+--------------------------------------+

|---------------- CS 130 ---------------|

|--PROJECT 4: FILE SYSTEMS--|

|-------DESIGN DOCUMENT-------|

+--------------------------------------+

GROUP

> Fill in the names and email addresses of your group members.

Yifan Zhu zhuyf1@shanghaitech.edu.cn
Zongze Li lizz@shanghaitech.edu.cn

PRELIMINARIES

> If you have any preliminary comments on your submission, notes for the TAs, or extra credit, please give them here.

> Please cite any offline or online sources you consulted while preparing your submission, other than the Pintos documentation, course text, lecture notes, and course staff.

INDEXED AND EXTENSIBLE FILES

DATA STRUCTURES

> A1: Copy here the declaration of each new or changed `struct' or` struct' member, global or static variable, `typedef', or
> enumeration. Identify the purpose of each in 25 words or less.

In `inode.h` and `inode.c` :

```
/* Define some variable about inode */
#define TABLE_L 128
#define DIR_L 8
#define INDIR_L 32

#define DIR_SECTOR (DIR_L * BLOCK_SECTOR_SIZE)
#define INDIR_SECTOR (INDIR_L * BLOCK_SECTOR_SIZE)
#define TABLE_SECTOR (TABLE_L * BLOCK_SECTOR_SIZE)
#define MAX_SIZE (DIR_L * BLOCK_SECTOR_SIZE + TABLE_L * INDIR_L * BLOCK_SECTOR_SIZE)

/* On-disk inode.
   Must be exactly BLOCK_SECTOR_SIZE bytes long. */
struct inode_disk
  {
    int i_type;                         /* 0 represent directory, 1 represent common file */
    int dir_b[DIR_L];                   /* Direct point array */
    int indir_b[INDIR_L];               /* Indirect point array */
    off_t length;                       /* File size in bytes. */
    unsigned magic;                     /* Magic number. */
    uint32_t unused[85];               /* Not used. */
  };
```

> A2: What is the maximum size of a file supported by your inode structure? Show your work.

We set 8 direct blocks and 32 indirect blocks in an inode, so we can have 8 * 512B + 32 * 128 * 512B = 2 MB maximum size of a file.

SYNCHRONIZATION

> A3: Explain how your code avoids a race if two processes attempt to extend a file at the same time.

We just use the lock to prevent two processes to access extend a file at the same time.(It needs to do this by syscall, that's the synchronization position.)

> A4: Suppose processes A and B both have file F open, both positioned at end-of-file. If A reads and B writes F at the same time, A may read all, part, or none of what B writes. However, A may not read data other than what B writes, e.g. if B writes nonzero data, A is not allowed to see all zeros. Explain how your code avoids this race.

Just same as A3, if a process want to read or write, it need to syscall, and so that it will be synchronized by lock at that time, so there is no race condition.

> A5: Explain how your synchronization design provides "fairness".
> File access is "fair" if readers cannot indefinitely block writers
> or vice versa. That is, many processes reading from a file cannot
> prevent forever another process from writing the file, and many
> processes writing to a file cannot prevent another process forever
> from reading the file.

We just excuate fireness by processes's priority themselves, if a reader or writer have a higher priority, it will get resource at first, or they have same priority, they will
have the same possibility to get resource firstly.

RATIONALE

> A6: Is your inode structure a multilevel index? If so, why did you
> choose this particular combination of direct, indirect, and doubly
> indirect blocks? If not, why did you choose an alternative inode
> structure, and what advantages and disadvantages does your
> structure have, compared to a multilevel index?

Yes, our inode structure is a multilevel index(two level), 8 direct blocks and 32 indirect blocks.
I choose this value because i consulted an elder who pass all pintos tests, and i learn that for pintos, 2MB file is enough for it's manipulation
and 8 direct blocks and cut down the cost about file access.

SUBDIRECTORIES

DATA STRUCTURES

> B1: Copy here the declaration of each new or changed `struct' or` struct' member, global or
> static variable, `typedef', or
> enumeration. Identify the purpose of each in 25 words or less.

```
struct thread
{
  ......

  /* The directory you are currently in */
  struct dir *pwd;

  ......
};
```

The pwd in thread tells us which directory we are currently in.

```
struct ofile
{
  ......

  struct dir* dir_pointer;

  ......
};
```

Represents directory pointer.

ALGORITHMS

> B2: Describe your code for traversing a user-specified path. How
> do traversals of absolute and relative paths differ?

We first split the given path into two parts: a directory and a name, which we do by calling path_divide (). In this function, we first check if the path starts with '/', if it does, we start from the root directory, otherwise we start from the current directory, we traverse the path according to the '/' in the path. While traversing relative and absolute paths is similar, we determine different traversal starts based on whether the path begins with '/' or not.

SYNCHRONIZATION

> B4: How do you prevent races on directory entries? For example,
> only one of two simultaneous attempts to remove a single file
> should succeed, as should only one of two simultaneous attempts to
> create a file with the same name, and so on.

As with the other system calls, locking is used during the execution of the system call to ensure that the operations are atomic and thus avoid races.

> B5: Does your implementation allow a directory to be removed if it
> is open by a process or if it is in use as a process's current
> working directory? If so, what happens to that process's future
> file system operations? If not, how do you prevent it?

Our implementation does not allow this. In the structure of the inode we have a variable open_cnt that keeps track of how many processes are currently opening the file and does not allow to remove if open_cnt is greater than 1.

RATIONALE

> B6: Explain why you chose to represent the current directory of a
> process the way you did.

We add pwd to the thread structure to represent the current directory, which is very consistent with the general idea that each thread has a current directory, and the directory is opened and closed with the thread, which also facilitates our debug process to some degree.

BUFFER CACHE

DATA STRUCTURES

> C1: Copy here the declaration of each new or changed `struct' or` struct' member, global or static variable, `typedef', or
> enumeration. Identify the purpose of each in 25 words or less.

In `cache.h` and `cache.c` :

```
/* For read ahead function */
struct read_ahead
{
    int id;
    struct list_elem elem;
    struct list read_ahead_list;
    struct lock read_ahead_lock;
};

struct cache_entry
{
    /* Block representation */
    char buffer[BLOCK_SECTOR_SIZE];
    /* Dirty bit*/
    bool dirt;
    /* Assess whether touched this entry */
    bool touch;
    /* Second chance bit */
    int second_chance;
    /* Access lock */
    struct lock ele_lock;
    /* Entry size */
    int size;
    /* Record entry_id in cache */
    int id;
};

/* Clock hand */
int clock_hand;

/* Lock for cache */
static struct lock cache_lock;

/* Cache array */
static struct cache_entry cache[64];
```

ALGORITHMS

> C2: Describe how your cache replacement algorithm chooses a cache block to evict.

We use the clock algorithm, we have aa global clock hand `clock_hand` and give each block a `second_chance` , when chance equal to 0, it will be evicted.

```
/* Clock hand */
int clock_hand;

/* Update hand */
int update_hand()
{
    int ans = clock_hand;
    clock_hand = (clock_hand + 1) % 64;
    return ans;
}

/* Get free entry in cache */
int c_fetch()
{
    int ans;
    for(;;)
    {
        if(cache[clock_hand].second_chance == 0)
        {
            if(cache[clock_hand].dirt){
            block_write(fs_device,cache[clock_hand].id,cache[clock_hand].buffer);
            }
            ans = update_hand();
            break;
        }
        else
        {
            cache[clock_hand].second_chance = 0;
            update_hand();
        }
    }
    return ans;
}
```

> C3: Describe your implementation of write-behind.

We just create another thread to charge it, let li flush back every 1 second.

```
/* Thread function */
void flushall()
{
    for(;;)
    {
        /* Flush with period 1s */
        timer_msleep(1000);
        flush_all();
    }
}
```

> C4: Describe your implementation of read-ahead.

Same like write_head, we just create a new thread charge it, when condition satisfied, it will check the `read_ahead_list` , and use our `c_fetch()` function, find corresponding next block according to `id` , so that read both of them to cache.

SYNCHRONIZATION

> C5: When one process is actively reading or writing data in a buffer cache block, how are other processes prevented from evicting that block?

We use a global lock `static struct lock cache_lock` , if one process is actively reading or writing data, it will handle this lock and prevent
other process get in at the same time also will not evict that block.

> C6: During the eviction of a block from the cache, how are other processes prevented from attempting to access the block?

Just same as C5, we use `c_fetch()` to get a free block which called by `c_read()` or `c_write()` , and `static struct lock cache_lock` can sychronize all processes.

RATIONALE

> C7: Describe a file workload likely to benefit from buffer caching, and workloads likely to benefit from read-ahead and write-behind.

For buffer caching : Massive repeated access to the same file.
For read-ahead : Large sequential access to files.
For write-ahead : Machine is not very stable.

SURVEY QUESTIONS

Answering these questions is optional, but it will help us improve the course in future quarters. Feel free to tell us anything you want--these questions are just to spur your thoughts. You may also choose to respond anonymously in the course evaluations at the end of the quarter.

> In your opinion, was this assignment, or any one of the three problems in it, too easy or too hard? Did it take too long or too little time?

> Did you find that working on a particular part of the assignment gave you greater insight into some aspect of OS design?

> Is there some particular fact or hint we should give students in future quarters to help them solve the problems? Conversely, did you find any of our guidance to be misleading?

> Do you have any suggestions for the TAs to more effectively assist students in future quarters?

> Any other comments?