Practical Work 2: RPC File Transfer

Distributed Systems - DS2026

Nguyen Viet Hung
Student ID: 23BI14188

December 12, 2025

1 RPC Service Design

Our file transfer service is designed based on the **Sun RPC (ONC RPC)** model using a request-response pattern. Instead of using raw sockets, the client invokes a remote procedure named `READ_FILE`.

The design flow is as follows:

1. The **Client** calls the local stub function with the filename, offset, and chunk size.
2. The **Client Stub** marshals these parameters into a standard format (XDR) and sends them over the network (UDP).
3. The **Server Stub** receives the request, unmarshals the parameters, and calls the actual server implementation.
4. The **Server** reads the requested file chunk and returns it.

2 System Organization

The system is organized into three main components: Interface Definition, Client Side, and Server Side. The build process is automated using a `Makefile`.

- **IDL (Interface Definition Language):**
  - `transfer.x`: Defines the `FileChunk` structure and the `READ_FILE` procedure.
- **Generated Files (by rpcgen):**

  - `transfer.h`: Header file containing struct definitions.
  - `transfer_clnt.c`: Client stub implementation.
  - `transfer_svc.c`: Server stub (main loop) implementation.
  - `transfer_xdr.c`: Data conversion logic (XDR).
- **Application Logic:**
  - `client.c`: Handles command line arguments, loops to request file chunks, and writes to local disk.
  - `server.c`: Implements the file reading logic using `fseek` and `fread`.

3 Implementation

3.1 Interface Definition (transfer.x)

We defined a `Params` struct to pass multiple arguments (filename, offset, size) since RPC typically accepts a single argument.

```
struct FileChunk {
    opaque data<>;       /* File content */
    int bytes_read;      /* Actual bytes read */
};

struct Params {
    string filename<>;
    int offset;
    int size;
};

program FILE_TRANSFER_PROG {
    version FILE_TRANSFER_VERS {
        FileChunk READ_FILE(Params) = 1;
    } = 1;
} = 0x31230000;
```

3.2 Server Implementation (server.c)

The server uses `fseek` to jump to the correct offset and `malloc` to allocate memory for the response.

```
FileChunk * read_file_1_svc(Params *argp, struct svc_req *rqstp) {
    static FileChunk result;
    FILE *f = fopen(argp->filename, "rb");

```

```
    // ... Error handling omitted ...

    fseek(f, argp->offset, SEEK_SET);

    // Allocate memory for response (dynamic allocation)
    char *mem_buffer = (char *)malloc(4096);
    int bytes = fread(mem_buffer, 1, argp->size, f);
    fclose(f);

    result.data.data_val = mem_buffer;
    result.data.data_len = bytes;
    result.bytes_read = bytes;

    return &result;
}
```

3.3 Client Implementation (client.c)

The client requests data in a loop until the bytes read are 0 (EOF).

```
while (is_running) {
    result = read_file_1(&args, clnt); // RPC Call

    if (result->bytes_read > 0) {
        fwrite(result->data.data_val, 1, result->data.data_len,
   f_dest);
        args.offset += result->bytes_read; // Update offset
        printf("Received %d bytes...\n", args.offset);
    } else {
        is_running = 0; // End of file
    }
}
```

4 Task Distribution

- **Nguyen Viet Hung (23BI14188):**
  - Designed the RPC interface (.x file).
  - Implemented Server and Client logic.
  - Fixed memory management bugs (malloc/free).
  - Wrote the report.