Question 1

```
import itertools

tasks = ['a1', 'a2', 'b1', 'b2', 'b3', 'c1', 'c2', 'c3', 'd1', 'd2']

def main():
    perms = itertools.permutations(tasks)
    perms = filter(is_in_order, perms)
    permList = list(perms)
    valid_order = len(permList)
    print("Ordered permutations: ", valid_order)
    print(permList[0])
    permList = list(filter(is_valid, permList))
    print(permList[0])
    valid_constraint = len(permList)
    print("Valid permutations: ", valid_constraint)

    print(valid_constraint / valid_order)

def is_in_order(perm: list[str]) -> bool:
    if perm.index('a1') > perm.index('a2'):
        return False
    if perm.index('b1') > perm.index('b2') or perm.index('b2') > perm.index('b3'):
        return False
    if perm.index('c1') > perm.index('c2') or perm.index('c2') > perm.index('c3'):
        return False
    if perm.index('d1') > perm.index('d2'):
        return False
    return True

def is_valid(perm: list[str]) -> bool:
    if perm.index('a1') > perm.index('b3'):
        return False
    if perm.index('c2') > perm.index('a2'):
        return False
    if perm.index('d1') > perm.index('b2'):
        return False
    return True

if __name__ == '__main__':
    main()
```

Output

```
Ordered permutations:  25200
```

```
('a1', 'a2', 'b1', 'b2', 'b3', 'c1', 'c2', 'c3', 'd1', 'd2')
('a1', 'b1', 'c1', 'c2', 'a2', 'c3', 'd1', 'b2', 'b3', 'd2')
Valid permutations:  11144
0.44222222222222224
```

Thus, with 25200 ordered permutations, only 11144 are valid. This means that 44.22% of the ordered permutations are valid.

Question 2

All of them except D is impossible.

D (223) is possible -¿

Thread	Instruction	x	output
B	Get 4	4	
C	Get 4	4	
B	2 = 4 - 2	4	
C	6 = 4 + 2	4	
C	Set 6	6	
B	Set 2	2	
B	Print x	2	2
C	Print x	2	22
A	Get 2	2	22
A	3 = 2 + 1	2	22
A	Set 3	3	22
A	Print x	3	223

Question 3

Advantages

Simpler to construct and debug.

Disadvantages

Requires thoughtful planning and design of what each layer does and what layers are needed. Each layer adds some overhead when passing data between layers.

Question 4

Calling 3 forks would result in 8 processes.

Question 5

Only the Shared Memory Segments are shared.

Question 6

For 2 processing cores with a 65% parallel component we get a speedup of 1.48. For 4 processing cores with a 65% parallel component we get a speedup of 1.95.