CS 3214

thread intro
locks (mutexes)
signal/wait condition variables
bounded buffer aka producer/consumer

→ mutex ←
→ cv ←

Today:
- Semaphores
- more issues / concurrency problems deadlock
- atomics
- performance consideration

1965 Dijkstra
1. a counter (<=0 then wait)
2. a set of waiting threads

- counter = # waiting threads

"P" prolaag ["try decrease"] / down "wait"
"V" verhoog ["increase"] / up "post"

Mutual exclusion w/ semaphores

{ { { } { }

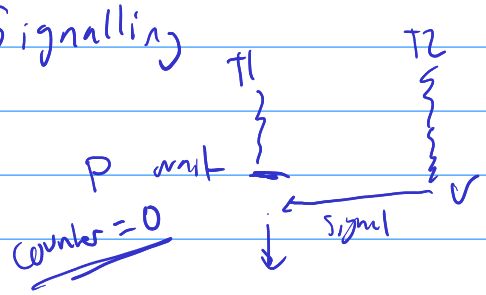
P counter = 1

critical section [} { counter = 0 → 

V counter = 1

Binary Semaphore = Mutual exclusion

Signalling



CV vs. semaphore

semaphore: wait/post always match

Semaphore: good if only protected state for condition is counter

cv: have to have associated mutex

cv: good for complex conditions (state)

Concurrency problems

1. Atomicity violation

T1: [if (thd->procinfo) fputs(thd->procinfo)] atomic

T2: [thd->procinfo = NULL;

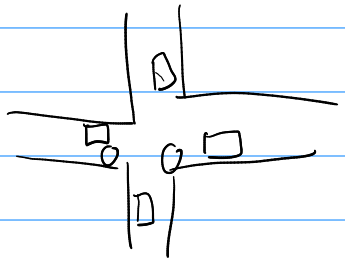
97%

2. Order violation

T1 | void init() {
mThread = createThread(...)

T2: | void mMain(){
mState = mThread->state;

3. Deadlock "the deadly embrace"



Conditions for deadlock

1. circular wait
2. hold & wait
3. exclusive access (mutex)
4. no preemption (can't forcibly remove locks)

- prevention
- avoidance
- recovery
- ignoring

Prevention:

- lock-free data structures (atomic CAS / TAS)
  ✗ [exclusive access]
- ordered lock acquisition ← used most in practice
  ✗ [circular wait]
- trylock + get locks all at once → livelock
  ✗ [hold & wait]
-

Deadlock avoidance: Dijkstra: smart scheduler

Banker's Algorithm (not used much in practice)

	T1	T2	T3	T4
L1	y	y	n	n
L2	y	y	y	n

Recovery: keep track of "waits for"

if detect cycle:

- preempt
- kill process
- kill all processes
- reboot