PART 10: DEADLOCK & STARVATION

Starvation: when a task has higher priority over other tasks & never yields to allow other task to access the shared resources

↳ how to solve this?

① → make sure high priority tasks always yield some time to the processor → eg. waiting for a mutex/semaphore or through a delay function to put itself to sleep

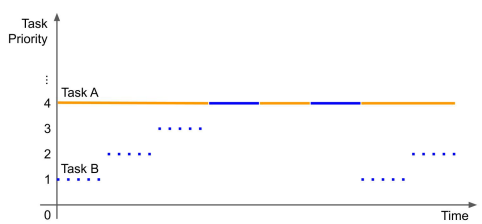
② → built into scheduler or rely on a higher priority task to monitor how long other tasks have been asleep

↳ if a lower priority has been asleep for some time, its priority will gradually increase until it can run.

↳ once it has run, its priority level is dropped to the original level

AGING

Aging



DEADLOCK → system comes to a halt as all threads are waiting circularly to be released

solution → ① give priority / hierarchy of value to the 'lock' (semaphore / mutex)

↳ the task must pick up the lowest priority

↳ prevents deadlock because the last task must wait for the lowest value lock before continuing with their task

② using 'arbitrator' → the lock is tied to a global mutex

→ a task must request access to the mutex which decreases its value — thus allows a task to run.

→ the mutex is not available to other tasks as long as the first task is still running

↓

solves deadlock, but doesn't allow tasks to run in parallel