```c
#include "type.h"

void set_bit(char *buf, int bit)
{
    int i, j;
    //mailmans
    i = bit / 8;
    j = bit % 8;
    buf[i] |= (1 << j);
}

void decFreeInodes(int dev)
{
    char buf[BLKSIZE];
    //decrement free inode count in SUPER block

    get_block(dev, 1, buf); //get super block
    sp = (SUPER *)buf;//cast the buf to super block
    sp->s_free_inodes_count--;//decrement free inode count in super
    put_block(dev, 1, buf);//put the block back into memory

    get_block(dev, 2, buf);//get group descriptor block
    gp = (GD *)buf;//cast the buf to group descriptor
    gp->bg_free_inodes_count--;//decrement free inode count in group descriptor
    put_block(dev, 2, buf);//put the block back into memory
}

int tst_bit(char *buf, int bit)
{
    int i, j;
    //mailmans
    i = bit / 8;
    j = bit % 8;

    if(buf[i] & (1 << j))
        return 1;

    return 0;
}


int ialloc(int dev)
{
    int i;
    char buf[BLKSIZE];

    get_block(dev, imap, buf);//gets the block of the inode bitmap

    for(i = 0; i < ninodes; i++)
    {
        if(tst_bit(buf, i) == 0) //if the bit at the current super_pot in the bitmap
is 0 then we have a free inode
        {
            set_bit(buf, i); //set the bit of position i in imap block to 1
            decFreeInodes(dev); //decrement free inode count
            put_block(dev, imap, buf); //puts the block back into memory

            return i + 1;
        }
    }
    printf("Error, there are no free inodes.\n");
    return 0;
```

```c
}

//allocates block
int balloc(int dev)
{
    int i;
    char buf[BLKSIZE];

    get_block(dev, bmap, buf); //open block map

    for(i = 0; i < nblocks; i++)//loop
    {
        if(tst_bit(buf, i) == 0)//checks to see if it is occupied or not
        {
            set_bit(buf, i); //sets the bit to 1
            decFreeInodes(dev); //decrement count of nodes
            put_block(dev, bmap, buf);//put block back into memeory

            return i;
        }
    }
    printf("Error, there are no free blocks.\n");
    return 0;
}

//deallocates an inode
int idealloc(int dev, int ino)
{
    char buf[1024];
    int byte;
    int bit;

    get_block(dev, imap, buf);//call get block to get the inode bitmap

    byte = ino / 8;
    bit = ino % 8;

    buf[byte] &= ~(1 << bit);//clear the bit that indicates whether or not the node
is occupied

    put_block(dev, imap, buf);//put the imap block back into memory

    get_block(dev, 1, buf);//get super block
    sp = (SUPER *)buf;//cast the buf to super type
    sp->s_free_blocks_count++; //increment number of free block in the super block
    put_block(dev, 1, buf);//put block back into memory

    get_block(dev, 2, buf);//get group descriptor block
    gp = (GD *)buf;//cast to group descriptor
    gp->bg_free_blocks_count++;//increment number of free blocks in group descriptor
    put_block(dev, 2, buf);//put block back into memory
}

//deallocate block
int bdealloc(int dev, int bno)
{
    char buf[1024];
    int byte;
    int bit;

    get_block(dev, bmap, buf);//get block of block bitmap from memory
    //mailmans
    byte = bno / 8;
    bit = bno % 8;

    buf[byte] &= ~(1 << bit);//clears the bit that idicates whether it is occupied or
not

    put_block(dev, bmap, buf);//put the block back into memory

    //free blocks
    get_block(dev, 1, buf);//get super block
    sp = (SUPER *)buf;//cast to super pointer
    sp->s_free_blocks_count++;//increment free block count in super block
    put_block(dev, 1, buf);//put block back into memory

    get_block(dev, 2, buf);//get group descriptor block
    gp = (GD *)buf;//cast to group descriptor pointer
    gp->bg_free_blocks_count++;//increment free block count in group descriptor
    put_block(dev, 2, buf);//put block back into memory

    return 0;
}
```