1. **Reasons for synchronization (5 pts)**
Check those that are feasible reasons for the OS to provide synchronization kernel calls.

- [ ] coordinate use of I/O buffers
- [x] coordinate a parent process and a child process using a shared data structure
- [ ] coordinate a process printing to two different disks
- [x] prevent data from getting overwritten before it has been read
- [x] cause one process to begin execution after it has been signaled by another

2. **Synchronization Solutions (6 pts)**

a. List one disadvantage of using software solutions for synchronization of application programs.

- Error prone
- Busy waiting
- Some software solutions only work for two processes

b. Why must the code for the producer/consumer problem be synchronized?

- To prevent the producer from overwriting data that has not been consumed.
- To prevent the consumer from consuming old data (that it previously consumed.)

c. What is the atomic test-and-set(target) hardware instruction used for?

- To protect critical sections of code usually in the kernel.
- To implement P() and V() as critical sections.

3. **Semaphore Facts (6 pts)**

a. Which of the following is true of the semaphore operations wait and signal?

- are also specified by B() and W() -- *blahooeyglugen* (means "wait" in Dutch) and *whohagglegilgen* (means "signal" in Dutch)
- [x] must be executed atomically
- [x] operate on binary semaphores or counting semaphores
- [x] implemented with *busywaiting* version or *blocking* version.
- more error prone for use in mutual exclusion than software solutions
- [x] are used for both application level and kernel level synchronization problems
4. Semaphore Usage (8 pts)

Consider 20 processes, P1, P2, …, P20. Suppose the first ten processes are executing

P(mutex); critical section; V(mutex); /* semaphore mutex was initialized to 4 */

The second group of ten processes are executing;

P(cutex); critical section; V(cutex); /* semaphore cutex was initialized to 3 */

Assume the underlying OS kernel supports blocking implementation of P() and V(). Assume only these 20 processes are active in the system.

<table>
<thead>
<tr>
<th>Question</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>How many total processes could be executing in their critical sections concurrently?</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>How many processes could be blocked on semaphore mutex?</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>If P11 and P12 only are in critical section, how many processes could be blocked on cutex?</td>
<td>0</td>
<td>0 (sorry about the wrong answer in previous version of solutions)</td>
</tr>
<tr>
<td>How many processes could be executing V(mutex) code concurrently?</td>
<td>0</td>
<td>1 (V() is atomic so only one at a time)</td>
</tr>
<tr>
<td>How many processes could be executing either P(mutex) or P(cutex) concurrently?</td>
<td>0</td>
<td>2 (one each for P(mutex) and P(cutex))</td>
</tr>
<tr>
<td>How many processes could be in the ready queue if one process is blocked on mutex and one process is blocked on cutex?</td>
<td>0 assuming the others are all blocked on I/O 17 assuming none are doing I/O (one is running)</td>
<td>17 (one process is running)</td>
</tr>
</tbody>
</table>

Assume the OS implemented semaphores with a busywaiting implementation (instead of blocking implementation). Would any of your answers above be different? ___X___ YES  _____ No
Explain!