Operating Systems W6L2 - Deadlocks (ctd)

Class	Operating Systems
Created	@Oct 7, 2020 10:10 PM
Materials	
Reviewed	☐
Type	Lecture

Checking for Deadlocks Recap

- What are the 3 methods?
    1. Check every time a resource is requested
    2. Every k minutes
    3. When CPU utilization has dropped below a threshold
- Threshold checking is dynamic, based on things like history and whatnot
- Performance counters are per core

Recovery From Deadlock

- 3 options once a deadlock is detected
    1. Preemption
    2. Rollback
    3. Killing processes

Preemptions

- Temporarily reallocate a resource
- Requirement of manual intervention (i.e. printer)
- Varies a lot on nature of the resource
- All in all, preemptive recovery is frequently impossible → Taking a resource means things may crash (but not as expensive)

Rollback

- Have processes *checkpointed* frequently
- ▼ What is the checkpoint of a process?

  State written to a file so that it can be restarted later
- In deadlocks, a process requiring a needed resource is rolled back to a point prior to that resource being acquired

Killing

- Kill processes in the chain until deadlock is resolved
- A "victim" can also be a process thati s not in this chain
- Part of OS design is simply killing as few processes possible

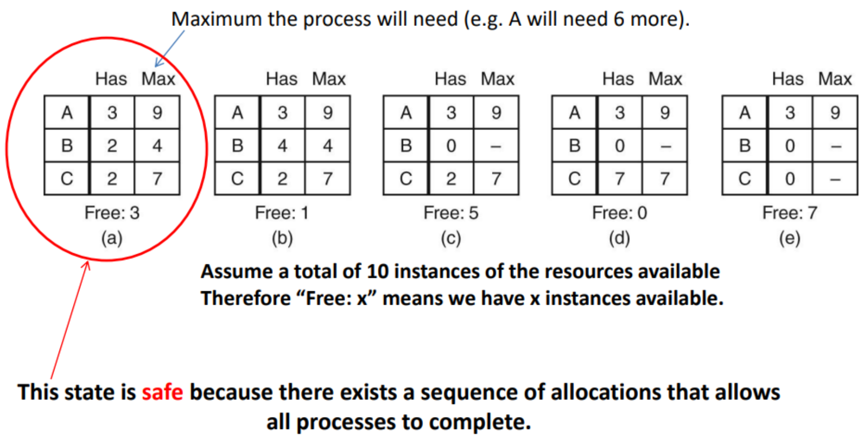
Dynamic Avoidance

- Resources are requested one at a time (in most systems) and a resource is only granted if it is **safe** to do so
  - Safe as in it won't cause a deadlock
- ▼ What is required for a state to be safe?

  A state is safe if there is one scheduling order in which every process can run to completion even if all of them suddenly request their maximum number of resources immediately. *An unsafe state is not a deadlock state.*

Safe vs. Unsafe

- Allocation Diagram (`a` is safe, `b, c, d` are unsafe)



Banker's Algorithm

- If a request leads to a safe state, grant access to resources (Dijstrka 1965)
- What is the main idea of this algorithm?

  The algorithm checks to see if it has enough resources, i.e. a bank has enough money for a loan. If all resources can be distributed, or reused and recycled, then the state is marked as safe.

- Example Diagrams

Process	Tape drives	Plotters	Printers	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	Printers	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

- This algorithm is nice in theory, but in practice it is useless → Processes rarely know what the maxnimum number of resources needed, number of processes is not fixed, and resources can suddenly vanish

> ❗ Continuous mention here for the 4 deadlock conditions. **Can you recall them?**

Deadlock Prevention

- Deadlock avoidance is essentially impossible, but we can work on it by using the earlier conditions and seeing if at least one of them is not met

> ❗ Tackling each condition (from 4 above) one by one. See slides on NYU Classes for bullets.

- Sometimes things cannot be done, both on homeworks and quizzes but also in real life, so you just need to explain *why* and keep in mind a question may not have an answer

- And don't be misled by over information!

▼ Approach for the four conditions

Any one rule is enough to avoid a deadlock / You only need to violate one condition

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

Approach	Resource Allocation Policy	Different Schemes	Major Advantages	Major Disadvantages
Prevention	Conservative; undercommits resources	Requesting all resources at once	•Works well for processes that perform a single burst of activity •No preemption necessary	•Inefficient •Delays process initiation •Future resource requirements must be known by processes
		Preemption	•Convenient when applied to resources whose state can be saved and restored easily	•Preempts more often than necessary
		Resource ordering	•Feasible to enforce via compile-time checks •Needs no run-time computation since problem is solved in system design	•Disallows incremental resource requests
Avoidance	Midway between that of detection and prevention	Manipulate to find at least one safe path	•No preemption necessary	•Future resource requirements must be known by OS •Processes can be blocked for long periods
Detection	Very liberal; requested resources are granted where possible	Invoke periodically to test for deadlock	•Never delays process initiation •Facilitates online handling	•Inherent preemption losses

Conclusions

- This wraps up both race conditions and deadlocks
- Be careful around words like "always"
- This also wraps up the idea of virtualization, in case you've been following along with the textbook

---

🎉 **THIS IS THE END TO THE MIDTERM MATERIAL!** The midterm will not contain anything beyond this point.