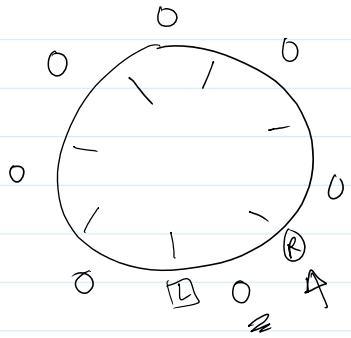
# Lecture 31

04 October 2018 22:23

synchronization & threads

0. semaphores.
   - locking
   - signaling & notification, co-ordination

1. dining philosophers problem.
   - 'n' philosophers — think! eat. — two spoons / forks
   - 'n' forks / spoons.




← top-view of a philosophers dining table.

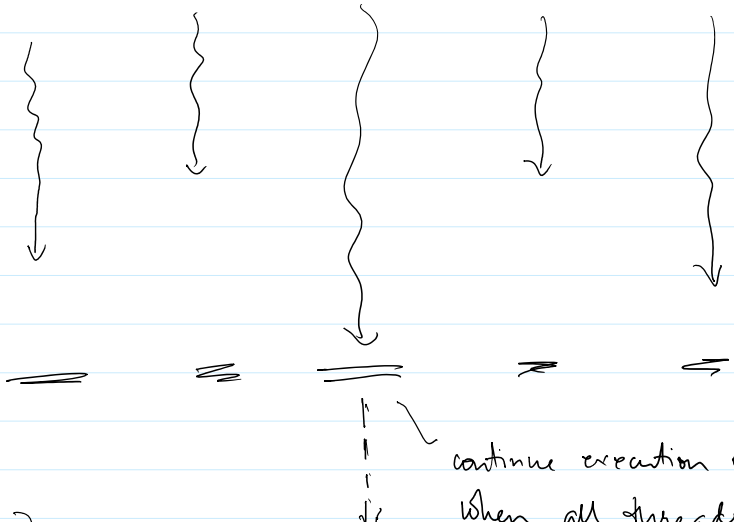
— need two spoons to eat.

all day do:

pick left fork()
pick right fork()
eat
put down right fork()
put down left fork()
think! ← non-deterministic

'n' philosopher picks right fork first!

2. barriers



H.W

— how to implement a barrier using C.Vs. & semaphores.

continue execution only when all threads

& semaphores.

continue execution only when all threads reach this pt. / condition.

barrier ( ___ );

(3) threads || process
threads

user-level threads
Kernel threads

pthreads
Kernel-level threads

multi-execution context.

juxtaposed with multi-process prog. vs. multi-threaded prog.

(4) thread : - share memory
- pc (program counter)
- stack
- register storage / space // context
- id.

|| execution entities of a process which

pthreads impln.
(light weight process)

user-level threads

- needs no Kernel support
- sees only one process
- user-level handling / multiplexing of <pc, stack, regs> to run multiple threads.

Kernel-level threads

- ~~fork~~ clone ( )

replicates the PCB of everything except.

run multiple threads.