Deadlock

Deadlock

Starvation $\underset{\xleftarrow{O}}{\xrightarrow{X}}$ Deadlock

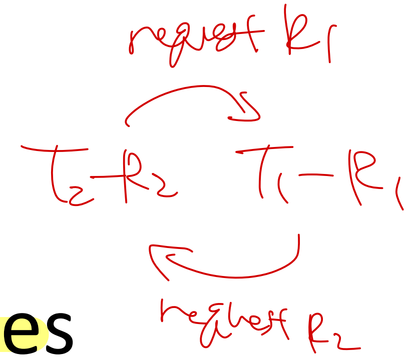
- **Synchronization is a live gun**
  - We can easily shoot ourselves in the foot 자기발등을 찍다.
  - Incorrect use of synchronization can block all processes
- **If one process tries to access a resource that the other process holds, and vice-versa, they can never make progress**
- **We call this situation deadlock, and we'll look at:**
  - Definition and conditions necessary for deadlock
  - Representation of deadlock conditions
  - Approaches to dealing with deadlock

Deadlock Definition

- Resource: any (passive) thing needed by a thread to do its job (CPU, disk space, memory, lock)
  - – Preemptable: can be taken away by OS
  - – Non-preemptable: must leave with thread

- Starvation: thread waits indefinitely

- Deadlock: circular waiting for resources

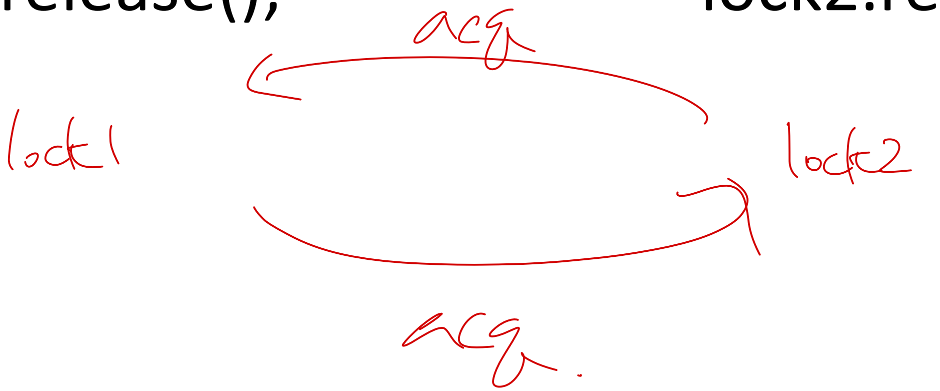


Example: two locks

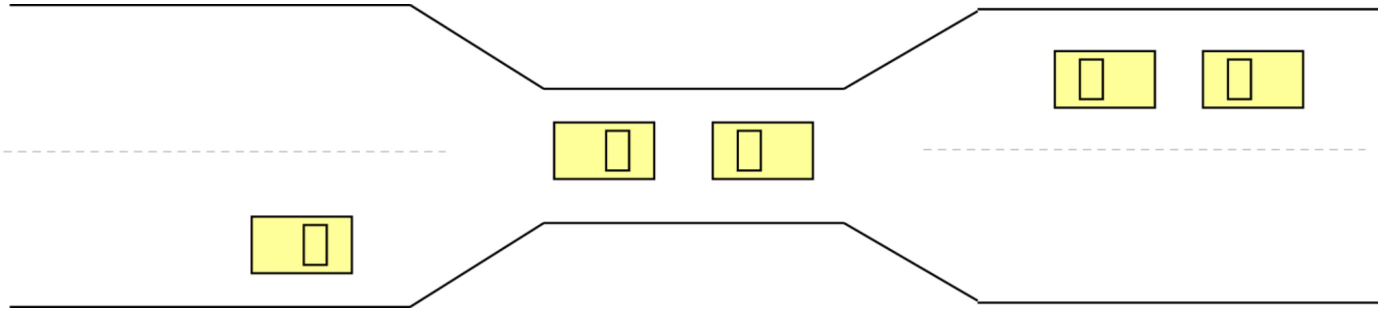
Circular waiting

Thread A	Thread B
lock1.acquire();	lock2.acquire();
lock2.acquire();	lock1.acquire();
lock2.release();	lock1.release();
lock1.release();	lock2.release();

critical section.



Deadlocks in real world



- **Real issue is *resources* & how required**
- **E.g., bridge only allows traffic in one direction**
  - Each section of a bridge can be viewed as a resource.
  - If a deadlock occurs, it can be resolved if one car backs up (preempt resources and rollback).
  - Several cars may have to be backed up if a deadlock occurs.
  - Starvation is possible.

Necessary Conditions for Deadlock

- Limited access to resources
  - If infinite resources, no deadlock!
- No preemption
  - Once a thread acquires a resource, its ownership cannot be revoked until the thread releases it
- Wait while holding
  - One process holding one resource and wait for another resource
  - When requests are multiple and independent
- Circular waiting
  - There is a set of waiting thread such that each thread is waiting for a resource held by another

Prevent by eliminating one condition

- Deadlock conditions
  - Limited access to resources
  - No preemption
  - Wait while holding
  - Circular waiting

- All of 1-4 conditions are necessary for deadlock

- Two approaches to deal with deadlock:
  - Pro-active: make the conditions not occur → 예방
  - Reactive: When deadlock happens, do a corrective action
    → 사후처리

How to avoid deadlock?

- Limited access to resources
  - Virtualize resource -> 자원이 복수개 있다고 착각하게 해주면 됨.
  - Threads contend to use CPU registers and they can be context-switched out while using some registers
    - Copy of the registers to memory and let the registers to be used by other threads

- No preemption
  - Simply allowing preempt using resource
  - Virtual memory: virtual address space is allocated to a process but physical memory can be taken away by OS

How to avoid deadlock?

- Wait while holding
  - Wait until holding all resources at once (must know what resources can be used in advance

- Circular waiting
  - Single lock for entire system: problem?
  - **Lock ordering**: always acquire locks in a fixed order

Example: two locks

1 -> 2 lock order

Thread A

```
lock1.acquire();
lock2.acquire();
lock2.release();
lock1.release();
```

Thread B

```
Lock1.acquire();
Lock2.acquire();
Lock2.release();
lock1.release();
```

Deadlock recovery

- **Once a deadlock is detected, we have two options…**

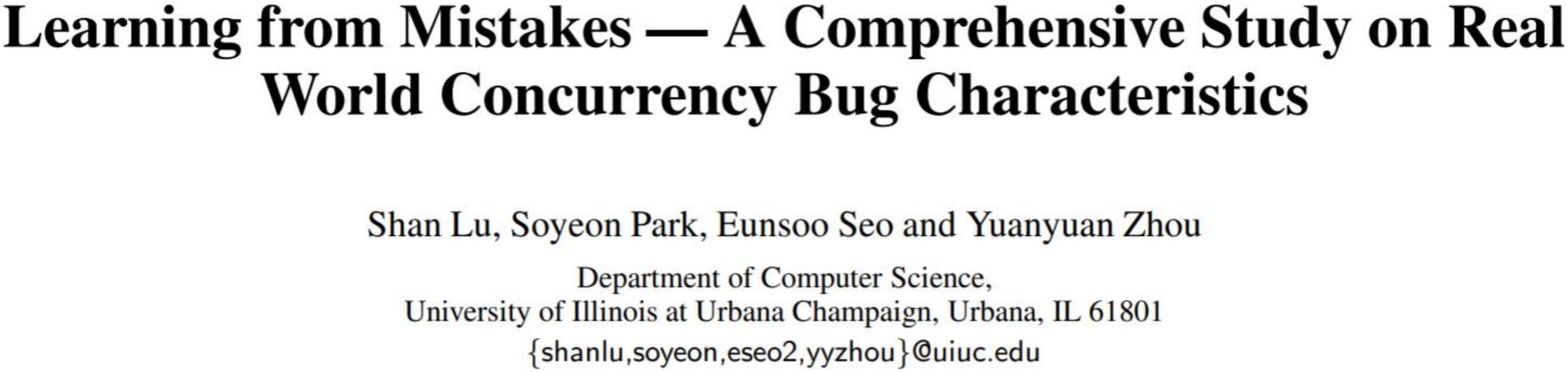
1. Abort processes → 죽이기

- Abort all deadlocked processes
  - Processes need to start over again
- Abort one process at a time until cycle is eliminated
  - System needs to rerun detection after each abort

2. Preempt resources (force their release) → 자원 뺏기

- Need to select process and resource to preempt
- Need to rollback process to previous state
- Need to prevent starvation

For knowledge-hungry students



Learning from Mistakes — A Comprehensive Study on Real World Concurrency Bug Characteristics

Shan Lu, Soyeon Park, Eunsoo Seo and Yuanyuan Zhou

Department of Computer Science,
University of Illinois at Urbana Champaign, Urbana, IL 61801
{shanlu,soyeon,eseo2,yyzhou}@uiuc.edu