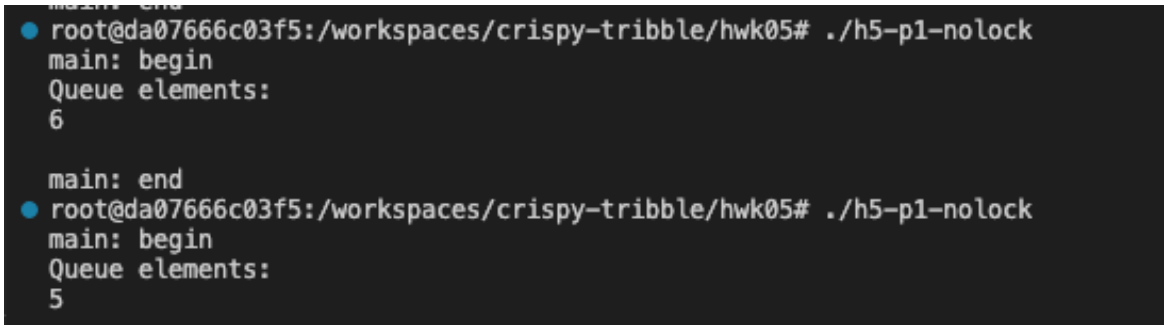
Problem 1

1. **Test your program without using any locks. Simulate a sequence of enqueue and dequeue operations. Comments on the results and highlights the data race problem that is expected to occur.**

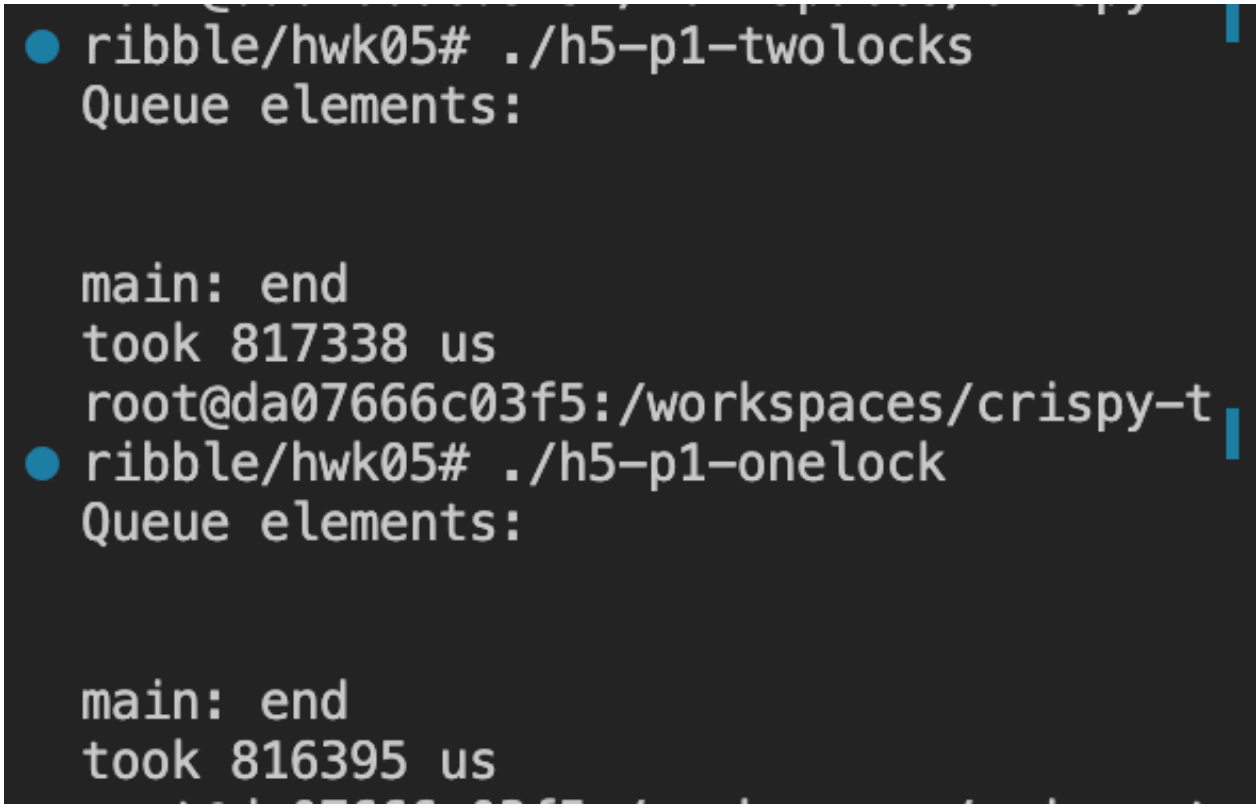
   since there is no lock involved, if we enque 0, 5, 6 with 3 threads; and deque with 2 threads, we should see one element remain in the queue. The data race problem here is that you can not ensure all the enque/deque calls are finished properly, which means, while the tail pointer was accessed by one of the enq threads, it is possible that one of the other thread will acquire the same tail from the memory (since all threads share the same address space), thus an enque will result in nothing get enqued;



refer to the above screen capture, we should get 6 as the only remaining element, however 5 could be printed out, since 6 is not enqued;

2. **Compare the performance of your program by using two locks (as in the slides) as opposed to using only one lock for both the enqueue and dequeue operations. Simulate a large number of overlapped (not sequential) enqueues and dequeues operations by all threads and use the system time (this is for simplicity as it is not the best way) to compare the performance of these two approaches. Comments on the results.**

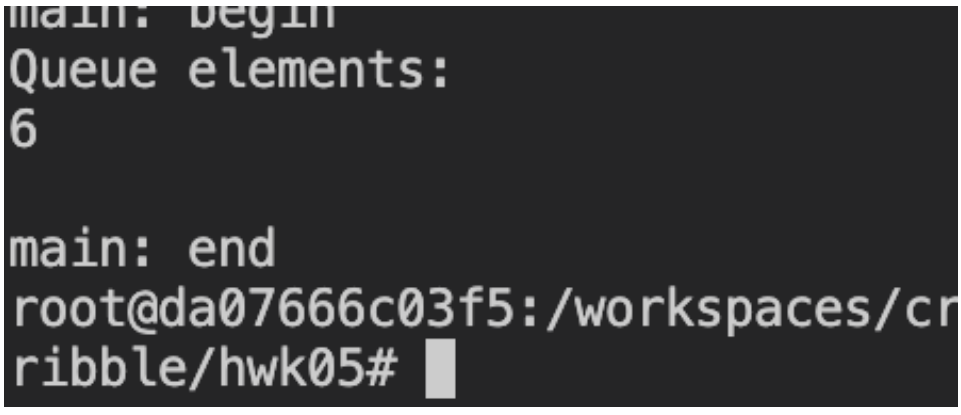
   According to the performance (time cost in us), the one lock approach does not show too much difference compare to the two locks approach



mutex lock implementation is architecture-specific. most of the mutexes will cost almost nothing if there is native hardware support, others will cost a lot. In the above casel, we can see little difference on the performance.

3. **Why do we need a dummy node in case two locks are deployed? Support your answer by testing the code without using a dummy node.**

   if there is no dummy node, think about this edge case, when there is only one node remains in the queue, so both the enque and deque thread can access that value, a deque would failed as an enque is not done.



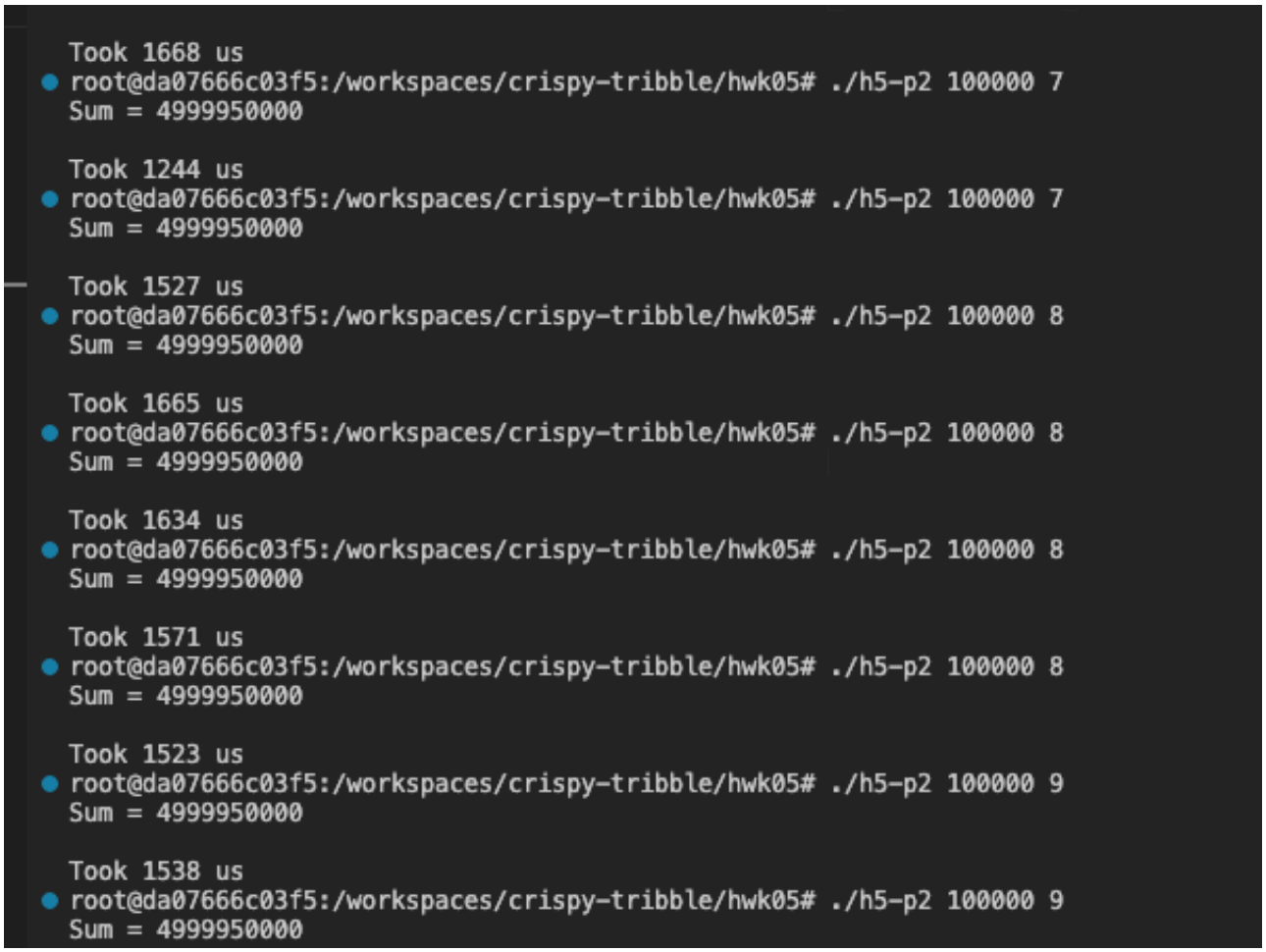
Problem 2

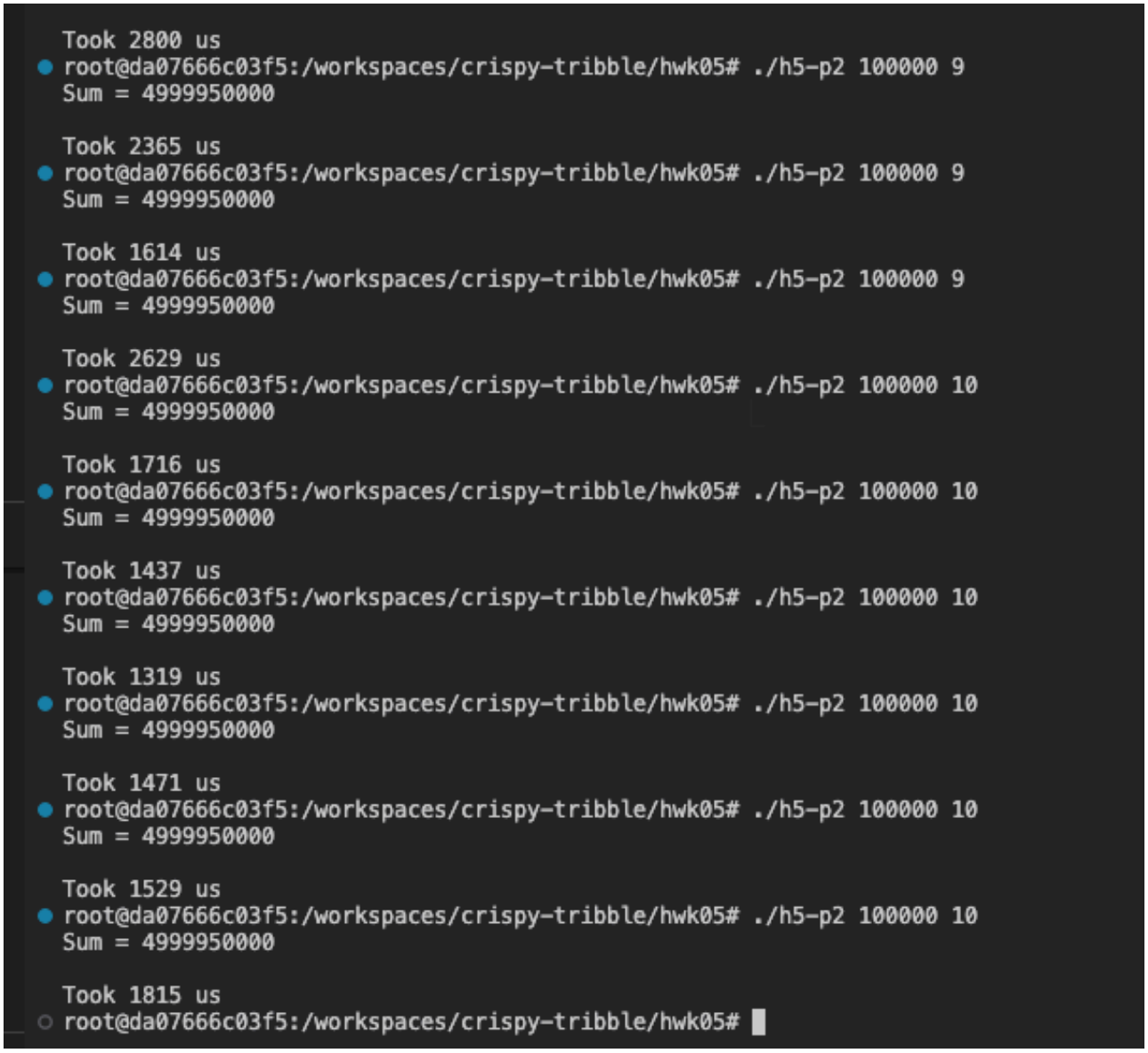
Test Result

num_threads	time_take (us)
1	1100
2	1337
3	1100
4	1200
5	1200
6	1124
7	1388
8	1432
9	1923
10	1817

---

screen shots provided




Took 2800 us
root@da07666c03f5:/workspaces/crispy-tribble/hwk05# ./h5-p2 100000 9
Sum = 4999950000
Took 2365 us
root@da07666c03f5:/workspaces/crispy-tribble/hwk05# ./h5-p2 100000 9
Sum = 4999950000
Took 1614 us
root@da07666c03f5:/workspaces/crispy-tribble/hwk05# ./h5-p2 100000 9
Sum = 4999950000
Took 2629 us
root@da07666c03f5:/workspaces/crispy-tribble/hwk05# ./h5-p2 100000 10
Sum = 4999950000
Took 1716 us
root@da07666c03f5:/workspaces/crispy-tribble/hwk05# ./h5-p2 100000 10
Sum = 4999950000
Took 1437 us
root@da07666c03f5:/workspaces/crispy-tribble/hwk05# ./h5-p2 100000 10
Sum = 4999950000
Took 1319 us
root@da07666c03f5:/workspaces/crispy-tribble/hwk05# ./h5-p2 100000 10
Sum = 4999950000
Took 1471 us
root@da07666c03f5:/workspaces/crispy-tribble/hwk05# ./h5-p2 100000 10
Sum = 4999950000
Took 1529 us
root@da07666c03f5:/workspaces/crispy-tribble/hwk05# ./h5-p2 100000 10
Sum = 4999950000
Took 1815 us
root@da07666c03f5:/workspaces/crispy-tribble/hwk05#

Problem 3

Problem 3 (20 Points)

The following table represents the current state of a system in which four resources *A*, *B*, *C* *D* are needed by the shown four processes. The system contains the following total instance each resource: 6 of *A*, 4 of *B*, 4 of *C*, 2 of *D*.

6 4 4 2 — 2 2 2 0

	Processes	Allocation				Max				Need				Available			
		A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
F 1	P_0	2	0	1	1	3	2	1	1	1	2	0	0	4	2	3	1
F 4	P_1	1	1	0	0	1	2	0	2	0	1	0	2	5	4	4	2
F 2	P_2	1	0	1	0	3	2	1	0	2	2	0	0	5	2	4	1
F 3	P_3	0	1	0	1	2	1	0	1	2	0	0	0	5	3	4	2

Answer the following questions using the Banker's algorithm:

a) Fill the missing entries in columns *Need* and *Available* in the above table.
b) Is the system in a safe state? Explain why.
c) Can a request from P_2 of (2,2,0,0) be granted? Explain why.

b) Yes, since $P_0 \rightarrow P_2 \rightarrow P_3 \rightarrow P_1$ satisfies safty creteria

100%

c)

6 4 4 2

0 0 2 2

	Processes	Allocation				Max				Need				Available			
		A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
2	P_0	2	0	1	1	3	2	1	1	1	2	0	0	5	2	4	3
3	P_1	1	1	0	0	1	2	0	2	0	1	0	2	6	3	4	3
1	P_2	13	02	1	0	3	2	1	0	1	0	0	0	3	2	3	2
4	P_3	0	1	0	1	2	1	0	1	2	0	0	0	6	4	4	4

It can be granted;

$P_2 \rightarrow P_0 \rightarrow P_1 \rightarrow P_3$ satisfies.