CSC 343 Operating Systems, Fall 2020

Dr. Dale E. Parson, Assignment 2, Implementing unbounded-buffer & bounded-buffer Producer->Consumer dataflow.

This assignment is due via **make turnitin** from the BoundedBuffer2020 directory by **11:59 PM on Thursday October 15**. There is a 10% penalty for each day it is late, and I will not accept solutions after I go over my solution in class.

Added 10/1/2020: You must log into mcgonagall from acad using this command to run **make test**:
ssh mcgonagall
All testing must run on mcgonagall to reduce processor load on acad. ALSO, From an email on 10/1:

If you have previously copied (prior to October 1 at noon)
~parson/OpSys/BoundedBuffer2020.problem.zip
into your account, but have not started working on it, please re-copy it from my account.

If you have already started working on it, just cd into your BoundedBuffer2020 project directory and do the following:

```
cp   ~parson/OpSys/BoundedBuffer2020/makefile   makefile
```

See my email to the class from 10/1 at 11:53 AM for the explanation of the makefile changes.

Answers in README.txt are worth 15% of this assignment, so remember that working code is not the end of the requirements.

Perform the following steps to get my handout. You will code and test on acad.

```
cd  $HOME                 # or start out in your login directory
mkdir  OpSys  # All of this semester's work goes under here, skip if you did it before.
cd  ./OpSys
cp  ~parson/OpSys/BoundedBuffer2020.problem.zip  BoundedBuffer2020.problem.zip
unzip BoundedBuffer2020.problem.zip
cd ./BoundedBuffer2020
make clean test
```

Do NOT change the name of project directory BoundedBuffer2020 after you unzip it. My automated testing depends on this directory having this name. Testing fails within the handout directory as follows:

```
$ make clean test
The final action cpu in machine thread, from state initSharedData to state loopConsumer has no matching event from loopConsumer.

Traceback (most recent call last):
  … (Ignore the details.)
ValueError: The final action cpu in machine thread, from state initSharedData to state loopConsumer has no matching event from loopConsumer.
```

All of the detailed instructions for your code additions appear in **STUDENT** comments in source file **UnboundedBuffer2020.stm**, which is one of the three files you will change. The final file is

README.txt. We will go over your project requirements in class on 9/30.

If you get an error message at run-time that gives an index into __codeTable__ like this:

```
Traceback (most recent call last):
  File "BoundedBuffer2020.py", line 758, in <module>
    main()
  File "BoundedBuffer2020.py", line 694, in main
    scheduler.__run__()
  File "/home/kutztown.edu/parson/OpSys/state2codeV17/CSC343Sim.py", line 145, in __run__
    waitingObject.__generator__.__next__()  # run() the model
  File "BoundedBuffer2020.py", line 457, in run
    if eval(__codeTable__[39],globals,locals):
  File "nofile", line 1, in <module>
NameError: name 'ln' is not defined
```

Just run this decode.py command with that index to see the original source code.

```
$ python decode.py BoundedBuffer2020.py 39
```

```
__codeTable__[39] = compile('ln(pcb.inplay[othertid]) == 0','nofile','eval'),
```

<u>**STUDENT STEPS**</u> (always look for STUDENT comments in the code):

1. Complete the code in handout file UnboundedBuffer.stm per detailed STUDENT instructions, worth 60% of the assignment.
2. 15% of project: After **make test_ubb** works correctly on UnboundedBuffer.stm, type:
   cp UnboundedBuffer.stm   BoundedBuffer.stm
   then edit BoundedBuffer.stm and make the following changes:

   2a. Change the value of **Qmult = 1.0** in BoundedBuffer.stm to your value from this table. Each student has their own unique Qmult value, which makes the Consumer run that much slower, on average, than the Producer.

01: mbart974	Qmult = 1.0
02: relli137	Qmult = 1.1
03: efarr327	Qmult = 1.2
04: aglac280	Qmult = 1.3
05: ngolo273	Qmult = 1.4
06: jkoss254	Qmult = 1.5
07: ckunz125	Qmult = 1.6
08: clong579	Qmult = 1.7
09: pmanl065	Qmult = 1.8
10: mmasa309	Qmult = 1.9
11: nmore903	Qmult = 2.0
12: cocon040	Qmult = 2.1
13: oolae507	Qmult = 2.2
14: mpate337	Qmult = 2.3
15: aprit540	Qmult = 2.4
16: arain245	Qmult = 2.5
17: areic400	Qmult = 2.6
18: nrew088	Qmult = 2.7

19: rrile340	Qmult = 2.8
20: aschn868	Qmult = 2.9
21: ashei894	Qmult = 3.0
22: jurib113	Qmult = 3.1

We will discuss the relationship of these Qmult values to the Gaussian sample() function parameters on September 30. http://faculty.kutztown.edu/parson/fall2013/distributions.pdf

You MUST use the Qmult value assigned to you in order not to lose points. Next, assign some positive integer (> 0) into variable Qbound in BoundedBuffer.stm. Then, run **make test_bb** to test BoundedBuffer.stm. If the test fails, Qbound – the size of the bounded buffer – is too small. If the test succeeds, Qbound may be too big. You must find the **smallest** value of Qbound that succeeds in **make test_bb** with your individual Qmult. Both Qmult and Qbound will be different for every student.

3. 10% of project: After **make test_bb** works correctly on BoundedBuffer.stm, type:
cp BoundedBuffer.stm PolledRendevous.stm
then edit PolledRendevous.stm and make the following changes:

Set these two variables to these values, creating a 1-place bounded Queue and a Consumer thread that runs, on average, 10 times slower than the Producer thread:

Qbound = 1, Qmult = 10.0,

Run **make test_pr** to verify that the single-entry bounded buffer does not pass all 100 messages from the Producer to the Consumer.

Next, use the return value from the Producer's call to PCB.FIFI.enq(…) to determine whether the Producer has succeeded in enqueuing the message into the queue. You may do this:

YOURVARIABLE = pcb.FIFO.enq(MessagesSent);

YOURVARIABLE will have the value None if the buffer had no remaining space to enqueue MessagesSent, and will have a non-None value if enq() succeeded. You can use YOURVARIABLE as a Boolean in a single-statement if construct like this:

if YOURVARIABLE: Run 1 action statement.

You could also use YOURVARIABLE as a Boolean value as part of a single-statement while loop or a transition's guard condition. However, **DO NOT** literally compare YOURVARIABLE to Python True or False. You can use it as a Boolean, but its value is never True or False. Note these examples:

```
>>> YOURVARIABLE = None;
>>> if YOURVARIABLE: print("goody");        # This is valid.
...

>>> YOURVARIABLE = ('some value returned by enq other than None');
>>> if YOURVARIABLE: print("goody");        # This is valid
...
goody
```

```
>>> YOURVARIABLE = None;
>>> if YOURVARIABLE == True: print("goody");       # This is invalid; it's never == True or False.
...
>>> if YOURVARIABLE == False: print("goody");      # This is invalid; it's never == True or False.
...
>>> if not YOURVARIABLE: print("goody");           # This is valid
...
goody
```

What you must do is to have the Producer continue to try to enq() the same message, over and over, until pcb.FIFO.enq(MessagesSent) accepts the message. In PolledRendevous.stm, the Producer polls pcb.FIFO via enq()'s return value until pcb.FIFO accepts the message. Since there is only 1 position in bounded buffer pcb.FIFO, this approach almost acts as a rendezvous, since the Producer and Consumer threads must synchronize closely. A true rendezvous requires them both to interact with the FIFO concurrently; this STM's 1-position Queue actually allows the Producer to proceed without rendezvous with the Consumer, when the 1 slot is already available without looping. Most of the time that won't happen, since the Consumer runs more slowly, on average, than the Producer.

My solution does not change the state machine graph. You may add a transition if you want, but that is probably an over-complicated design.

4. 15% of project: Complete file README.txt as you did for assignment 1. Run **make clean test** after doing so to ensure that everything is good to go.

A successful test run appears as follows.

$ **make clean test**

```
COMPILING Unbounded Buffer
/bin/bash -c "PYTHONPATH=/home/kutztown.edu/parson/OpSys:.:.. /usr/local/bin/python3.7
/home/kutztown.edu/parson/OpSys/state2codeV17/State2CodeParser.py UnboundedBuffer.stm
UnboundedBuffer.dot UnboundedBuffer.py CSC343Compile CSC343Compile"
INFO: Blocking function spawn is in mid-transition from thread.initSharedData -> loopProducer, so its
completion event will not trigger a state change.
COMPILING COMPLETED
SIMULATING (TESTING) UnboundedBuffer
/bin/bash -c "PYTHONPATH=/home/kutztown.edu/parson/OpSys:.:.. STMLOGDIR=~parson/tmp time
/usr/local/bin/python3.7 UnboundedBuffer.py 2 4 500000 12345 3"
MSG cmd line: ['UnboundedBuffer.py', '2', '4', '500000', '12345', '3'], usage USAGE: python
THISFILE.py NUMCONTEXTS NUMFASTIO SIMTIME SEED|None LOGLEVEL

Scheduler exiting at time 5331 within time limit 500000, simulation has finished.
0.06user 0.01system 0:00.10elapsed 78%CPU (0avgtext+0avgdata 10200maxresident)k
0inputs+56outputs (0major+5065minor)pagefaults 0swaps
/bin/bash -c 'chmod 666 ~parson/tmp/parson_STM*'
grep "MESSAGES" UnboundedBuffer.log | sed -e 's/^.*MSG/MSG/g' | sort > UnboundedBuffer.out
/bin/bash -c "PYTHONPATH=/home/kutztown.edu/parson/OpSys:.:.. /usr/local/bin/python3.7
crunchlog.py UnboundedBuffer.log"

DIFFing UnboundedBuffer_crunch.py UnboundedBuffer_crunch.ref
OK: MEAN_loopProducer at 20.0% tolerance.
```

```
OK: MEAN_loopConsumer at 20.0% tolerance.

diff UnboundedBuffer.out UnboundedBuffer.ref
```
COMPLETED (OK) SIMULATING (TESTING) UnboundedBuffer
```
COMPILING BoundedBuffer
/bin/bash -c "PYTHONPATH=/home/kutztown.edu/parson/OpSys:.:.. /usr/local/bin/python3.7
/home/kutztown.edu/parson/OpSys/state2codeV17/State2CodeParser.py BoundedBuffer.stm
BoundedBuffer.dot BoundedBuffer.py CSC343Compile CSC343Compile"
INFO: Blocking function spawn is in mid-transition from thread.initSharedData -> loopProducer, so its
completion event will not trigger a state change.
COMPILING COMPLETED
SIMULATING (TESTING) BoundedBuffer
/bin/bash -c "PYTHONPATH=/home/kutztown.edu/parson/OpSys:.:.. STMLOGDIR=~parson/tmp time
/usr/local/bin/python3.7 BoundedBuffer.py 2 4 500000 12345 3"
MSG cmd line: ['BoundedBuffer.py', '2', '4', '500000', '12345', '3'], usage USAGE: python THISFILE.py
NUMCONTEXTS NUMFASTIO SIMTIME SEED|None LOGLEVEL

Scheduler exiting at time 50127 within time limit 500000, simulation has finished.
0.09user 0.01system 0:00.15elapsed 70%CPU (0avgtext+0avgdata 10204maxresident)k
0inputs+56outputs (0major+5063minor)pagefaults 0swaps
/bin/bash -c 'chmod 666 ~parson/tmp/parson_STM*'
grep "MESSAGES" BoundedBuffer.log | sed -e 's/^.*MSG/MSG/g' | sort > BoundedBuffer.out
# /bin/bash -c "PYTHONPATH=/home/kutztown.edu/parson/OpSys:.:.. /usr/local/bin/python3.7
crunchlog.py BoundedBuffer.log"
diff BoundedBuffer.out BoundedBuffer.ref
```
COMPLETED (OK) SIMULATING (TESTING) BoundedBuffer
```
COMPILING PolledRendevous
/bin/bash -c "PYTHONPATH=/home/kutztown.edu/parson/OpSys:.:.. /usr/local/bin/python3.7
/home/kutztown.edu/parson/OpSys/state2codeV17/State2CodeParser.py PolledRendevous.stm
PolledRendevous.dot PolledRendevous.py CSC343Compile CSC343Compile"
INFO: Blocking function spawn is in mid-transition from thread.initSharedData -> loopProducer, so its
completion event will not trigger a state change.
COMPILING COMPLETED
SIMULATING (TESTING) PolledRendevous
/bin/bash -c "PYTHONPATH=/home/kutztown.edu/parson/OpSys:.:.. STMLOGDIR=~parson/tmp time
/usr/local/bin/python3.7 PolledRendevous.py 2 4 500000 12345 3"
MSG cmd line: ['PolledRendevous.py', '2', '4', '500000', '12345', '3'], usage USAGE: python THISFILE.py
NUMCONTEXTS NUMFASTIO SIMTIME SEED|None LOGLEVEL

Scheduler exiting at time 50127 within time limit 500000, simulation has finished.
0.15user 0.02system 0:00.21elapsed 80%CPU (0avgtext+0avgdata 10192maxresident)k
0inputs+240outputs (0major+5057minor)pagefaults 0swaps
/bin/bash -c 'chmod 666 ~parson/tmp/parson_STM*'
grep "MESSAGES" PolledRendevous.log | sed -e 's/^.*MSG/MSG/g' | sort > PolledRendevous.out
# /bin/bash -c "PYTHONPATH=/home/kutztown.edu/parson/OpSys:.:.. /usr/local/bin/python3.7
crunchlog.py PolledRendevous.log"
diff PolledRendevous.out PolledRendevous.ref
```
COMPLETED (OK) SIMULATING (TESTING) PolledRendevous

You can see the successful extracted messages from your program by doing this.

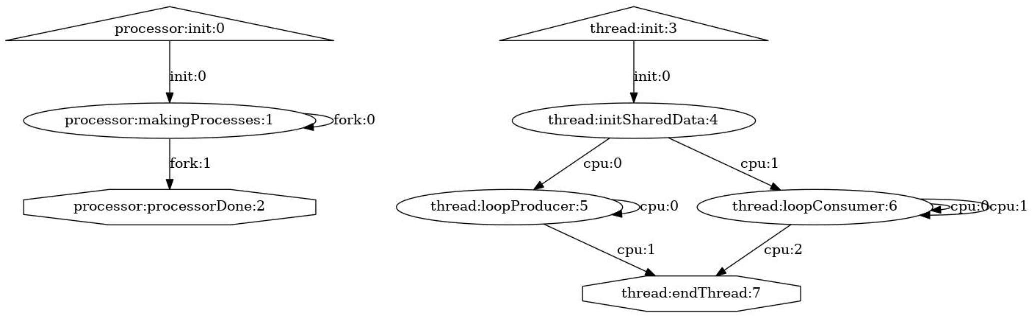
```
$ grep MESSAGES *.log
BoundedBuffer.log:000000005175,MSG,thread 0 process 0,PRODUCER SENT 100 MESSAGE.
BoundedBuffer.log:000000050127,MSG,thread 1 process 0,CONSUMER RECEIVED 100 MESSAGES.
BoundedBuffer.log:000000050127,MSG,thread 1 process 0,CONSUMER MESSAGES: [1, 2, 3, 4, 5, 6,
7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35,
36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63,
64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91,
92, 93, 94, 95, 96, 97, 98, 99, 100]
PolledRendevous.log:000000049343,MSG,thread 0 process 0,PRODUCER SENT 100 MESSAGES.
PolledRendevous.log:000000050127,MSG,thread 1 process 0,CONSUMER RECEIVED 100
MESSAGES.
PolledRendevous.log:000000050127,MSG,thread 1 process 0,CONSUMER MESSAGES: [0, 1, 2, 3, 4,
5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62,
63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90,
91, 92, 93, 94, 95, 96, 97, 98, 99]
UnboundedBuffer.log:000000005175,MSG,thread 0 process 0,PRODUCER SENT 100 MESSAGES.
UnboundedBuffer.log:000000005331,MSG,thread 1 process 0,CONSUMER RECEIVED 100
MESSAGES.
UnboundedBuffer.log:000000005331,MSG,thread 1 process 0,CONSUMER MESSAGES: [1, 2, 3, 4, 5,
6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62,
63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90,
91, 92, 93, 94, 95, 96, 97, 98, 99, 100]
```

Any time a COMPILE succeeds, you can look at the graph for your state machine by running **make graphs** and then inspecting https://acad.kutztown.edu/~STUDENT/UnboundedBuffer.jpg, where STUDENT is your login ID. If you can't get at it with a browser this way, use WinSCP or FileZilla to copy the JPEG file from your project directory to your local machine. Below is the final, correct graph.

Once **make clean test** passes, **ANSWER THE QUESTIONS IN FILE README.txt** included in this project directory. Follow all instructions in README.txt.

Finally, turn it in by entering **make turnitin** and following the prompt. We do not use the turnin script in this course; instead **make turnitin** turns in the project; it prompts you for a carriage return (Enter) to complete its work.

I will distribute grades via email before the next class after the due date.



https://acad.kutztown.edu/~parson/UnboundedBuffer.jpg