Makiah Heinzmann & Taiya Williams
11442136 11644614

```
#! /bin/bash
mkfs diskimage 1440
gcc main.c
./a.out

#### Program Running ####
checking EXT2 FS ....EXT2 FS OK
bmp=8 imap=9 inode_start = 10
init()
mount_root()
root refCount = 1
creating P0 as running process
root refCount = 2
input command : [ls | cd | pwd | mkdir | quit | creat] pwd
cmd=pwd pathname=
CWD = /
input command : [ls | cd | pwd | mkdir | quit | creat] ls
cmd=ls pathname=
ls
dwrxw-xw-x   3   0   0  Mar 29 18:44:45 2020     1024    .
dwrxw-xw-x   3   0   0  Mar 29 18:44:45 2020     1024    ..
dwrx------   2   0   0  Mar 29 18:44:45 2020    12288    lost+found

input command : [ls | cd | pwd | mkdir | quit | creat] mkdir dir1
cmd=mkdir pathname=dir1
Starting at the local directory!
Parent Path: .
New directory name: dir1
Finding parent inode value...
getino: pathname=.
tokenize .
.
==========================================
getino: i=0 name[0]=.
search for . in MINODE = [3, 2]
  ino   rlen  nlen  name
   2    12     1    .
found . : ino = 2
Found! 2
Mounting parent inode...
Mounted!
Testing for directory and availability...
search for dir1 in MINODE = [3, 2]
  ino   rlen  nlen  name
   2    12     1    .
   2    12     2    ..
  11  1000    10    lost+found
```

```
Found directory, and name available.
allocated ino = 12
NOT DARK: Inode: 12 Block: 47
Stepping to last entry in data block...
Checking record: .
Checking record: ..
Found last entry: lost+found
input command : [ls | cd | pwd | mkdir | quit | creat] ls
cmd=ls pathname=
ls
dwrxw-xw-x   4   0   0  Mar 29 18:44:45 2020     1024    .
dwrxw-xw-x   4   0   0  Mar 29 18:44:45 2020     1024    ..
dwrx------   2   0   0  Mar 29 18:44:45 2020    12288    lost+found
dwrxw-xw-x   2   0   0  Mar 29 18:45:07 2020     1024    dir1

input command : [ls | cd | pwd | mkdir | quit | creat] cd dir1
cmd=cd pathname=dir1
chdir dir1
getino: pathname=dir1
tokenize dir1
dir1
========================================
getino: i=0 name[0]=dir1
search for dir1 in MINODE = [3, 2]
  ino   rlen  nlen  name
   2    12     1    .
   2    12     2    ..
  11    20    10    lost+found
  12   980     4    dir1
found dir1 : ino = 12
input command : [ls | cd | pwd | mkdir | quit | creat] creat file
cmd=creat pathname=file
getino: pathname=.
tokenize .
.
========================================
getino: i=0 name[0]=.
search for . in MINODE = [3, 12]
  ino   rlen  nlen  name
  12    12     1    .
found . : ino = 12
search for file in MINODE = [3, 12]
  ino   rlen  nlen  name
  12    12     1    .
   2  1012     2    ..
allocated ino = 13
SPARKLES ARE NOT DARK: Inode: 13
Stepping to last entry in data block...
Checking record: .
Found last entry: ..
```

```
input command : [ls | cd | pwd | mkdir | quit | creat] ls
cmd=ls pathname=
ls
dwrxw-xw-x   2   0   0  Mar 29 18:45:07 2020     1024    .
dwrxw-xw-x   4   0   0  Mar 29 18:44:45 2020     1024    ..
-wr-w--w--   1   0   0  Mar 29 18:45:21 2020        0    file

input command : [ls | cd | pwd | mkdir | quit | creat] mkdir dir2
cmd=mkdir pathname=dir2
Starting at the local directory!
Parent Path: .
New directory name: dir2
Finding parent inode value...
getino: pathname=.
tokenize .
.
=========================================
getino: i=0 name[0]=.
search for . in MINODE = [3, 12]
  ino   rlen  nlen  name
  12    12     1    .
found . : ino = 12
Found! 12
Mounting parent inode...
Mounted!
Testing for directory and availability...
search for dir2 in MINODE = [3, 12]
  ino   rlen  nlen  name
  12    12     1    .
   2    12     2    ..
  13  1000     4    file
Found directory, and name available.
allocated ino = 14
NOT DARK: Inode: 14 Block: 48
Stepping to last entry in data block...
Checking record: .
Checking record: ..
Found last entry: file
input command : [ls | cd | pwd | mkdir | quit | creat] cd dir2
cmd=cd pathname=dir2
chdir dir2
getino: pathname=dir2
tokenize dir2
dir2
=========================================
getino: i=0 name[0]=dir2
search for dir2 in MINODE = [3, 12]
  ino   rlen  nlen  name
  12    12     1    .
   2    12     2    ..
```

```
  13    12     4    file
  14   988     4    dir2
found dir2 : ino = 14
input command : [ls | cd | pwd | mkdir | quit | creat] ls
cmd=ls pathname=
ls
dwrxw-xw-x   2   0   0  Mar 29 18:45:35 2020     1024    .
dwrxw-xw-x   3   0   0  Mar 29 18:45:07 2020     1024    ..

input command : [ls | cd | pwd | mkdir | quit | creat] pwd
cmd=pwd pathname=
CWD = /dir1/dir2
input command : [ls | cd | pwd | mkdir | quit | creat] creat soupfile
cmd=creat pathname=soupfile
getino: pathname=.
tokenize .
.
========================================
getino: i=0 name[0]=.
search for . in MINODE = [3, 14]
  ino   rlen  nlen  name
  14    12     1    .
found . : ino = 14
search for soupfile in MINODE = [3, 14]
  ino   rlen  nlen  name
  14    12     1    .
  12  1012     2    ..
allocated ino = 15
SPARKLES ARE NOT DARK: Inode: 15
Stepping to last entry in data block...
Checking record: .
Found last entry: ..
input command : [ls | cd | pwd | mkdir | quit | creat] ls
cmd=ls pathname=
ls
dwrxw-xw-x   2   0   0  Mar 29 18:45:35 2020     1024    .
dwrxw-xw-x   3   0   0  Mar 29 18:45:07 2020     1024    ..
-wr-w--w--   1   0   0  Mar 29 18:45:58 2020        0    soupfile

input command : [ls | cd | pwd | mkdir | quit | creat] cd ..
cmd=cd pathname=..
chdir ..
getino: pathname=..
tokenize ..
..
========================================
getino: i=0 name[0]=..
search for .. in MINODE = [3, 14]
  ino   rlen  nlen  name
  14    12     1    .
```

```
  12    12     2    ..
found .. : ino = 12
input command : [ls | cd | pwd | mkdir | quit | creat] pwd
cmd=pwd pathname=
CWD = /dir1
input command : [ls | cd | pwd | mkdir | quit | creat] ls
cmd=ls pathname=
ls
dwrxw-xw-x   3   0   0  Mar 29 18:45:07 2020     1024    .
dwrxw-xw-x   4   0   0  Mar 29 18:44:45 2020     1024    ..
-wr-w--w--   1   0   0  Mar 29 18:45:21 2020        0    file
dwrxw-xw-x   2   0   0  Mar 29 18:45:35 2020     1024    dir2

input command : [ls | cd | pwd | mkdir | quit | creat] ls dir2
cmd=ls pathname=dir2
ls dir2
getino: pathname=dir2
tokenize dir2
dir2
========================================
getino: i=0 name[0]=dir2
search for dir2 in MINODE = [3, 12]
  ino   rlen  nlen  name
  12    12     1    .
   2    12     2    ..
  13    12     4    file
  14   988     4    dir2
found dir2 : ino = 14
dwrxw-xw-x   2   0   0  Mar 29 18:45:35 2020     1024    .
dwrxw-xw-x   3   0   0  Mar 29 18:45:07 2020     1024    ..
-wr-w--w--   1   0   0  Mar 29 18:45:58 2020        0    soupfile

input command : [ls | cd | pwd | mkdir | quit | creat] cd ..
cmd=cd pathname=..
chdir ..
getino: pathname=..
tokenize ..
..
========================================
getino: i=0 name[0]=..
search for .. in MINODE = [3, 12]
  ino   rlen  nlen  name
  12    12     1    .
   2    12     2    ..
found .. : ino = 2
input command : [ls | cd | pwd | mkdir | quit | creat] pwd
cmd=pwd pathname=
CWD = /
input command : [ls | cd | pwd | mkdir | quit | creat] ls
cmd=ls pathname=
```

```
ls
dwrxw-xw-x   4   0   0  Mar 29 18:44:45 2020     1024    .
dwrxw-xw-x   4   0   0  Mar 29 18:44:45 2020     1024    ..
dwrx------   2   0   0  Mar 29 18:44:45 2020    12288    lost+found
dwrxw-xw-x   3   0   0  Mar 29 18:45:07 2020     1024    dir1

input command : [ls | cd | pwd | mkdir | quit | creat] creat filefilefile
cmd=creat pathname=filefilefile
getino: pathname=.
tokenize .
.
=========================================
getino: i=0 name[0]=.
search for . in MINODE = [3, 2]
  ino   rlen  nlen  name
   2    12     1    .
found . : ino = 2
search for filefilefile in MINODE = [3, 2]
  ino   rlen  nlen  name
   2    12     1    .
   2    12     2    ..
  11    20    10    lost+found
  12   980     4    dir1
allocated ino = 16
SPARKLES ARE NOT DARK: Inode: 16
Stepping to last entry in data block...
Checking record: .
Checking record: ..
Checking record: lost+found
Found last entry: dir1
input command : [ls | cd | pwd | mkdir | quit | creat] ls
cmd=ls pathname=
ls
dwrxw-xw-x   4   0   0  Mar 29 18:44:45 2020     1024    .
dwrxw-xw-x   4   0   0  Mar 29 18:44:45 2020     1024    ..
dwrx------   2   0   0  Mar 29 18:44:45 2020    12288    lost+found
dwrxw-xw-x   3   0   0  Mar 29 18:45:07 2020     1024    dir1
-wr-w--w--   1   0   0  Mar 29 18:46:43 2020        0    filefilefile

input command : [ls | cd | pwd | mkdir | quit | creat] ls dir1
cmd=ls pathname=dir1
ls dir1
getino: pathname=dir1
tokenize dir1
dir1
=========================================
getino: i=0 name[0]=dir1
search for dir1 in MINODE = [3, 2]
  ino   rlen  nlen  name
   2    12     1    .
```

```
   2    12     2    ..
  11    20    10    lost+found
  12    12     4    dir1
found dir1 : ino = 12
dwrxw-xw-x   3   0   0  Mar 29 18:45:07 2020     1024    .
dwrxw-xw-x   4   0   0  Mar 29 18:44:45 2020     1024    ..
-wr-w--w--   1   0   0  Mar 29 18:45:21 2020        0    file
dwrxw-xw-x   2   0   0  Mar 29 18:45:35 2020     1024    dir2

input command : [ls | cd | pwd | mkdir | quit | creat] ls dir1/dir2
cmd=ls pathname=dir1/dir2
ls dir1/dir2
getino: pathname=dir1/dir2
tokenize dir1/dir2
dir1  dir2
========================================
getino: i=0 name[0]=dir1
search for dir1 in MINODE = [3, 2]
  ino   rlen  nlen  name
   2    12     1    .
   2    12     2    ..
  11    20    10    lost+found
  12    12     4    dir1
found dir1 : ino = 12
========================================
getino: i=1 name[1]=dir2
search for dir2 in MINODE = [3, 12]
  ino   rlen  nlen  name
  12    12     1    .
   2    12     2    ..
  13    12     4    file
  14   988     4    dir2
found dir2 : ino = 14
dwrxw-xw-x   2   0   0  Mar 29 18:45:35 2020     1024    .
dwrxw-xw-x   3   0   0  Mar 29 18:45:07 2020     1024    ..
-wr-w--w--   1   0   0  Mar 29 18:45:58 2020        0    soupfile

input command : [ls | cd | pwd | mkdir | quit | creat] quit
cmd=quit pathname=
```

//// mkdir.c ////

```
int makeDirectory(MINODE * parentInode, char * childName);
int enter_name(MINODE * parentInode, int childInodeNum, char * childName);

int tryMakeDirectory(char * path) {
    MINODE * start = NULL;
    if (path[0] == '/') {
        printf("Starting at the root!\n");
        start = root;
        dev = root->dev;
```

```
    } else {
        printf("Starting at the local directory!\n");
        start = running->cwd;
        dev = running->cwd->dev;
    }

    char * path2 = strdup(path);

    char * childName = basename(path);
    char * parentPath = dirname(path2);
    if (strcmp(parentPath, "") == 0) parentPath = ".";
    printf("Parent Path: %s\nNew directory name: %s\n", parentPath,
childName);
    printf("Finding parent inode value...\n");
    // getchar();
    int parentInodeNum = getino(parentPath);
    printf("Found! %d\nMounting parent inode...\n", parentInodeNum);
    // getchar();
    MINODE * parentMInode = iget(dev, parentInodeNum);
    printf("Mounted!\nTesting for directory and availability...\n");
    // getchar();
    if (S_ISDIR(parentMInode->INODE.i_mode)) {
        if (search(parentMInode, childName) == 0) {
            printf("Found directory, and name available.\n");
            // getchar();
            makeDirectory(parentMInode, childName);
            parentMInode->refCount++;
            parentMInode->dirty = 1;
            iput(parentMInode);
            free(path2);
            return 1;
        } else {
            printf("%s already exists in %s\n", childName, parentPath);
            free(path2);
            return 0;
        }
    } else {
        printf("%s is not a directory\n", parentPath);
        free(path2);
        return 0;
    }
}

int makeDirectory(MINODE * parentInode, char * childName) {
    // fix pino->dev all over
    MINODE * mounted;
    int allocatedInode = ialloc(parentInode->dev);
    int allocatedBlock = balloc(parentInode->dev);
    printf("NOT DARK: Inode: %d Block: %d\n", allocatedInode,
allocatedBlock);
```

```
    mounted = iget(parentInode->dev, allocatedInode);
    INODE * pInode = &(mounted->INODE);

    pInode->i_mode = 040755;
    pInode->i_uid  = running->uid;
    pInode->i_gid  = running->gid;
    pInode->i_size = BLKSIZE;
    pInode->i_links_count = 2;
    pInode->i_atime = time(0L);
    pInode->i_ctime = pInode->i_atime;
    pInode->i_mtime = pInode->i_atime;

    pInode->i_blocks = 2; //(BLKSIZE/512 > 0) ? BLKSIZE/512 : 1;
    pInode->i_block[0] = allocatedBlock;
    for (int i = 1; i < 15; i++) {
        pInode->i_block[i] = 0;
    }

    mounted->dirty = 1;

    char buffer[BLKSIZE];
    memset(buffer, '\0', BLKSIZE);
    //Child inode information
    char * cp = buffer;
    dp = (DIR *) cp;
    dp->inode = mounted->ino;
    dp->name_len = 1;
    dp->rec_len = 12;
    strncpy(dp->name, ".", 1);

    //Parent inode information
    cp += dp->rec_len;
    dp = (DIR *) cp;
    dp->inode = parentInode->ino;
    dp->name_len = 2;
    dp->rec_len = BLKSIZE - 12;
    strncpy(dp->name, "..", 2);

    put_block(parentInode->dev, allocatedBlock, buffer);
    parentInode->INODE.i_links_count++;
    enter_name(parentInode, allocatedInode, childName);
    iput(mounted);
}
```

//// createfile.c ////

```
int createFile(MINODE * parentInode, char * childName);

int tryCreate(char * path) {
    MINODE * start = NULL;
```

```
    if (path[0] == '/') {
        start = root;
        dev = root->dev;
    } else {
        start = running->cwd;
        dev = running->cwd->dev;
    }

    char * path2 = strdup(path);

    char * childName = basename(path);
    char * parentPath = dirname(path2);

    int parentInodeNum = getino(parentPath);
    MINODE * parentMInode = iget(dev, parentInodeNum);

    if (S_ISDIR(parentMInode->INODE.i_mode)) {
        if (search(parentMInode, childName) == 0) {
            createFile(parentMInode, childName);
            iput(parentMInode);
            free(path2);
            return 1;
        } else {
            printf("%s already exists in %s\n", childName, parentPath);
            free(path2);
            return 1;
        }
    } else {
        printf("%s is not a directory\n", parentPath);
        free(path2);
        return 0;
    }
}

int createFile(MINODE * parentInode, char * childName) {
    MINODE * mounted;
    int allocatedInode = ialloc(parentInode->dev);
    printf("SPARKLES ARE NOT DARK: Inode: %d\n", allocatedInode);

    mounted = iget(parentInode->dev, allocatedInode);
    INODE * pInode = &(mounted->INODE);

    pInode->i_mode = 010644;
    pInode->i_uid  = running->uid;
    pInode->i_gid  = running->gid;
    pInode->i_size = 0;
    pInode->i_links_count = 1;
    pInode->i_atime = time(0L);
    pInode->i_ctime = pInode->i_atime;
    pInode->i_mtime = pInode->i_atime;
```

```
    pInode->i_blocks = 0;
    for (int i = 0; i < 15; i++) {
        pInode->i_block[i] = 0;
    }

    mounted->dirty = 1;
    enter_name(parentInode, allocatedInode, childName);
    iput(mounted);
}
```

//// util.c ////

```
/*********** util.c file ****************/

int get_block(int dev, int blk, char *buf)
{
   lseek(dev, (long)blk*BLKSIZE, 0);
   read(dev, buf, BLKSIZE);
}
int put_block(int dev, int blk, char *buf)
{
   lseek(dev, (long)blk*BLKSIZE, 0);
   write(dev, buf, BLKSIZE);
}

int tokenize(char *pathname)
{
  int i;
  char *s;
  printf("tokenize %s\n", pathname);

  strcpy(gpath, pathname);   // tokens are in global gpath[ ]
  n = 0;

  s = strtok(gpath, "/");
  while(s){
    name[n] = s;
    n++;
    s = strtok(0, "/");
  }

  for (i= 0; i<n; i++)
    printf("%s  ", name[i]);
  printf("\n");
}

// return minode pointer to loaded INODE
MINODE *iget(int dev, int ino)
{
  int i;
```

```
  MINODE *mip;
  char buf[BLKSIZE];
  int blk, offset;
  INODE *ip;

  for (i=0; i<NMINODE; i++){
    mip = &minode[i];
    if (mip->dev == dev && mip->ino == ino){
       mip->refCount++;
       //printf("found [%d %d] as minode[%d] in core\n", dev, ino, i);
       return mip;
    }
  }

  for (i=0; i<NMINODE; i++){
    mip = &minode[i];
    if (mip->refCount == 0){
       //printf("allocating NEW minode[%d] for [%d %d]\n", i, dev, ino);
       mip->refCount = 1;
       mip->dev = dev;
       mip->ino = ino;

       // get INODE of ino into buf[ ]
       blk    = (ino-1)/8 + inode_start;
       offset = (ino-1) % 8;

       //printf("iget: ino=%d blk=%d offset=%d\n", ino, blk, offset);

       get_block(dev, blk, buf);
       ip = (INODE *)buf + offset;
       // copy INODE to mp->INODE
       mip->INODE = *ip;
       return mip;
    }
  }
  printf("PANIC: no more free minodes\n");
  return 0;
}

void iput(MINODE *mip) {
   int i, block, offset;
   char buffer[BLKSIZE];
   INODE *ip;

   mip->refCount--;

   if (mip->refCount > 0)  // minode is still in use
      return;
   if (!mip->dirty)        // INODE has not changed; no need to write back
      return;
```

```
   /* write INODE back to disk */
   /***** NOTE ****************************************
    For mountroot, we never MODIFY any loaded INODE
                  so no need to write it back
   FOR LATER WROK: MUST write INODE back to disk if refCount==0 && DIRTY

   Write YOUR code here to write INODE back to disk
   *****************************************************/
   block  = (mip->ino - 1) / 8 + inode_start;
   offset = (mip->ino - 1) % 8;
   get_block(mip->dev, block, buffer);
   ip = (INODE *)buffer + offset;
   *ip = mip->INODE;
   put_block(mip->dev, block, buffer);
}

int search(MINODE *mip, char *name) {
   char *cp, c, sbuf[BLKSIZE], temp[256];
   DIR *dp;
   INODE *ip;

   printf("search for %s in MINODE = [%d, %d]\n", name, mip->dev, mip->ino);
   ip = &(mip->INODE);

   /*** search for name in mip's data blocks: ASSUME i_block[0] ONLY ***/

   get_block(dev, ip->i_block[0], sbuf);
   dp = (DIR *)sbuf;
   cp = sbuf;
   printf("  ino   rlen  nlen  name\n");

   while (cp - sbuf < BLKSIZE) {
      strncpy(temp, dp->name, dp->name_len);
      temp[dp->name_len] = 0;

      printf("%4d  %4d  %4d    %s\n",
            dp->inode, dp->rec_len, dp->name_len, temp);
      if (strcmp(temp, name) == 0) {
         printf("found %s : ino = %d\n", temp, dp->inode);
         return dp->inode;
      }

      cp += dp->rec_len;
      dp = (DIR *)cp;
   }
   return 0;
}

int getino(char *pathname) {
```

```
   int i, ino, blk, disp;
   char buf[BLKSIZE];
   INODE *ip;
   MINODE *mip;

   printf("getino: pathname=%s\n", pathname);
   if (strcmp(pathname, "/") == 0) return 2;

   // starting mip = root OR CWD
   if (pathname[0] == '/') {
      mip = root;
   } else {
      mip = running->cwd;
   }

   mip->refCount++;         // because we iput(mip) later

   tokenize(pathname);

   for (i=0; i<n; i++){
      printf("=========================================\n");
      printf("getino: i=%d name[%d]=%s\n", i, i, name[i]);

      ino = search(mip, name[i]);

      if (ino==0){
         iput(mip);
         printf("name %s does not exist\n", name[i]);
         return 0;
      }
      iput(mip);                // release current mip
      mip = iget(dev, ino);     // get next mip
   }

   iput(mip);                   // release mip
   return ino;
}

/***************** WE WROTE THIS ***************/
int findmyname(MINODE *parent, u32 myino, char *myname) {
   char buffer[BLKSIZE], * current = buffer;
   DIR * dirPtr = (DIR *) current;

   get_block(parent->dev, parent->INODE.i_block[0], buffer);

   while(myino != dirPtr->inode) {
      current += dirPtr->rec_len;
      dirPtr = (DIR *) current;
   }
   strncpy(myname, dirPtr->name, dirPtr->name_len);
```

```
   myname[dirPtr->name_len] = '\0';
   //printf("\n%s\n", myname); //TODO-rm
}
/*********************************************/

int findino(MINODE *mip, u32 *myino) {
   // myino = ino of . return ino of ..
   char buf[BLKSIZE], *cp;
   DIR *dp;

   get_block(mip->dev, mip->INODE.i_block[0], buf);
   cp = buf;
   dp = (DIR *)buf;
   *myino = dp->inode;
   cp += dp->rec_len;
   dp = (DIR *)cp;
   return dp->inode;
}

/************** WE ADDED BITMAP FUNCTIONS *****************/

int tst_bit(char *buf, int bit) {
   int bytenumber = bit / 8;
   int bitnumber  = bit % 8;
   if (buf[bytenumber] & (1 << bitnumber))
      return 1;
   else
      return 0;
}

int set_bit(char *buf, int bit) {
   int bytenumber = bit / 8;
   int bitnumber  = bit % 8;
   buf[bytenumber] |= (1 << bitnumber);
}

int clr_bit(char *buf, int bit) {
   int bytenumber = bit / 8;
   int bitnumber  = bit % 8;
   buf[bytenumber] &= ~(1 << bitnumber);
}

/************** ALLOCATION FUNCTIONS **********************/


int ialloc(int dev)  // allocate an inode number from inode_bitmap
{
  int  i;
  char buf[BLKSIZE];
```

```
// read inode_bitmap block
  get_block(dev, imap, buf);

  for (i=0; i < ninodes; i++){
    if (tst_bit(buf, i)==0){
        set_bit(buf, i);
        put_block(dev, imap, buf);
        printf("allocated ino = %d\n", i+1); // bits count from 0; ino from 1
        sp->s_free_inodes_count--;
        gp->bg_free_inodes_count--;
        return i+1;
    }
  }
  return 0;
}

int balloc(int dev) {
   u32 total_blocks = sp->s_blocks_count;
   char buf[BLKSIZE];
   get_block(dev, bmap, buf);
   for (int i=0; i < total_blocks; i++) {
      if (tst_bit(buf, i) == 0) {
         set_bit(buf, i);
         put_block(dev, bmap, buf);
         sp->s_free_blocks_count--;
         gp->bg_free_blocks_count--;
         return i+1;
      }
   }
   return 0;
}

int enter_name(MINODE * parentInode, int childInodeNum, char * childName) {
    char buffer[BLKSIZE];
    memset(buffer, '\0', BLKSIZE);
    u16 needed_length = 4*((11+strlen(childName))/4);

    int i = 0;
    for(i = 0; i < 12; i++) {
        if (parentInode->INODE.i_block[i] == 0) {
            printf("No other entries in data block...\n");
            break;
        }

        get_block(parentInode->dev, parentInode->INODE.i_block[i], buffer);
        char * cp = buffer;
        dp = (DIR *) cp;

        printf("Stepping to last entry in data block...\n");
        while(cp + dp->rec_len < buffer + BLKSIZE) {
```

```
            printf("Checking record: %.*s\n", dp->name_len, dp->name);
            cp += dp->rec_len;
            dp = (DIR *) cp;
        }

        printf("Found last entry: %.*s\n", dp->name_len, dp->name);
        u16 new_ideal_length = 4*((11 + dp->name_len)/4);
        u16 remaining_length = dp->rec_len - new_ideal_length;

        if (remaining_length >= needed_length) {
            dp->rec_len = new_ideal_length;
            cp += dp->rec_len;
            dp = (DIR *)cp;
            dp->inode = childInodeNum;
            dp->rec_len = remaining_length;
            dp->name_len = strlen(childName);
            strncpy(dp->name, childName, dp->name_len);
            put_block(parentInode->dev, parentInode->INODE.i_block[i],
buffer);
            return 0;
        }
    }
    printf("Allocating new data block...\n");
    //Reach here, no remaining blocks. Increment number of blocks by 1 and
allocate a enw data block
    int allocatedBlock = balloc(parentInode->dev);
    parentInode->INODE.i_blocks++;
    parentInode->INODE.i_block[i] = allocatedBlock;
    parentInode->INODE.i_size += BLKSIZE;

    dp = (DIR *) buffer;
    dp->inode = childInodeNum;
    dp->name_len = strlen(childName);
    dp->rec_len = BLKSIZE;
    strncpy(dp->name, childName, dp->name_len);
    put_block(parentInode->dev, allocatedBlock, buffer);
    return 0;
}
```

//// cd_ls_pwd.c ////

```
/************* cd_ls_pwd.c file **************/

int chdir(char *pathname) {
    printf("chdir %s\n", pathname);
    // printf("under construction READ textbook HOW TO chdir!!!!\n");
    // READ Chapter 11.7.3 HOW TO chdir
    int inode = getino(pathname);

    MINODE * min = iget(dev, inode);
```

```
    if (S_ISDIR(min->INODE.i_mode)) {
        iput(running->cwd);
        running->cwd = min;
    } else {
        printf("Failure: [ %s ] Not a directory!\n", pathname);
    }
}

int ls_file(MINODE *mip, char *name)
{
  // printf("ls_file: to be done: READ textbook for HOW TO!!!!\n");
  // READ Chapter 11.7.3 HOW TO ls
  char type, perm[10] = "wrxwrxwrx";
  __u16 mode = mip->INODE.i_mode;
  if (S_ISDIR(mode)) type = 'd'; else type = '-';
  for (int i = 0; i < 9; i++) if (!(mode & (1 << i))) perm[8 - i] = '-';
  __u16 links = mip->INODE.i_links_count;
  __u16 owner = mip->INODE.i_uid;
  __u16 group = mip->INODE.i_gid;
  time_t date = mip->INODE.i_mtime;
  __u32 size = mip->INODE.i_size;
  printf("%c%s% 4d% 4d% 4d  %.20s % 8d    %s\n",
    type, perm, links, owner, group, ctime(&date)+4, size, name);
}

int ls_dir(MINODE *mip)
{
  // printf("ls_dir: list CWD's file names; YOU do it for ls -l\n");

  char buf[BLKSIZE], temp[256];
  DIR *dp;
  char *cp;

  // Assume DIR has only one data block i_block[0]
  get_block(dev, mip->INODE.i_block[0], buf);
  dp = (DIR *)buf;
  cp = buf;

  while (cp < buf + BLKSIZE){
     strncpy(temp, dp->name, dp->name_len);
     temp[dp->name_len] = 0;

     // printf("[%d %s]  ", dp->inode, temp); // print [inode# name]
     ls_file(iget(dev, dp->inode), temp);

     cp += dp->rec_len;
     dp = (DIR *)cp;
  }
  printf("\n");
}
```

```
int ls(char *pathname)
{
  printf("ls %s\n", pathname);
  //printf("ls CWD only! YOU do it for ANY pathname\n");
  if (pathname[0] != '\0') {
    MINODE * min = iget(dev, getino(pathname));
    if (S_ISDIR(min->INODE.i_mode)) {
      ls_dir(min);
      iput(min);
    } else
      printf("Failure: [ %s ] Not a directory!\n", pathname);
  } else
    ls_dir(running->cwd);
}

/*************** Algorithm of pwd ***************
 *  rpwd( MINODE *wd){
 *      (1). if (wd == root) return;
 *      (2). from wd->INODE.i_block[0], get my_ino and parent_ino
 *      (3). pip = iget(dev, parent_ino);
 *      (4). from pip->INODE.i_block[]: get my_name string by my_ino as LOCAL
 *      (5). rpwd(pip);
 *      // recursive call rpwd( pip) with parent minode
 */

void recursivePWD(MINODE *curNode) {
    if (curNode != root) {
        int myINode = 0;
        int parentINode = findino(curNode, &myINode);
        MINODE * parent = iget(dev, parentINode);
        char curName[256];
        findmyname(parent, myINode, curName);
        recursivePWD(parent);
        iput(parent);
        printf("/%s", curName);
    }
}

void pwd(MINODE *wd){
    printf("CWD = ");
    if (wd == root) printf("/");
    recursivePWD(wd);
    printf("\n");
}
```

//// type.h ////

```
/*************** type.h file ************************/
typedef unsigned char  u8;
typedef unsigned short u16;
```

```
typedef unsigned int   u32;

typedef struct ext2_super_block SUPER;
typedef struct ext2_group_desc  GD;
typedef struct ext2_inode       INODE;
typedef struct ext2_dir_entry_2 DIR;

SUPER *sp;
GD    *gp;
INODE *ip;
DIR   *dp;

#define FREE        0
#define READY       1

#define BLKSIZE  1024
#define NMINODE   128
#define NFD        16
#define NPROC       2

typedef struct minode {
  INODE INODE;
  int dev, ino;
  int refCount;
  int dirty;
  int mounted;
  struct mntable *mptr;
} MINODE;

typedef struct oft {
  int  mode;
  int  refCount;
  MINODE *mptr;
  int  offset;
} OFT;

typedef struct proc {
  struct proc *next;
  int          pid;
  int          status;
  int          uid, gid;
  MINODE      *cwd;
  OFT         *fd[NFD];
} PROC;
```

//// globals.h ////

```
// global variables
MINODE minode[NMINODE];
MINODE *root;
```

```
PROC   proc[NPROC], *running;

char gpath[128]; // global for tokenized components
char *name[32];  // assume at most 32 components in pathname
int   n;         // number of component strings

int fd, dev;
int nblocks, ninodes, bmap, imap, inode_start; // disk parameters
```

//// main.c ////

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <ext2fs/ext2_fs.h>
#include <string.h>
#include <libgen.h>
#include <sys/stat.h>
#include <time.h>
#include <stddef.h>

#include "type.h"
#include "globals.h"
#include "util.c"
#include "cd_ls_pwd.c"
#include "mkdir.c"
#include "createfile.c"

int init()
{
  int i, j;
  MINODE *mip;
  PROC   *p;

  printf("init()\n");

  for (i=0; i<NMINODE; i++){
    mip = &minode[i];
    mip->dev = mip->ino = 0;
    mip->refCount = 0;
    mip->mounted = 0;
    mip->mptr = 0;
  }
  for (i=0; i<NPROC; i++){
    p = &proc[i];
    p->pid = i;
    p->uid = p->gid = 0;
    p->cwd = 0;
    p->status = FREE;
    for (j=0; j<NFD; j++)
```

```
      p->fd[j] = 0;
  }
}

// load root INODE and set root pointer to it
int mount_root()
{
  printf("mount_root()\n");
  root = iget(dev, 2);
}

int quit()
{
  int i;
  MINODE *mip;
  for (i=0; i<NMINODE; i++){
    mip = &minode[i];
    while (mip->refCount > 0)
      iput(mip);
  }
  exit(0);
}

char *disk = "diskimage";
int main(int argc, char *argv[ ])
{
  int ino;
  char spbuf[BLKSIZE], gpbuf[BLKSIZE];
  char line[128], cmd[32], pathname[128];

  printf("checking EXT2 FS ....");
  if ((fd = open(disk, O_RDWR)) < 0){
    printf("open %s failed\n", disk);
    exit(1);
  }
  dev = fd;    // fd is the global dev

  /********** read super block  ****************/
  get_block(dev, 1, spbuf);
  sp = (SUPER *)spbuf;

  /* verify it's an ext2 file system ***********/
  if (sp->s_magic != 0xEF53){
      printf("magic = %x is not an ext2 filesystem\n", sp->s_magic);
      exit(1);
  }
  printf("EXT2 FS OK\n");
  ninodes = sp->s_inodes_count;
  nblocks = sp->s_blocks_count;
```

```
get_block(dev, 2, gpbuf);
gp = (GD *)gpbuf;

bmap = gp->bg_block_bitmap;
imap = gp->bg_inode_bitmap;
inode_start = gp->bg_inode_table;
printf("bmp=%d imap=%d inode_start = %d\n", bmap, imap, inode_start);

init();
mount_root();
printf("root refCount = %d\n", root->refCount);

printf("creating P0 as running process\n");
running = &proc[0];
running->status = READY;
running->cwd = iget(dev, 2);
printf("root refCount = %d\n", root->refCount);

// WRTIE code here to create P1 as a USER process

while(1){
  printf("input command : [ls | cd | pwd | mkdir | quit | creat] ");
  fgets(line, 128, stdin);
  line[strlen(line)-1] = '\0';

  if (line[0] == '\0')
    continue;
  pathname[0] = '\0';

  sscanf(line, "%s %s", cmd, pathname);
  printf("cmd=%s pathname=%s\n", cmd, pathname);

  if (strcmp(cmd, "ls") == 0)
    ls(pathname);
  else if (strcmp(cmd, "cd") == 0)
    chdir(pathname);
  else if (strcmp(cmd, "pwd") == 0)
    pwd(running->cwd);
  else if (strcmp(cmd, "quit") == 0)
    quit();
  else if (strcmp(cmd, "mkdir") == 0) {
    if (strcmp(pathname, "") != 0) {
      if (tryMakeDirectory(pathname) == 0) {
        printf("mkdir %s failed\n", pathname);
      }
    } else {
      printf("Error: No path specified!\n");
    }
  }
  else if (strcmp(cmd, "creat") == 0) {
```

```
      if (tryCreate(pathname) == 0) {
        printf("creat %s failed\n", pathname);
      }
    }
  }
}
```