CS330 - Mid Semester Exam

Q1.

(i) (a), (b), (c), (d), (e)

(ii) (c)

(iii) (a), (b), (c), (d), (e)

(iv) {(a), (b), (c) (e) } OR {(a), (c) (e)}

(v) (a), (d), (e)

Q2.

(i) **False:** A child created using `vfork` shares the page table pages and memory pages with the parent. So, TLB flush is not required as CR3 is not switched when child is scheduled after the parent.

(ii) **False:** Inverted page table mapping (PFN to virtual page address) is required to correctly implement shared mappings. For example, when a page is shared by two processes (say P, Q) and if P deletes the shared page, page table entry of P corresponding to the shared page is cleared. To update the page table mapping of Q (as the page no longer exists) a reverse mapping is required.

(iii) **True:** If the user process accesses a virtual address outside the virtual address areas allocated to it by the OS, the OS can kill the process without changing any page table mapping. If cleaning up page tables during the process termination is considered as changing the entry, the answer can be False (To be explicitly mentioned).

(iv) **True:** If swapping is required to free PFNs to handle the page fault, Disk I/O is required (assuming swap device is part of a hard disk). Therefore, it is prudent to put the faulting process into waiting state.

(v) **True:** In a scenario when all accesses are to unique pages.

Q3. Output of the program will be as follows if we assume that all processes executing `p2` finish their execution before the processes executing `p1` finish. This is the most likely case because all processes executing the `printf()` in line #22 of `p1.c` have to sleep for one second. Even otherwise, only the output order may change.

```
var1=0 var2=0
var1=0 var2=0
var1=0 var2=0
var1=3 var2=3
var1=3 var2=3
```

Explanation:
Before entering the `for` loop, `pid = M`, where `M` is the PID of main process.

For loop iteration 1 (i=0)
A new child process, C_1 is created due to invocation of `fork()` by main process
$var1 = 1$ both in main process and C_1
"If" condition (line #17) is false for both main process and C_1, so `execl` is not invoked
$var2 = 1$ both in main process and C_1
For loop iteration 2 (i=1)

Two new child process, C_2 and C_3 are created due to invocation of `fork()` by main process and C_1
$var1 = 2$ in all four processes
"If" condition (line #17) is false only for main process. All other processes C_1, C_2, C_3 execute `p2` through `execl` call. Therefore, there will be three lines of output printing the values of var1 and var2 in p2.c which is zero.
$var2 = 2$ in main process. All other processes never come back here.
For loop iteration 3 (i=2)
A new child process, C_4 is created due to invocation of `fork()` by main process
$var1 = 3$ both in main process and C_4
"If" condition (line #17) is false for both main process and C_4, so `execl` is not invoked
$var2 = 3$ both in main process and C_4

Both main process and C_4 exit the for loop and sleep for one second. In the mean time, C_1, C_2 and C_3 execute the p2.c program binary and as a result `var1=0 var2=0` is printed thrice. After one second, main process and C_4 print the values of $var1 = 3$ and $var2 = 3$ (printed twice).

Q4.

(i) 2^{53} bytes

(ii) 2^{30} entries
# of entries in L4 = # of page table pages in L3 = 2^{10}
# of entries in L3 = # of page table pages in L2 = $2^{10} \times 2^{10}$
# of entries in L2 = $2^{10} \times 2^{10} \times 2^{10} = 2^{30}$

(iii) Given, Page table entry size = 64 bits, Page size = 8KB, Max PA = 2^{57} bytes
# of physical pages = $2^{57} \div 2^{13} = 2^{44}$
Therefore, # of bits required in PTE entries to specify the next level address = 44
# of flag bits = 5
So, unused bits = 64 - 5 - 44 = 15 bits

(iv) Virtual address size = 4GB = $2^{32} \div 2^{13} = 2^{19}$ pages
$\Rightarrow$ # of entries required at L1 = 2^{19}

Best case (3 Marks)
L1: 2^{19} entries require 2^9 page table pages
L2: 2^9 entries require 1 page table page
L3: 1 entry require 1 page table page
L4: 1 entry require 1 page table page
Total page table memory = $(2^9 + 1 + 1 + 1) \times 8$ KB

Worst case (4 Marks)
L1: 2^{19} entries can be spread across 2^{19} different page table pages (max. page table pages = 2^{30})
L2: 2^{19} entries can be spread across 2^{19} different page table pages (max. page table pages = 2^{20})
L3: 2^{19} entries can be spread across 2^{10} different page table pages (max. page table pages = 2^{10})
L4: 2^{10} entries require 1 page table page (max. page table page = 1)
Total page table memory = $(2^{19} + 2^{19} + 2^{10} + 1) \times 8$ KB

Q5.

(i) The child process will successfully complete. The parent process will access an invalid memory location causing segfault. This is true irrespective of order of execution as value of `ptr` is NULL in parent even if the child executes first and allocates legitimate address for `ptr`.

(ii) Introduction of `wait()` system call does not change the behavior of the parent, it still crashes. Because, the parent memory state is not impacted by the child's execution.

(ii) The parent will print 'A' and both the programs will terminate successfully. `vfork()` suspends the parent and executes the child using the parent's pages tables. Therefore, the parent will see all the changes in memory layout done by the child.