

Chapter 7: Process Synchronization

- Background
- The Critical-Section Problem
- Semaphores
- A Classical Problem of Synchronization
 - The Dining-Philosophers Problem

Background

- Concurrent access to shared data may result in data inconsistency.
- Maintaining data consistency requires mechanisms to ensure the orderly execution of cooperating processes.

The Critical-Section Problem

- n processes all competing to use some shared data.
- Each process has a code segment, called *critical section*, in which the shared data is accessed.
- Problem – ensure that when one process is executing in its critical section, no other process is allowed to execute in its critical section.
- Structure of process P_i

repeat

entry section

critical section

exit section

remainder section

until false;

Solution to Critical-Section Problem

1. **Mutual Exclusion.** If process P_i is executing in its critical section, then no other processes can be executing in their critical sections.
2. **Progress.** If no process is executing in its critical section and there exist some processes that wish to enter their critical section, then the selection of the processes that will enter the critical section next cannot be postponed indefinitely.
3. **Bounded Waiting.** A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

Semaphores

- Semaphore is a synchronization tool which can be used to deal with the critical-section problem.
- A semaphore is a protected variable whose value can be accessed and altered only by the operations P (wait) and V (signal) and an initialization operation we shall call semaphore-initialize.

Semaphores (Cont.)

- The classical definitions of wait and signal are:
 - The P operation (wait) on semaphore S, written P(S), operates as follows:
if $S > 0$ then $S := S - 1$;
else (wait on S)
 - The V operation (signal) on semaphore S, written V(S), operates as follows:
if (one or more processes are waiting on S)
then (let one of these processes proceed)
else $S := S + 1$

Two Types of Semaphores

- *Counting* semaphore – integer value can range over an unrestricted domain.
- *Binary* semaphore – integer value can range only between 0 and 1; can be simpler to implement.
- Can implement a counting semaphore S as a binary semaphore.
- Starvation – indefinite blocking. A process may never be removed from the semaphore queue in which it is suspended.
- The implementation of P and V guarantees that processes will not suffer indefinite postponement.

Example: Critical Section of n Processes

- The n processes share a semaphore initialized to 1.
- Shared variables
 - **var** *mutex* : semaphore
 - initially *mutex* = 1 /* mutual exclusion
- Each process P_i is organized as follows:

```
repeat
    wait(mutex);
    critical section
    signal(mutex);
    remainder section
until false;
```
- This is a Mutual-Exclusion implementation with semaphores.

Semaphore as General Synchronization Tool

- Consider two concurrently running processes:
 - P1 with a statement S1 and P2 with a statement S2.
- Suppose that we require that S2 be executed only after S1 has completed.
 - This can be done by letting P1 and P2 share a common semaphore *synch*, initialized to 0, and by inserting the statements in process P1:

S1;

signal (synch);

and the statements in process P2:

wait (synch);

S2;

- Because *synch* is initialized to 0, P2 will execute S2 only after P1 has invoked *signal (synch)*, which is after S1.

Example: Deadlock and Starvation

- Deadlock – two or more processes are waiting indefinitely for an event that can be caused by only one of the waiting processes.
- Let S and Q be two semaphores initialized to 1

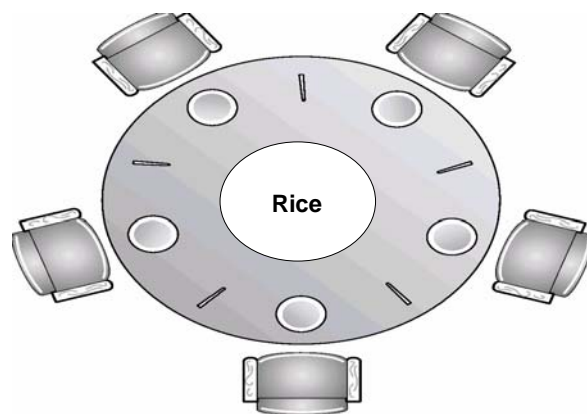
	P_0	P_1
step1:	<i>wait(S);</i>	<i>wait(Q);</i>
step2:	<i>wait(Q);</i>	<i>wait(S);</i>
	⋮	⋮
step3:	<i>signal(S);</i>	<i>signal(Q);</i>
step4:	<i>signal(Q)</i>	<i>signal(S);</i>

Example: Deadlock and Starvation (Cont.)

- Step 1: Suppose that P0 executes wait(S) and then P1 executes wait(Q).
- Step 2: When P0 executes wait(Q), it must wait until P1 executes signal(Q).
- Step 2: Also, when P1 executes wait(S), it must wait until P0 executes signal(S).
- Step 3 & 4: Since these signal operations cannot be executed, P0 and P1 are deadlocked.
- A set of processes is in a deadlock state when every process in the set is waiting for an event that can be caused only by another process in the set.

A Classical Problem of Synchronization

- Dining-Philosophers Problem:



Dining-Philosophers Problem

- Table, a bowl of rice in center, five chairs, and five chopsticks, also in each chair there is a philosopher.
- A philosopher may pick up only one chopstick at a time.
- When a hungry philosopher has both her chopsticks at the same time, she eats without releasing her chopsticks.
- When she is finished eating, she puts down both of her chopsticks and starts thinking.
- From time to time, a philosopher gets hungry and tries to pick up the two chopsticks that are closest to her (chopsticks between her left and right neighbors).

Dining-Philosophers Problem (Cont.)

- Solution:
 - Present each chopstick by a semaphore.
 - A philosopher tries to grab the chopstick by executing a wait operation on that semaphore.
 - She releases her chopsticks by executing the signal operation on the appropriate semaphores.
- This solution guarantees that no two neighbors are eating simultaneously, but it could cause a deadlock.
- Suppose all five philosophers become hungry simultaneously, and each grabs her left chopstick. When each philosopher tries to grab her right chopstick, she will cause the possibility that the philosophers will starve to death.

Dining-Philosophers Problem (Cont.)

- We present a solution to the dining-philosophers problem that ensure freedom from deadlock.
 - Allow at most 4 philosophers to be sitting simultaneously at the table.
 - Allow a philosopher to pick up her chopstick only if both chopsticks are available.
 - An odd philosopher picks up first her left chopstick and then her right chopstick, whereas, an even philosopher picks up her right chopstick first and then her left chopstick.
- Note: A deadlock-free solution does not necessarily eliminate the possibility of starvation.