COMP 530 Introduction to Operating Systems

Fall 2017  
Kevin Jeffay

Worksheet 6, September 20

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Your Name: |  | You worked with: |  | +1/blank/-1: |  |
|  | Aaron Zhang |  | Daniel Estrada |  | +1 |  |
|  |  |  | Jonathan Ng |  | +1 |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

1. Recall the producer/consumer implementation that uses a shared counter from Lecture 5 (see page 8 of the lecture notes). Assume for this question that (by magic) the increment and decrement operations on count are atomic.

|  |  |
| --- | --- |
| process *Producer*  var *c* : char  begin  loop  <*produce a character “c”*>  while *count* = *n* do  NOOP  end while  *buffer*[*nextIn*] := *c*  *nextIn* := *nextIn*+*1* mod *n*  *count* := *count* + 1  end loop  end *Producer* | process *Consumer*  var *c* : char  begin  loop  while *count* = 0do  NOOP  end while  *c* := *buffer*[*nextOut*]  *nextOut* := *nextOut*+*1* mod *n*  *count* := *count* - 1  <*consume a character “c”*>  end loop  end *Consumer* |

*a*) Is this implementation correct? Explain.

*b*) If the increment and decrement operations on *count* were moved to be immediately after the while-loop, would this effect the correctness of the solution? Explain.

Use this space to continue answering question 1.

2. Recall the producer/consumer implementation from Lecture 5 that does not use a shared counter:

|  |  |
| --- | --- |
| process *Producer*  var *c* : char  begin  loop  <*produce a character “c”*>  while *nextIn*+1 mod *n = nextOut* do  NOOP  end while  *buffer*[*nextIn*] := *c*  *nextIn* := *nextIn*+*1* mod *n*  end loop  end *Producer* | process *Consumer*  var *c* : char  begin  loop  while *nextIn* = *nextOut* do  NOOP  end while  *c* := *buffer*[*nextOut*]  *nextOut* := *nextOut*+*1* mod *n*  <*consume a character “c”*>  end loop  end *Consumer* |

*a*) Is there a critical section on the operations on the shared, global variables *nextIn* and *nextOut*? Explain why or why not.

*b*) Is there a critical section on the operations on the shared, global variable *buffer*? Explain why or why not.

*c*) Was there a critical section on these same variables in question 1?

3. Considering again the code from question 1 (except this time *not* assuming that the operations on count are atomic). What would be the effect of executing the producer and consumer processes using a non-preemptive scheduling policy?

*a*) Would the producer/consumer implementation be correct in such a case?

*b*) Is there any downside to using non-preemptive scheduling?

4. Recall Peterson’s mutual exclusion algorithm:

|  |  |
| --- | --- |
| process P1  begin  loop  inCS[1] := TRUE  turn := 1  while turn = 1 AND inCS[2] do  NOOP  end while  <critical section>  inCS[1] := FALSE  :  end loop end P1 | process P2  begin  loop  inCS[2] := TRUE  turn := 2  while turn = 2 AND inCS[1] do  NOOP  end while  <critical section>  inCS[2] := FALSE  :  end loop end P2 |

WITHOUT LOOKING UP THE SOLUTION ON THE INTERNET (BECAUSE DOING SO WILL ENSURE YOU LEARN NOTHING!!), extend Peterson’s algorithm so that it works for a set of *n* processes.