CIS 452 - Operating Systems Concepts

Nathan Bowman

Images taken from Silberschatz book

Deadlock

Deadlock occurs when concurrent processes are waiting on one another in such a way that none of them can ever make progress again

We have already seen some examples of deadlock, such as in our first attempt to solve the Dining Philosophers problem

We are going to learn more about what deadlock is, how it occurs, and how to avoid it

“When two trains approach each other at a crossing, both shall come to a full stop and neither shall start up again until the other has gone.” ~ Law in Kansas in Early 20th century, as quoted in your textbook

Studying deadlock requires a model of system resources

System resources are finite and divided into classes, e.g.:

- 4 printers
- 2 CPUs
- 1 disk drive

Any resources within the same class are indistinguishable to the program from one another (if

Processes access resources in a request-use-release sequence

Request: request resource and wait if unavailable

Use: operate on resource

Release: notify system that process is done with resource

Request and release generally must be system calls because resource allocation is controlled by OS

- `open()/close()`, `malloc()/free()`, etc.

If resource is not managed by OS, semaphore can be used

- `wait()/signal()`

"A set of processes is in a deadlocked state when every process in the set is waiting for an event that can be caused only by another process in the set." ~ your textbook

Process A owns resource X and needs Y

Process B owns resource Y and needs X

Characterizing deadlock

Necessary conditions for deadlock

- mutual exclusion
- hold and wait
- no preemption
- circular wait

As we discuss these, think about how they applied in the Dining Philosophers problem

Mutual exclusion -- at least one resource is held in *non-sharable* way

Hold and wait -- process must be holding a resource and waiting to acquire resources held by other processes

No preemption -- resource can be released only voluntarily by process holding it

Circular wait -- Set `{P0, P1, ..., Pn}` of waiting processes exists such that

- `P0 is waiting for resource held by P1`
- `P1 is waiting for resource held by P2`
- …
- `Pn is waiting for resource held by P0`

These four conditions are *necessary* for deadlock to occur

Stopping any one of them makes deadlock impossible

Example deadlock

```
A                   B

acquire(lock1)      acquire(lock2)
acquire(lock2)      acquire(lock1)

...                 ...


release(lock1)      release(lock2)
release(lock2)      release(lock1)
```

We will see a visual way of checking for deadlocks next time