```c
#include "type.h"

/*
Breakdown pathname into parent path and child path.
Get the ino of that parent pathname.
Load that ino from disk into memory.

Allocate inode and populates inode with correct information
Allocates one block and creates . and .. dir entries in that block
Puts inode back onto disk.
*/

void make_dir(char path[124])
{
        int i, ino;
        MINODE *pmip;
        INODE *pip;

        char buf[1024];
        char temp1[1024], temp2[1024];
        char parent_name[1024], child_name[1024];

        strcpy(temp1, path);
        strcpy(temp2, path);
        strcpy(parent_name, dirname(temp1));
        strcpy(child_name, basename(temp2));

        //get parent ino
        ino = get_Inode(running->cwd, parent_name);
        pmip = iget(dev, ino);//get memory inode
        pip = &pmip->INODE;//point to the indivdual inode that is in memory

        if(!pmip)//checks to see whether or not the parent inode exists
        {
                printf("Error, parent does not exist.\n");
                return;
        }

        //check if dir
        if(!S_ISDIR(pip->i_mode))//checks to see if parent inode is a file
        {
                printf("Error, parent is not a directory.\n");
                return;
        }

        if(get_Inode(running->cwd, path) != 0)//check if dir already exists
        {
                printf("Error, %s already exists.\n", path);
                return;
        }

        my_mkdir(pmip, child_name);//calls mkdir with parent memory pointer and child
name

        //++link and update time
        pip->i_links_count++;//update the link count for the parent inode pointer
        pip->i_atime = time(0L);//update access time on parent inode
        pmip->dirty = 1;//set parent inode dirty to 1


        iput(pmip);//disuper_pose of memory inode pointer
        return;
}

/*
Loads dir blocks into buf, goes to last dir entry in that block,
changes that last rec_len to fit new entry
If block won't fit dir entry iterate to next block
*/

int enter_name(MINODE *mip, int myino, char *myname)
{
        int i;
        INODE *parent_ip = &mip->INODE;

        char buf[1024];
        char *cp;
        DIR *dp;

        int need_len = 0, ideal = 0, remain = 0;
        int block_num = 0, block_size = 1024;

        for(i = 0; i < parent_ip->i_size / BLKSIZE; i++) //walks through
        {
                if(parent_ip->i_block[i] == 0) //no iblocks
                        break;

                block_num = parent_ip->i_block[i]; //sets the block num to

                get_block(dev, block_num, buf);

                dp = (DIR*)buf;
                cp = buf;

                need_len = 4 * ( (8 + strlen(myname) + 3) / 4);

                while(cp + dp->rec_len < buf + BLKSIZE)//walks to the end of the entry
                {
                        cp += dp->rec_len;
                        dp = (DIR*)cp;
                }

                cp = (char*)dp;//casts the dir pointer to a char pointer

                //ideal length uses name len of last dir entry
                ideal = 4 * ( (8 + dp->name_len + 3) / 4);

                remain = dp->rec_len - ideal;//gets remaining size

                if(remain >= need_len)//checks to see if we are good on space
                {
                        //set the stats of the directory
                        dp->rec_len = ideal;

                        cp += dp->rec_len;
                        dp = (DIR*)cp;


                        dp->inode = myino;
                        dp->rec_len = block_size - ((u32)cp - (u32)buf);
                        dp->name_len = strlen(myname);
                        dp->file_type = EXT2_FT_DIR;
                        strcpy(dp->name, myname);
                        put_block(dev, block_num, buf);

                        return 1;
                }
        }



        block_num = balloc(dev);
        parent_ip->i_block[i] = block_num;

        parent_ip->i_size += BLKSIZE;
        mip->dirty = 1;

        get_block(dev, block_num, buf);

        dp = (DIR*)buf;
        cp = buf;

        dp->inode = myino; //set inode to myino
        dp->rec_len = 1024; //reset length to 1024
        dp->name_len = strlen(myname); //set name to myname
        dp->file_type = EXT2_FT_DIR; //set dir type to EXT2 compatible
        strcpy(dp->name, myname); //set the dir pointer name to myname

        put_block(dev, block_num, buf);

        return 1;
}

void my_mkdir(MINODE *pmip, char *child_name)
{
        int ino = ialloc(dev);
        int block_num = balloc(dev);
        int i;

        MINODE *mip = iget(dev, ino);
        INODE *ip = &mip->INODE;

        char *cp, buf[1024];
        DIR *dp;

        ip->i_mode = 0x41ED; //OR 040755: DIR type and permissions
        ip->i_uid  = running->uid; //owner uid
        ip->i_gid  = running->gid; //group id
        //we set the size to blksize to because that is the size of a dir
        ip->i_size = BLKSIZE; //size in bytes
        ip->i_links_count = 2; //links count=2 because of . and ..
        ip->i_atime = time(0L); //set access time to current time
        ip->i_ctime = time(0L); //set creation time to current time
        ip->i_mtime = time(0L); //set modify time to current time

        //. and ..
        ip->i_blocks = 2;
        ip->i_block[0] = block_num;

        for(i = 1; i < 15; i++)//sets the data blocks to unoccupied
                ip->i_block[i] = 0;

        mip->dirty = 1;
        iput(mip);

        // . and ..
        get_block(dev, block_num, buf);//gets the block that was created for the dir

        dp = (DIR*)buf;
        cp = buf;

        //creates the . entry
        dp->inode = ino;
        dp->rec_len = 4 * (( 8 + 1 + 3) / 4);
        dp->name_len = strlen(".");
        dp->file_type = (u8)EXT2_FT_DIR;
        dp->name[0] = '.'; //name .

        cp += dp->rec_len;//increments to the next super_pace in the dir
        dp = (DIR*)cp;

        //creates the .. entry
        dp->inode = pmip->ino;
        dp->rec_len = 1012;
        dp->name_len = strlen("..");
        dp->file_type = (u8)EXT2_FT_DIR;
        dp->name[0] = '.';
        dp->name[1] = '.'; //second makes (..)

        put_block(dev, block_num, buf);//puts the block back into memory

        enter_name(pmip, ino, child_name);

        return 1;
}
```