

Deadlock

Deadlock

How to Prevent It

Nick Urbanik <nicku(at)nicku.org>

Copyright Conditions: GNU FDL (see <http://www.gnu.org/licenses/fdl.html>)

A computing department

What is deadlock? How do we prevent it?

Reference:

William Stallings, *Operating Systems and Design Principles*, 4th Edition, 2001, Chapter 6

What is deadlock?

OSSI — Deadlock — ver. 1.1 — p. 1/26

- A set of processes or threads is in deadlock if:
- Each process is waiting for something that can only be offered by another process in the set.
- The set of processes is stuck
- To the user, they appear to have hung

OSSI — Deadlock — ver. 1.1 — p. 3/26

Concurrent Systems

OSSI — Deadlock — ver. 1.1 — p. 2/26

- A *concurrent system* is one where more than one process or thread is executing at the same time
 - I.e., is running or is ready to run
- Examples:
 - operating system
 - multiprocessing application (e.g., Apache 1.3.x)
 - multithreaded application (e.g., Apache 2.x.x)
- Deadlock can occur in a concurrent system

OSSI — Deadlock — ver. 1.1 — p. 4/26

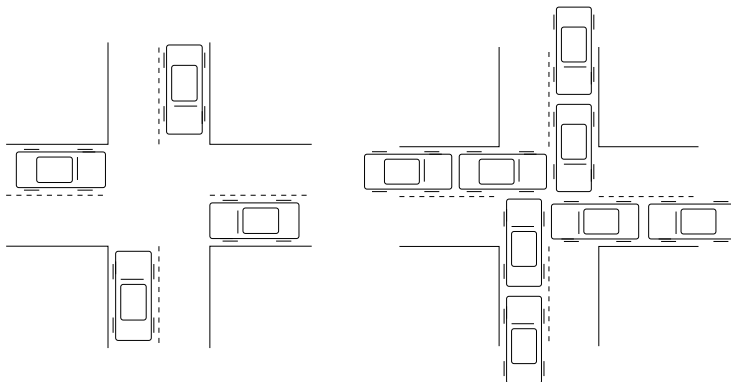
Starvation

- **Starvation** is a second danger for concurrent systems besides deadlock
- Involves a process not getting access to a resource because other processes are unfairly granted access
- We do not discuss starvation further here.

Gridlock on physical road

- Deadlock is like gridlock at an intersection
- Cars cannot move forward, because the space in front is occupied by another car
- Cannot move back
- Very similar to deadlock in OS

Gridlock — 2



Requirements for Deadlock

- Mutual exclusion
 - only one process can use a resource at one time
 - result of locking access to the resource
- Hold and Wait
 - Processes in the set holding resources given earlier can request new resources
- No Preemption
 - Resources given to process cannot be taken away forcibly by OS or other process
 - the process needs to surrender the resource itself
- Circular Wait
 - Each process is waiting for a resource held by another process in the set (the actual deadlock)

Deadlock Avoidance

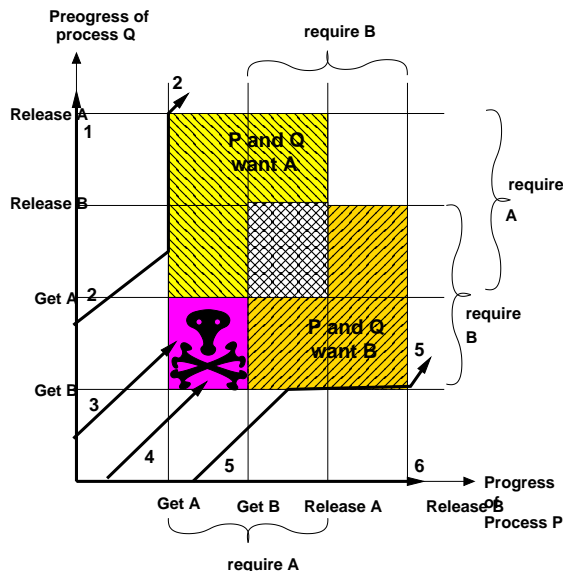
- The first three conditions are necessary for deadlock to occur
- The fourth condition can result because the first three are true
- A set of processes can reach an *unsafe* state where deadlock is possible
- *Deadlock avoidance* involves detecting unsafe states and not allocating resources that would cause an unsafe state
 - We do not investigate deadlock avoidance here.
- Need balance cost of deadlock against cost of preventing it

Two processes that can deadlock

Process P	Process Q
• • •	• • •
Get A	Get B
• • •	• • •
Get B	Get A
• • •	• • •
Release A	Release B
• • •	• • •
Release B	Release A
• • •	• • •

Deadlock Example — 2

OSSI — Deadlock — ver. 1.1 — p. 9/26



1. Only Q executes.
2. Q gets B, then A. P blocks waiting for A, resumes after Q releases A
3. P gets A, then B. Q blocks waiting for B, resumes after P releases B
4. Only P executes.
5. P gets A, then B. Q blocks waiting for B, resumes after P releases B
6. Only Q executes.

OSSI — Deadlock — ver. 1.1 — p. 11/26

Deadlock Example — 3

OSSI — Deadlock — ver. 1.1 — p. 10/26

- Path 1: only Q executes, no deadlock
- Path 6: only P executes, no deadlock
- Path 2: Q gets B, then A. P executes, blocks waiting for A, resumes after Q releases A. No deadlock.
- Path 5: P gets A, then B. Q executes, blocks waiting for B, resumes after P releases B. No deadlock.

OSSI — Deadlock — ver. 1.1 — p. 12/26

Deadlock Example — 4

- No problem if only P or Q executing
- No problem if Q gets B then A, P executes, but blocks waiting for A. Q releases A and B. P can then run OK
- No problem if either process gets both resources before the other starts.

Deadlock Example — 5

- But if:
- Q gets B, then P gets A (**Path 3**), or
- P gets A, then Q gets B (**Path 4**)
- Deadlock must happen since both will block waiting for the other.

How to prevent deadlock?

OSSI — Deadlock — ver. 1.1 — p. 13/26

- Two main methods:
 - Indirect method: prevent one of the first three conditions
- Direct method:
 - Prevent the last condition.
- All methods of prevention may have some **cost** in
 - execution time, or
 - more limited access to resources
 - design and programming time

Indirect: preventing mutual exclusion

OSSI — Deadlock — ver. 1.1 — p. 14/26

- This condition depends on the nature of the resource.
- If resource must be locked, then the OS must support mutual exclusion.
 - If concurrent processes share data, there is a **danger of data corruption**
 - i.e., two or more threads both write to same file at the same time

Indirect: prevent hold & wait — 1

- Process does not proceed until allocated all resources it will ever need.
- Wasteful, since:
 - process may wait much longer for all resources rather than enough to start with.
 - Resources locked while not being used.

Indirect: prevent hold & wait — 2

- Another way to prevent hold and wait:
- Process holds only one resource at one time
- Example: modify P as shown
- Note no deadlock possible even if do not change Q

Indirect: prevent hold & wait — 3

OSSI — Deadlock — ver. 1.1 — p. 17/26

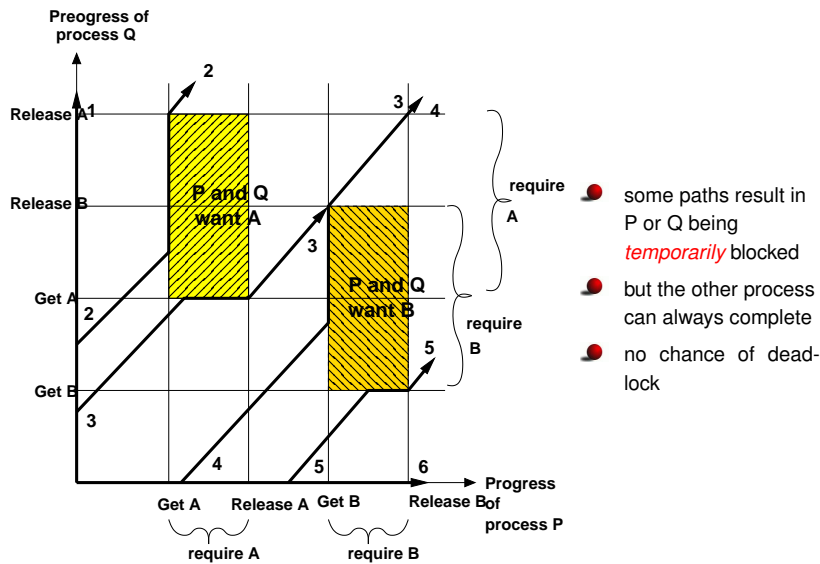
- Another strategy is for a thread or process to test if additional resources are available before waiting for them
- If any resource is not available,
 - then release all resources,
 - yield the CPU and then try again
 - yield = give up, i.e, thread voluntarily goes to the end of the scheduler queue for threads of its own priority
 - means another thread gets the CPU instead
- See `backoff.c` from *Programming with POSIX Threads*, pp. 67–69

Indirect: prevent hold & wait 3

OSSI — Deadlock — ver. 1.1 — p. 18/26

Process P	Process Q
...	...
Get A	Get B
...	...
Release A	Get A
...	...
Get B	Release B
...	...
Release B	Release A
...	...

Indirect: prevent hold & wait 4



OSSI — Deadlock — ver. 1.1 — p. 21/26

Direct: prevent circular wait

- Define an order in which resources are always requested
- For example, in previous example, if always allocate A then B, no deadlock can occur.
- This method is a part of strategy used in Linux and Windows operating system design

OSSI — Deadlock — ver. 1.1 — p. 23/26

Indirect: allow preemption

- OS or concurrent application could order a hierarchy of processes
- Highest priority could always get resources used by a lower priority process.
- Drawback: must be able to resume the preempted task at the point where the resource was taken away.

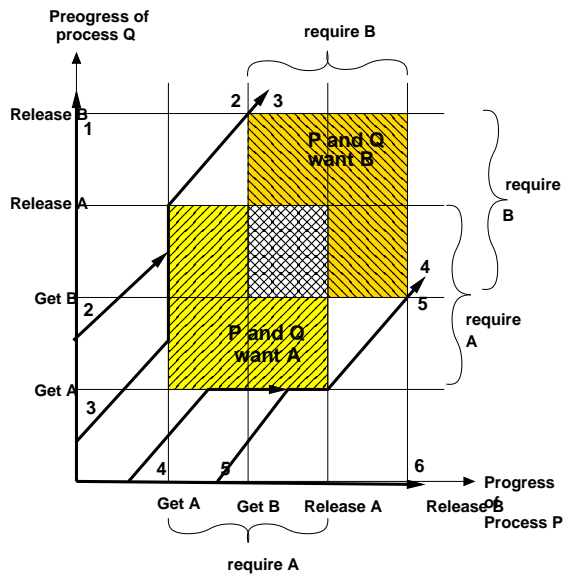
OSSI — Deadlock — ver. 1.1 — p. 22/26

Direct: prevent circular wait — 2

Process P	Process Q
...	...
Get A	Get A
...	...
Get B	Get B
...	...
Release A	Release A
...	...
Release B	Release B
...	...

OSSI — Deadlock — ver. 1.1 — p. 24/26

Direct: prevent circular wait — 3



- some paths result in P or Q being *temporarily* blocked
- but the other process can always complete
- again, no possibility of deadlock

Conclusion

- Deadlock is very undesirable
- Occurs within a set of processes or threads
- The processes “lock up”, each process waiting for the other.
- 4 conditions *all* required for deadlock
- deadlock avoidance detects and avoids *unsafe* states
- Prevention involves removing/preventing *one* or more of these conditions