1. Context switching

The answer to question 4, hw1 includes the following code:

```c
int sp1, sp2;
void do_yield12(void) {
    do_switch(&sp1, &sp2);
}
```

(a) Can the new stack pointer (sp2) be passed by value, as in:

```c
int sp1, sp2;
void do_yield12(void) {
    do_switch(&sp1, sp2);
}
```

Yes. The do_switch() function will be able to take this value and use it to set the new stack pointer value.

(b) Can the old stack pointer (sp1) be returned by value, as in:

```c
int sp1, sp2;
void do_yield12(void) {
    sp1 = do_switch(&sp2);
}
```

No. In this code do_switch() is called in thread 1, but returns in thread 2, so the old stack pointer will not be stored in sp1.
2. Synchronization primitives

There are several ways you can distinguish the first and second thread, as shown above. Note that the semaphore solution has a subtle bug – after thread 2 executes “up(S2)”, a third thread could enter and modify ‘val’ before thread 1 is able to retrieve it. The fix to this is slightly hard to follow:

The order of operations is:

1. thread 1 enters S1, modifies count and val, leaves S1, and then sleeps on S2.
2. Thread 2 enters S1, modifies count and val, then signals S2 to wake up thread 1, and returns. Note that S1 is still held at this point.
3. Thread 1 wakes up, makes a copy of ‘val’, and then releases S1 before returning.

Note that if a third thread came in between steps 2 and 3 it would block on S1, which hasn't been released yet.
3. Address translation

First we split the two 20-bit page numbers into 10-bit halves:

\[ 70301 = 0111 \ 0000 \ 0011 \ 0000 \ 0001 \]
\[ = 01 \ 1100 \ 0000 \ (0x1C0), \ 11 \ 0000 \ 0001 \ (0x301) \]

\[ 08002 = 0000 \ 1000 \ 0000 \ 0000 \ 0010 \]
\[ = 00 \ 0010 \ 0000 \ (0x020), \ 00 \ 0000 \ 0010 \ (0x002) \]

Page 00000:
index 0x1C0 (addr 00000700) : pageno = 00001 (00002)
index 0x020 (addr 00000080) : pageno = 00002 (00001)

Page 00001: (00002)
index 0x301 (addr 00001C04) : pageno = 00003 (00004)

Page 00002: (00001)
index 0x002 (addr 00002008) : pageno = 00004 (00003)

Note that there are two correct answers, one shown in black and one in gray above.

4. Page replacement

a) Simulate LRU with a memory size of 4. For each step, show the pages resident in memory and whether the access is a hit or a miss. (15 points) Note that the bottom row is the least recently used, and gets evicted on a miss.

```
1  4  3  1  2  6  4  7  1  3  2  5  1  9  2  2  3  1  5  4
```

```
<table>
<thead>
<tr>
<th>H</th>
<th>H</th>
<th>H</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>
```

b) Simulate the clock algorithm with a memory size of 4.

```
1  4  3  1  2  6  4  7  1  3  2  5  1  9  2  2  3  1  5  4
```

```
<table>
<thead>
<tr>
<th>H</th>
<th>H</th>
<th>H</th>
<th>H</th>
<th>H</th>
<th>H</th>
<th>H</th>
<th>H</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \rightarrow ]</td>
<td>[ 1 ]</td>
<td>[ 1 ]</td>
<td>[ 1 ]</td>
<td>[ 1 ]</td>
<td>[ 1 ]</td>
<td>[ 1 ]</td>
<td>[ 1 ]</td>
<td>[ 1 ]</td>
</tr>
<tr>
<td>[ \rightarrow ]</td>
<td>[ 4 ]</td>
<td>[ 4 ]</td>
<td>[ 4 ]</td>
<td>[ 4 ]</td>
<td>[ 4 ]</td>
<td>[ 4 ]</td>
<td>[ 4 ]</td>
<td>[ 4 ]</td>
</tr>
<tr>
<td>[ \rightarrow ]</td>
<td>[ 3 ]</td>
<td>[ 3 ]</td>
<td>[ 3 ]</td>
<td>[ 3 ]</td>
<td>[ 3 ]</td>
<td>[ 3 ]</td>
<td>[ 3 ]</td>
<td>[ 3 ]</td>
</tr>
<tr>
<td>[ \rightarrow ]</td>
<td>[ 2 ]</td>
<td>[ 2 ]</td>
<td>[ 2 ]</td>
<td>[ 2 ]</td>
<td>[ 2 ]</td>
<td>[ 2 ]</td>
<td>[ 2 ]</td>
<td>[ 2 ]</td>
</tr>
</tbody>
</table>
```