction

- The dining philosopher problem is considered a classic synchronization problem as it is an example of large class of concurrency-control problem.
- It is a simple representation of the need to allocate several resources among several processes in a deadlock-free and starvation-free manner.
- In 1965, Edsger Dijkstra set an examination question on a synchronization problem where five computers competed for access to five shared tape drive peripherals.
- Soon afterwards the problem was retold by Tony Hoare as the dining philosophers problem.

Problem Statement

- Consider five philosophers who spend their lives thinking and eating. The philosophers share a circular table surrounded by five chairs, each belonging to one philosopher. In the center of the table is a bowl of rice, and the table is laid with five chopsticks.
- When a philosopher thinks, he does not interact with his colleagues.
- From time to time, a philosopher gets hungry and tries to pick up the two chopsticks that are closest to him (the chopsticks that are between him and his left and right neighbors). A philosopher may pick up only one chopstick at a time.
- When a hungry philosopher has both his chopsticks at the same time, he eats without releasing his chopsticks. When he is finished eating, he puts down both of his chopsticks and start thinking again.

Issue

Suppose that all five philosophers become hungry simultaneously and grabs his left chopstick. When each philosopher tries to grab his right chopstick, he will be delayed forever.

Fig 1 – Sitting arrangement

System Model

- The Dining Philosopher describes a model of a system consisting of five sequential processes/threads represented by five philosophers , all running asynchronously.
- The Thread share a common data – five chopsticks . Each philosopher (Thread) compete to acquire two chopsticks (resources) closest to him.
- Table Arrangement :
  - All five philosophers sitting around a round table are uniquely labeled with an integer value ranging from 0 to 5 as shown above.
  - Each chopstick lies between two philosophers and also has its own unique label number.
  - Philosopher 1 has access to chopstick 1 and 2 only, Philosopher 2 has access to chopstick 2 and 3, Philosopher 3 to chopstick 3 and 4, Philosopher 4 to chopstick 4 and 5 and lastly Philosopher 5 has access to chopstick 5 and 1 only.

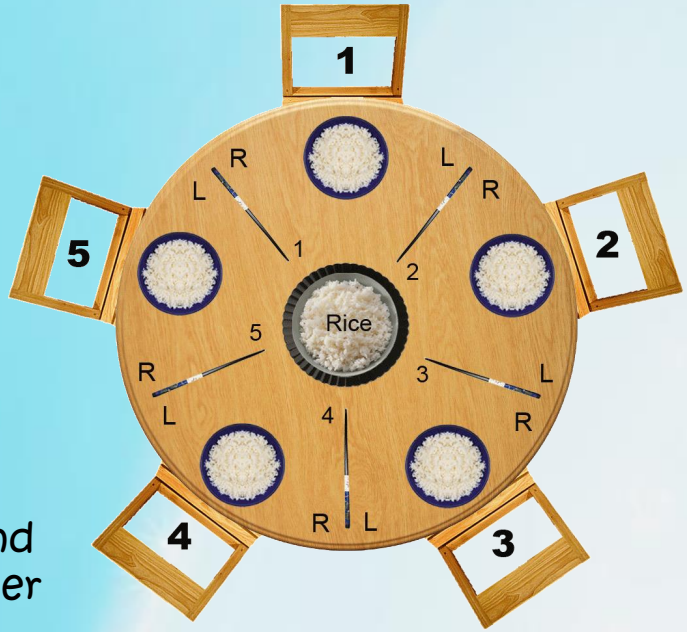

Fig 2 – Table arrangement

System Model

- States of a Philosopher :
  - 'Thinking' State : This is the initial state of a philosopher in which he simply thinks without any desire to eat or to perform any kind of action.
  - 'Waiting' State : This is the state where the philosopher waits to acquire chopsticks so that he can eat. A philosopher may be in waiting state if he wishes to eat and has either one or no chopsticks.
  - 'Eating' State : This state signifies that the philosopher has bath the chopsticks and is eating rice (executing).
- Rules for Philosophers :
  - All philosophers participating are initially at 'thinking' state where they think without desire to eat rice or own a chopstick.
  - Any philosopher who wishes to eat must first put into a 'waiting' state before he can perform any action like picking up chopsticks.
  - A philosopher may pick one chopstick at a time.

System Model

- A philosopher can change its state from waiting to eating (execute its body) if and only if he has two chopsticks at the same time. Else the philosopher will continue its state of 'waiting'.
- Once in 'eating' state, a philosopher eats for 5 seconds and then releases both the chopsticks (release resources) so that they are made available for other philosophers.
- Moreover, when a philosopher has completed eating once, he exits the dining hall which indicates that the thread has been removed from scheduling queue thereby freeing-up the memory. This also makes sure that the same philosopher does not get a chance to eat again and again thereby preventing **starvation.**

- Critical Section : 'Act of picking up a chopstick by philosopher is the critical section'. That is, the body of the code where each philosopher as a thread tries to pick up a chopstick is the critical section. This is because only one philosopher can pick a chopstick at a time. So, if two philosopher wishes to acquire a common chopstick then only that philosopher is awarded with the chopstick who has already entered his critical section.

System Model

- States of a System :
  - 'Safe' State : It is the desired state of the system that signifies that all philosophers competing with each other for chopsticks, respectfully get their chance to eat.
  - 'Deadlock' State : This state indicates an unsafe system, where the system goes into an endless loop since every philosopher with one chopstick each, is waiting for other one to release his chopstick which brings the system into a halt.

- Rules for running the system :
  - The system runs as an infinite loop until all philosophers participating get their chance to eat.
  - Each loop performs three phases.
  - At the start of the loop, all philosophers try to pick one chopstick available on the table.

System Model

- Next, all philosophers are again given a chance to pick a chopstick. Those who already have a chopstick seek for another one while those who are waiting empty handed seek for their first chopstick.
- Lastly, all those philosophers having two chopsticks eat for five seconds without releasing their chopsticks while others still remain in waiting state.
- The loop continues to perform one of the above three phase for respective philosophers till all of them leaves the dining hall.
- Once everybody has finished eating, the control is forcefully pulled out of the loop.

Running the Program

Deadlock Demonstration

- Initially all philosopher are in 'thinking' state where they are simply thinking.
- All five philosophers are fired up asynchronously where each philosopher are first set to 'waiting' state before they are allowed to pick chopsticks, denoted by 'bowl of rice" in their thinking bubble.
- Each philosopher then try to pick their left chopstick one by one. Philosopher still remain in waiting state since they require another chopstick.
- Next, philosopher again try to acquire their second chopstick.
- Since, every philosopher owns one chopstick each and there are no chopsticks on table, each philosopher now waits for other philosopher to give his chopstick. DEADLOCK has occurred.

Remedy I

Allow at most four philosopher to be sitting simultaneously at the table.

- In this solution to deadlock, one of the philosopher is forcefully pulled off from table i.e. the thread is not allowed to enter scheduling queue.
- The system, in its initial state (philosophers in 'thinking' state) is initiated where all four philosopher changes their state to 'waiting'.
- Each philosopher gets a chance to pick their left chopstick during first phase of the loop.
- During second phase of the loop, one of the philosopher adjacent to empty seat is always left with second chopstick available.
- Thus, this philosopher grabs the chopstick, changes its state to 'eating' , eats for five seconds, releases both the chopstick and exits the hall.
- As soon as first philosopher finishes eating and releases its chopstick, the next adjacent philosopher now gets his second chopstick in next loop iteration. Thus, he eats , releases chopstick and exits.
- This procedure is followed like a chain reaction as the loop iterates till all four philosopher gets their chance to eat. System is now STABLE.

Remedy 2

Allow a philosopher to pick up his chopsticks only if both chopsticks are available.

- This solution defines a protocol, in which a philosopher is allowed to pick up his chopsticks only if both are available. Even if one chopstick is available, the philosopher is not allowed to pick it.
- In such a way, during first loop run, two of the five philosophers are able to pick their respective chopstick with one chopstick still left on table. Both of these philosopher eats for five seconds simultaneously , releases their chopsticks and exits the hall.
- In next iteration of the loop, two of the three philosopher still waiting gets their chopstick set and eat for five second simultaneously.
- Next iteration causes the last philosopher waiting to grab his chopsticks and eat for five seconds
- The system is STABLE. No deadlock encountered.

Remedy 3

Asymmetric Solution

- The asymmetric solution defines another protocol, in which an odd philosopher picks up his left chopstick and then his right chopstick, whereas an even philosopher picks up his right chopstick and then his left chopstick.

- In first loop , first phase, three out of five philosopher which happens to be the odd numbered philosopher pick up their left chopstick first. In next phase, two of the three philosopher mentioned are able to pick their right available chopstick. Thus, they eat for five seconds and releases their chopsticks.

- So, as the loop iterates, each philosopher gets their chance to get in a chain reaction fashion.

- The system is STABLE.

About

Programming Language : JAVA
Softwares Used : NetBeans IDE 6.9.1
Java SE Development Kit 6

Java Concepts Used

Java Basics
MultiThreading
Java Swings

This Project is part of summer training
in Java Programming Language
20 July - 8 August 2011

gagandeep.taurus91@gmail.com | 9717568636

Presented By :

Gagandeep Singh

CSE/IT, Guru Tegh Bahadur Institute of Technology,
G-8 Area, Rajouri Garden, New Delhi-110064
00413202708
CSE-1 , Semester-5
gagandeep.taurus91@gmail.com
Year 2011