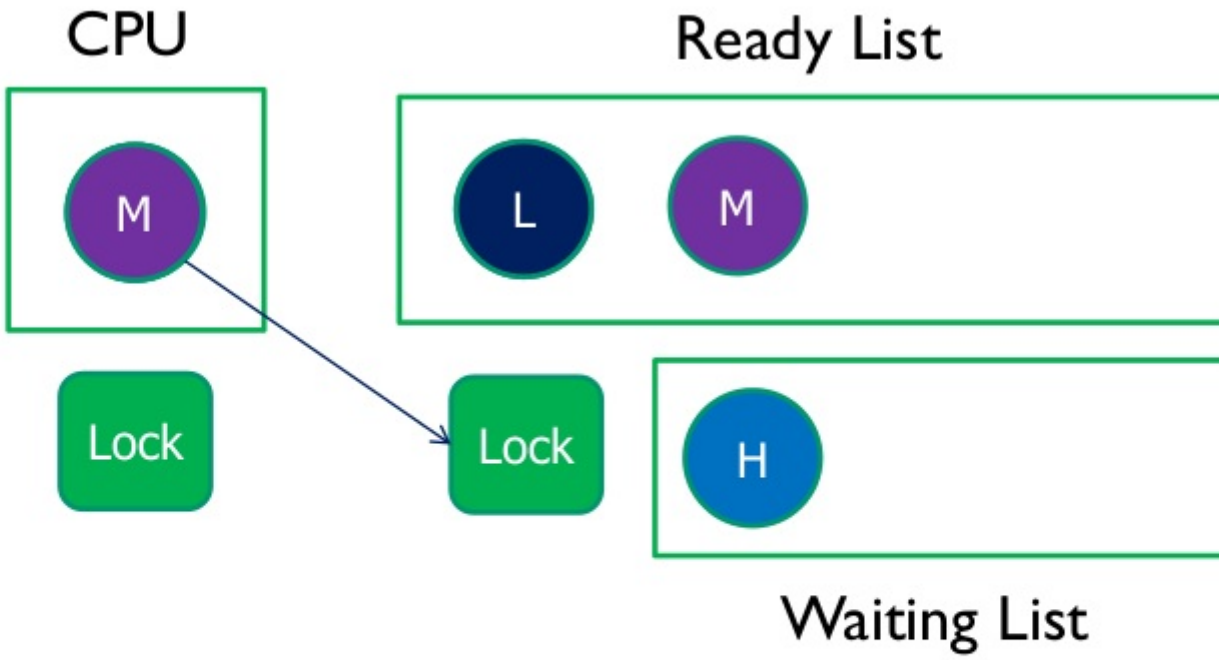
Priority Donating

Intruduction

Priority Inversion

One issue with priority scheduling is "priority inversion". Consider high, medium, and low priority threads H, M, and L, respectively. If H needs to wait for L (for instance, for a lock held by L), and M is on the ready list, then H will never get the CPU because the low priority thread will not get any CPU time. A partial fix for this problem is for H to "donate" its priority to L while L is holding the lock, then recall the donation once L releases (and thus H acquires) the lock.



- **Input order : L, H, M**
- **Execution order : M, L, H**

- Priority inversion problem

A situation where a higher-priority job is unable to run because a lower-priority job is holding a resource it needs,such as a lock.

- Solution: Priority Inheritance

The higher-priority job **donate** its priority to the lower-priority job holding the resource it requires (**effective priority**).

The beneficiary of the inheritance will now have a higher scheduling priority.

– Get scheduled to run sooner.

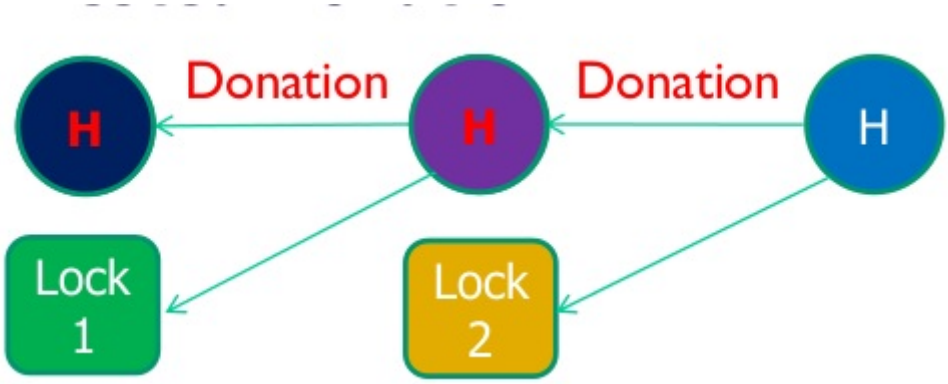
It can finish its work and release the resource, at which point to the original priority is returned to the job.

Priority Donating

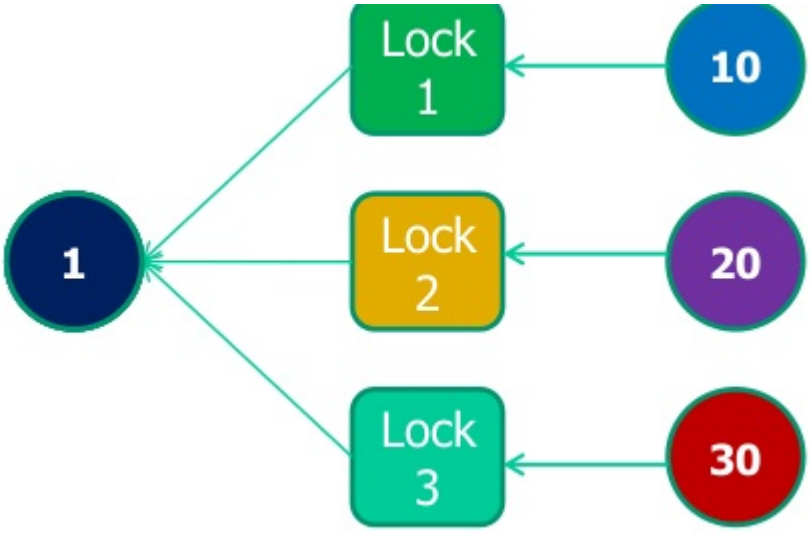
Implement priority donation. You will need to account for all different situations in which priority donation is required. Be sure to handle multiple donations, in which multiple priorities are donated to a single thread. You must also handle nested donation: if H is waiting on a lock that M holds and M is waiting on a lock that L holds, then both M and L should be boosted to H's priority. If necessary, you may impose a reasonable limit on depth of nested priority donation, such as 8 levels.

You must implement priority donation for locks. You need not implement priority donation for the other Pintos synchronization constructs. You do need to implement priority scheduling in all cases.

- Nested Donation



- Multiple Donation



Some suggestion:

Before you begin writing code, you should read all source code of synch.c and thread.c.

The function you need to rewrite:

lock_acquire()

lock_release()

sema_up()

sema_down()

thread_set_priority()

In thread_set_priority(), you need to consider three scenarios.

Situation 1: Current thread hasn't been donated. So you need to set both old_priority (Add in thread. It's use to record the original priority) and priority.

Situation 2: Current thread has been donated. But we need to set priority now. If the new priority is less than the priority that the thread get by donating. We only need to set old_priority.

Situation 3: Current thread has been donated. But it will be donated again. We needn't to change old_priority.

Maybe you need to add a new function to help thread_set_priority().

Requirement

You have two weeks to pass all following tests. But you need to hand in homework every week:

1.alarm-single

2.alarm-multiple

3.alarm-simultaneous

4.alarm-priority

5.alarm-zero

6.alarm-negative

7.priority-change

8.priority-donate-one

9.priority-donate-multiple

8.priority-donate-multiple2

10.priority-donate-nest

11.priority-donate-sema

12.priority-donate-lower

13.priority-fifo

14.priority-preempt

15.priority-sema

16.priority-donate-chain

Run `make check` in ***src/threads/*** to make sure your design can pass all these tests.

You should download and **fill** the [**DESIGNDOC**] and archive your code(**the whole pintos folder**) and DESIGNDOC into a **zip** file **named sid_nameInPinYin_vid.zip(e.g 12330441_zhangsan_v0.zip)** and upload to the ftp.