```c
#include "type.h"


void init()
{
   int i;

   running = malloc(sizeof(PROC));//malloc's the size of PROC for running

   //sets the proccess id, owner id and cwd for proccess 1 and 2
   proc[0].pid = 1;
   proc[0].uid = 0;
   proc[0].cwd = 0;

   proc[1].pid = 2;
   proc[1].uid = 1;
   proc[1].cwd = 0;

   running = &proc[0];//set the running proc to 0

   for(i = 0; i < 100; i++)
      minode[i].refCount = 0;//set the ref count for the memory inodes to 0

   root = 0;//set the root mip to 0
}

/*
Load superblock, set globals
Load group descriptor block(s), set globals
Get minode of inode 2 (is always root)
Set running->cwd to minode you just loaded
*/

void mount_root(char device_name[64])
{
   char buf[1024];
   dev = open(device_name, O_RDWR); //open the device that user entered

   if(dev < 0)//checks to see whether or not the deice was opened correctly
   {
      printf("Error, could not open %s.\n", device_name);
      exit(0);
   }

   get_block(dev, SUPERBLOCK, buf);
   sp = (SUPER *)buf; //super
   //EXT2 FS
   if(sp->s_magic != 0xEF53) //checks the magic number to see if it is an EXT2
   {
      printf("That was not an EXT2 filesystem.\n");
      exit(1);
   }

   //sets the inode count and block count
   ninodes = sp->s_inodes_count;
   nblocks = sp->s_blocks_count;

   get_block(dev, GDBLOCK, buf);//gets the group descriptor block and puts it in buf
   gp = (GD *)buf; //sets the group block pointer to casted buf

   //save the globals inode map and block map to the ones super_pecified in the group
descriptor
```

```
   imap = gp->bg_inode_bitmap;
   bmap = gp->bg_block_bitmap;

   //sets the inode beginning to the start of the inode table
   inodeBeginBlock = gp->bg_inode_table;
   root = iget(dev, 2); //root is always 2
   proc[0].cwd = root;
   proc[1].cwd = root;

   root->refCount = 3;//sets the roor refCount to 3

   printf("%s has been mounted sucessfully.\n", device_name);
}

//iterate through and print info

int ls_file(MINODE *mip, char *name)
{
  int k;
  int count = 0;
  u16 mode, mask;
  char mydate[32], *s, *cp, ss[32];

  mode = mip->INODE.i_mode;


  if (S_ISDIR(mode))//checks to see if it is a dir then prints
      putchar('d');
  else if (S_ISLNK(mode))//checks link mode
      putchar('l');
  else
      putchar('-');

   //loops through and checks the permission on the inode, and prints accordingly
   mask = 000400;
   for (k=0; k<3; k++){
      if (mode & mask)
         putchar('r');
      else
         putchar('-');
      mask = mask >> 1;

     if (mode & mask)
        putchar('w');
     else
        putchar('-');
        mask = mask >> 1;

     if (mode & mask)
        putchar('x');
     else
        putchar('-');
        mask = mask >> 1;

     }
     printf("   %4d", mip->INODE.i_links_count);
     printf("   %4d", mip->INODE.i_uid);
     printf("  %4d", mip->INODE.i_gid);
     printf("  ");

     s = mydate;
     s = (char *)ctime(&mip->INODE.i_ctime);
```

```
     s = s + 4;
     strncpy(ss, s, 12);
     ss[12] = 0;

     printf("%s", ss);//prints the time
     printf("%8ld",   mip->INODE.i_size);

     printf("    %s", name);

     if (S_ISLNK(mode))
        printf(" -> %s", (char *)mip->INODE.i_block);
     printf("\n");

}

//iterates through to use ls file (names only in dir entries)

int ls_dir(MINODE *mip)
{
  int i;
  char sbuf[BLKSIZE], temp[256];
  DIR *dp;
  char *cp;
  MINODE *dip;

  for (i=0; i<12; i++){ /* search direct blocks only */
     if (mip->INODE.i_block[i] == 0) //block is not occupied
         return 0;

     get_block(mip->dev, mip->INODE.i_block[i], sbuf);//gets the block that the inode
points to
     dp = (DIR *)sbuf;
     cp = sbuf;

     while (cp < sbuf + BLKSIZE){//walk through the dir block
        strncpy(temp, dp->name, dp->name_len);//copies the name and into temp and
sends it to ls file to print the stats
        temp[dp->name_len] = 0;//sets the last index to null
        dip = iget(dev, dp->inode); //gets the inode pointer for the item in the
directory
        ls_file(dip, temp);//sends that inode pointer to ls file to print
        iput(dip);//disposes of inode pointer

        cp += dp->rec_len;//increments to the next item in the block
        dp = (DIR *)cp;//casts the current pointer to a dir pointer
     }
  }
}

/*
ls no argument:
getino of running->cwd
iterate through dir entries and print info of ino's from entries (load inode using
mailman alg)

ls argument:
getino of pathname
iterate through dir entires and print info of ino's from entries
*/

int ls(char *pathname)
{
```

```
  MINODE *mip;
  u16 mode;
  int dev, ino;
  printf("=============================================================\n");

  if (pathname[0] == 0)
    ls_dir(running->cwd);//sends the current directory to ls
  else
  {
    dev = root->dev;
    ino = get_Inode(dev, pathname);//gets the inode of the pathname that was specified
      if (ino==0)//checks to see if the inode was found
      {
         printf("no such path %s\n", pathname);
         return -1;
      }
    mip = iget(dev, ino);//gets the memory inode for the path
    mode = mip->INODE.i_mode;//gets the mode for inode

    if (!S_ISDIR(mode))//checks to see if it is a file
      ls_file(mip, (char *)basename(pathname));
    else
      ls_dir(mip);//if it is a dir then send it to ls dir
    iput(mip);//dispose of the memory inode pointer
  }
  printf("=============================================================\n");
}

/*
gets minode (mailman's algorithm) of pathname and sets running->cwd to that minode
*/

void cd(char *pathname)
{
   int ino = 0;

   MINODE *mip = running->cwd;
   MINODE *newmip = NULL;

   if (!strcmp(pathname, "")) //if nothing is entered for cd then go to root
   {
      running->cwd = root; //set cwd to root
      return;
   }

   if (!strcmp(pathname, "/"))//checks whether or not we want to cd into the root
   {
      running->cwd = root; //set cwd to root
      return;
   }

   ino = get_Inode(mip, pathname); //gets the inode of the path that we want to go

   if(ino == 0)//name doesnt exist in inode table
   {
      printf("The directory %s does not exist.\n", pathname);
      return;
   }

   newmip = iget(dev, ino); //get memory inode pointer

   if(!S_ISDIR(newmip->INODE.i_mode))//checks to see if it is a file
```

```
   {
      printf("%s is not a directory.\n", pathname); //can't cd to a file

      iput(newmip);///dispose of memory inode
      return;
   }

   running->cwd = newmip; //set current working to memory inode pointer
   iput(newmip); //dispose of memery inode

   return;
}

int quit()
{
   int i;

   for(i = 0; i < NMINODE; i++)
   {
      if(minode[i].refCount > 0 && minode[i].dirty == 1)//takes the memory inodes
that have references and arent dirty and makes their ref counts 1 then diposes of them
      {
         minode[i].refCount = 1;
         iput(&minode[i]);
      }
   }
   printf("Exiting...\n");
   exit(0);
}
```