# Solution

## Problem 1: (21 points)

1. [1] [(9-0)+(15-2)+(20-6)+(22-10)]/4 = 12ms

   [2] [(0-0)+(9-2)+(15-6)+(20-10)]/4 = 6.5ms

   [3] [(9-0)+(14-6)+(16-10)+(22-2)]/4 = 10.75ms

   [4] [(0-0)+(9-6)+(14-10)+(16-2)]/4 = 5.25ms

   [5] [(8-2)+(12-10)+(15-6)+(22-0)]/4 = 9.75ms

   [6] [(0-0)+(2-2)+(8-6)+(10-10)]/4 = 0.5ms

   [7] (1)[(22-0)+(18-2)+(21-6)+(16-10)]/4 = 14.75ms

   (2)15ms

   [8] (1)[(0-0)+(2-2)+(7-6)+(11-10)] = 0.5ms

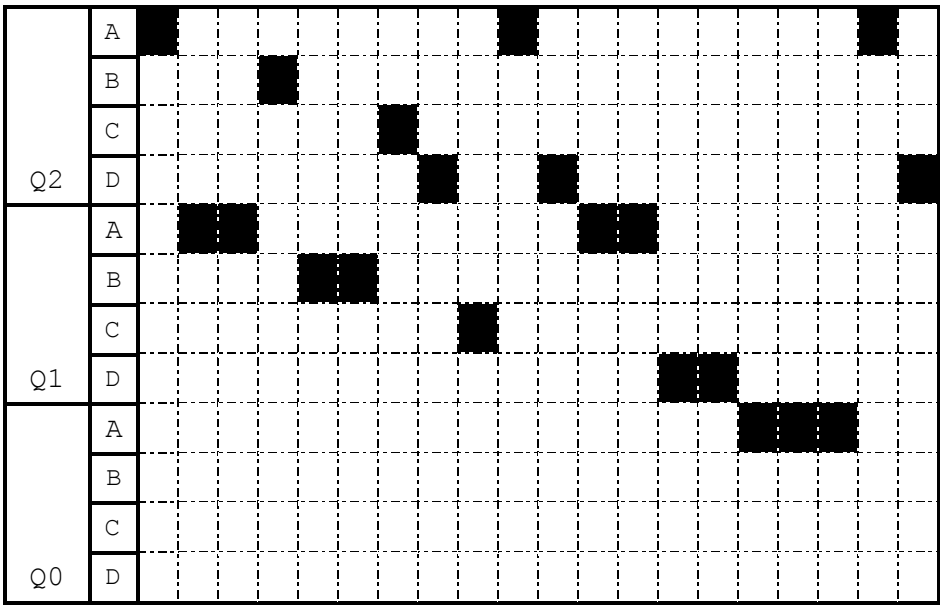
   (2)0.75ms

2. 1) [1] 3 [2] 3 [3] 6

   [4] 9 [5] 27

   2) The time between two priority boosting is too short. Jobs can hardly run in Q2.

(The Figure shows the details of execution flow)



## Problem 2: (29 points)

1. [1] 9 [2] 3 [3] 15 [4] 14

2. [1] 4 [2] 3 [3] 3 [4] 8
   [5] Y [6] N [7] 16 [8] 2de0
   [9] 0 [10] 1 [11] 1 [12] 0
   [13] N [14] N [15] 0f [16] 1fcc

3. CR3 is used to store the base physical address of L1 page table.
   If virtual address is used, you need to translate it into physical address first since you should use it to find L1 page table in the main memory before doing translation. Then it will cause recursive translation.
   The value of CR3 is part of each process context. So it will be changed after context switch.

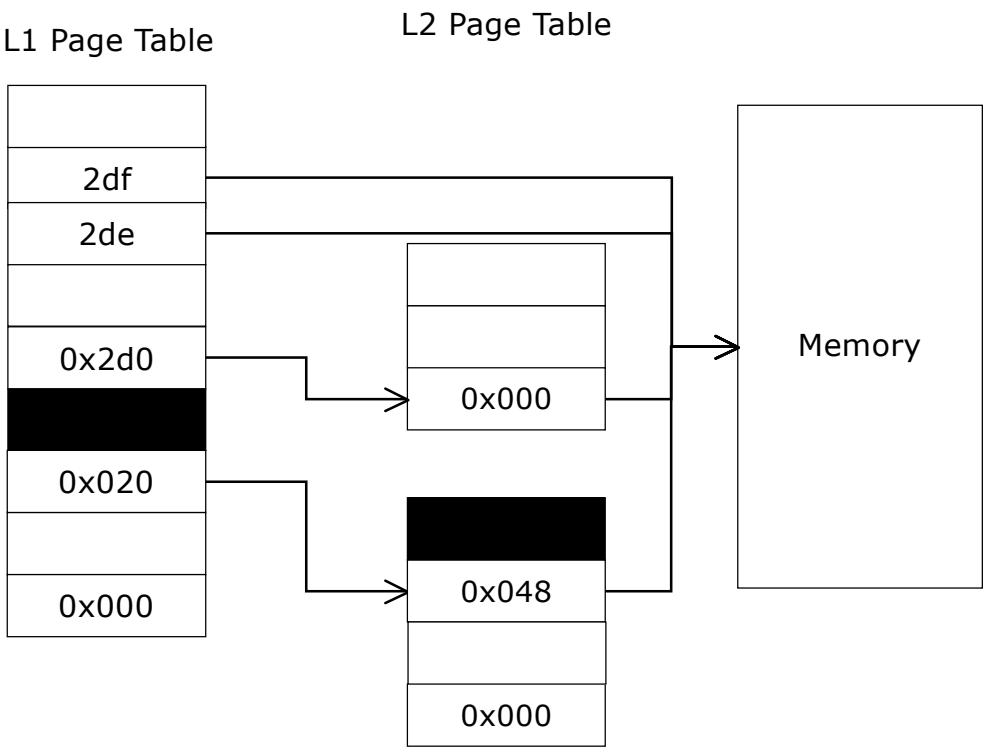
## Problem 3: (28 points)

1 [1] [31:22] [2] [11:0]
   [3] (0xffffff+1)/2^22 = 2^2 = 4
   [4] (0xffffff+1)/2^12 = 2^12

2 2048 page faults. 0 of them will handle COW. Because the code writes the whole sbuf array, raising 1024 page faults, and reads the whole pbuf array, raising another 1024 page faults. Since pbuf is only read not written, no COW will be handled.

3 [1]0xb7c00000 [2]0xef400000
   [3]0xb7c00800 [4]0xef400800

4




**Problem 4: (22 points)**

1. It works correctly. Assume one thread call lock(), if the lock is acquired by another thread, lock() will spin. Otherwise, the TAS still keeps that there is only one thread can acquire the lock.

2. When there are many threads acquiring the same lock (i.e., the contention is high), the performance is better.
TAS is only used to try to get the lock when it is likely to be free. Thus the expensive atomic memory operations happen less often.

1. [1] L4 [2] L1 [3] L6
   [4] L8 [5] U4

2. Before one thread calls park(), another thread calls unpark(), then the first thread sleeps forever.

3. Eliminate the race when several threads acquire the same lock (i.e., race on flag). Eliminate the race on queue.