```c
#include "type.h"

/*
Finds logical data based off of offset (mailman's algorithm) and loads correct block
From there it'll copy the data from buf into the file starting at the offset
Put the block back
Return the number of bytes written
*/

int write_helper(int file_d, char buf[], int nbytes)
{
    int i, j;
    int *ip;
    int counter = 0;
    int size =0;
    int remain;
    int logic_block, startByte;
    int cur_block;
    int indirect_cur_block, indirect_off;
    int *indirect, *double_indirect;
    int double_index = 0, indirect_index = 0, double_block = 0;

    OFT *oftp;
    MINODE *mip;
    char write_buf[1024];
    char *cur_pointer, *cq = buf;

    if(file_d < 0 || file_d >= NFD)
    {
        printf("Error, invalid file descriptor.\n");
        return;
    }

    //loop OFT
    for(i = 0; i < NOFT; i++)
    {
        if(OpenFileTable[i].inodeptr == running->fd[file_d]->inodeptr)
        {
            oftp = running->fd[file_d];
            break;
        }
    }

    if(!oftp || (oftp->mode != 1 && oftp->mode != 2 && oftp->mode != 3))
    {
        printf("Error, mode not set to write.\n");
        return;
    }

    mip = oftp->inodeptr;


    //while there are bytes to be written
    while(nbytes)
    {
        //mailmans
        logic_block = oftp->offset / BLKSIZE;
        startByte = oftp->offset % BLKSIZE;

        if(logic_block < 12) //direct blocks
        {
```

```
            if(mip->INODE.i_block[logic_block] == 0) //unoccupied this need to
allocate
                mip->INODE.i_block[logic_block] = balloc(mip->dev);//allocate a block

            cur_block = mip->INODE.i_block[logic_block];//assign our block variable
to the recently allocated block

        }
        else if(logic_block >= 12 && logic_block < 256 + 12)//else if we are in the
indirect blocks
        {
            //indirect

            if(!mip->INODE.i_block[12])//if there is no block that the indirect
points to then we need to make one
            {
                mip->INODE.i_block[logic_block] = balloc(mip->dev); //allocate a
block for the indirect pointer
            }
            //zero out
            get_block(mip->dev, mip->INODE.i_block[12], write_buf);//get the block
that was allocated
            indirect = (int *)write_buf;

            if(indirect[logic_block - 12] == 0)
            {
                indirect[logic_block - 12] = balloc(mip->dev);
                put_block(mip->dev, mip->INODE.i_block[12], write_buf);
            }
            cur_block = indirect[logic_block - 12];
        }
        else
        {
            //double indirect
            if(mip->INODE.i_block[13] == 0)//checks to see if there is a block
allocated
            {
                mip->INODE.i_block[13] = balloc(mip->dev);//allocates a block for
that inode
            }

            double_index = (logic_block - (12+256)) / 256;
            indirect_index = (logic_block - (12+256)) % 256;

            get_block(mip->dev, mip->INODE.i_block[13], write_buf);
            double_indirect = (int *)write_buf;

            cur_block = double_indirect[double_index];

            if(!cur_block)//cur_block doesnt exist
            {
                double_indirect[double_index] = balloc(mip->dev);
                cur_block = double_indirect[double_index];
                put_block(mip->dev, mip->INODE.i_block[13], write_buf);
            }

            get_block(mip->dev, cur_block, write_buf);//get the block from memory
            double_block = cur_block;

            indirect = (int *)write_buf;

            cur_block = indirect[indirect_index];
```

```
            if(!cur_block)//checks to see if the farthest block is allocated
            {
                indirect[indirect_index] = balloc(mip->dev);
                cur_block = indirect[indirect_index];
                put_block(mip->dev, double_block, write_buf);
            }
        }

        get_block(mip->dev, cur_block, write_buf);//gets the block
        cur_pointer = write_buf + startByte;//sets the current pointer
        remain = BLKSIZE - startByte;//gets remaining size, which at this point is
total size

        if(remain <= nbytes)
                {
                        size = remain;
                }
                else if(nbytes <= remain)
                {
                        size = nbytes;
                }



                memcpy(cur_pointer, buf, size);
                oftp->offset += size;
                if(oftp->offset > oftp->inodeptr->INODE.i_size)
                {
                        mip->INODE.i_size += size;
                }
                counter += size;
                nbytes -= size;
                remain -= size;

        put_block(mip->dev, cur_block, write_buf);//put the block that we just wrote
to, back to memory
    }

    mip->dirty = 1;//set memory inode to 1
    return counter;
}

void my_write(char *path)
{
    int i, file_d, nbytes;
    char *buf = (char*)malloc( (strlen(third) + 1) * sizeof(char*) );

    OFT *ofile_pointer;

    //checks
    if(!path)
    {
        printf("Error, no file name was given.\n");
        return;
    }

    if(!third)
    {
        printf("Error, no text to write.\n");
        return;
    }
```

```
    file_d = atoi(path);
    for(i = 0; i < NOFT; i++)
    {

        ofile_pointer = &OpenFileTable[i];

        if(ofile_pointer->refCount == 0)
        {
            printf("Error, bad file descriptor.\n");
            return;
        }

        //check mode
        if(i == file_d)
        {
            if(ofile_pointer->mode == 1 || ofile_pointer->mode == 2 || ofile_pointer-
>mode == 3)
                break;

            else
            {
                printf("Error, wrong mode for writing.\n");
                return;
            }
        }
    }

    strcpy(buf, third);
    nbytes = strlen(buf);
    write_helper(file_d, buf, nbytes);
    return;
}
```