```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <time.h>
#include <ext2fs/ext2_fs.h>

typedef struct ext2_super_block SUPER;
typedef struct ext2_group_desc  GROUPD;
typedef struct ext2_inode       INODE;
typedef struct ext2_dir_entry_2 DIR;


void  put_block(int dev, int block, char* buffer);
char* get_block(int dev, int block);

INODE get_inode(int dev, int inode_number);

SUPER* get_super(int dev);
void put_super(int dev, SUPER* buffer);

GROUPD* get_GROUPD(int dev);
void put_GROUPD(int dev, GROUPD* buffer);

char* get_bmap(int dev);
void put_bmap(int dev, char* buffer);

void set_free_inodes(int dev, int change);
void set_free_blocks(int dev, int change);

int balloc(int dev);
int bfree (int dev, int block);

int get_block_size  (int dev);
int get_blocks_count(int dev);
int get_inodes_count(int dev);

int test_bit (char* buffer, int bit);
void set_bit  (char** buffer, int bit);
void clear_bit(char** buffer, int bit);



void put_block(int dev, int block, char* buffer)
{
    int bytes_written;
    int block_size = get_block_size(dev);

    lseek(dev, (long)(block * block_size), 0);
    bytes_written = write(dev, buffer, block_size);
    free(buffer);
}

char* get_block(int dev, int block)
{
    int bread;
    int block_size = get_block_size(dev);
    char* buffer = (char*)malloc(block_size);

    lseek(dev, (long)(block * block_size), 0);
    bread = read(dev, buffer, block_size);
```

```
    return buffer;
}

INODE get_inode(int dev, int inode_number)
{
    SUPER* sp = get_super(dev);
    GROUPD* gp = get_GROUPD(dev);

    int block_group = (inode_number - 1) / sp->s_inodes_per_group;
    int local_index = (inode_number - 1) % sp->s_inodes_per_group;

    int block = (block_group * sp->s_blocks_per_group) + gp->bg_inode_table;

    INODE* inode_table = (INODE*)get_block(dev, block);
    INODE inode = inode_table[local_index];

    free(sp);
    free(gp);
    free(inode_table);
    return inode;
}

SUPER* get_super(int dev)
{
    int bread;

    SUPER* sp = (SUPER*)malloc(1024);

    lseek(dev, (long)1024, 0);
    bread = read(dev, sp, 1024);
    return sp;
}

void put_super(int dev, SUPER* buffer)
{
    int written;

    lseek(dev, (long)1024, 0);
    written = write(dev, buffer, 1024);
    free(buffer);
}

GROUPD* get_GROUPD(int dev)
{

    if(get_block_size(dev) > 1024 + 1024)
        return (GROUPD*)get_block(dev, 1);

    return (GROUPD*)get_block(dev, 2);
}

void put_GROUPD(int dev, GROUPD* buffer)
{
    if(get_block_size(dev) > 1024 + 1024)
        put_block(dev, 1, (char*)buffer);
    else
        put_block(dev, 2, (char*)buffer);
}

char* get_bmap(int dev)
{
    GROUPD* gp = get_GROUPD(dev);
```

```
    char* bmap = get_block(dev, gp->bg_block_bitmap);

    free(gp);
    return bmap;
}

void put_bmap(int dev, char* buffer)
{
    GROUPD* gp = get_GROUPD(dev);
    put_block(dev, gp->bg_block_bitmap, buffer);

    free(gp);
}



void set_free_inodes(int dev, int change)
{
    SUPER* sp = get_super(dev);
    sp->s_free_inodes_count += change;
    put_super(dev, sp);

    GROUPD* gp = get_GROUPD(dev);
    gp->bg_free_inodes_count += change;
    put_GROUPD(dev, gp);
}

void set_free_blocks(int dev, int change)
{
    SUPER* sp = get_super(dev);
    sp->s_free_blocks_count += change;
    put_super(dev, sp);

    GROUPD* gp = get_GROUPD(dev);
    gp->bg_free_blocks_count += change;
    put_GROUPD(dev, gp);
}


int balloc(int dev)
{
    int block_count = get_blocks_count(dev);
    char* bmap = get_bmap(dev);

    int  i;
    for (i = 0; i < block_count; i++)
    {
        if (test_bit(bmap, i) == 0)
        {
            set_bit(&bmap, i);
            set_free_blocks(dev, -1);
            put_bmap(dev, bmap);

            return i;
        }
    }
    return -1;
}


int bfree(int dev, int block)
{
```

```
    char* bmap = get_bmap(dev);

    clear_bit(&bmap, block);
    set_free_blocks(dev, +1);

    put_bmap(dev, bmap);
}

int get_block_size(int dev)
{
    SUPER* sp = get_super(dev);
    int block_size = 1024 << sp->s_log_block_size;

    free (sp);
    return block_size;
}
int get_blocks_count(int dev)
{
    SUPER* sp = get_super(dev);
    int blocks_count = sp->s_blocks_count;

    free(sp);
    return blocks_count;
}

int get_inodes_count(int dev)
{
    SUPER* sp = get_super(dev);
    int inodes_count = sp->s_inodes_count;

    free(sp);
    return inodes_count;
}

int test_bit(char* buffer, int bit)
{
    int byte = bit / 8;
    bit %= 8;

    if (buffer[byte] & (1 << bit))
        return 1;
    return 0;
}

void set_bit(char** buffer, int bit)
{
    int byte = bit / 8;
    bit %= 8;

    (*buffer)[byte] |= (1 << bit);
}

void clear_bit(char** buffer, int bit)
{
    int byte = bit / 8;
    bit %= 8;

    (*buffer)[byte] &= ~(1 << bit);
}
```