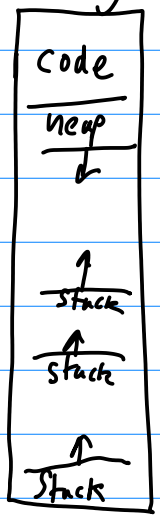
CS 3214 lecture #14 locking

ex3 : tomorrow!
p2: soon!

Threads

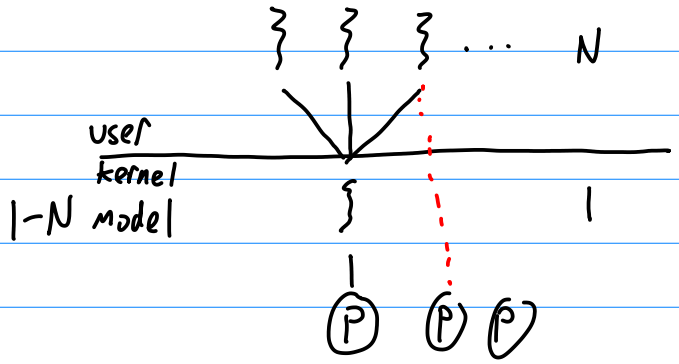
/proc/maps

multi threading



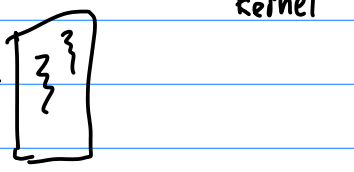
each thread has own
- CPU regs (PC, sp...)
- stack

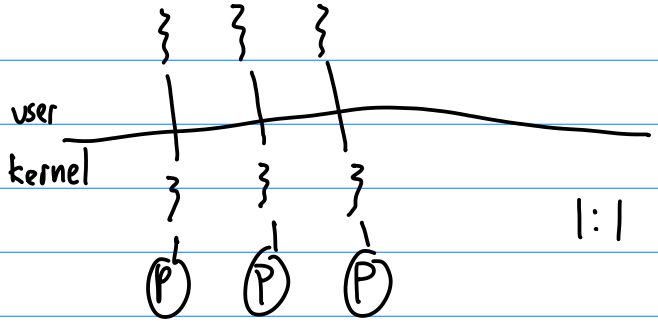
Do you need kernel support? NO
- user-threads
  - cooperative threads
    + ctxt switches are fast



- kernel-supported threads
  + multiple CPUs
  + preemption
  + blocking

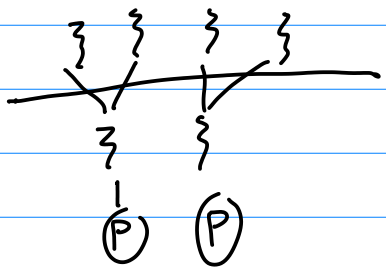
process
- address space
- threads





1:1 threading model

today:
1:1 threading model
- futex (fast user mutex)

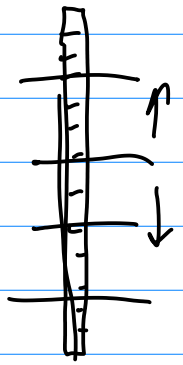


M:N threading model
- Solaris
- windows fibers

pthreads: POSIX threads

1 process
multiple threads

if process exits, all threads are cleaned up



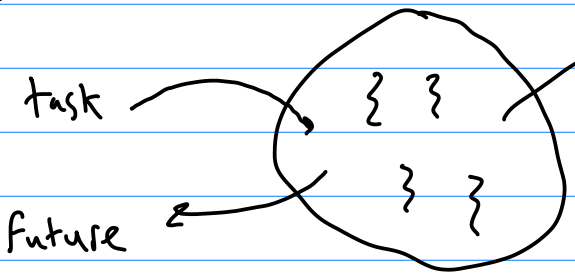
what size tasks?

#threads >> #CPUs

→ too many threads - switching overhead
→ too few threads - not making use of all resources

thread pools

one abstraction over thread pool "futures"



READY + RUNNING ≈ #CPUs

future.get blocks until task complete

Data races
- shared variables
- results depend on execution order

mov_
add_
mov

not atomic

"critical section"

"evil scheduler"

intermittent

How do you solve it?

avoid sharing
- CPU ready queue
- malloc (region-based)

management cost

"lock" the data → 1 thread can access while locked