BIGGER FILES FOR XV6 作业报告

软工2班 3017218092 侯雨茜

一、准备工作

xv6系统安装成功后，可以直接使用下面这条命令在终端编译并运行

```
make qemu-nox
```

1.1 修改Makefile文件

1. 将 `CPUS := 2` 修改为 `CPUS := 1`
2. 在 `QEMUOPTS` 前，添加 `QEMUEXTRA = -snapshot`

修改后的Makefile文件代码片段如下：

```
ifndef CPUS
#CPUS := 2
CPUS := 1
endif
QEMUEXTRA = -snapshot
QEMUOPTS = -drive file=fs.img,index=1,media=disk,format=raw -drive
```

1.2 修改param.h文件

将 `#define FSSIZE 1000 // size of file system in blocks`

修改为 `#define FSSIZE // size of file system in blocks`

修改后的param.h文件代码片段如下：

```
#define NBUF         (MAXOPBLOCKS*3)  // size of disk block cache
//#define FSSIZE       1000  // size of file system in blocks
#define FSSIZE       20000  // size of file system in blocks
```

1.3 添加big.c文件

1. 将big.c文件放入xv6-public文件目录下

   big.c文件如下：

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

int
main()
{
  char buf[512];
  int fd, i, sectors;

  fd = open("big.file", O_CREATE | O_WRONLY);
  if(fd < 0){
    printf(2, "big: cannot open big.file for writing\n");
    exit();
  }

  sectors = 0;
  while(1){
    *(int*)buf = sectors;
    int cc = write(fd, buf, sizeof(buf));
    if(cc <= 0)
      break;
    sectors++;
  if (sectors % 100 == 0)
    printf(2, ".");
  }

  printf(1, "\nwrote %d sectors\n", sectors);

  close(fd);
  fd = open("big.file", O_RDONLY);
  if(fd < 0){
    printf(2, "big: cannot re-open big.file for reading\n");
    exit();
  }
```

```
  for(i = 0; i < sectors; i++){
    int cc = read(fd, buf, sizeof(buf));
    if(cc <= 0){
      printf(2, "big: read error at sector %d\n", i);
      exit();
    }
    if(*(int*)buf != i){
      printf(2, "big: read the wrong data (%d) for sector %d\n",
             *(int*)buf, i);
      exit();
    }
  }

  printf(1, "done; ok\n");

  exit();
}
```

2. 在Makefile文件中的UPROGS列表中添加big.c文件

   修改后的Makefile文件如下：

```
OBJS = \
  bio.o\
  console.o\
  exec.o\
  file.o\
  fs.o\
  ide.o\
  ioapic.o\
  kalloc.o\
  kbd.o\
  lapic.o\
  log.o\
  main.o\
  mp.o\
  picirq.o\
  pipe.o\
  proc.o\
  sleeplock.o\
  spinlock.o\
  string.o\
  swtch.o\
  syscall.o\
  sysfile.o\
  sysproc.o\
  trapasm.o\
  trap.o\
  uart.o\
```

```
  vectors.o\
  vm.o\

# Cross-compiling (e.g., on Mac OS X)
# TOOLPREFIX = i386-jos-elf

# Using native tools (e.g., on X86 Linux)
#TOOLPREFIX =

# Try to infer the correct TOOLPREFIX if not set
ifndef TOOLPREFIX
TOOLPREFIX := $(shell if i386-jos-elf-objdump -i 2>&1 | grep '^elf32-i386$$' >/dev/null 2>&1; \
  then echo 'i386-jos-elf-'; \
  elif objdump -i 2>&1 | grep 'elf32-i386' >/dev/null 2>&1; \
  then echo ''; \
  else echo "***" 1>&2; \
  echo "*** Error: Couldn't find an i386-*-elf version of GCC/binutils." 1>&2; \
  echo "*** Is the directory with i386-jos-elf-gcc in your PATH?" 1>&2; \
  echo "*** If your i386-*-elf toolchain is installed with a command" 1>&2; \
  echo "*** prefix other than 'i386-jos-elf-', set your TOOLPREFIX" 1>&2; \
  echo "*** environment variable to that prefix and run 'make' again." 1>&2; \
  echo "*** To turn off this error, run 'gmake TOOLPREFIX= ...'." 1>&2; \
  echo "***" 1>&2; exit 1; fi)
endif

# If the makefile can't find QEMU, specify its path here
 QEMU = qemu-system-i386

# Try to infer the correct QEMU
ifndef QEMU
QEMU = $(shell if which qemu > /dev/null; \
  then echo qemu; exit; \
  elif which qemu-system-i386 > /dev/null; \
  then echo qemu-system-i386; exit; \
  elif which qemu-system-x86_64 > /dev/null; \
  then echo qemu-system-x86_64; exit; \
  else \
  qemu=/Applications/Q.app/Contents/MacOS/i386-softmmu.app/Contents/MacOS/i386-softmmu; \
  if test -x $$qemu; then echo $$qemu; exit; fi; fi; \
  echo "***" 1>&2; \
```

```
  echo "*** Error: Couldn't find a working QEMU executable." 1>&2; \
  echo "*** Is the directory containing the qemu binary in your PATH" 1>&2; \
  echo "*** or have you tried setting the QEMU variable in Makefile?" 1>&2; \
  echo "***" 1>&2; exit 1)
endif

CC = $(TOOLPREFIX)gcc
AS = $(TOOLPREFIX)gas
LD = $(TOOLPREFIX)ld
OBJCOPY = $(TOOLPREFIX)objcopy
OBJDUMP = $(TOOLPREFIX)objdump
CFLAGS = -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer
CFLAGS += $(shell $(CC) -fno-stack-protector -E -x c /dev/null >/dev/null 2>&1 && echo -fno-stack-protector)
ASFLAGS = -m32 -gdwarf-2 -Wa,-divide
# FreeBSD ld wants ``elf_i386_fbsd''
LDFLAGS += -m $(shell $(LD) -V | grep elf_i386 2>/dev/null | head -n 1)

# Disable PIE when possible (for Ubuntu 16.10 toolchain)
ifneq ($(shell $(CC) -dumpspecs 2>/dev/null | grep -e '[^f]no-pie'),)
CFLAGS += -fno-pie -no-pie
endif
ifneq ($(shell $(CC) -dumpspecs 2>/dev/null | grep -e '[^f]nopie'),)
CFLAGS += -fno-pie -nopie
endif

xv6.img: bootblock kernel
  dd if=/dev/zero of=xv6.img count=10000
  dd if=bootblock of=xv6.img conv=notrunc
  dd if=kernel of=xv6.img seek=1 conv=notrunc

xv6memfs.img: bootblock kernelmemfs
  dd if=/dev/zero of=xv6memfs.img count=10000
  dd if=bootblock of=xv6memfs.img conv=notrunc
  dd if=kernelmemfs of=xv6memfs.img seek=1 conv=notrunc

bootblock: bootasm.S bootmain.c
  $(CC) $(CFLAGS) -fno-pic -O -nostdinc -I. -c bootmain.c
  $(CC) $(CFLAGS) -fno-pic -nostdinc -I. -c bootasm.S
  $(LD) $(LDFLAGS) -N -e start -Ttext 0x7C00 -o bootblock.o bootasm.o bootmain.o
  $(OBJDUMP) -S bootblock.o > bootblock.asm
```

```
  $(OBJCOPY) -S -O binary -j .text bootblock.o bootblock
  ./sign.pl bootblock

entryother: entryother.S
  $(CC) $(CFLAGS) -fno-pic -nostdinc -I. -c entryother.S
  $(LD) $(LDFLAGS) -N -e start -Ttext 0x7000 -o bootblockother.o entryother.o
  $(OBJCOPY) -S -O binary -j .text bootblockother.o entryother
  $(OBJDUMP) -S bootblockother.o > entryother.asm

initcode: initcode.S
  $(CC) $(CFLAGS) -nostdinc -I. -c initcode.S
  $(LD) $(LDFLAGS) -N -e start -Ttext 0 -o initcode.out initcode.o
  $(OBJCOPY) -S -O binary initcode.out initcode
  $(OBJDUMP) -S initcode.o > initcode.asm

kernel: $(OBJS) entry.o entryother initcode kernel.ld
  $(LD) $(LDFLAGS) -T kernel.ld -o kernel entry.o $(OBJS) -b binary initcode entryother
  $(OBJDUMP) -S kernel > kernel.asm
  $(OBJDUMP) -t kernel | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$$/d' > kernel.sym

# kernelmemfs is a copy of kernel that maintains the
# disk image in memory instead of writing to a disk.
# This is not so useful for testing persistent storage or
# exploring disk buffering implementations, but it is
# great for testing the kernel on real hardware without
# needing a scratch disk.
MEMFSOBJS = $(filter-out ide.o,$(OBJS)) memide.o
kernelmemfs: $(MEMFSOBJS) entry.o entryother initcode kernel.ld fs.img
  $(LD) $(LDFLAGS) -T kernel.ld -o kernelmemfs entry.o  $(MEMFSOBJS) -b binary initcode entryother fs.img
  $(OBJDUMP) -S kernelmemfs > kernelmemfs.asm
  $(OBJDUMP) -t kernelmemfs | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$$/d' > kernelmemfs.sym

tags: $(OBJS) entryother.S _init
  etags *.S *.c

vectors.S: vectors.pl
  ./vectors.pl > vectors.S

ULIB = ulib.o usys.o printf.o umalloc.o

_%: %.o $(ULIB)
  $(LD) $(LDFLAGS) -N -e main -Ttext 0 -o $@ $^
```

```
150   $(OBJDUMP) -S $@ > $*.asm
151   $(OBJDUMP) -t $@ | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$$/d'
> $*.sym
152
153 _forktest: forktest.o $(ULIB)
154   # forktest has less library code linked in - needs to be small
155   # in order to be able to max out the proc table.
156   $(LD) $(LDFLAGS) -N -e main -Ttext 0 -o _forktest forktest.o
ulib.o usys.o
157   $(OBJDUMP) -S _forktest > forktest.asm
158
159 mkfs: mkfs.c fs.h
160   gcc -Werror -Wall -o mkfs mkfs.c
161
162 # Prevent deletion of intermediate files, e.g. cat.o, after
first build, so
163 # that disk image changes after first build are persistent until
clean.  More
164 # details:
165 # http://www.gnu.org/software/make/manual/html_node/Chained-
Rules.html
166 .PRECIOUS: %.o
167
168 UPROGS=\
169   _cat\
170   _echo\
171   _forktest\
172   _grep\
173   _init\
174   _kill\
175   _ln\
176   _ls\
177   _mkdir\
178   _rm\
179   _sh\
180   _stressfs\
181   _usertests\
182   _wc\
183   _zombie\
184   _big\
185
186 fs.img: mkfs README $(UPROGS)
187   ./mkfs fs.img README $(UPROGS)
188
189 -include *.d
190
191 clean:
192   rm -f *.tex *.dvi *.idx *.aux *.log *.ind *.ilg \
193   *.o *.d *.asm *.sym vectors.S bootblock entryother \
```

```
  initcode initcode.out kernel xv6.img fs.img kernelmemfs \
  xv6memfs.img mkfs .gdbinit \
  $(UPROGS)

# make a printout
FILES = $(shell grep -v '^\#' runoff.list)
PRINT = runoff.list runoff.spec README toc.hdr toc.ftr $(FILES)

xv6.pdf: $(PRINT)
  ./runoff
  ls -l xv6.pdf

print: xv6.pdf

# run in emulators

bochs : fs.img xv6.img
  if [ ! -e .bochsrc ]; then ln -s dot-bochsrc .bochsrc; fi
  bochs -q

# try to generate a unique GDB port
GDBPORT = $(shell expr `id -u` % 5000 + 25000)
# QEMU's gdb stub command line changed in 0.11
QEMUGDB = $(shell if $(QEMU) -help | grep -q '^-gdb'; \
  then echo "-gdb tcp::$(GDBPORT)"; \
  else echo "-s -p $(GDBPORT)"; fi)
ifndef CPUS
#CPUS := 2
CPUS := 1
endif
QEMUEXTRA = -snapshot
QEMUOPTS = -drive file=fs.img,index=1,media=disk,format=raw -drive file=xv6.img,index=0,media=disk,format=raw -smp $(CPUS) -m 512 $(QEMUEXTRA)

qemu: fs.img xv6.img
  $(QEMU) -serial mon:stdio $(QEMUOPTS)

qemu-memfs: xv6memfs.img
  $(QEMU) -drive file=xv6memfs.img,index=0,media=disk,format=raw -smp $(CPUS) -m 256

qemu-nox: fs.img xv6.img
  $(QEMU) -nographic $(QEMUOPTS)

.gdbinit: .gdbinit.tmpl
  sed "s/localhost:1234/localhost:$(GDBPORT)/" < $^ > $@

qemu-gdb: fs.img xv6.img .gdbinit
```

```
  @echo "*** Now run 'gdb'." 1>&2
  $(QEMU) -serial mon:stdio $(QEMUOPTS) -S $(QEMUGDB)

qemu-nox-gdb: fs.img xv6.img .gdbinit
  @echo "*** Now run 'gdb'." 1>&2
  $(QEMU) -nographic $(QEMUOPTS) -S $(QEMUGDB)

# CUT HERE
# prepare dist for students
# after running make dist, probably want to
# rename it to rev0 or rev1 or so on and then
# check in that version.

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 zombie.c\
  printf.c umalloc.c\
  README dot-bochsrc *.pl toc.* runoff runoff1 runoff.list\
  .gdbinit.tmpl gdbutil\

dist:
  rm -rf dist
  mkdir dist
  for i in $(FILES); \
  do \
    grep -v PAGEBREAK $$i >dist/$$i; \
  done
  sed '/CUT HERE/,$$d' Makefile >dist/Makefile
  echo >dist/runoff.spec
  cp $(EXTRA) dist

dist-test:
  rm -rf dist
  make dist
  rm -rf dist-test
  mkdir dist-test
  cp dist/* dist-test
  cd dist-test; $(MAKE) print
  cd dist-test; $(MAKE) bochs || true
  cd dist-test; $(MAKE) qemu

# update this rule (change rev#) when it is time to
# make a new revision.
tar:
  rm -rf /tmp/xv6
  mkdir -p /tmp/xv6
  cp dist/* dist/.gdbinit.tmpl /tmp/xv6
  (cd /tmp; tar cf - xv6) | gzip >xv6-rev10.tar.gz  # the next
one will be 10 (9/17)
```

```
288
289 .PHONY: dist-test dist
```

1.4 重新运行xv6

1. 在xv6-public目录下输入`make qumu-nox`，使xv6在qemu中运行
2. 输入命令big，会显示有140个sectors：

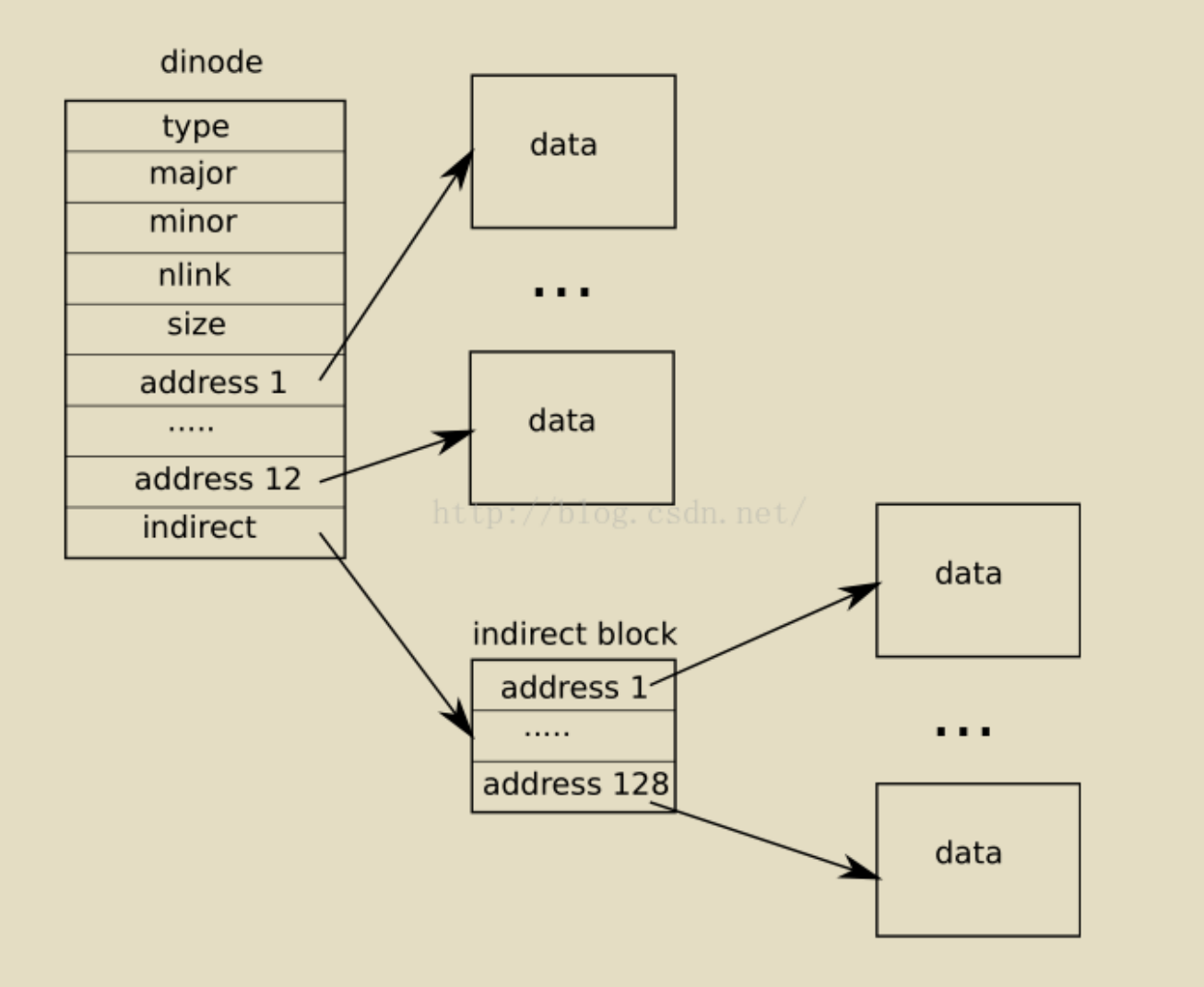
```
yuqianhou@yuqianhou-VirtualBox:~$ cd /home/yuqianhou/xv6-public
yuqianhou@yuqianhou-VirtualBox:~/xv6-public$ make qemu-nox
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-
pointer -fno-stack-protector -fno-pie -no-pie   -c -o umalloc.o umalloc.c
ld -m    elf_i386 -N -e main -Ttext 0 -o _cat cat.o ulib.o usys.o printf.o umalloc.o
objdump -S _cat > cat.asm
objdump -t _cat | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > cat.sym
ld -m    elf_i386 -N -e main -Ttext 0 -o _echo echo.o ulib.o usys.o printf.o umalloc.o
objdump -S _echo > echo.asm
objdump -t _echo | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > echo.sym
# forktest has less library code linked in - needs to be small
# in order to be able to max out the proc table.
ld -m    elf_i386 -N -e main -Ttext 0 -o _forktest forktest.o ulib.o usys.o
objdump -S _forktest > forktest.asm
ld -m    elf_i386 -N -e main -Ttext 0 -o _grep grep.o ulib.o usys.o printf.o umalloc.o
objdump -S _grep > grep.asm
objdump -t _grep | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > grep.sym
ld -m    elf_i386 -N -e main -Ttext 0 -o _init init.o ulib.o usys.o printf.o umalloc.o
objdump -S _init > init.asm
objdump -t _init | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > init.sym
ld -m    elf_i386 -N -e main -Ttext 0 -o _kill kill.o ulib.o usys.o printf.o umalloc.o
objdump -S _kill > kill.asm
objdump -t _kill | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > kill.sym
ld -m    elf_i386 -N -e main -Ttext 0 -o _ln ln.o ulib.o usys.o printf.o umalloc.o
objdump -S _ln > ln.asm
objdump -t _ln | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > ln.sym
ld -m    elf_i386 -N -e main -Ttext 0 -o _ls ls.o ulib.o usys.o printf.o umalloc.o
objdump -S _ls > ls.asm
objdump -t _ls | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > ls.sym
ld -m    elf_i386 -N -e main -Ttext 0 -o _mkdir mkdir.o ulib.o usys.o printf.o umalloc.o
objdump -S _mkdir > mkdir.asm
objdump -t _mkdir | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > mkdir.sym
ld -m    elf_i386 -N -e main -Ttext 0 -o _rm rm.o ulib.o usys.o printf.o umalloc.o
objdump -S _rm > rm.asm
objdump -t _rm | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > rm.sym
ld -m    elf_i386 -N -e main -Ttext 0 -o _sh sh.o ulib.o usys.o printf.o umalloc.o
objdump -S _sh > sh.asm
objdump -t _sh | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > sh.sym
ld -m    elf_i386 -N -e main -Ttext 0 -o _stressfs stressfs.o ulib.o usys.o printf.o umalloc.o
objdump -S _stressfs > stressfs.asm
objdump -t _stressfs | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > stressfs.sym
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-
```

```
pointer -fno-stack-protector -fno-pie -no-pie   -c -o spinlock.o spinlock.c
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-
pointer -fno-stack-protector -fno-pie -no-pie   -c -o syscall.o syscall.c
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-
pointer -fno-stack-protector -fno-pie -no-pie   -c -o sysfile.o sysfile.c
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-
pointer -fno-stack-protector -fno-pie -no-pie   -c -o sysproc.o sysproc.c
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-
pointer -fno-stack-protector -fno-pie -no-pie   -c -o trap.o trap.c
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-
pointer -fno-stack-protector -fno-pie -no-pie   -c -o uart.o uart.c
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-
pointer -fno-stack-protector -fno-pie -no-pie   -c -o vm.o vm.c
ld -m    elf_i386 -T kernel.ld -o kernel entry.o bio.o console.o exec.o file.o fs.o ide.o ioapic.o kallo
c.o kbd.o lapic.o log.o main.o mp.o picirq.o pipe.o proc.o sleeplock.o spinlock.o string.o swtch.o sysca
ll.o sysfile.o sysproc.o trapasm.o trap.o uart.o vectors.o vm.o  -b binary initcode entryother
objdump -S kernel > kernel.asm
objdump -t kernel | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > kernel.sym
dd if=/dev/zero of=xv6.img count=10000
记录了10000+0 的读入
记录了10000+0 的写出
5120000 bytes (5.1 MB, 4.9 MiB) copied, 0.0230371 s, 222 MB/s
dd if=bootblock of=xv6.img conv=notrunc
记录了1+0 的读入
记录了1+0 的写出
512 bytes copied, 0.00011645 s, 4.4 MB/s
dd if=kernel of=xv6.img seek=1 conv=notrunc
记录了333+1 的读入
记录了333+1 的写出
170804 bytes (171 kB, 167 KiB) copied, 0.000982721 s, 174 MB/s
qemu-system-i386 -nographic -drive file=fs.img,index=1,media=disk,format=raw -drive file=xv6.img,index=0
,media=disk,format=raw -smp 1 -m 512 -snapshot
xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ big
main-loop: WARNING: I/O thread spun for 1000 iterations
.
wrote 140 sectors
done; ok
$
```

二、问题需求分析

2.1 xv6中的inode分析

由下图可知，一个inode有12个direct pointers分别指向12个disk blocks，还有一个indirect pointer指向另一个indirect block。这个indirect block有 `BSIZE / sizeof(uint) = 128` 个指针指向disk blocks。因此，一个inode可以指向 `12 +128 = 140` 个数据块。也就是运行了 `big` 命令后输出的140个sectors。

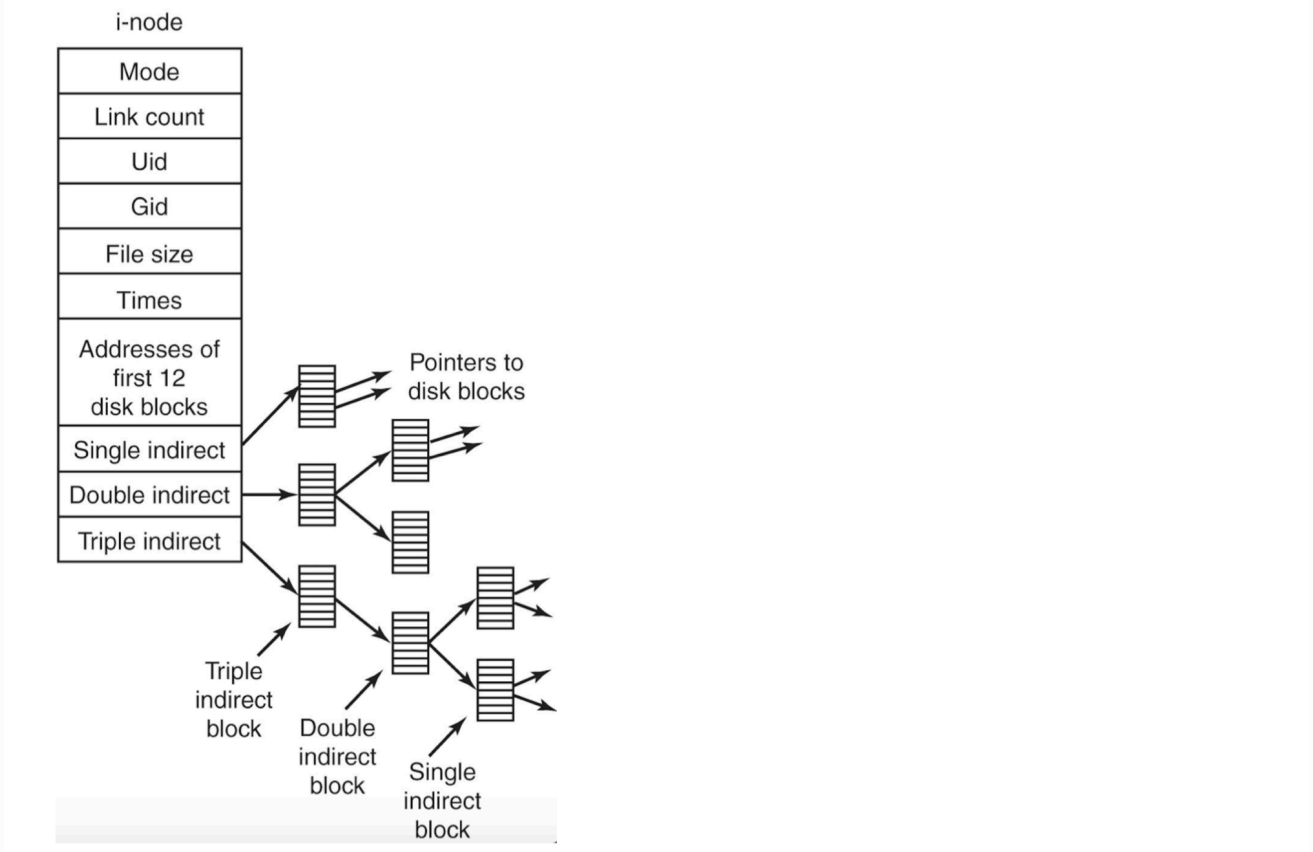


2.2 问题描述

修改 `bmap()` 函数，以便除了direct blocks和singly-indirect block之外，还实现doubly-indirect block。只有使用11个direct blocks，而不是12个，才能为新的doubly-indirect block腾出空间。无权更改磁盘inode的大小。ip-> addrs []的前11个元素应该是direct blocks；第十二个应该是一个singly-indirect block（就像当前的一样）；第13个应该是新的doubly-indirect block。

您无需修改xv6即可处理带有doubly-indirect blocks的文件删除。

如果一切顺利， `big` 的执行结果可以写入16523个sectors，这需要几十秒钟才能完成。



三、修改xv6代码的过程

3.1 修改fs.h文件

由于需要使用11个direct blocks，为新的doubly-indirect block腾出空间，因此需要对fs.h文件中的参数做一些修改：

1. 将 `#define NDIRECT 12` 修改为 `#define NDIRECT 11`
2. 将 `#define MAXFILE (NDIRECT + NINDIRECT)`

   修改为 `#define MAXFILE (NDIRECT + NINDIRECT + NINDIRECT * NINDIRECT)`
3. 将 `uint addrs[NDIRECT+1]; // Data block addresses`

   修改为 `uint addrs[NDIRECT+2]; // Data block addresses`

修改后的fs.h文件如下：

```
1  // On-disk file system format.
2  // Both the kernel and user programs use this header file.
3
4
```

```
5  #define ROOTINO 1  // root i-number
6  #define BSIZE 512  // block size
7
8  // Disk layout:
9  // [ boot block | super block | log | inode blocks |
10 //                                          free bit map | data
   blocks]
11 //
12 // mkfs computes the super block and builds an initial file system.
   The
13 // super block describes the disk layout:
14 struct superblock {
15   uint size;         // Size of file system image (blocks)
16   uint nblocks;      // Number of data blocks
17   uint ninodes;      // Number of inodes.
18   uint nlog;         // Number of log blocks
19   uint logstart;     // Block number of first log block
20   uint inodestart;   // Block number of first inode block
21   uint bmapstart;    // Block number of first free map block
22 };
23
24 //#define NDIRECT 12
25 #define NDIRECT 11
26 #define NINDIRECT (BSIZE / sizeof(uint))
27 //#define MAXFILE (NDIRECT + NINDIRECT)
28 #define MAXFILE (NDIRECT + NINDIRECT + NINDIRECT * NINDIRECT)
29
30 // On-disk inode structure
31 struct dinode {
32   short type;           // File type
33   short major;          // Major device number (T_DEV only)
34   short minor;          // Minor device number (T_DEV only)
35   short nlink;          // Number of links to inode in file system
36   uint size;            // Size of file (bytes)
37   //uint addrs[NDIRECT+1];   // Data block addresses
38     uint addrs[NDIRECT+2];   // Data block addresses
39 };
40
41 // Inodes per block.
42 #define IPB           (BSIZE / sizeof(struct dinode))
43
44 // Block containing inode i
45 #define IBLOCK(i, sb)     ((i) / IPB + sb.inodestart)
46
47 // Bitmap bits per block
48 #define BPB           (BSIZE*8)
49
50 // Block of free map containing bit for block b
51 #define BBLOCK(b, sb) (b/BPB + sb.bmapstart)
```

```
52
53  // Directory is a file containing a sequence of dirent structures.
54  #define DIRSIZ 14
55
56  struct dirent {
57    ushort inum;
58    char name[DIRSIZ];
59  };
```

3.2 修改fs.c文件

需要将原来的 `12+128` 改成 `11+128+128*128`：

修改后的bmap()函数如下：

```
1   static uint
2   bmap(struct inode *ip, uint bn)
3   {
4     uint addr, *a, *indirect, *doubleIndirect, indirectIdx,
    doubleIndirectIdx;
5     struct buf *bp, *bp2;
6
7     if(bn < NDIRECT){
8       if((addr = ip->addrs[bn]) == 0)
9         ip->addrs[bn] = addr = balloc(ip->dev);
10      return addr;
11    }
12    bn -= NDIRECT;
13
14    if(bn < NINDIRECT){
15      // Load indirect block, allocating if necessary.
16      if((addr = ip->addrs[NDIRECT]) == 0)
17        ip->addrs[NDIRECT] = addr = balloc(ip->dev);
18      bp = bread(ip->dev, addr);
19      a = (uint*)bp->data;
20      if((addr = a[bn]) == 0){
21        a[bn] = addr = balloc(ip->dev);
22        log_write(bp);
23      }
24      brelse(bp);
25      return addr;
26    }
27      bn -= NINDIRECT;
28      if (bn < NINDIRECT * NINDIRECT) {
29          // Load first indirect block, allocating if necessary.
```

```
30          if ((addr = ip->addrs[NDIRECT + 1]) == 0) {
31              ip->addrs[NDIRECT + 1] = addr = balloc(ip->dev);
32          }
33          bp = bread(ip->dev, addr);
34          indirect = (uint *)bp->data;
35          indirectIdx = bn / NINDIRECT;
36
37          if ((addr = indirect[indirectIdx]) == 0) {
38              addr = indirect[indirectIdx] = balloc(ip->dev);
39              log_write(bp);
40          }
41
42          bp2 = bread(ip->dev, addr);
43          doubleIndirect = (uint *)bp2->data;
44          doubleIndirectIdx = bn % NINDIRECT;
45
46          if ((addr = doubleIndirect[doubleIndirectIdx]) == 0) {
47              addr = doubleIndirect[doubleIndirectIdx] = balloc(ip-
    >dev);
48              log_write(bp2);
49          }
50
51          brelse(bp2);
52          brelse(bp);
53          return addr;
54      }
55
56    panic("bmap: out of range");
57  }
```

四、测试结果

重新编译运行xv6，重新运行big命令：

```
xv6...
cpu0: starting 0
sb: size 20000 nblocks 19937 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ big
.......................................main-loop: WARNING: I/O thread spun for 1000 iterations
.....................................................................................................
.....................
wrote 16523 sectors
done; ok
$
```

由上图可知，big命令运行后过了几十秒，sectors从140变成了16523（`= 11(singly-indirect blocks) + 128(direct blocks) + 128*128(doubly-indirect blocks)`）。