Lab-12
Name = Harshit Kedia
Roll Non- CS17B103

---

1) We know that xv6 only can handle file size upto 71,680 bytes using the inbuilt default file system. Our *create_file* uses a while loop and prints into a file named *myfile.txt* .
If we cross the size bound of 71,680, we get a panic call as:
panic("bmap: out of range");
Creating such huge file takes quite a lot of time in qemu xv6.

2) The new file type T_DUP_BLK_FILE is successfully supported. As mentioned in question, it allocates two successive free blocks when available to the *addrs* arrray of inode structure.
For implementing this, whenever we visit functions *readi, writei, bmap, balloc.*
For this file type, the calcuation of block number and offset to access is calculated on function readi/writei, as the bmap function returns the aoppropriate block number from hard disk accordingly. We check *if (offset>512), then block+1* needs to be acessed, *else block* returned by bmap is itself sufficient.
For allocation we check the bitmp array and find two consecutive positioned 0 mapped (unallocated) blocks in disk. Also after applying patch of q2, the files henceforth created will be of type DUP_BLK_FILE and maximum size supported is now 77,824 bytes.
Creation, Reading and Writing for this file is successfully implemented. On crossing the size limit, same panic error is called as part-1.

3) The third type, called T_EXT_FILE is indeed an interesting type, we allocate all the consecutive free blocks available, and needed to a single entry in *inode->addrs array*. Hence we can store huge amount of data in this format (if consecutive empty blocks are available). The 2nd entry in *inode->addrs* stores the ending block number in this consecutive blocks. Hence we can support 6 such interval type block lists.
For balloc, we find the first free block in the bitmap, if the block number == previous ending block +1, then these blocks are in succession. Hence we can merge this block in to the previous interval (check for the presence of previous interval first, it may be the first time any memory is allocated to the file). So update as *inode->addrs[b_no+1]++;* else if the block is successor of previous used block, this is a beginning of a new continous set of blocksthat can be declared to the file. We can have a total of 6 such intervals supported in array size of NDIRECT=12.
The indirectly referenced blocks are not modified, so if the 6 intervals are somehow filled, we can address to them by (*offset - BSIZE*Σ(End_block_no – Start_block_no +1)*)/*BSIZE*.
The same formula can be used to parse through the filled intervals to find the interval number during future read/write's.
Creation, Reading, Writing of this file type is successfully implemented. The maximum size of this type is not fixed, it depends on the external fragmentaion of the disk, it can be as low as *(128+6) blocks* (only one free block available at a time consecutivey in disk) to as high as potentially infinity.