CSCE 451/851
Operating Systems Principles

Deadlock

Steve Goddard
goddard@cse.unl.edu

http://www.cse.unl.edu/~goddard/Courses/CSCE451

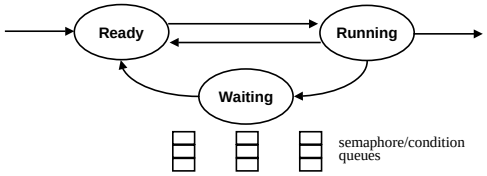
Deadlock
Review — Deadlock in mutual exclusion algorithms

- Each process is "waiting" for the other process to do something
  - » Nothing (constructive) is ever done — both processes are forever stuck

```
global var incS : array[1..2] of boolean := (FALSE,FALSE)

process P1                      process P2
begin                           begin
  loop                            loop
    incS[1] := TRUE                 incS[2] := TRUE
    while incS[2] do                while incS[1] do
      NOOP                            NOOP
    end while                       end while
    <critical section>              <critical section>
    incS[1] := FALSE                incS[2] := FALSE
  end loop                        end loop
end P1                          end P2
```

Deadlock
Definition



A set of processes are deadlocked when every process in the set is waiting for an event that can only be generated by some process in the set

Example: a computer system with two processes

- a process that is printing a large PostScript job is waiting for more memory
- a visualization process with lots of memory is waiting to use the printer

3

A Formal Model of Deadlock
Definitions

- Basic components of any resource allocation problem
  - » Processes
  - » Resources
- Paradigm of resource usage

```
process P
begin
  request(resource-type, amount)
  <use the resource>
  release(resource-type, amount)
end P
```

4

Resources

Two classes of resources

- Serially reusable (*SR*) resources
  - » a constant number of units
  - » boolean state (allocated or unallocated)
  - » no sharing
  - » units of the resource created by the system (not by processes using the resource)
- Consumable resources (*CR*)
  - » the number of available units of the resource varies over time
  - » a producer process may release units of the resource it did not acquire (*i.e.*, a process may create units of the resource)
  - » in general, acquired resources are not returned, they are consumed

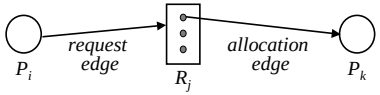
5

A Graph Theoretic Model of Deadlock

The resource allocation graph (*RAG*)

- Model the state of a computer system as a directed graph $G = (V, E)$
  - » V = the set of vertices = $\{P_1, ..., P_n\} \cup \{R_1, ..., R_n\}$

  - » E = the set of edges =
    {edges *from* a resource *to* a process} ∪
    {edges *from* a process *to* a resource}

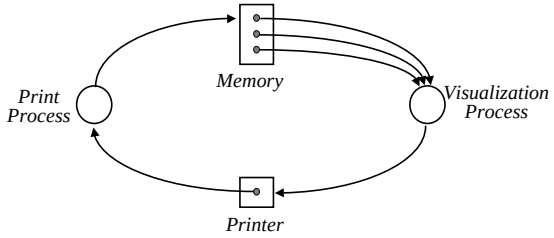


6

Resource Allocation Graphs
Examples

- A printing process that is waiting for more memory & a large memory process that is waiting to print

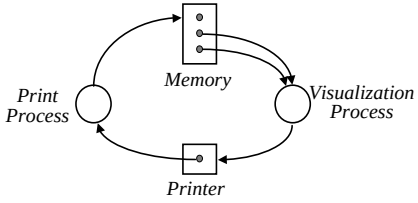
$G = \{print, visualization\} \cup \{memory, printer\}$



7

A Graph Theoretic Model of Deadlock
Resource allocation graphs & deadlock

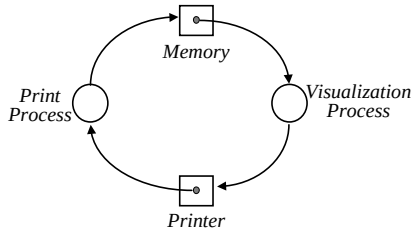
- Theorem: *If a graph does not contain a cycle then no processes are deadlocked*
  - » A cycle in a *RAG is* a necessary condition for deadlock
- Is the existence of a cycle a sufficient condition?



8

A Graph Theoretic Model of Deadlock
Resource allocation graphs & deadlock

- Theorem: *If there is only a single unit of all resources then a set of processes are deadlocked iff the processes & resources form a cycle in the RAG*



9

Using the Theory
An operational definition of deadlock

- A set of processes are deadlocked *iff* the following conditions hold simultaneously
  1. Mutual exclusion is required
  2. A process is in a "hold-and-wait" state
  3. Preemption is not allowed
  4. Circular waiting exists
     (A cycle exists in the *RAG*)

10

Dealing With Deadlock
Deadlock prevention & avoidance

- Adopt some resource allocation protocol that ensures deadlock can never occur
  - » Deadlock prevention
  - » Deadlock avoidance
- Deadlock prevention
  - » Ensure that one of the four deadlock conditions never occurs
    - ❖ Mutex?
    - ❖ Hold & Wait?
    - ❖ Non-preemption?
    - ❖ Circular waiting?

11

Dealing With Deadlock
Deadlock avoidance

Examine each resource request and determine whether or not granting the request can lead to deadlock

- Define a set of vectors & matrices that characterize the current state of all resources & processes
  - » *resource allocation state matrix*
    n_{ij} = the number of units of resource j held by process i
  - » *maximum claim matrix*
    n_{ij} = the maximum number of units of resource i that the process j will ever require simultaneously
  - » *available vector*
    n_i = the number of units of resource i that are unallocated

$$\begin{array}{c} \\ P_1 \\ P_2 \\ P_3 \\ \vdots \\ P_p \end{array} \begin{array}{c} \begin{array}{cccc} R_1 & R_2 & R_3 & \ldots & R_r \end{array} \\ \begin{bmatrix} n_{1,1} & n_{1,2} & n_{1,3} & \ldots & n_{1,r} \\ n_{2,1} & n_{2,2} & & & \\ n_{3,1} & & \ddots & & \vdots \\ \vdots & & & & \\ n_{p,1} & & \ldots & & n_{p,r} \end{bmatrix} \end{array}$$

$$<n_1, n_2, n_3, \ldots, n_r>$$

12

Deadlock Avoidance
State definitions

- A *resource allocation state* is *safe* if the system can allocate resources to each process up to its maximum claim such that the system can not deadlock
  - » There must be an ordering of the processes P_1, P_2, ..., P_p, such that for all processes P_i,

$$MAX\_CLAIM_{P_i} - ALLOCATION_{P_i} \le AVAIL + \sum_{j=1}^{i-1} ALLOCATION_{P_j}$$

    i.e., the number of resources that P_i can request is less than the resources available now plus those held by lower numbered processes in the sequence

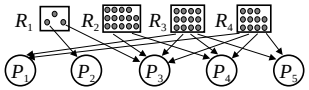
  - » This ordering of processes is called a *safe sequence*
    - If a safe sequence exists then there exists a process (P_1) that can execute to completion
    - P_j can execute to completion at worst after processes P_1- P_{j-1} complete

13

Deadlock Avoidance
Example

- A computer system with 5 processes and 4 resources

	ALLOCATION				*MAX_CLAIM*				*AVAILABLE*			
	R_1	R_2	R_3	R_4	R_1	R_2	R_3	R_4	R_1	R_2	R_3	R_4
P_1	0	0	1	2	0	0	1	2	1	5	2	0
P_2	1	0	0	0	1	7	5	0				
P_3	1	3	5	3	2	3	5	6				
P_4	0	6	3	2	0	6	5	2				
P_5	0	0	1	4	0	6	5	6				



- Is this system in a safe state?
  - » Does there exist a safe sequence?
    - an ordering of the processes such that:

$$MAX\_CLAIM_{P_i} - ALLOCATION_{P_i} < AVAIL + \sum_{j=1}^{i-1} ALLOCATION_{P_j}$$

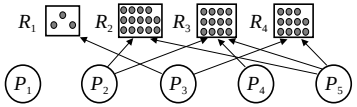
14

Deadlock Avoidance Example

Safe sequence computation

1. Compute the largest possible resource request a process can make

	MAX_CLAIM					*ALLOCATION*					*MAX_REQUEST*				
	R_1	R_2	R_3	R_4		R_1	R_2	R_3	R_4		R_1	R_2	R_3	R_4
P_1	0	0	1	2		0	0	1	2		0	0	0	0
P_2	1	7	5	0	–	1	0	0	0	=	0	7	5	0
P_3	2	3	5	6		1	3	5	3		1	0	0	3
P_4	0	6	5	2		0	6	3	2		0	0	2	0
P_5	0	6	5	6		0	0	1	4		0	6	4	2



2. Attempt to build a safe sequence

Deadlock Avoidance Example

Safe sequence computation

	ALLOCATION				*MAX_CLAIM*				*AVAILABLE*				*MAX_REQUEST*			
	R_1	R_2	R_3	R_4	R_1	R_2	R_3	R_4	R_1	R_2	R_3	R_4	R_1	R_2	R_3	R_4
P_1	0	0	1	2	0	0	1	2	1	5	2	0	0	0	0	0
P_2	1	0	0	0	1	7	5	0					0	7	5	0
P_3	1	3	5	3	2	3	5	6					1	0	0	3
P_4	0	6	3	2	0	6	5	2					0	0	2	0
P_5	0	0	1	4	0	6	5	6					0	6	4	2

- » Does there exist a process P_i such that $MAX\_REQUEST_{P_i} \leq AVAILABLE$?
- » If no, then there is no safe sequence, the state is unsafe
- » If yes, add P_i to the sequence
- » Set $AVAILABLE = AVAILABLE + ALLOCATION_{P_i}$

Deadlock Avoidance Example
Safe sequence computation

	ALLOCATION				*MAX_CLAIM*				*AVAILABLE*				*MAX_REQUEST*			
	R_1	R_2	R_3	R_4	R_1	R_2	R_3	R_4	R_1	R_2	R_3	R_4	R_1	R_2	R_3	R_4
P_1	**0**	**0**	**1**	**2**	**0**	**0**	**1**	**2**	**2**	**1**	**0**	**0**	**0**	**0**	**0**	**0**
P_2	**0**	**4**	**2**	**0**	**1**	**7**	**5**	**0**					**1**	**3**	**3**	**0**
P_3	**1**	**3**	**5**	**3**	**2**	**3**	**5**	**6**					**1**	**0**	**0**	**3**
P_4	**0**	**6**	**3**	**2**	**0**	**6**	**5**	**2**					**0**	**0**	**2**	**0**
P_5	**0**	**0**	**1**	**4**	**0**	**6**	**5**	**6**					**0**	**6**	**4**	**2**

- What if P_2 wants to change its allocation to <0, 4, 2, 0>?

17

Deadlock Avoidance Example
Banker's algorithm

Avoiding a "run on the bank"

- Simulate the effect of granting a process's resource allocation request
- Then check to see if a safe sequence exists

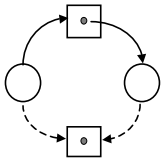
Issues

- Complexity?

18

Banker's Algorithm
Interesting special cases

- Single instance resources
- Introduce a new edge into the *RAG* — a *claim edge*
  - » Indicates that a process *may* request a resource in the future



- A request can be granted only if the conversion of a claim edge into an allocation edge does not create a cycle

19

Dealing With Deadlock
Deadlock detection & recovery

- Deadlock prevention & avoidance
  - » Resource allocation protocols that prohibit deadlock
- The common approach
  - » Let the system deadlock & *then* deal with it
    - ❖ Detect that a set of processes are deadlocked
    - ❖ Recover from the deadlock

20

Deadlock Detection & Recovery
Detecting deadlock

- Run Banker's algorithm & see if a safe sequence exists
  - » Replace *MAX_REQUEST* with simply "*REQUEST*"
  - » If a safe sequence does not exist then the system is deadlocked
- How often should the OS check for deadlock?
  - » After every resource request?
  - » Only when we suspect deadlock has occurred?

21

Deadlock Detection & Recovery
Recovering from deadlock

- Abort all deadlocked processes & reclaim their resources
- Abort one process at a time until all cycles in the *RAG* are eliminated
- Where to start?
  - » Low priority processes
  - » Processes with the most resources
  - » ...

22