HOMEWORK 2:

Name: Jeshalraj Thakaria

UFID: 8321-3912

Part 1:

2.3:

a) Following are the problems that cache designers may face if they include synonyms in their design:

1) Larger storage or cache size required to store the pairs as synonyms may cause more than one reference to same object just bound to different names.

2) More number of modifications required in the cache when any changes are made to the common object pointed by synonyms, as each and every entry of synonym pointing to that object needs to be changed. i.e. complexity is added to cache data management.

3) Cache designers would need to take a look into the concurrency and atomicity for the cache as multiple paths point to same object. Immediate changes would be needed in every synonym path to ensure that changes are reflected in the cache and the next time if we need to directly resolve name from the recent lookup list in cache, then we have the correct data. Additional info along with name, object pairs may be used to keep track if the data in cache is the most recent and valid for e.g. something analogous to dirty bit in memory caches that show that the data is modified.

b) If every object has a unique id then it solves many of the problems:

1) Every object has unique id therefore before storing it in recently lookup cache we can match the id with entries in the cache and simply modify the entry if a match is found, if not we can add the entry to the cache. This way there will not be any duplicates of the same data stored in cache, therefore we can work with lower cache sizes as compared to the caches in part a.

2) Concurrency and atomicity will not be a big problem as there would be only one entry with a given id to be modified if any changes are made to the object.

3) Cache structure can be kept simpler and cache data management will not be as complex as is the case with the part a.

2.4:

NO, Louis is not right as he overlooks many problems that might arise using his system:

1) The system assumed is multiuser system, each user will have his/her own ROT with all the search paths. There will arise a case where more than one user will have entries in ROT bound to the same object. In this case any changes made by one user will have to be reflected in the tables of all the other users' ROT. This will have its own overhead for modifying ROT tables.
2) Handling concurrency and atomicity is a complex task in a multi user system and providing this ROT table with entries to the directories or file path names would add to the complexity. Additional steps would be needed to ensure that all the entries accessed from ROT are updated.
3) In case there is no match in ROT for a given path, the system would need to turn to conventional search path. This will take additional time now to go through the search path and display result. Thus if there is a miss, users may feel the system working even slower than before.

C)

i. 1) First the root inode no. **1** is accessed in block no. **4** which points to root directory in block no. **14**
2) Then the "programs" name is linked to inode no. **7** which is in block no. **5**
3) From block no. 5 the next in path would be block no. **23** where the name "pong.c" is located.
4) The name "pong.c" is linked to the inode no. **9** which is in block no. **6** and has block no. **61** as next address.
5) Finally the block no. **61** has the data which is accessed.

ii.

1) 'Myhardlink' name is created in root directory(block no **14**) and bound to inode no. **9** and no change in inode table is made, just this name would be added in the root directory block no. **14**
2) 'Mysoftlink' name is created in root directory, bound to inode no. **10**(assumed an empty inode no. which is of type symbolic) that points to an empty block no. **27**(assumed to be empty) where the data '/programs/pong.c' is stored, now as the inode is of type symbolic the procedure would further treat this data as a path name and call general path to inode conversion code which would then traverse the path as in problem i) to get the program pong.c.