بسم الله الرحمن الرحیم

«سیستم عامل»

۱

جلسه ۱۱: بن‌بست (۳)

یادآوری

Resources and deadlocks

- **Processes need access to resources in order to make progress**

- **Examples of computer resources**
  - printers
  - disk drives
  - kernel data structures (scheduling queues …)
  - locks/semaphores to protect critical sections

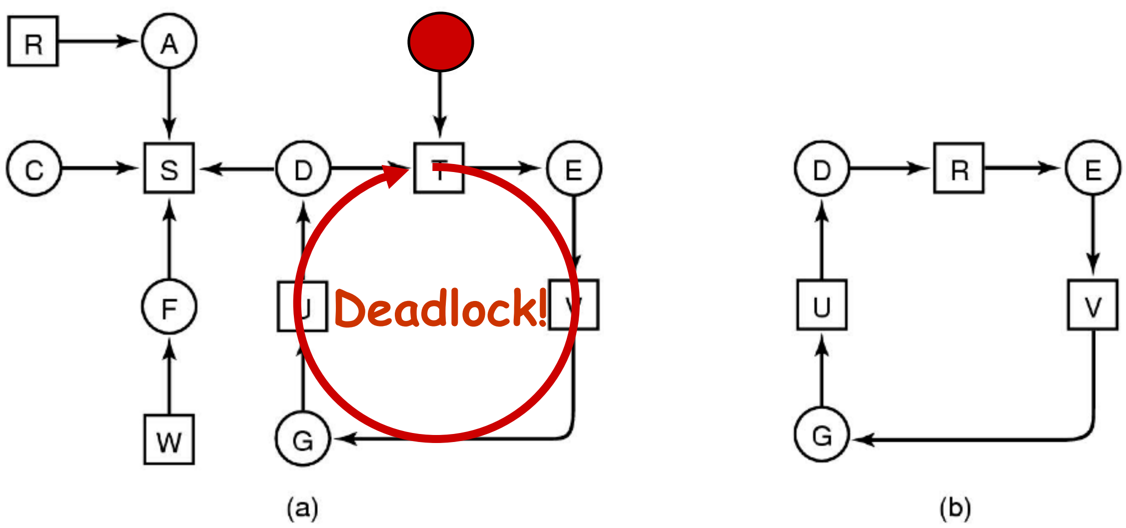
- **Suppose a process holds resource A and requests resource B**
  - at the same time another process holds B and requests A
  - both are blocked and remain so … this is deadlock

Dealing with deadlock

- **Four general strategies**
  - Ignore the problem
    - **Hmm… advantages, disadvantages?**
  - Detection and recovery
  - Dynamic avoidance via careful resource allocation
  - Prevention, by structurally negating one of the four necessary conditions

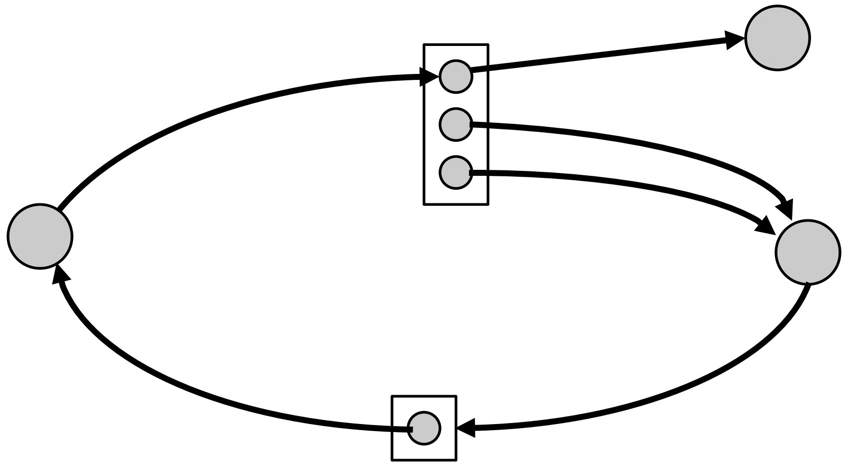
Deadlock detection (1 resource of each)

- Do a depth-first-search on the resource allocation graph

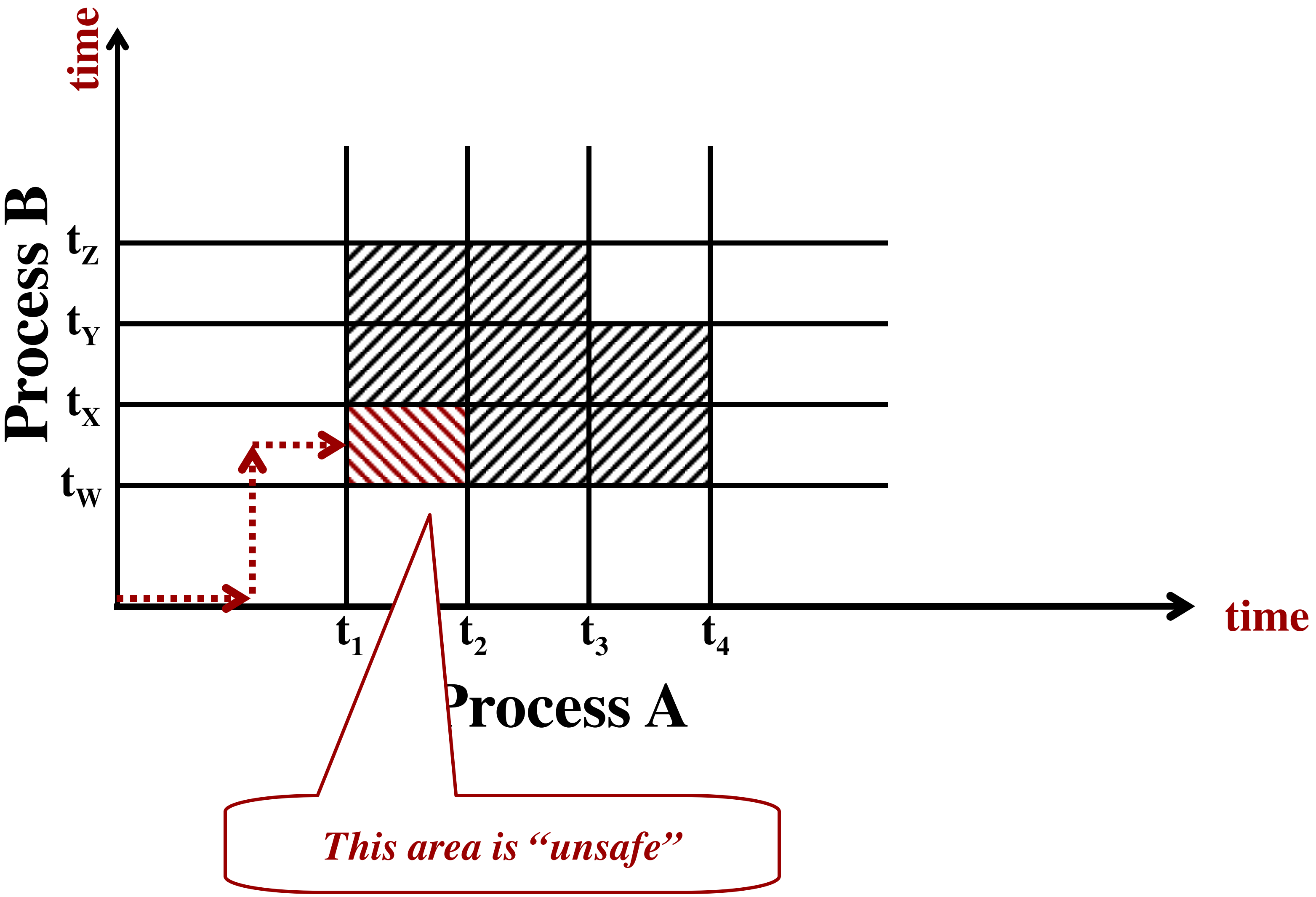


Deadlock modeling with multiple resources

- **<u>Theorem</u>: If a graph does not contain a cycle then no processes are deadlocked**
  - A cycle in a RAG is a <u>necessary</u> condition for deadlock
  - Is it a <u>sufficient</u> condition?



Avoidance using process-resource trajectories



Safe states

- **The current state:**

  "which processes hold which resources"

- **A "safe" state:**
  - No deadlock, and
  - There is some scheduling order in which every process can run to completion even if all of them request their maximum number of units immediately

- **The Banker's Algorithm:**
  - Goal: Avoid unsafe states!!!
  - When a process requests more units, should the system grant the request or make it wait?

Avoidance with multiple resources

Total resource vector

Resources in existence
$(E_1, E_2, E_3, \ldots, E_m)$

Current allocation matrix

$$\begin{bmatrix} C_{11} & C_{12} & C_{13} & \cdots & C_{1m} \\ C_{21} & C_{22} & C_{23} & \cdots & C_{2m} \\ \vdots & \vdots & \vdots & & \vdots \\ C_{n1} & C_{n2} & C_{n3} & \cdots & C_{nm} \end{bmatrix}$$

Row n is current allocation to process n

Available resource vector

Resources available
$(A_1, A_2, A_3, \ldots, A_m)$

Maximum Request Vector

$$\begin{bmatrix} R_{11} & R_{12} & R_{13} & \cdots & R_{1m} \\ R_{21} & R_{22} & R_{23} & \cdots & R_{2m} \\ \vdots & \vdots & \vdots & & \vdots \\ R_{n1} & R_{n2} & R_{n3} & \cdots & R_{nm} \end{bmatrix}$$

Row 2 is what process 2 might need

Note: These are the max. <u>possible</u> requests, <u>which we assume are known ahead of time!</u>

Banker's algorithm for multiple resources

- Look for a row, R, whose unmet resource needs are all smaller than or equal to A. If no such row exists, the system will eventually deadlock since no process can run to completion
- Assume the process of the row chosen requests all the resources that it needs (which is guaranteed to be possible) and finishes. Mark that process as terminated and add all its resources to A vector
- Repeat steps 1 and 2, until either all process are marked terminated, in which case the initial state was safe, or until deadlock occurs, in which case it was not

Avoidance with multiple resources

Total resource vector

Resources in existence
$(E_1, E_2, E_3, \ldots, E_m)$

Current allocation matrix

$$\begin{bmatrix} C_{11} & C_{12} & C_{13} & \cdots & C_{1m} \\ C_{21} & C_{22} & C_{23} & \cdots & C_{2m} \\ \vdots & \vdots & \vdots & & \vdots \\ C_{n1} & C_{n2} & C_{n3} & \cdots & C_{nm} \end{bmatrix}$$

Row n is current allocation to process n

Available resource vector

Resources available
$(A_1, A_2, A_3, \ldots, A_m)$

Maximum Request Vector

$$\begin{bmatrix} R_{11} & R_{12} & R_{13} & \cdots & R_{1m} \\ R_{21} & R_{22} & R_{23} & \cdots & R_{2m} \\ \vdots & \vdots & \vdots & & \vdots \\ R_{n1} & R_{n2} & R_{n3} & \cdots & R_{nm} \end{bmatrix}$$

Row 2 is what process 2 might need

Run algorithm on every resource request!

Avoidance with multiple resources

$$\begin{array}{cccccc} & \text{Tape drives} & \text{Plotters} & \text{Scanners} & \text{CD Roms} & \\ E = (& 4 & 2 & 3 & 1 &) \end{array}$$

$$\begin{array}{cccccc} & \text{Tape drives} & \text{Plotters} & \text{Scanners} & \text{CD Roms} & \\ A = (& 2 & 1 & 0 & 0 &) \end{array}$$

Current allocation matrix

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

Max request matrix

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix}$$

Avoidance with multiple resources

$$\begin{array}{ccccc} & \text{Tape drives} & \text{Plotters} & \text{Scanners} & \text{CD Roms} \\ E = (& 4 & 2 & 3 & 1\ ) \end{array}$$

$$\begin{array}{ccccc} & \text{Tape drives} & \text{Plotters} & \text{Scanners} & \text{CD Roms} \\ A = (& 2 & 1 & 0 & 0\ ) \end{array}$$

Current allocation matrix

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

Max request matrix

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix}$$

Avoidance with multiple resources

$$\begin{array}{ccccc} & \text{Tape drives} & \text{Plotters} & \text{Scanners} & \text{CD Roms} \\ E = (& 4 & 2 & 3 & 1 \ ) \end{array}$$

$$\begin{array}{ccccc} & \text{Tape drives} & \text{Plotters} & \text{Scanners} & \text{CD Roms} \\ A = (& 2 & 1 & 0 & 0 \ ) \end{array}$$

Current allocation matrix

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

Max request matrix

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix}$$

Avoidance with multiple resources

$$\begin{array}{rcccc} & \text{Tape drives} & \text{Plotters} & \text{Scanners} & \text{CD Roms} \\ E = (& 4 & 2 & 3 & 1\ ) \end{array}$$

$$\begin{array}{rcccc} & \text{Tape drives} & \text{Plotters} & \text{Scanners} & \text{CD Roms} \\ A = (& 2 & 1 & 0 & 0\ ) \\ & \mathbf{2} & \mathbf{2} & \mathbf{2} & \mathbf{0} \end{array}$$

Current allocation matrix

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

Max request matrix

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix}$$

Avoidance with multiple resources

$$\begin{array}{lcccc} & \text{Tape drives} & \text{Plotters} & \text{Scanners} & \text{CD Roms} \\ E = (& 4 & 2 & 3 & 1\ ) \end{array}$$

$$\begin{array}{lcccc} & \text{Tape drives} & \text{Plotters} & \text{Scanners} & \text{CD Roms} \\ A = (& 2 & 1 & 0 & 0\ ) \\ & \mathbf{2} & \mathbf{2} & \mathbf{2} & \mathbf{0} \end{array}$$

Current allocation matrix

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ \cancel{0} & \cancel{1} & \cancel{2} & \cancel{0} \end{bmatrix}$$

Max request matrix

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ \cancel{2} & \cancel{1} & \cancel{0} & \cancel{0} \end{bmatrix}$$

Avoidance with multiple resources

$$\begin{array}{rcccc} & \text{Tape drives} & \text{Plotters} & \text{Scanners} & \text{CD Roms} \\ E = (& 4 & 2 & 3 & 1\ ) \end{array}$$

$$\begin{array}{rcccc} & \text{Tape drives} & \text{Plotters} & \text{Scanners} & \text{CD Roms} \\ A = (& 2 & 1 & 0 & 0\ ) \\ & \mathbf{2} & \mathbf{2} & \mathbf{2} & \mathbf{0} \\ & \mathbf{4} & \mathbf{2} & \mathbf{2} & \mathbf{1} \end{array}$$

Current allocation matrix

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ \cancel{2} & \cancel{0} & \cancel{0} & \cancel{1} \\ \cancel{0} & \cancel{1} & \cancel{2} & \cancel{0} \end{bmatrix}$$

Max request matrix

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ \cancel{1} & \cancel{0} & \cancel{1} & \cancel{0} \\ \cancel{2} & \cancel{1} & \cancel{0} & \cancel{0} \end{bmatrix}$$

Problems with deadlock avoidance

- **Deadlock avoidance is often impossible**
  - because you don't know in advance what resources a process will need!
- **Alternative approach "deadlock prevention"**
  - Make deadlock impossible!
  - Attack one of the four conditions that are necessary for deadlock to be possible

Deadlock prevention

- **Conditions necessary for deadlock:**

  Mutual exclusion condition

  Hold and wait condition

  No preemption condition

  Circular wait condition

Deadlock prevention

- **Attacking mutual exclusion?**
  - a bad idea for some resource types
    - resource could be corrupted
  - works for some kinds of resources in certain situations
    - eg., when a resource can be partitioned
- **Attacking no preemption?**
  - a bad idea for some resource types
    - resource may be left in an inconsistent state
  - may work in some situations
    - checkpointing and rollback of idempotent operations

Deadlock prevention

- **Attacking hold and wait?**
  - Require processes to request all resources before they begin!
  - Process must know ahead of time
  - Process must tell system its "max potential needs"
    - eg., like in the bankers algorithm
    - When problems occur a process must release all its resources and start again

Attacking the conditions

- **Attacking circular waiting?**
  - Number each of the resources
  - Require each process to acquire lower numbered resources before higher numbered resources
  - More precisely: "A process is not allowed to request a resource whose number is lower than the highest numbered resource it currently holds"

Recall this example of deadlock

Thread A:

```
acquire (resource_1)
acquire (resource_2)
use resources 1 & 2
release (resource_2)
release (resource_1)
```

Thread B:

```
acquire (resource_2)
acquire (resource_1)
use resources 1 & 2
release (resource_1)
release (resource_2)
```

Assume that resources are ordered:

1. Resource_1
2. Resource_2
3. ...etc...

Recall this example of deadlock

Thread A:

```
acquire (resource_1)
acquire (resource_2)
use resources 1 & 2
release (resource_2)
release (resource_1)
```

Thread B:

```
acquire (resource_2)
acquire (resource_1)
use resources 1 & 2
release (resource_1)
release (resource_2)
```

- Assume that resources are ordered:
- 1. Resource_1
- 2. Resource_2
- 3. ...etc...
- Thread B violates the ordering!

Why Does Resource Ordering Work?

- **Assume deadlock has occurred.**
- **Process A**
  - holds X
  - requests Y
- **Process B**
  - holds Y
  - requests Z
- **Process C**
  - holds Z
  - requests X

Why Does Resource Ordering Work?

- **Assume deadlock has occurred.**
- **Process A**
  - holds X
  - requests Y

$X < Y$

- **Process B**
  - holds Y
  - requests Z
- **Process C**
  - holds Z
  - requests X

Why Does Resource Ordering Work?

- **Assume deadlock has occurred.**
- **Process A**
  - holds X
  - requests Y

$X < Y$

- **Process B**
  - holds Y
  - requests Z

$Y < Z$

- **Process C**
  - holds Z
  - requests X

Why Does Resource Ordering Work?

- **Assume deadlock has occurred.**
- **Process A**
  - holds X
  - requests Y

$X < Y$

- **Process B**
  - holds Y
  - requests Z

$Y < Z$

- **Process C**
  - holds Z
  - requests X

$Z < X$

Why Does Resource Ordering Work?

- Assume deadlock has occurred.
- Process A
  - holds X
  - requests Y

  $X < Y$

- Process B
  - holds Y
  - requests Z

  $Y < Z$

- Process C
  - holds Z
  - requests X

  $Z < X$

This is impossible!

Why Does Resource Ordering Work?

- Assume deadlock has occurred.
- Process A
  - holds X
  - requests Y

$X < Y$

- Process B
  - holds Y
  - requests Z

$Y < Z$

- Process C
  - holds Z
  - requests X

$Z < X$

This is impossible!

Therefore the assumption must be false!

Resource Ordering

- **The chief problem:**
  - It may be hard to come up with an acceptable ordering of resources!

- **Still,this is the most useful approach in an OS**
  1. **ProcessControlBlock**
  2. **FileControlBlock**
  3. **Page Frames**

- **Also, the problem of resources with multiple units is not addressed.**

mm/filemap.c lock ordering

```
/*
 * Lock ordering:
 *  ->i_mmap_lock               (vmtruncate)
 *    ->private_lock            (__free_pte->__set_page_dirty_buffers)
 *      ->swap_lock             (exclusive_swap_page, others)
 *        ->mapping->tree_lock
 *  ->i_mutex
 *    ->i_mmap_lock             (truncate->unmap_mapping_range)
 *  ->mmap_sem
 *    ->i_mmap_lock
 *      ->page_table_lock or pte_lock   (various, mainly in memory.c)
 *        ->mapping->tree_lock  (arch-dependent flush_dcache_mmap_lock)
 *  ->mmap_sem
 *    ->lock_page               (access_process_vm)
 *  ->mmap_sem
 *    ->i_mutex                 (msync)
 *  ->i_mutex
 *    ->i_alloc_sem             (various)
 *  ->inode_lock
 *    ->sb_lock                 (fs/fs-writeback.c)
 *    ->mapping->tree_lock      (__sync_single_inode)
 *  ->i_mmap_lock
 *    ->anon_vma.lock           (vma_adjust)
 *  ->anon_vma.lock
 *    ->page_table_lock or pte_lock     (anon_vma_prepare and various)
 *  ->page_table_lock or pte_lock
 *    ->swap_lock               (try_to_unmap_one)
 *    ->private_lock            (try_to_unmap_one)
 *    ->tree_lock               (try_to_unmap_one)
 *    ->zone.lru_lock           (follow_page->mark_page_accessed)
 *    ->zone.lru_lock           (check_pte_range->isolate_lru_page)
 *    ->private_lock            (page_remove_rmap->set_page_dirty)
 *    ->tree_lock               (page_remove_rmap->set_page_dirty)
 *    ->inode_lock              (page_remove_rmap->set_page_dirty)
 *    ->inode_lock              (zap_pte_range->set_page_dirty)
 *    ->private_lock            (zap_pte_range->__set_page_dirty_buffers)
 *  ->task->proc_lock
 *    ->dcache_lock             (proc_pid_lookup)
 */
```

A word on starvation

- **Starvation and deadlock are two different things**
  - With deadlock – no work is being accomplished for the processes that are deadlocked, because processes are waiting for each other. Once present, it will not go away.

  - With starvation – work (progress) is getting done, however, a particular set of processes may not be getting any work done because they cannot obtain the resource they need

Quiz

- What is deadlock?
- What conditions must hold for deadlock to be possible?
- What are the main approaches for dealing with deadlock?
- Why does resource ordering help?