George Suarez
David Cruz
CSE 460

Lab 10 - Exploring XV6 File System

1. Examining the header file *fs.h* and the program *fs.c* that implements the file system.

fs.h

```
// On-disk file system format.
// Both the kernel and user programs use this header file.


#define ROOTINO 1  // root i-number
#define BSIZE 512  // block size

// Disk layout:
// [ boot block | super block | log | inode blocks |
//                                          free bit map | data blocks]
//
// mkfs computes the super block and builds an initial file system. The
// super block describes the disk layout:
struct superblock {
  uint size;         // Size of file system image (blocks)
  uint nblocks;      // Number of data blocks
  uint ninodes;      // Number of inodes.
  uint nlog;         // Number of log blocks
  uint logstart;     // Block number of first log block
  uint inodestart;   // Block number of first inode block
  uint bmapstart;    // Block number of first free map block
};

#define NDIRECT 12
#define NINDIRECT (BSIZE / sizeof(uint))
#define MAXFILE (NDIRECT + NINDIRECT)

// On-disk inode structure
struct dinode {
  short type;           // File type
  short major;          // Major device number (T_DEV only)
  short minor;          // Minor device number (T_DEV only)
  short nlink;          // Number of links to inode in file system
  uint size;            // Size of file (bytes)
  uint addrs[NDIRECT+1];   // Data block addresses
};

// Inodes per block.
#define IPB           (BSIZE / sizeof(struct dinode))

// Block containing inode i
#define IBLOCK(i, sb)     ((i) / IPB + sb.inodestart)

// Bitmap bits per block
#define BPB           (BSIZE*8)

// Block of free map containing bit for block b
#define BBLOCK(b, sb) (b/BPB + sb.bmapstart)

// Directory is a file containing a sequence of dirent structures.
#define DIRSIZ 14

struct dirent {
  ushort inum;
  char name[DIRSIZ];
};
```

fs.c

```
// File system implementation.  Five layers:
//   + Blocks: allocator for raw disk blocks.
//   + Log: crash recovery for multi-step updates.
//   + Files: inode allocator, reading, writing, metadata.
//   + Directories: inode with special contents (list of other inodes!)
//   + Names: paths like /usr/rtm/xv6/fs.c for convenient naming.
//
// This file contains the low-level file system manipulation
// routines.  The (higher-level) system call implementations
// are in sysfile.c.

#include "types.h"
#include "defs.h"
#include "param.h"
#include "stat.h"
#include "mmu.h"
#include "proc.h"
#include "spinlock.h"
#include "sleeplock.h"
#include "fs.h"
#include "buf.h"
#include "file.h"

#define min(a, b) ((a) < (b) ? (a) : (b))
static void itrunc(struct inode*);
// there should be one superblock per disk device, but we run with
// only one device
struct superblock sb;

// Read the super block.
void
readsb(int dev, struct superblock *sb)
{
  struct buf *bp;

  bp = bread(dev, 1);
  memmove(sb, bp->data, sizeof(*sb));
  brelse(bp);
}

// Zero a block.
static void
bzero(int dev, int bno)
{
  struct buf *bp;

  bp = bread(dev, bno);
  memset(bp->data, 0, BSIZE);
  log_write(bp);
  brelse(bp);
}
```

2. Add to a system call named pfs() in fs.c that prints out information of the file system

   a. Adding the name to *syscall.h*

```
#define SYS_mkdir  20
#define SYS_close  21
#define SYS_cps    22
#define SYS_chpr   23
#define SYS_pfs    24
```

   b. Adding the function prototype to *defs.h*

```
int             namecmp(const char*, const char*);
struct inode*   namei(char*);
struct inode*   nameiparent(char*, char*);
int             readi(struct inode*, char*, uint, uint);
void            stati(struct inode*, struct stat*);
int             writei(struct inode*, char*, uint, uint);
int             pfs(void);
```

c. Added the function prototype to *users.h*

```
int sleep(int);
int uptime(void);
int cps ( void );
int chpr (int pid, int priority);
int pfs(void);
```

d. Added the function call to *syscall.c*

```
int
sys_cps ( void )
{
  return cps ();
}

int
sys_chpr ( void )
{
  int pid, pr;
  if (argint(0, &pid) < 0)
    return -1;
  if (argint(1, &pr) < 0)
    return -1;

  return chpr ( pid, pr );
}

int sys_pfs(void)
{
  return pfs();
}
```

e. Added the call to *usys.S*

```
SYSCALL(sbrk)
SYSCALL(sleep)
SYSCALL(uptime)
SYSCALL(cps)
SYSCALL(chpr)
SYSCALL(pfs)
```

f. Added the call to *syscall.c*

```
extern int sys_chpr(void);
extern int sys_pfs(void);
[SYS_cps]     sys_cps,
[SYS_chpr]    sys_chpr,
[SYS_pfs]     sys_pfs,
```

g. Add *pfs()* to *fs.c*

```
int
pfs()
{
  cprintf("sb: size  nblocks  ninodes nlog logstart inodestart bmap-start Inodes-per-block Bitmap-bits-per-block\n");
  cprintf("   %d\t    %d\t     %d     %d        %d \t %d \t  %d   \t %d   \t  %d\n", sb.size, sb.nblocks,
          sb.ninodes, sb.nlog, sb.logstart, sb.inodestart, sb.bmapstart, IPB,  BPB);

  return 0;
}
```

3. Write another program, say *printfs.c* and make other appropriate changes as you did in previous two labs to call **pfs**().

   a. Implementing *printfs.c*

```
#include "types.h"
#include "stat.h"
#include "user.h"
#include "fcntl.h"
#include "fs.h"

int main(int argc, char *argv[])
{
   pfs();
   exit();
}
```

   b. Adding *printfs.c* to the Makefile

```
UPROGS=\
        _cat\
        _echo\
        _forktest\
        _grep\
        _init\
        _kill\
        _ln\
        _ls\
        _mkdir\
        _rm\
        _sh\
        _stressfs\
        _usertests\
        _wc\
        _nice\
        _foo\
        _cp\
        _ps\
        _printfs\
        _zombie\


EXTRA=\
       mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c kill.c\
       ln.c ls.c mkdir.c rm.c stressfs.c usertests.c wc.c printfs.c nice.c cp.c ps.c foo.c zombie.c\
       printf.c umalloc.c\
       README dot-bochsrc *.pl toc.* runoff runoff1 runoff.list\
       .gdbinit.tmpl gdbutil\
```

c. Output

```
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ printfs
sb: size  nblocks  ninodes nlog logstart inodestart bmap-start Inodes-per-block Bitmap-bits-per-block
   1000     941      200     30        2         32       58     8        4096
$
```

4. Add to your program and function call that prints out the free blocks and allocated blocks of the system as well as the number of free blocks and the number of allocated blocks in the system. The output of your program could look like the following:

a. Modified *pfs.c* function in *fs.c*

```
int
pfs()
{
  int b, bi, m;
  struct buf *bp;
  int countt = 0, countf = 0, counta = 0, bn = 0, nb = 0;
  bp = 0;
  cprintf("\nFree blocks: \n");
  for(b = 0; b < sb.size; b += BPB){
   nb++;
   bp = bread(1, BBLOCK(b, sb));
   for(bi = 0; bi < BPB && b + bi < sb.size; bi++){
    m = 1 << (bi % 8);
    ++countt;
    if((bp->data[bi/8] & m) == 0) { //checks to see if block is free
      if((countf % 16) == 0) cprintf("\n");
         cprintf( "%d ", bn );
         ++countf;
     }
     bn++;
     }
     brelse(bp);
   }
   cprintf("\nTotal free blocks = %d", countf);
   cprintf("\nAllocated blocks:\n");
   bn = 0;

   for(b = 0; b < sb.size; b += BPB) {
     bp = bread(1, BBLOCK(b, sb));
     for (bi = 0; bi < BPB && b + bi < sb.size; bi++) {
        m = 1 << (bi % 8);
        if((bp->data[bi/8] & m) != 0) { //checks to see if block isallocated
         ++counta;
        if((counta % 16) == 0) {
           cprintf("\n");
       }
       cprintf("%d", bn);
     }
     bn++;
   }
   brelse(bp);
 }
 cprintf("\nTotal allocated blocks = %d\n", counta);
 cprintf("\nTall blocks = %d\n", countt);
 return 0;
}
"fs.c" 718L, 16925C written
```

b. Output

```
xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32
init: starting sh
$ printfs

Free blocks:

712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727
728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743
744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759
760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775
776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791
792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807
808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823
824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839
840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855
856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871
872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887
888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903
904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919
920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935
936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951
952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967
968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983
984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999
Total free blocks = 288

Allocated blocks:
01234567891011121314
15161718192021222324252627282930
31323334353637383940414243444546
47484950515253545556575859606162
63646566676869707172737475767778
79808182838485868788899091929394
95969798991001011021031041051061071081091110
11111211311411511611711811912012112212312412512 6
12712812913013113213313413513613713813914014114 2
14314414514614714814915015115215315415515615715 8
15916016116216316416516616716816917017117217317 4
17517617717817918018118218318418518618718818919 0
19119219319419519619719819920020120220320420520 6
20720820921021121221321421521621721821922022122 2
22322422522622722822923023123223323423523623723 8
23924024124224324424524624724824925025125225325 4
25525625725825926026126226326426526626726826927 0
27127227327427527627727827928028128228328428528 6
28728828929029129229329429529629729829930030130 2
30330430530630730830931031131231331431531631731 8
31932032132232332432532632732832933033133233333 4
33533633733833934034134234334434534634734834935 0
35135235335435535635735835936036136236336436536 6
36736836937037137237337437537637737837938038138 2
38338438538638738838939039139239339439539639739 8
39940040140240340440540640740840941041141241341 4
41541641741841942042142242342442542642742842943 0
43143243343443543643743843944044144244344444544 6
44744844945045145245345445545645745845946046146 2
46346446546646746846947047147247347447547647747 8
47948048148248348448548648748848949049149249349 4
49549649749849950050150250350450550650750850951 0
51151251351451551651751851952052152252352452552 6
52752852953053153253353453553653753853954054154 2
54354454554654754854955055155255355455555655755 8
55956056156256356456556656756856957057157257357 4
57557657757857958058158258358458558658758858959 0
59159259359459559659759859960060160260360460560 6
60760860961061161261361461561661761861962062162 2
62362462562662762862963063163263363463563663763 8
63964064164264364464564664764864965065165265365 4
65565665765865966066166266366466566666766866967 0
67167267367467567667767867968068168268368468568 6
68768868969069169269369469569669769869970070170 2
70370470570670770870971071
Total allocated blocks = 712

Tall blocks = 1000
```