Haoyang Yuan
haoyangy
15-640 / PIT
02/05/2016

Report for Project1

Serialization

According to the TA post at Piazza, I would like to use two struct for client request serialization and send them each once. A first header structure with constant size will contains the message length and the operation type {opcode, total length}. The server will continue to receive this kind of header packet. The second header, which should contains all the parameters of operation {open for example: flag, mode, file_len, file_name}. If the parameter is a string, the last member of structure is defined as "unsigned data char[0]", which has a length of 0 and will points to the following string.

```
typedef struct
{  int opcode;
   int total_len;
   unsigned char data[0];
} request_header_t;
```

```
typedef struct
        {int filename_len;
unsigned char data[0];
} gettree_request_header_t;
```

Deserialization

The Server will first gets the header packet to know how many space is needed to malloc for the next step. And the Server will create the space with a specific operation structure pointer. Thus it can use structure member to invoke all the receiving data with their correct relative positions. For the client, it will use a constant space for some functions which will only return value and errno, and for other complicated return data, it will do the job as previous.

Tree Serialization and Deserialization

I will use BFS to traverse the tree structure. And do the following serialization for each node. I will use a queue to store each node, and use BFS to get the sons of each node. I will add sons to the queue, whenever I pop out a node. And for each node I will store the {length of name, name, number of sons}, in the serialization process to map them in to a string for transmit. The descendent relationship is contained in the queue sequence.

When deserialization from the string, I will set up a queue first. Since I can read the first int to get the length of the name, and then read out the name, the next int should be the number of sons. Thus I can create a new node to simulate the original node on server. Since I know the number of sons, I should know how many nodes from now on in the string are the sons of first one. So I create the empty nodes and set up the descendent relationship with the previous node as sons. Finally I put them in the queue, to wait for pop out and fill the the information from Transmitted string. After all the loops while no more nodes in the queue, the first node should be the root of all tree structure.