Synchronization

The Dark Side of Concurrency

With interleaved executions, the order in which processes execute at runtime is *nondeterministic*.

- depends on the exact order and timing of process arrivals
- depends on exact timing of asynchronous devices (disk, clock)
- depends on scheduling policies

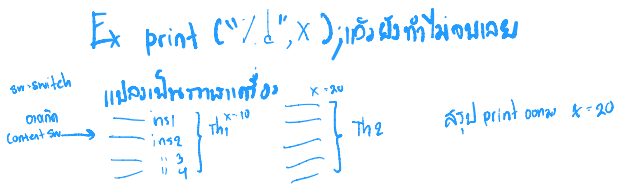
Some schedule interleavings may lead to incorrect behavior.

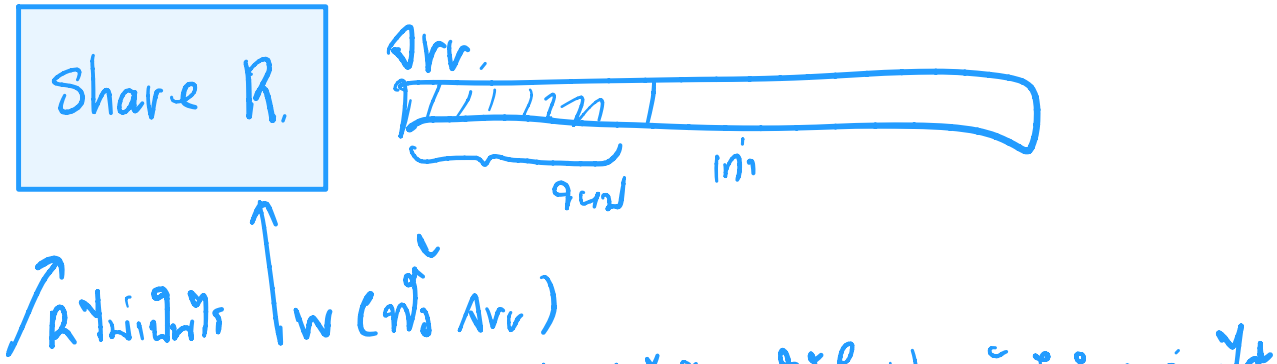
- Open the bay doors *before* you release the bomb.
- Two people can't wash dishes in the same sink at the same time.

The system must provide a way to coordinate concurrent activities to avoid incorrect interleavings.

Synchronization Motivation

- When threads concurrently read/write shared memory, program behavior is undefined
  - Two threads write to the same variable; which one should win?
- Thread schedule is non-deterministic
  - Behavior changes when re-run program
- Compiler/hardware instruction reordering
- Multi-word operations are not atomic





R ไม่เป็นไร W (ทั้ง Arr)

ถ้าอยู่ ใหม่-เก่า แก้ อัพเดทใน finish แล้วถึงทำการอ่านได้

Question: Can this panic?

Thread 1

```
p = someComputation();
pInitialized = true;
```

Thread 2

```
while (!pInitialized)
    ;
q = someFunction(p);
if (q != someFunction(p))
    panic
```

Why Reordering?

- Why do compilers reorder instructions?
  - Efficient code generation requires analyzing control/data dependency
  - If variables can spontaneously change, most compiler optimizations become impossible
- Why do CPUs reorder instructions?
  - Write buffering: allow next instruction to execute while write is being completed

Fix: **memory barrier** protect

  - Instruction to compiler/CPU
  - All ops before barrier complete before barrier returns
  - No op after barrier starts until barrier returns

Too Much Milk Example

ดูตู้เจอนมหมดออกไปซื้อ

	Person A	Person B
12:30	Look in fridge. Out of milk. (หมดนม)	
12:35	Leave for store. (ซื้อ)	
12:40	Arrive at store.	Look in fridge. Out of milk. (เจอนมหมด)
12:45	Buy milk.	Leave for store.
12:50	Arrive home, put milk away.	Arrive at store.
12:55		Buy milk.
1:00		Arrive home, put milk away. Oh no!

สรุป มีนม 2 ขวด → เกิดความซ้ำซ้อน / แย่งกันอัพเดต

Race condition

ใช้ lock แก้ปัญหา

↓

Definitions

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

Mutual exclusion: only one thread does a particular thing at a time

- **Critical section:** piece of code that only one thread can execute at once

Lock: prevent someone from doing something

- Lock before entering critical section, before accessing shared data
- Unlock when leaving, after done accessing shared data
- Wait if locked (all synchronization involves waiting!)

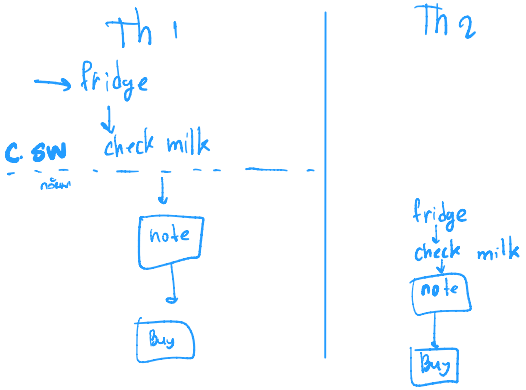
ทำให้เกิด Race condition ได้อยู่

Too Much Milk, Try #1

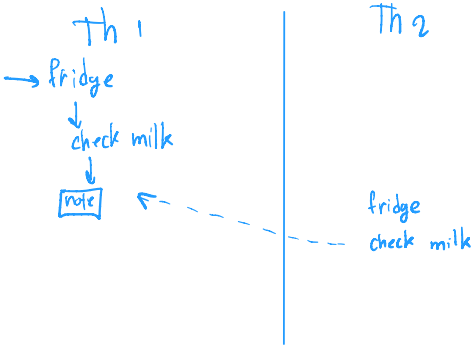
- Correctness property
  - Someone buys if needed (liveness)
  - At most one person buys (safety)
- Try #1: leave a note

```
if (!note)
    if (!milk) {
        leave note
        buy milk
        remove note
    }
```

Worse



Best



Worse

Too Much Milk, Try #2

Thread A

```
leave note A
if (!note B) {
    if (!milk)
        buy milk
}
remove note A
```

Thread B

```
leave note B
if (!noteA) {
    if (!milk)
        buy milk
}
remove note B
```

Too Much Milk, Try #3

Thread A

```
leave note A
while (note B) // X
    do nothing;
if (!milk)
    buy milk;
remove note A
```

Thread B

```
leave note B
if (!noteA) {   // Y
    if (!milk)
        buy milk
}
remove note B
```

Can guarantee at X and Y that either:

(i) Safe for me to buy

(ii) Other will buy, ok to quit

Lessons

- Solution is complicated
  - “obvious” code often has bugs
- Modern compilers/architectures reorder instructions
  - Making reasoning even more difficult
- Generalizing to many threads/processors
  - Even more complex: see Peterson’s algorithm

อัลกอริทึม ซับซ้อนหมากขึ้น

Roadmap

Concurrent Applications

Shared Objects

Bounded Buffer Barrier

Synchronization Variables

Semaphores Locks Condition Variables

Atomic Instructions

Interrupt Disable Test-and-Set

Hardware

Multiple Processors Hardware Interrupts

Locks

- Lock::acquire
  - wait until lock is free, then take it
- Lock::release
  - release lock, waking up anyone waiting for it

1. At most one lock holder at a time (safety)
2. If no one holding, acquire gets lock (progress)
3. If all lock holders finish and no higher priority waiters, waiter eventually gets lock (progress)

Too Much Milk, #4

Locks allow concurrent code to be much simpler:

```
lock.acquire();
if (!milk)
    buy milk
lock.release();
```

} critical

Lock Example: Malloc/Free

```
char *malloc (n) {
    heaplock.acquire();
    p = allocate memory
    heaplock.release();
    return p;
}
```

```
void free(char *p) {
    heaplock.acquire();
    put p back on free list
    heaplock.release();
}
```

Rules for Using Locks

- Lock is initially free
- Always acquire before accessing shared data structure
  - Beginning of procedure!
- Always release after finishing with shared data
  - End of procedure!
  - Only the lock holder can release
  - DO NOT throw lock for someone else to release
- Never access shared data without lock
  - Danger!

ผลลัพธ์ที่ออกใช้ไม่ได้

- ถ้า set p = Null code ทำให้ผิด
- ถ้า p != Null code ก็ใช้ไม่ได้

Will this code work?

```
if (p == NULL) {
    lock.acquire();
    if (p == NULL) {
        p = newP();
    }
    lock.release();
}
use p->field1
```

```
newP() {
    p = malloc(sizeof(p));
    p->field1 = …
    p->field2 = …
    return p;
}
```

Semaphores

- Semaphore has a non-negative integer value
  - P() atomically waits for value to become > 0, then decrements
  - V() atomically increments value (waking up waiter if needed)
- Semaphores are like integers except:
  - Only operations are P and V
  - Operations are atomic
    - If value is 1, two P's will result in value 0 and one waiter
- Semaphores are useful for
  - Unlocked wait: interrupt handler, fork/join

ใช้กำหนดลำดับ: ใช้ระหว่างการทำงาน Thr ใดๆ & ใช้คู่กับ lock

Condition Variables

func

- Waiting inside a critical section
  - Called only when holding a lock

สำเป็นเท่ากันเลย

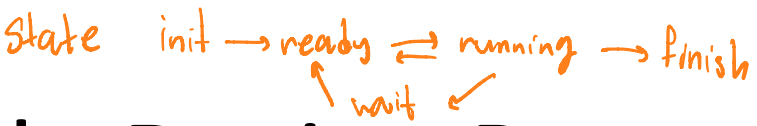
- Wait: atomically release lock and relinquish processor
  - Reacquire the lock when wakened

ส่งไปปลุก wait

- Signal: wake up a waiter, if any จะปลุกแค่ 1 Thr ตามอัลกอริทึม
- Broadcast: wake up all waiters, if any

ปลุกทุกตัวให้แย่งกันทำงาน ปลุกทุกตัวให้ตื่น

Condition Variable Design Pattern




```
methodThatWaits() {
   lock.acquire();
   // Read/write shared state
   while (!testSharedState()) {
      cv.wait(&lock);
   }
   // Read/write shared state
   lock.release();
}
```

```
methodThatSignals() {
   lock.acquire();
   // Read/write shared state
   // If testSharedState is now true
   cv.signal(&lock);
   // Read/write shared state
   lock.release();
}
```

C#

lock 1 Thr ทำได้ทีละ 1

- Lock

```
lock (_Lock)
{
    ...
}
```

ล็อกผ่าน

lock.acquire

lock.release

ถึง { } compile ไม่ผ่าน

- Condition Variable

```
static object _Lock = new object();

lock (_Lock)
{
    ...
    Monitor.Wait(_Lock);
    or
    Monitor.Pulse(_Lock);
    Monitor.PulseALL(_Lock);
    ...
}
```

Signal

Broadcast

C#

- Semaphore

เข้าไป ทำพร้อมกันได้

P ลดค่า 1

V เพิ่มค่า 1

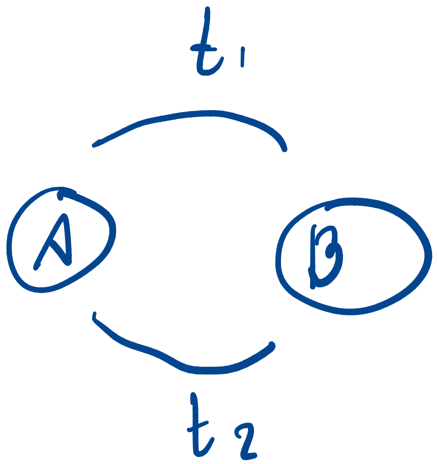
ลดพร้อม, เพิ่มพร้อม

ถ้าต้องการ 3 Thr ทำงานพร้อมกัน (3, 1)

```
Semaphore s = new Semaphore(1, 1);

s.WaitOne();   // P()
  …
  …
  …
s.Release(); //V()
```

ถ้ามี share resouce



Dead Lock

t1

Content sw *

```
lock (_lockA )   ① t1 ถือ lockA
{
    lock (_lockB )   ③ t1 จะใช้ lock B ไม่ได้เพราะ t2 ถืออยู่
    {   }

}
```

t2

t2 ถือ lock B

```
lock (_lockB )   ②
{
    lock (_lockA )   ④ t2 ไม่ได้ใช้ _lock A เพราะ t1 ถืออยู่
    {   }

}
```

Dead Loc วิธีแก้

t1

```
lock (_lockA )
{
    lock (_lockB )
    {   }

}
```

t2

```
lock (_lockA )
{
    lock (_lockB )
    {   }

}
```

wait while holding   DeadLock อีกประเภท

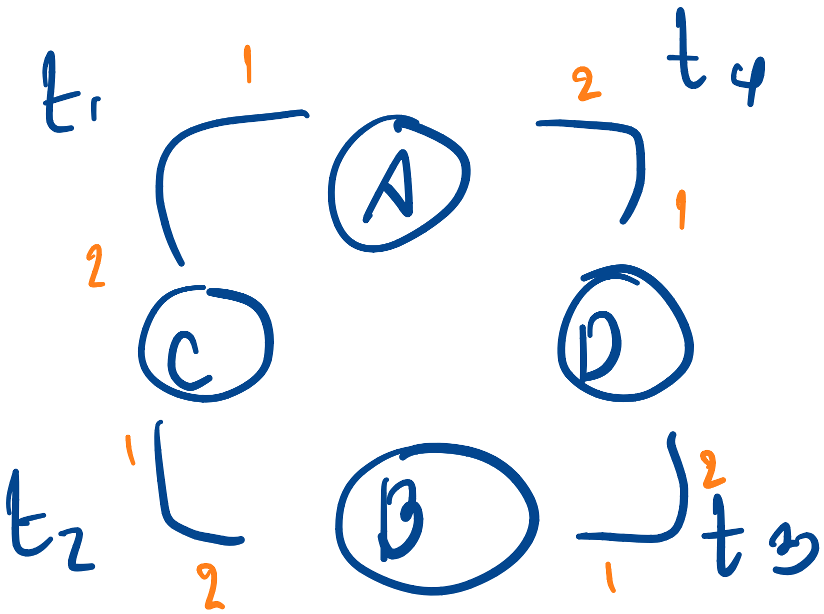
thr ใช้ไม่ได้

```
t1

lock (_lockA)                     lock (_lockA)
{                                 {
    thread.wait();                    thread.signal();
}                                 }
```

Dead Lock
อีกประเภท

Circular chain of request



t1 . acquire (A)

t . acquire

t . acquire

t . acquire

- ลำดับการ acquire
Ex t1 acquire (A) ก่อน แล้วค่อย (B)

ปัญหา

เกิดสิ่งนี้
(เกิดนี้)
1. limit access to resource → Lock
(ส่งผลตามมา) 2. Circular chain of request : Deadlock
(ส่งผลตามมา) 3. wait while holding

(1,2,3) (1,2) (1,3) ก็เกิด Deadlock

(3,2) ไม่เกิด Deadlock เพราะไม่มี lock

ไม่เกิด Deadlock
4. No Preemption ปัจจัยภายนอก / การแทรกแซง

Ex. การทำ Context SW , ค้างอยู่ ค่าเดิมเป็น 10 นาที

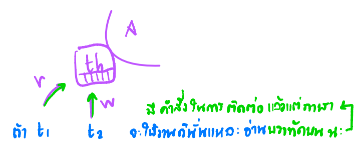
วิธีแก้

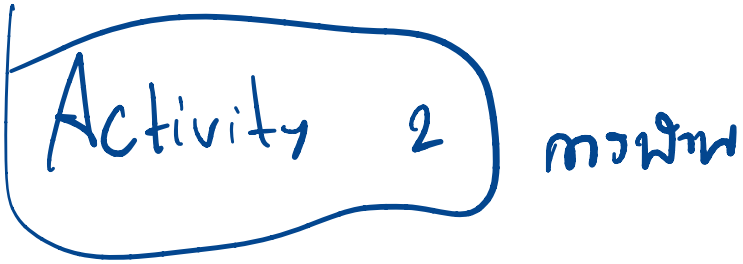
- เพิ่ม Thr 5 เพื่อ reset การทำงาน
- บางภาษารองรับการเช็คว่า acquire lock สำเร็จมั้ย "acquire ได้ทั้งหมด = ทำงาน"

```
Ex if ( LockA. acquire ())
       if ( LockB. acquire ())
          if (LockC . acquire ())
              "ทำงาน"

   else   lockA. release();
          lockB. release();
```

- หลีกเลี่ยงการใช้ lock จะ read /write ทำผ่าน [th]





Synchronization

Example & Exercise

1. Discussion

- ให้ศึกษาโปรแกรม Ex00, Ex01, Ex02 และ Ex03
- ก่อนจะทดลอง run โปรแกรมให้ นศ อภิปรายกันในกลุ่มเกี่ยวกับ
  - ผลลัพธ์จากแต่ละโปรแกรม ในแง่ความถูกต้อง
  - ความเร็วในการทำงานคาดว่าโปรแกรมใดจะเร็วที่สุด เพราะเหตุใด
- ทำการ run แต่ละโปรแกรมเพื่อทดสอบสมมุติฐานที่ได้อภิปรายกันไว้
- เปรียบเทียบผลที่คาดการณ์กับผลที่ run ได้
  - หาเหตุผลสนับสนุน
- สรุป

Ex-00

```
using System;
using System.Diagnostics;
using System.Threading;

namespace OS_Sync_Ex_01
{
    class Program
    {
        private static int sum = 0;

        static void plus()
        {
            int i;
            for (i = 1; i < 1000001; i++)
                sum += i;
        }

        static void minus()
        {
            int i;
            for (i = 0; i < 1000000; i++)
                sum -= i;
        }

        static void Main(string[] args)
        {
            Stopwatch sw = new Stopwatch();
            Console.WriteLine("Start...");
            sw.Start();
            plus();
            minus();
            sw.Stop();
            Console.WriteLine("sum = {0}", sum);
            Console.WriteLine("Time used: " + sw.ElapsedMilliseconds.ToString() + "ms");
        }
    }
}
```

Ex-01

```
using System;
using System.Diagnostics;
using System.Threading;

namespace OS_Sync_Ex_01
{
    class Program
    {
        private static int sum = 0;

        static void plus()
        {
            int i;
            for (i = 1; i < 1000001; i++)
                sum += i;
        }

        static void minus()
        {
            int i;
            for (i = 0; i < 1000000; i++)
                sum -= i;
        }

        static void Main(string[] args)
        {
            Thread P = new Thread(new ThreadStart(plus));
            Thread M = new Thread(new ThreadStart(minus));

            Stopwatch sw = new Stopwatch();
            Console.WriteLine("Start...");
            sw.Start();

            P.Start();
            M.Start();

            P.Join();
            M.Join();

            sw.Stop();
            Console.WriteLine("sum = {0}", sum);
            Console.WriteLine("Time used: " + sw.ElapsedMilliseconds.ToString() + "ms");
        }
    }
}
```

Ex-02

```
using System;
using System.Diagnostics;
using System.Threading;

namespace OS_Sync_Ex_01
{
    class Program
    {
        private static int sum = 0;
        private static object _Lock = new object();

        static void plus()
        {
            int i;
            for (i = 1; i < 1000001; i++)
                lock (_Lock)
                {
                    sum += i;
                }
        }

        static void minus()
        {
            int i;
            for (i = 0; i < 1000000; i++)
                lock (_Lock)
                {
                    sum -= i;
                }
        }

        static void Main(string[] args)
        {
            Thread P = new Thread(new ThreadStart(plus));
            Thread M = new Thread(new ThreadStart(minus));

            Stopwatch sw = new Stopwatch();
            Console.WriteLine("Start...");
            sw.Start();

            P.Start();
            M.Start();

            P.Join();
            M.Join();

            sw.Stop();
            Console.WriteLine("sum = {0}", sum);
            Console.WriteLine("Time used: " + sw.ElapsedMilliseconds.ToString() + "ms");
        }
    }
}
```

Ex-03

```
using System;
using System.Diagnostics;
using System.Threading;

namespace OS_Sync_Ex_01
{
    0 references
    class Program
    {
        private static int sum = 0;
        private static object _Lock = new object();

        1 reference
        static void plus()
        {
            int i;
            lock (_Lock)
            {
                for (i = 1; i < 1000001; i++)
                    sum += i;
            }
        }

        1 reference
        static void minus()
        {
            int i;
            lock (_Lock)
            {
                for (i = 0; i < 1000000; i++)
                    sum -= i;
            }
        }
        2
        0 references
        static void Main(string[] args)
        {
            Thread P = new Thread(new ThreadStart(plus));
            Thread M = new Thread(new ThreadStart(minus));

            Stopwatch sw = new Stopwatch();
            Console.WriteLine("Start...");
            sw.Start();

            P.Start();
            M.Start();

            P.Join();
            M.Join();

            sw.Stop();
            Console.WriteLine("sum = {0}", sum);
            Console.WriteLine("Time used: " + sw.ElapsedMilliseconds.ToString() + "ms");
        }
    }
}
```

ตัวแทนส่ง

2. Modification

ดัดแปลง

- ให้ดัดแปลงแก้ไขโปรแกรม **Ex-04** ให้ทำงานแล้วได้ผลลัพธ์ดังรูป **R-01** ในหน้าถัดไป

Ex-04

```
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();
        }
    }
}
```

รับ จาก console

R-01

```
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
```

ตัวแทนสง

3. Modification

- ให้ดัดแปลงแก้ไขโปรแกรม Ex-05 ให้ทำงานแล้วได้ผลลัพธ์ดังรูป R-02 ในหน้าถัดไป

Ex-05

```
using System;
using System.Threading;

namespace cv_lab
{
    class Program
    {
        private static string x ="";
        private static int exitflag = 0;
        private static int updateFlag = 0;

        static void ThReadX(object i)
        {
            while(exitflag == 0)
            {
                    if (x != "exit")
                    {
                        Console.WriteLine("***Thread {0} : x = {1}***",i,x);
                    }
            }
            Console.WriteLine("---Thread {0} exit---", i);
        }
        static void ThWriteX()
        {
            string xx;
            while(exitflag == 0)
            {
                    Console.Write("Input: ");
                    xx = Console.ReadLine();
                    if (xx == "exit")
                        exitflag = 1;
                    x = xx;
            }
        }
        static void Main(string[] args)
        {
            Thread A = new Thread(ThWriteX);
            Thread B = new Thread(ThReadX);
            Thread C = new Thread(ThReadX);
            Thread D = new Thread(ThReadX);

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

R-02

มี 1 Thr เท่านั้น ที่แสดงผล
L Thr 2, 3 ไม่ต้อง
แสดงผล ซ้ำซ้อน อีก

ลำดับ Thr ไม่ต้องตาม
ของ Pattern ไปสังเกตุ
ดีๆ

```
Input: 1
***Thread 1 : x = 1***
Input: 2
***Thread 3 : x = 2***
Input: 3
***Thread 3 : x = 3***
Input: 4
***Thread 3 : x = 4***
Input: 5
***Thread 3 : x = 5***
Input: 6
***Thread 1 : x = 6***
Input: 7
***Thread 1 : x = 7***
Input: 1111
***Thread 1 : x = 1111***
Input: 99
***Thread 3 : x = 99***
Input: exit
---Thread 2 exit---
---Thread 3 exit---
---Thread 1 exit---
```

Case S.2

20 ต.ค. Present

Front ตัวแรก

Back ตัวท้าย

Count จำนวนใหลิตาวิเปล่า ?

ให้ไฟล์โปรแกรมใส่ ชื่อ - รหัส เป็น comment