Design Document: Asg1

Jinghao Shen
CruzID: jshen30

1 Goals

The goal for this assignment is to implement a simple single-threaded RPC server that will support the math functions and file functions. The server will respond to a standard RPC protocol, which is given, providing the results of several file system functions(read, write, list, unlink, status) that the server will implement. The program will run the server in a directory, and requests for files will be served from under that directory.

2 Design

This program first set up sockets for basic client and server communication.
Get host name and port from arguments.

```
uint16_t function
read(socket, recvBuf, 0)
Store the first 2 bytes in function (same way as get an uint16_t from buffer)
Store the next four bytes in the return buffer as identifier
```

The server is in a while loop which the condition is always true. The server can only be manually closed.
Depending on the function called retrieve the needed bytes.
Then, convert values to and from the given big-endian input, wire format, to a data that can be stored in five different types of buffer, uint8_t, uint16_t, uint32_t, uint64_t, and uint8_t *, using shift.
Do several unit tests on this part.

For uint:

```
uint8_t recvBuf[]
uint(8/16/32/64)_t variable
for (from startIndex to bytesNeed)
        variable = variable << 8 | recvBuf[index]
```

For words:

```
Get the first 2 bytes (same way as get an uint16_t from buffer)
Count = length of the string
for(from startIndex to count)
        uint8_t *str = str << 8 | recvBuf[index]
```

Based on the function call, the program uses “switch” to call different functions.

To do the math functions:

```
function(buffer[] without function call and identifier, operator, ifError){
    a = read 8 bytes from the buffer
    b = read another 8 bytes from the buffer
    if(overflow)
        ifError = 22;
    else
        ifError = 0;
    return a+b / a-b / a*b
```

To do the file functions:

Read/Write:

```
function(buffer[] without function call and identifier, read/write, ifError){
    Filename = get size of file name, then get file name
    Offset = next 8 bytes as offset
    bufSize = next 2 bytes as bufSize
    fd = open(file)
    lseek(fd, offset, SEEK_SET)
    if(read)
        res = read(buffer, size)
    if(write)
        Buffer = read next bufSize length of bytes
        res = write(buffer,size)
    return res
```

Create/FileSize:

```
function(buffer[] without function call and identifier, create/size, ifError){
    Filename = get size of file name, then get file name
    if(create)
        creat(filename, O_CREAT | O_EXCL)
        ifError = 17 if file already exists
    if(size)
        struct stat sb;
        stat(filename, &sb)
        ifError = 2 if no such file
        return sb.st_size
```

Store the return value from the function

```
sendBuf = identifier | ifError | result
```

Write back to the client.

Read after write back, if there's more bytes read, repeat.