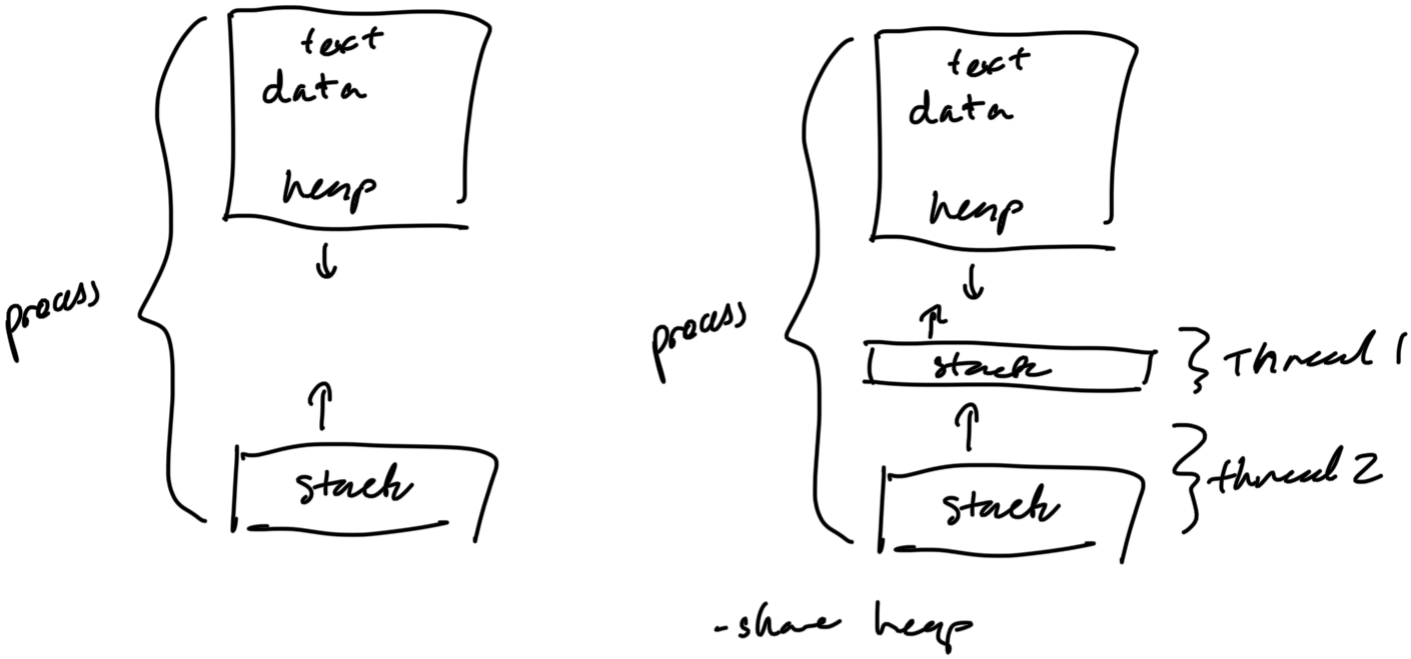
Multithreading 3/13/23

1 process per CPU core

a process has 1+ threads
Threads in process run concurrently, share memory



-share heap

Threads
- faster to create & destroy
- web server can do stuff while waiting for I/O

easier to switch between threads vs processes

POSIX thread library
<pthread.h>
create - new thread
join - wait for thread to complete, joins 2 threads
-works for UNIX & windows

thread priority
- scheduling
- higher, executed sooner

Ex) 1 CPU / thread

Context switching: save program state. Faster w/ thread

operations are not atomic aka no one point @ same time
& locations as another
"race conditions"

to do so, you can emulate atomic memory w/ software
Critical regions:
- 1 critical region @ a time, prevents Thread from working on data @ same time
- semaphore / mutex

use mutex to lock threads
enter critical region. lock thread, as you leave, unlock

Deadlock
- 2 processes mutex 2 things @ same time, each hold 1 & refuse to give up
- to prevent, need rules, either get 0 or 2 resources

2 kinds of resources
- preemptable: can be taken away w/ no issue (pencil)
- non preemptable: bad stuff happens (sandwich)

you can make a graph of resources to keep track

deadlocks are finicky & time dependant
- insidious to find
- timing dependant

Lots of professional programmers just pretend it doesn't exist
- abundant resources, less likely to happen
  - not optimal technically

how to recover from deadlock?
- kill processes
- roll back: restore program previous state, have to save program state

Deadlock detection
- OS handles

how to avoid deadlock
- queue up
- impose rules
- steal preemptable resources

livelock: scheduling algorithm bug
- priority queue bug
- events at low priority never executed bc high priority stuff keeps being generated