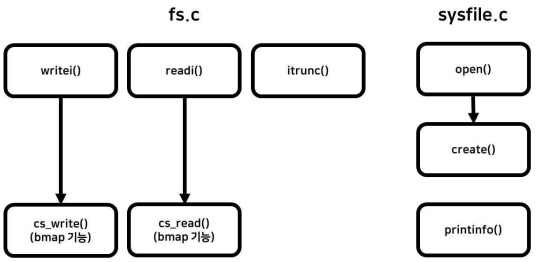
1. 개요

1) CS 기반 파일 시스템 구현
- 기존 파일 시스템에서 관리되는 파일과의 구분을 위해 stat.h에 CS 기반 파일 타입(T_CS) 추가
- CS 기반 파일을 생성하기 위해 fcntl.h에 플래그(O_CS) 추가
- CS 기반 파일을 위한 데이터 할당 및 삭제 메커니즘 구현
* inode의 direct 블록(4B)을 번호 영역(3B)과 길이 영역(1B)으로 나누어 사용
* 번호영역: 할당되는 데이터 블록의 시작 번호, 길이영역: 연속으로 할당되는 데이터 블록의 개수
* 비어 있는 direct 블록 할당 받아 쓰기 수행
    하나의 direct 블록에서 관리하는 길이 영역의 크기(1B)를 초과한 경우
    데이터 블록의 연속적인 할당이 중단된 경우
* CS 기반 파일 삭제 시 데이터 블록의 시작 번호부터 길이만큼 기존에 할당되었던 데이터 블록을 모두 해제
* CS 기반 파일 시스템에서 indirect 블록은 사용하지 않음
* 할당할 수 있는 데이터 블록이나 direct 블록의 범위를 초과할 경우 범위 내까지만 데이터 할당 후 에러 메시지 출력

2) 파일 정보 출력 함수 구현
- 파일 디스크립터(fd)가 인자로 주어지면 파일의 정보를 출력하는 printinfo() 함수 구현
- 함수에서 출력되는 파일의 정보는 파일의 inode 번호, 타입, 크기, direct 블록에 저장된 내용임
- 기존 파일 시스템에서 관리하는 파일:
    파일 타입: FILE, direct 블록 정보: 사용중인 direct 블록에 저장된 내용
- CS 기반 파일 시스템에서 관리하는 파일:
    파일 타입: CS, direct 블록 정보: 사용중인 direct 블록에 저장된 내용&(번호, 길이) 정보



fs.c
writei():
    CS 타입일 경우 bamp() 대신 cs_write() 사용 후 리턴 값으로 디스크 번호 계산 추가
    파일 범위를 초과하는지 체크
readi(): CS 타입일 경우 bamp() 대신 cs_read() 사용 후 리턴 값으로 디스크 번호 계산 추가
itrunc(): CS 타입일 경우 direct 블록 할당 해제 추가
cs_write():
    bmap 기능 수행, 비어있으면 direct 블록에 번호 및 길이 할당, 비어있지 않으면 길이 1 증가
    연속하지 않으면 인덱스 1 추가
    인덱스가 최대 direct 블록을 초과하는지 체크
cs_read():
    bamp 기능 수행, 인덱스를 수정하고 direct 블록 값을 리턴
    인덱스가 최대 direct 블록을 초과하는지 체크
sysfile.c
open(): 타입이 CS일 경우 추가
create(): 타입이 CS일 경우 추가
printinfo(): 파일 정보 출력

2. 소스코드

* 할당 가능한 디스크 블록 초과 시 기존 fs.c의 balloc()의 panic("balloc: out of blocks") 에러 출력

1) fs.c

```
403 //20182705
404 //inode type이 CS일 경우, writei에 사용되는 bmap 함수의 역할 수행
405 static uint
406 cs_write(struct inode *ip, uint bn)
407 {
408   //direct 블록 저장 변수
409   uint addr;
410   //번호가 저장될 direct 블록의 기존 bmap 인덱스를 255로 나눈 몫으로 나눠 사용
411   //하나의 블록은 255개의 디스크 블록 저장 가
412   bn = bn / 255;
413
414   //인덱스가 최대 direct 블록 수보다 작을 경우
415   if(bn < NDIRECT){
416     //inode의 type이 CS일 경우
417     addr = ip->addrs[bn];
418     //새로운 디스크 블록 번호 할당
419     uint b = balloc(ip->dev);
420
421     //이전에 데이터 블록의 연속적 할당이 중단된 경우: direct 블록 인덱스 1증가
422     //direct 블록에서의 개수가 255로 꽉 차있을 경우: direct 블록 인덱스 1증가
423     if((addr != 0 && (addr>>8) + (addr&255) != b) || (addr&255) ==255)
424       bn++;
425
426     //지정된 direct 블록이 비어있을 경우:
427     //상위 3B - 새로 할당된 디스크 블록 번호, 하위 1B - 블록 개수 1로 초기화
428     //지정된 direct 블록이 비어있지 않을 경우
429     //상위 3B - 처음 할당된 디스크 블록 번호, 하위 1B - 블록 개수 1 증가
430     if((addr = ip->addrs[bn]) == 0)
431       addr = (b << 8) + 1;
432     else
433       addr += 1;
434
435     //direct의 블록에 저장
436     ip->addrs[bn] = addr;
437
438     //저장한 블록 리턴
439     return addr;
440   }
441
442   //가능한 direct 범위 초과: 에러 출력
443   panic("cs_write: out of range");
444 }
```

```
446 //20182705
447 //inode type이 CS일 경우, readi에 사용되는 bamp 함수 역할 수행
448 static uint
449 cs_read(struct inode *ip, uint bn)
450 {
451   uint dir_buf[NDIRECT];  //각 direct 블록의 수
452   uint length = bn;       //direct 블록에서의 번호, bn으로 초기화
453   uint idx = 0;           //속한 direct 블록의 번호
454
455   //인덱스가 최대 direct 블록 수보다 작을 경우
456   if(bn/255 < NDIRECT){
457     //Direct 블록 수만큼 반복
458     for(int i=0; i<NDIRECT; i++){
459       //direct 블록의 개수 저장
460       //만약 해당 Direct 블록이 비어있으면 탈출
461       if((dir_buf[i] = (ip->addrs[i])&255)==0)
462         break;
463       //bn(offset/BIZE) Direct블록의 디스크 블록수보다 적을 경우: 인덱스 저장 후 탈출
464       if(length < dir_buf[i]){
465         idx = i;
466         break;
467       }
468       //bn이 Direct블록의 디스크 블록수보다 많거나 같을 경우: length-블록 수
469       else
470         length  -= dir_buf[i];
471     }
472     //direct 블록 인덱스 및 해당 블록의 순서를 사용하여 디스크 블록 계산 및 리턴
473     uint addr = (ip->addrs[idx]>>8) + length;
474     return addr;
475     //가능한 direct 범위 초과: 에러 출력
476   }
477   panic("cs_read: out of range");
478 }
```

```
473 static void
474 itrunc(struct inode *ip)
475 {
476   int i, j;
477   struct buf *bp;
478   uint *a;
479   //20182705
480   //inode의 type이 CS일 경우
481   if(ip->type==4){
482     //direct 블록 탐색
483     for(i = 0; i < NDIRECT; i++){
484       //해당 direct 블록이 비어 있지 않을 경우
485       if(ip->addrs[i]){
486         //시작 번호부터 개수만큼의 디스크 할당 해제
487         for(j = 0; j < (ip->addrs[i]&255); j++)
488             bfree(ip->dev, (ip->addrs[i] >> 8) + j);
489         //direct 블록 할당 해제
490         ip->addrs[i] = 0;
491       }
492     }
493   }
494   //inode type이 CS가 아닐 경우
495   else{
496     for(i = 0; i < NDIRECT; i++){
497       if(ip->addrs[i]){
498         bfree(ip->dev, ip->addrs[i]);
499         ip->addrs[i] = 0;
500       }
501     }
502   }
```

```
520 //20182705
521 //cs파일 inode direct 블록 디스크 할당 해제
522 void
523 cs_trunc(struct inode *ip)
524 {
525   itrunc(ip);
526 }
```

```
555 int
556 readi(struct inode *ip, char *dst, uint off, uint n)
557 {
558   uint tot, m;
559   struct buf *bp;
560
561   if(ip->type == T_DEV){
562     if(ip->major < 0 || ip->major >= NDEV || !devsw[ip->major].read)
563       return -1;
564     return devsw[ip->major].read(ip, dst, n);
565   }
566
567   if(off > ip->size || off + n < off)
568     return -1;
569   if(off + n > ip->size)
570     n = ip->size - off;
571
572   for(tot=0; tot<n; tot+=m, off+=m, dst+=m){
573     //20182705
574     //inode type이 cs인 경우
575     if(ip->type==4){
576       bp = bread(ip->dev, cs_read(ip, off/BSIZE));
577     }
578     //inode type이 cs가 아닐 경우
579     else
580       bp = bread(ip->dev, bmap(ip, off/BSIZE));
581     m = min(n - tot, BSIZE - off%BSIZE);
582     memmove(dst, bp->data + off%BSIZE, m);
583     brelse(bp);
584   }
585   return n;
586 }
```

```
591 int
592 writei(struct inode *ip, char *src, uint off, uint n)
593 {
594   uint tot, m;
595   struct buf *bp;
596
597   if(ip->type == T_DEV){
598     if(ip->major < 0 || ip->major >= NDEV || !devsw[ip->major].write)
599       return -1;
600     return devsw[ip->major].write(ip, src, n);
601   }
602
603   if(off > ip->size || off + n < off)
604     return -1;
605
606   //20182705
607   //inode type이 cs인 경우
608   if(ip->type==4){
609     //direct 블록의 범위를 초과한 경우: 에러 출력
610     if(off + n > NDIRECT*BSIZE*255)
611       return -1;
612   }
613   //inode type이 cs가 아닌 경우
614   else{
615     if(off + n > MAXFILE*BSIZE)
616       return -1;
617    }
618
619   for(tot=0; tot<n; tot+=m, off+=m, src+=m){
620     //20182705
621     //inode type이 cs인 경우
622     if(ip->type==4){
623       //저장된 direct 블록을 불러옴
624       uint temp = cs_write(ip, off/BSIZE);
625       //초기값과 길이를 사용하여 해당 번호를 계산
626       uint a = (temp >> 8) + (temp & 255) - 1;
627       //해당 번호의 디스크 정보 가져옴
628       bp = bread(ip->dev, a);
629     }
630     //inode type이 cs가 아닌 경
631     else
632       bp = bread(ip->dev, bmap(ip, off/BSIZE));
633     m = min(n - tot, BSIZE - off%BSIZE);
634     memmove(bp->data + off%BSIZE, src, m);
635     log_write(bp);
```

2) sysfile.c

```
241 static struct inode*
242 create(char *path, short type, short major, short minor)
243 {
244   struct inode *ip, *dp;
245   char name[DIRSIZ];
246
247   if((dp = nameiparent(path, name)) == 0)
248     return 0;
249   ilock(dp);
250
251   if((ip = dirlookup(dp, name, 0)) != 0){
252     iunlockput(dp);
253     ilock(ip);
254     if(type == T_FILE && ip->type == T_FILE){
255       return ip;
256     }
257     //20182705
258     //CS파일이 존재할 시, inode direct 초기화 후 inode 리턴
259     else if(type == T_CS && ip->type==T_CS){
260       cs_trunc(ip);
261       return ip;
262     }
263     iunlockput(ip);
264     return 0;
265   }
266
267   if((ip = ialloc(dp->dev, type)) == 0)
268     panic("create: ialloc");
269
270   ilock(ip);
271   ip->major = major;
272   ip->minor = minor;
273   ip->nlink = 1;
274   //20182705
275   //inode type 변경
276   ip->type = type;
277   iupdate(ip);
278
279   if(type == T_DIR){  // Create . and .. entries.
280     dp->nlink++;  // for ".."
281     iupdate(dp);
282     // No ip->nlink++ for ".": avoid cyclic ref count.
283     if(dirlink(ip, ".", ip->inum) < 0 || dirlink(ip, "..", dp->inum) < 0)
284       panic("create dots");
285   }
```

```
295 int
296 sys_open(void)
297 {
298   char *path;
299   int fd, omode;
300   struct file *f;
301   struct inode *ip;
302
303   if(argstr(0, &path) < 0 || argint(1, &omode) < 0)
304     return -1;
305
306   begin_op();
307
308   //20182705
309   //open모드에 O_CS가 있을 경우
310   if(omode & O_CS){
311     //CS 타입의 inode 생성
312     ip = create(path, T_CS, 0, 0);
313     if(ip == 0){
314       end_op();
315       return -1;
316     }
317   } else if(omode & O_CREATE){
318     ip = create(path, T_FILE, 0, 0);
319     if(ip == 0){
320       end_op();
321       return -1;
322     }
323   } else {
324     if((ip = namei(path)) == 0){
325       end_op();
326       return -1;
327     }
328     ilock(ip);
329     if(ip->type == T_DIR && omode != O_RDONLY){
330       iunlockput(ip);
331       end_op();
332       return -1;
333     }
334   }
335
```

```
465 //20182705
466 //printinfo 함수 정의
467 int
468 sys_printinfo(void)
469 {
470   struct file *f; //읽을 파일
471   int n = 256;    //이름 버퍼 길이
472   char *p;        //이름 버퍼
473
474   //파일 포인터나 파일 이름이 입력되지 않았을 경우: -1 리턴
475   if(argfd(0, 0, &f) < 0 || argptr(1, &p, n) < 0)
476     return -1;
477
478   cprintf("FILE NAME: %s\n", p);                  //파일 이름
479   cprintf("INODE NUM: %d\n", f->ip->inum);        //INODE 번호력
480   if(f->ip->type == 2)                            //파일 타입
481     cprintf("FILE TYPE: FILE\n");                 //File일 경우
482   else if(f->ip->type == 4)
483     cprintf("FILE TYPE: CS\n");                   //CS일 경우
484   cprintf("FILE SIZE: %d Bytes\n", f->ip->size);  //파일 크기
485   cprintf("DIRECT BLOCK INFO:\n");                //DIRECT 블럭 정보
486   if(f->ip->type == 4){                           //CS일 경우
487     for(int i=0; i<NDIRECT; i++){                 //블럭 순회
488       if(f->ip->addrs[i])                         //블럭이 비어있지 않은 경우: 값 출력(초기 번호, 길이)
489         cprintf("[%d] %d (num: %d, length: %d)\n", i, f->ip->addrs[i], f->ip->addrs[i]/256, f->ip->addrs[i]%256);
490     }
491   }
492   else{                                           //CS가 아닌 경우
493     for(int i=0; i<NDIRECT; i++){                 //블럭 순회
494       if(f->ip->addrs[i])                         //블럭이 비어있지 않은 경우: 번호 출
495         cprintf("[%d] %d\n", i, f->ip->addrs[i]);
496     }
497   }
498
499   return 0;
500 }
```

3) Makefile

test 시스템콜 추가

```
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   _test\
```

```
251 EXTRA=\
252   mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c kill.c\
253   ln.c ls.c mkdir.c rm.c stressfs.c usertests.c wc.c zombie.c test.c\
254   printf.c umalloc.c\
255   README dot-bochsrc *.pl toc.* runoff runoff1 runoff.list\
256   .gdbinit.tmpl gdbutil\
```

4) syscall.h

```
23 #define SYS_printinfo 22  //20182705 printinfo 추가
```

5) syscall.c

```
106 extern int sys_printinfo(void); //20182705 printinfo 추가
```

```
130 [SYS_printinfo] sys_printinfo,  //20128705 printinfo 추가
```

6) user.h

```
26 int printinfo(int, const void*);  //20182705 printinfo 시스템콜 추가
```

7) usys.S

```
32 SYSCALL(printinfo)  //20182705 printinfo 추가
```

8) fcntl.h

```
#define O_RDONLY  0x000
#define O_WRONLY  0x001
#define O_RDWR    0x002
#define O_CREATE  0x200
#define O_CS    0x020 //20182705 open()함수 호출 시 추가한 플래그 사용 위한 루틴
```

9) stat.h

```
#define T_DIR  1   // Directory
#define T_FILE 2   // File
#define T_DEV  3   // Device
#define T_CS   4   // 20182705 Continuous Sector based Fil
```

10) defs.h

```
// fs.c
void            readsb(int dev, struct superblock *sb);
int             dirlink(struct inode*, char*, uint);
struct inode*   dirlookup(struct inode*, char*, uint*);
struct inode*   ialloc(uint, short);
struct inode*   idup(struct inode*);
void            iinit(int dev);
void            ilock(struct inode*);
void            iput(struct inode*);
void            iunlock(struct inode*);
void            iunlockput(struct inode*);
void            iupdate(struct inode*);
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);
void            cs_trunc(struct inode *ip); //20182705 cs_trunc 함수 추가
```

11) test.c

수정사항 없음

```
#include "types.h"
#include "param.h"
#include "stat.h"
#include "mmu.h"
#include "proc.h"
#include "fs.h"
#include "spinlock.h"
#include "sleeplock.h"
#include "file.h"
#include "fcntl.h"
#include "user.h"

#define BUFMAX 1024

void write_to_norm(char* fname);
```

```
int main()
{
  int fd;
  int i, size;
  char buf[BUFMAX];
  char* fname = "test_cs";

  if ((fd = open(fname, O_CREATE | O_CS | O_RDWR)) <= 0) {
    printf(2, "ERROR: open failed in cs file\n");
    exit();
  }

  for (i = 0; i < BUFMAX; i++)
    buf[i] = 'a';

  for (i = 0; i < 130; i++) {
    if ((size = write(fd, buf, BUFMAX)) != BUFMAX) {
      printf(2, "ERROR: write failed in cs file\n");
      exit();
    }

    if (i == 50)
      write_to_norm("test_norm");
  }


  printinfo(fd, fname);

  close(fd);
  exit();
}
```

```
void write_to_norm(char* fname)
{
  int fd;
  int i, size;
  char buf[BUFMAX];

  if ((fd = open(fname, O_CREATE | O_RDWR)) <= 0) {
    printf(2, "ERROR: open failed in normal file\n");
    exit();
  }

  for (i = 0; i < BUFMAX; i++)
    buf[i] = 'b';

  for (i = 0; i < 2; i++) {
    if ((size = write(fd, buf, BUFMAX)) != BUFMAX) {
      printf(2, "ERROR: write failed in normal file\n");
      exit();
    }
  }

  printinfo(fd, fname);

  close(fd);
}
```

3. 결과

1) 테스트 케이스1 - CS 기반 파일에 연속적으로 데이터를 쓰는 경우

```
$ kibum@Ubuntu:~/dd/xv6-public$ make qemu
qemu-system-i386 -serial mon:stdio -drive file=fs.img,index=1,media=disk,format=raw -drive file=xv6.img,index=0,media=disk,format=raw -smp 2 -m 512
xv6...
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
$ test
FILE NAME: test_cs
INODE NUM: 20
FILE TYPE: CS
FILE SIZE: 133120 Bytes
DIRECT BLOCK INFO:
[0] 180735 (num: 705, length: 255)
[1] 245765 (num: 960, length: 5)
$ ls
.              1 1 512
..             1 1 512
README         2 2 2286
cat            2 3 16276
echo           2 4 15128
forktest       2 5 9440
grep           2 6 18492
init           2 7 15712
kill           2 8 15156
ln             2 9 15012
ls             2 10 17644
mkdir          2 11 15252
rm             2 12 15236
sh             2 13 27876
stressfs       2 14 16144
usertests      2 15 67252
wc             2 16 17008
zombie         2 17 14824
test           2 18 19372
console        3 19 0
test_cs        4 20 133120
$ wc test_cs
0 1 133120 test_cs
$
```

2) 테스트 케이스2 - CS 기반 파일에 불연속적으로 데이터를 쓰는 경우

```
$ kibum@Ubuntu:~/dd/xv6-public$ make qemu
qemu-system-i386 -serial mon:stdio -drive file=fs.img,index=1,media=disk,format=raw -drive file=xv6.img,index=0,media=disk,format=raw -smp 2 -m 512
xv6...
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
$ test
FILE NAME: test_norm
INODE NUM: 21
FILE TYPE: FILE
FILE SIZE: 2048 Bytes
DIRECT BLOCK INFO:
[0] 808
[1] 809
[2] 810
[3] 811
FILE NAME: test_cs
INODE NUM: 20
FILE TYPE: CS
FILE SIZE: 133120 Bytes
DIRECT BLOCK INFO:
[0] 180838 (num: 706, length: 102)
[1] 208030 (num: 812, length: 158)
$ ls
```

```
$ ls
.              1 1 512
..             1 1 512
README         2 2 2286
cat            2 3 16276
echo           2 4 15128
forktest       2 5 9440
grep           2 6 18492
init           2 7 15712
kill           2 8 15156
ln             2 9 15012
ls             2 10 17644
mkdir          2 11 15252
rm             2 12 15236
sh             2 13 27876
stressfs       2 14 16144
usertests      2 15 67252
wc             2 16 17008
zombie         2 17 14824
test           2 18 19668
console        3 19 0
test_cs        4 20 133120
test_norm      2 21 2048
$ wc test_cs
0 1 133120 test_cs
$ wc test_norm
0 1 2048 test_norm
$
```