CS 370: Operating Systems
[Deadlocks]

Computer Science
Colorado State University

Instructor: Louis-Noel Pouchet
Spring 2024

** Lecture slides created by: Shrideep Pallickara

Topics covered in this lecture

- Ostrich Algorithm
- Deadlock Prevention
- Deadlock Avoidance

THE OSTRICH ALGORITHM

Ostrich Algorithm

- Stick your head in the sand; pretend there is no problem at all

- Reactions
  - Mathematician: Unacceptable; prevent at all costs
  - Engineers: How often? Costs? Etc.

OS suffer from deadlocks that are not even detected [1/3]

- Number of processes in the system
  - Total determined by slots in the process table
    - Slots are a finite resource
- Maximum number of open files
  - Restricted by size of the inode table
- Swap space on the disk

OS suffer from deadlocks that are not even detected [2/3]

- Every OS table represents a **finite** resource
- Should we abolish all of these because collection of n processes
  1. Might claim $1/n$ th of the total AND
  2. Then try to claim another one
- Most users prefer occasional deadlock to a restrictive policy
  - E.g. All users: 1 process, 1 open file …. one everything is far too restrictive

OS suffer from deadlocks that are not even detected [3/3]

- If deadlock elimination is free
  - No discussions
- But the price is often high
  - Inconvenient restrictions on processes
- Tradeoff
  - Between **convenience** and **correctness**

Deadlock Characterization

Deadlocks: Necessary Conditions (I)

- **Mutual Exclusion**
  - At least one resource held in *nonsharable* mode
  - When a resource is being used
    - Another requesting process must wait for its release
- **Hold-and-wait**
  - A process must hold one resource
  - Wait to acquire additional resources
    - Which are currently held by other processes

Deadlocks: Necessary Conditions (II)

- **No preemption**
  - Resources cannot be preempted
  - Only voluntary release by process holding it

- **Circular wait**
  - A set of $\{P_0, P_1, \ldots, P_n\}$ waiting processes must exist
    - $P_0 \rightarrow P_1; P_1 \rightarrow P_2, \ldots, P_n \rightarrow P_0$
  - Implies hold-and-wait

DEADLOCK PREVENTION

Deadlock Prevention

- Ensure that **one** of the necessary conditions for deadlocks *cannot* occur
  1. Mutual exclusion
  2. Hold and wait
  3. No preemption
  4. Circular wait

Mutual exclusion must hold for non-sharable resources, but …

- Sharable resources do not require mutually exclusive access
  - *Cannot be involved* in a deadlock
- A process never needs to wait for sharable resource
  - Read-only files
- Some resources are *intrinsically nonsharable*
  - So denying mutual exclusion often not possible

Deadlock Prevention: Ensure hold-and-wait never occurs in the system [Strategy 1]

- Process must request and be allocated all its resources **before** execution
  - Resource requests must precede other system calls

- E.g. copy data from DVD drive, sort file & print
  - Printer needed only at the end
  - BUT process will hold printer for the *entire* execution

Deadlock Prevention: Ensure hold-and-wait never occurs in the system [Strategy 2]

- Allow a process to request resources *only when it has none*
  - *Release* all resources, *before requesting* additional ones

- E.g. copy data from DVD drive, sort file & print
  - First request DVD and disk file
    - Copy and release resources
  - Then request file and printer

Disadvantages of protocols doing hold-and-wait

- **Low resource utilization**
  - Resources are allocated but unused for long durations

- **Starvation**
  - If a process needs several popular resources
    - Popular resource might always be *allocated to some other* process

Deadlock Prevention: Eliminate the preemption constraint [1/2]

- {C1} If a process is holding some resources
- {C2} Process requests another resource
  - Cannot be immediately allocated

- All resources currently held by process is **preempted**
  - Preempted resources added to list of resources process is waiting for

Deadlock Prevention: Eliminate the preemption constraint [2/2]

- Process requests resources that are not currently available
  - If resources allocated to another waiting process
    - Preempt resources from the second process and assign it to the first one
- Often applied when resource state can be ***saved and restored***
  - CPU registers and memory space
  - Unsuitable for tape drives

Deadlock Prevention: Eliminating Circular wait

- Impose **total ordering** of all resource types
  - Assign each resource type a unique number
  - One-to-one function `F:R→N`

```
F(tape drive) = 1;
F(printer) = 12
```

① Request resources in ***increasing* order**

② If several instances of a resource type needed?

  - Single request for all them must be issued

Requesting resources in an increasing order of enumeration

- Process initially requested R_i
- This process can now request R_j ONLY IF

  $F(R_j) > F(R_i)$

- Alternatively, process requesting R_j must have released resources R_i such that

  $F(R_i) >= F(R_j)$

- Eliminates circular wait

Hierarchy of resources and deadlock prevention

- Hierarchy by itself does not prevent deadlocks
  - Developed programs **must follow ordering**

- `F` *based on* ***order*** *of usage* of resources
  - Tape drive needed before printing
    - `F(tape drive) < F(printer)`

Deadlock Prevention: **Summary**

- Prevent deadlocks by **restraining** how requests are made.
  - Ensure at least 1 of the 4 conditions cannot occur

- Side effects:
  - Low device utilization
  - Reduced system throughput

Dining Philosophers: Deadlock prevention (1)

- Mutual exclusion
  - Philosophers can *share* a chopstick

- Hold-and-wait
  - Philosopher should release the first chopstick if it cannot obtain the second one

Dining Philosophers: Deadlock prevention (2)

- Preemption
  - Philosophers can *forcibly take* each other's chopstick
- Circular-wait
  - Number the chopsticks
  - Pick up chopsticks in ascending order
    - Pick the lower numbered one before the higher numbered one

Deadlock Avoidance

Deadlock avoidance

- Require *additional* information about **how** resources are to be requested

- Knowledge about sequence of requests and releases for processes
  - Allows us to decide if resource allocation *could cause a future deadlock*
  - Process P: Tape drive, then printer
  - Process Q: Printer, then tape drive

Deadlock avoidance: Handling resource requests

- For each resource request:
  - Decide whether or not process should wait
    - To avoid possible **future** deadlock

- Predicated on:
  1. Currently available resources
  2. Currently allocated resources
  3. Future requests and releases of each process

Avoidance algorithms differ in the amount and type of information needed

- **Resource allocation state**
  - Number of available and allocated resources
  - Maximum demands of processes
- Dynamically **examine** resource allocation state
  - Ensure circular-wait cannot exist
- Simplest model:
  - Declare maximum number of resources for each type
  - Use information to avoid deadlock

Safe sequence

- **Sequence** of processes <$P_1, P_2, \ldots, P_n$> for the current allocation state
- Resource requests made by P_i can be satisfied by:
  - Currently available resources
  - Resources held by P_j where $j < i$
    - If needed resources not available, P_i can wait
  - In general, when P_i terminates, P_{i+1} can obtain its needed resources
- If no such sequence exists: system state is **unsafe**

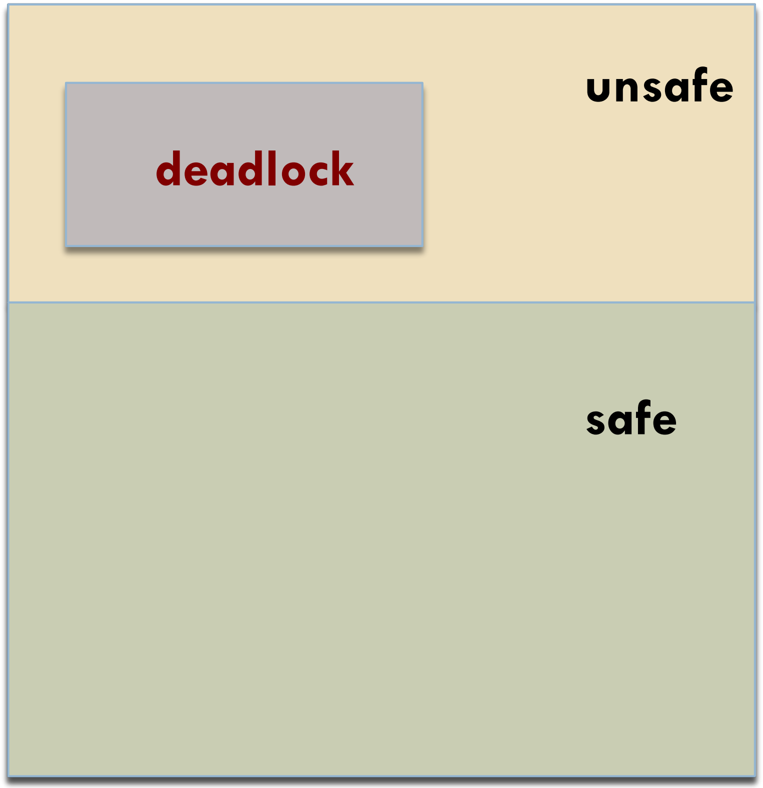
Deadlock avoidance: Safe states

- If the system can:
  1. Allocate resources to each process in **some order**
     - Up to the *maximum* for the process
  2. Still avoid deadlock

Safe states and deadlocks

- A system is safe ONLY IF there is a **safe sequence**
- A safe state is not a deadlocked state
  - Deadlocked state is an unsafe state
  - Not all unsafe states are deadlocks

State spaces



Unsafe states

- A unsafe state *may lead* to deadlock
- **Behavior** of processes controls unsafe states
- Cannot prevent processes from requesting resources such that deadlocks occur

Example: 12 Tape drives available in the system

	Maximum Needs	Current Needs
P_0	10	5
P_1	4	2
P_2	9	2

Before T0:
3 drives available

Safe sequence
<P_1, P_0 , P_2>

- At time **T0** the system is in a safe state
- P_1 can be given 2 tape drives
- When P_1 releases its resources; there are 5 drives
- P_0 uses 5 and subsequently releases them (# 10 now)
- P_2 can then proceed

Example: 12 Tape drives available in the system

	Maximum Needs	Current Needs
P_0	10	5
P_1	4	2
P_2	9	2

Before T1:
3 drives available

- At time **T1**, P_2 is allocated 1 tape drive

Example: 12 Tape drives available in the system

	Maximum Needs	Current Needs
P_0	10	5
P_1	4	2
P_2	9	3

After T1:
2 drives available

- At time **T1**, P_2 is allocated 1 tape drive
- Only P_1 can proceed.
- When P_1 releases its resources; there are 4 drives
  - P_0 needs 5 and P_2 needs 6
- **Mistake** in granting P_2 additional tape drive

Crux of deadlock avoidance algorithms

- **Ensure** that the system will always remain in a safe state
- Resource allocation request **granted** only if it will leave the system in a safe state

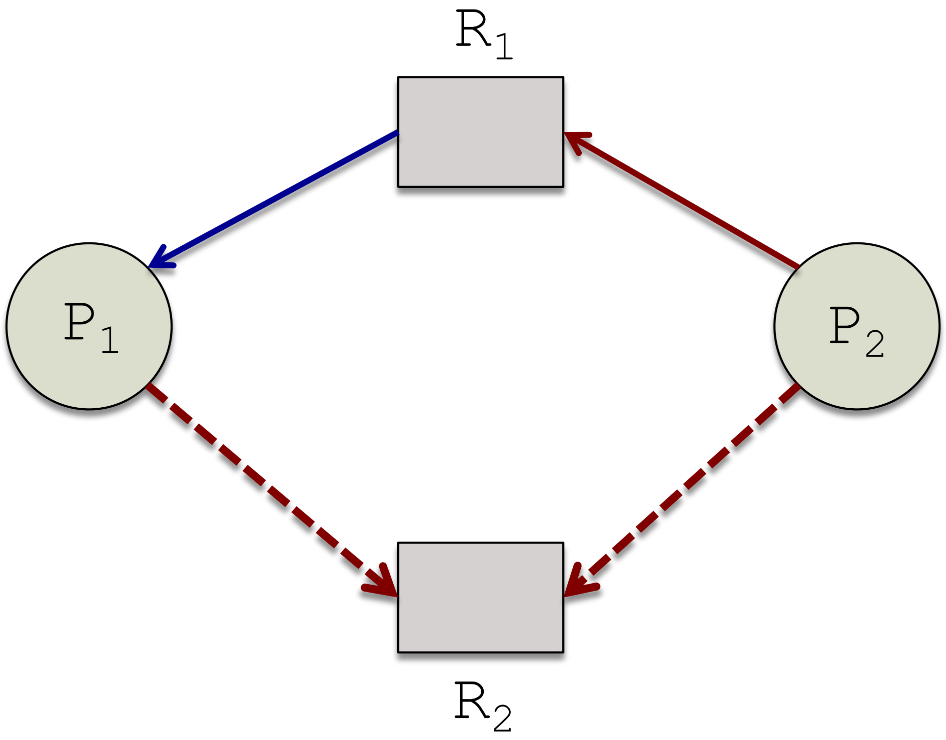
Resource Allocation Graph Algorithm

Claim edges

- Indicates that a process P_i may request a resource R_j at some time in the future.

- Representation:
  - Same direction as request
  - Dotted line

Resource allocation graph with a claim edge



Conversion of claim edges

- When process P_i requests resource R_j
  - Claim edge converted to a request edge

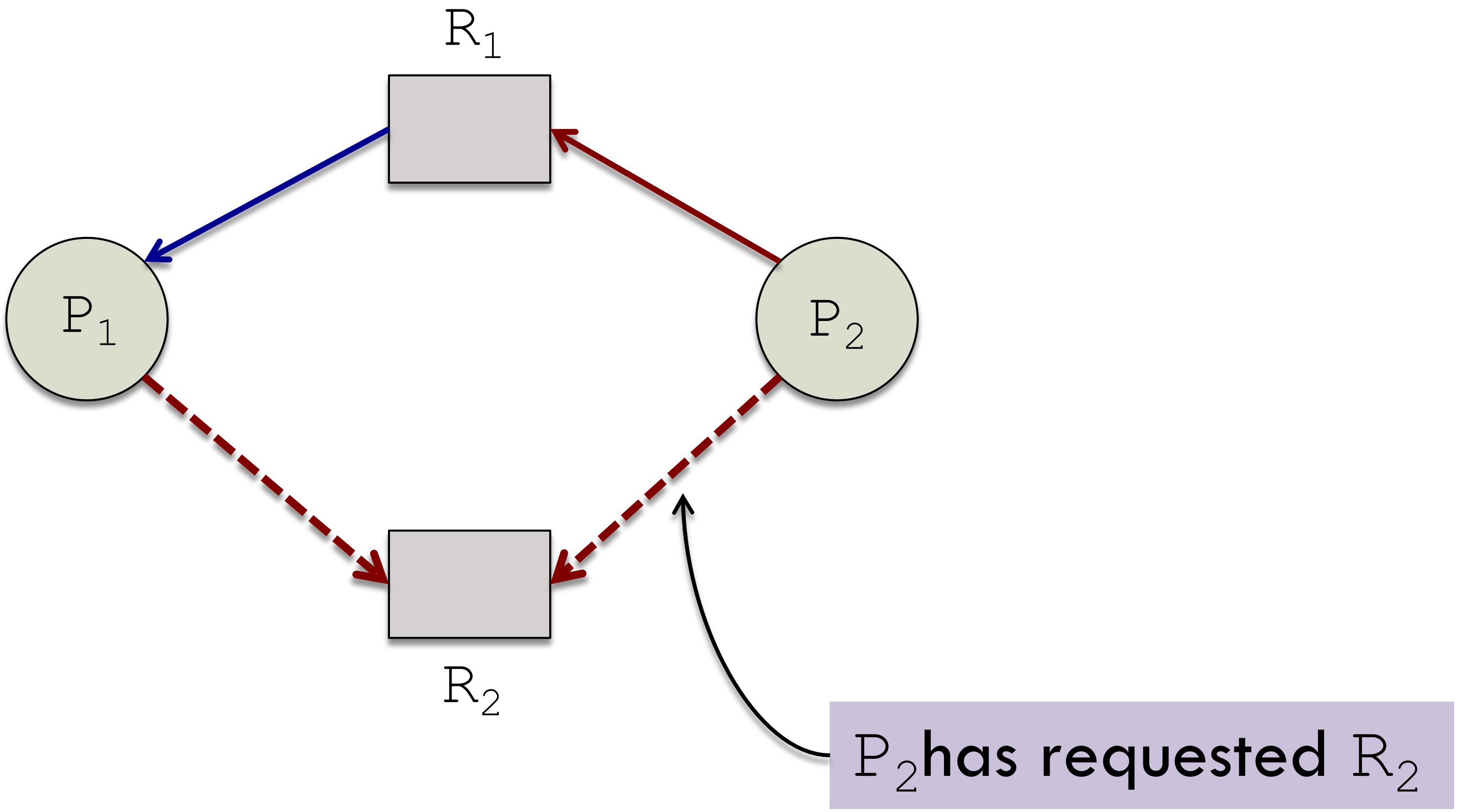
- When resource R_j released by P_i
  - The assignment edge $R_j \rightarrow P_i$ is **reconverted** to a claim edge $P_i \rightarrow R_j$

Allocating resources

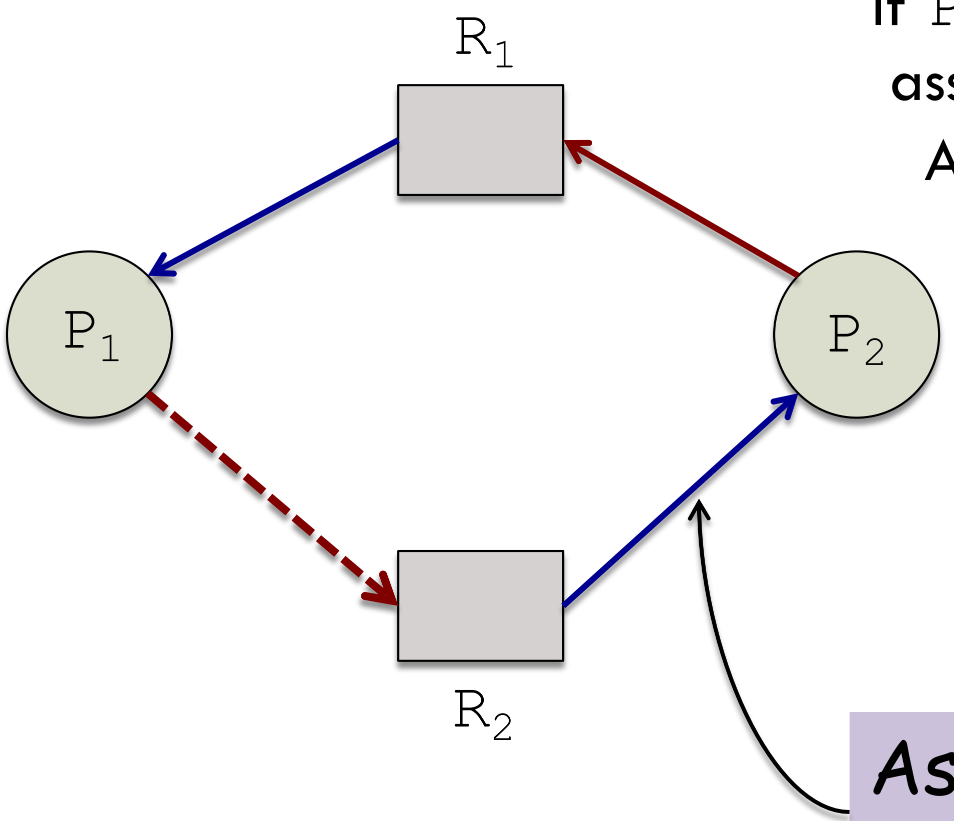
- When process P_i requests resource R_j

- Request granted only if
  - Converting claim edge to $P_i \rightarrow R_j$ to an assignment edge $R_j \rightarrow P_i$ *does not result* in a **cycle**

Using the allocation graph to allocate resources safely



Using the allocation graph to allocate resources safely



If P_1 requests R_2 after it's assigned to P_2?

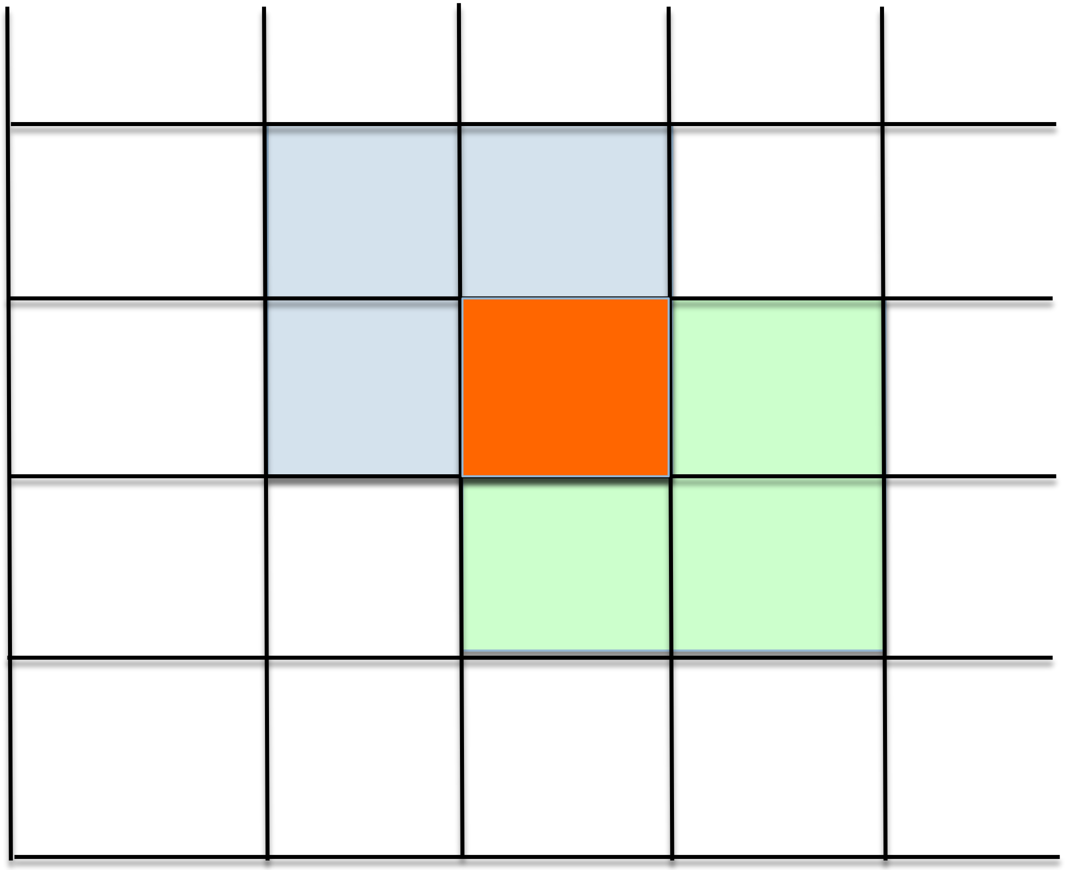
A deadlock will occur

Assignment leads to a cycle

Resource allocation graph algorithm

- Not applicable in systems with multiple resource instances

Resource Trajectories



The contents of this slide-set are based on the following references

- *Avi Silberschatz, Peter Galvin, Greg Gagne. Operating Systems Concepts, 9th edition. John Wiley & Sons, Inc. ISBN-13: 978-1118063330.* [Chapter 7]

- *Andrew S Tanenbaum. Modern Operating Systems. 4th Edition, 2014. Prentice Hall. ISBN: 013359162X/ 978-0133591620. [Chapter 6]*