CS3214 lecture #17 "deadlock"

Project due tomorrow
Test 2 29-30

multithreading threads vs procecrs
locking / mutexes
signaling / condition variables
Semaphores - internal counter P

Concurrency Problems

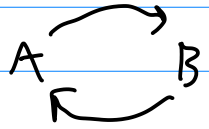
97% { 1. atomicity violation ← mutex
2. order violations ← signaling

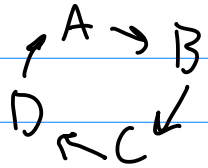
T1: init() mthread = RR_CreateThread(nMain,...)
T2: nMain() mState = mthread → state

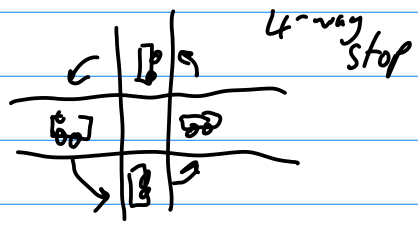
state
P
mutex_lock
while(look at counter)
sleep
mutex_unlock
V

3. deadlock
Dijkstra: "the deady embrace"

everyone waits for something the other has
none release until they get what they're waiting for







s: from: bob
to: alice

t: from: alice
to: bob

bob
alice

deadlock!

Conditions for deadlock to occur

→ 1. exclusive access (mutex)
2. hold + wait (thread holds resources while waiting for others)
→ 3. no preemption (can't forceably remove resources)
4. circular wait

Solutions
- prevent
- avoid
- recover
- ignore

atomic [if(var=0) var=1...]

cmpxchg

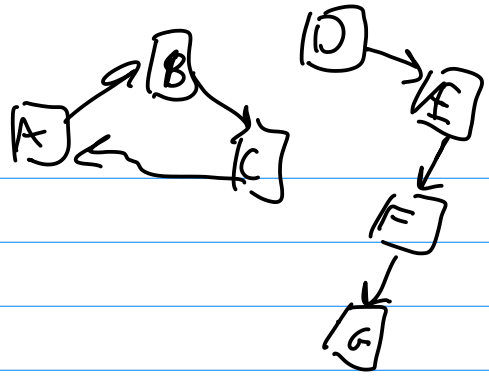
Prevention: remove a condition
1. exclusive access (mutex) - lock free data structures
"atomics" primitives: Compare + swap (CAS), test + set (TAS)
2. hold + wait: try to grab all locks (mutex_trylock)
if you don't get all, release all
difficult (encapsulation)
livelock
3. no preemption: not really feasible
4. circular wait: ordering locks (most often used)

Deadlock Avoidance
"smart scheduling"

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

don't run T1, T2 at same time

Dijkstra: "Banker's Algorithm"
not used much in practice



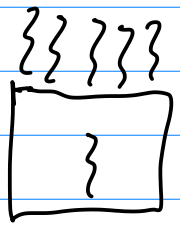
Deadlock Recovery

keep track of "waits for" graph

- preempt
- kill process → will something be lost
- kill all processes — backup state — DBMS
- Reboot — kernel ensures kernel endangered

deady embrace

Performance ← always after correctness



one big lock — Linux
Python GIL
gtk GUI

idea: split lock up

A, B, C

Dangers:

1. Atomicity violation: A, B must be independent!
2. deadlocks: locking order
3. more frequent lock/unlock overhead

futex — fast path (user) atomic cmpxchg
no mode switch!
slow path: (kernel)

Correctness is crucial