Lab 10: Exploring XV6 File System

Jorge Valtierra
&
Luis Garcia

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

A file system is for organizing and storing data in a systematic way. File systems typically support sharing of data among users and applications, as well as persistence so that data is still available after a reboot. The xv6 file system provides Unix-like files, directories, and pathnames.
Done.

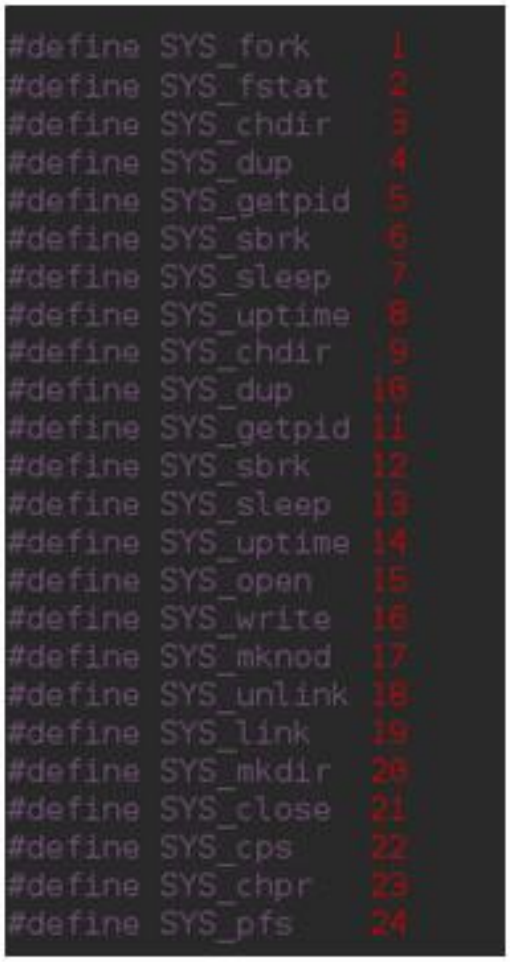
```
-------------
pathnames
-------------
directories
-------------
inodes
-------------
blocks
-------------
```

2 . Add to a system call named pfs() in *fs.c* that prints out information of the file system. For example, the function can be like:

Execute:

```
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;
}
```



```
#define SYS_fork    1
#define SYS_fstat   2
#define SYS_chdir   3
#define SYS_dup     4
#define SYS_getpid  5
#define SYS_sbrk    6
#define SYS_sleep   7
#define SYS_uptime  8
#define SYS_chdir   9
#define SYS_dup     10
#define SYS_getpid  11
#define SYS_sbrk    12
#define SYS_sleep   13
#define SYS_uptime  14
#define SYS_open    15
#define SYS_write   16
#define SYS_mknod   17
#define SYS_unlink  18
#define SYS_link    19
#define SYS_mkdir   20
#define SYS_close   21
#define SYS_cps     22
#define SYS_chpr    23
#define SYS_pfs     24
```

```
// system calls
int fork(void);
int exit(void) __attribute__((noreturn));
int wait(void);
int pipe(int*);
int write(int, void*, int);
int read(int, void*, int);
int close(int);
int kill(int);
int exec(char*, char**);
int open(char*, int);
int mknod(char*, short, short);
int unlink(char*);
int fstat(int fd, struct stat*);
int link(char*, char*);
int mkdir(char*);
int chdir(char*);
int dup(int);
int getpid(void);
char* sbrk(int);
int sleep(int);
int uptime(void);
int cps(void);
int chpr(int, int);
int fps(void);
```

3. Write another program, say *printfs.c* and make other appropriate changes as you did in previous two labs to call pfs(). When you execute *printfs* in xv6, you should see an output similar to the following:

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

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

  exit();
}
```

```
$ 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
$
```

```
$ 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 it 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:

```
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){  // Is block free?
        if ( (countf % 16) == 0 ) cprintf("\n");
        cprintf("%d  ", bn );
        ++countf;
      }
      bn++;
    }
    brelse(bp);
  }
  cprintf("\nTotal free blocks = %d", countf );

  cprintf("\nAlocated blocks:\n");
  bn = 0;
  ......
}
```

Execute:

```
Free blocks:
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
........
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 = 272

Alocated blocks:
0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15
16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31
........
704  705  706  707  708  709  710  711  712  713  714  715  716  717  718
```

```
719
720  721  722  723  724  725  726  727
Total allocated blocks = 728
```

```
$ printfs

Free blocks:

562  563  564  565  566  567  568  569  570  571  572  573  574  575  576  577
578  579  580  581  582  583  584  585  586  587  588  589  590  591  592  593
594  595  596  597  598  599  600  601  602  603  604  605  606  607  608  609
610  611  612  613  614  615  616  617  618  619  620  621  622  623  624  625
626  627  628  629  630  631  632  633  634  635  636  637  638  639  640  641
642  643  644  645  646  647  648  649  650  651  652  653  654  655  656  657
658  659  660  661  662  663  664  665  666  667  668  669  670  671  672  673
674  675  676  677  678  679  680  681  682  683  684  685  686  687  688  689
690  691  692  693  694  695  696  697  698  699  700  701  702  703  704  705
706  707  708  709  710  711  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 = 438
```

Summary:

We have successfully finish the following questions that were assigned to us. The following lab consisted of implements the file system. As well as, creating a program called printfs.c that will print the following figure. Also, a program that will print out the free blocks and allocate blocks of the system. Finally, we added the system that prints out information of the file system.

Score 20/20