Dr. Hakim Mellah
Department of Computer Science & Software Engineering
Concordia University, Montreal, Canada

Lecture 7: Synchronization Examples

COMP 346: Operating Systems
winter 2020

These slides has been extracted, modified and updated from original slides of :

- ***Operating System Concepts, 10th Edition***, by: S**ilberschatz/Galvin/Gagne**, published by John Wiley & Sons

Classical Problems of Synchronization

- Classical problems used to test newly-proposed synchronization schemes
  - Bounded-Buffer Problem
  - Readers and Writers Problem

Bounded-Buffer Problem

- ***n*** buffers, each can hold one item
- Semaphore **`mutex`** initialized to the value 1
- Semaphore **`full`** initialized to the value 0
- Semaphore **`empty`** initialized to the value n

Bounded Buffer Problem (Cont.)

- The structure of the producer process

```
do {
      ...
     /* produce an item in next_produced */
      ...
    wait(empty);
    wait(mutex);
      ...
     /* add next produced to the buffer */
      ...
    signal(mutex);
    signal(full);
} while (true);
```

Bounded Buffer Problem (Cont.)

- The structure of the consumer process

```
Do {
    wait(full);
    wait(mutex);
       ...
    /* remove an item from buffer to next_consumed */
       ...
    signal(mutex);
    signal(empty);
       ...
    /* consume the item in next consumed */
       ...
 } while (true);
```

Readers-Writers Problem

- A data set is shared among a number of concurrent processes
  - Readers – only read the data set; they do ***not*** perform any updates
  - Writers – can both read and write
- Problem – allow multiple readers to read at the same time
  - Only one single writer can access the shared data at the same time
- Several variations of how readers and writers are considered – all involve some form of priorities
- Shared Data
  - Data set
  - Semaphore **`rw_mutex`** initialized to 1
  - Semaphore **`mutex`** initialized to 1
  - Integer **`read_count`** initialized to 0

Readers-Writers Problem (Cont.)

- The structure of a writer process

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

Readers-Writers Problem (Cont.)

- The structure of a reader process

```
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);
```

Readers-Writers Problem Variations

- ***First*** variation – no reader kept waiting unless writer has permission to use shared object
- ***Second*** variation – once writer is ready, it performs the write ASAP
- Both may have starvation leading to even more variations
- Problem is solved on some systems by kernel providing reader-writer locks