Recitation 9

TA Hanxiong Chen
hc691@rutgers.edu

Running Queue and Waiting Queue (joining queue)

See blackboard

Synchronization

Problem

- Threads may share data
- Data consistency must be maintained

Example

Suppose a thread wants to withdraw $5 from a bank account and another thread wants to deposit $10 to the same account

What should the balance be after the two transactions have been completed?

What might happen instead if the two transactions were executed concurrently?

Synchronization (cont'd)

The balance might be CB – 5

- Thread 1 reads CB (current balance)
- Thread 2 reads CB
- Thread 2 computes CB + 10 and saves new balance
- Thread 1 computes CB – 5 and saves new balance

The balance might be CB + 10

- How?

Synchronization matters when we are doing operations on **shared data**

Deadlock

A deadlock situation can arise if and only if all of the following conditions hold simultaneously in a system:

- Mutual Exclusion (resources)
- Hold and Wait (threads)
- No-preemption (thread->resource)
- Circular Wait (status)

These four conditions are known as the *Coffman conditions*

Synchronization Primitives

Most common primitives

- Locks (mutual exclusion)
- Condition variables: introduce signal mechanism
- Semaphores
- Monitors: Locks + condition variables
- Barriers

Lock (mutex) example

```
public class BankAccount
{
        Lock aLock = new Lock;
        int balance = 0;
        ...
public void deposit(int amount)
{
        aLock.acquire();
        balance = balance + amount;
        aLock.release();
}
```

```
public void withdraw(int amount)
{
        aLock.acquire();
        balance = balance - amount;
        aLock.release();
}
```

Mutex (lock) vs Semaphore

A semaphore provides a couple of different things than a mutex (lock):

- Firstly, it allows a resource to be concurrently accessed by at most N threads at any time. The value of N is dependent on the application. P() operation for decrement and V() operation for increment.
- It provides a signalling mechanism wait() - signal() type of semantics. V() operation always call signal() to notify the threads who are waiting for the resources.
- sem_wait()--P operation; sem_post()--V operation

Mutex (lock) vs Semaphore

Mutex lock looks like a binary semaphore

```
class Semaphore {
     private unsigned counter;
     ...
     public synchronized void P() {
     while (counter == 0)
          try { wait(); } catch (Exception e) {}
     counter--;      //counter = 0
     }

     public synchronized void V() {
     counter++;      //counter = 1
     notify();
     }
}
```

Mutex & semaphore

Mutex and binary semaphore are different!

- Threads are scheduled in a queue with mutex. (FCFS)
- Only the one acquires the lock can release the lock. (cannot lock twice)
- Any thread can do P() if we have available resources. Any thread who did P() can do V().
- Threads using semaphore are scheduled in a waiting queue and wait for a signal to be waken up. (FCFS is not necessary)
- Let’s see a demo

Demo

Write a program to print out with two threads:

*

**

***

****

*****

******

T0 thread only print line 2 4 6
T1 thread only print line 1 3 5

Conditional Variable (CV)

- A conditional variable is generally used for solving the busy waiting problems.
- Wait (c)
  - Atomically:
    - release the mutex *m*,
    - move this thread from the "ready queue" to "wait-queue" (a.k.a. "sleep-queue") of threads, and
    - sleep this thread. (Context is synchronously yielded to another thread.)
  - Once this thread is subsequently notified/signalled and resumed, then automatically re-acquire the mutex *m*.
- Signal(c)
  - wake up at most one waiting process/thread
  - if no waiting processes, signal is lost
- Broadcast
  - Wake up all the waiting threads (no broadcast in semaphore)

Conditional Variable VS Semaphore

- They are both using signal mechanism
- CV is mainly focusing on the condition synchronization, while semaphore is dealing with the number of available resources.
- Conditional variable has atomic operations but semaphore doesn’t.
- If no thread is in waiting queue, the signal of cv will lost. However, semaphore is still need to do V().

Monitor [3]

- A programming language construct that supports controlled access to shared data
- Monitor is a software module that encapsulates:
  - shared data structures
  - procedures that operate on the shared data
  - synchronization between concurrent processes that invoke those procedures
- Monitor protects the data from unstructured access
  - – guarantees only access data through procedures, hence in legitimate ways

Monitor [3]

- Mutual exclusion
  - only one process can be executing inside at any time
  - if a second process tries to enter a monitor procedure, it blocks until the first has left the monitor
    - more restrictive than semaphores!
    - but easier to use most of the time
- Once inside, a process may discover it can't continue, and may wish to sleep
  - or, allow some other waiting process to continue
  - condition variables provided within monitor
    - processes can wait or signal others to continue
    - condition variable can only be accessed from inside monitor

Producer/Consumer: Monitor

Shared variables

```
        cond_t not_empty, not_full;
        Int slots = 0;
        Mutex_t lock;
```

Producer

```
while(1) {
        mutex_lock(&lock);
        while (slots == N)
                cond_wait (&not_full, &lock);
        myi = get_empty (&buffer);
        Fill (&buffer[myi]);
        slots++;
        cond_signal (&not_empty);
        mutex_unlock (&lock);
}
```

Consumer

```
while(1) {
        mutex_lock(&lock);
        while (slots == N)
                cond_wait (&not_empty, &lock);
        myj = get_empty (&buffer);
        Use (&buffer[myj]);
        slots--;
        cond_signal (&not_empty);
        mutex_unlock (&lock);
}
```

When to use them?

Now we have learnt mutex, semaphore, condition variable, monitor. So when to use each of them?

- Only one thread can visit the critical section at one time and the visiting order is not a problem -- Mutex.
- More than one threads can visit a shared resource and you are focusing on the resource availability -- Semaphore
- Focusing on condition and want to solve busy waiting issue -- condition variable
- Want to encapsulate data and operations to keep data from being unstructured accessing -- monitor

Reference

1. https://inst.eecs.berkeley.edu/~cs162/fa13/hand-outs/synch.html
2. https://www.quora.com/What-is-the-difference-between-mutex-condition-variable-semaphore-and-monitor
3. http://pages.cs.wisc.edu/~swift/classes/cs537-sp09/lectures/22-monitors.pdf
4. https://docs.oracle.com/cd/E19455-01/806-5257/sync-31/index.html
5. https://jlmedina123.wordpress.com/2014/04/08/255/
6. http://www.cs.utexas.edu/users/witchel/372/lectures/08.Semaphore-Monitors.pdf