CPSC 457 T03/T04

Week 4 Day 1

Xining Chen

Agenda

- Condition variable
- unique_lock
- lock_guard
- Semaphores
- Assignment 3

Condition variables

- A synchronization primitive
- Used together with mutexes
- *When to use:* critical sections w/ loops waiting for some ‘condition’ to happen
- This ‘condition’ can only become true if another thread runs its critical section

```
// critical section protected with mutex m
lock(m)
  …
  while( ! some_condition ) { wait(cv); }
  …
unlock(m)
```

```
// some other thread …
lock(m)
  …
  some_condition = TRUE;
  signal(cv);
  …
unlock(m)
```

Download CPSC 457 – Synchronization – Code.pdf from D2L under W4D1

- Exercise + Demo

CV in C++

unique_lock in C++

lock_guard in C++

Semaphores

- Semaphores can be used to provide mutual exclusion.
- A semaphore has integer value.
- A locked semaphore can be unlocked by any thread
- Three operations: initialization, decrement (wait), increment (signal)
- decrement and increment must be executed **atomically**

Mutex	Semaphore
Binary operation/value (lock/unlock)	Integer value
Lock/unlock done by the same thread	Locked threads can be unlocked by any thread

Binary semaphore

- Behave very similarly to mutex locks.
- On systems that do not support mutex locks, binary semaphores can be used instead to provide mutual exclusion.
- Note: A locked thread can be unlocked by any thread using a semaphore.

Counting semaphores

- Can be used to control access to some given resource consisting of a finite number of instances.
  - **Initialize** the semaphore to the **number of resources available**
  - Each process that uses a resource performs a **wait()** operation (aka. decrement resource count)
  - When the count for the semaphore goes to 0, all resources are being used.
  - Additional processes that wish to use a resource will now be blocked until count > 0.

Assignment 3

Ignore 0,1

Input:	25	4012009	165
Factors:	1, 5	1, 2003, 4012009 * Typo on A3	1, 3, 5, ..., 165

Ignore 0,1

Input:	25	4012009	165
Factors:	1, 5	1, 2003, 4012009 * Typo on A3	1, 3, 5, ..., 165
Smallest non-trivial factor:	5	2003	3

Ignore 0,1

Input:	25	4012009	165
Factors:	1, 5	1, 2003, 4012009 * Typo on A3	1, 3, 5, ..., 165
Smallest non-trivial factor:	5	2003	3

$$\Sigma = 5 + 2003 + 3 = 2011$$

Output

Solution 1 : Let $N=2$

$$\left.\begin{array}{l}5\\10\\33\\1123\end{array}\right\}\text{Thread \#1}$$

$$\left.\begin{array}{l}56\\3\\10000\end{array}\right\}\text{Thread \#2}$$

Solution 1 : Let N = 2

$$\left.\begin{matrix}5\\10\\33\\1123\end{matrix}\right]\text{Thread \#1}$$

$$\left.\begin{matrix}56\\3\\10000\end{matrix}\right]\text{Thread \#2}$$

Not Good.

Ex://

#1 $\begin{bmatrix}132456799817\\10\\33\\1123\end{bmatrix}$

#2 $\begin{bmatrix}56\\3\\100\end{bmatrix}$

Thread #1 bottleneck.

⇒ runtime will be similar to single thread code.

Solution #2 : Let N = 2

$5 \rightarrow 1$

$10 \rightarrow 2$

$33 \rightarrow 1$

$1123789103 \rightarrow 2$

$56 \rightarrow 1$

$3 \rightarrow 1$

$10000 \rightarrow 1$

when 1 finishes

2 finishes.

Solution #2 : Let N=2

5 → 1

10 → 2

33 → 1

1123789103 → 2

56 → 1

3 → 1

10000 → 1

when 1 finishes

2 finishes.

Ok solution.

However, consider input: 1123789103.

Runtime?

Solution #3: Let N=2

5 → 1 2

10 → 1 2

33

1123789103 → 1 2

56 → 1 2

3

10000 → 1 2

Parallelize getSmallestDivisor().

Assignment 3 suggestions

1. Implement solution #2 first.
2. Use mutex, condition variable, semaphores, and pthread_barrier to try and implement solution #3.

Next Time

- pthread versions of everything
- Assignment 3 help (?)