Lecture 19 - Deadlocks

CprE 308

February 24, 2013

Review

Review

- Three Examples of Deaclock
  - Multiple Mutex Locks
  - Producer-Consumer Problem
  - Dining PHilosopher Problem

Review: Producer Consumer using Semaphores

Shared Variables

- count (number of items in buffer)
- buffer
- N (maximum size of buffer)

Semaphores

- Empty - semaphore initialized to N (number of free slots in buffer)
- Full - semaphore initialized to zero (number of items in buffer)

Review: Producer Consumer using Semaphores (Example)

Producer

```
while(TRUE) {
  item = produce();
  down(Empty);
  lock(mutex);
  insert(item,buffer);
  count++;
  unlock(mutex);
  up(Full);
```

Consumer

```
while(TRUE) {
  down(Full);
  lock(mutex);
  item = remove(buffer);
  count--;
  unlock(mutex);
  up(Empty);
  consume(item);
}
```

Review: Taking Multiple Locks

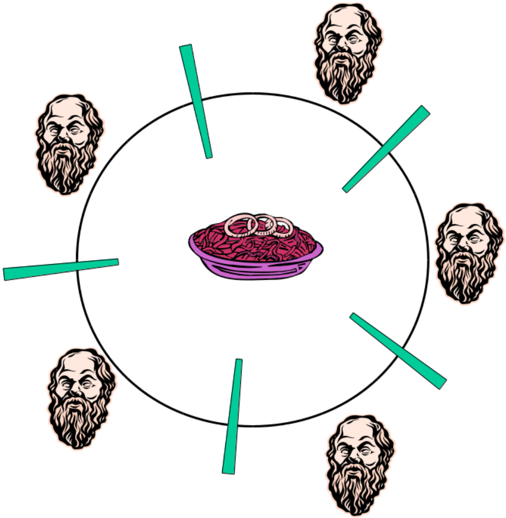
Thread A

```
proc1() {
  pthread_mutex_lock(&m1);
  /* use object 1 */
  pthread_mutex_lock(&m2);
  /* use objects 1 and 2 */
  pthread_mutex_unlock(&m2);
  pthread_mutex_unlock(&m1);
}
```

Thread B

```
proc2() {
  pthread_mutex_lock(&m2);
  /* use object 2 */
  pthread_mutex_lock(&m1);
  /* use objects 1 and 2 */
  pthread_mutex_unlock(&m1);
  pthread_mutex_unlock(&m2);
}
```

Dining Philosophers



Intro

Deadlocks

Chapter 6

1. Resources
2. Introduction to Deadlocks
3. The Ostrich Algorithm
4. Deadlock detection and recovery
5. Deadlock avoidance
6. Deadlock prevention
7. Other issues

Resources

- Examples of computer resources
  - printers
  - tape drives
  - tables
- Processes need access to resources in reasonable order
- Suppose a process holds resource A and requests resource B
  - At same time another process holds B and requests A
  - Both are blocked and remain so

Resources (Cont)

- Deadlocks occur when...
  - Process are granted exclusive access to devices
  - We refer to these devices generally as *resources*
- Preemptable resources
  - Can be taken away from a process with no ill effects
- Nonpreemptable resources
  - will cause the process to fail if taken away

Resource (Cont)

- Sequences of events required to use a resource
  1. Request the resource
  2. Use the resource
  3. Release the resource
- Must wait if request is denied
  - Requesting process may be blocked
  - May fail with error code

Introduction to Deadlocks

- Formal definition:

  A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause.

- Usually the event is release of a currently held resource
- None of the processes can. . .
  - run
  - release resources
  - be awakened

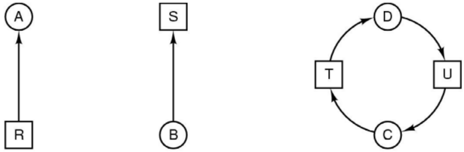
Four Conditions for Deadlock

1. Mutual Exclusion condition
   - each resource assigned to 1 process or is available
2. Hold and wait condition
   - process holding resources can request additional
3. No preemption condition
   - previously granted resources cannot be forcibly taken away
4. Circular wait condition
   - must be a circular chain of 2 or more processes
   - each is waiting for resource held by next members of the chain

Modeling

Deadlock Modeling

- Modeled with directed graphs



(a) (b) (c)

- resource R assigned to process A
- process B is requesting/waiting for resource S
- process C and D are in deadlock over resources T and U

Deadlock Modeling

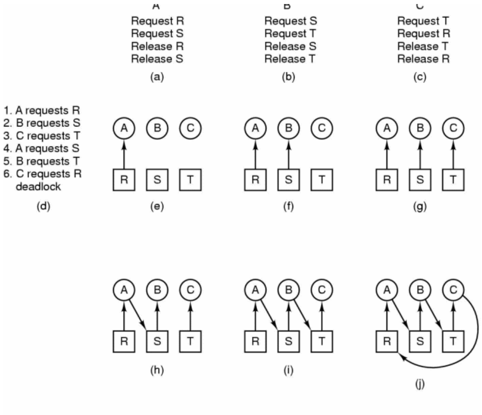


Figure : How deadlock occurs

Deadlock Strategies

Strategies for dealing with deadlocks

1. Just ignore the problem altogether
2. Detection and recovery
3. Dynamic avoidance
   - careful resource allocation
4. Prevention
   - Negating one of the four necessary conditions

Ostrich Algorithm

Ostrich Algorithm

Stick your head in the sand

- Pretend there is no problem at all

Detection

Deadlock Detection and Recovery

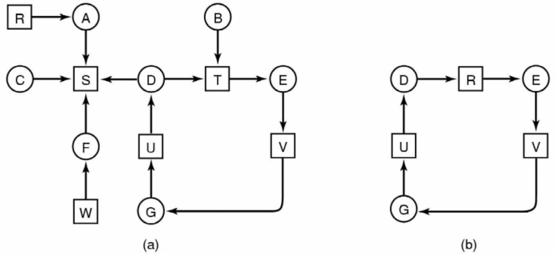
Do not attempt to *prevent* deadlocks

- Instead, detect after the fact and then recover

Deadlock Detection

- Suppose you were given
  - The resource currently held by each
  - The additional resources each process wants
- Question:
  - Is the system deadlocked?

Detection with One Resource of Each Type



- Note the resource ownership and requests
- A cycle can be found within the graph, denoting deadlock

Deadlock Detection Rule (Algorithm)

- Is there some order of execution of the processes which allows all of them to complete?
  - If yes, then not deadlocked
  - If no such order exists, then deadlocked
- After each process finishes, it releases resources back to the system

Example 1

Resource Type	Copies
A	2
B	2
C	2

Process	Resources Held	Resources Needed
P1	A=1, B=1	1 of B
P2	B=1, C=1	1 of C
P3	C=1, A=1	none

Not deadlocked, since P3 can run, releasing resources for P2, which releases resource to P1 upon finishing

Example 2

Resource Type	Copies
A	2
B	2
C	2

Process	Resources Held	Resources Needed
P1	A=1, B=1	1 of B
P2	B=1, C=1	1 of C
P3	C=1, A=1	1 of A

Deadlocked, since nobody can make progress

Example 3

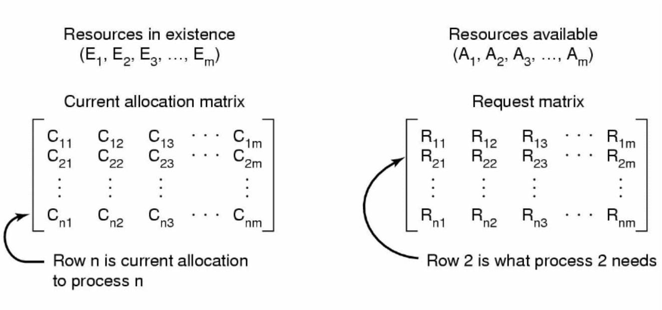
Resource Type	Copies
A	2
B	2
C	2

Process	Resources Held	Resources Needed
P1	A=1, B=1	1 of B
P2	B=1, C=1	2 of A
P3	C=1, A=1	None

Deadlocked:

- Let P3 complete
- P3 releases resources. But then, nobody else can proceed

Detection with Multiple Resources of Each Type



Data structures needed by deadlock detection algorithm

Detection with Multiple Resources of Each Type (cont.)

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

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

Current allocation matrix

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

Request matrix

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

An example for the deadlock detection algorithm

Recovery from Deadlock

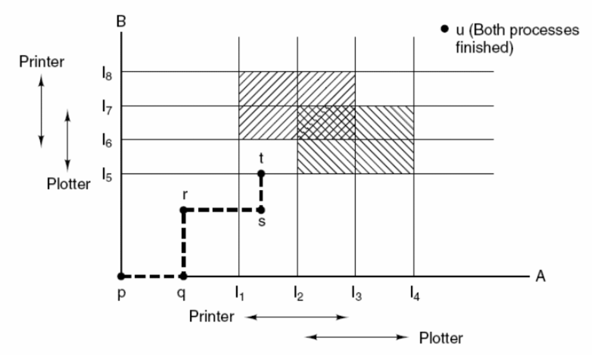
- Recovery through Preemption
  - Take resource away from owner and give it to another
- Recovery through Rollback
  - Roll back execution to checkpoint
- Recovery through Killing-Process
  - Kill process to break cycle

Avoidance

Deadlock Avoidance

- Be very conservative when granting resources
- Don't grant a resource if it could lead to a *potential deadlock*
- Not very practical, since this is too much overhead for granting resources

Resource Trajectories



Safe and Unsafe States

	Has	Max
A	3	9
B	2	4
C	2	7

Free: 3

(a)

	Has	Max
A	3	9
B	4	4
C	2	7

Free: 1

(b)

	Has	Max
A	3	9
B	0	–
C	2	7

Free: 5

(c)

	Has	Max
A	3	9
B	0	–
C	7	7

Free: 0

(d)

	Has	Max
A	3	9
B	0	–
C	0	–

Free: 7

(e)

Demonstration that the state in (a) is safe

Safe and Unsafe States (cont.)

	Has	Max
A	3	9
B	2	4
C	2	7

Free: 3

(a)

	Has	Max
A	4	9
B	2	4
C	2	7

Free: 2

(b)

	Has	Max
A	4	9
B	4	4
C	2	7

Free: 0

(c)

	Has	Max
A	4	9
B	—	—
C	2	7

Free: 4

(d)

Demonstration that the state in (b) is not safe

The Banker's Algorithm for a Single Resource

	Has	Max
A	0	6
B	0	5
C	0	4
D	0	7

Free: 10

(a)

	Has	Max
A	1	6
B	1	5
C	2	4
D	4	7

Free: 2

(b)

	Has	Max
A	1	6
B	2	5
C	2	4
D	4	7

Free: 1

(c)

- Three resource allocation states
  - Safe
  - Safe
  - Unsafe

Banker's Algorithm for Multiple Resources

Process	Tape drives	Plotters	Scanners	CD ROMs
A	3	0	1	1
B	0	1	0	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0

Resources assigned

Process	Tape drives	Plotters	Scanners	CD ROMs
A	1	1	0	0
B	0	1	1	2
C	3	1	0	0
D	0	0	1	0
E	2	1	1	0

Resources still needed

E = (6342)
P = (5322)
A = (1020)

Example of a banker's algorithm with multiple resources

Prevention

Deadlock Prevention

- Attacking the mutual exclusion condition
- Attacking the hold and wait condition
- Attacking the no preemption condition
- Attacking the circular wait condition

Attacking the Mutual Exclusion Condition

Somehow allow multiple processes to use resources

- Example: Printer Spooling

Attacking the Hold and Wait Condition

- Require processes to request resources before starting
  - a process never has to wait for what it needs
- Problems
  - may not have required resources at start of run
  - also ties up resources other processes could be using
- Variation:
  - process must give up all resources
  - then request all immediately needed

Attacking the No Preemption Condition

- This is not a viable option
- Consider a process given the printer
  - halfway through its job
  - now forcibly take away printer
  - !????
  - Profit(?)

Attacking the Circular Wait Condition

- Normally ordered resources
- A resource graph

Deadlock Prevention summaries

Condition	Approach
Mutual exclusion	Spool everything
Hold and wait	Request all resources initially
No preemption	Take resources away
Circular wait	Order resources numerically

Nonresource Deadlocks

- Possible for two processes to deadlock
  - each is waiting for the other to do some task
- Can happen with semaphores
  - each process required to do a `down()` on two semaphores (*mutex* and another)
  - if done in wrong order, deadlock results

Other Issues

- Two-phase locking
- Communication deadlocks
- Livelock
- Starvation

Communication Deadlocks

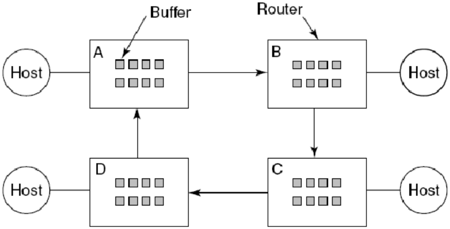


Figure : A resource deadlock in a network

Livelock

```
void process_1(void) {
  enter_region(&resource_1);
  enter_region(&resource_2);
  use_both_resources();
  leave_region(&resource_2);
  leave_region(&resource_1);
}
```

```
void process_2(void) {
  enter_region(&resource_2);
  enter_region(&resource_1);
  use_both_resources();
  leave_region(&resource_1);
  leave_region(&resource_2);
}
```