

# Homework1

**System programming 2016**

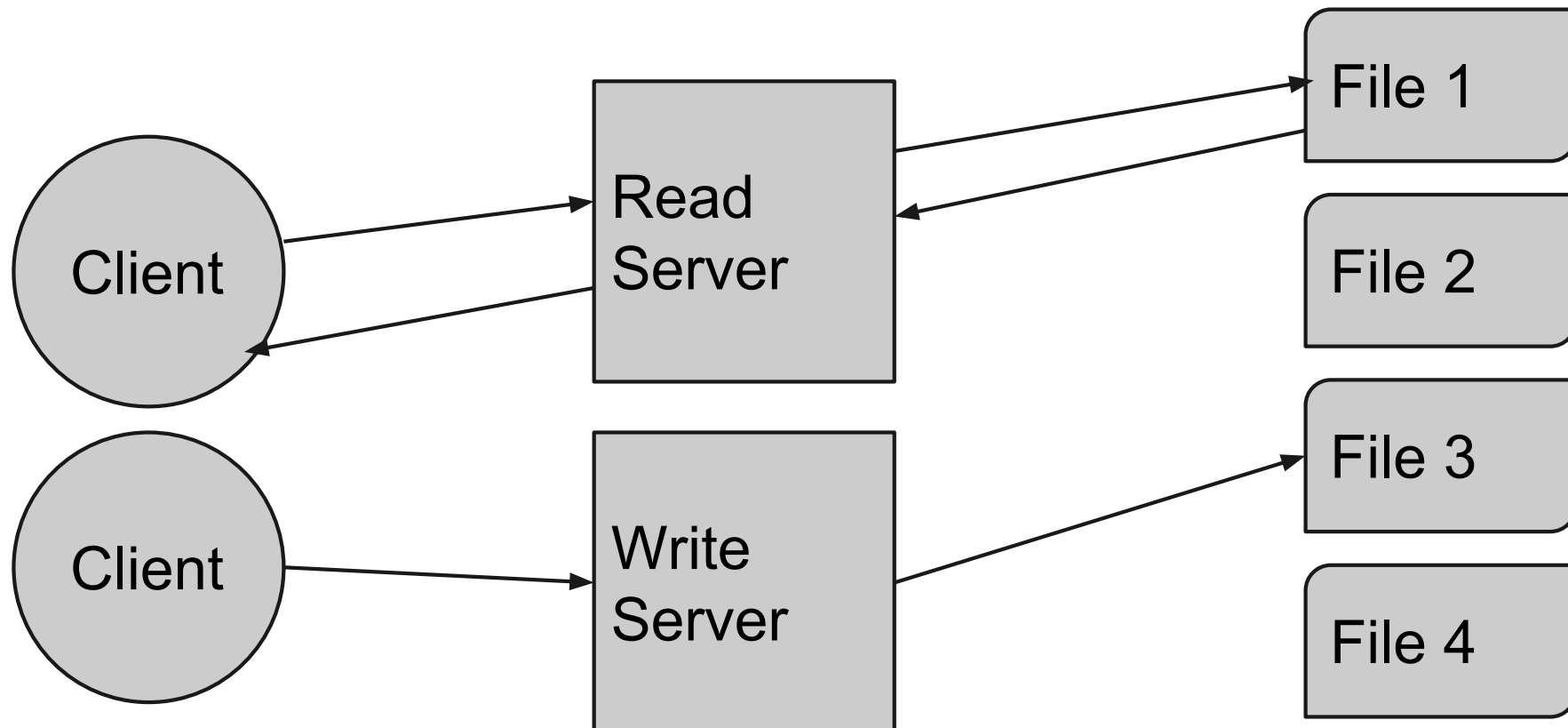
# Introduction

In this assignment, you have to modify a simple read/write Server, implement I/O multiplexing and file locking on it.

You don't have to handle the network communication. We will handle that part.

# We will provide

A Simple Read And Write Server  
Handle only one request at a time.



# DEMO

## Server:

```
~/SP$ ./read_server 4000

starting on      , port 4000, fd 3, maxconn 1024...
getting a new request... fd 4 from 127.0.0.1
Opening file [sp_sample_file.txt]
Done reading file [sp_sample_file.txt]
getting a new request... fd 4 from 127.0.0.1
Opening file [Makefile]
Done reading file [Makefile]
```

Inputs (request from client) are colored purple, to distinguish from the response from server.

## Client:

```
~/SP$ telnet 127.0.0.1 4000
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
sp_sample_file.txt
ACCEPT
sp_sample_content
Connection closed by foreign host.
~/SP$ telnet 127.0.0.1 4000
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
Makefile
ACCEPT
all: server.c
    gcc server.c -o write_server
    gcc server.c -D READ_SERVER -o read_server

clean:
    rm -f read_server write_server
Connection closed by foreign host.
```

# Your job is

A Multiplexing Read and Write Server

Handle multiple requests simultaneously.  
Lock files when doing operations on them.

file des	0	1	2	3	4	5	6	7	8
flag	1	1	1	1	0	0	0	0	0

```
typedef struct {
    char host[512];
    int conn_fd;
    char buf[512];
    size_t buf_len;
    char* filename;
    int header_done;
} request;
```

requestP	3	4	5	6
conn_fd	3	4	5	6



`server.listen_fd`



Any new incoming clients after accept() will be placed here

**Use a requestP table to simulate the file descriptor table!!**

# What should you do?

`select()`

Allows a program to monitor multiple file descriptors.

Waits until one or more of the file descriptors become “ready” for some class of I/O operation.

If we have a nonblocking descriptor that we want to read from, we call `select()` with timeout value of, say, 5 sec, and `select()` will block for up to 5 seconds.

`fcntl()`

Allows a program to obtain a lock for a file.

# Detail

You can download the files and spec from:

1. CEIBA
2. Facebook group: System Programming 2016



# How can I get full score

1. `read_server` returns the file content correctly.
2. `write_server` reads the request and writes to the file correctly.
3. Two requests issued to `read_server`.
4. Two requests issued to `write_server`.
5. Requests to `read_server` **and** `write_server` at the same time.
6. Multiple request to `read_server` and `write_server`.

# Example

## Server:

```
~/SP$ ./write_server 4000
starting on      , port 4000, fd 3, maxconn 1024...
getting a new request... fd 4 from 127.0.0.1
Opening file [file1]
getting a new request... fd 5 from 127.0.0.1
Opening file [file2]
Done writing file [file2]
getting a new request... fd 6 from 127.0.0.1
[file1] is locked, request rejected.
Done writing file [file1]
```

Inputs (request from client) are colored purple, to distinguish from the response from server.

## Clients:

```
~/SP$ telnet 127.0.0.1 4000
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
file1
ACCEPT
content of file 1
^]
telnet> quit
Connection closed.
```

```
~/SP$ telnet 127.0.0.1 4000
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
file2
ACCEPT
content of file 2
^]
telnet> quit
Connection closed.
```

```
~/SP$ telnet 127.0.0.1 4000
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
file1
REJECT
Connection closed by foreign host.
```

# Pseudo code

```
while (1) {  
    use select() to find which file descriptors are ready.  
    accept() new connections if there's any.  
    for each request with file descriptor ready {  
        use handle_read to read from the request.  
        if this is the first time read of the request {  
            check lock and obtain lock for the file, reject when can't lock.  
            responds to client that the request is accepted.  
            open the file descriptor for the file.  
        }  
        if is write server {  
            read from the request, and write it to the file.  
        } else {  
            get part of the file to buffer, and responds them back to client.  
        }  
    }  
}
```

# Submission

1. `server.c` (as well as all other `*.c`)
2. `Makefile`
3. `README.txt`

You should put files in a directory named with your student id, and tar the directory.

If your name and student ID is b05902000

The tared file will be named `SPHW1_b05902000.tar.gz`

please do not use `.rar` or other file type.

Submit your program before **10/25 23:59**

# Question

Ptt2: SysProgram

or

Come to lab302

or

Write an e-mail (Prefix your e-mail title with [SP])

or

Facebook group: System Programming 2016

Check if your question has been asked in P2 before  
you come or email!