Concurrency

Week 5

Semaphores
Deadlocks

Topics

- Managing Critical Section
  - Mutex (discussed last week)
  - Semaphores
- Deadlocks
  - Definition
  - Deadlocks with Mutex
  - Deadlocks With Semaphores
- Practical Exercises

Learning Outcomes

- At the end of this lesson, you will be able to:

1. Describe the concept of a Semaphore and how it can provide safe access to shared resources.
2. Explain what deadlocks are and how they can be prevented
3. Apply deadlock-free protection of shared resources using semaphores.

What is the Main Idea of the Course?

- The following four questions summarize this course.
- Questions:
  1. ***How can we run multiple tasks simultaneously***? We discuss multitasking and multi-threading to answer this question.
  2. ***How can simultaneous tasks share resources***? We discuss resource types and how they can be shared.
  3. ***What problems may we have when we share resources***? We discuss race conditions and the inconsistency of data to answer this question.
  4. ***What solution do we have for these problems***? We discuss semaphores, and mutex locks to answer this question.

Part One!

Part One!

- In this part, we will discuss semaphores and how they can be used to avoid race conditions. Try to find answers to the following questions:

1. What operations are allowed on a semaphore?
2. How can a semaphore guarantee mutual exclusion in a critical section?
3. Do solutions for race conditions using semaphores avoid starvation?
4. How are counting semaphores used in message-passing systems?
5. How do you compare semaphore with mutex?

Semaphores

Semaphore **P()**

Semaphore: an integer combined with P and V operations

- Call P to enter a critical section
  - from Dutch:
    - P→ Proberen, or Passeren, or Pakken.
- It is also known as wait operation

- **If semaphore value > 0, it can be *decremented* by one**
  - Atomic decrement
- If semaphore value == 0, calling P will cause the process to wait (blocks the process/thread)

Semaphore V()

Semaphore: an integer combined with P and V operations

- Call V to exit a critical section
  - from Dutch:
    - V→ Verhogen or Vrijgave
- It is also known as signal operation

- **Increments** the value of the semaphore by one
- Allows the process that is waiting on V to continue

Semaphore

Semaphore: an **integer** combined with P and V operations

- For mutual exclusions problems, semaphore values are initialized to 1. These semaphores can get only two values: 0 and 1, and therefore are called binary semaphores

```
Process P
// some code
//  s is a semaphore
P(s);
    // critical section
V(s);
// remainder section
```

```
Process Q
// some code
//  s is a semaphore
P(s);
    // critical section
V(s);
// remainder section
```

Example 1: Bridge

Remember the Bridge problem from the previous week?

- Assume two or more observers are counting events using a shared record named Total.
- Observers use semaphores to avoid race conditions
- Global *Semaphore S* = 1

```
Process Observer

if Observed():
    P(S);
        temp = Read(Total)
        temp = temp + 1
        Store(temp , Total)
    V(S);
```

Example 2: Travel Agency

- A sales agent who is making a reservation for a flight should
  - Decrement the value of a semaphore (P)
  - Read the reservation list
  - Update the list
  - Increment the value of the semaphore (V)

- What happens if the sales agent forgets to complete the reservation (stays inside its critical section)? Discuss.

Example 3: Printer Spool

- To add an item to the print spool a process uses the following code:
- Global *Semaphore S* initialized to 1

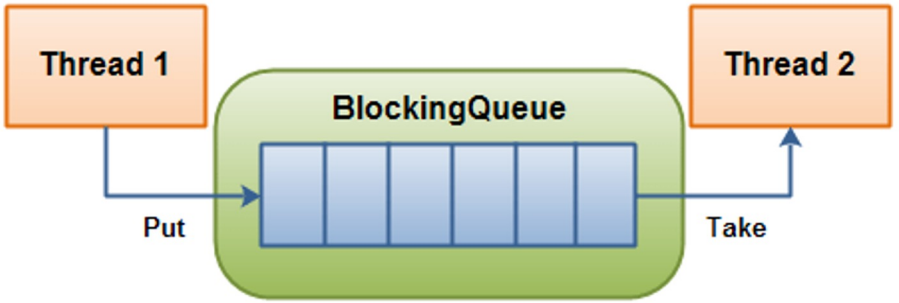
```
Data = Prepare()

P(S);
   Spool[in] = Data
   in = in + 1
V(s);
```

Producer-Consumer

Remember: Producers and Consumers share a buffer …

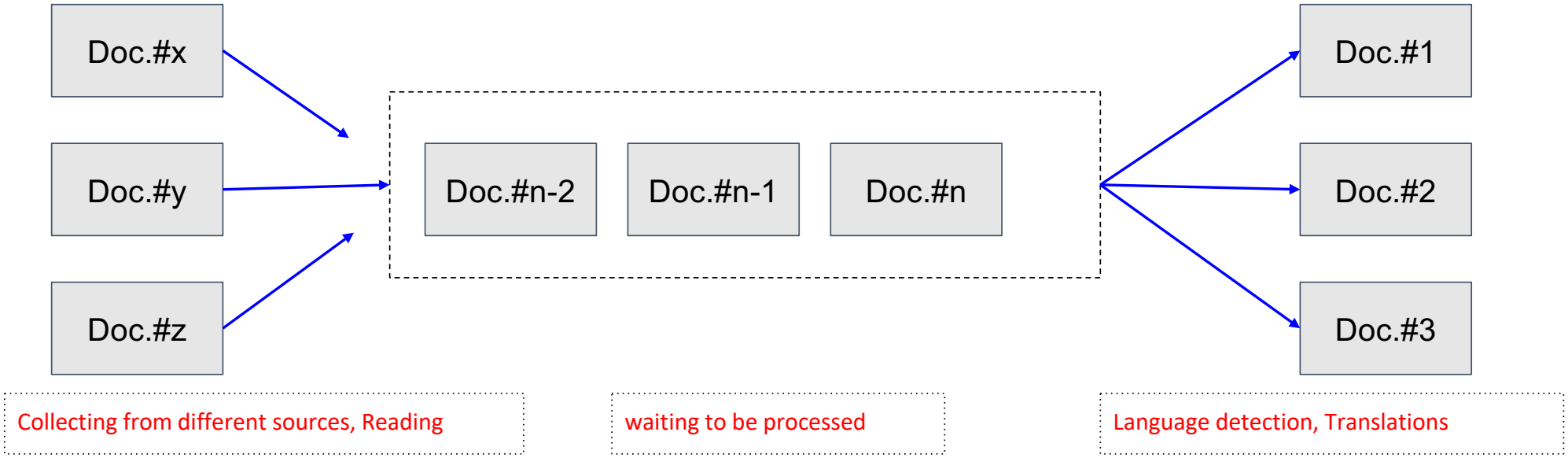
- How can Producers and Consumers inform each other about the state of the buffer?
- Consumer prefers to pick the item when it is ready, otherwise iteratively it has to check the buffer



Producer-Consumer: Example

Translating documents:

- Translator threads need to be informed when a document is ready for translation.



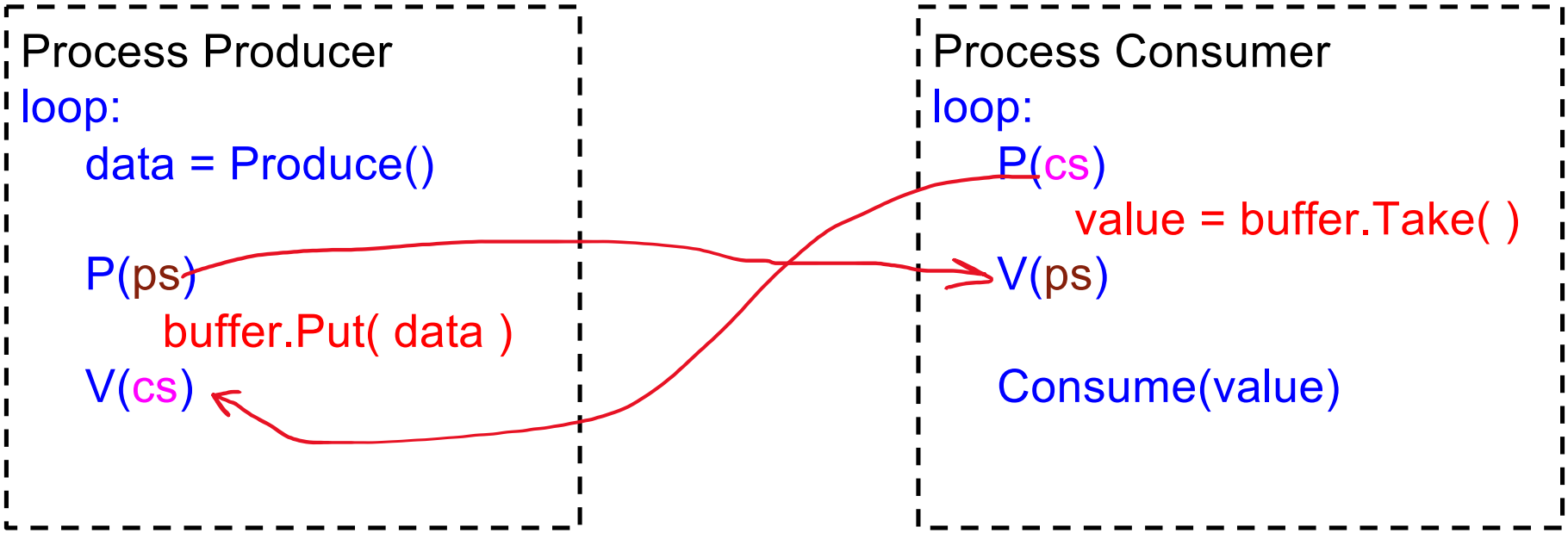
Producer-Consumer

See how semaphores can help us …

- Assume a shared buffer with only one space
- Define two semaphores: *p*s and *c*s
- Initialize ps=1 and cs=0

Producer-Consumer

Assume a shared buffer with only one space, we have two semaphores: ps and cs, Initializations are ps=1 and cs=0 ...



Producer-Consumer

Assume a shared buffer with only one space, we have two semaphores: ps and cs, Initializations are ps=1 and cs=0 ...

Process Producer

```
loop:
    data = Produce()

    P(ps)
        buffer.Put( data )
    V(cs)
```

Scenarios:
Start: buffer is empty,
if the consumer is faster, it has to wait in P(cs). **Why?**

Process Consumer

```
loop:
    P(cs)
        value = buffer.Take( )
    V(ps)

    Consume(value)
```

Producer-Consumer

Assume a shared buffer with only one space, we have two semaphores: ps and cs, Initializations are ps=1 and cs=0 ...

Process Producer

```
loop:
    data = Produce()

    P(ps)
        buffer.Put( data )
    V(cs)
```

Scenarios:
buffer is empty,
if the consumer is faster, it has to wait in P(cs). Why?

There will be a point where a producer will produce data and ...

Process Consumer

```
loop:
    P(cs)
        value = buffer.Take( )
    V(ps)

    Consume(value)
```

Producer-Consumer

Assume a shared buffer with only one space, we have two semaphores: ps and cs, Initializations are ps=1 and cs=0 ...

Process Producer

```
loop:
    data = Produce()

    P(ps)
        buffer.Put( data )
    V(cs)
```

Scenarios:
buffer is empty,
if the consumer is faster, it has to wait in P(cs). Why?

… it will be the first one to enter the critical section.
Why?

Process Consumer

```
loop:
    P(cs)
        value = buffer.Take( )
    V(ps)

    Consume(value)
```

Producer-Consumer

Assume a shared buffer with only one space, we have two semaphores: ps and cs, Initializations are ps=1 and cs=0 ...

Process Producer

```
loop:
    data = Produce()

    P(ps)
        buffer.Put( data )
    V(cs)
```

Scenarios:
buffer is empty,
if the consumer is faster, it has to wait in P(cs). Why?

… when there is one producer in the critical section, other producers and all the consumers have to wait. **Why?**

Process Consumer

```
loop:
    P(cs)
        value = buffer.Take( )
    V(ps)

    Consume(value)
```

Producer-Consumer

Assume a shared buffer with only one space, we have two semaphores: ps and cs, Initializations are ps=1 and cs=0 ...

Process Producer

```
loop:
    data = Produce()

    P(ps)
        buffer.Put( data )
    V(cs)
```

… when there is one producer in the critical section, other producers and all the consumers have to wait. **Why?**
The producer finishes Put(). It executes V(cs). The waiting consumer will enter the critical section. **Why?**

Process Consumer

```
loop:
    P(cs)
        value = buffer.Take( )
    V(ps)

    Consume(value)
```

Producer-Consumer

Assume a shared buffer with only one space, we have two semaphores: ps and cs, Initializations are ps=1 and cs=0 ...

Process Producer

```
loop:
    data = Produce()

    P(ps)
        buffer.Put( data )
    V(cs)
```

… when there is one consumer in the critical section, other consumers and all the producers have to wait. **Why?**
The consumer finishes Take(). It executes V(ps). The waiting producer can enter the critical section. **Why?**

Process Consumer

```
loop:
    P(cs)
        value = buffer.Take( )
    V(ps)

    Consume(value)
```

Producer-Consumer

Assume a shared buffer with only one space, we have two semaphores: ps and cs, Initializations are ps=1 and cs=0 ...

Process Producer

```
loop:
    data = Produce()

    P(ps)
        buffer.Put( data )
    V(cs)
```

Interested to implement?
Check the exercises ...

Process Consumer

```
loop:
    P(cs)
        value = buffer.Take( )
    V(ps)

    Consume(value)
```

Producer-Consumer: Extended

We can extend the definition of semaphores to include all positive integers. These semaphores are called counting semaphores.

- Assume a shared buffer with ***n*** empty space.
- Define two semaphores: ***ps*** and ***cs****.*
- Initialize ***ps***=***n*** and ***cs***=0.
- ***ps*** counts the number of free slots in the buffer, and ***cs*** counts the number of data items stored in the buffer.

Producer-Consumer: Extended

- In this example, ps counts the number of free slots in the buffer and cs counts the number of data items inserted into the buffer.

Process Producer

```
loop:
    data = Produce()

    P(ps)
        P(lock)
            buffer.Put( data )
        V(lock)
    V(cs)
```

Here we need another semaphore to provide mutual exclusion in accessing the buffer.

What should be the initial value of the lock?

Process Consumer

```
loop:
    P(cs)
        P(lock)
            value = buffer.Take( )
        V(lock)
    V(ps)

    Consume(value)
```

Part One!

- In this part, we will discuss semaphores and how they can be used to avoid race conditions. Try to find answers to the following questions:

1. What operations are allowed on a semaphore?
2. How can a semaphore guarantee mutual exclusion in a critical section?
3. Do solutions for race conditions using semaphores avoid starvation?
4. How are counting semaphores used in message-passing systems?
5. How do you compare semaphore with mutex?

Part Two!

Part Two!

- In this part, we will discuss Deadlocks. Try to find the answer to the following questions:

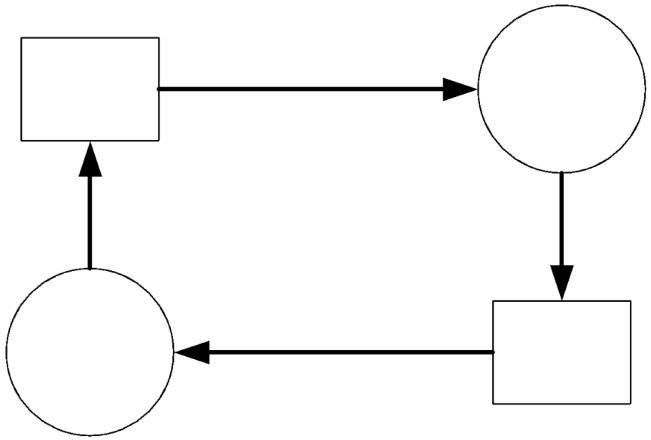
1. What are the required conditions to have a deadlock?
2. What kind of resources do not cause any deadlock?
3. What mistake in the producer-consumer example can cause a deadlock?
4. May we have a deadlock if we use a mutex instead of a semaphore in the producer-consumer example?
5. To prevent deadlocks, the resources should be allocated in a specific order. What order can we use in allocating forks to philosophers?

Deadlocks

Deadlocks

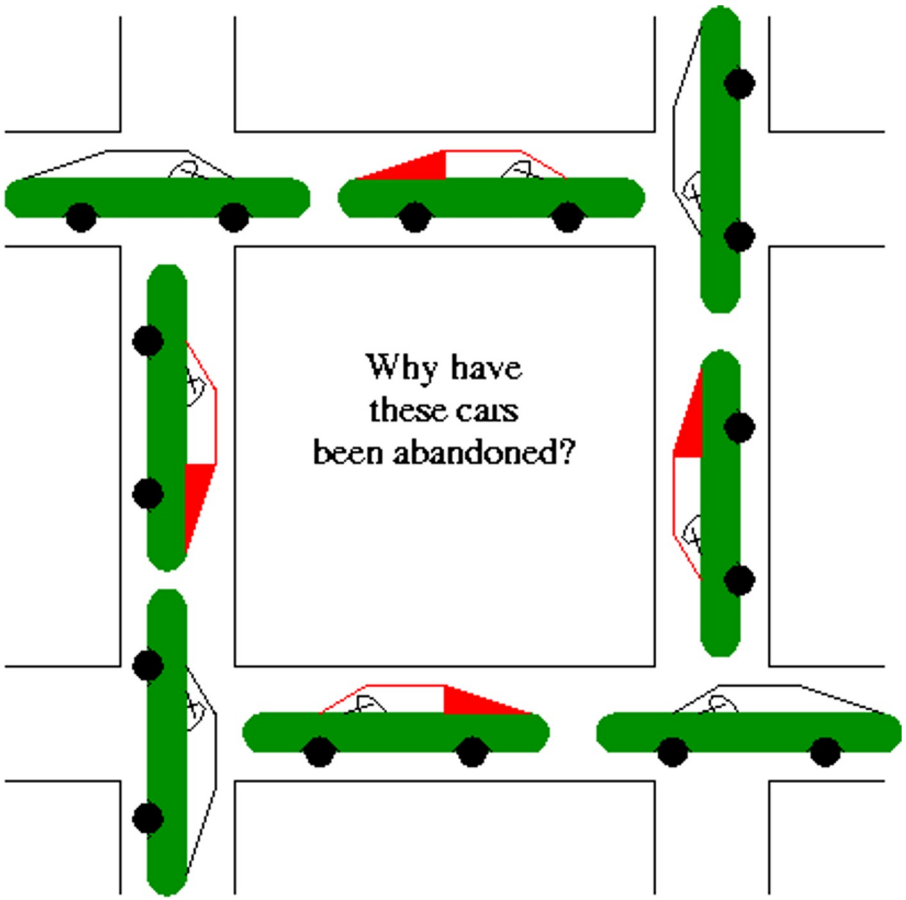
- When each process holds a resource needed by the other process, each process relies on the other process to make a progress.

- This situation when no progress is made is called a **deadlock** condition.



Example 1

- What are the resources here?
- Who is waiting for what?

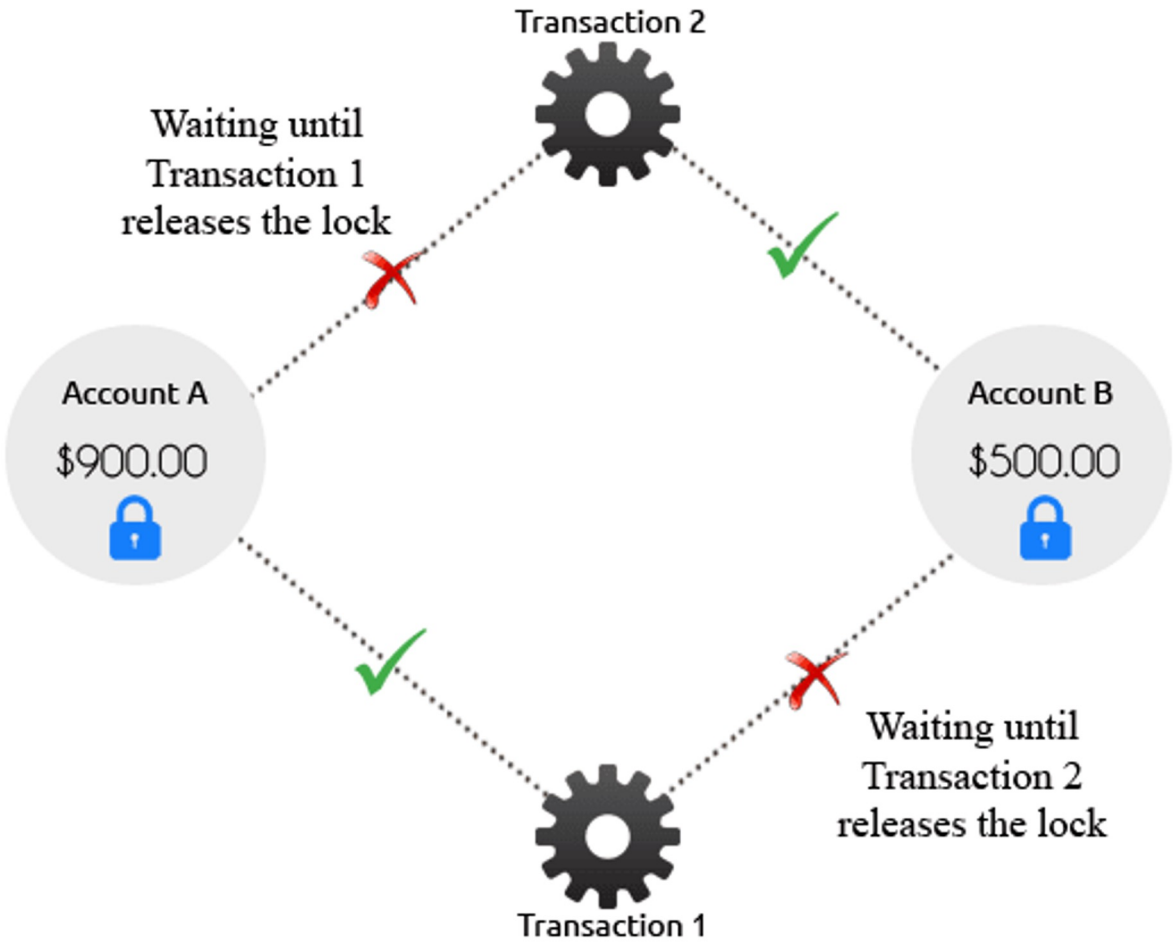


Example 2

- What are the resources here?
- Who is waiting for what?

Example 3

- What are the resources here
- Who is waiting for what?



Conditions to Have a Deadlock

- The four deadlock conditions are:
  - Exclusive resource access; no simultaneous sharing (only sequentially reusable).
  - Incremental allocation: processes do not request all resources together (hold and wait).
  - Circular waiting for resources.
  - No resource preemption.
- These are necessary conditions; if any of them are missing, a deadlock can’t occur.

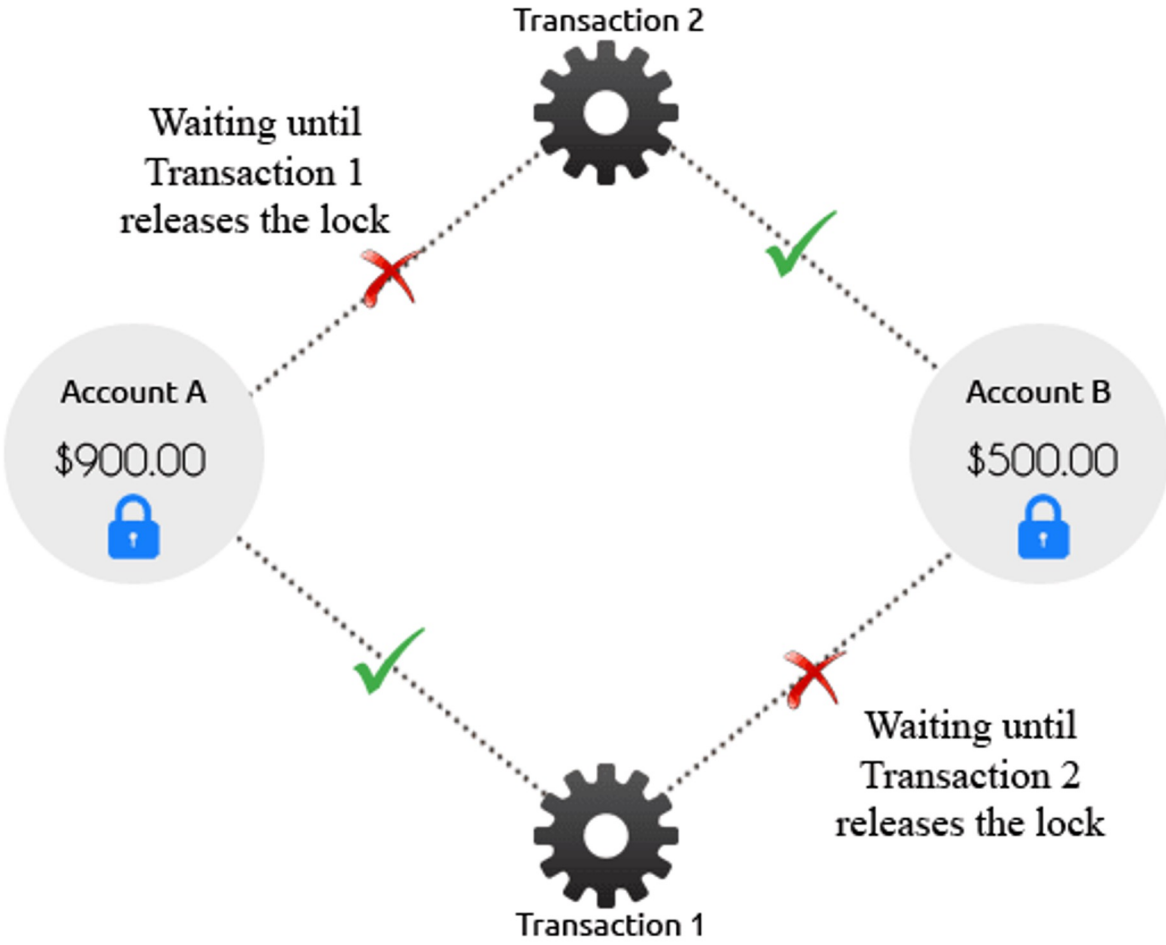
Example 2 (Revised)

- Each process holds a resource and is waiting for the next resource.

- Before allocating resources, the manager should make sure that a circular waiting will not be created. How?

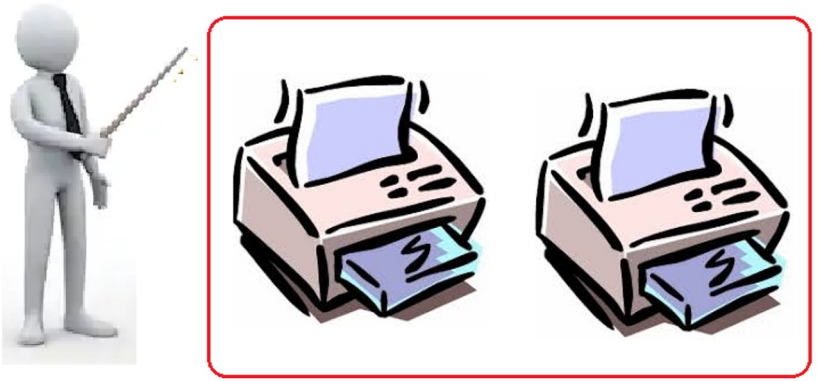
Discuss

- How can we avoid deadlocks here?



Deadlock with Semaphores

- A semaphore may be viewed as a resource counter.

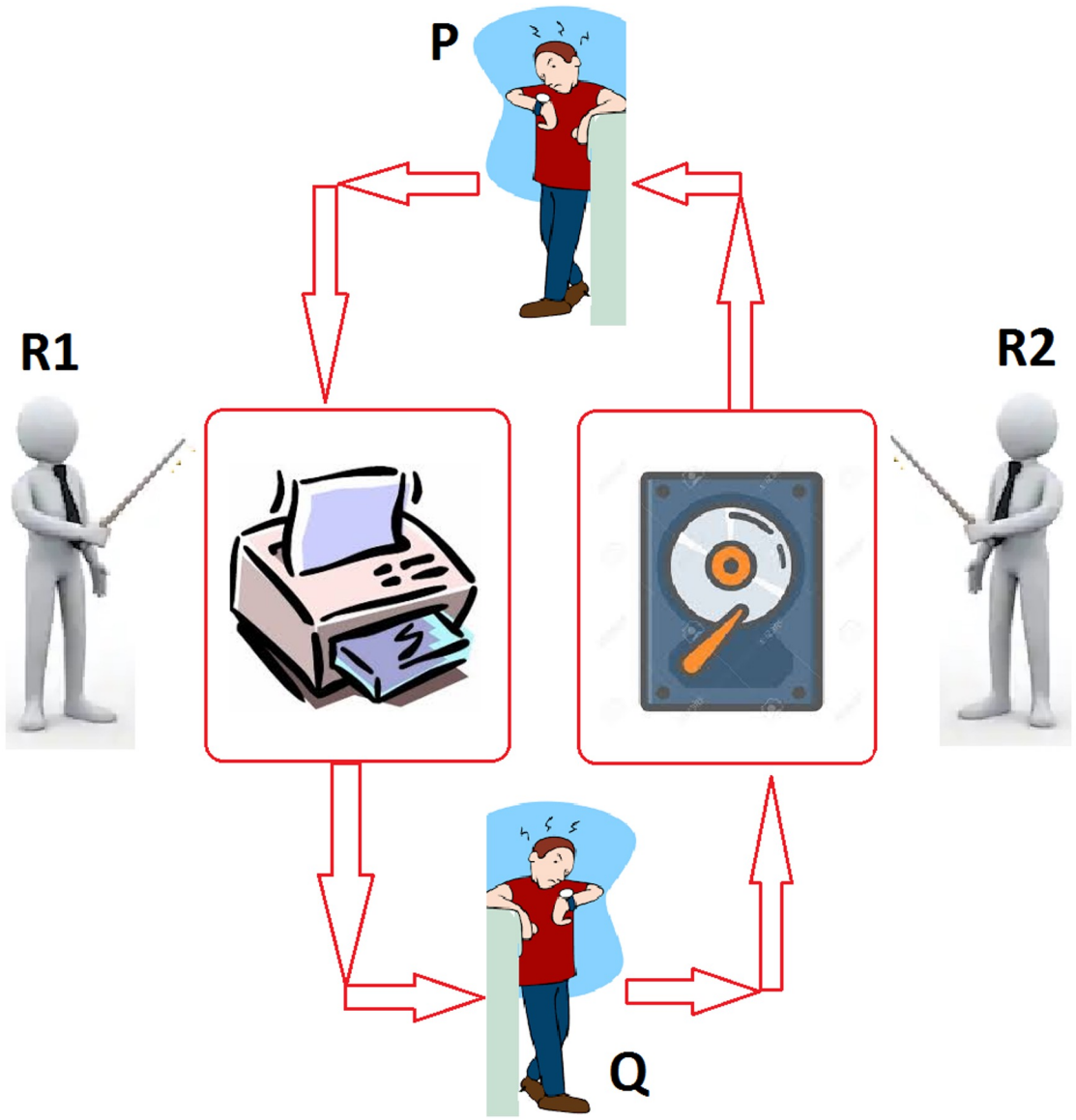


- For instance, a semaphore can be used to represent printers.

- If we have two printers, the initial value of the semaphore will be 2.

Deadlock with Semaphores

- Assume processes P and Q are going to use resources R1 and R2.
- P requests the resources in R2, R1 order
- Q requests the resources in R1, R2 order


P
R1
R2
Q

Discuss

Semaphore S1=1, S2=1

Process P	Process Q
P(S1)	P(S2)
P(S2)	P(S1)
Run_critical_section_code()	Run_critical_section_code()
V(S2)	V(S1)
V(S1)	V(S2)

It is likely to have a deadlock running the above example. Why we are not sure if a deadlock will happen or not?

Mutex and Deadlock

- Deadlocks can happen with Mutex (or Lock) as well
- A Mutex is a lock which can have one of the two states (locked/unlocked)
- Only the process that locks a Mutex can open it.
- A process trying to lock an already locked Mutex is **blocked**

Mutex and Deadlock

- Assume Mutex M is used to enforce exclusive access to a resource

- Process P locks Mutex M

- Then P tries to lock it again.

- But because M is already locked, P will wait until M is unlocked

Discuss

- How long will process P wait? Why?

- What solutions do you suggest?

Deadlock Avoidance

- If any one of the four required deadlock conditions is violated, the deadlock will not happen.
- In practice, we cannot change the type of resource.
- The only condition that the resource manager can control is allocating in a way that circular hold-and-wait queues are not created.

Dining Philosopher

Dining Philosophers is another classical problem which studies Deadlocks …

- When can a deadlock happen?
- Are you interested in implementing it? Then ..

Extra video → https://www.youtube.com/watch?v=MYgmmJJfdBg

Part Two!

- In this part, we will discuss Deadlocks. Try to find the answer to the following questions:

1. What are the required conditions to have a deadlock?
2. What kind of resources do not cause any deadlock?
3. What mistake in the producer-consumer example can cause a deadlock?
4. May we have a deadlock if we use a mutex instead of a semaphore in the producer-consumer example?
5. To prevent deadlocks, the resources should be allocated in a specific order. What order can we use in allocating forks to philosophers?

Time to Apply

Week 5: Exercisez