91.515 Operating System Assign2

Name: Chang Liu **Email: chang_liu@student.uml.edu**

Self-evaluation

I think I have done a great job in this second project, if I graded, I shall give 90% for the overall scores, my job mainly includes:
1) Write code implementation, including two major methods, aka process scope and system scope, which is only one required to finish our experiment, please refer to the separated files for details.
2) Run a large amount of testing data to verify my assumption of the running result, which also includes my extra test with fixed Q-depths of different dozen size.
3) I use many tools to monitor and get the precise data of 50% deadlock, even making the amount as precise as possible to avoid coincidence, including running the script 100 times and calculate the average.

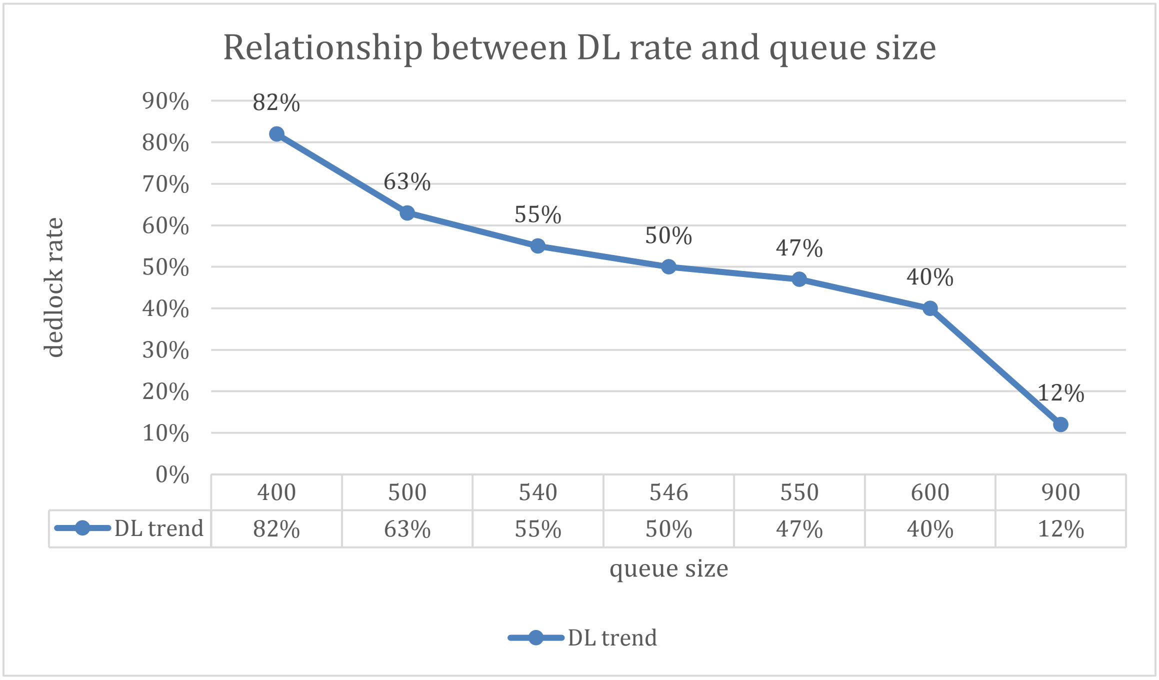
Q1: proper queue size

For this kind of configurations, I did some experiments to get the right value of queue size for the 50% deadlock rate, as the following result:

Table 1. Raw experiment data of Q-depth with DL rate

Q-depth	Deadlock rate
400	82%
.500	63%
540	55%
546	50%
550	47%
600	40%
900	12%

For the complete graph of the deadlock rate with the changes of Q-depths, please see the following graph:



	400	500	540	546	550	600	900
DL trend	82%	63%	55%	50%	47%	40%	12%

Figure 1. Graph of DL rate with different queue size

Analysis:
For a given queue size, we could calculated the deadlock rate, then changing it from 900 to another smaller value to get the trend of deadlock, here I know that it is decreasing as the queue size increase, so we should find smaller value than 900, find the proper area of around 50%.
Then, why should the deadlock rate decrease as the queue size increases? Here as there are fixed number of consumer and producers, no other factors are changed, so the larger the queue size it, the less unlikely it will fill the queue up. Since filling up the queue up will cause more likely the consumer and producers are waiting for resources, it causes more deadlock. In a word, increasing queue size will decrease the deadlock rate.

Q2: different dozen-size configurations

Under the condition of the fixed configurations in the Q1, here as the question required, I changed the dozen size from fixed 200 to 100, 150, 250 and 300 to test the deadlock rate.

Configurations:
30 producers, 50 consumers, Q-depth (queue size) is **546**, which is the configuration of 50% deadlock rate's queue size.

Table 2. Raw experiment data of dozen size with DL rate

dozen size	deadlock rate
100	16%
150	36%
200	50%(as Q1 shows)
250	64%
300	74%

For the complete graph of the deadlock rate with the changes of dozen size, please see the following graph:

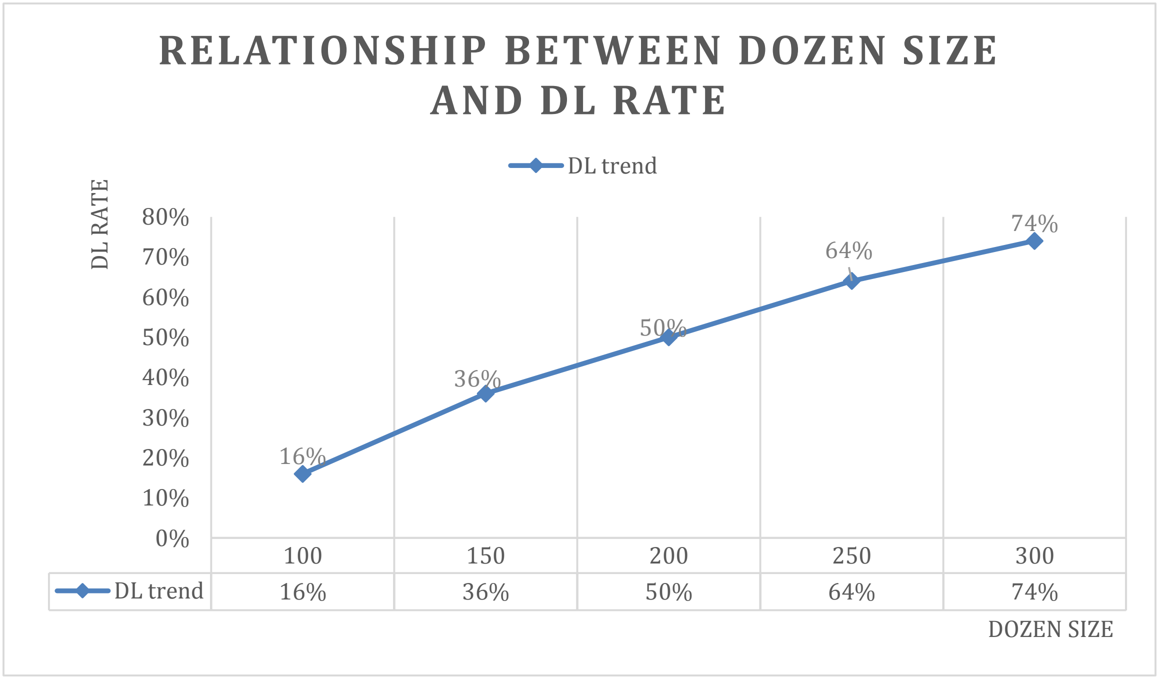


Figure 2. Graph of DL rate and dozen size

Analysis:
My conclusion about this test has some flaws at first, because I mistake it as the Q1 question, since the larger dozen size is, I think it should be less deadlock rate. However, after doing the experiment, I find that the conclusion is the opposite. I'm reconsidering my conclusion of the deadlock issue.
After I carefully checked the dozen size definition, I find that it is only related with the executing times of each consumer thread, which is defined as NUM_DOZ_TO_CONS, that means the larger this value is, the longer it will take to finish the executing of each thread. Then we can image, with larger dozen size, there will be more possibility that all of these consumer threads co-exist, so they are catching the resource more easily!

Then the conclusion is very obvious, the larger dozen size is, the higher deadlock rate is.
From this experiment, I figure out the different influence of queue size and dozen size on the deadlock rate.

Problems encountered in project

When doing this project, I met some weird problems which makes me very upset, however after debugging I could figure it out successfully, the debugging process is trivial but some procedure need to be recorded for reference and learning:
1) Wrong or uuencoded saving files name.
This problem is caused by the writing function, here I use snprintf to make the file as a string, and the last of the string is a thread-id, however, snprintf needs a n parameter for the length of the filename, I assume it was short, while in real situation it is long enough as 14960, which makes the overall name "cons14960" exceed the maximum of the total string, resulting in the weird file name.
2) Saving to file's data is wrong.
This problem takes me a whole afternoon to fix it. The situation is that when I use printf, the output seems right, but when I use IO operating function like fputs, write function, it always write 2 or other single value to the output file, not the serial number that I needed.
After a long time of debugging, I find it also the snprintf function, as the n is not the size of characters, but the size of digit it should contain, so if I use a very limited n value, then it will not save to file normally, leaving the remaining part to show the next line or table space, after using a simple test.c file, I figure out the reason and change the size to be a large one, this bug has been solved.
3) Others.
Other problems are quite simple and straight, since I could find out when there is any compiling or executing error. I use a MACRO to define my writing to consumer output file to control it better, I also doubled checked the output file content, which should contain thread-id, time and dozen number sequence.

Addition

About the experiment, I have some puzzles with the result, since in different machine the deadlock rate varies so much, I use on my machine(Ubuntu14.04, 4 CPU units, 8GB memory), it has a nearly 60% deadlock rate, but in the server machine, which is cs.uml.edu or mercury.cs.uml.edu, the former one is Ubuntu14.04 with 4 CPU units, the latter is RedHat6.5 Enterprise with 8 CPU units, the output is different in mercury as it has 8 cores, so it shows CPU0 ~ CPU7 are all existed, BUT the deadlock is nearly 100% in this situation.

Regarding to this situation, I asked the Prof. about it, the answer is simple, as the deadlock rate is related with different hardware and software, so we should stick to one platform and do our experiment, there is no fixed answer for it as it varies in many machines. In this case, I only test on my local machine to get the Q-depth and different deadlock rate. However, I have also run some test on the server, even though some situation is different, I could find that it nearly be 10% more than in local machine.

Extra work

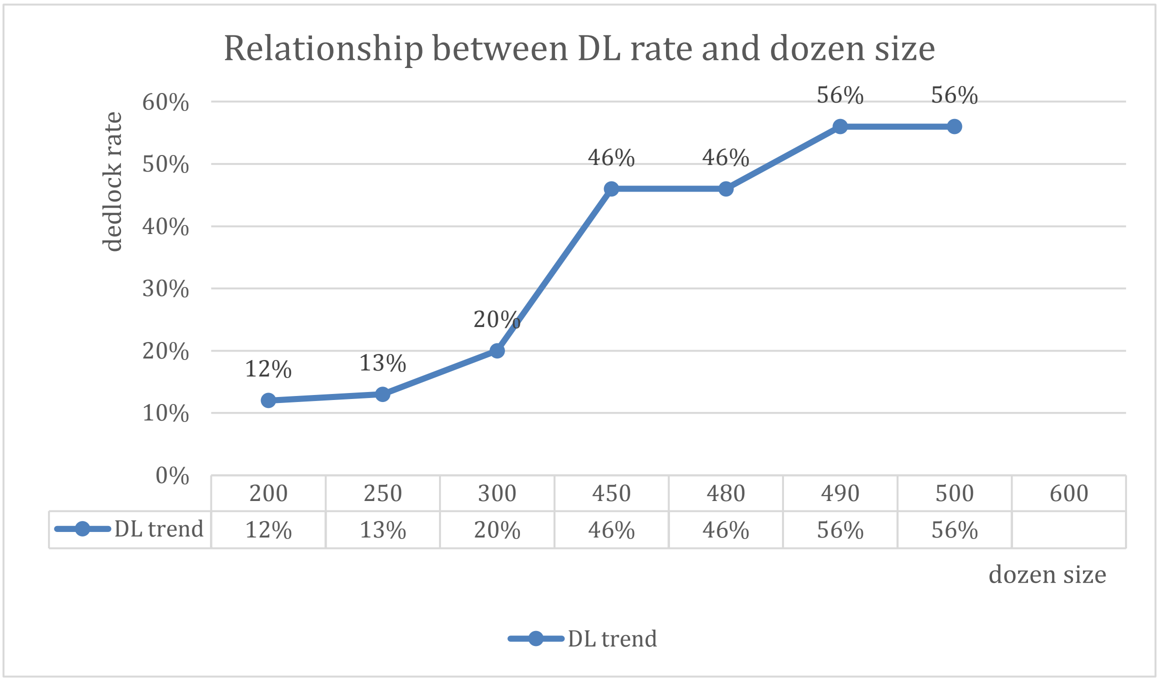
I order to get another relationship with the value of consumer dozens in a fixed value of Q-depth, I also did some experiments of the following configurations:

30 producers, 50 consumers, 900 Q-depth (Number of slots)

Changing the dozen size and calculate the deadlock rate, as the following table shows:

Table 3. Raw experiment data with dozen size and deadlock rate

Consumer doze size	deadlock rate
200	12%
250	13%
300	20%
450	46%
480	46%
490	56%
500	56%
600	61%



	200	250	300	450	480	490	500	600
DL trend	12%	13%	20%	46%	46%	56%	56%	

Figure 3. Graph of DL rate and dozen size (Q-depth is fixed in 900, different from 200 before)

In this situation, we could know that in that fixed configuration, as the consumer doze size increase, the deadlock rate increases, which is in our expectation. Because the larger the consumer dozen size is, the more probability that the consumer thread will be conflicted with each other. In my experiment, I find that in some very similar dozen size, the deadlock rate is very close, maybe it is because they are not in linear relation, so it may have some equal values. In this configuration, the 50% deadlock consumer size is around 480 ~ 490.

Comparing with the above experiment, as the consumer size is always 200, the result is a little different, because the relationship of Q-depth and dozen size with deadlock rate is quite different.