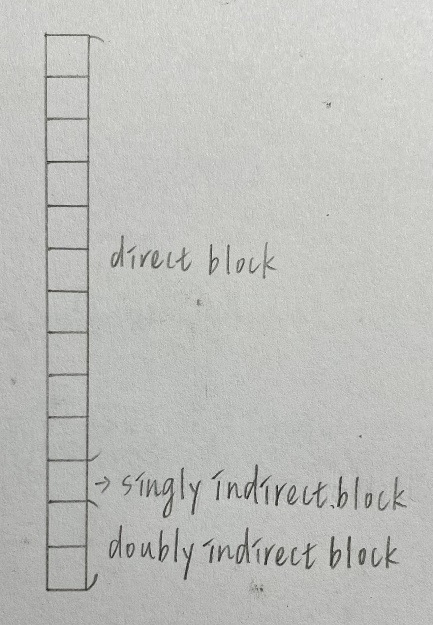
1.

(a) 一個singly indirect block可以指向256個block，而一個doubly indirect block可以指向256個singly indirect block，也就是256 \* 256 = 65536個block。由於10 + 256 + 2 \* 65536 = 131338 > 66666，因此我的inode structure為10個direct block，1個singly indirect block，以及2個doubly indirect block，如下圖所示：



在access第n個block時（n = 66666），會依照下列步驟依序檢查：

1. 檢查n是否小於10，若小於10則該block存在direct block中，由於66666 > 10，因此並非存在direct block中。

2. 檢查n是否小於10 + 256 = 266，若小於266則該block需藉由singly indirect block access，由於66666 > 266，因此並非藉由singly indirect block access。

3. 檢查n是否小於10 + 256 + 65536 = 65802，若小於65802則該block需藉由第一個doubly indirect block access，由於66666 > 65802，因此並非藉由第一個doubly indirect block access。

4. 檢查n是否小於10 + 256 + 2 \* 65536 = 131338，若小於131338則該block需藉由第二個doubly indirect block access，由於66666 < 131338，因此藉由第二個doubly indirect block access。

(b) 在symlink()中，會將該symbolic link所指到的file的檔名以及檔名的長度存進該symbolic link的inode中，因此在symlink(“/b”, “/a”)時會將檔名”/b”以及檔名的長度2存進/a的inode中。

在symopen()中，會檢查該file的type是否為SYMLINK，並且檢查該file的mode是否有O\_NOFOLLOW，若type為SYMLINK且mode沒有O\_NOFOLLOW，便會使用while迴圈讀出的inode中所存的檔名，並接著檢查該檔名的inode的type是否為SYMLINK，while迴圈會執行到該檔案不是symbolic link，或是深度超過20時便會終止以避免產生無窮迴圈。因此在symopen(“/a”)時，會檢查/a的type是否為SYMLINK，並且檢查該file的mode是否有O\_NOFOLLOW，由於/a為symbolic link，因此會跑while迴圈依序檢查該inode中所存的檔名所對應的檔案是否為symbolic link，由於/a的inode中所存的檔名為”/b”，且/b為一個file，因此while迴圈會終止並且開啟/b。

(c) 在sysopen()中，會呼叫namei()並呼叫namex()得到該檔名所對應的inode，而namex()中會使用skipelem()檢查目前的directory是否為symbolic link，若該inode的type為SYMLINK，則會將路徑改為該symbolic link所指到的路徑加上該symbolic link後餘的子路徑，並且回到原本路徑的inode開始重新檢視，while迴圈會執行到該路徑不存在symbolic link，或是深度超過20時便會終止以避免產生無窮迴圈。因此在namex(“/x/a/b”)時，會依序檢查/x、/x/a，以及/x/a/b是否為symbolic link，由於/x/a為symbolic link，因此會將路徑改為/x/a所指到的路徑/y加上該symbolic link剩餘的子路徑/b，因此路徑會變為/y/b，並且會再次從原本路徑的inode開始重新檢視，由於/y及/y/b皆不是symbolic link，因此while迴圈會終止並回傳/y/b的inode。

2. 無。