client -> server int sizeclient -> server int sizeclient -> server int sizeclient -> server int sizeclient -> server int sizeint opcode - 0 int flag mode_t mode string pathint fd size_t count server -> client int sizeint fd size_t countint fd size_t countserver -> client int sizeserver -> client int sizeserver -> client int sizeint opcode - 0 int return int returnint opcode - 2 size_t return int errnosize_t return int opcode - 2 ssize_t return int errno	
int opcode - 0int opcode - 1int opcode - 2int opcode - 2int flagint fdint fdint fdmode_t modeserver -> clientsize_t countstring pathserver -> clientstring bufint sizeint opcode - 1server -> clientint sizeint returnint sizeint opcode - 3int opcode - 0int errnoint opcode - 2ssize_t returnint returnssize_t returnvalue	
int flag mode_t mode string pathint fdint fdint fdstring pathserver -> client int sizestring bufserver -> client int sizeserver -> client int opcode - 1server -> client int sizeint size int opcode - 0 int returnint opcode - 2 sit errnoint opcode - 2 ssize_t return	
mode_t mode string pathserver -> client int sizesize_t count string bufsize_t count string bufserver -> client int sizeserver -> client int opcode - 1 int opcode - 2 int opcode - 2 size_t returnint opcode - 3 size_t return	
string pathserver -> client int sizestring bufserver -> client int sizeint opcode - 1 int returnserver -> client int sizeint size int opcode - 3 int opcode - 2 ssize_t return	
int size server -> client int opcode - 1 server -> client int size int opcode - 3 int opcode - 0 int return int return int return ssize_t return value	
server -> clientint opcode - 1server -> clientint sizeint sizeint returnint sizeint opcode - 3int opcode - 0int errnoint opcode - 2ssize_t returnint returnssize_t returnvalue	
int sizeint returnint sizeint opcode - 3int opcode - 0int errnoint opcode - 2ssize_t returnint returnssize_t returnvalue	
int opcode - 0 int errno int opcode - 2 ssize_t return value	
int return ssize_t return value	
int errno int errno int errno	
string buf	
Iseek _xstat unlink getdirentries	
client -> server   client -> server   client -> server   client -> server	
int size int size int size	
int opcode - 4 int opcode - 5 int opcode - 6 int opcode - 7	
int fildes int var string pathname int fd	
off_t offset struct stat size_t nbytes	
int whence string path server -> client off_t basep	
int size	
server -> client   server -> client   int opcode - 6   server -> client	
int size int return int size	
int opcode - 4 int opcode - 5 int errno int opcode - 7	
off_t return int return ssize_t return	
int errno int error int errno	
getdirtree freedirtree	
client -> server client -> server	
int size int size	
int opcode - 8 int opcode 9	
string path string path	
int number	
server -> client string path	
size t size int number	
int opcode - 8	
int success	
int errno	
string path server -> client	
int number int size	
string path int opcode 9	
int number int success	
int errno	

How to avoid timeout caused by recv() and send()

My implementation: the first four bytes of request and response is an int, which represent the length of this request/response. I check return value of each recv() and send() and use loop in order to receive the exact value.

How to discriminate local file operation and remote file operation For remote file operation, all fd will be added an offset of 512. So for fd within[0, 512], it is considered as local file. For fd within [512, 1024], it will considered as remote file.

How to deal with bad file descripter

For remote file fd, I use a fd\_set to store fd info. When an fd is open or close, I use FD\_SET and FD\_CLR to update its state. When a client tries to operate on invalid file descripter. It will return -1 and set errno directly.