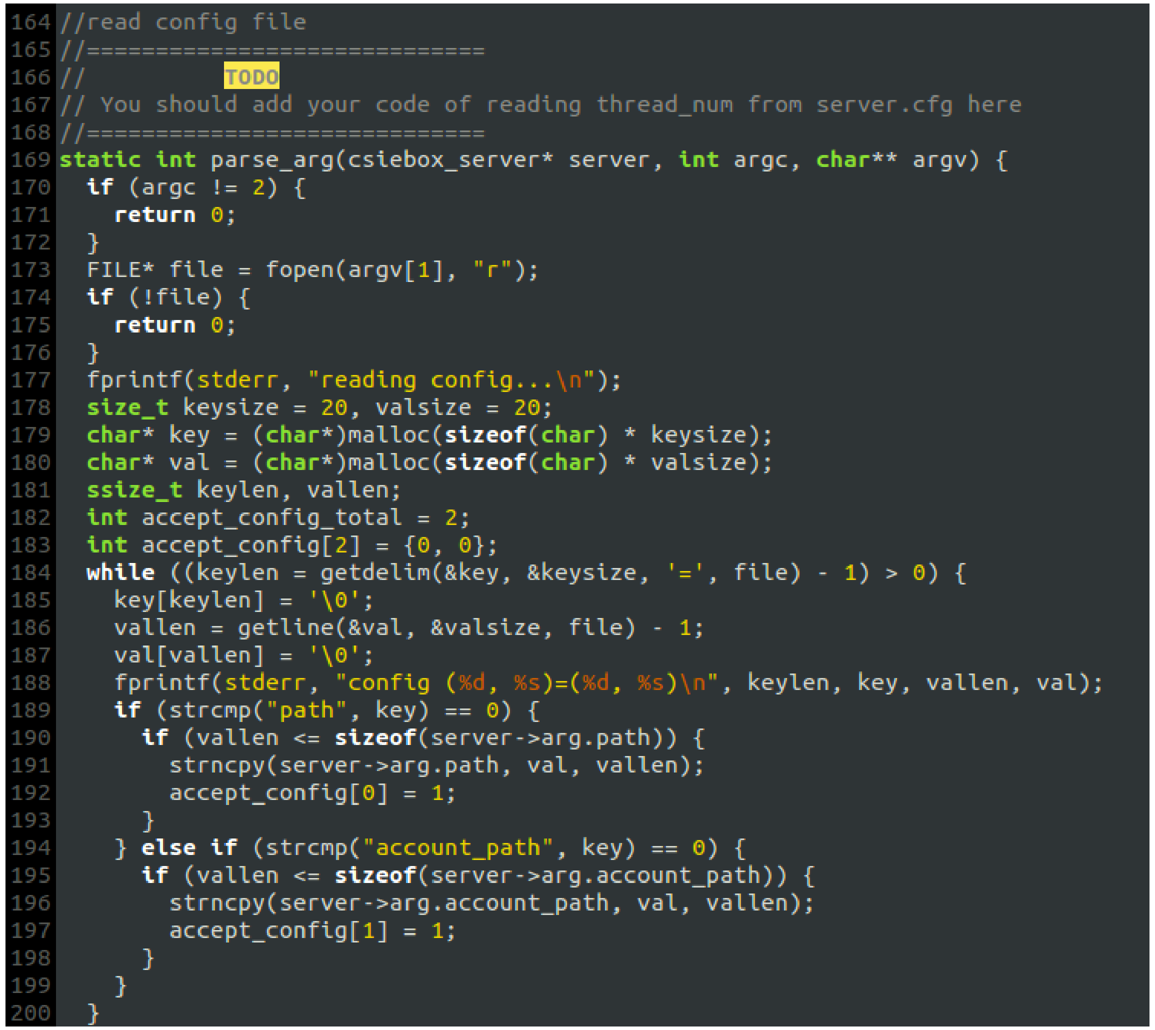
MP4 Tutorial

Sample code of MP3

- You can download it from github
- Some important files:
  - src/csiebox_server.c: contain most of the server code
  - src/server_function.c: contain functions for synchronization in server side
  - src/csiebox_client.c: contain most of the client code
  - src/client_function.c: contain functions for synchronization in client side

Where you should add your code

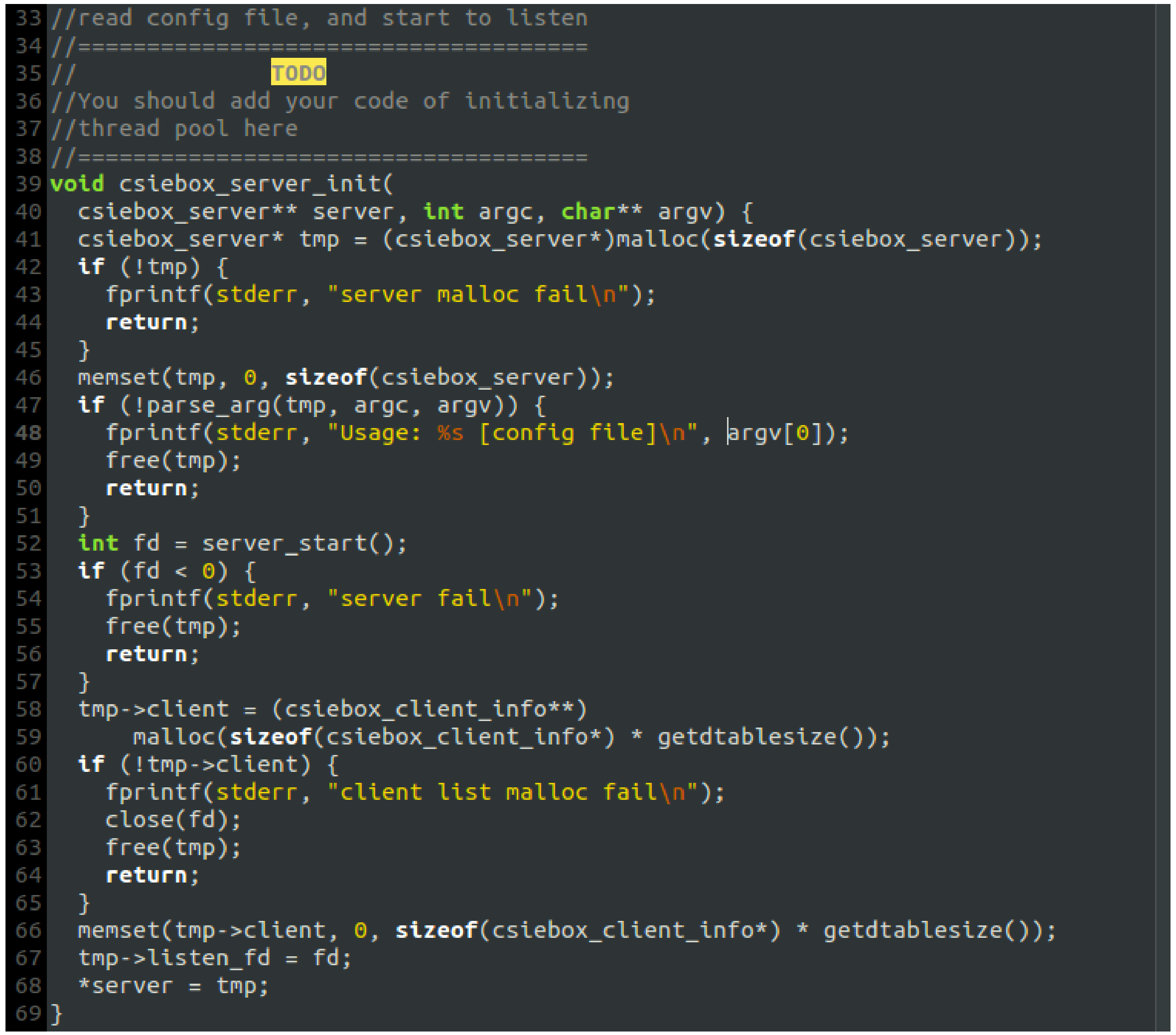
- parse_arg() in src/csiebox_server.c
- You should add your code of reading thread_num from server.cfg
- You might also need to add a member for thread_num in struct csiebox_server in include/csiebox_server.h



```c
//read config file
//============================
//          TODO
// You should add your code of reading thread_num from server.cfg here
//============================
static int parse_arg(csiebox_server* server, int argc, char** argv) {
  if (argc != 2) {
    return 0;
  }
  FILE* file = fopen(argv[1], "r");
  if (!file) {
    return 0;
  }
  fprintf(stderr, "reading config...\n");
  size_t keysize = 20, valsize = 20;
  char* key = (char*)malloc(sizeof(char) * keysize);
  char* val = (char*)malloc(sizeof(char) * valsize);
  ssize_t keylen, vallen;
  int accept_config_total = 2;
  int accept_config[2] = {0, 0};
  while ((keylen = getdelim(&key, &keysize, '=', file) - 1) > 0) {
    key[keylen] = '\0';
    vallen = getline(&val, &valsize, file) - 1;
    val[vallen] = '\0';
    fprintf(stderr, "config (%d, %s)=(%d, %s)\n", keylen, key, vallen, val);
    if (strcmp("path", key) == 0) {
      if (vallen <= sizeof(server->arg.path)) {
        strncpy(server->arg.path, val, vallen);
        accept_config[0] = 1;
      }
    } else if (strcmp("account_path", key) == 0) {
      if (vallen <= sizeof(server->arg.account_path)) {
        strncpy(server->arg.account_path, val, vallen);
        accept_config[1] = 1;
      }
    }
  }
```

Where you should add your code

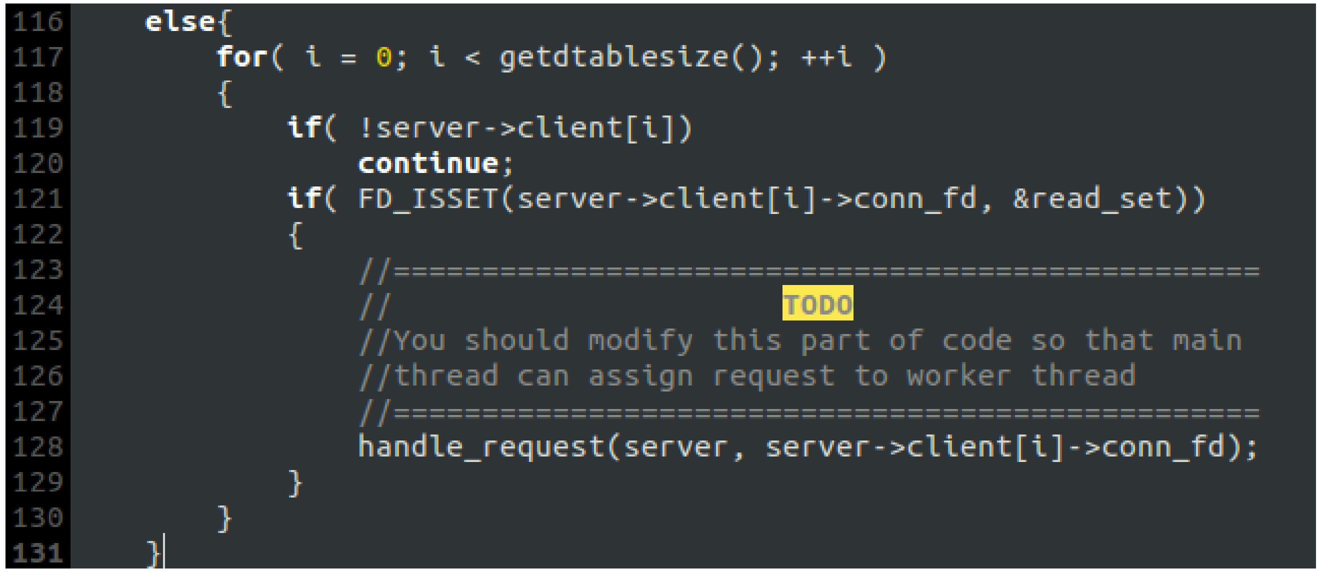
- csiebox_server_init() in src/csiebox_server.c
- You should add your code of initializing thread pool here
- You might also need to add some member for thread in struct csiebox_server in include/csiebox_server.h



```
//read config file, and start to listen
//=====================================
//                   TODO
//You should add your code of initializing
//thread pool here
//=====================================
void csiebox_server_init(
  csiebox_server** server, int argc, char** argv) {
  csiebox_server* tmp = (csiebox_server*)malloc(sizeof(csiebox_server));
  if (!tmp) {
    fprintf(stderr, "server malloc fail\n");
    return;
  }
  memset(tmp, 0, sizeof(csiebox_server));
  if (!parse_arg(tmp, argc, argv)) {
    fprintf(stderr, "Usage: %s [config file]\n", argv[0]);
    free(tmp);
    return;
  }
  int fd = server_start();
  if (fd < 0) {
    fprintf(stderr, "server fail\n");
    free(tmp);
    return;
  }
  tmp->client = (csiebox_client_info**)
      malloc(sizeof(csiebox_client_info*) * getdtablesize());
  if (!tmp->client) {
    fprintf(stderr, "client list malloc fail\n");
    close(fd);
    free(tmp);
    return;
  }
  memset(tmp->client, 0, sizeof(csiebox_client_info*) * getdtablesize());
  tmp->listen_fd = fd;
  *server = tmp;
}
```

Where you should add your code

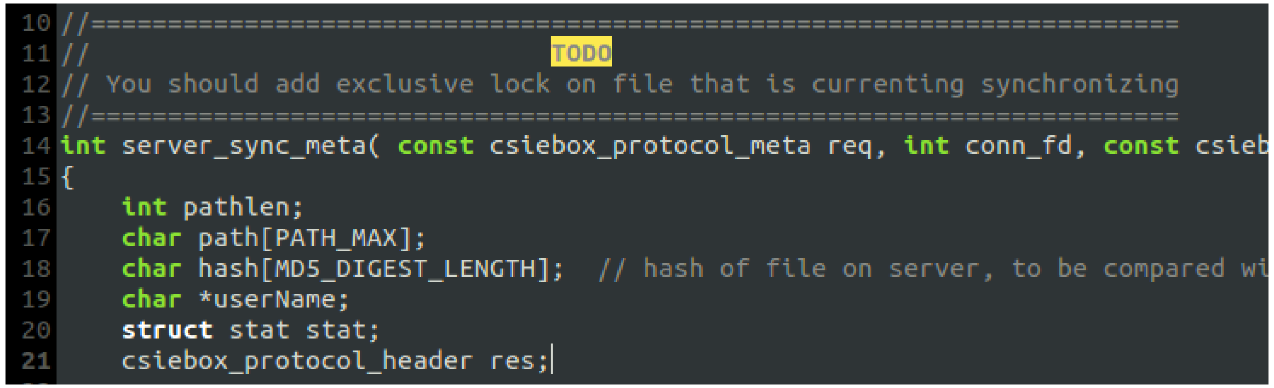
- csiebox_server_run() in src/csiebox_server.c
- You should modify how server handle request from client, so that the main thread can assign request to worker threads.
- Feel free to modify the structure of sample code to fit your need.



```
    else{
        for( i = 0; i < getdtablesize(); ++i )
        {
            if( !server->client[i])
                continue;
            if( FD_ISSET(server->client[i]->conn_fd, &read_set))
            {
                //==================================================
                //                        TODO
                //You should modify this part of code so that main
                //thread can assign request to worker thread
                //==================================================
                handle_request(server, server->client[i]->conn_fd);
            }
        }
    }
```

Where you should add your code

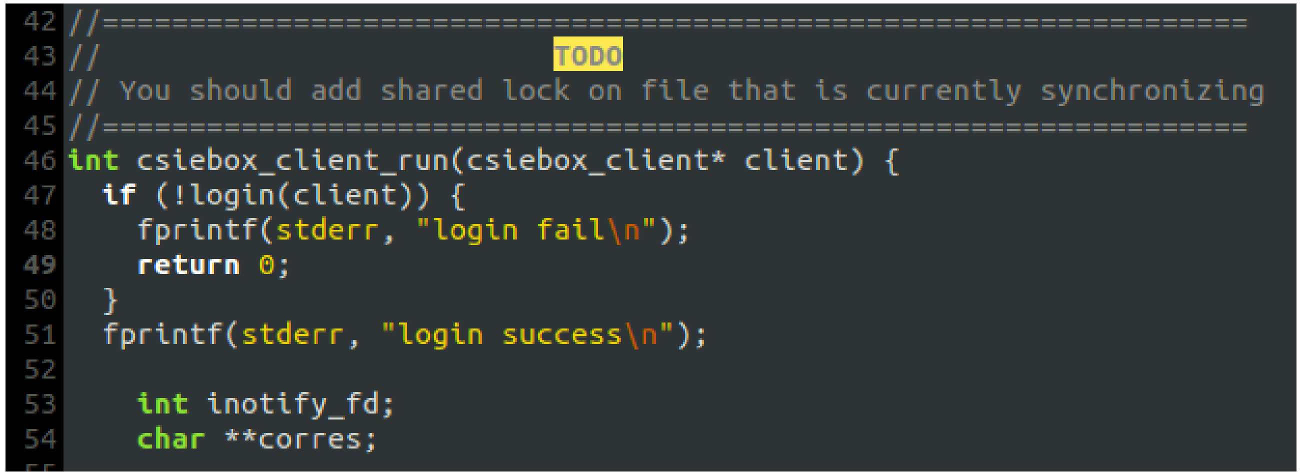
- server_sync_meta() in server_function.c
- You should add exclusive lock on the file that is currently synchronizing



```
//========================================================================
//                              TODO
// You should add exclusive lock on file that is currenting synchronizing
//========================================================================
int server_sync_meta( const csiebox_protocol_meta req, int conn_fd, const csieb
{
    int pathlen;
    char path[PATH_MAX];
    char hash[MD5_DIGEST_LENGTH];  // hash of file on server, to be compared wi
    char *userName;
    struct stat stat;
    csiebox_protocol_header res;
```

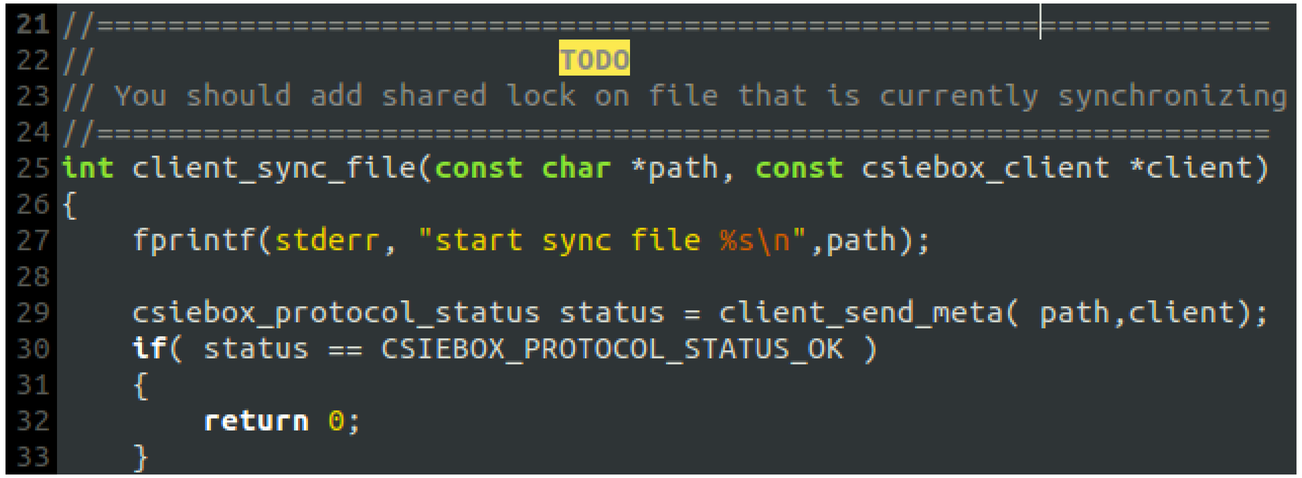
Where you should add your code

- csiebox_client_run() in csiebox_client.c and client_sync_file() in client_function.c
- You should add shared lock on the file that is currently synchronizing.



```
//===================================================================
//                              TODO
// You should add shared lock on file that is currently synchronizing
//===================================================================
int csiebox_client_run(csiebox_client* client) {
  if (!login(client)) {
    fprintf(stderr, "login fail\n");
    return 0;
  }
  fprintf(stderr, "login success\n");

    int inotify_fd;
    char **corres;
```



```
//===================================================================
//                              TODO
// You should add shared lock on file that is currently synchronizing
//===================================================================
int client_sync_file(const char *path, const csiebox_client *client)
{
    fprintf(stderr, "start sync file %s\n",path);

    csiebox_protocol_status status = client_send_meta( path,client);
    if( status == CSIEBOX_PROTOCOL_STATUS_OK )
    {
        return 0;
    }
```

Thread Pool

- If you are not familiar with the concept of thread pool, or don’t know how to implement, you can review the third part of the class at 11/30 on Youtube, and try to understand the sample code on ceiba
- There are also many tutorial and example of thread pool implementation online, like this and this

File Lock

- Use fcntl(2)
- There are two type of lock:
  - Shared lock(read lock): There can be any number of process holding shared lock on a file
  - Exclusive lock(write lock): exclusive lock exclude all other locks, both shared and excludsive
- File lock is advisory. It works only if all processes follow the rule
  - If a process A holds exclusive lock on a file, process B can still open the file and write to it if B doesn’t check file lock first

How to submit your code

- Push to github
- It should be like:

```
SP16MP4-your_student_ID
├──bin
├──src
└──include
```

Deadline

- Deadline postponed to 12/18