Design and description of EXTENT file block allocate

1. Header files and functions that being changed

- fcntl.h

  this file define the file flag value, add O_EXTENT to this file so that when open() is called with this flag, kernel can recognize it

```
#define O_RDONLY  0x000
#define O_WRONLY  0x001
#define O_RDWR    0x002
#define O_CREATE  0x200
#define O_EXTENT  0x010
```

- stat.h

  this file define the file type, add T_FILE_EXTENT macro so that kernel can identify extent file type

```
#define T_FILE 2   // File
#define T_DEV  3   // Device
#define T_FILE_EXTENT 4   // EXTENT TYPE File
struct stat {
  short type;  // Type of file
  int dev;     // File system's disk device
  uint ino;    // Inode number
```

- fs.h

  add NEXTENT to this define, NEXTENT is the max size of each extent, since we use last 1 byte of each ip->addrs[NDIRECT+1] as length, NEXTENT=2^8-1

```
#define NDIRECT 12
#define NINDIRECT (BSIZE / sizeof(uint))
#define NEXTENT 256
#define MAXFILE (NDIRECT + NINDIRECT)
```

- sys_open() in sysfile.c

  add condition of create EXTENT file, if flag==O_EXTENT, create file with type T_FILE_EXTENT

```
sys_open(void)
{
  char *path;
  int fd, omode;
  struct file *f;
  struct inode *ip;
  if(argstr(0, &path) < 0 || argint(1, &omode) < 0)
```

```
    return -1;


  begin_op();


  if(omode & O_CREATE){
if(omode&O_EXTENT){
  ip = create(path, T_FILE_EXTENT, 0, 0);
}else{
    ip = create(path, T_FILE, 0, 0);
}
```

- create() in sysfile.c

  On condition that when create can't find file, it will create new file, add new file of type==T_FILE_EXTENT

```
 if((ip = dirlookup(dp, name, &off)) != 0){
    iunlockput(dp);
    ilock(ip);
    if((type == ip->type)&&((type == T_FILE)||(type == T_FILE_EXTENT)))
      return ip;
    iunlockput(ip);
    return 0;
 }
```

- bmap() in fs.c

  add EXTENT file map condition, if file type is EXTENT and first addr==0, that means no data block has been allocated for the file yet, call balloc() to allocate one any block as first data block of the file

```
if((ip->type==T_FILE_EXTENT) && bn<NEXTENT*13){



    if(ip->addrs[0]==0) {
      start= balloc(ip->dev);
      length=1;
      ip->addrs[0]=addr=(start<<8)|length;
      cprintf("bmap reach addrs[0]==0 return %p \n",start);
      return start;
    }else{
          cprintf("bmap reach addrs[0]!=0 \n");
```

```
int cadrr=0;
while((addr=ip->addrs[cadrr])!=0&&cadrr<13){

    start=(uint) (addr >> 8);
    length=(uint) (addr & 0xFF);
    if((bn=bn-((int) length))<0){
      cprintf("bmap reach found existing bn block, return bn block address %p \n",start+(bn+length));
        return start+(bn+length);
    }
    cadrr++;
            }
```

- if the first address!=0, that means at least one datablock has been assignated to this file, we need to look at the data next to the allocated block to see if tne continuous block is available, if yes, use this next continuous block as new allocated block, if no, allocate another block anywhere, use the first 3 bytes of the next unused ip->addr pointing to new block pointer, last 1 byte of this unused ip->addr records length

```
if(bn!=0)     panic("bmap: allocate more than one block \n");
 bp = bread(ip->dev, BBLOCK(start, sb));
 next=start+length;

   m = 1 << (next % 8);
     if(((bp->data[next/8] & m) != 0)|(length==16)){
        brelse(bp);
        start= balloc(ip->dev);
        length=1;
        ip->addrs[cadrr]=addr=(start<<8)|length;
        cprintf("bmap reach next block not available, return new block address %p
ip->addrs[%p]=%p\n",start,cadrr,addr);


        return start;
    }else{    // block free
         bp->data[next/8] |= m;    // Mark block in use.
         log_write(bp);
         brelse(bp);
         bzero(ip->dev, next);
         length++;
```

```
                    ip->addrs[cadrr-1]=addr=(start<<8)|length;
                   cprintf("bmap reach next block available, return next block address %p
ip->addrs[%p]=%p\n",start+length-1,cadrr-1,addr);
                    return start+length-1;


              }
```

2. **Block allocate strategy**
   - Since we have the data struct of uint ip->addrs[NDIRECT+1], which is an array of 13 uint.
   - Use first 3 bytes of each uint as pointer to pointing the start of continuous blocks, last 1 byte of the each uint to record the length of continuous blocks.
     So we would have 13 continous blocks that we can use for the EXTENT file
   - If the ip->addrs[0]==0, that means no data block has been allocated to file yet, then just allocate any one block
   - If ip->addrs[0]!=0, we need to iterate the ip-addrs[] to find the bn block, if the length is less than bn, that means we need create one new block
   - To create one new block, first we see if the next block of the end continuous blocks are available, if yes, keep the start point, just length++
   - If the next block of the end continuous blocks is not available, we need allocate one block somewhere else, use a new ip-addr[] to record the new block and mark length=1