CS330 - End Semester Exam

Q1.

(i) (e)

(ii) {(a) (b) (d)} or {a, b}

(iii) (a) (b) (e)

(iv) (b) (c) (d)

(v) (a) (c) (e)

Q2.

(i) **False:** An OS has additional responsibilities than serving the application processes through implementation of some functionalities. For example, OS enforces isolation, ensures separation of privileges etc. which can not be guaranteed by a library.

(ii) **True:** Limited direct execution in principle allows the user processes to execute without OS intervention in general but with certain restrictions. Allowing user process to execute without the OS interventions and restricting the access to some sensitive resource at the same time require hardware support.

(iii) **False:** If mounted on two different mount points, modification in one will impact the other and result in consistency issues. The file systems assume exclusive control of the meta-data (like super block and inode) and will fail to provide correct behavior. For example, two inodes of a same file can update the data mapping resulting in an inconsistent state.

(iv) **True:** If the dynamic priority considers aging with schemes like priority elevation, priority boosting, it can address the problem of starvation.

(v) **True:** As threads of a process share memory, there is no need of OS orchestration if the communication can be synchronized using user-space locking techniques (spin locks).

Q3. **(1.25 * 4)**
Output of the program will be as follows,

```
Value is = 7
Value is = 7
Value is = 7
Value is = 7
```

Explanation:

Iteration (at end)	Process	nums	i
1	Parent	3, 4, 3	1
	Child-1 (created by Parent)	3, 4, 3	1
2	Parent	3, 4, 7	2
	Child-1	3, 4, 7	2
	Child-2 (created by Parent)	3, 4, 7	2
	Child-3 (created by Child-1)	3, 4, 7	2

Q4. The problem with the provided implementation is that concurrent insertions can create issues if they are inserted in the same position. For example, if the list already has three elements 1, 7, 15 and two threads try to insert 5 and 3 concurrently one of them can got lost. A possible solution is given below.

```
void insert_sorted(unsigned element)
{
   struct node *tmp = &head, *newnode;
   while(tmp->next && tmp->next->item < element)
          tmp = tmp->next;
   newnode = malloc(sizeof(struct node));
   if(!newnode)
          exit(-1);

   newnode->item = element;
   newnode->s = SEM_INIT(1);
   sem_wait(&tmp->s);
   /*Required to maintain sorted order*/
   while(tmp->next && tmp->next->item < element){
           sem_post(&tmp->s);
           tmp = tmp->next;
           sem_wait(&tmp->s);
   }
   newnode->next = tmp->next;
   tmp->next = newnode;
   sem_post(&tmp->s);
   }
   return;
}
```

Q5. One possible implementation using the scheme discussed in class.

```
int lock;
void write_lock()
{
   while(atomic_add(&lock, -0x1000000) != 0)
         atomic_add(&lock, 0x1000000);
}
void write_unlock()
{
   atomic_add(&lock, 0x1000000);
}
void read_lock()
{
     while(atomic_add(&lock, -1) < 0)
           atomic_add(&lock, 1);
}
void read_unlock()
{
     atomic_add(&lock, 1);
}
```

In the above implementation, the number of concurrent readers allowed to acquire the lock are 0x1000000. Note that, the above implementation also limits the number of writers trying for lock (one is allowed to enter obviously) if preemption is not disabled during acquiring the write lock.

Q6. **(3 + 5)**
Max no. of blocks pointed by single indirect block = 4KB / 4B = 1024

Max no. of blocks used by a directory = 4 + 1024 = 1028
(i) size of each dentry = 32 bytes
dentries per block = 4KB / 32B = 128
Max files in the directory = 128 * 1028 (= 131584)
(Calculation changes slightly if `.` and `..` are assumed as default)
(ii) No. of dentries (and inodes) to be found = 5
For each level,
Minimum block reads for dentry search = 1
Maximum block reads for dentry search = 1028 + 1 (+1 for indirect block)
# of block reads for child inode = 1
# of blocks to read the file itself = 1
minimum blocks to read = 5 * (1 (dir) + 1 (inode)) + 1 = 11 (if no inode caching assumed)
minimum blocks to read = 1 (inode) + 5 * 1 + 1 = 7 (if inode caching assumed)
max blocks to read = 5 * (1029 (dir) + 1 (inode)) + 1 = 5151

Q7. Open ended design question. Your design should avoid TLB flush on context switch as much as possible without compromising OS scalability (#of processes).

Q8. Open ended design question. Your design should optimize the memory usage for caching, explain how operations like lookup, read and update work.