B03902001 駱定暄

1. There is a client updating changes to the file on the server side. In the meanwhile, the second client also wants to modify the same file on the server. It should follow the 4 steps below:
   (1)The first coming request will be accepted and the second coming request will be blocked. However, there should be an index to record that this file needs to be merged and which connection file-descriptor we can get the header and data we want.
   (2)Get the file meta and data from the second client's conn_fd.
   (3)Create a temporary file to store the file content from the second client.
   (4)Use file_merger to merge the temporary file and the original file into a new file. Then, name the new file after the original one and remove both old version and the temporary file.
2.
   (1)
   To implement csiebox in multi-process, I still have to use select() to listen to the connection-file-descriptors. When a new event occurs, instead of directly calling handle_request() in the previous homework, I will put the event information into a structure named "Job" in server_run() and then call fork() to create a new process. Hence, the child process will receive the new task from the conn_fd. The chile process will call exit(0) and terminate after it finish the job.

```
struct Job{
    csiebox_server *server;
    int conn_fd;
    fd_set *master;
};
```

   (2)
   Memory: Processes consumes more memory than threads, because it have to copy the whole parent process into a new space. On the other hand, creating a new thread only requires a new stack memory space. In addition, processes cannot share resource very well because of the distinct process memory.
   Time: Creating a new process takes more time than creating a new thread, because parent process have to copy data into child process. Furthermore, process context switch cost is very high and switching among threads do not need to do context switch.
   Reliability: a buggy thread may crash all the other threads in the process.