Concurrency: Deadlock and Starvation

Chapter 6

Deadlock

- Permanent blocking of a set of processes that either compete for system resources or communicate with each other
- No efficient solution
- Involve conflicting needs for resources by two or more processes

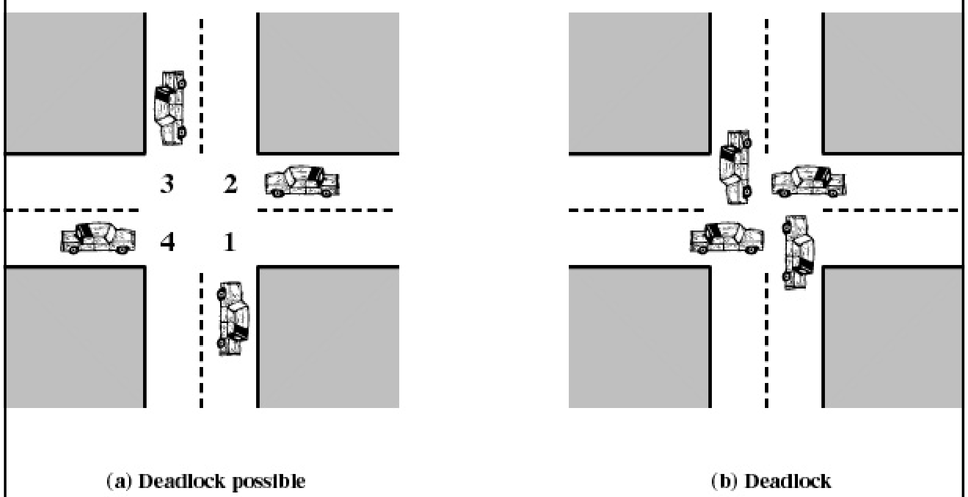


Figure 6.1 Illustration of Deadlock

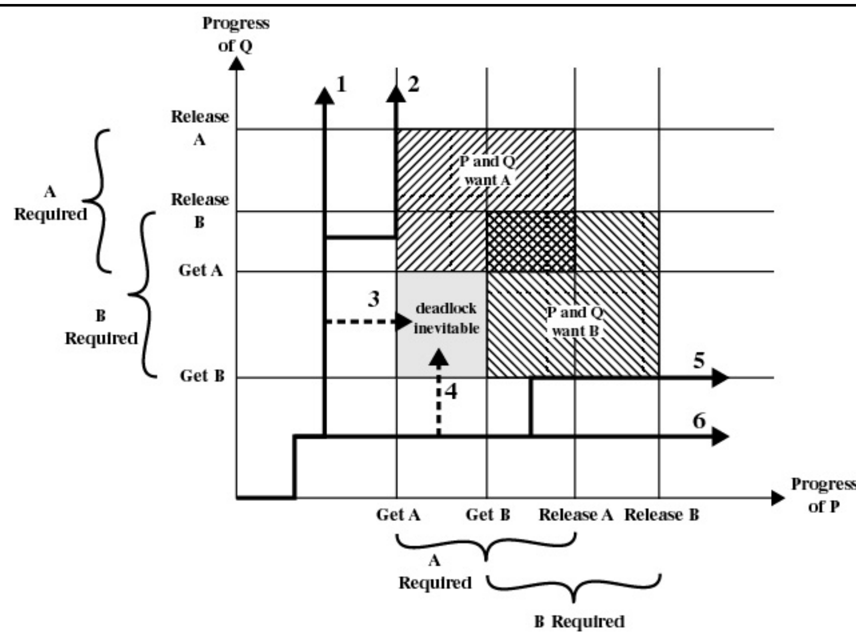


Figure 6.2 Example of Deadlock [BACO98]

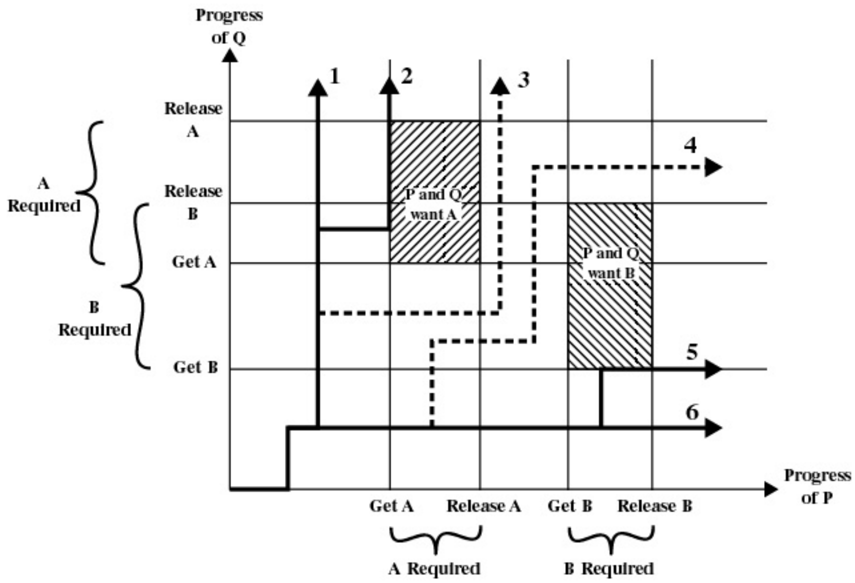


Figure 6.3 Example of No Deadlock [BACO98]

Reusable Resources

- Used by one process at a time and not depleted by that use
- Processes obtain resources that they later release for reuse by other processes
- Processors, I/O channels, main and secondary memory, files, databases, and semaphores
- Deadlock occurs if each process holds one resource and requests the other

6

Example of Deadlock

Process P

Step	Action
p_0	Request (D)
p_1	Lock (D)
p_2	Request (T)
p_3	Lock (T)
p_4	Perform function
p_5	Unlock (D)
p_6	Unlock (T)

Process Q

Step	Action
q_0	Request (T)
q_1	Lock (T)
q_2	Request (D)
q_3	Lock (D)
q_4	Perform function
q_5	Unlock (T)
q_6	Unlock (D)

Figure 6.4 Example of Two Processes Competing for Reusable Resources

7

Another Example of Deadlock

- Space is available for allocation of 200K bytes, and the following sequence of events occur

```
P1
. . .
Request 80K bytes;
. . .
Request 60K bytes;
```

```
P2
. . .
Request 70K bytes;
. . .
Request 80K bytes;
```

- Deadlock occurs if both processes progress to their second request

8

Consumable Resources

- Created (produced) and destroyed (consumed) by a process
- Interrupts, signals, messages, and information in I/O buffers
- Deadlock may occur if a Receive message is blocking
- May take a rare combination of events to cause deadlock

9

Example of Deadlock

- Deadlock occurs if receive is blocking

```
P1
. . .
Receive(P2);
. . .
Send(P2, M1);
```

```
P2
. . .
Receive(P1);
. . .
Send(P1, M2);
```

10

Conditions for Deadlock

- Mutual exclusion
  - only one process may use a resource at a time
- Hold-and-wait
  - A process request all of its required resources at one time

11

Conditions for Deadlock

- No preemption
  - If a process holding certain resources is denied a further request, that process must release its original resources
  - If a process requests a resource that is currently held by another process, the operating system may preempt the second process and require it to release its resources

12

Conditions for Deadlock

- Circular wait
  - Prevented by defining a linear ordering of resource types

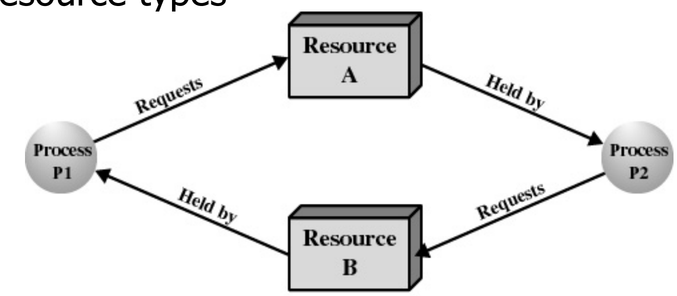


Figure 6.5 Circular Wait

Deadlock Avoidance

- A decision is made dynamically whether the current resource allocation request will, if granted, potentially lead to a deadlock
- Requires knowledge of future process request

Two Approaches to Deadlock Avoidance

- Do not start a process if its demands might lead to deadlock
- Do not grant an incremental resource request to a process if this allocation might lead to deadlock

Resource Allocation Denial

- Referred to as the banker's algorithm
- State of the system is the current allocation of resources to process
- Safe state is where there is at least one sequence that does not result in deadlock
- Unsafe state is a state that is not safe

Determination of a Safe State Initial State

Claim Matrix

	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

Allocation Matrix

	R1	R2	R3
P1	1	0	0
P2	6	1	2
P3	2	1	1
P4	0	0	2

Resource Vector

R1	R2	R3
9	3	6

Available Vector

R1	R2	R3
0	1	1

(a) Initial state

17

Determination of a Safe State P2 Runs to Completion

Claim Matrix

	R1	R2	R3
P1	3	2	2
P2	0	0	0
P3	3	1	4
P4	4	2	2

Allocation Matrix

	R1	R2	R3
P1	1	0	0
P2	0	0	0
P3	2	1	1
P4	0	0	2

Available Vector

R1	R2	R3
6	2	3

(b) P2 runs to completion

18

Determination of a Safe State P1 Runs to Completion

Claim Matrix

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	3	1	4
P4	4	2	2

Allocation Matrix

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	2	1	1
P4	0	0	2

Available Vector

R1	R2	R3
7	2	3

(c) P1 runs to completion

Determination of a Safe State P3 Runs to Completion

Claim Matrix

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	0	0	0
P4	4	2	2

Allocation Matrix

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	0	0	0
P4	0	0	2

Available Vector

R1	R2	R3
9	3	4

(d) P3 runs to completion

Determination of an Unsafe State

Claim Matrix

	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

Allocation Matrix

	R1	R2	R3
P1	1	0	0
P2	5	1	1
P3	2	1	1
P4	0	0	2

Resource Vector

R1	R2	R3
9	3	6

Available Vector

R1	R2	R3
1	1	2

(a) Initial state

Determination of an Unsafe State

Claim Matrix

	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

Allocation Matrix

	R1	R2	R3
P1	2	0	1
P2	5	1	1
P3	2	1	1
P4	0	0	2

Available Vector

R1	R2	R3
0	1	1

(b) P1 requests one unit each of R1 and R3

Deadlock Avoidance

- Maximum resource requirement must be stated in advance
- Processes under consideration must be independent; no synchronization requirements
- There must be a fixed number of resources to allocate
- No process may exit while holding resources

Deadlock Detection

	R1	R2	R3	R4	R5
P1	0	1	0	0	1
P2	0	0	1	0	1
P3	0	0	0	0	1
P4	1	0	1	0	1

Request Matrix Q

	R1	R2	R3	R4	R5
P1	1	0	1	1	0
P2	1	1	0	0	0
P3	0	0	0	1	0
P4	0	0	0	0	0

Allocation Matrix A

R1	R2	R3	R4	R5
2	1	1	2	1

Resource Vector

R1	R2	R3	R4	R5
0	0	0	0	1

Available Vector

Figure 6.9 Example for Deadlock Detection

Strategies once Deadlock Detected

- Abort all deadlocked processes
- Back up each deadlocked process to some previously defined checkpoint, and restart all process
  - original deadlock may occur
- Successively abort deadlocked processes until deadlock no longer exists
- Successively preempt resources until deadlock no longer exists

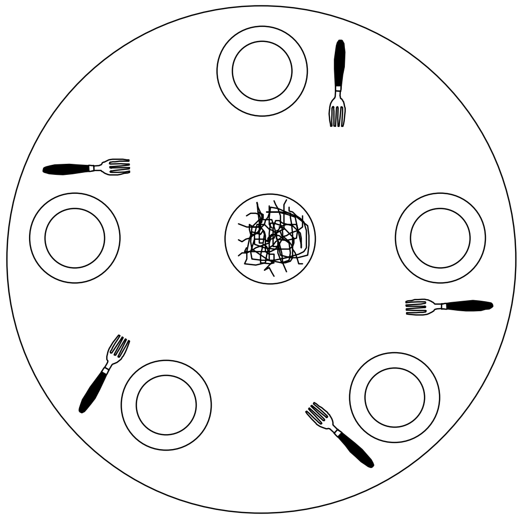
25

Selection Criteria Deadlocked Processes

- Least amount of processor time consumed so far
- Least number of lines of output produced so far
- Most estimated time remaining
- Least total resources allocated so far
- Lowest priority

26

Dining Philosophers Problem



27

UNIX Concurrency Mechanisms

- Pipes
- Messages
- Shared memory
- Semaphores
- Signals

28

Solaris Thread Synchronization Primitives

- Mutual exclusion (mutex) locks
- Semaphores
- Multiple readers, single writer (readers/writer) locks
- Condition variables

29

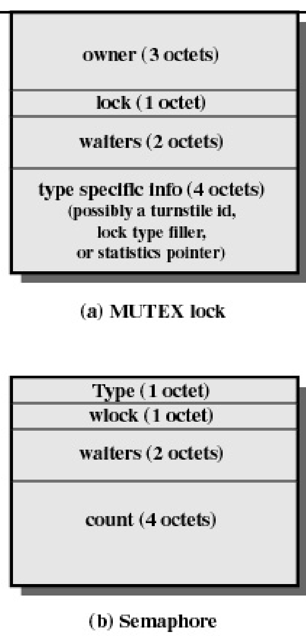


Figure 6.13 Solaris Synchronization Data Structures

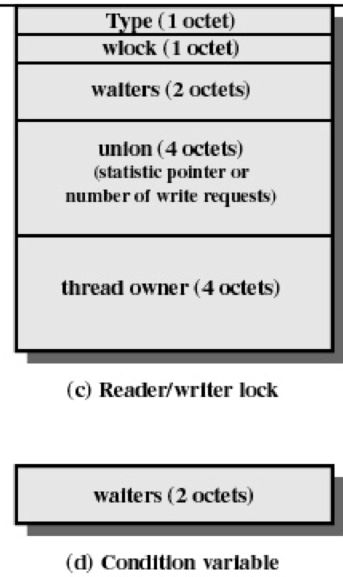


Figure 6.13 Solaris Synchronization Data Structures

Windows 2000 Concurrency Mechanisms

- Process
- Thread
- File
- Console input
- File change notification
- Mutex
- Semaphore
- Event
- Waitable timer

32