Concurrency Beyond Locks Assignment Report

Instruction:

Time your fixed-up version of pump.py (or pump-fixed.py above) with a buffer sizes of 1 and 100.

Now modify it to use semaphores, with release() and acquire(), or to use condition variables, with wait() and notify().

Predict whether the new version will be faster --- specifically whether it will require less real time, less user time, and less system time --- and explain why.

Time your new version.

Were your predictions correct?

If not, how can you explain what the results?

Explanation:  
Prediction: New version will be faster for real, user, will less time, and sys time will be about the same. The reason is because the original code only uses one variable to handle threads. Inside of the while loop, it repeatedly releases and acquires. When while loop breaks, it have to go to the last line “bufLock.release()” in order to unlock the lock and wake the other thread. By using condition variable, inside the loop will be redundant and with condition variable, notify() can be useful since it wakes up other thread.

In order to modified the given file pump-fixed.py I used condition variable. With condition variable, I used methods stated in instructions, wait() and notify().

Firstly, I introduced the variable that will be needed for condition variable. Imported Condition from threading and introduced two aditinal variable to the program. charAdded and charRemove. Both varaibale assigned Condition(bufLock) where bufLock is a lock.

Secondary, I modified inside of while-loop on both methods, putChar and getChar. I switched both lines of code, “bufLock.release()” and “bufLock.acquire()”, to charAdded.wait() – in putChar or charRemoved.wait() – in getChar() method.

Third, Since there are wait()s, now it is time to put notify() which handles waking up other threads. Conditional variable only works when the lock is being locked, thus, I put nofity() methods right before bufLock is released on both methods, getChar() and putChar().

Lastly, I added charRemoved and charAdded variable to global variable.

Result:  
 The result was partially expected, and the other part was wrong. Real, user time was reduced with modified version of code as expected. In addition, sys time marked less time by half comparing to original code. Sys is the amount of CPU time that kernel spend within the processes, in other words, the executing CPU time spent in system calls within the kernel. The original version uses one lock to handle every action in program which was reduced in modified version due to the introduction of two additional variable. Within loop, it only wait() meaning sleeping until the other wakes up the sleeping condition variable. Furthermore, in order to run other action (producer -> consumer, or consumer -> producer), it will be handle by charAdded and charRemoved which is more efficient as the processes running in kernel.

Code:

Pump-fixed.py (Not modified):

1. #!/usr/bin/env python3
2. # pump.py Nigel Ward, 2021. Based on John Osterhout's Producer/Consumer code.
3. import sys, threading, time
4. global count, putIndex, getIndex, cbuffer, bufLock
5. def pumpProducer():
6. print('starting Producer')
7. arpaciQuote = '"Yeats famously said “Education is not the filling of a pail but the lighting of a fire.” He was right but wrong at the same time. You do have to “fill the pail” a bit, and these notes are certainly here to help with that part of your education; after all, when you go to interview at Google, and they ask you a trick question about how to use semaphores, it might be good to actually know what a semaphore is, right? But Yeats’s larger point is obviously on the mark: the real point of education is to get you interested in something, to learn something more about the subject matter on your own and not just what you have to digest to get a good grade in some class. As one of our fathers (Remzi’s dad, Vedat Arpaci) used to say, “Learn beyond the classroom”.'
8. for charToSend in arpaciQuote:
9. putChar(charToSend)
10. putChar(0) # the solution!!
12. def putChar(character):
13. global count, putIndex, bufLock
14. bufLock.acquire()
15. while count >= bufsize:
16. #print("waiting to send", end="")
17. bufLock.release()
18. bufLock.acquire()
19. count += 1
20. cbuffer[putIndex] = character
21. putIndex += 1
22. if putIndex == bufsize:
23. putIndex = 0
24. bufLock.release()
25. def pumpConsumer():
26. print('starting Consumer')
27. while (1):
28. for i in range(100):
29. c = getChar()
30. if c == 0:
31. return
32. print(c,end="")
33. print("") # newline
34. def getChar():
35. global count, getIndex, bufLock
36. bufLock.acquire()
37. while (count == 0):
38. #print("waiting to receive", end="")
39. bufLock.release()
40. bufLock.acquire()
41. count -= 1
42. c = cbuffer[getIndex]
43. getIndex += 1
44. if (getIndex == bufsize):
45. getIndex = 0
46. bufLock.release()
47. return c
48. ### main ###
49. if len(sys.argv) != 2:
50. print(len(sys.argv))
51. print("usage: pump bufferSize")
52. exit(1)
53. bufsize = int(sys.argv[1])
54. cbuffer = ['x'] \* bufsize # circular buffer; x means uninitialized
55. count = putIndex = getIndex = 0
56. bufLock = threading.Lock()
57. consumer = threading.Thread(target=pumpConsumer)
58. consumer.start()
59. producer = threading.Thread(target=pumpProducer)
60. producer.start()
61. producer.join()

Pump-fixed-modifed.py (Modified):

1. #!/usr/bin/env python3
2. # pump.py Nigel Ward, 2021. Based on John Osterhout's Producer/Consumer code.
3. import sys, threading, time
4. from threading import Condition
5. global count, putIndex, getIndex, cbuffer, bufLock
6. def pumpProducer():
7. print('starting Producer')
8. arpaciQuote = '"Yeats famously said “Education is not the filling of a pail but the lighting of a fire.” He was right but wrong at the same time. You do have to “fill the pail” a bit, and these notes are certainly here to help with that part of your education; after all, when you go to interview at Google, and they ask you a trick question about how to use semaphores, it might be good to actually know what a semaphore is, right? But Yeats’s larger point is obviously on the mark: the real point of education is to get you interested in something, to learn something more about the subject matter on your own and not just what you have to digest to get a good grade in some class. As one of our fathers (Remzi’s dad, Vedat Arpaci) used to say, “Learn beyond the classroom”.'
9. for charToSend in arpaciQuote:
10. putChar(charToSend)
11. putChar(0) # the solution!!
13. def putChar(character):
14. global count, putIndex, bufLock, charRemoved, charAdded
15. bufLock.acquire()
16. while count >= bufsize:
17. charRemoved.wait()
18. count += 1
19. cbuffer[putIndex] = character
20. putIndex += 1
21. if putIndex == bufsize:
22. putIndex = 0
23. charAdded.notify()
24. bufLock.release()
25. def pumpConsumer():
26. print('starting Consumer')
27. while (1):
28. for i in range(100):
29. c = getChar()
30. if c == 0:
31. return
32. print(c,end="")
33. print("") # newline
34. def getChar():
35. global count, getIndex, bufLock, charRemoved, charAdded
36. bufLock.acquire()
37. while (count == 0):
38. charAdded.wait()
39. count -= 1
40. c = cbuffer[getIndex]
41. getIndex += 1
42. if (getIndex == bufsize):
43. getIndex = 0
44. charRemoved.notify()
45. bufLock.release()
46. return c
47. ### main ###
48. if len(sys.argv) != 2:
49. print(len(sys.argv))
50. print("usage: pump bufferSize")
51. exit(1)
52. bufsize = int(sys.argv[1])
53. cbuffer = ['x'] \* bufsize # circular buffer; x means uninitialized
54. count = putIndex = getIndex = 0
55. bufLock = threading.Lock()
56. charAdded = Condition(bufLock)
57. charRemoved = Condition(bufLock)
58. consumer = threading.Thread(target=pumpConsumer)
59. consumer.start()
60. producer = threading.Thread(target=pumpProducer)
61. producer.start()
62. producer.join()

Output:

Command: time python3 pump-fixed-modified.py 10

Text

Description automatically generated

Command: time python3 pump-fixed.py 10

Text

Description automatically generated