

CS Bridge Module 24 Concurrency and Deadlocks

1. Concurrency and Deadlocks

1.1 CS Bridge: Concurrency and Deadlocks



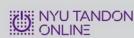
NYU TANDON
ONLINE

CS Bridge: Concurrency and Deadlocks

Module 24
Dan Katz

1.2 Concurrency and Deadlocks - Intro

Introduction



- Reminder about threads
- Features of having multiple threads
- Possible ways to have asynchrony
- Critical Sections
- Examples of concurrency issues
- Mutual Exclusion
- Software solutions for Mutual exclusion
- Hardware Options
- Semaphores
- Deadlocks
- Solutions to deadlocks
- Dining Philosopher's problem



1.3 Reminder about threads

Reminder About Threads

NYU TANDON
ONLINE

- Threads all share the resources of the process.
- Threads run as if they were a separate program.
- Threads can run asynchronously.



1.4 Features of having multiple threads

Features of Having Multiple Threads

NYU TANDON
ONLINE

- Ease of communication
- Effective solution to prevent blocking while reading data
- Threads are relatively easy to create
- Risk of asynchrony

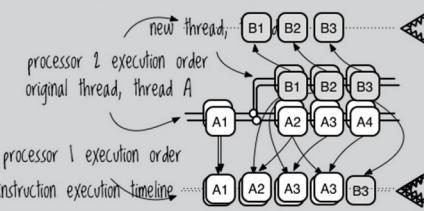


1.5 Asynchrony

Asynchrony

NYU TANDON
ONLINE

- Asynchrony occurs when two threads are running seemingly at the same time.
- For Example:



Two threads, two processors, executing two threads
Source: <http://www.bravenewcode.com/concurrency/>



Notes:

Needed Animations: 1) Two threads (wavy lines), different colors, in a box labelled "READY." One thread moves to a box labelled "CPU/RUNNING." Stays in the CPU box for a short time, then a red light labelled "INTERRUPT" appears next to CPU box. Thread moves from CPU box to READY box, short delay, INTERRUPT light goes out, then the other thread is moved from ready box to CPU box.

1.6 Critical Sections

Critical Sections

NYU TANDON
ONLINE

- Sometimes code will be written with the expectation that once we enter a particular piece of code, it will run through to completion without being interrupted.
- If the code IS interrupted, synchronization can occur.
- In worst-case scenario, data could become corrupt.
- The programmer must identify a "Critical Section" of code which, once entered, must prohibit any other thread from entering a critical section on the same resource.
- Critical sections should be as small as possible.



1.7 Supplier/Demander explanation

Supplier/Demander Explanation

NYU TANDON
ONLINE



the bufferCount can easily go higher than 500.



1.8 Steps to producing a problem -1

NYU TANDON
ONLINE

Steps to Producing a Problem

```
int bufferCount;
char buffer[500];

void supplyThread(char input) //This is thread #1
{
    bool done=false;
    while (!done)
    {
        if (bufferCount==500)
        {
            bufferCount++;
            buffer[bufferCount-1]=input;
            done=true;
        }
        else
        {
            sleep(500);
            done=false;
        }
    }
}
```

bufferCount

At this point in the code, the bufferCount IS 500. Thread 2 has already been created, and is currently waiting for the condition bufferCount==500 to become true. Why? We only have storage for 500 items in the buffer. Thread 2 puts its input into the buffer and, appropriately increments the bufferCount. Thread 2 is finished and ends.

1.9 Double Update/Missing Update

NYU TANDON
ONLINE

Exploration Update/Missing Update

- Two transactions are happening on different processors at the exact same moment in time:
`void Deposit(double amount)`

- The balance starts out at \$100
 - The first transaction is a deposit of \$50
 - The second transaction is a withdraw of \$100
- $\$100 + \$50 - \$100 = \50 , lets see if that's true

```
void Withdraw(double amount)
{
    double newbalance = balance-amount; //2
    balance=newbalance //4
}
```

A medium shot of a man from the waist up. He is wearing a light blue button-down shirt over a white t-shirt. He has short, light-colored hair and is looking directly at the camera with a neutral expression. The background is plain white.

1.10 Double Update/Missing Update -1

NYU TANDON
ONLINE

Double Update/Missing Update

```
double balance;
double balance;
```

0

```
void Deposit(double amount)
```

```
void Withdraw(double amount)
```

```
{  
    double newBalance = balance + amount / 1;  
    double newBalance = balance - amount / 1;  
    balance = newBalance / 3;  
    balance = newBalance;  
}
```

}

```
void Withdraw(double amount)
```

```
void Withdraw(double amount)
```

```
{  
    double newBalance = balance - amount / 2;  
    double newBalance = balance - amount / 2;  
    balance = newBalance / 4;  
    balance = newBalance / 4;  
}
```

}

Program Counter

Program Counter

amount	newbalance
50	150

INTERRUPT

Program Counter

Program Counter

amount	newbalance
100	0

Now that we've dealt with the interrupt, it's time to go back and finish thread 1. So the balance This thread has deposited 50 and withdrawn 100, but since the interrupt happened, the balance is still well.

1.11 Critical sections identified

Critical Sections Identified

NYU TANDON
ONLINE

- In the Supplier/demander example, the checking and changing of the buffer and bufferCount must be “atomic.”
- In the Double update/missing update problem, the entire function should be atomic.
- The need for these sections of code to be run atomically, indicates that they are critical sections.

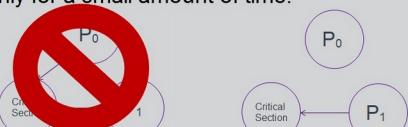


1.12 Mutual Exclusion rules

Mutual Exclusion Rules

NYU TANDON
ONLINE

- No two threads may be in a critical section at the same time.
- When no thread is in a critical section any threads that requests entry must be allowed in without delay.
- A thread may remain inside a critical section only for a small amount of time.



1.13 Knowledge Check

(Multiple Choice, 10 points, 4 attempts permitted)

Knowledge Check



Which of the following is NOT a Mutual Exclusion rule?

- At most 2 threads may be in a critical section at the same time
- If no thread is in a critical section, any thread that requests entry must be allowed in without delay
- A thread may remain inside a critical section for only a small period of time

Correct	Choice	Feedback
X	At most 2 threads may be in a critical section at the same time	That's right! In reality, only one thread may be in a critical section at any given time
	If no thread is in a critical section, any thread that requests entry must be allowed in without delay	Any delay for a thread that requests entry could cause delays to other threads!
	A thread may remain inside a critical section for only a small period of time	A thread that stays inside the critical section for too long is depriving other threads of resources!

Incorrect (Slide Layer)

Knowledge Check

NYU TANDON
ONLINE

Which of the following is NOT a Mutual Exclusion rule?

At most 2 threads may be in a critical section at the same time

If no thread is allowed in a critical section, then another must be

A thread must leave a critical section after a fixed period of time

Correct

That's right! In reality, only one thread may be in a critical section at any given time

Continue

Incorrect (Slide Layer)

Knowledge Check

NYU TANDON
ONLINE

Which of the following is NOT a Mutual Exclusion rule?

At most 2 threads may be in a critical section at the same time

If no thread is allowed in a critical section, then another must be

A thread must leave a critical section after a fixed period of time

Incorrect

Any delay for a thread that requests entry could cause delays to other threads!

Continue

Incorrect (Slide Layer)

Knowledge Check

NYU TANDON
ONLINE

Which of the following is NOT a Mutual Exclusion rule?

At most 2 threads may be in a critical section at the same time

If no thread is allowed in a critical section, then another must be

A thread must leave a critical section after a fixed period of time

Incorrect

A thread that stays inside the critical section for too long is depriving other threads of resources!

Continue

1.14 Fundamental mutual exclusion

Fundamental Mutual Exclusion

NYU TANDON ONLINE

- The system bus provides a fundamental way to provide mutual exclusion.
- Memory cannot be accessed by two processors at the same time - one will win and the other will have to wait until the next cycle.

1.15 Software Solutions for Mutual Exclusion

Peterson's Algorithm for Mutual Exclusion

NYU TANDON ONLINE

- We have better solutions today but originally, Peterson's algorithm provided a fundamental way to protect two threads from accessing the same resource at the same time.
- Peterson provided each thread with a Boolean flag to indicate that thread wanted access to its critical section.
- Additionally, there was a turn variable to overcome the problem of checking/changing the flag being a critical section itself.
- When a thread wants to enter a critical section, it raises its flag and offers the turn to the other thread.
- If both threads want access, the turn variable decides who wins

Notes:

1.16 Hardware Solutions

Hardware Solutions



NYU TANDON
ONLINE

Disable Interrupts

Test and Set

Exchange

Disable Interrupts (Slide Layer)

Hardware Solutions



NYU TANDON
ONLINE

Disable Interrupts

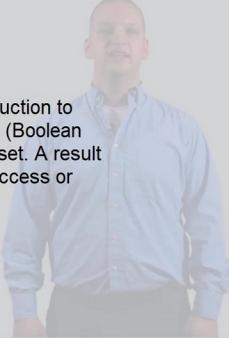
- Prevent the CPU from being interrupted and you prevent asynchrony
- Is this a good idea?

Test and Set

Exchange

Test and Set (Slide Layer)

Hardware Solutions



NYU TANDON
ONLINE

Disable Interrupts

Test and Set

- Provide a single ML instruction to check a memory location (Boolean flag) and set it if it is not set. A result is returned to indicate success or failure.

Exchange

Exchange (Slide Layer)

Hardware Solutions

NYU TANDON
ONLINE

- Disable Interrupts
- Test and Set
- Exchange
 - The location in memory (maybe a flag) is swapped with a register.



1.17 Semaphores

Semaphores

NYU TANDON
ONLINE

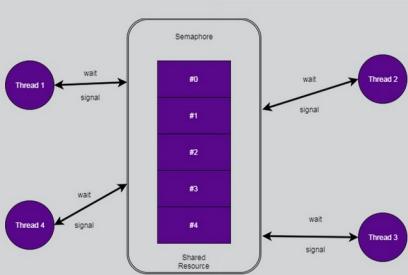
- Semaphores are a common solution programmers use to protect critical sections.
- The OS provides some of the service, but most work is done in user space
- A semaphore is a structure which fundamental communications data structure
 - Primarily two functions are used, signal and wait.
 - Signals can be queued, or if a thread is waiting, a signal will cause it to be released
 - Wait causes the thread to block if there are no signals to be consumed, the thread is queued to await the next signal.



1.18 Semaphores - How to use them

Semaphores - How To Use Them

NYU TANDON
ONLINE



1.19 Semaphore internals

Semaphore Internals

NYU TANDON
ONLINE

- Internally semaphores usually keep some counter of the number of signals which have been sent.
- Wait causes the counter to decrement.
- If the counter ever becomes negative, the thread which caused it to go negative, and all subsequent threads, will be blocked and placed on a queue. This is where the OS is invoked, to perform the block.
- When a signal is sent, the counter is incremented and if the counter is not positive, the next thread on the queue is released.
- Since the act of checking and modifying the counter is a critical section, the semaphore usually uses a hardware solution to prevent asynchrony in itself.

1.20 Deadlocks

Deadlocks

NYU TANDON
ONLINE

- If a set of threads are all waiting for each other, then there is no way that any one of them will complete.
- Usually this results from one thread waiting for another thread to release a resource.
- This is a permanent block which cannot resolve itself over time.
- There is no efficient solution today.

1.21 A real deadlock

A Real Deadlock

NYU TANDON
ONLINE

The diagram illustrates a real-world deadlock using a parking lot as an analogy. Two cars, labeled 'Process 1' and 'Process 2', are shown. Process 1 is parked in a space that overlaps with the entrance of another space. Process 2 is parked in a space that overlaps with the exit of Process 1's space. Both cars are facing towards the center of the intersection, effectively blocking each other's path. This visualizes how two processes can wait indefinitely for resources held by the other, similar to a software deadlock between two threads waiting for the same resources.

Source: <http://csunplugged.org/wp-content/uploads/2015/03/deadlock.jpg> 1286488735

Notes:

Source: <http://csunplugged.org/wp-content/uploads/2015/03/deadlock.jpg> 1286488735

1.22 Deadlock resource types

Deadlock Resource Types

NYU TANDON
ONLINE

- Reusable
 - Once the use of the resource is complete, the resource can be used again by another thread.
 - Examples include memory, devices, data structures, semaphores, etc
- Consumable
 - Once used, the resource is gone
 - Examples include interrupts, signals, messages and data in IO Buffers



1.23 Items required for a deadlock

Items Required For a Deadlock

NYU TANDON
ONLINE

- Mutual Exclusion
- Hold-and-wait
- No Preemption
- Circular Wait



Mutual Exclusion (Slide Layer)

Items Required For a Deadlock

NYU TANDON
ONLINE

Mutual Exclusion	• Only one thread can use a resource at any given time
Hold-and-wait	
No Preemption	
Circular Wait	



Hold And Wait (Slide Layer)

Items Required For a Deadlock

NYU TANDON
ONLINE

Mutual Exclusion	
Hold-and-wait	• While a thread is "waiting" on the allocation of a new resource, it retains all existing locks
No Preemption	
Circular Wait	

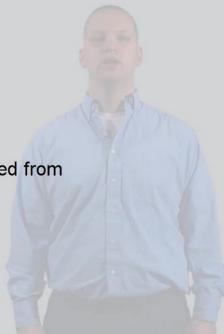


No Preemption (Slide Layer)

Items Required For a Deadlock

NYU TANDON
ONLINE

Mutual Exclusion	
Hold-and-wait	
No Preemption	• A resource cannot be removed from a thread forcefully
Circular Wait	



Circular Wait (Slide Layer)

Items Required For a Deadlock

NYU TANDON
ONLINE

Mutual Exclusion	
Hold-and-wait	
No Preemption	
Circular Wait	<ul style="list-style-type: none">• A closed chain of threads in which the last thread is waiting on the list



1.24 Knowledge Check

(Multiple Choice, 10 points, 4 attempts permitted)

Knowledge Check

NYU TANDON
ONLINE

Which of the following is NOT a requirement for a deadlock?

Mutual Exclusion
 Hold and Wait
 Preemption
 Circular Wait

Correct	Choice	Feedback
	Mutual Exclusion	In this case, only one thread can use a resource at any given time. This is one of the requirements for a deadlock.
	Hold and Wait	This is where a thread is holding all existing locks, while waiting for allocation of a new resource. This

		contributes to a deadlock.
X	Preemption	Correct!
	Circular Wait	This is where in a closed chain of threads, the last thread is waiting for the first thread to finish. This contributes to a deadlock.

Incorrect (Slide Layer)

Knowledge Check 

Which of the following is NOT a requirement for a deadlock?

Mutual Exclusion
 Hold and Wait
 Preemption
 Circular Wait

Incorrect

In this case, only one thread can use a resource at any given time. This is one of the requirements for a deadlock.

Continue

Incorrect (Slide Layer)

Knowledge Check 

Which of the following is NOT a requirement for a deadlock?

Mutual Exclusion
 Hold and Wait
 Preemption
 Circular Wait

Incorrect

This is where a thread is holding all existing locks, while waiting for allocation of a new resource. This contributes to a deadlock.

Continue

Correct (Slide Layer)

Knowledge Check

NYU TANDON
ONLINE

Which of the following is NOT a requirement for a deadlock?

Mutual Exclusion

Hold and Wait

Preemption

Circular Wait

Correct

Correct!

Continue

Incorrect (Slide Layer)

Knowledge Check

NYU TANDON
ONLINE

Which of the following is NOT a requirement for a deadlock?

Mutual Exclusion

Hold and Wait

Preemption

Circular Wait

Incorrect

This is where in a closed chain of threads, the last thread is waiting for the first thread to finish. This contributes to a deadlock.

Continue

1.25 Solutions for deadlocks

Solutions For Deadlocks

NYU TANDON
ONLINE

- Deadlock Prevention
 - Eliminate one of the four requirements
- Deadlock Avoidance
 - Before each allocation check to make sure a deadlock is not possible, do not make an allocation that could cause a deadlock
- Deadlock Detection
 - Allow deadlocks to happen and then resolve it when it does

Mutual Exclusion	Hold and Wait
Circular Wait	No Preemption



1.26 Deadlock Prevention

Deadlock Prevention

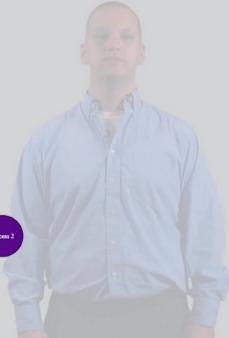
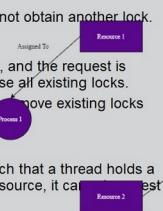
- Eliminating one of the four
- Mutual Exclusion
 - This is not often possible as mutual exclusion is usually required.
- Hold and wait
 - The system can require that all locks be requested simultaneously.
 - If a thread holds a lock, it cannot obtain another lock.
- No preemption
 - If a thread request a new lock, and the request is denied, the thread must release all existing locks.
 - The OS may have authority to remove existing locks individually.
- Circular wait 
 - The resources are ordered such that a thread holds a lock on a higher numbered resource, it cannot request a lower numbered resource.



Untitled Layer 1 (Slide Layer)

Deadlock Prevention

- Eliminating one of the four
- Mutual Exclusion
 - This is not often possible as mutual exclusion is usually required.
- Hold and wait
 - The system can require that all locks be requested simultaneously.
 - If a thread holds a lock, it cannot obtain another lock.
- No preemption
 - If a thread request a new lock, and the request is denied, the thread must release all existing locks.
 - The OS may have authority to remove existing locks individually.
- Circular wait 
 - The resources are ordered such that a thread holds a lock on a higher numbered resource, it cannot request a lower numbered resource.



1.27 Deadlock Avoidance

Deadlock Avoidance

- Required that the Operating System know, ahead of time, which resources a thread will request before it is finished
- Every time a thread makes a new request, the OS runs an algorithm to see if that request will cause the system to deadlock.
 - If it will, the request is denied or queued
- This usually uses a “Banker’s algorithm.”



1.28 Deadlock detection

Deadlock Detection

NYU TANDON
ONLINE

- This is not at all restrictive
- Recognize that deadlocks will occur and that a detection algorithm will need to be periodically run
- Once a deadlock is detected, the OS should take action to resolve the deadlock
- Solutions to existing deadlocks
 - Rollback to a previously unlocked state (requires transaction logs)
 - Abort all deadlocked threads
 - Abort a thread, check for deadlock, repeat
 - Preempt resources until the deadlock is resolved



1.29 Dining Philosophers problem.

Dining Philosophers Problem.

NYU TANDON
ONLINE

- Presented by Dr. Edsger Dijkstra as a concurrency control problem
- In a house live 5 philosophers numbered 0 thru 4. They alternate between thinking and eating. When it is time to eat, a circular table is set for them with each philosopher having a setting consisting of one plate and one fork. The meal they will eat is particularly difficult to eat kind of, spaghetti. To eat this meal, they must have two forks. There are two forks next to each plate, so that presents no difficulty, as a consequence, however, no two neighbors may be eating simultaneously.



Source: https://spin.atomicobject.com/wp-content/uploads/dining-philosophers.jpg

Notes:

Source: <https://spin.atomicobject.com/wp-content/uploads/dining-philosophers.jpg>

1.30 The dining deadlock

The Dining Deadlock

NYU TANDON
ONLINE

- If all philosophers are allowed to take a fork, all will have one fork and none will ever eat
- Dijkstra proposed resource ordering the forks, so that the last philosopher had to request the zero numbered fork before he could request the fork numbered 4.
- Another solution is to have a semaphore to prevent the fifth philosopher from entering the room.
 - A semaphore is constructed with 4 signals queued, and the philosopher must request access to the room before requesting access to any fork.



1.31 Knowledge Check

(Multiple Response, 10 points, 4 attempts permitted)

Knowledge Check

NYU TANDON
ONLINE

Once a deadlock has occurred, what are some solutions available?
(Select all that apply)

Rollback to a previously unlocked state (requires transaction logs)
 Abort all deadlocked threads
 Abort a thread, check for deadlock, repeat
 Ignore deadlocked threads, and abort any new threads
 Abort the most recent thread that caused the deadlock

Correct	Choice
X	Rollback to a previously unlocked state (requires transaction logs)
X	Abort all deadlocked threads
X	Abort a thread, check for deadlock, repeat
	Ignore deadlocked threads, and abort any new threads
	Abort the most recent thread that caused the deadlock

Feedback when correct:

That's right! You selected the correct response.

Feedback when incorrect:

You did not select the correct response.

Correct (Slide Layer)

Knowledge Check

Once a deadlock has occurred, what are some solutions available?
(Select all that apply)

Rollback to Correct
 Abort all deadlocked threads
 Abort a thread
 Ignore deadlocked threads, and abort any new threads
 Abort the most recent thread that caused the deadlock

That's right! You selected the correct response.

Continue

NYU TANDON
ONLINE

Incorrect (Slide Layer)

Knowledge Check

Once a deadlock has occurred, what are some solutions available?
(Select all that apply)

Rollback to Incorrect
 Abort all deadlocked threads
 Abort a thread
 Ignore deadlocked threads, and abort any new threads
 Abort the most recent thread that caused the deadlock

You did not select the correct response.

Continue

NYU TANDON
ONLINE

Try Again (Slide Layer)

Knowledge Check

Once a deadlock has occurred, what are some solutions available?
(Select all that apply)

Rollback to _____
 Abort all deadlocked threads
 Abort a thread
 Ignore deadlocked threads, and abort any new threads
 Abort the most recent thread that caused the deadlock

Incorrect
That is incorrect. Please try again.

Try Again

1.32 Concurrency and Deadlocks - Conclusion

Conclusion

NYU TANDON
ONLINE

- Reminder about threads
- Features of having multiple threads
- Possible ways to have asynchrony
- Critical Sections
- Examples of concurrency issues
- Mutual Exclusion
- Software solutions for Mutual exclusion
- Hardware Options
- Semaphores
- Deadlocks
- Solutions to deadlocks
- Dining Philosopher's problem



1.33 Results Slide

(Results Slide, 0 points, 1 attempt permitted)

Results

NYU TANDON
ONLINE

Your Score: %Results.ScorePercent%% (%Results.ScorePoints% points)

Passing Score: %Results.PassPercent%% (%Results.PassPoints% points)

Result:

[Retry Quiz](#)

[Review Quiz](#)

Results for
1.13 Knowledge Check
1.24 Knowledge Check
1.31 Knowledge Check

Result slide properties

Passing 80%

Score

Success (Slide Layer)

Results NYU TANDON
ONLINE

Your Score: %Results.ScorePercent%% (%Results.ScorePoints% points)

Passing Score: %Results.PassPercent%% (%Results.PassPoints% points)

Result:

✓ Congratulations, you passed.

[Retry Quiz](#)
[Review Quiz](#)

Failure (Slide Layer)

Results NYU TANDON
ONLINE

Your Score: %Results.ScorePercent%% (%Results.ScorePoints% points)

Passing Score: %Results.PassPercent%% (%Results.PassPoints% points)

Result:

✗ You did not pass.

[Retry Quiz](#)
[Review Quiz](#)

1.34 End of Module

NYU TANDON
ONLINE

End of Module

[Exit](#)

