The Dining-Philosophers Problem

The Dining-Philosophers Problem

- 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 centre of the table is a bowl of rice, and the table is laid with five single chopsticks .

The situation of the dining philosophers.

The Dining-Philosophers Problem

- When a philosopher thinks, she does not interact with her colleagues.
- From time to time, a philosopher gets hungry and tries to pick up the two chopsticks that are closest to her (the chopsticks that are between her and her left and right neighbors).
- A philosopher may pick up only one chopstick at a time.
- Obviously, she cannot pick up a chopstick that is already in the hand of a neighbour.
- When a hungry philosopher has both her chopsticks at the same time, she eats without releasing the chopsticks.
- When she is finished eating, she puts down both chopsticks and starts thinking again.

The Dining-Philosophers Problem

- The dining-philosophers problem is considered a classic synchronization problem.
- It is a simple representation of the need to allocate several resources among several processes in a deadlock-free and starvation-free manner.
- One simple solution is to represent each chopstick with a semaphore.
- A philosopher tries to grab a chopstick by executing a wait() operation on that semaphore. She releases her chopsticks by executing the signal() operation on the appropriate semaphores.

The Dining-Philosophers Problem

- Thus, the shared data are

semaphore chopstick[5];

where all the elements of chopstick are initialized to 1

```
do {
   wait(chopstick[i]);
   wait(chopstick[(i+1) % 5]);
      . . .
   /* eat for awhile */
      . . .
   signal(chopstick[i]);
   signal(chopstick[(i+1) % 5]);
      . . .
   /* think for awhile */
      . . .
} while (true);
```

The structure of philosopher *i*.

The Dining-Philosophers Problem

Deadlock

- Suppose that all five philosophers become hungry at the same time and each grabs her left chopstick.
- All the elements of chopstick will now be equal to 0.
- When each philosopher tries to grab her right chopstick, she will be delayed forever.

<u>Remedies to the deadlock problem :</u>

- Allow at most four philosophers to be sitting simultaneously at the table.
- Allow a philosopher to pick up her chopsticks only if both chopsticks are available.
- Use an asymmetric solution—that is, an odd-numbered philosopher picks up first her left chopstick and then her right chopstick, whereas an even numbered philosopher picks up her right chopstick and then her left chopstick.

Readers–Writers Problem

Readers–Writers Problem

- Suppose that a database is to be shared among several concurrent processes.

Readers

- Those processes want only to read the database.

Writers

- Those processes want to update (that is, to read and write) the database.
- *If two readers access the shared data* simultaneously, no adverse effects will result.
- If a writer and some other process (either a reader or a writer) access the database simultaneously, adverse effects will result.

Readers–Writers Problem

- We require that the writers have exclusive access to the shared database while writing to the database.
- This synchronization problem is referred to as the **readers–writers problem.**

<u>The readers–writers problem has several variations,</u>

First readers–writers problem

- Requires that no reader be kept waiting unless a writer has already obtained permission to use the shared object.
- In other words, no reader should wait for other readers to finish simply because a writer is waiting.

***Second readers*–writers problem**

- Requires that, once a writer is ready, that writer perform its write as soon as possible.
- In other words, if a writer is waiting to access the object, no new readers may start reading.

- A solution to either problem may result in starvation. In the first case, writers may starve; in the second case, readers may starve.

Solution to the First Readers Writers Problem

- Solution to the first readers–writers problem, the reader processes share the following data structures:

  semaphore rw mutex = 1;

  semaphore mutex = 1;

  int read count = 0;
- The semaphores mutex and rw mutex are initialized to 1; read count is initialized to 0.
- The semaphore rw mutex is common to both reader and writer processes.
- The mutex semaphore is used to ensure mutual exclusion when the variable read count is updated.
- The read count variable keeps track of how many processes are currently reading the object.

Solution to the First Readers Writers Problem

- The semaphore rw mutex functions as a mutual exclusion semaphore for the writers.
- It is also used by the first or last reader that enters or exits the critical section.
- It is not used by readers who enter or exit while other readers are in their critical sections

```
do {
   wait(rw_mutex);
      . . .
   /* writing is performed */
      . . .
   signal(rw_mutex);
} while (true);
```

The structure of a writer process.

Solution to the First Readers Writers Problem

```
do {
   wait(mutex);
   read_count++;
   if (read_count == 1)
      wait(rw_mutex);
   signal(mutex);
      . . .
   /* reading is performed */
      . . .
   wait(mutex);
   read_count--;
   if (read_count == 0)
      signal(rw_mutex);
   signal(mutex);
} while (true);
```

The structure of a reader process.

Solution to the First Readers Writers Problem

- If a writer is in the critical section and *n readers are waiting, then one reader is queued on rw mutex, and n − 1 readers are queued on mutex.*
- *When a writer executes* signal(rw mutex),we may resume the execution of either the waiting readers or a single waiting writer. The selection is made by the scheduler.