Design Document

<u>Andrew ID: hkhalid</u>

Serialization of data:

I used rpc_header and rpc_body structs with the size of the body in the rpc_header. For the RPCs on the client side, I packed the version number, header, operation code, flags, and body size in the rpc_header. Then I packed all data in the rpc_body to send to the server. The server first receives the rpc_header and then receives the rpc_body up to the body size defined in the rpc_header.

Serialization with structs made it easy to deal with the data and deserialize it on the server side. If in the future, we plan to add one more field to either the header or body, we can simply increase the version number and add new fields to the following structs without making any major changes to the code, making the code scalable.

```
typedef struct {
  unsigned int version;
  int code;
  int flags;
  int body_size;

} rpc_header;

typedef struct {
  int in;
  int inout;
  int out;
  off_t offset;
  char data[];
} rpc_body;
```

Design choices and decisions:

1. I added FD_BIAS (file descriptor bias) to the remote file descriptors while dealing with them on the client side. It helped prevent unwanted closing of the file descriptors and distinguished the local file descriptors from the remote file descriptors.
2. To minimize latency, I turned off Nagle's algorithm.
3. To handle multiple clients, I used forking as mentioned during the recitations.