```
#include "type.h"


int isEmptyDir(MINODE *mip)
{
        char buf[1024];
        INODE *ip = &mip->INODE; //set the inode pointer the memory pointer inode
        char *cp;
        char name[64];
        DIR *dp;

        if(ip->i_links_count > 2)//checks to see if the dir has more than 2 links
        {
                printf("Error, the dir has files.\n");
                return 1;
        }
        else if(ip->i_links_count == 2)//it has two links but we still need to check
whether or not it is empty
        {
                //could still have files
                if(ip->i_block[1])//if the inode is not zero then check it
                {
                        get_block(dev, ip->i_block[1], buf); //get the block that is

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

                        while(cp < buf + 1024)//while we are inside the buf
                        {
                                strncpy(name, dp->name, dp->name_len);//copy the
directory name into name to be checked
                                name[dp->name_len] = 0; //set the last index in the
array to null

                                if(strcmp(name, ".") != 0 && strcmp(name, "..") !=
0)//check to see if the names are anything besides . and ..
                                {
                                        //not empty
                                        printf("Directory is not empty.\n");
                                        return 1;
                                }
                        }
                }
        }
        else
        {
                printf("Dir is empty\n");
                return 0;
        }
}

void rm_child(MINODE *parent, char *name)
{
        int i;
        INODE *p_ip = &parent->INODE;
        DIR *dp;
        DIR *prev_dp;
        DIR *last_dp;
        char buf[1024];
        char *cp;
        char temp[64];
        char *last_cp;
        int start, end;


        for(i = 0; i < 12; i++)//loop through direct blocks
        {
                if(p_ip->i_block[i] == 0)//if the inode block is not occupied return
                        return;

                get_block(dev, p_ip->i_block[i], buf);//
                cp = buf;
                dp = (DIR*)buf;


                while(cp < buf + 1024)//while we are inside the buf
                {
                        strncpy(temp, dp->name, dp->name_len);//cope the dir name
into the temp variable
                        temp[dp->name_len] = 0;//set the last index to nul


                        if(!strcmp(temp, name))//if the names are the same
                        {
                                if(cp == buf && cp + dp->rec_len == buf + 1024)//
there is only one item in the block
                                {
                                        free(buf);
                                        bdealloc(dev, ip->i_block[i]);

                                        p_ip->i_size -= 1024;//substract 1024 from
the inode block size

                                        while(p_ip->i_block[i + 1] && i + 1 < 12)//
direct block loop
                                        {
                                                i++;
                                                get_block(dev, p_ip->i_block[i], buf);
                                                put_block(dev, p_ip->i_block[i - 1],
buf);
                                        }
                                }
                                else if(cp + dp->rec_len == buf + 1024)//delete last
entry
                                {
                                        prev_dp->rec_len += dp->rec_len;
                                        put_block(dev, p_ip->i_block[i], buf);
                                }
                                else
                                {
                                        //not last entry
                                        last_dp = (DIR*)buf;
                                        last_cp = buf;

                                        //last entry
                                        while(last_cp + last_dp->rec_len < buf +
BLKSIZE)//last entry
                                        {
                                                last_cp += last_dp->rec_len;
                                                last_dp = (DIR*)last_cp;
                                        }

                                        last_dp->rec_len += dp->rec_len;
                                        start = cp + dp->rec_len;
                                        end = buf + 1024;

                                        memmove(cp, start, end - start);//copy over
the start of the record into current

                                        put_block(dev, p_ip->i_block[i], buf);//

                                }

                                parent->dirty = 1;//set the parent inode dirty to 1
                                iput(parent);//dispose of the parent memory inode
                                return;
                        }

                        prev_dp = dp;//assign previous directory
                        cp += dp->rec_len;//increment the current pointer
                        dp = (DIR*)cp;//cast current pointer to directory
                }
        }

        return;
}

/*
Gets minode of parent in pathname
Gets minode of child
Checks if dir is empty and can be removed, deallocates bno's inside and ino of dir
Then removes dir entry of child from parent

if first/middle, remove dir entry, move all entires after left rec_len of removed,
and add rec_len to last
if last, remove dir entry, add rec_len to previous dir entry
if only, deallocate data block, modify parent's file size, move all subsequent data
blocks left one
*/

void remove_dir(char *path)
{
        int i;
        int ino, parent_ino;
        MINODE *mip;
        MINODE *p_mip;
        INODE *ip;
        INODE *p_ip;
        char temp[64], child[64];

        if(!path)//checks to see if the path is valid
        {
                printf("Error, no directory name given.\n");
                return;
        }

        strcpy(temp, path);
        strcpy(child, basename(temp));//gets the basename of the path

        ino = get_Inode(running->cwd, path);//gets the inode that is associated with
the path entered
        mip = iget(dev, ino);//gets the memory inode of the path entered

        if(!mip)//checks to see if a memory inode was found, if not print error
        {
                printf("Error, mip does not exist.\n");
                return;
        }

        if(!S_ISDIR(mip->INODE.i_mode))//checks to see if it is a dir or not
        {
                printf("Error, %s not a directory.\n", temp);
                return;
        }

        if(isEmptyDir(mip))//checks to see if the dir is empty
        {
                printf("Error, directory not empty.\n");
                return 0;
        }

        ip = &mip->INODE;//set the inode pointer to the memory inode pointer

        findino(mip, &ino, &parent_ino);//looks through the inode to find the
        p_mip = iget(dev, parent_ino);//gets the parent memory inode pointer
        p_ip = &p_mip->INODE;//sets the parent inode pointer to the memory inode
pointer

        for(i = 0; i < 15 && ip->i_block[i] != 0; i++)//loops through and calls block
dealocate to deallocate the blocks
                bdealloc(dev, ip->i_block[i]);

        idealloc(dev, ino);//deallocates the inode

        rm_child(p_mip, child);//removes child entry of parent memory inode

        //updates the parent inode stats
        p_ip->i_links_count--;
        p_ip->i_atime = time(0L);
        p_ip->i_mtime = time(0L);
        p_mip->dirty = 1;

        iput(p_mip);//disposes of the parent memory inode
        mip->dirty = 1;
        iput(mip);//disposes of the memory inode

        return;
}
```