

Synchronization

Experiments

1. No synchronization

```
[using System.Threading;  
namespace TestThreadNolock  
{  
    class Program  
    {  
        private static int x = 0; Share resource  
        static object _lock = new object();  
  
        static void FuncA() A กົດມູນ  
        {  
            int xx = 0;  
            while (xx < 50) ຂຸນ 50 ຂົນ → ກໍາຕະເປີມຄ່າເຮືອອງ  
            {  
                Console.WriteLine("FuncA: round:{0} x={1}", xx, x);  
                x++;  
                xx++;  
            }  
        }  
  
        static void FuncB() B กົດມູນ  
        {  
            int xx = 0;  
            while (xx < 50) ຂຸນ 50 ຂົນ → ກໍາຕະເປີມຄ່າເຮືອອງ  
            {  
                Console.WriteLine("==FuncB: round:{0} x={1}", xx, x);  
                x++;  
                xx++;  
            }  
        }  
  
        static void Main(string[] args)  
        {  
            Thread A = new Thread(new ThreadStart(FuncA));  
            Thread B = new Thread(new ThreadStart(FuncB));  
            A.Start();  
            B.Start();  
            }  
        }  
    }  
}
```

50 → 500 → 5000 ...

ບັນ ເລີ່ມກົດມູນ

2. Lock

```
using System.Threading;  
  
namespace TestThreadNolock  
{  
    class Program  
    {  
        private static int x = 0;  
        static object _lock = new object();  
  
        static void FuncA()  
        {  
            int xx = 0;  
            while (xx < 50)  
            {  
                lock (_lock)  
                {  
                    Console.WriteLine("FuncA: round:{0} x={1}", xx, x);  
                    xx++;  
                }  
                xx++;  
            }  
        }  
  
        static void FuncB()  
        {  
            int xx = 0;  
            while (xx < 50)  
            {  
                lock (_lock)  
                {  
                    Console.WriteLine("==FuncB: round:{0} x={1}", xx, x);  
                    xx++;  
                }  
                xx++;  
            }  
        }  
  
        static void Main(string[] args)  
        {  
            Thread A = new Thread(new ThreadStart(FuncA));  
            Thread B = new Thread(new ThreadStart(FuncB));  
            A.Start();  
            B.Start();  
        }  
    }  
}
```

9! Lock

9! Lock

3. No Synchronization

Desired Output

```
Input: 1  
X = 1  
Input: 2  
X = 2  
Input: 3  
X = 3  
Input: 4  
X = 4  
Input: 5  
X = 5  
Input: 6  
X = 6  
Input: 7  
X = 7  
Input: 8  
X = 8  
Input: 9  
X = 9  
Input: 99  
X = 99  
Input: 999  
X = 999  
Input: exit  
Thread 1 exit
```

```
using System.Threading;  
  
namespace OS_Sync_01  
{  
    class Program  
    {  
        private static string x = "";  
        private static int exitflag = 0;  
  
        static void ThReadX()  
        {  
            while(exitflag==0)  
                Console.WriteLine("X = {0}", x);  
        }  
        static void ThWriteX()  
        {  
            string xx;  
            while (exitflag == 0)  
            {  
                Console.Write("Input: ");  
                xx = Console.ReadLine();  
                if (xx == "exit")  
                    exitflag = 1;  
                else  
                    x = xx;  
            }  
        }  
        static void Main(string[] args)  
        {  
            Thread A = new Thread(ThReadX);  
            Thread B = new Thread(ThWriteX);  
  
            A.Start();  
            B.Start();  
        }  
    }  
}
```

4. Try #1

```
static void ThReadX(Object i)
{
    while (exitflag == 0)
    {
        while (updateFlag == 0) ;
        if (x != "exit")
        {
            Console.WriteLine("Thread {0} : X = {1}", i, x);
            updateFlag = 0;
        }
    }
}
static void ThWriteX()
{
    string xx;
    while (exitflag == 0)
    {
        Console.Write("Input: ");
        xx = Console.ReadLine();
        if (xx == "exit")
            exitflag = 1;
        x = xx;
        updateFlag = 1;
    }
}
```

5. Try #2

```
static void Main(string[] args)
{
    Thread A = new Thread(ThReadX);
    Thread B = new Thread(ThWriteX);
    Thread C = new Thread(ThReadX);
    Thread D = new Thread(ThReadX);

    A.Start(1);
    B.Start();
    C.Start(2);
    D.Start(3);
}
```

6. Condition Variable

มี 4 Thread
1 thread รับ input
สืบ 3 thread รอ input
ถ้า 3 thread ได้ input
ก็ 3 thread ต้อง share
thread ที่รับ input ออก

```
using System.Threading;  
namespace OS_Sync_04  
{  
    class Program  
    {  
        private static string x = "";  
        private static int exitflag = 0;  
        private static int updateFlag = 0;  
        private static object _Lock = new object();  
  
        static void ThReadX(object i)  
        {  
            while (exitflag == 0)  
            {  
                lock (_Lock)  
                {  
                    while (updateFlag == 0)  
                        Monitor.Wait(_Lock);  
                    if (x != "exit")  
                    {  
                        Console.WriteLine("Thread {0} : X = {1}", i, x);  
                        //Console.WriteLine("X = {0}", x);  
                        updateFlag = 0;  
                    }  
                }  
                Console.WriteLine("Thread {0} exit", i);  
            }  
        }  
        static void ThWriteX()  
        {  
            string xx;  
            while (exitflag == 0)  
            {  
                lock (_Lock)  
                {  
                    Console.Write("Input: ");  
                    xx = Console.ReadLine();  
                    if (xx == "exit")  
                        exitflag = 1;  
                    LOCK  
                    x = xx;  
                    updateFlag = 1;  
                    //Monitor.PulseAll(_Lock);  
                    Monitor.Pulse(_Lock);  
                    Thread.Sleep(100);  
                }  
            }  
        }  
        static void Main(string[] args)  
        {  
            Thread A = new Thread(ThReadX);  
            Thread B = new Thread(ThWriteX);  
            Thread C = new Thread(ThReadX);  
            Thread D = new Thread(ThReadX);  
  
            A.Start(1);  
            B.Start();  
            C.Start(2);  
            D.Start(3);  
        }  
    }  
}
```

ไม่ใช้ lock ทุกที่

จะเข้าสู่ queue

เข้าสู่ lock ไม่ถูก sleep
(ไม่ต้อง share resource)

ไม่ต้อง release lock in check ใหม่

ก็จะสังเกต concurrent ลักษณะ

ເສື່ອງໃຈ pulse ກົບ pulse all thread ກ່ອນກວາເປັນ Random

ຜລລົພບຂອງກວາຄລ້າຍກົດ ແຕກກະທານ ເຊິ່ງບໍລິຫານໄມ່ແມ່ນແກ້ໄຂ

pulse → thread ຊູກປຸກ / ຊູກເລືອກຈາກ ຕົ້ນ frame work

ປຸກກວາເສີຍງົງ ຕົ້ນ ການປຸກ ສໍາອສນະ: wait ໃນ ready

pulse all → ກົດ 3 thread ຈະ ຊູກປຸກ (wait → ready) ພໍອມກັນ

ເຮັດໄປແຍ່ງກືນກະທານ ອົກ ສຳ Lock ມາກຳວັນໄອັດວິນ

ກິດກຳ ອາຍືນກືນ scheduler

Definitions

Race condition: output of a concurrent program depends on the order of operations between threads.

Mutual exclusion: *ก็เป็น thread ก็ต้องมี 1 ช่องเวลา = ก็ต้อง lock*
only one thread does a particular thing at a time

锁 lock ครอบ

– **Critical section:** piece of code that only one thread can execute at once 1 thread / ช่องเวลา
share resource

lock ตอนนี้

– **Lock before entering critical section,** before accessing shared data

unlock ตอนออก

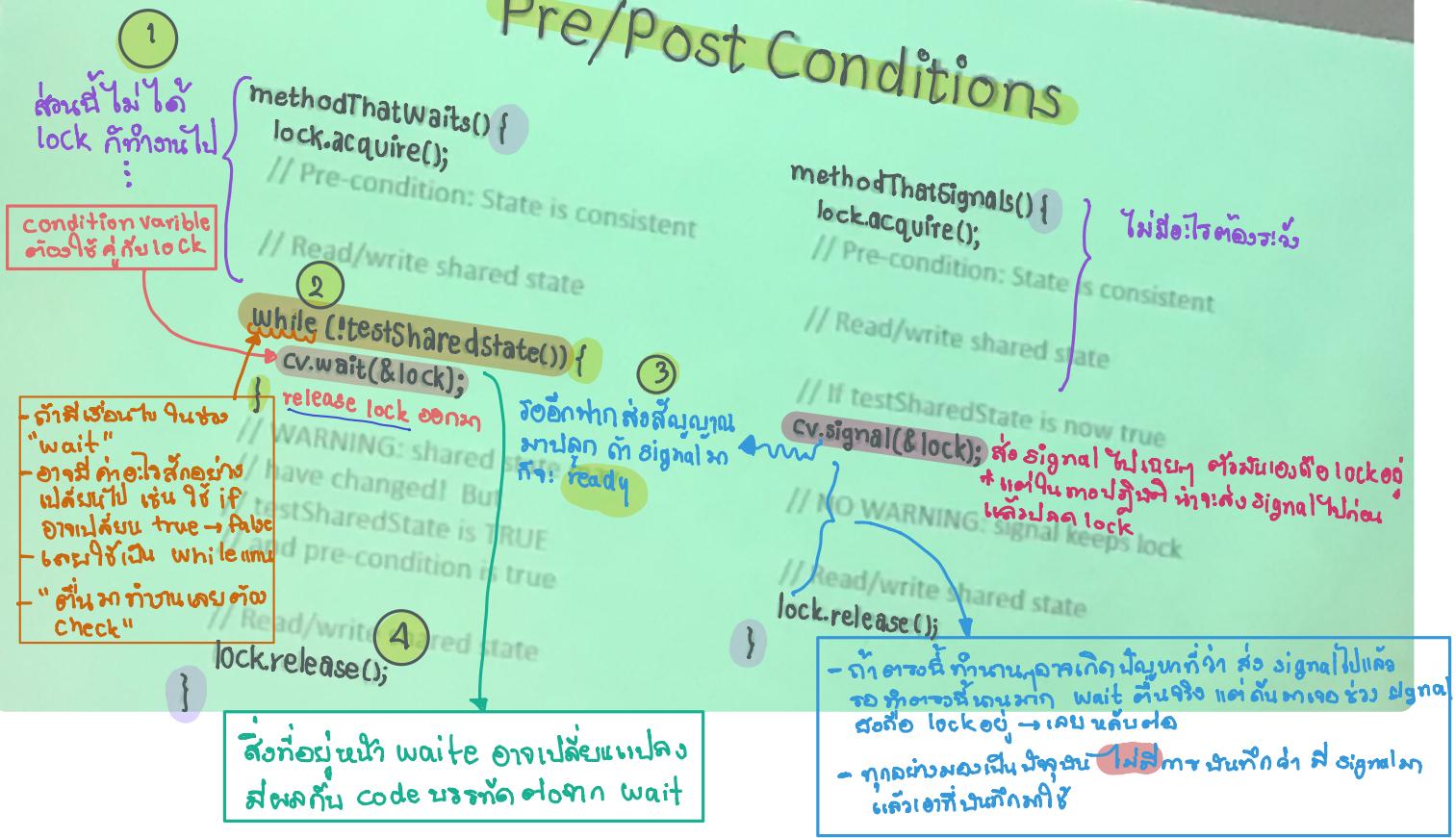
– **Unlock when leaving,** after done accessing shared data

– Wait if locked (all synchronization involves waiting!)

Pre/Post Conditions

- What is state of the bounded buffer at lock acquire?
 - $\text{front} \leq \text{tail}$
 - $\text{front} + \text{MAX} \geq \text{tail}$
- These are also true on return from wait
- And at lock release
- Allows for proof of correctness

Pre/Post Conditions



Condition Variables

- ใช้คู่กัน lock และ
- **ALWAYS hold lock when calling wait, signal, broadcast**
 - Condition variable is sync FOR shared state
- **บังคับ การเขียนฟังก์ชัน signal หรือ check ต้องรู้ว่า ที่ thread ที่ wait อยู่ มีอยู่ สักกี่ตัว ไม่มีกี่ตัว**
 - **ALWAYS hold lock when accessing shared state**
- **Condition variable is memoryless**
 - If signal when no one is waiting, no op
 - If wait before signal, waiter wakes up
- **Wait atomically releases lock**
 - What if wait, then release?
 - What if release, then wait?

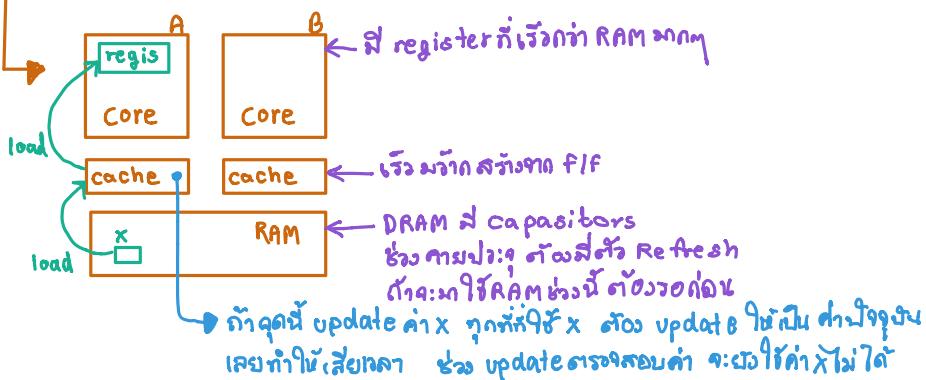
Condition Variables, cont'd

๑๐๗

- เมื่อหูก ปุ๊ก แล้ว wait → ready ให้หนึ่น ๆ คล้อ release lock
 - When a thread is woken up from wait, it may not run immediately
 - Signal/broadcast put thread on ready list
 - When lock is released, anyone might acquire it
 - Wait MUST be in a loop
 - while (needToWait()) {
 condition.Wait(lock);
}
 - Simplifies implementation
 - Of condition variables and locks
 - Of code that uses condition variables and locks
- ยังไง lock ไม่ได้
กรณีนี้ thread
จะเข้าสู่ ready อยู่
ตัวเขียนต้องปลด lock
ก่อนแล้ว
- wait แล้วต้องอยู่ใน loop
ต้นทางเดินของ lock
ต้องเข้าสู่ sleep ต่อไป
แล้วกลับมาเป็น wait state อีก
เมื่อเข้าสู่ ก็ต้องร่วม share resource ต่อ

Synchronization Performance

- A program with lots of concurrent threads can still have poor performance on a multiprocessor:
 - Overhead of creating threads, if not needed กรณีการสร้าง thread มากเกินไป ทำให้ overhead เสียเวลา
 - Lock contention: only one thread at a time can hold a given lock กรณีที่ต้องการเข้าใช้锁 แต่锁มีจำนวนจำกัด ทำให้ overhead เสียเวลา
 - Shared data protected by a lock may ping back and forth between cores กรณีข้อมูลที่ต้องการเข้าใช้ต้องผ่าน锁 แต่锁ตั้งอยู่ใน core ที่ต่างกัน ทำให้ overhead เสียเวลา
 - False sharing: communication between cores even for data that is not shared กรณีข้อมูลที่ไม่ใช่ของ core ที่ต้องการเข้าใช้ แต่ core ที่ต้องการเข้าใช้ต้องรู้ว่าต้องการเข้าใช้ข้อมูลนี้ ทำให้ overhead เสียเวลา

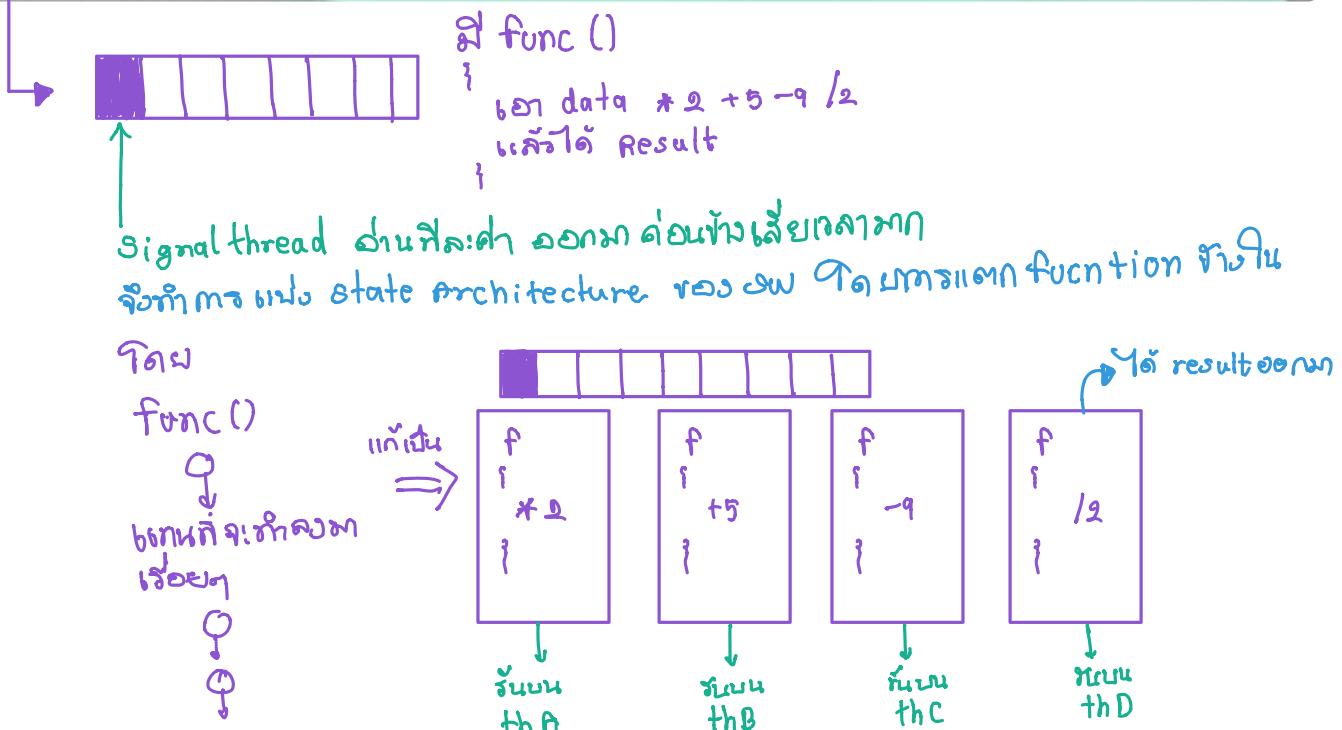


จัด X ใน Array จะ update กี่ครั้ง: update เป็น Block ซึ่งเป็น Block นึง เป็นไปได้
โดยค่อนข้างนาน

Reducing Lock Contention

- ## Using Lock Contention

 - "粒度锁 lock"
 - Fine-grained locking กรณี lock ที่มีลักษณะเดียวกันหลายตัว lock ร่วมกัน
แบบ partition object into subsets, each protected by its own local lock แต่ละตัว
 - Example: hash table buckets กรณี split Array เป็น 2 ก้อน กรณีหลาย thread ทำงาน
 - Per-processor data structures 
 - Partition object so that most/all accesses are made by one processor
 - Example: per-processor heap
 - Ownership/Staged architecture
 - Only one thread at a time accesses shared data
 - Example: pipeline of threads



"ກຳໄຟ້ເວັນ"

Deadlock Definition

▶ នូវ Resource នៅលើ តាម share ក្នុង ផែនវគ្គមិថយក lock តាមនីហរម៉ាស៊ី → ឧបករណ៍
"SO Resource" នៅមួរប៉ុច តាមកិច្ច ឬវិធានបន្ទាន់ខាងក្រោម នៅពីរ តើកីន តួកនឹង

ဗုံးများ th ၁ ရှိခဲ့သူ Resource A → စုစုံ Resource B
th ၂ ရှိခဲ့သူ Resource B → စုစုံ Resource C
"အပေါ်အပေါ် ရှိခဲ့သူများ စုစုံမှုများ ရှိခဲ့သူများ"

"Dead lock" Example: two locks

กรณี run พร้อมกัน → Deadlock

Thread A

```

lock1.acquire();
lock2.acquire(); รอลๆ รอของ B
lock2.release();
lock1.release();

```

Thread B

```

lock2.acquire();
lock1.acquire(); รอลๆ รอของ A
lock1.release();
lock2.release();

```

เมื่อกำหนดให้ thread ที่มี lock สองตัว ก็ต้องรอกกัน แต่เมื่อสักพักผ่านไปแล้ว scheduler จัดการ



"Dead lock" Two locks and a condition variable

ถ้า th A ซึ่ง lock 1 มาก่อน และ th B ซึ่ง lock 2 มาก่อน

Thread A

```

lock1.acquire();
...
lock2.acquire();
while (need to wait) {
    condition.wait(lock2);
}
lock2.release();
...
lock1.release();

```

- ปล่อย lock 2 ไป
- เนื่องจาก锁 1 อยู่ในปั๊ะอยู่
- กรณี th B ไม่ได้ lock
- Signal หากรับ lock 2 แล้ว
- A ทำการunlock ไม่ได้

Thread B

```

lock1.acquire();
...
lock2.acquire();
...
condition.signal(lock2);
...
lock2.release();
...
lock1.release();

```

~~condition.signal(lock2);~~

" การเกิด Dead lock "

Necessary Conditions for Deadlock

- Limited access to resources resource จำกัด เมื่อนี้ หลักเกี่ยงไม่ได้
 - If infinite resources, no deadlock!
- No preemption ต้อง OS หน่วยคำนวณ ออก หลักเกี่ยงไม่ได้
 - If resources are virtual, can break deadlock
- Multiple independent requests
 - "wait while holding" กรณี lock ที่ไปใน wait state หลักเกี่ยงได้
- Circular chain of requests เช่น Ex. Two lock หลักเกี่ยงได้

Preventing Deadlock

- ความคุ้มครองการเขียนโปรแกรม
 - Exploit or limit program behavior
 - ใช้รูปแบบการ lock เช่น ใช้สับล็อกกิ้ง pattern เช่น ก่อน lock 1 และ lock 2 ต้องรู้ว่า Resource A, B ที่ไหนก่อน แล้วกัน lock 2 ก่อน
 - Limit program from doing anything that might lead to deadlock
- Predict the future ใช้ปัจจัยต่างๆ ในการตัดสินใจ
 - If we know what program will do, we can tell if granting a resource might lead to deadlock
- Detect and recover
 - อาจฝึกอบรม programme ให้ detect กรณีงานของ thread ที่อยู่ใน deadlock แล้ว อาจต้อง kill หรือตั้งกลับไปทำงานใหม่
 - If we can rollback a thread, we can fix a deadlock once it occurs