CISC 324 -- Winter 2017

Assignment 2 due Wednesday Feb 1, in lecture

Lab 3 is due one week later, Wednesday Feb 8, in lecture

If you can’t make it to lecture, then hand the assignment/lab in early by bringing it to the School of Computing Office in Goodwin 557. After hours, you can slide the assignment under the door of Goodwin 557; label it as “for Professor Blostein”.

Readings for Assignment 2

In the course reader:
- Pages 19-28  Programming language constructs for creating processes; unpredictable outcomes; critical sections; semaphores; precedence graphs; readers and writers problem
- Pages 32-37  A starvation-free solution to the readers/writers problem (using semaphores)

In the textbook:
- Sections 3.1 to 3.4  Processes: scheduling, operations, communication
- Section 5.1 (6.1 in 8th edition)  Introductory information for process synchronization
- Section 5.6 (6.5 in 8th edition)  Semaphores. Skip the busy-waiting definition of semaphore at the start of Section 5.6. Instead, focus on the semaphore definition in section 5.6.2 that uses a list of processes (a list of PCBs). These two definitions are consistent with each other: they define the same behaviour for the semaphore operations wait and signal. However, the definition in section 5.6.2 has the advantage that busy-waiting is avoided: a waiting process becomes blocked (the operating system puts the PCB on a semaphore queue) rather than wasting CPU time constantly checking whether the wait is over.

Assignment 2 Questions

(1) A process is said to be “a program in execution”. Briefly describe how the operating system maintains the connection between a process and its corresponding program. Illustrate how it is possible for several processes to be executing the same program (for example: several users simultaneously running the same text editor).

(2) When a process isn't executing, the PCB (process control block) stores the information needed for later restart of the process. Which of the following information should be stored in the PCB, and which does not need to be stored there? Briefly justify your answers.
   a. The values in the general-purpose CPU registers.
   b. A copy of the program (the machine code) that the process is executing.
   c. A copy of the disk areas (the files) that the process has access to.

(3) J and K are shared variables stored in main memory, and initialized to 1. What are the possible outcomes (final pairs of values for J and K) if the following code is executed concurrently?

```
process 1
J = K+3;
K = J+J;
```

To answer this question, you have to think in terms of machine instructions because a context switch can occur after any machine instruction. The compiler translates the above code to machine instructions such as the following.

```
process 1
A1 MOV R1, K
A2 ADD R1, 3
A3 MOV J, R1

process 2
B1 MOV R3, J
B2 ADD R3, J
B3 MOV K, R3
```

Hint: To find all possible outcomes, look at all possible orderings of the instructions that access shared memory. In this problem you have to consider all possible orderings of A1, A3, B1, B2, B3. Since A2 does not access shared memory, it does not interact with other processes and you can lump it together with A1 or A3 in the instruction ordering. Remember that A1 must execute before A3; also B1, B2, B3 must execute in that order. In summary: you need to find all ways to interleave the list A1 A3 with the list B1 B2 B3. You can treat A2 as bundled with instruction A1, or bundled with instruction A3.
(4) Draw a precedence graph that illustrates the parallelism present in the following code. Start by drawing nine graph nodes, labeled S0 to S8. Then draw an edge to connect node Si to node Sj if computation Si must complete before computation Sj can begin. (Reminder: “cobegin/coend” stands for “concurrent begin / concurrent end”. Sometimes this is called “parbegin/parend”, short for “parallel begin / parallel end”. See pages 24-27 of the course reader for examples.)

begin
  S0
  cobegin
    S1
    S2
    begin
      S3
      S4
    end
    cobegin
      S5
      S6
    coend
    S7
  end
  coend
  S8
end

(5) (a) Write cobegin/coend code for the following precedence graph. Make your program express as much parallelism as possible within the limitations of cobegin/coend, while being sure to enforce all the constraints that are in the precedence graph. (The best solutions I know of introduce two or three extra precedence edges. Try to find one of these solutions.)

(b) Express the precedence graph from part (a) using semaphores. Show the initial value of each semaphore. Hint: As is discussed in lecture, one way to do this is to define semaphores with names such as S1_done, S2_done. If Si has to complete before Sj starts you could say:

\[
\begin{align*}
\text{process } i & \quad \text{<code for } S_i> \\
& \quad \text{Acquire(S}_i\text{done)} \\
& \quad \text{Release(S}_i\text{done)} \\
& \quad <\text{code for } S_j> \\
\end{align*}
\]

(6) Processes P and Q are executing concurrently, using a shared semaphore named \texttt{sem} that is initialized to 3.

\[
\begin{align*}
\text{var } \texttt{sem}: \texttt{semaphore} & := 3; \\
\text{Process P} & \quad \text{Process Q} \\
\text{loop} & \quad \text{loop} \\
& \quad \text{acquire(}\texttt{sem}); \\
& \quad \text{printf(}“\text{Q speaking”); \\
& \quad \text{release(}\texttt{sem}) \\
& \quad \text{endloop} \\
& \quad \text{endloop}
\end{align*}
\]

Pnum and Qnum are the number of times processes P and Q have printed their messages. Choose the correct answer, and write a brief justification.

(a) Pnum = Qnum-3  
(b) Pnum = Qnum+3  
(c) Pnum ≤ Qnum-3  
(d) Pnum ≤ Qnum+3  
(e) Pnum ≥ Qnum-3  
(f) Pnum ≥ Qnum+3  
(g) Pnum does not depend on Qnum  
(h) none of the above
(7) The following processes are executing concurrently. This code does not occur in a loop: each process Pi executes its worki_code exactly once.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;pre-code&gt;</td>
<td>&lt;pre-code&gt;</td>
<td>&lt;pre-code&gt;</td>
<td>&lt;pre-code&gt;</td>
<td>&lt;pre-code&gt;</td>
</tr>
<tr>
<td>work1_code</td>
<td>work2_code</td>
<td>work3_code</td>
<td>work4_code</td>
<td>work5_code</td>
</tr>
<tr>
<td>&lt;post-code&gt;</td>
<td>&lt;post-code&gt;</td>
<td>&lt;post-code&gt;</td>
<td>&lt;post-code&gt;</td>
<td>&lt;post-code&gt;</td>
</tr>
</tbody>
</table>

(a) Use semaphores to enforce the following constraint: Process P1 can start executing work1_code only when two or more of the other four processes have reached their <post-code>. State the initial value of any semaphore(s) you use. Add the necessary acquire and release calls to the <pre-code> and <post-code> sections of the processes.

(b) Use semaphores to enforce the following constraint: Process P1 can start executing work1_code only when at least one of the following is true:

- P2 has finished work2_code and P3 has finished work3_code
- P4 has finished work4_code and P5 has finished work5_code

Your solution should not force P2 or P3 or P4 or P5 to wait in the <post-code>. Each of these processes needs to be able to execute their <post-code> without having to wait for other processes to finish. Your code can use shared variables (such as counters) in addition to semaphores, or you can create additional processes to help with the synchronization.

A note about lab 3

Lab 3 is described on pages 77-80 of the course reader. This lab is challenging because it is your first time writing synchronization code.

As a first step, study and execute the given code where readers have priority. Make sure that you can tell from looking at the output that readers are indeed getting priority.

Then plan how to create code that gives priority to writers.

- For ideas on how to design and implement this type of synchronization, study the starvation free solution given on pages 32-37 of the course reader.
- As it says in step 3 of the lab instructions, first decide what semaphores and integer counters you need, and then figure out how to test the counters to decide whether a thread needs to acquire some semaphore or release some semaphore.

Plan carefully before you code!