Solution

Problem 1:

1 [1] 7ms [2] 3.5ms [3] 6ms [4] 0.5ms

[5] 8.75ms [6] 0.75ms

2 1) A:0ms, B:0ms, F:3ms

2) every 6ms

Problem 2:

1

[1]	[2]	[3]	[4]	[5]
1	1	2	2	1
2	3	3	3	2
3	4	4	4	4

2

[6]	[7]	[8]	[9]	[10]
1	1*	1	1	1
3*	3	2	2	2
2	4	4*	4*	4*

3. To implement the LRU policy, we need too many extra space to store information related to the access order (like an access timestamp or an extra access stack) and too many extra time to get the last recent page (like sorting of timestamps and updating the stack).

So clock algorithm is wildly used as an approximating of LRU policy. It takes history into consideration while maintaining an acceptable time overhead and space overhead.

Problem 3: (27 points)

1 [1] 8 [2] 4

[3] 10 [4] 16

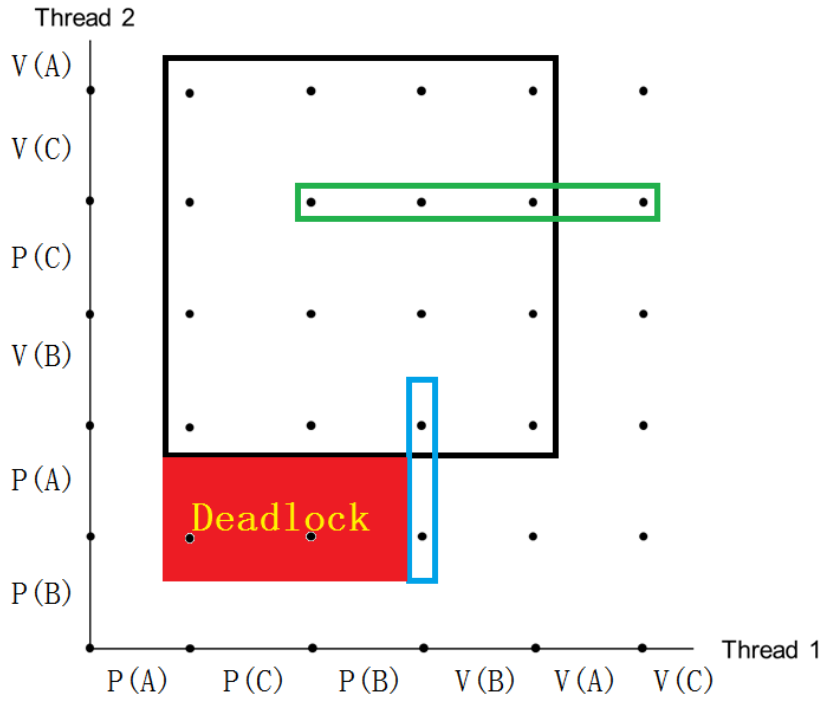
[5] 3 [6] 69888B (256+256*16+256*16*16)

2 [1] N [2] 3 [3] N [4] d91c [5] N

[6] N [7] 2/5 [8] Y [9] --- [10] ---

Problem 4: concurrency (10 points)

1 Yes.

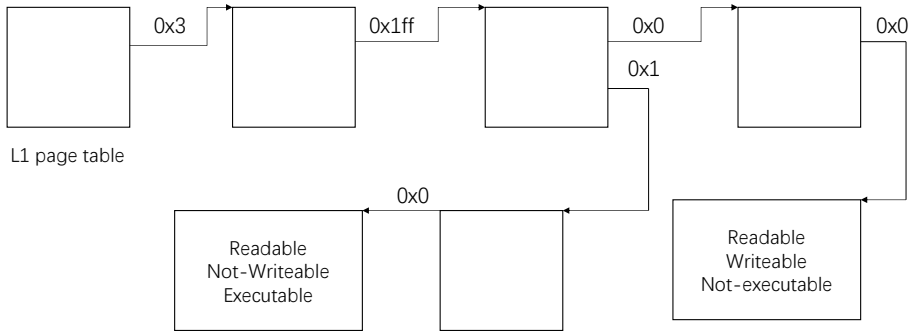


The rectangles are unreachable regions for this program. If the program goes into the DEADLOCK region, it can never go out, since the program can only move up or move right in this graph.

2 1&2

Problem 5: Memory Mapping

1



2 line 16/line 12&16

3 a[0] = 1

b[0] = 3

a[0] = 1

b[0] = 4

4 a.txt: 1

b.txt: 0

Problem 6: Lock

1 [1] &lock->head [2] node [3] NULL

2 a. Wakeup/waiting race:

```
Thread 1 call lock() ...          Thread 2 call unlock() ...
old->next = node;
interrupt: switch to Thread 2

                                      unpark();
                                  interrupt: switch to Thread 1
park();
```

In this situation no one will unpark thread 1 anymore.
Solution: Add setpark() before line 23.

b. This implementation of lock should trust every thread. If any modify its node_t structure maliciously, it may cause other thread cannot run/monopolize CPU/deadlock/...
Solution: No good solution.

3 Unlock does not return at line 34 means the CAS at line 33 failed, which implies there is another thread acquiring the lock. Line 36 and 37 will wait until that thread finishes its acquisition and sets this node's next field. Then we can unpark that thread at line 39.

4 The lock is fair. The nodes are linked through their arrival time, which is a FIFO queue.
The performance is good. It do not need to spin. There's no waste and starvation problem. But it need more space than other locks.