|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Page # | Process # | Valid bit | Rel Ptr. |
| 256 | 104 | 1 | 0 | 0 |
| 264 | 22 | 1 | 0 | 0 |
| 272 | 33 | 2 | 1 | 1 |
| 280 | 81 | 1 | 1 | 0 |
| 288 | 805 | 1 | 1 | 0 |
| 296 | 19 | 1 | 1 | -1 |
| 304 | 404 | 2 | 1 | -1 |
| 312 | 26 | 1 | 0 | 0 |

Consider an inverted page table system with hashing.

For each of the following virtual addresses, determine whether a page fault occurs and if a page fault does not occur, translate the virtual address to a physical address.

In each of these examples, the process making the request is number 1. Assume the inverted page table starts at address 256 (which corresponds to frame 0).

This system uses hashing. But we’ve given you in the question where the given page number will hash to. Once having hashed to the first value, use the relative pointers to see where to go from there.

a) 81 | 105 (81 hashes to address 280)

3 | 105

b) 19 | 2 (19 hashes to address 304)

5 | 2

c) 104 | 3 (104 hashes to address 0)

page fault

d) 33 | 100 (33 hashes to address 272)

page fault

\* If you want to cheat, the document in the course repo has the answers written in white text