```c
#include "type.h"

int read_file(char *path)
{
        int nbytes = atoi(third), flag = 0;// atoi third to get the number of bytes
we want to read
        int fd = 0;

        char buf[nbytes + 1];//creates buf of size nbytes+1

        MINODE *mip;
        INODE* ip;

        strcpy(buf, "");//clears out the buf

        if (!strcmp(path, ""))//checks to see if there was a file descriptor entered
        {
                printf("No file descriptor super_pecified.\n");
                return;
        }

        fd = atoi(path);//converts the path to a file descriptor number

        if (!strcmp(third, ""))//checks to see if there was a byte amount
super_pecified
        {
                printf("No byte amount super_pecified.\n");
                return;
        }

        flag = read_helper(fd, buf, nbytes);//tries to read from the file descriptor
into the buf, nbytes
        if (flag == -1)//checks if the read helper was successful
        {
                printf("Error, couldn't read file.\n");
                strcpy(third, "");
                return;
        }

        buf[flag] = '\0';//goes to the end of the buf an inserts a null terminator
        printf("%s\n", buf);//prints the buf to the screen
        return flag;
}

/*
Find logical data based off of offset (mailman's algorithm) and loads correct block
From there it'll load the data desired into buf
Return number of bytes put into buf
*/

int read_helper(int fd, char *buf, int nbytes)
{
        MINODE *mip;
        OFT *oftp;
        int count = 0;
        int logic_block, blk, startByte, remain, ino;
        int avil;
        int *ip;
        int size;

        int indirect_blk;
        int indirect_off;
```

```
        int buf2[BLKSIZE];

        char *cq, *cp;
        char readbuf[1024];
        char temp[1024];

        oftp = running->fd[fd];//sets the open file table pointer procces file
descriptor array index
        mip = oftp->inodeptr;//sets the memory inode pointer to the open file table
inode pointer

        avil = mip->INODE.i_size - oftp->offset;//sets the available to size - offset
        cq = buf;

        while(nbytes && avil)//loops while there are more bytes to read

        {
                //mailmans
                logic_block = oftp->offset / BLKSIZE;
                startByte = oftp->offset % BLKSIZE;

                if(logic_block < 12)//direct block
                        blk = mip->INODE.i_block[logic_block];//sets the block to the
inode data block

                else if(logic_block >= 12 && logic_block < 256 + 12)//indirect blocks
                {
                        get_block(mip->dev, mip->INODE.i_block[12], readbuf);

                        ip = (int *)readbuf + logic_block - 12;
                        blk = *ip;
                }
                else//double indirect blocks
                {
                        get_block(mip->dev, mip->INODE.i_block[13], readbuf);//gets
the indirect block

                        //mailmans
                        indirect_blk = (logic_block - 256 - 12) / 256;
                        indirect_off = (logic_block - 256 - 12) % 256;

                        ip = (int *)readbuf + indirect_blk;//get the indirect block
number

                        get_block(mip->dev, *ip, readbuf);//gets the block that the
inode pointer points at

                        ip = (int *)readbuf + indirect_off;
                        blk = *ip;//set the working block to the indirect block number
                }

                get_block(mip->dev, blk, readbuf);//gets the data block that was
previously assigned

                cp = readbuf + startByte;//sets the start point in the file

                remain = BLKSIZE - startByte;//calculates the remaining bytes

                if(avil <= remain && avil <= nbytes)
                {
                        size = avil;
```

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

                memcpy(cq, cp, size);
                oftp->offset += size;
                count += size;
                avil -= size;
                nbytes -= size;
                remain -= size;

        }
        return count;
}
```