Programming Assignment #6
A User-Space File System

Introduction to Operating Systems

Prof. Li-Pin Chang

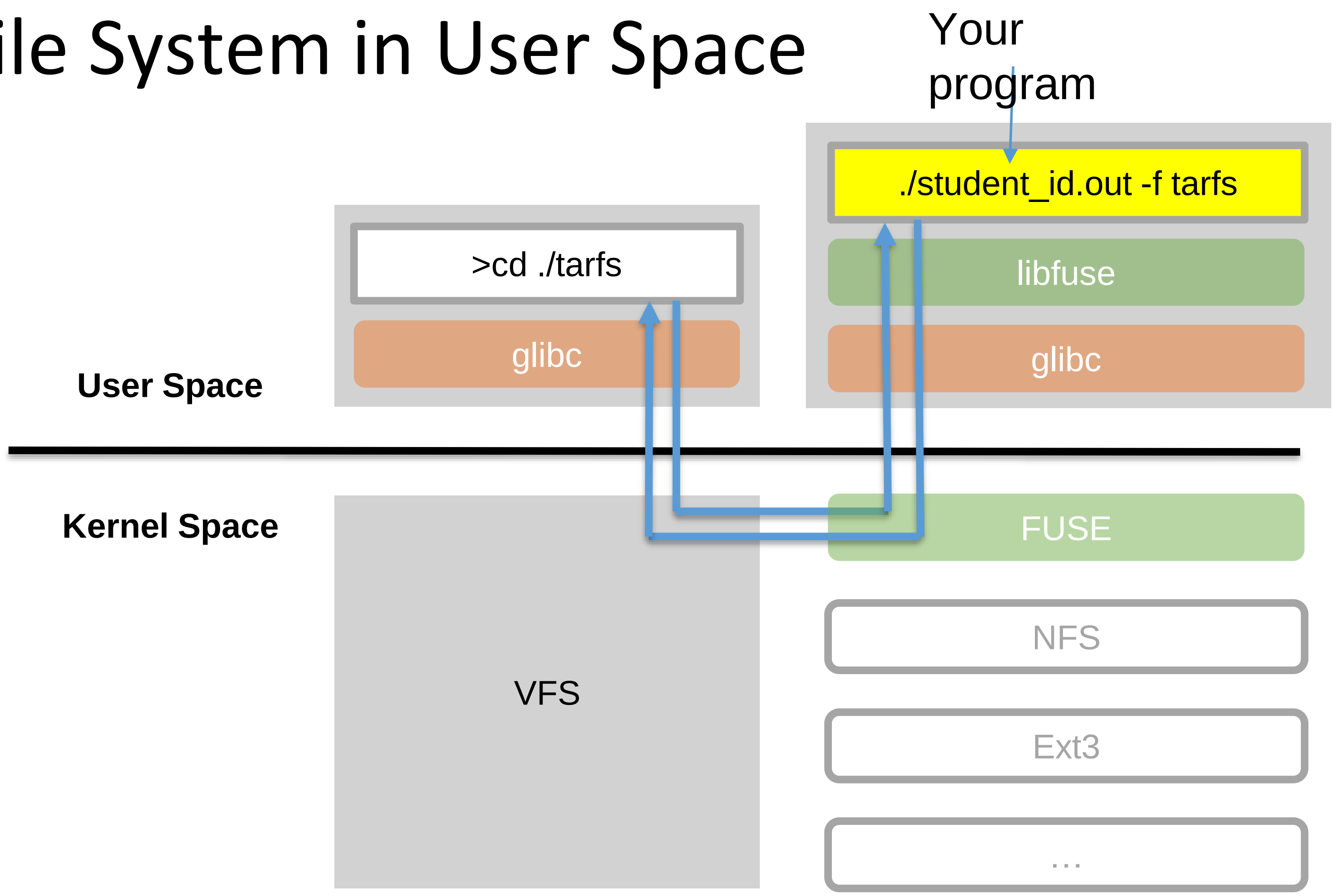
CS@NYCU

Objective

- Implementing a user-space file system that mounts a tar file to a specified directory
  - Files in the tar files can be accessed through the directory tree

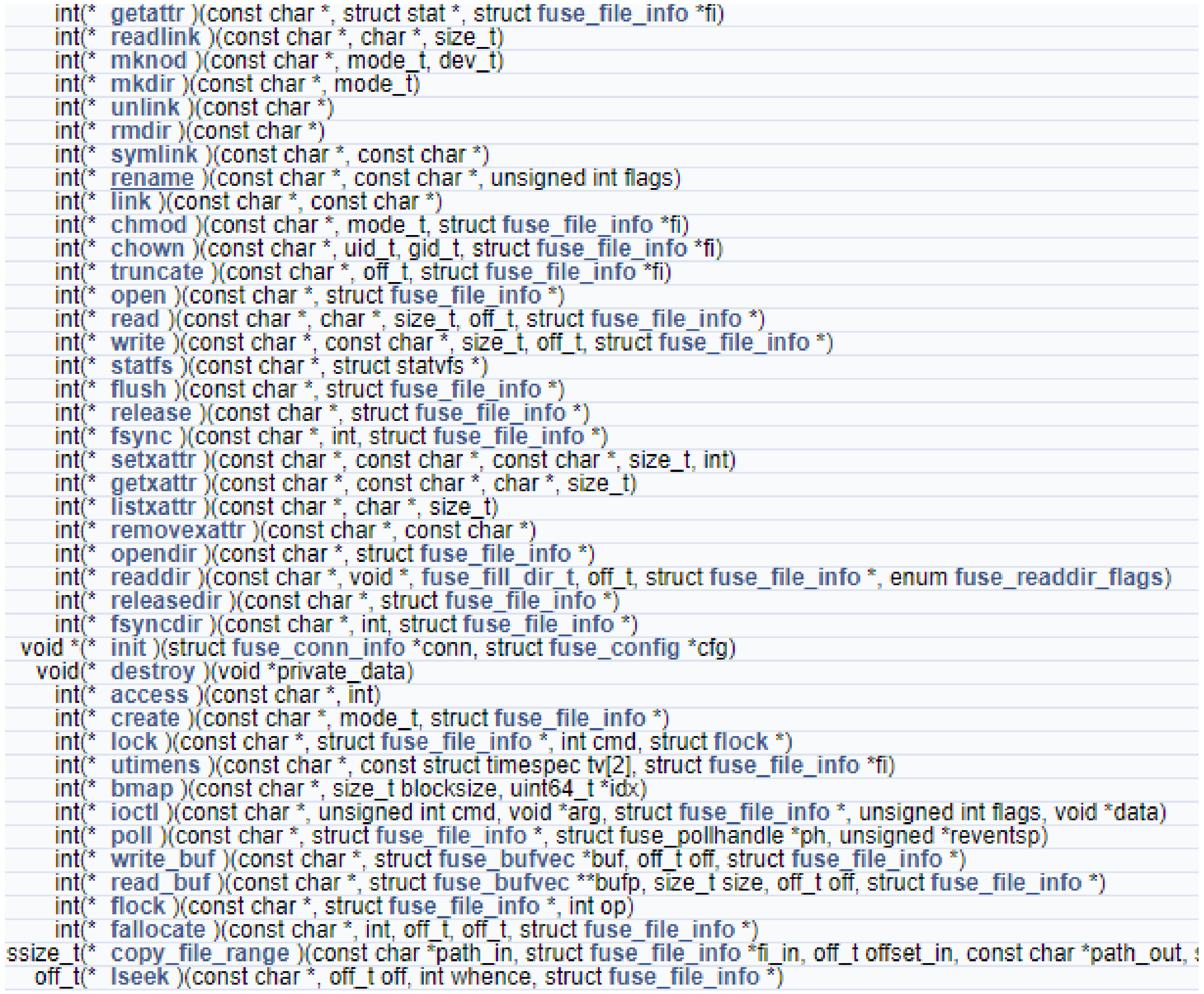
- This assignment is based on FUSE of Linux
  - Your program will run as a FUSE server
  - Test your FUSE server from another terminal

FUSE – File System in User Space



- FUSE: a kernel component plus a user-space library
- Purpose: accessing existing files/services through the file system interface
  - E.g., an FTP file system, a zip file system, etc.

The Complete FUSE Operation Set



```
   int(*  getattr )(const char *, struct stat *, struct fuse_file_info *fi)
   int(*  readlink )(const char *, char *, size_t)
   int(*  mknod )(const char *, mode_t, dev_t)
   int(*  mkdir )(const char *, mode_t)
   int(*  unlink )(const char *)
   int(*  rmdir )(const char *)
   int(*  symlink )(const char *, const char *)
   int(*  rename )(const char *, const char *, unsigned int flags)
   int(*  link )(const char *, const char *)
   int(*  chmod )(const char *, mode_t, struct fuse_file_info *fi)
   int(*  chown )(const char *, uid_t, gid_t, struct fuse_file_info *fi)
   int(*  truncate )(const char *, off_t, struct fuse_file_info *fi)
   int(*  open )(const char *, struct fuse_file_info *)
   int(*  read )(const char *, char *, size_t, off_t, struct fuse_file_info *)
   int(*  write )(const char *, const char *, size_t, off_t, struct fuse_file_info *)
   int(*  statfs )(const char *, struct statvfs *)
   int(*  flush )(const char *, struct fuse_file_info *)
   int(*  release )(const char *, struct fuse_file_info *)
   int(*  fsync )(const char *, int, struct fuse_file_info *)
   int(*  setxattr )(const char *, const char *, const char *, size_t, int)
   int(*  getxattr )(const char *, const char *, char *, size_t)
   int(*  listxattr )(const char *, char *, size_t)
   int(*  removexattr )(const char *, const char *)
   int(*  opendir )(const char *, struct fuse_file_info *)
   int(*  readdir )(const char *, void *, fuse_fill_dir_t, off_t, struct fuse_file_info *, enum fuse_readdir_flags)
   int(*  releasedir )(const char *, struct fuse_file_info *)
   int(*  fsyncdir )(const char *, int, struct fuse_file_info *)
 void *(*  init )(struct fuse_conn_info *conn, struct fuse_config *cfg)
  void(*  destroy )(void *private_data)
   int(*  access )(const char *, int)
   int(*  create )(const char *, mode_t, struct fuse_file_info *)
   int(*  lock )(const char *, struct fuse_file_info *, int cmd, struct flock *)
   int(*  utimens )(const char *, const struct timespec tv[2], struct fuse_file_info *fi)
   int(*  bmap )(const char *, size_t blocksize, uint64_t *idx)
   int(*  ioctl )(const char *, unsigned int cmd, void *arg, struct fuse_file_info *, unsigned int flags, void *data)
   int(*  poll )(const char *, struct fuse_file_info *, struct fuse_pollhandle *ph, unsigned *reventsp)
   int(*  write_buf )(const char *, struct fuse_bufvec *buf, off_t off, struct fuse_file_info *)
   int(*  read_buf )(const char *, struct fuse_bufvec **bufp, size_t size, off_t off, struct fuse_file_info *)
   int(*  flock )(const char *, struct fuse_file_info *, int op)
   int(*  fallocate )(const char *, int, off_t, off_t, struct fuse_file_info *)
ssize_t(*  copy_file_range )(const char *path_in, struct fuse_file_info *fi_in, off_t offset_in, const char *path_out,
  off_t(*  lseek )(const char *, off_t off, int whence, struct fuse_file_info *)
```

Specification

- Your server must support the following operations
  - Listing directories
  - Reading files
  - Handling soft link
- Files from us
  - hw6.zip that contains everything: tar files, test script, etc.
- You write
  - A file server “{student_ID}.c” or “{student_ID}.cpp”
- Notice
  - The input filename is hardcoded as “test.tar” (you can check the script)
  - You must turn in one c/cpp file only

Necessary FUSE Operations

```
struct fuse_operations {
        int (*readdir)(const char *, void *, fuse_fill_dir_t, off_t, struct fuse_file_info *);
        int (*getattr)(const char *, struct stat *);
        int (*read)(const char *, char *, size_t, off_t, struct fuse_file_info *);
        int (*readlink)(const char *path, void *buffer, size_t size);
        //many other functions…
}
```

- The complete FUSE operation set contains many callback functions, but only three are necessary to this assignment
    - **readdir**: Get a list of files and directories that reside in the directory. (Get file names only)
    - **getattr**: Get attributes of a file/directory.
    - **read**: Get the content of a file
    - **readlink**: get a symbolic link
- Leave null to the other operations

readdir

```
int readdir(const char *path, void *buffer, fuse_fill_dir_t filler, off_t offset, struct fuse_file_info *fi);
```

- **Arguments**
  - path: (full) relative path to the file/directory.
  - buffer: store file names into this buffer using the provided filler
  - filler: FUSE callback function to fill file names into the buffer, e.g.,
    - filler(buffer, “file1.txt”, NULL, 0);
    - filler(buffer, “dir1”, NULL, 0);
    - The function will handle internal buffer organization
  - offset and fi: Not used in this assignment
- **Return values**
  - Always return 0.

getattr

```
int getattr(const char *path, struct stat *st);
```

- **Arguments**
  - path: (full) relative path to the file/directory.
  - st: You should fill the necessary fields of this structure.
- About structure stat: https://pubs.opengroup.org/onlinepubs/009695399/basedefs/sys/stat.h.html
- Necessary Fields of st: st_uid, st_gid, st_mtime, st_size and st_mode
  - st_mode of the root directory ("/") should be set to: S_IFDIR | 0444 (act like a read only directory)
  - Other directories: S_IFDIR | accessMode
  - Regular files: S_IFREG | accessMode
- **Return values**
  - Return 0 on success.
  - Return a nonzero value on failure. (If cannot find the specified file/directory)

read

```
int read(const char *path, char *buffer, size_t size, off_t offset, struct fuse_file_info *fi);
```

- **Arguments**
  - path: (full) relative path to the file/directory.
  - buffer: You should store the requested file content into this buffer.
  - size: Max # of chars to store in the buffer. (Should not overrun)
  - offset: Skip *offset* chars from the beginning of the file and then start reading.
  - fi: Not used in this assignment
- **Return values**
  - Return number of bytes read successfully.

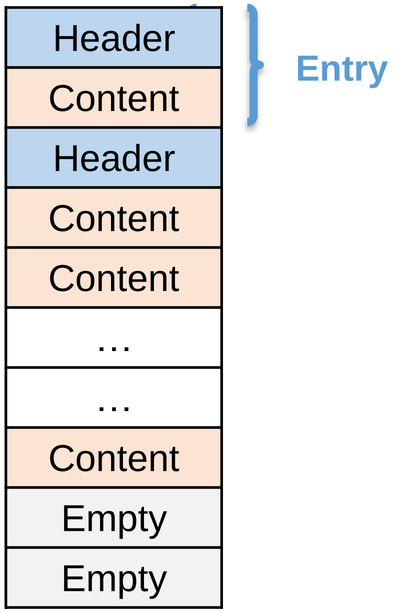
readlink

```
int readlink(const char *path, void *buffer, size_t size);
```

- **Arguments**
  - path: The path of the symbolic link to be resolved
  - buffer: A buffer to store the target of the symbolic link (the path it points to)
  - size_t: The maximum size of the buffer
- **Return values**
  - Always return 0.

Tar File Format

- A tar file contains a series of entries, each of which contains a header and contents
    - One entry per file
    - Header: metadata of a file
    - Contents: contents of the file
- You are responsible for reading and parsing the information of a tar file
- Detailed explanation of tar format: https://www.systutorials.com/docs/linux/man/5-tar/



Skeleton of Your FUSE Server

```
#define FUSE_USE_VERSION 30
#include <fuse.h>
#include <string.h>
int my_readdir(const char *path, void *buffer, fuse_fill_dir_t filler, off_t offset, struct fuse_file_info *fi) { /*do something*/ }
int my_getattr(const char *path, struct stat *st) { /*do something*/ }
int my_read(const char *path, char *buffer, size_t size, off_t offset, struct fuse_file_info *fi) { /*do something*/ }
int readlink(const char *path, void *buffer, size_t size) {/* do something */ }
static struct fuse_operations op;
int main(int argc, char *argv[])
{
    memset(&op, 0, sizeof(op));
    op.getattr = my_getattr;
    op.readdir = my_readdir;
    op.read = my_read;
    op.readlink = my_readlink;
    return fuse_main(argc, argv, &op, NULL);
}
```

Remarks on readlink (soft link)

- The callback ***readlink*** handles symbolic links
- TAR has dedicate records for symbolic link, check the TAR format
- Don’t worry about dangling link. The target file is always present in our test cases.
- Your symbolic link should retain its attributes, and the test script must be able to read the target file correctly

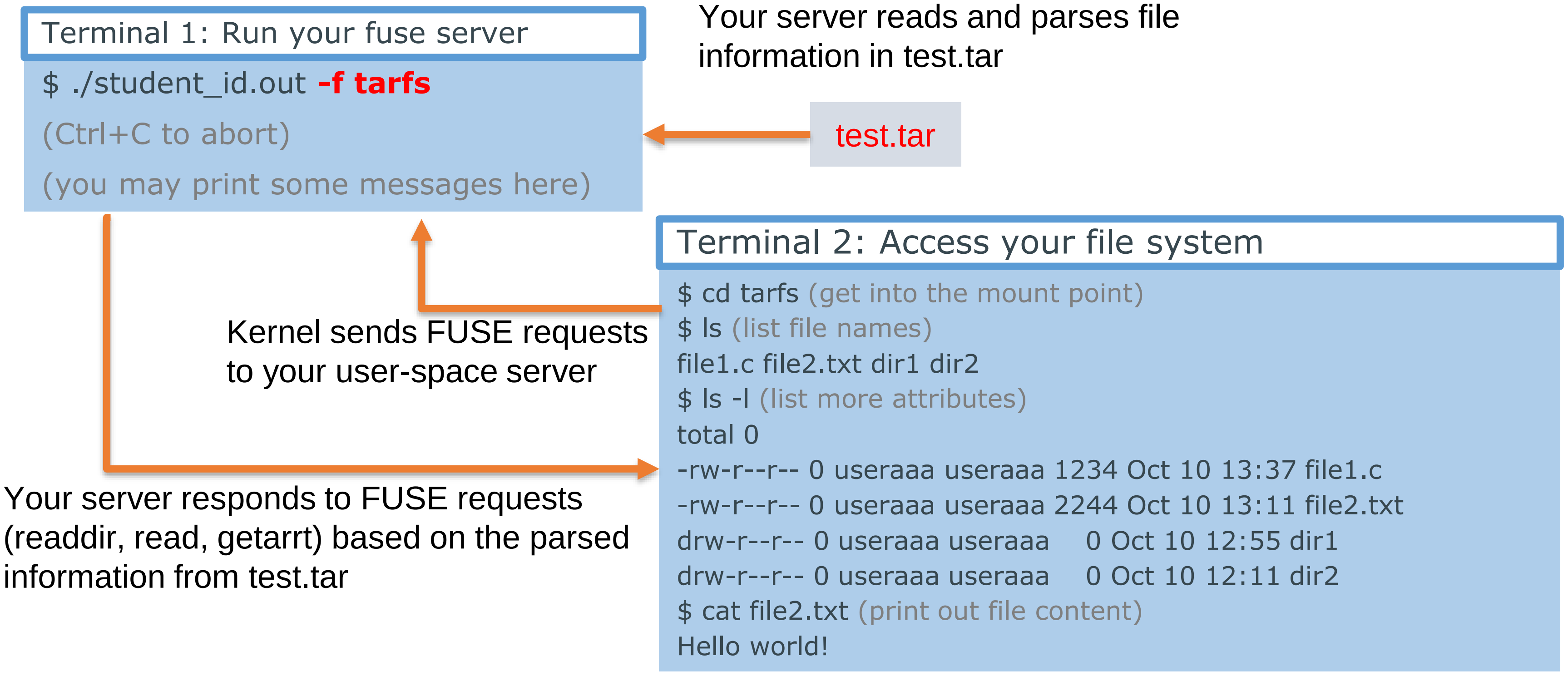
Compiling Your FUSE server

- Install FUSE for your Ubuntu VM
  - `sudo apt install libfuse-dev`
- Compile

```
$ gcc student_id.c -o student_id.out `pkg-config fuse --cflags --libs`
OR
$ g++ student_id.cpp -o student_id.out `pkg-config fuse --cflags --libs`
```

Running and Testing Your FUSE server

這是手動測試的方法，助教評分會用下頁的自動測試腳本！



Terminal 1: Run your fuse server

```
$ ./student_id.out -f tarfs
(Ctrl+C to abort)
(you may print some messages here)
```

Your server reads and parses file information in test.tar

test.tar

Kernel sends FUSE requests to your user-space server

Terminal 2: Access your file system

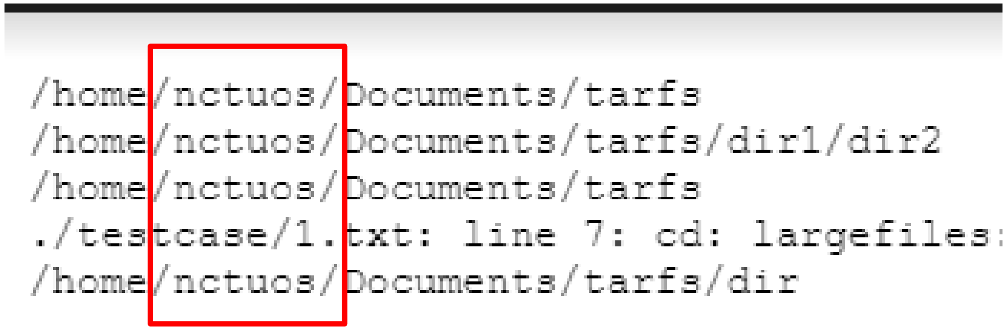
```
$ cd tarfs (get into the mount point)
$ ls (list file names)
file1.c file2.txt dir1 dir2
$ ls -l (list more attributes)
total 0
-rw-r--r-- 0 useraaa useraaa 1234 Oct 10 13:37 file1.c
-rw-r--r-- 0 useraaa useraaa 2244 Oct 10 13:11 file2.txt
drw-r--r-- 0 useraaa useraaa    0 Oct 10 12:55 dir1
drw-r--r-- 0 useraaa useraaa    0 Oct 10 12:11 dir2
$ cat file2.txt (print out file content)
Hello world!
```

Your server responds to FUSE requests (readdir, read, getarrt) based on the parsed information from test.tar

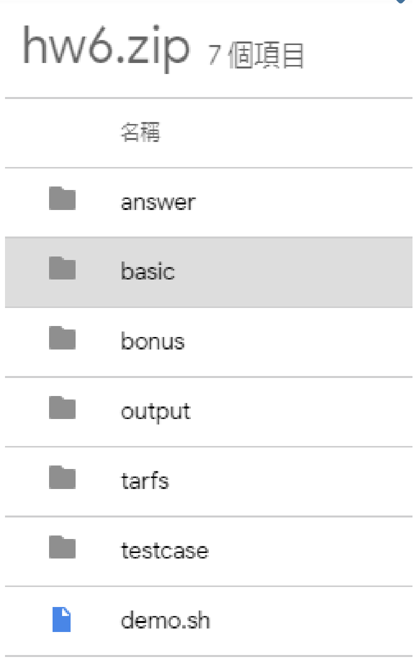
"tarfs" is an empty directory, used as a mount point of your FUSE server

Test Script

1. Download and extract hw6.zip
2. Put your executable file in the same directory as demo.sh
3. In the following files, change "nctuos" to your user account name
   - ./answer/1.txt
   - ./answer/2.txt
   - ./answer/5.txt
4. Add an executable attribute to ./testcase/*txt



```
/home/nctuos/Documents/tarfs
/home/nctuos/Documents/tarfs/dir1/dir2
/home/nctuos/Documents/tarfs
./testcase/1.txt: line 7: cd: largefiles:
/home/nctuos/Documents/tarfs/dir
```



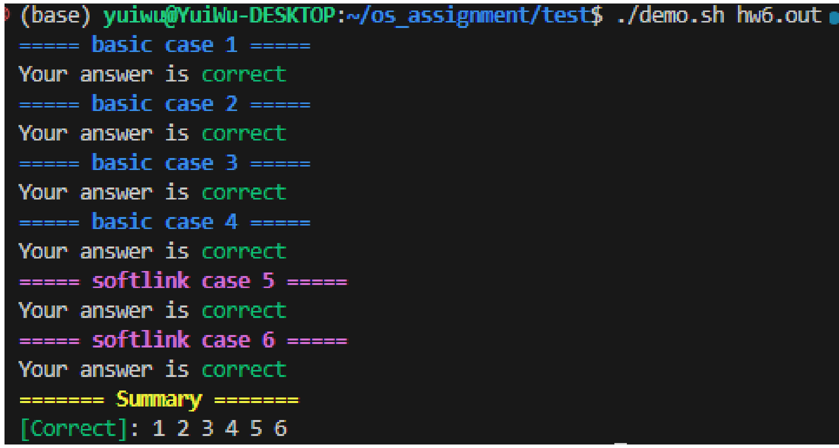
hw6.zip 7 個項目

名稱
answer
basic
bonus
output
tarfs
testcase
demo.sh

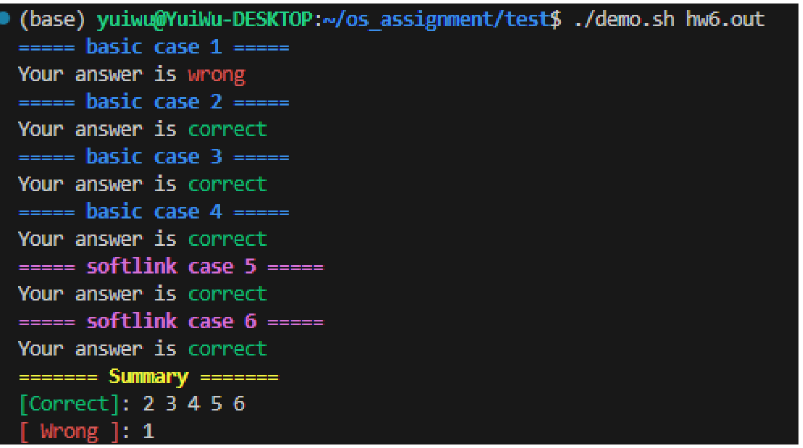
student_id.out

1. Run the script: ./demo.sh <pathname of your FUSE server>

Testing Results



```
(base) yuiwu@YuiWu-DESKTOP:~/os_assignment/test$ ./demo.sh hw6.out
===== basic case 1 =====
Your answer is correct
===== basic case 2 =====
Your answer is correct
===== basic case 3 =====
Your answer is correct
===== basic case 4 =====
Your answer is correct
===== softlink case 5 =====
Your answer is correct
===== softlink case 6 =====
Your answer is correct
======= Summary =======
[Correct]: 1 2 3 4 5 6
```



```
(base) yuiwu@YuiWu-DESKTOP:~/os_assignment/test$ ./demo.sh hw6.out
===== basic case 1 =====
Your answer is wrong
===== basic case 2 =====
Your answer is correct
===== basic case 3 =====
Your answer is correct
===== basic case 4 =====
Your answer is correct
===== softlink case 5 =====
Your answer is correct
===== softlink case 6 =====
Your answer is correct
======= Summary =======
[Correct]: 2 3 4 5 6
[ Wrong ]: 1
```

All pass

Some errors

Basic: 1,2,3,4
Softlink: 5,6

Remarks

- If you get a broken mount point during testing, use the following command to force unmount
  - sudo umount -l <your_mount_point>
- Do not use external library to parse tar files; parse on your own!
- Do not untar files from test.tar and copy them to the mount point… this is cheating!!!

Testing OS Environment

- Ubuntu 22.04+
- Physical installation, VM, or WSL

Credits

- 吳雅柔 吳宥毅 helped design this assignment
- Questions should be directed to the current TAs