1. [10] Given two processes A and B, define $A \rightarrow B$ if B is waiting for A. For example, there might be a shared lock that is held by A, and B has attempted to acquire it. Using this relation between processes, and an appropriate data structure based on it, how could you detect deadlock among a set of processes? (An example might be helpful.)

   The relation $A \rightarrow B$ defines an edge in a directed graph. Accumulate all of these edges (involving processes in the set as vertices) into a graph, and then look for cycles in the graph. If there is a cycle, then there is deadlock among the processes.

   [5] directed graph
   [5] cycle implies deadlock

   The graph does not need to be a resource allocation graph, because there is no explicit information about the resources, only about the processes. The difference is that the edges are Process $\rightarrow$ Process, not Process $\rightarrow$ Resource $\rightarrow$ Process.

   No timers, or discussion of waiting time, run time, scheduler priority, extra “watchdog” processes, etc.

   The symbol $\rightarrow$ is an arrow; it's possible that some people did not recognize this.

   The typical low score was 3 points, for not mentioning graph or cycle.

2. In a virtual memory system, the hardware and software cooperate to maintain several tables.

   (a) [5] Describe how the Translation Lookaside Buffer is related to the Page Table.

   The TLB is a cache for Page Table entries. Address translation from virtual address to physical address first looks for the (page number, frame number) pair in the TLB, and if not found there it will look in the PT, and then update the TLB so the next translation will be faster.

   (b) [5] Describe how the Page Table is related to the Frame Table.

   Each process has its own Page Table, which contains entries like (page number, frame number, etc.). There is only one Frame Table for the entire system; it contains entries like (frame number, page number, process number, etc.). Here, “etc.” means “usage and control bits”. The first part of the PT/FT entry could be implicit, if the table is an array.

   The Frame Table has all the information from the union of all the page tables, but it is arranged differently.

   An inverted page table looks like a frame table, but it is restricted to one process.
Frame tables are not related to stack frames, which hold function arguments, local variables and return addresses. Page tables do not contain pages, and frame tables do not contain frames; they contain information about pages and frames.

There must be some mechanism for finding a free frame, but the Frame Table is only a small part of that.

Most answers to this question were not very good.

(c) [10] Assuming a two-level hierarchical Page Table, describe (diagram, algorithm, etc.) how the bits of a virtual address are translated to the bits of a physical address, using a combination of the Translation Lookaside Buffer, the Page Table, and the Frame Table.

The diagram of the address bits should be something like

<table>
<thead>
<tr>
<th>virtual address</th>
<th>page number</th>
<th>offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1-level page table)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(2-level page table)</td>
<td>outer page number</td>
<td>inner page number</td>
</tr>
<tr>
<td>physical address</td>
<td>frame number</td>
<td>offset</td>
</tr>
</tbody>
</table>

First, assume we don’t get a page fault.

1. Separate the virtual address bits into (page number) and (offset within the page).

2. TLB –
Search for the page number; if it is found, extract the frame number, and proceed to step 4. This much is independent of the page table design.

3a. One-level page table –
Use the page number to index into the page table (or, as a search key into the page table) to find the frame number.

3b. Two-level page table –
Divide the one-level page table into pages (the inner page tables), and use another small table (the outer page table) to find them.
Separate the virtual address bits into (outer-page-number), (inner-page-number), (offset).
Use the outer-page-number to index into the outer page table, finding the address of one of the inner page tables. Use the inner-page-number to index into that table, finding the frame number.
4. Concatenate the (frame number) and (offset within the page) to form the physical address.

5. Update the appropriate referenced/modified bits in the page table and frame table. (but that wasn’t really part of the question)

On a page fault, the page table and frame table will need to be updated, but after that we proceed as described previously.