OS MP4 FileSystem

組員
黃心佑
蔡玉倫

Achievement:

Part I

Understanding NachOS file system

Part I : Trace filesystem code

After entering ../build.linux/nachos -f, NachOS kernel would start to format the filesytem. Let's go through what formatting does

```
PersistentBitmap *freeMap = new PersistentBitmap(NumSectors);
Directory *directory = new Directory(NumDirEntries);
FileHeader *mapHdr = new FileHeader;
FileHeader *dirHdr = new FileHeader;
```

1. Create a bitmap freeMap for managing all the sectors in DISK
2. Create a root directory
3. Create file headers for freeMap and the root directory, respectively
4. By 3., we can know NachOS treats both freeMap and the root directory as files

```
#define FreeMapSector         0
#define DirectorySector       1
```

```
freeMap->Mark(FreeMapSector);
freeMap->Mark(DirectorySector);
```

1. Mark sector #0 and #1 as the used ones
2. One is for freeMap and the other one is for the root directory

Part I : Trace filesystem code

```
ASSERT(mapHdr->Allocate(freeMap, FreeMapFileSize));
ASSERT(dirHdr->Allocate(freeMap, DirectoryFileSize));
```

1.Allocate the data sectors for freeMap and the root directory, respectively
2.Data in freeMap is a bitmap for all the sectors
3.Data in the root directory is the files' sector number in the directory

```
mapHdr->WriteBack(FreeMapSector);
dirHdr->WriteBack(DirectorySector);
```

1. after allocating data sectors, the data sector records in both headers are change
2. it save the both headers to sector#0 and #1, respectively.

```
freeMapFile = new OpenFile(FreeMapSector);
directoryFile = new OpenFile(DirectorySector);
```

1. create the OpenFile instances for them.

```
DEBUG(dbgFile, "Writing bitmap and directory back to disk.");
freeMap->WriteBack(freeMapFile);     // flush change

directory->WriteBack(directoryFile);
```

1. because there are some free sectors allocated, it must write the bitmap back to DISK
2. Also, the data in Directory is empty, it would write this information back to DISK

Part I : Trace filesystem code

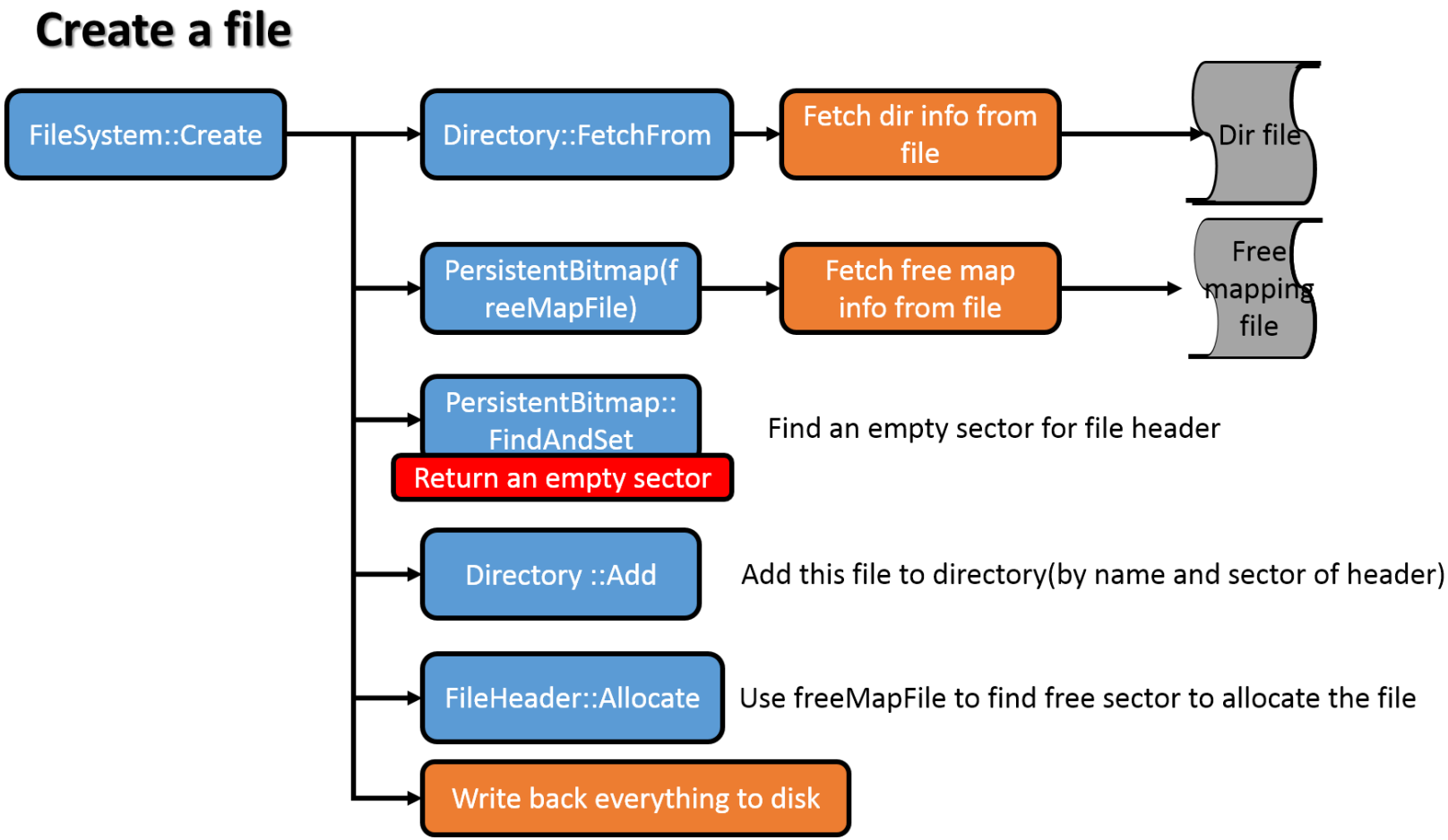
```
class OpenFile {
  public:
...
  private:
    FileHeader *hdr;
    int seekPosition;
};
```

```
class FileHeader {
  public:
    ...
  private:
    ...

    int numBytes;           // Number of bytes
    int numSectors;         // Number of data s
    int fileDescriptor;
    int numLevel;
    int dataSectors[NumDirect];     // Disk sec

                        // block in the file
};
```

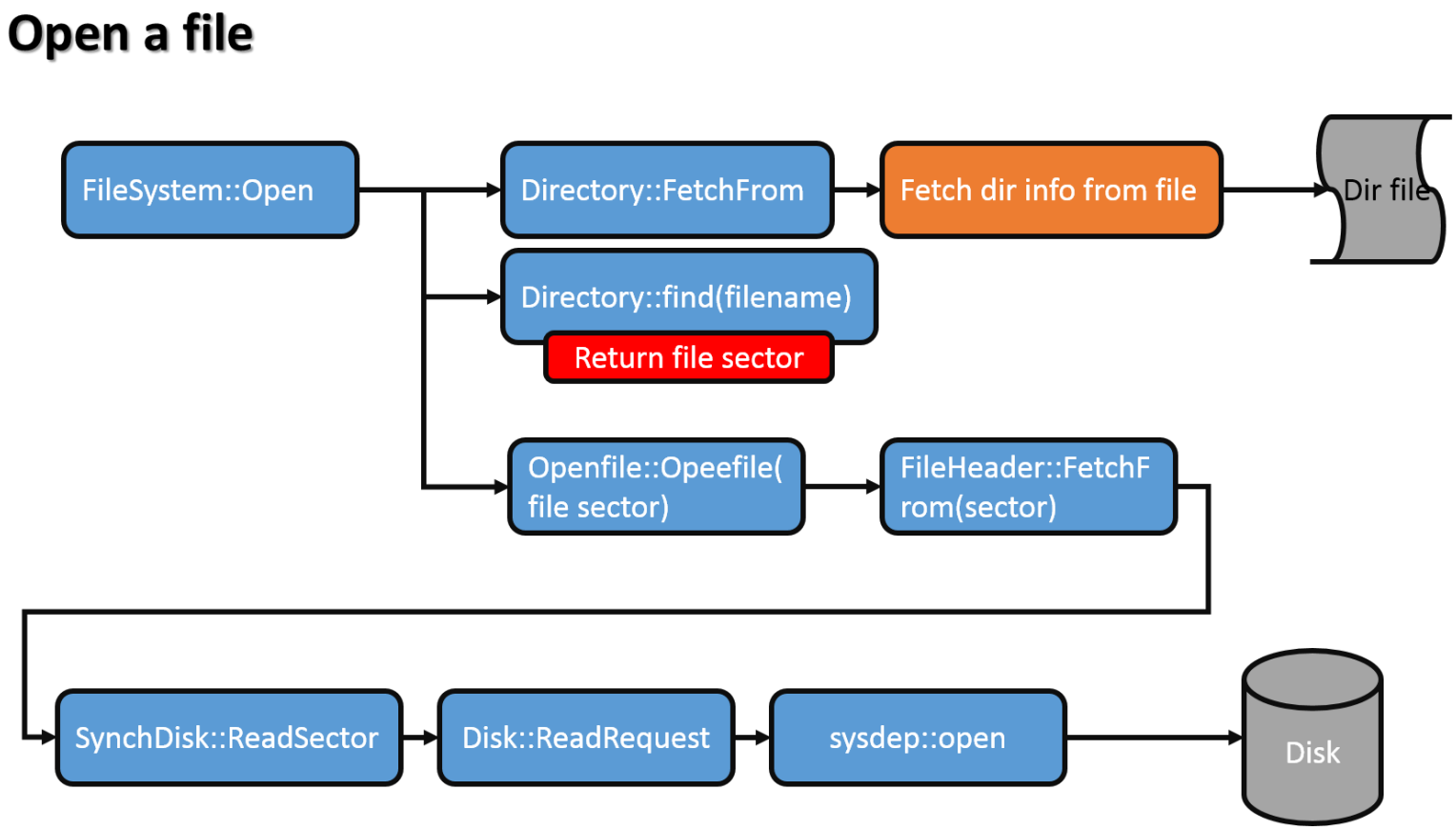
1.The file header of a file is in OpenFile instance.
2.Once NachOS creates a OpenFile instance for a file, the file header is also created.
3.we can see that the data sector number information is store in dataSectors[], so it implements **Index Allocation Scheme** for NachOS FileSystem

Part I : Trace filesystem code

The Creating a file flow can be shown as the function block figure in brief:



The Opening a file flow can be shown as the function block figure in brief:



Part I : Answer the Questions

(1) Explain how does the NachOS FS manage and find free block space? Where is this information stored on the raw disk (which sector)?

A:

用一個bitmap來管理所有的free sectors，而NachOS也會視這個bitmap為一個file來管理，OpenFile的address存於Filesystem::freeMapFile中。

而這個file的header存在第0個sector (in filesys.cc: #define FreeMapSector 0)

(2) What is the maximum disk size can be handled by the current implementation? Explain why.

A: 128KB

In disk.*

DiskSize = (MagicSize + (NumSectors * SectorSize));

NumSectors = (SectorsPerTrack * NumTracks);

where

SectorSize = 128;

SectorsPerTrack = 32;

NumTracks = 32;

MagicSize = sizeof(int) = 4;

so we got

DiskSize = 4 + (32 * 32) * 128

= 131,076 bytes

131,076 bytes/2^10=128KB

(3) Explain how does the NachOS FS manage the directory data structure? Where is this information stored on the raw disk (which sector)?

A:

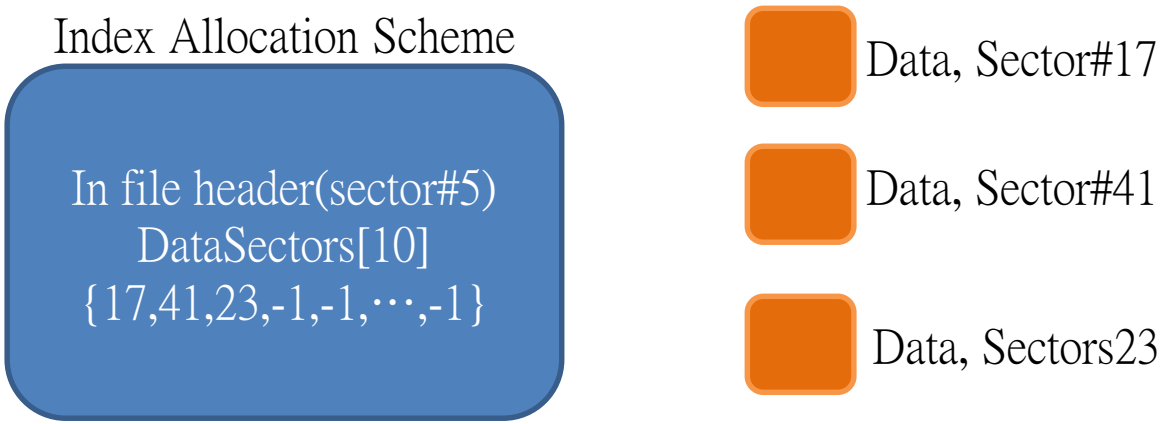
跟free sectors一樣，用一個file來存這些資訊，OpenFile會存放於Filesystem::directoryFile，而這個file header存在第1個sector (in filesys.cc: #define DirectorySector 1).

而directory的data會另外根據希望的最大的files個數存放在足夠的data sector中，每個file在directory所存放的資訊(DirectoryEntry)如下圖

```
class DirectoryEntry {
  public:
    bool inUse;                 // Is this direc
    int sector;                 // Location on d
                        //   FileHeader for this
    char name[FileNameMaxLen + 1];  // Text
                        // the trailing '\0'
};
```

(4) Explain what information is stored in an inode, and use a figure to illustrate the disk allocation scheme of current implementation.

```
int numBytes; //這個file總共多少bytes
int numSectors; //這個file總共占多少sectors
int dataSectors[NumDirect]; //使用Disk上哪些sector
```



(5) Why a file is limited to 4KB in the current implementation?

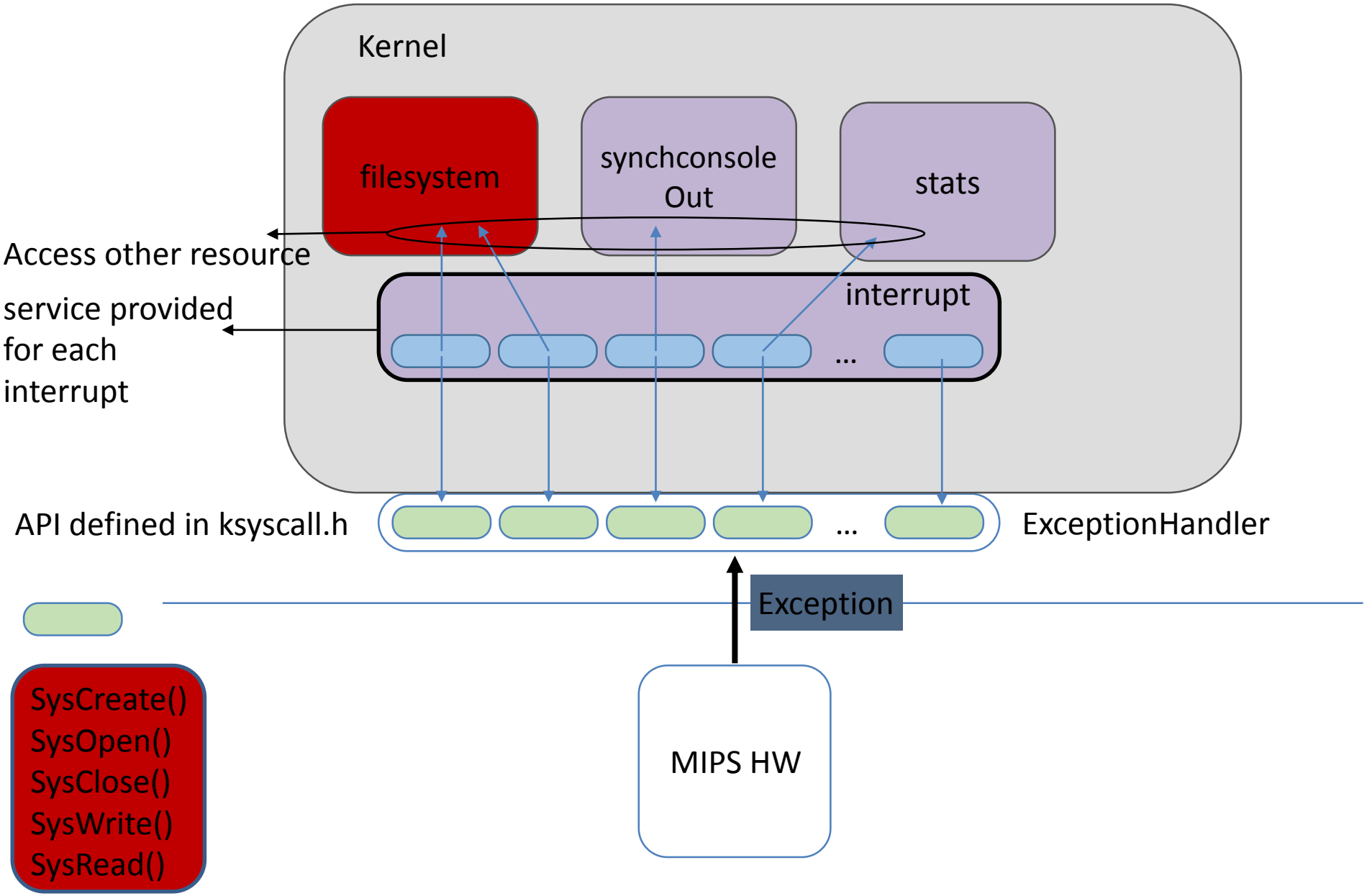
MaxFileSize = (NumDirect * SectorSize)
NumDirect = ((SectorSize - 2 * sizeof(int)) / sizeof(int)), where
SectorSize = 128; sizeof(int) = 4
so we got
NumDirect = (128- 2*4)/4 = 30 // 因為 numBytes and numSectors佔了兩個integer
MaxFileSize = 30 * 128 = 3840 bytes => 略小於4KB

Achievement:

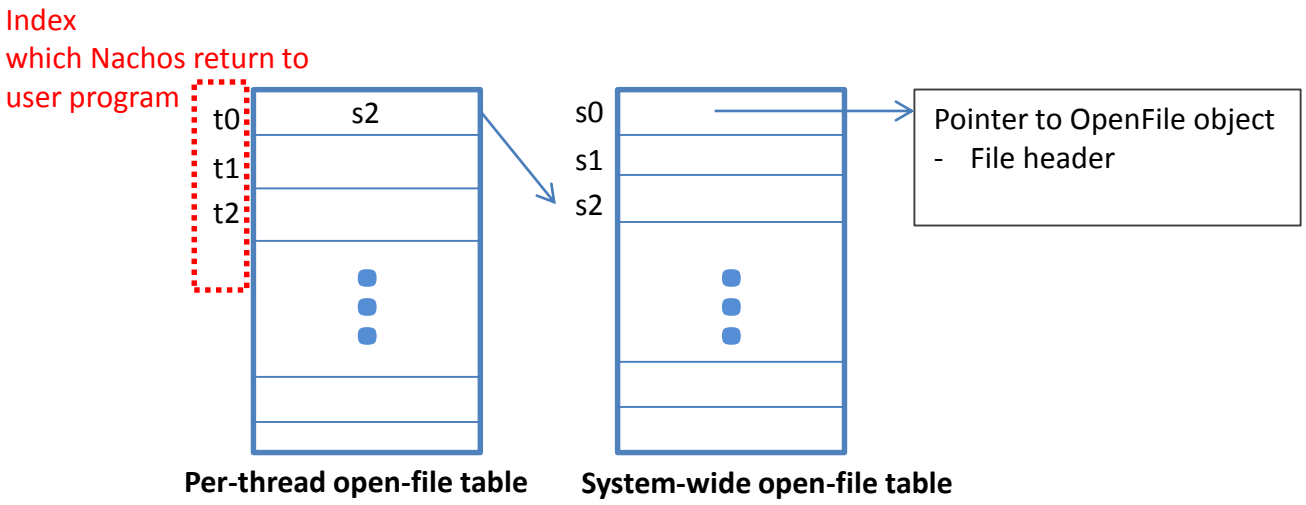
Part II a

Combine your MP1 file system call interface with NachOS FS

System Call Exception Handle Mechanism



1. Recall that we implemented Create(), Open(), Read(), Write() and Close() Systemcall as shown in the previous slide
2. For MP4, we use the real NachOS filesystem methods instead and the functions in exception.cc, ksyscall.h, kernel.cc, and interrupt.cc are the same as MP1
3. For MP4, we still need to implement openfile table mapping as shown below



Part II a : Open a File

```
int Kernel::Open(char *filename)
{
    OpenFile *opFile = fileSystem->Open(filename);

    if(opFile){
      int fd = opFile->GetFd();
      fileSystem->SetOpenFileTable(fd,opFile);

      return fd;
    }
    return -1;
}
```

```
OpenFile *
FileSystem::Open(char *name)
{
...
sector = directory->Find(name);
// TODO: allocate a new entry in system-wide table[done]
if (sector >= 0){

   openFile = IsOpenBefore(sector);
   if(openFile)
   {
      openFile->AddOpenCount(1);
   }
   else{
      openFile = new OpenFile(sector);  // name was four
      if(GetSysFd(&fd)){
         openFile->SetFd(fd);
         int tmp_count = openFile->AddOpenCount(1);
         DEBUG(dbgMp4, "Open file in FileSystem::Open,"<
      }else{
         delete openFile;
         openFile = NULL;
      }
   }
}
...
}
```

```
class FileSystem {
public:
...
private:
...
map<int, OpenFile*> sysOpFileTable;
...
}
```

When a file is being opened,

1. we have a system-wild open file table in Class FileSystem
2. we would try to find the sector in which the file header stored
3. After getting the sector, we check if this file is opened before
4. If yes, we just add the file's open count and return the address of file's openfile instance.
   The action of opening is finished.
5. If no, we create a OpenFile instance for the opened file
6. Find a free space in the system-wild open file table
7. keep the index(fd) of the free space in the file header
8. keep the address of the OpenFile instance in the system-wild open file table

Part II a : Open a File

```
case SC_Open:
    val = kernel->machine->ReadRegister(4);
    {
        char *filename = &(kernel->machine->mainMemory[val]);
        OpenFileId fd = (OpenFileId)SysOpen(filename);
        OpenFileId fd_t = -1;
        if (kernel->currentThread->GetAvlEntry(&fd_t)&&(fd_t!=-1))
            kernel->currentThread->SetOpFileTable(fd,fd_t);

        DEBUG(dbgMp4, "SC_Open: open file fd="<<fd<<",  User Space thread:"<<fd_t);

        kernel->machine->WriteRegister(2, fd_t);
    }
    kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));
    kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);
    kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg)+4);
    return;
    ASSERTNOTREACHED();
    break;
```

For per-thread open file table, the mapping steps are very silimiar to the ones for the system-wild opne file table:

1. find a avaliable space in the per-thread open file table
2. keep the system table index in the space we find in step 1

Part II a : Close a File

```
int
FileSystem::Close(int fd){

    OpenFile *opFile = GetOpenFileTable(fd);

    if(opFile->MinusOpenCount(1)==0){
        sysOpFileTable.erase(fd);
        delete opFile;
    }
    return 1;
}
```

1. When a thread would like to close a file, Filesystem decreases the open file count of the file
2. If count=0, it means no thread opens the file. Filesystem can remove it from system-wild open file table

Achievement:

Part II b

Enhance the FS to let it support up to 32KB file size

Bonus I

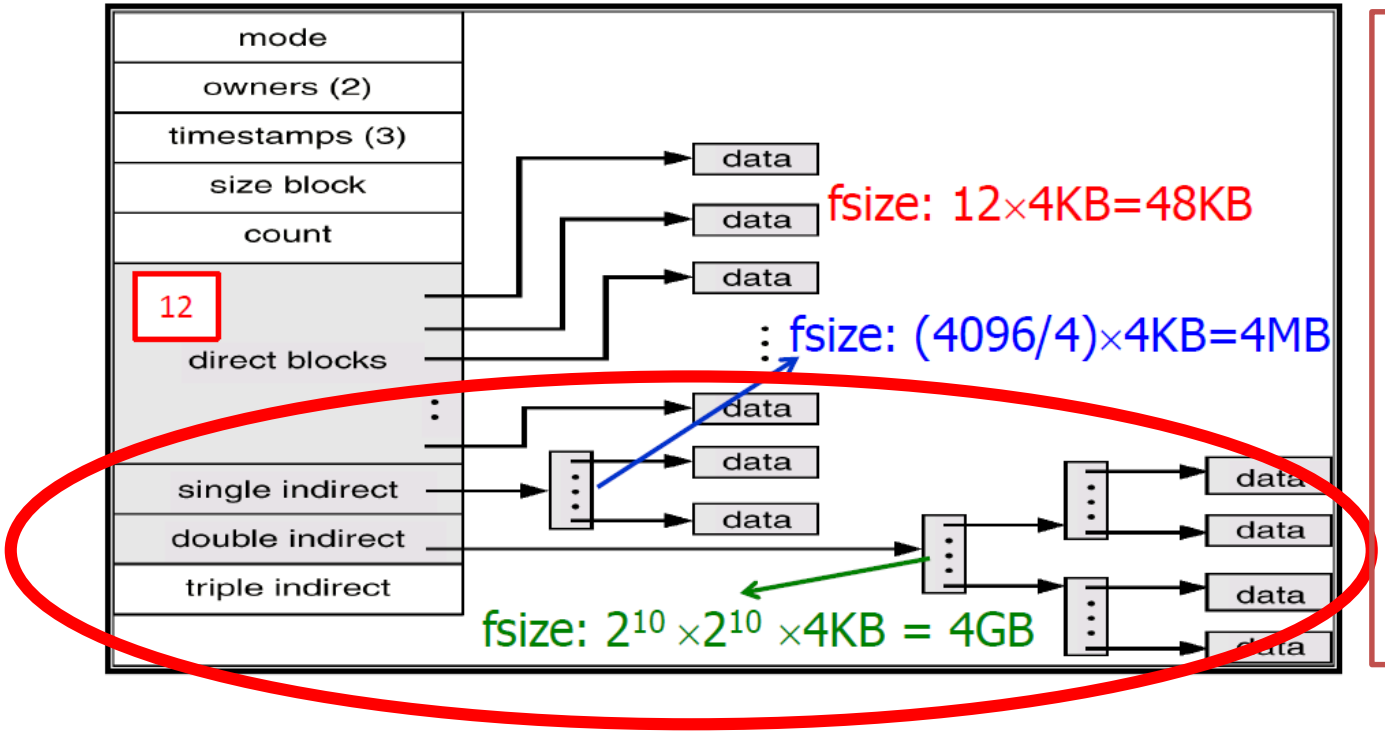
Enhance the Nachos to support even larger file size(64MB)

Bonus II

Multi-level header size

Part II b : Basic Design Concept

In order to achieve PartII_b assignment goal, we apply **Combined scheme in Indexed Allocation** for Disk allocation. It can support any file size if DISK size is big enough. The basic idea is very similar to the figure shown in the slide in ch 11

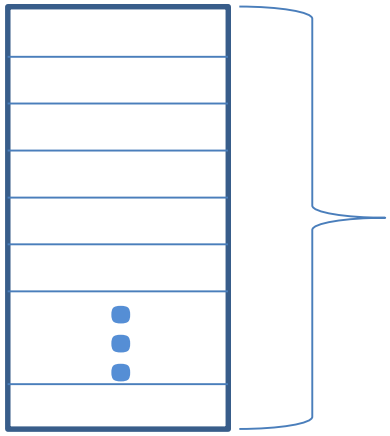


There are a few questions we should answer for this implementation:

1. How many sectors which a index sector can support at most?
2. How do we know the indirect layer the file needs?
3. How to support dynamic allocation on a known layer number?
4. Can we also achieve the bonusI&II goals ?

Let's answer the questions and have the implementation!

1. How many sectors which a index sector can support at most?



A index sector can store **32** sector number at most

```
#define NumInDirect  ((SectorSize)/sizeof(int))
//SectorSize = 128 bytes
class indirectTable {
  public:
    int dataSectors[NumInDirect];   // Disk sector

};
```

In filesys/filehdr.h

We define a new class **indirectTable** for a index sector object

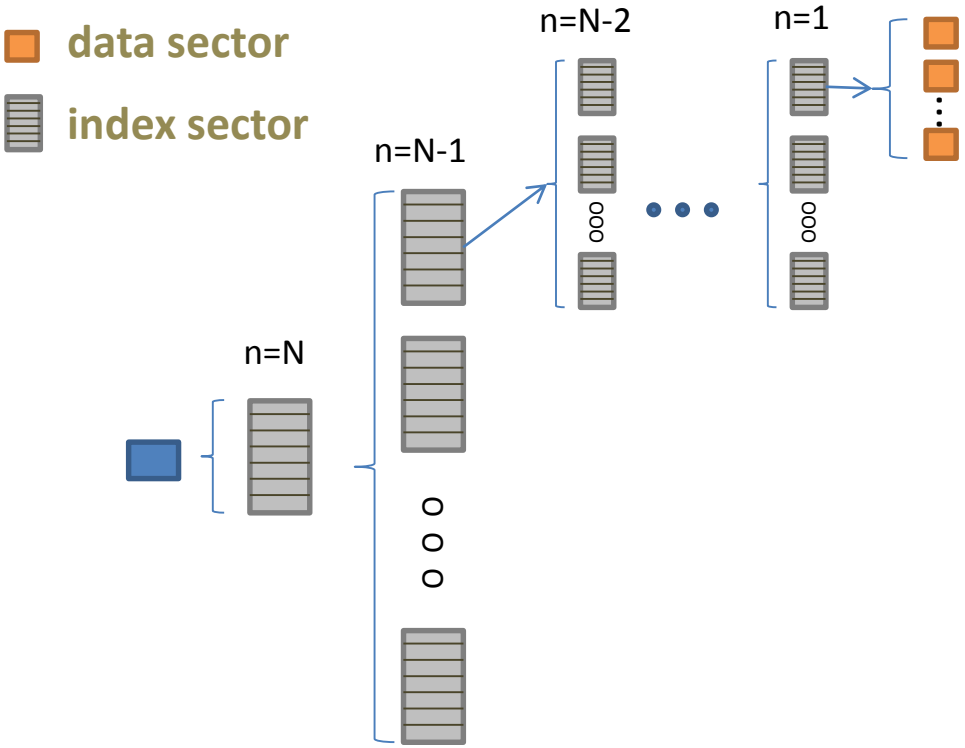
2. How do we know the indirect layer (N) the file needs?

We define the total number of sectors the file need = K,
then we can get the total number of layers N as shown below

$$N = \left\lceil \log_{32}(K) \right\rceil = \left\lceil \frac{\log K}{\log 32} \right\rceil$$

3. How to support dynamic allocation on a known layer number?

We recursively allocate free sectors for each index sector with decreasing n. When n is ZERO, the recursive allocator return. By applying the Answer #2&#3, we can achieve the goals of PartII_b、bonusI and bonusII, respectively.



4. Can we also achieve the bonusI&II goals ?

Yes. As explanation in Answer#3

Part II b : Allocation Implementation

When the filesystem create a new file, the disk allocation would perform the action as follows

```
bool
FileHeader::Allocate(PersistentBitmap *freeMap, int fileSize)
{
    numBytes        = fileSize;
    numSectors      = divRoundUp(fileSize, SectorSize);
    ...         //Get the number of sector the file needs
    indirectTable *indirTbl = new indirectTable;
    ...      //Create a new index sector for first indirect Layer

    numLevel = (int)(log10(numSectors)/log10(32))+1; //see how many indirect
     //Get the number of indirect layer the file needs
    if(numLevel>0)
        dataSectors[numLevel] = freeMap->FindAndSet(); //Find a free sector
    else                                               for the indirect
        return false;                                  sector

    AllocSector(freeMap, numLevel, indirTbl, &allocSecNum, numSectors);
     //start to allocate the sectors recursively
    kernel->synchDisk->WriteSector(dataSectors[numLevel], (char *)indirTbl);
     //Write back the first layer indirect sector to DISK
    ...
}
```

Part II b : Allocation Implementation

```
bool
FileHeader::AllocSector(PersistentBitmap *freeMap,
                        int n,
                        indirectTable *tbl,
                        int *allocSecNum,
                        int needSecNum)
{
    if(n<=0||((*allocSecNum)==needSecNum)) return FALSE;
    n--;  //for next level n, so it should be decreased

    indirectTable *indirTbl = new indirectTable;
    for(int i=0;(i<NumInDirect)&&((*allocSecNum)<needSecNum);i++){
        tbl->dataSectors[i]=freeMap->FindAndSet();
        memset(indirTbl, -1, sizeof(indirectTable));  // dummy operation to keep v
        if(AllocSector(freeMap,n,indirTbl,allocSecNum,needSecNum))
            kernel->synchDisk->WriteSector(tbl->dataSectors[i], (char *)indirTbl);
        else
            if((*allocSecNum)<needSecNum) (*allocSecNum)++;
    }
    delete indirTbl;
    return TRUE;
}
```

//The condition for stopping allocating free sectors

//the stop condition would be finishing allocating 32 sectors or the total data sectors the file needs

//get a new free sector and reset the indirect index sector

// keep going to next level and to see if the file needs more sectors (data or index)
// if it return yes, the next level is still a indirect index sector,
// if it return no, the next level is a data sector or no more data sectors

Part II b : DeAllocation Implementation

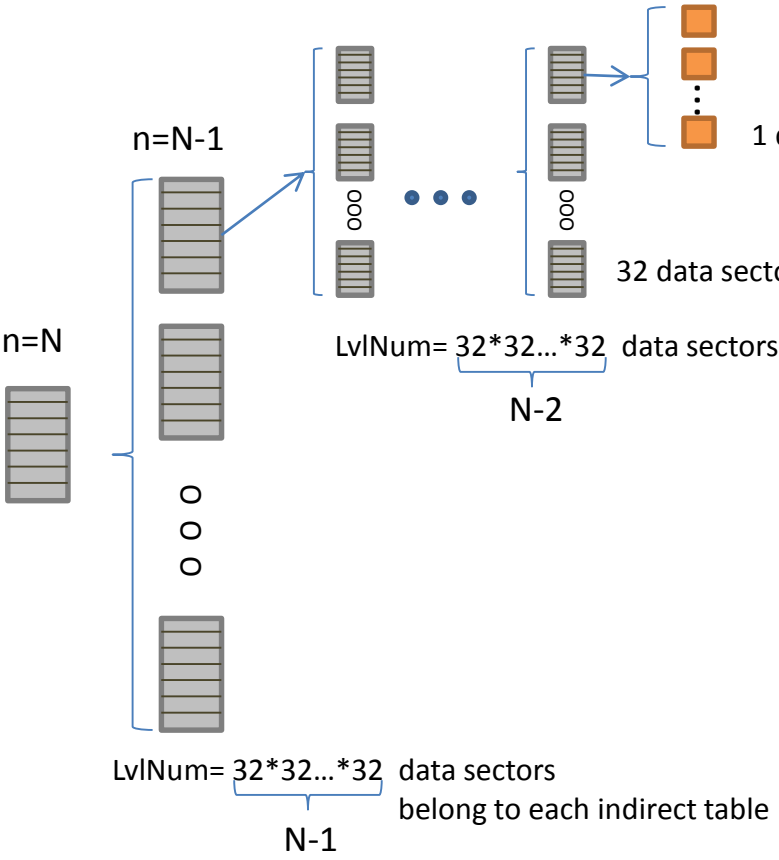
```
void
FileHeader::Deallocate(PersistentBitmap *freeMap)
{
    //if(numInDir>0){
    //int tmpNumSector = 0;
    int deallocSecNum = 0;
    indirectTable *indirTbl = new indirectTable;
    memset(indirTbl, -1, sizeof(indirectTable));  // dummy operation to keep va
      // create a new index sector for reading the first level sector allocation information
    kernel->synchDisk->ReadSector(dataSectors[numLevel], (char *)indirTbl);
      // reading the first level sector allocation information
    DeallocSector(freeMap, numLevel, indirTbl, &deallocSecNum, numSectors);
      // start to traverse each allocated sectors including index sectors and data sectors
    delete indirTbl;

}
```

Part II b : DeAllocation Implementation

```
bool
FileHeader::DeallocSector(PersistentBitmap *freeMap,
                          int n,
                          indirectTable *tbl,
                          int *deallocSecNum,
                          int needSecNum)
{
    if(n<=0||((*deallocSecNum)>=needSecNum)) return FALSE;
    n--;  //create a new indirect table for reading the allocating information for each sector number in tbl
    indirectTable *indirTbl = new indirectTable;

    for(int i=0;(i<NumInDirect)&&((*deallocSecNum)<needSecNum);i++)
    {
        memset(indirTbl, -1, sizeof(indirectTable));
        kernel->synchDisk->ReadSector(tbl->dataSectors[i], (char *)indirTbl);
        //reset the table and read the next sector content
        if( !DeallocSector(freeMap,n,indirTbl,deallocSecNum, needSecNum) ){
            if((*deallocSecNum)<needSecNum)(*deallocSecNum)++;
          // go to the next level recursively
        } // if it return false , the sector is a data sector or the whole de-allocation work is finished
        freeMap->Clear((int) tbl->dataSectors[i]);
         // to return the sector to free sector pool
    }

    delete indirTbl;
    return TRUE;
}
```

Part II b : DeAllocation Implementation

In Class OpenFile, we can see hdr->ByteToSector() in both WriteAt() and ReadAt() methods. It is responsible for seachring the sector which the target byte belong to in a file. We should also handle this translation carefully in our implementation.




```
int
GetSector(int secNum,int LvlNum, indirectTable *tbl)
{
    int thesector = secNum/LvlNum;
    int nextlevl  = secNum%LvlNum;
    ...
}
```

1. In each level, we try to find the indirect table the target sector belong to
2. The indirect table in each level can be found by the sector offset in that level divided by the number of sector belong to a indirect table
3. The reminder(nextlevl) would be the sector offset for the next level

Part II b : Get the target Sector

```
int
FileHeader::ByteToSector(int offset)
{

  int  sector = -1;

  indirectTable *indirTbl = new indirectTable;
  memset(indirTbl, -1, sizeof(indirectTable));  // dummy operation to keep val
  kernel->synchDisk->ReadSector(dataSectors[numLevel], (char *)indirTbl);

  sector = GetSector(offset/SectorSize,pow(NumInDirect,numLevel-1),indirTbl);

  delete indirTbl;
  return sector;
}
```

1. In ByteToSector(), we read the first level indirect table
2. Give the initial arguments for GetSector() and hope it returns the sector number the byte offset belong to

Part II b : Get the target Sector

```
int
GetSector(int secNum,int LvlNum, indirectTable *tbl)
{
    int thesector = secNum/LvlNum;
    int nextlevl  = secNum%LvlNum;
    int reSec     = -1;

    if(LvlNum==1){
       return  tbl->dataSectors[thesector];
        // only one descendant sector belong to , it must be a data sector
    }else{
        indirectTable *indirTbl = new indirectTable;
        kernel->synchDisk->ReadSector(tbl->dataSectors[thesector], (char *)indirTbl);
        reSec = GetSector(nextlevl,LvlNum/NumInDirect,indirTbl);
        //include more than one descendant sector, it must be a indirect table
        //keep reading the next level information
        delete indirTbl;
        return reSec;
    }
}
```

Bonus I :

Increase DISK size for larger file size

```
const int SectorSize = 128;     // number of bytes per disk sector
const int SectorsPerTrack  = 32;    // number of sectors per disk track
const int NumTracks = 32;       // number of tracks per disk
const int NumPlate = 512;       // number of tracks per disk
const int TracksPerPlate = 64;
const int NumSectors = (SectorsPerTrack * TracksPerPlate * NumPlate);
```

512*64*32*128=2^27 = 128MB

```
int
Bitmap::FindAndSet()
{
    for (int i = 0; i < numWords; i++) {
      if(~map[i]){
        unsigned int aWord = ~map[i];
        for(int j=0; j<BitsInWord; j++){
          if((1<<j)&aWord){

            int offset = i*BitsInWord+j;
            Mark(offset);
            return offset;
          }
        }
      }
    }
    return -1;
}
```

Improve the free sector searching algorithm

Bonus I :

We passed the larger file size test

```
-rw-rw-r--  1 stanle stanle     10000  1月 13 21:15 num_1000.txt
-rw-rw-r--  1 stanle stanle      1000  1月 13 21:15 num_100.txt
-rw-rw-r--  1 stanle stanle  67738890  1月 16 18:38 num_64M.txt
-rw-rw-r--  1 stanle stanle       457  1月 13 21:15 script
-rw-rw-r--  1 stanle stanle      1394  1月 13 21:15 segments.c
-rw-rw-r--  1 stanle stanle       451  1月 13 21:15 shell.c
-rw-rw-r--  1 stanle stanle      1315  1月 13 21:15 sort.c
```

Achievement:

Part III

Implement the subdirectory structure

Support up to 64 files/subdirectories per directory

Bonus III

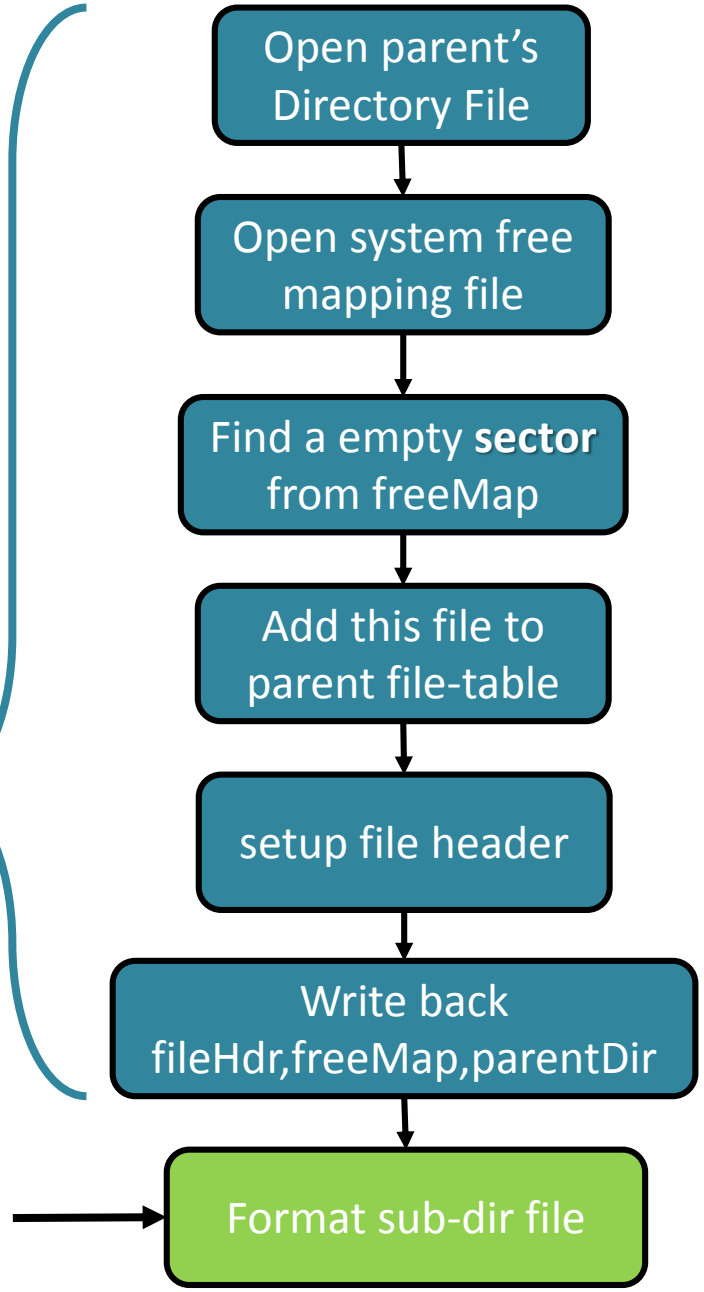
Recursive Operations on Directories

=> Support recursive remove of a directory or file

Create a Directory verse Create a File

可以發現，
Create一個File跟Create一個Directory過程大部份相同，
因為Directory的資訊都存在file裡面，
可以將Directory視作File來Create。

唯一不同的是，Directory file需要額外format



How to Format Root Directory

```
$ ../build.linux/nachos -f
```

In kernel.cc

Capture the '-f' argument

turn formatFlag on

And initialize filesystem with this flag

Clean :

No fetch information from files,

But just write back the initial states

Back to files.

In filesys.cc

FileSystem capture the flag, and format the disk

By clean directory file and freeMapping file.

Subdirectory

假設輸入的path為：/home/user/new_file

字串切割（implement in filesys.cc FileSystem::PreprocessPath）

會切割成：{“/”，“home/”，“user/”，“new_file”}

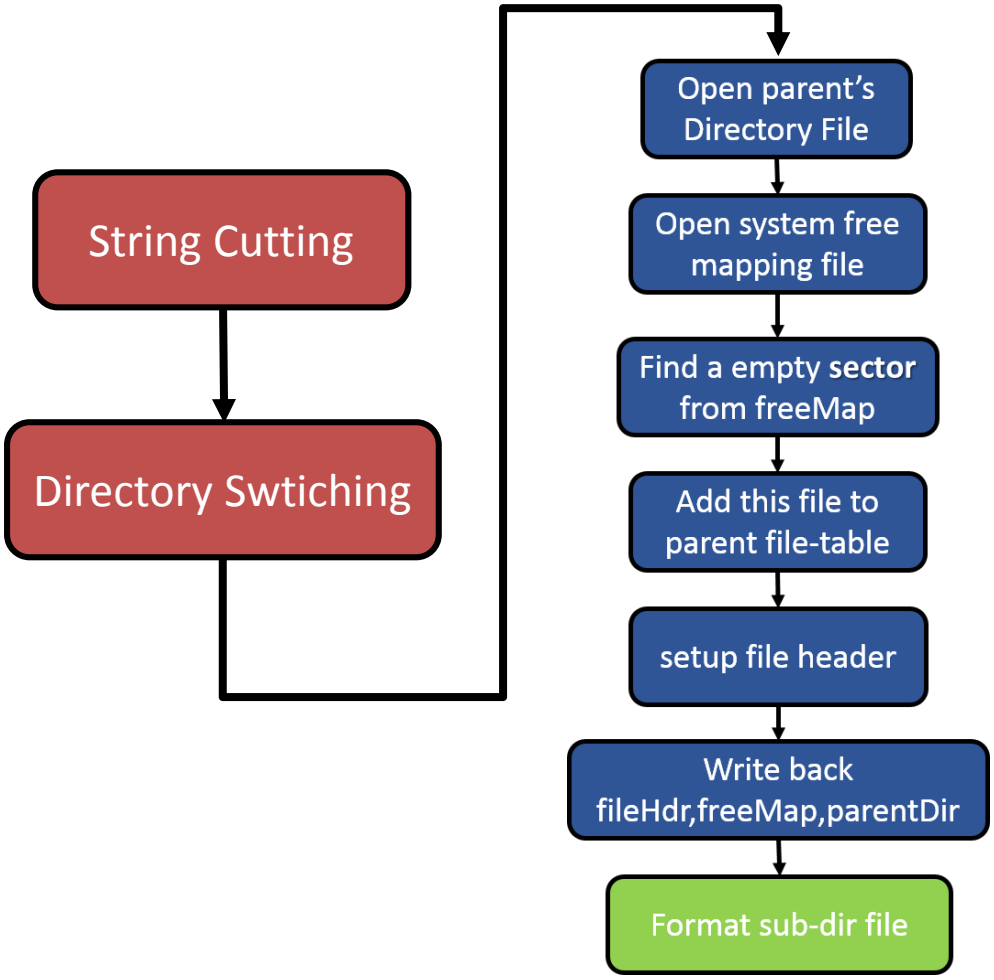
目錄轉換（implement in filesys.cc FileSystem::GoDirectory）

原本default的directory是在root下

這裡會把current directory切換到要access的file所在的位置

以這個列子就是切換到user/，而同時也換把傳入的name這個variable值改成new_file

Create File/dir under Subdirectory



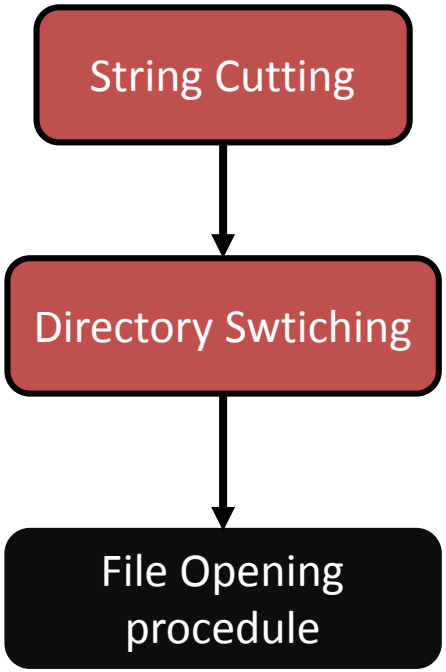
我們把file跟directory的create寫在一起(in filesys.cc FileSystem::Create)，因為他們的程序大部分相同。

並且更改作多可以儲存的file/subdirectory 個數:

#define NumDirEntries 64

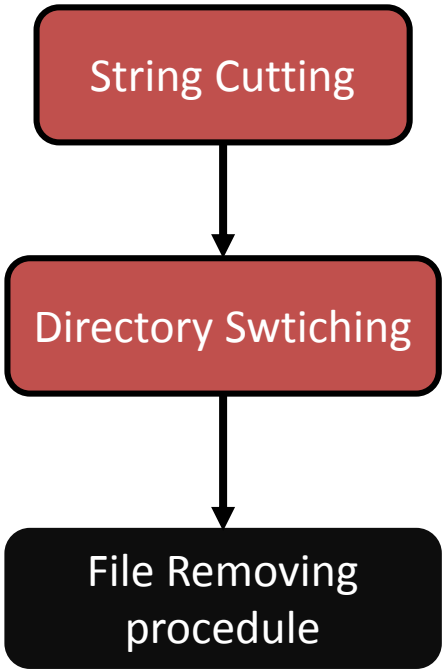
Open file under Subdirectory

Open只適用在File上，所以除了subdirectory的前置處理，其他都跟原本一樣。



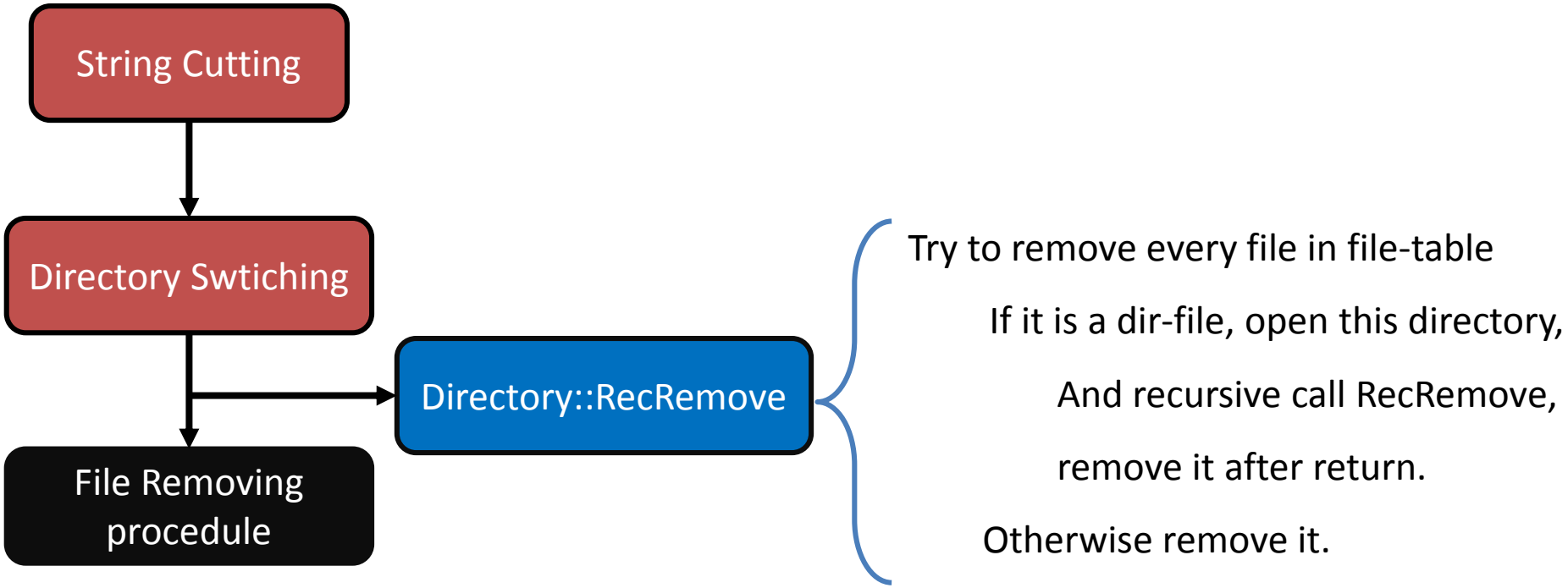
Remove file under Subdirectory

Remove只適用在File上，所以除了subdirectory的前置處理，其他都跟原本一樣。



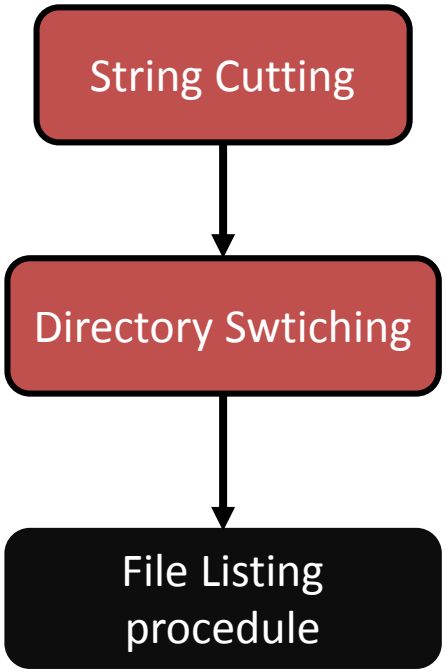
Recursive Remove under Subdirectory

Implement in directory.cc Directory::RecRemove , which is a recursive function.



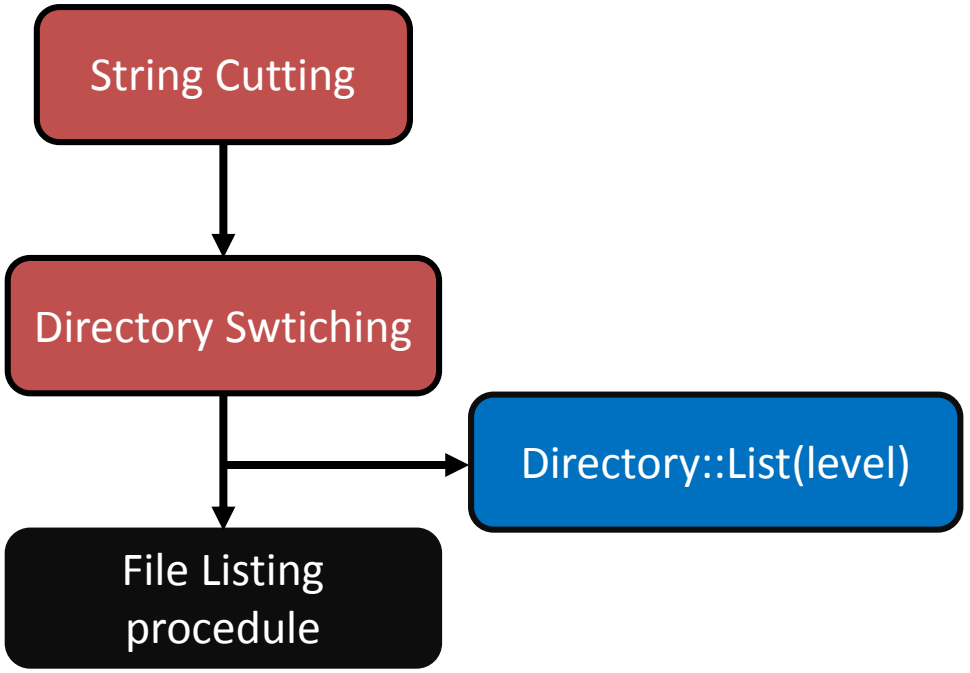
List files under Subdirectory

除了subdirectory的前置處理，其他都跟原本一樣。



Recursive List under Subdirectory

Implement in directory.cc Directory::List(level) , which is a recursive function.



Try to list every file in file-table

If it is a dir-file, list this dir's name

open this directory,

And recursive call list(level+1),

Otherwise list it.

工作分配
Trace code
協同合作

Coding:
Part II
蔡玉倫主寫
黃心佑補充
PartIII
黃心佑主寫
BonusI&II
蔡玉倫主寫
BonusIII
黃心佑主寫

Report:
Part I
黃心佑主寫
蔡玉倫補充
Part II
蔡玉倫主寫
Part III
黃心佑主寫