Computer Systems

Concepts to be covered:
- swapping
- pipes
- threads (partially)

SWAPPING

process swapped out

swapped io wait runnable

every once in a while
if a process is in IO wait for a long time
start swapping it
get input
if dest. is swapped
  start swapping it in

PIPEC

d. | grep x

write:
if full
  pipe.writer = self
  wait
append to buffer
if pipe.reader
  signal pipe.reader

read:
if empty
  pipe.reader = self
  wait
get data
if pipe.writer
  signal it

THREADS

- lightweight process
- parallelism
- robustness
- light context switching
- common data
- better resource allocation
- used for asynchronous operations
- share address space
Race Conditions

\[
\begin{align*}
\text{mov } x \rightarrow R1 & \quad x = x + 1 \\
R1++ & \\
\text{mov } R1 \rightarrow x & \\
\end{align*}
\]

- guard access to \( x \).

Critical Section Protection

- disable interrupts (uni process on only)
- atomic instructions
  
  ex: \text{ATOMIC INCR}
  
  \text{CMP, XCHG}
  
  \text{mutex, semaphore, monitors, rendezvous, events, spin lock, mutex-lock (my mutex) if mutex.locked is false}
  
  \text{mutex.unlock (---) set it}
  
  if \text{mutex.wait(1) wake first else}
p thread (posix thread)

pthread_mutex_lock(m)

while condition is not true
sleep_on(m, condvar)

mutex_unlock(m)

mutex_lock(m)
modify condition
signal_condvar(condvar)
mutex_unlock(m)

Recap:
threads + threads sync
pipes
[swapping]
process creation
I/O abstraction - file handles
command line as a program
scheduling: I/O wait, runnable
memory protection, users/supervisor mode
recap of context switching.