```
        +-------------------------+
        |         CS 140          |
        | PROJECT 4: FILE SYSTEMS |
        |     DESIGN DOCUMENT     |
        +-------------------------+
```

---- GROUP ----

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

Chuyi Zhao zhaochy1@shanghaitech.edu.cn
Bowen Xu xubw@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.

- *Pintos Guide*

  ```
  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 `struct inode_disk`:

```c
struct inode_disk
  {
    off_t length;                       /* File size in bytes. */
    block_sector_t direct[DIRECT_BLOCKS_COUNT];
    block_sector_t indirect;
    block_sector_t double_indirect;
    unsigned is_dir;
    unsigned magic;                     /* Magic number. */
    uint32_t unused[113];               /* Not used. */
  };
```

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

In one `inode`, there are 123 direct block + 1 indirect block +
1 doublely-indirect block. Given each indirect block linked to
128 sectors, one file may get at most 16635 sectors of storage.
The max size is 16635*512 = about 8MB.

---- SYNCHRONIZATION ----

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

The process should acquire the lock in `inode` before attempting to extend a file.

> 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.

A will be blocked by the lock until B finishes writing new data into the extended part of the file.

> 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.

A queue may be used to schedule the processs attempting to access or modify a file into a first come first serve order. However, we did not have it implemented.

---- 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?

We implemented the `inode` to be a multilevel index. There are 123 direct blocks, 1 indirect block and 1 doublely-indirect block in one `inode`. Such combination uses up the 512 byte space of a disk sector, and can handle files with its size up to 8MB while having sufficient direct blocks for small files.

```
             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.

In `struct inode`, `bool is_dir` is added. In `struct thread`, `struct dir* path` is added.

---- ALGORITHMS ----

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

Absolute paths start with a `\` while relative paths don't. If a relataive path is provided, the traversal starts from the `path` recorded in `struct thread`. Otherwise, the traversal starts from the root.

---- 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.

A process acquires a lock before it removes or creates a file, preventing possible 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?

Removing an opened directory or the working directory is not allowed. It simply does nothing if being attempted.

---- RATIONALE ----

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

It's quite concise to store the working directory in the `struct thread`.

```
             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.

We tried to implement the buffer cache but it eventually didn't works.

---- ALGORITHMS ----

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

We attempted to randomly choose a block not in use when eviction is needed.

> > C3: Describe your implementation of write-behind.

We attempted to write the blocks back to the disk when the file is closed or eviction happens.

> > C4: Describe your implementation of read-ahead.

We attempted to read 8 sectors at once since the page size is 4KB.

---- 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?

Since we didn't attempt to implement a LRU eviction, the process may
not be prevented from eviting the block.

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

Accessing to the blocks being evicted will be blocked until the eviction
process finishes.

---- RATIONALE ----

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

The workloads attempting to read or write a file byte by byte may benifit
from buffer caching. Read-ahead and write-behind can reduce the actual
disk operations in such workloads.

```
			   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?