Deadlock

Godmar Back

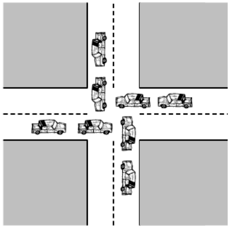
Virginia Tech

October 15, 2024

Deadlock

General Definition

A condition in which one or more related threads are blocked waiting for an event that will never occur because the blocked threads would be the ones to cause it. Resource version: ... threads are blocked waiting for resources that will never be granted because they are held by threads currently requesting resources.


(b) Deadlock

- Can be related to *resource contention* or to *lack of signaling or communication.*
- Typical for deadlock is that
  - (a) threads cannot make forward progress
  - (b) threads cannot easily back out
- Different from a system being merely idle in which all threads are blocked, but outside events will eventually unblock them

Resource Deadlock Detection

- Will focus on deadlocks involving reusable resources (e.g., mutexes)
- Reliable after-the-fact deadlock detection requires access to resource allocation graph:
  - Nodes are either processes or resources with 2 types of edges
  - From resource R_i to process P_k: process P_k holds resource R_i
  - From process P_k to resource R_i: process P_k is trying to acquire resource R_i
- In practice, finding this graph can be difficult, though some debuggers provide it, e.g. Windows [URL]

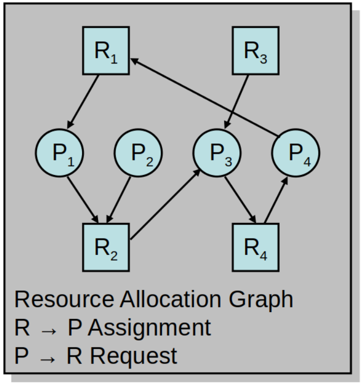


Figure 2: Resource Allocation Graph

Resource Deadlock

Coffman et al [1] identified the following four conditions for resource-related deadlock to occur

1. Exclusive Access: a resource is held exclusively by one thread
2. Hold and Wait: threads hold one resource while waiting to acquire another
3. No preemption: access to resources cannot be revoked, not even temporarily
4. Circular Wait: there is a cycle in the resource allocation graph

Observation

Conditions (1) to (3) are conditions we fundamentally require: e.g. locks provide exclusive, non-preemptible access to something, and it may be required to acquire multiple of them in sequence.

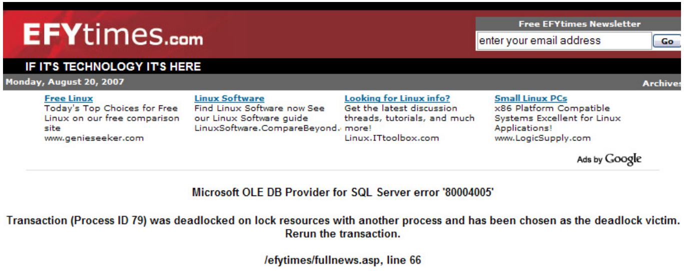
Strategies for Dealing with Deadlock

- Deadlock Recovery, e.g. after the fact
- Deadlock Prevention, e.g. remove one of the necessary conditions
- Deadlock Avoidance, e.g. adopt a strategy if none of the necessary conditions can be removed

Deadlock Recovery

In order of increasing severity

- Preempt access to resource (if possible)
- Back processes up: expensive, requires checkpointing and/or transaction mechanisms
- Kill involved processes until deadlock is resolved
- Kill all threads/processes involved
- Reboot


EFYtimes.com
Free EFYtimes Newsletter
enter your email address
Go
IF IT'S TECHNOLOGY IT'S HERE
Monday, August 20, 2007
Archives
Free Linux
Today's Top Choices for Free Linux on our free comparison site
www.genieseeker.com
Linux Software
Find Linux Software now See our Linux Software guide
LinuxSoftware.CompareBeyond.
Looking for Linux info?
Get the latest discussion threads, tutorials, and much more!
Linux.ITtoolbox.com
Small Linux PCs
x86 Platform Compatible Systems Excellent for Linux Applications!
www.LogicSupply.com
Ads by Google

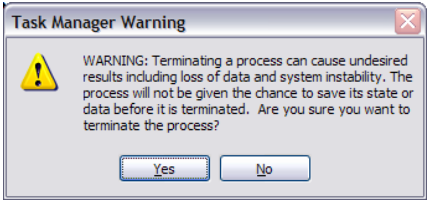
```
Microsoft OLE DB Provider for SQL Server error '80004005'

Transaction (Process ID 79) was deadlocked on lock resources with another process and has been chosen as the deadlock victim.
Rerun the transaction.

/efytimes/fullnews.asp, line 66
```

Side Note on Safe Termination

- Killing threads or processes (as part of deadlock recovery, or otherwise) is generally tricky
- Must avoid a situation where killed thread is in the middle of manipulating state that is accessed later
  - E.g., kill -9 kills an entire process, including all its state. Kernel ensures that no kernel state is endangered; but provides no guarantees about the state of resources such as files
  - Killing individual threads (i.e., via `pthread_cancel()` is not recommended/tricky to get right)


Task Manager Warning

WARNING: Terminating a process can cause undesired results including loss of data and system instability. The process will not be given the chance to save its state or data before it is terminated. Are you sure you want to terminate the process?

Deadlock Prevention

Idea

Deadlock cannot occur if one of the necessary conditions is removed

- (C1): (do not require) Exclusive Access:
  - if resources can be shared or duplicated, do so
- (C2): (avoid) Hold and Wait strategies:
  - Request all resources at once in a single operation – may be difficult to know in a modular system
  - Exploit `try_acquire` operation and drop resources already acquired upon failure, then retry – can be inefficient if resources are contended

Deadlock Prevention (cont'd)

Idea

Deadlock cannot occur if one of the necessary conditions is removed

- (C3): (allow for) Preemption:
  - Preempt access to resource - difficult to write code that is robust in the presence of such preemption
  - Virtualize resource (save and restore)
- (C4): (avoid) Circular Wait:
  - Create a partial order of all resources that may be held simultaneously - e.g., by taking their addresses; example: C++17 std::scoped_lock
  - Real-world systems often document locking order

Deadlock Avoidance

- If none of the necessary 4 conditions can be removed, can consider a strategy whether to allow or deny requested accesses to resources
- E.g. Banker's algorithm
  - Avoids "unsafe" states that might lead to deadlock
  - Requires knowledge of future resource demands
  - Requires capturing of all dependencies
- By and large, mostly theoretical and not used in practice

Practical Strategies

- Minimize likelihood of deadlock by applying prevention strategies wherever possible:
  - avoid unnecessarily fine-grained locking (share a lock)
  - define locking order if not possible
  - use tools that flag when locking order is violated
  - have clear signaling strategies
- Allow for deadlock recovery
  - Design system to minimize the amount of work that is lost or must be repeated if deadlock recovery necessitates killing of processes

Deadlock vs. Starvation

Starvation

Apparent lack of progress that could be fixed with a proper scheduling strategy:

- Strict priority scheduler might starve lower priority thread if higher priority threads are always READY
- Reader-writer locks may assign lock to only readers, starving writers

Deadlock

There is no scheduling policy that would allow forward progress

See Levine [2] for an attempt to extend the definition of deadlock to other lack of progress states

Conclusion

- Deadlock is a state where a set of threads is blocked waiting for a resource or event that could be produced only by a thread in the set
- For reusable resources, can be analyzed with a resource allocation graph
- Employ strategies for
  - Deadlock Detection & Recovery
  - Deadlock Prevention
- In general, risk of deadlock increases with finer granularity of locking: scalability vs robustness trade-off

References

[1] E. G. Coffman, M. Elphick, and A. Shoshani.
System deadlocks.
ACM Comput. Surv., 3(2):67–78, June 1971.

[2] Gertrude Neuman Levine.
Defining deadlock.
SIGOPS Oper. Syst. Rev., 37(1):54–64, January 2003.