1a)

* + has better performance
* + Each process manages its own threads
* - Blocking system call blocks all threads
* - OS blocks whole process during a page fault

bi) 6 different thread interleavings

* Y,x1,z,x2: x=0
* Z,x2,y,x1: x=0
* Y,z,x1,x2: x=-1
* Y,z,x2,x1: x=1
* Z,y,x1,x2: x=-1
* Z,y,x2,x1: x=1

Each value has equal chance of 1/3rd of occurring

ii) Lock around thread one by downing a semaphore for only thread1 to operate

iii) add semaphore operations after the y and z initialising such that the processes only continue once both of the have been initialised and then let whatever happens, happen

c) VIRTUALISATION IS NO LONGER ASSESSED WHOOPIE

It is now lol

1. Not really sure but I think it would be an obstacle depending on the type of hypervisor. If it’s a type 1 hypervisor that runs in Kernel mode, the Guest OS will run in User Mode so instructions from the guest OS will end up behaving differently than expected on the hypervisor
2. A type 2 hypervisor already runs inside another OS, so ti has to deal with all the overhead from that, plus it might have to do binary translation on the fly for certain instructions, whereas paravirtualization is pretty close to native since it acts the same way as “porting” the OS to the hypervisor’s architecture, with optimized calls and all.
3. Double paging problem arises when both one of the Guest OSes and the VMM decide to swap out the same page due to memory issues. The VM is not aware of the Gues OS’s actions beforehand and the same way around. The way to deal with this is using a balloon driver. The Guest OS will “inflate” the balloon by swapping pages to disk, and the more inflated a balloon is the more “pressure” the Guest OS has for swapping memory, which lets the VMM know how it should deal with memory/what to swap/what not to swap.

2a) The page table maps user logical addresses to physical addresses, providing this separation to allow virtual memory for processes.

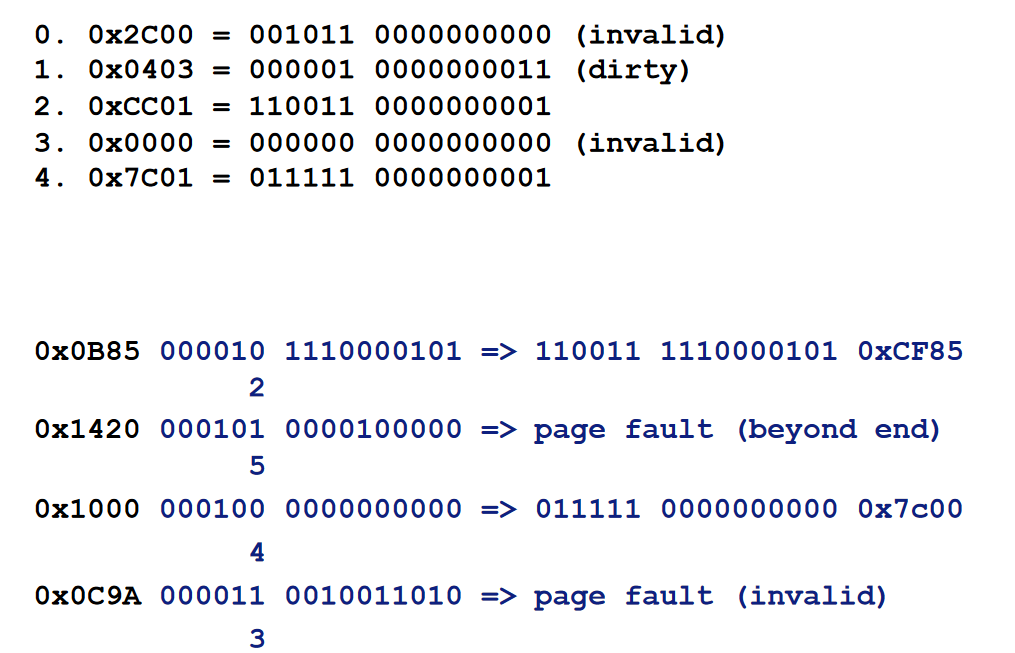
b)

* Page described by virtual address refers to page table entry that is not within page table’s bounds, e.g. page table has five entries but address asks for the sixth
* Invalid bit is set, in page table entry
* Process is attempting to write to a copy-on-write page

c) page size = 1024B = 2^10B => 6 bits for page number and 10 bits for page offset.

Thus total address space = 2 ^ 16 = 64KB

d)



e) Either page 0 or page 3 as their valid bits are not set so the frames they correspond to are no longer in memory?

I’d side with 0 more, in the above case we’ve accessed 2,4,3. Assuming a more ‘Last Recently Used’ approach, this leaves 0 and 1. Since 1 is valid and 0 is invalid we’d choose 0.