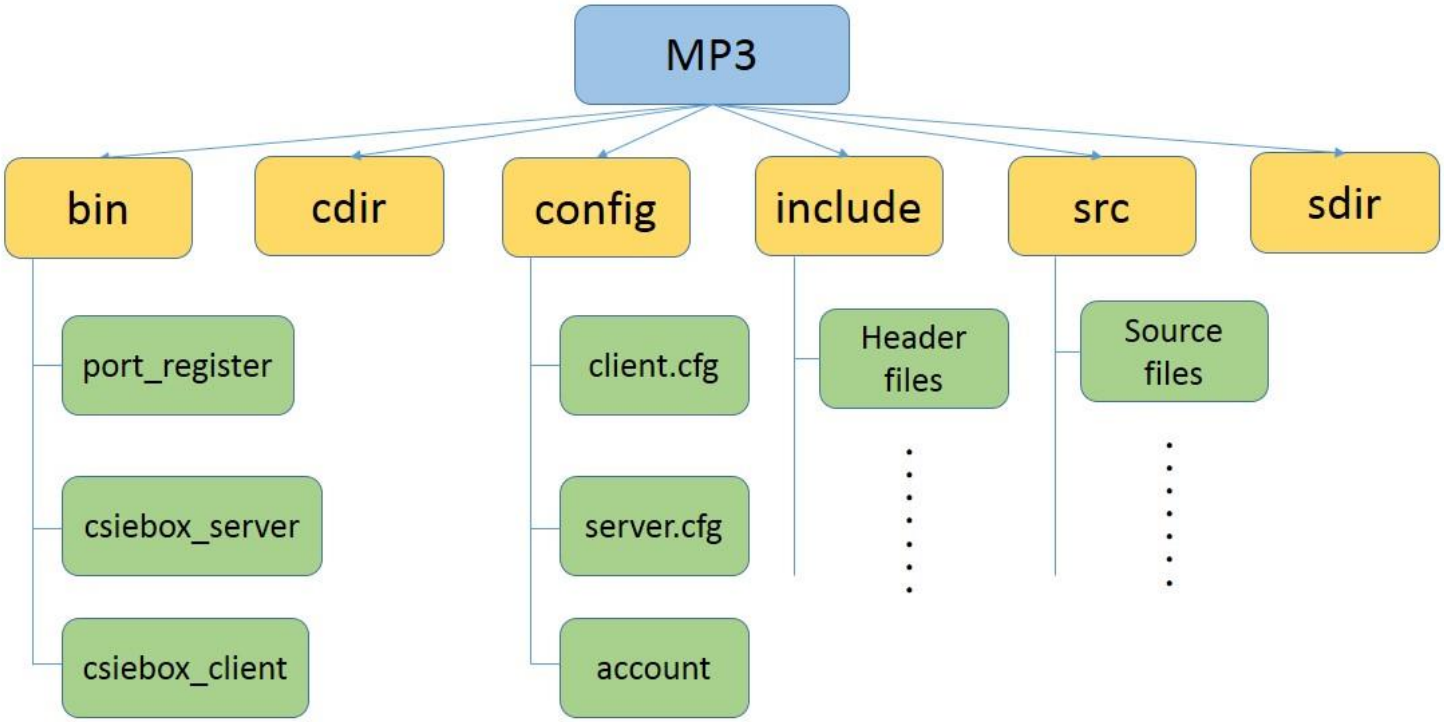
SYSTEM PROGRAMING HW3

Goal

- We are going to synchronize the files and directories between local repository and server side through Internet.
- You will need to run up multiple process in this homework.


MP3
bin
cdir
config
include
src
sdir
port_register
csiebox_server
csiebox_client
client.cfg
server.cfg
account
Header files
Source files

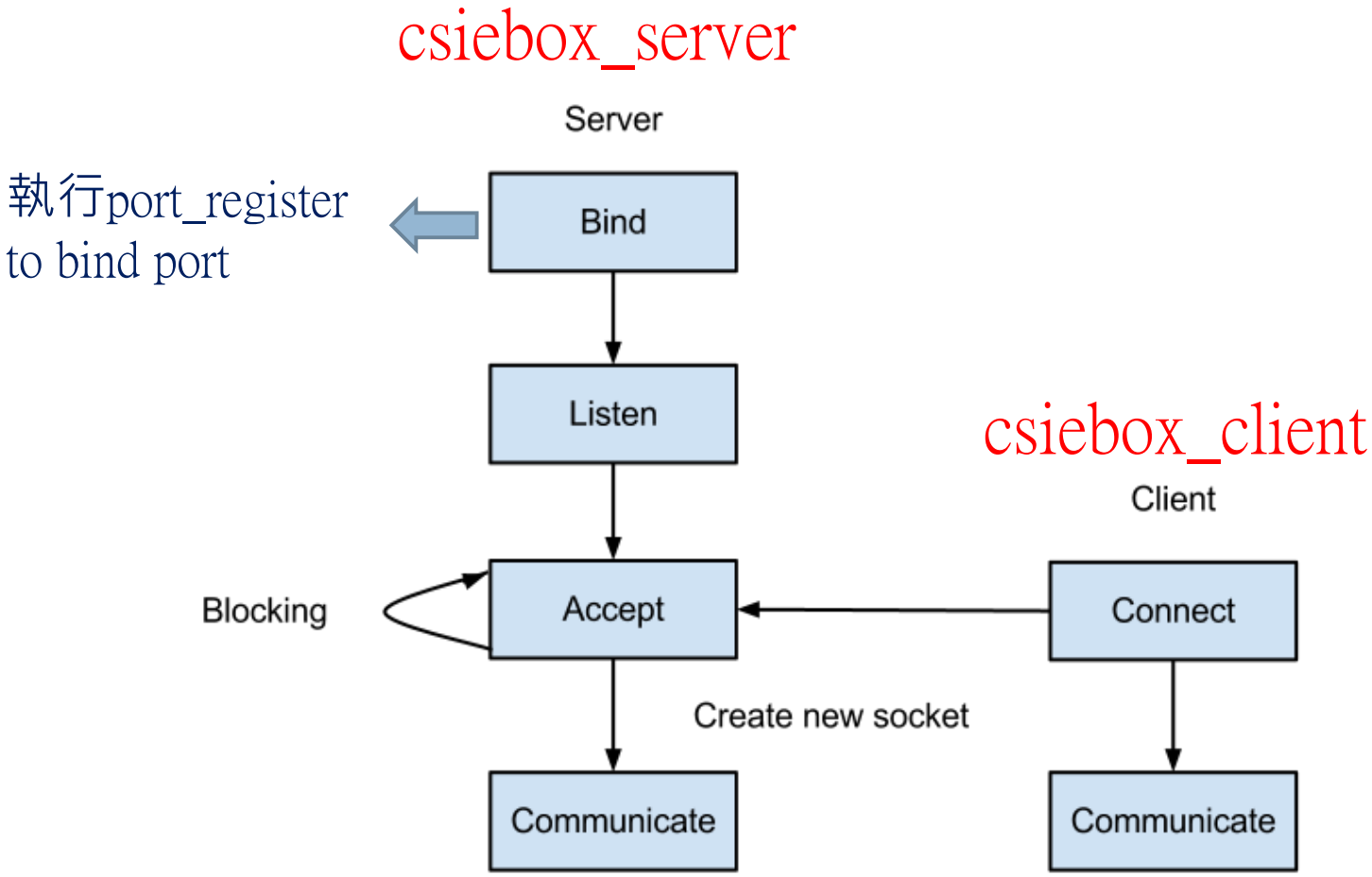
Architecture

- You will run up 2 process, csiebox_server and csiebox_client, and you need to use *socket* to build up communication between them.
- csiebox_server : it is the program for CSIEBox server. It needs to handle synchronization request from client and synchronize server directory accordingly.
- csiebox_client : It is the program for CSIEBox user. It needs to monitor client’ s local directory and send request to server for synchronization.

Socket

- UNIX Network Communicate API
- It's accessed as file descriptor, so you can communicate using read/write, or using socket's own recv/send API.
- recv_message(), send_message() in hw2

Architecture



port_register

- daemon provide by TA which provide port management service
- You need to run it before server and client

- TA would run this for you on linux1~linux15

- If you want to test connection at your own device, you should run port_register first
  - $ ./port_register &

How to run server and client?

1. Modify server config file named server.cfg
   - Path: the path of server directory, named “sdir” in example
   - Account_path: the path of account file, which has account info

```
path=/nfs/master/01/r01922088/SP/sp_hw2b02XXXXXX/sdir
account_path=/nfs/master/01/r01922088/SP/sp_hw2b02XXXXXX/config/account
```

Server.cfg

```
user,123
```

account

How to run server and client?

2. Modify client config file named client.cfg
   - Name: your account name, student ID in work station
   - Server: localhost
   - User
   - Passwd
   - Path: the path of client local repository, named “cdir” in example

```
name=r01922088
server=localhost
user=user
passwd=123
path=/nfs/master/01/r01922088/SP/sp_hw2b02XXXXXX/cdir
```

How to run server and client?

4. Cd to directory “src”, and type “make” to compile source code.

5. Cd to directory “bin”, execute process “port_register”. If you works on work station, you don’t have to do this, we have run it on work station.

How to run server and client?

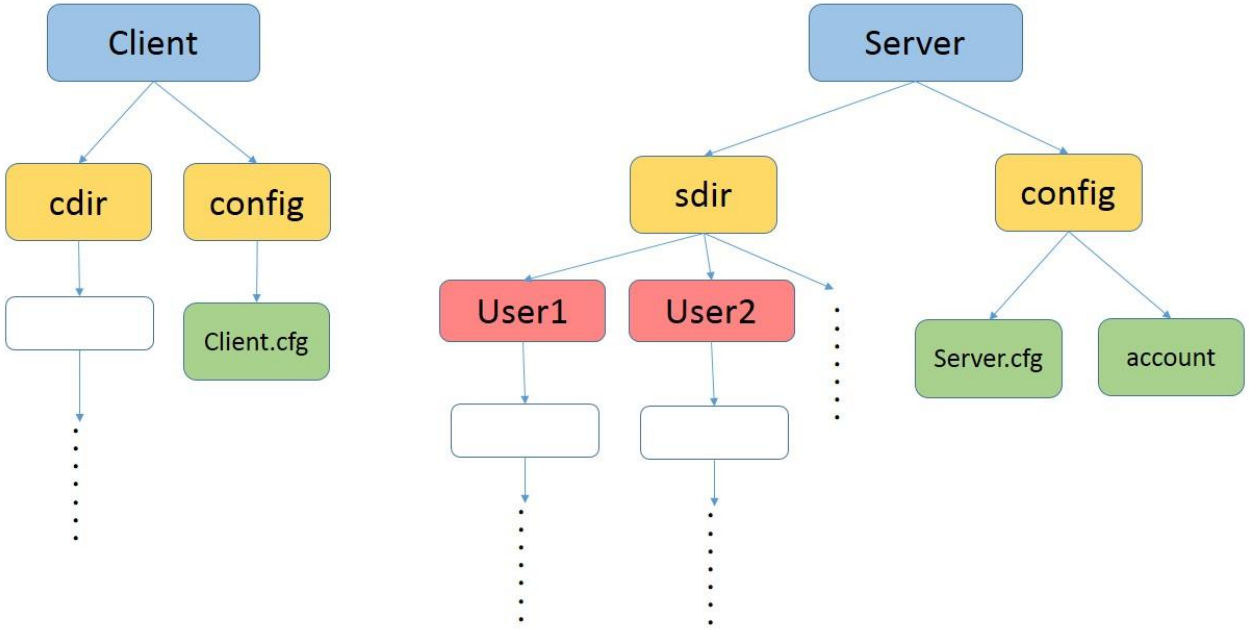
6. Execute process “csiebox_server” with path of server.cfg.

```
r01922088@linux1:~/SP/sp_hw2b02XXXXXX/bin> ./csiebox_server ../config/server.cfg
```

7. Execute process “csiebox_client” with path of client.cfg.
8. After finished the steps above, you should see that client successfully login to server.

Architecture

- You can treat sdir as server side, cdir as client side in your MP2 directory.
- All you need to do is sync file in client(cdir) to server(sdir).



Communication Protocol

- To synchronize client and server, we define 5 actions, which is login, sync meta, sync file, hard link, and rm.
- We have build up 5 types of header, one for each action, you need to use these headers to communicate between server and client.
- We also define rules of each action, you need to follow those rules to build up your CSIEBox client and server.

Five types of complete header

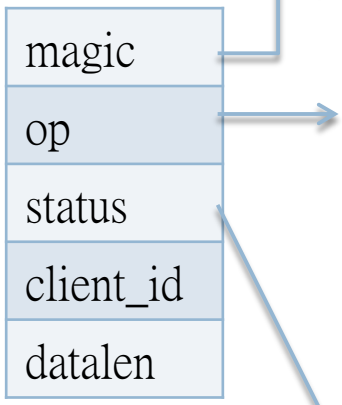
- csiebox_protocol_login
- csiebox_protocol_meta
- csiebox_protocol_file
- csiebox_protocol_hardlink
- csiebox_protocol_rm
- Each complete header has a common header
  - csiebox_protocol_header
- All of these headers are defined in **csiebox_common.h**.

Rules of actions

- You need to follow the rules of 5 actions and use those defined headers, but you can modify others if you want.
- In other words, the only things that you can't modify in HW2 is the rules of 5 actions and protocol header in **csiebox_common.h**

csiebox_protocol_header

```
typedef union {
  struct {
    uint8_t magic;
    uint8_t op;
    uint8_t status;
    uint16_t client_id;
    uint32_t datalen;
  } req;
  struct {
    uint8_t magic;
    uint8_t op;
    uint8_t status;
    uint16_t client_id;
    uint32_t datalen;
  } res;
  uint8_t bytes[9];
} csiebox_protocol_header;
```




```
typedef enum {
  CSIEBOX_PROTOCOL_MAGIC_REQ = 0x90,
  CSIEBOX_PROTOCOL_MAGIC_RES = 0x91,
} csiebox_protocol_magic;

typedef enum {
  CSIEBOX_PROTOCOL_OP_LOGIN = 0x00,
  CSIEBOX_PROTOCOL_OP_SYNC_META = 0x01,
  CSIEBOX_PROTOCOL_OP_SYNC_FILE = 0x02,
  CSIEBOX_PROTOCOL_OP_SYNC_HARDLINK = 0x03,
  CSIEBOX_PROTOCOL_OP_SYNC_END = 0x04,
  CSIEBOX_PROTOCOL_OP_RM = 0x05,
} csiebox_protocol_op;

typedef enum {
  CSIEBOX_PROTOCOL_STATUS_OK = 0x00,
  CSIEBOX_PROTOCOL_STATUS_FAIL = 0x01,
  CSIEBOX_PROTOCOL_STATUS_MORE = 0x02,
} csiebox_protocol_status;
```

csiebox_protocol_header

- Attribute Magic, defined for security purpose.
- Attribute Op defines operation code.
- Attribute Status defines return status.
- Attribute Client_id defines the identification of client processes, which will be used in multiple clients scenario.
- Attribute Datalen defines the length of optional attributes and will be used to identify the types of optional attributes.

csiebox_protocol_login

```
typedef union {
  struct {
    csiebox_protocol_header header;
    struct {
      uint8_t user[USER_LEN_MAX];
      uint8_t passwd_hash[MD5_DIGEST_LENGTH];
    } body;
  } message;
  uint8_t bytes[sizeof(csiebox_protocol_header) + MD5_DIGEST_LENGTH * 2];
} csiebox_protocol_login;
```

- Client sends user name and passwd_hash to server. If the user name and password matches, server returns OK in status field and client_id. Otherwise, FAIL is returned in status field.

csiebox_protocol_meta

```
typedef union {
  struct {
    csiebox_protocol_header header;
    struct {
      uint32_t pathlen;
      struct stat stat;
      uint8_t hash[MD5_DIGEST_LENGTH];
    } body;
  } message;
  uint8_t bytes[sizeof(csiebox_protocol_header) +
                4 +
                sizeof(struct stat) +
                MD5_DIGEST_LENGTH];
} csiebox_protocol_meta;
```

- When file/dir's status have been changed, client will send sync_meta request include file pathlen and meta stat to server followed by file/dir path.
- Client send this header include file pathlen and meta stat to server, and followed by file path.

csiebox_protocol_file

```
typedef union {
  struct {
    csiebox_protocol_header header;
    struct {
      uint64_t datalen;
    } body;
  } message;
  uint8_t bytes[sizeof(csiebox_protocol_header) + 8];
} csiebox_protocol_file;
```

- Sync file session consists of 2 steps in this operation.
- First, the client sends a sync_meta header which provides struct stat, file path length and file hash, and client sends file path to server followed.
- Second, Server has to sync meta, and it checks the file hash, if file hash on client is different from that on server, or the file is not exist, the server returns MORE and the client will send file data to server. Otherwise, the server returns OK and terminates the session.

csiebox_protocol_hardlink

```
typedef union {
  struct {
    csiebox_protocol_header header;
    struct {
      uint32_t srclen;
      uint32_t targetlen;
    } body;
  } message;
  uint8_t bytes[sizeof(csiebox_protocol_header) + 8];
} csiebox_protocol_hardlink;
```

- Client send this header include source path length and target path length to server, and followed by source and target path.
- If a hard link is successfully created, server returns OK in status field. Otherwise, it returns FAIL in status field.
- Hint: you can use lstat() to check Inode to determine either it is hard link or not.

csiebox_protocol_rm

```
typedef union {
  struct {
    csiebox_protocol_header header;
    struct {
      uint32_t pathlen;
    } body;
  } message;
  uint8_t bytes[sizeof(csiebox_protocol_header) + 4];
} csiebox_protocol_rm;
```

- Client send this header include path length to server, and followed by file/dir path.
- If the file/directory is successfully removed on the server, the server returns OK in status field. Otherwise, it returns FAIL in status field.

Example-Sync Meta client side

```
csiebox_protocol_meta req;
memset(&req,0,sizeof(csiebox_protocol_meta));
req.message.header.req.magic = CSIEBOX_PROTOCOL_MAGIC_REQ;
req.message.header.req.op = CSIEBOX_PROTOCOL_OP_SYNC_META;
req.message.header.req.datalen = sizeof(req) - sizeof(req.message.header);
req.message.body.pathlen = strlen(filePath);
lstat(filePath,&(req.message.body.stat));
if( S_ISREG(req.message.body.stat.st_mode ) )
    md5_file(path,req.message.body.hash);
if( S_ISLNK(req.message.body.stat.st_mode ) )
{
    char buf[PATH_MAX];
    int len;
    memset(buf,0,PATH_MAX);
    len = readlink(path,buf,PATH_MAX);
    md5(buf, len, req.message.body.hash );
}

// Send meta req to server
if (!send_message(client->conn_fd, &req, sizeof(req))) {
    fprintf(stderr, "send req fail\n");
    return -1;
}

// Send file path to server
if (!send_message(client->conn_fd, relaPath, strlen(filePath))) {
    fprintf(stderr, "send data fail\n");
    return -1;
}

csiebox_protocol_header header;
memset(&header, 0, sizeof(header));
if (recv_message(client->conn_fd, &header, sizeof(header))) {
    if (header.res.magic == CSIEBOX_PROTOCOL_MAGIC_RES &&
        header.res.op == CSIEBOX_PROTOCOL_OP_SYNC_META ) {
            fprintf(stderr, "receive from server: %04x\n",header.res.status );
            return (int) header.res.status;
        } else {
            fprintf(stderr,"receive from server fail\n");
            return -1;
    }
}
```

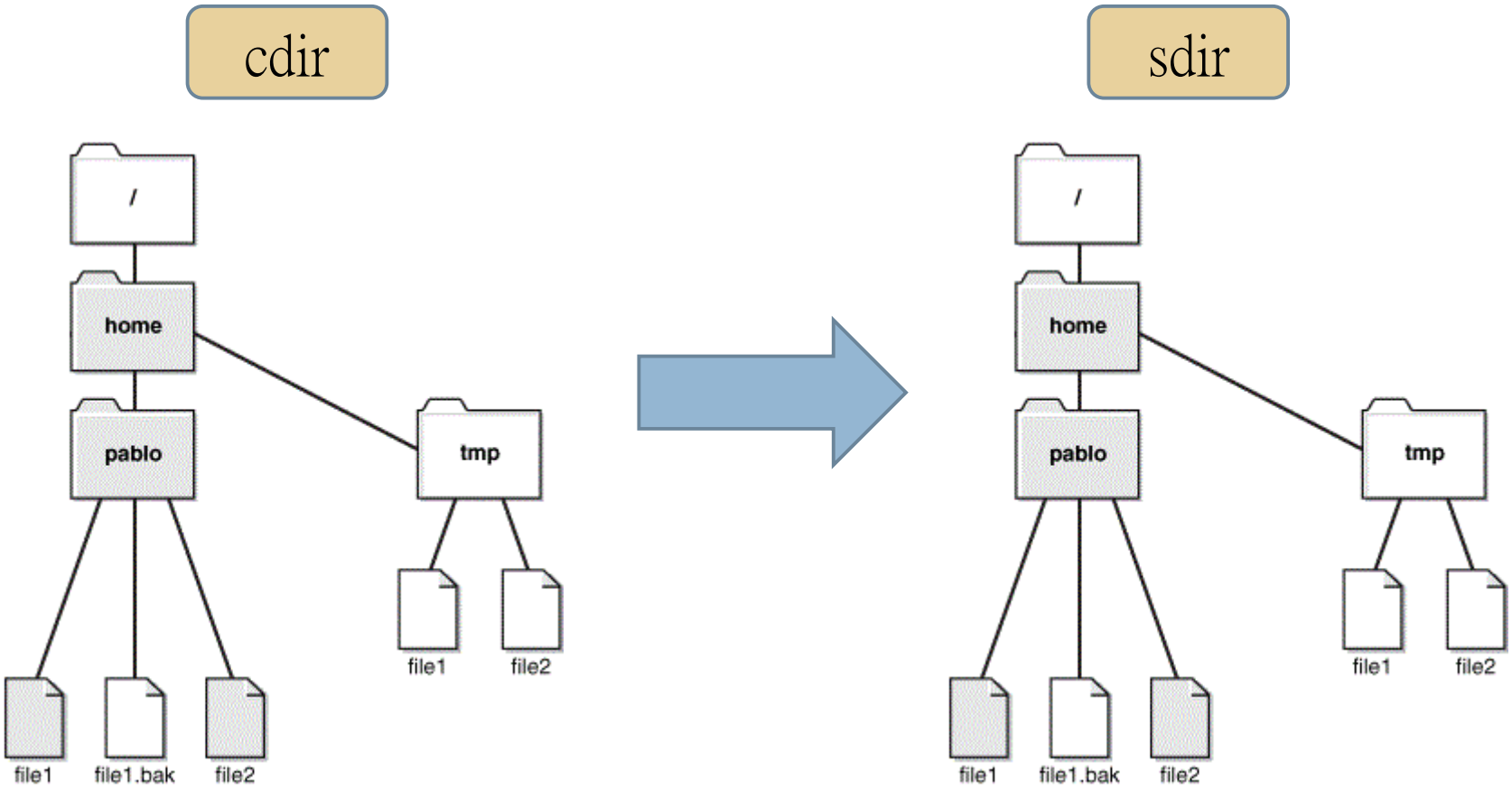
- More example can be found in csiebox_server.c and csiebox_client.c, where they handle user login.

Requirement

- Scan the directory
  - The process traverses/scans the local repository directory to find out the existed files and directories, and do the synchronization.
- In this step, the process starts from the top of local repository and transmits all the files and directories to the server side. You will use the provided socket communication template to transmit the files.
  - However, you have to handle the operations for maintaining directories including creating directories, reconstructing soft/hard link on the server sides, etc.

Assignment 1

- Part1: Scan the local directory and sync(upload) to server

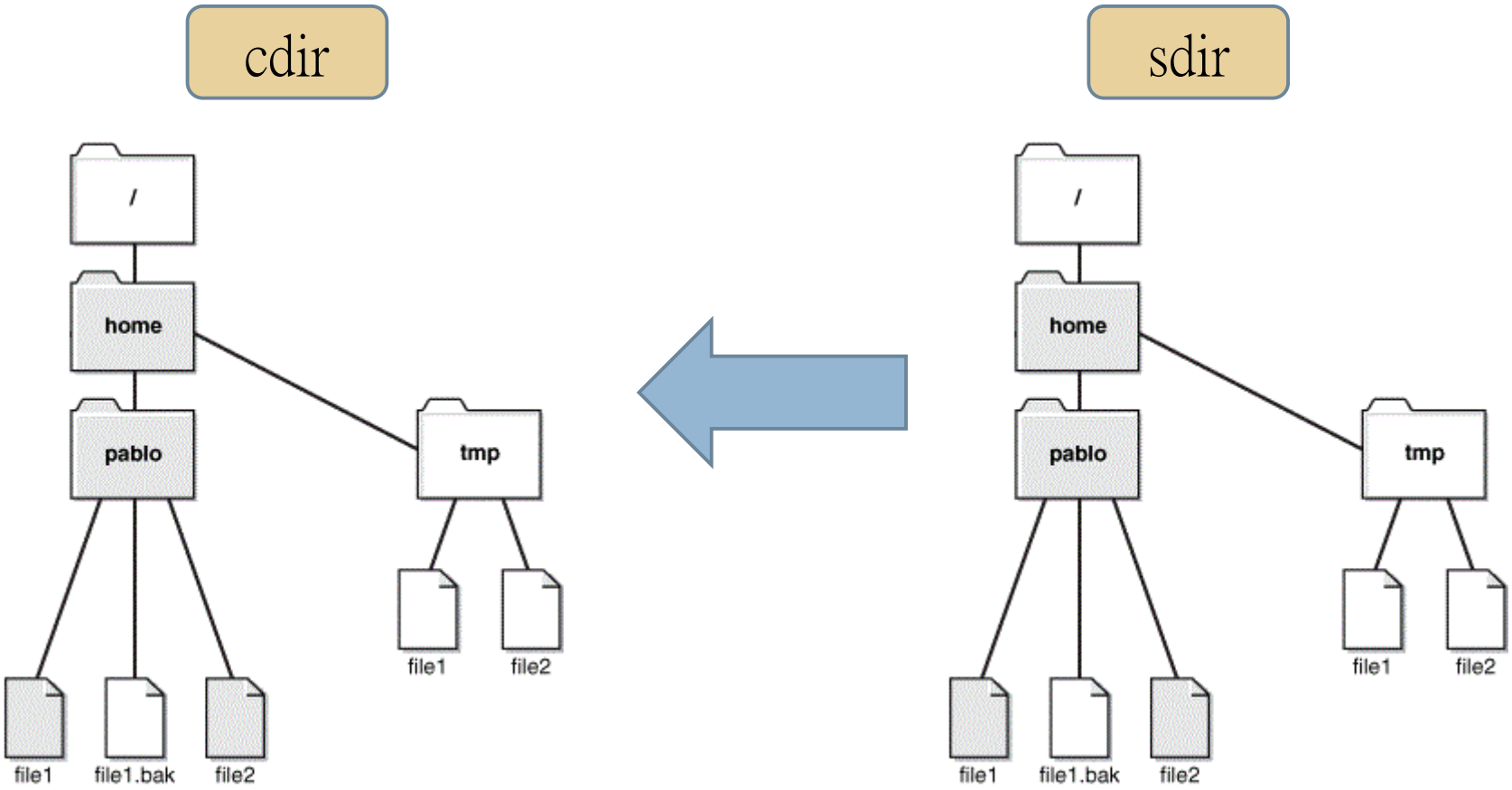


Assignment 1

- Part2: Monitor the changes on files, directories, hard link, and soft link on local repository
  - To keep the file/directories to be up-to-date with the local repository, the process has to monitor if there is any change on files/directories.
- You can use inotify API, which can monitor file system events, to catch the update and do corresponding operations.
  - Example program of using inotify API is available in the provided package, which is inotify_test.c.

Assignment 2

- Part1: Download remote repository to client



Assignment 2

- Part2: Monitor the changes on files, directories, hard link, and soft link on local repository

Assignment 3

- Part1: the same assignment 1, except that this time there will be many clients.
- You need to use I/O multiplexing on server side to handle requests from multiple clients.
- select(), poll(),epoll()
- Part2: Monitor the changes on files, directories, hard link, and soft link on local repository

Assignment 4

- Like assignment 3, except this time, some clients need to download from server instead of upload.

Requirement

- Note that you have to pay special attention on handling the changes on **links**.
- In the case of symbolic link, you need to update the value of link instead of updating the content it points to. Even if the file which the symbolic link points to is out of the local repository, or it is a bad link, or there is a loop in the symbolic links, we should do the synchronization anyway.

Requirement

- A hard link should be processed in 2 cases.
  - If it links to a file within the directory tree of local repository, it should remain to be a hard link on server side.
  - Otherwise, it should be treated as a regular file, rather than a hard link.

Grading

- Total 8 points.
- 2 points for each assignment

Due date

- 11:59PM, **November 23, 2016**.
- 5% of your credits will be deducted for every single day delay.
- There is absolutely **NO** tolerance for plagiarism. Zero points for plagiarism.

Meaning of files

- **csiebox_client.c:** this's the process to simulate client, you need to implement your code about client action in this file.
- **csiebox_server.c:** this's the process to simulate server, you need to implement your code about server action in this file.

Meaning of files

- **hash.c:** provide adata structure, which can help you to manage the relation of inotify id and monitored dir path.
  - By inotify API, you can use wd = inotify_add_watch(inotify_fd, "/PathOfDir_YouWantToMonitor", IN_CREATE | IN_DELETE | IN_ATTRIB | IN_MODIFY) to add a monitor on a directory.
  - wd is the inotify id and /PathOfDir_YouWantToMonitor is the directory path.

Meaning of files

- You can use put_into_hash(&(client->inotify_hash), (void*)inotify_path, wd) to store this relation.
- Use get_from_hash(&(client->inotify_hash), (void**)&wd_path, event->wd) to retrive the monitored directory path by event->wd.

Meaning of files

- **connect.c, port_register.c, csiebox_common.c:** connect between client process and server process.
- **inotify_test.c:** a test program about inotify, you need to learn how to use it, and write an inotify system in csiebox_client.c.

Linux command in C language

- getcwd: get current dir name.
- opendir, readdir: opendir can open a directory, and readdir can read a direrory. These functions can help you to traverse directory tree.
- chdir: change working directory.
- closedir: close a directory.
- mkdir: make directories.

Linux command in C language

- lstat: get file status. This help you to get metadata/inode information.
- readlink: print resolved symbolic links or canonical file names.
- symlink: symbolic link handling. (create sofylink)
- link: call the link function to create a link to a file. (create hardlink)
- rmdir: delete a directory.
- unlink: call the unlink function to remove the specified file.