Design and Implementation of A System for Remote File Access

1. OBJECTIVE

This project consolidates the basic knowledge about interprocess communication, remote invocation, and distributed file systems. Through this project, the students will gain hands-on experience in constructing client and server programs that use UDP as the transport protocol.

2. INTRODUCTION

This project is to be done in groups of **at most four students (the responsibility should be clearly divided among group members)**. The objective of the project is to design and implement a system for remote file access. This manual describes the required functions of the system. You should consult the lecture notes and/or the textbook while doing the project. You are required to demonstrate working programs and to submit a lab report and well-commented source code at the end of the project.

3. EXPERIMENT

3.1 Preliminaries

The program you are asked to write for this project uses UDP (not TCP) sockets. In addition to the lecture notes, you may also refer to the online notes on socket programming in the NTULearn course site.

3.2 System Description

In this project, you are asked to design and implement a system for remote file access based on client-server architecture. The files are stored on the local disk of the server. The server program implements a set of services for clients to remotely access the files. Meanwhile, the client program provides an interface for users to invoke these services. On receiving a request input from the user, the client sends the request to the server. The server performs the requested service and returns the result to the client. The client then presents the result on the console to the user. The client-server communication is carried out using UDP.

The services to be implemented by the server program and made available to the users through the client program include:

1. A service that allows a user to read the content of a file by specifying the file pathname, an offset (in bytes), and the number of bytes to read from the file. The service returns the given number of bytes of the file content starting from the designated offset in the file (the offset of the first byte in the file is 0). For example, if 2 bytes are read from the file content “abcd” at offset 1, the service returns “bc”. If the file does not exist on the server or if the offset exceeds the file length, an error message should be returned.

2. A service that allows a user to insert content into a file by specifying the file pathname, an offset (in bytes), and a sequence of bytes to write into the file. The service inserts the sequence of bytes into the file at the designated offset in the file. The original content of the file after the offset is pushed forward. For example, if “b” is inserted into the file content “acd” at offset 1, the updated file content becomes “abcd”. The service returns an acknowledgement to the client on successful insertion. If the file does not exist on the server or if the offset exceeds the file length, an error message should be returned.

3. A service that allows a user to monitor updates made to the content of a specified file at the server for a designated time period called monitor interval. To register, the client provides the file pathname and the length of monitor interval to the server. After registration, the Internet address and the port number of the client are recorded by the server. During the monitoring interval, every time an update is made by any client to the content of the file, the updated file content is sent by the server to the registered client(s) through callback. After the expiration of the monitor interval, the client record is removed from the server which will no longer deliver the file content to the client. For simplicity, you may assume that the user that has issued a register request for monitoring is blocked from inputting any new request until the monitor interval expires, i.e., the client simply waits for the updates from the server during the monitoring interval. As a result, you do not have to use multiple threads at a client. However, your implementation should allow multiple clients to monitor updates to a file concurrently.

In addition to the above services, you are required design and implement two more operations on the files through client-server communication. One of them should be idempotent and the other should be non-idempotent. Describe your design in the report.

You are also asked to implement client-side caching in the system, i.e., the file content read by the client is retained in a buffer of the client program. For simplicity, you may assume that the cache capacity is sufficiently large (so, no cache replacement algorithm needs to be implemented). The cache is used to speed up read operations. If the file content requested by the client is available in the cache, the client may read the cached content directly without contacting the server. On the other hand, the updates made by the client to file content are always sent to the server immediately. The approximate one-copy update semantics (used by NFS as described in the lecture) is required to be implemented to maintain cache consistency. The freshness interval *t* is specified as an argument in the command that starts the client.

3.3 Requirements and Hints

In this project, you are expected to design request/reply message formats, fully implement marshalling/unmarshalling and the fault-tolerance measures by yourself. You may use a byte array to store the marshaled data. Do NOT use any existing RMI, RPC, CORBA, Java object serialization facilities and input/output stream classes in Java.

You are required to write a server program and a client program using the system calls in UDP socket programming only. To enable the client to communicate with the server, you can assume that the server address and port number are known to the client (e.g., the server address and port number may be specified as arguments in the command that starts the client). The IP address of the computer can be obtained with the “ipconfig” command in Command Prompt. On the other hand, the server does not know the client address in advance. The client address is obtained by the server when it receives a request from a client. There can be multiple clients running concurrently on different computers in the system. For simplicity, you may assume that the requests made by the clients are well separated in time so that before finishing processing a request, the server will not receive any new request from any client. As a result, you do not have to create a new thread at the server to serve each request received.

The client provides an interface that repeatedly asks the user to enter a request and sends the request to the server. Each reply or error message returned from the server should be printed on the screen. The interface provided by the client should include an option for the user to terminate the client. For simplicity, you may design a text-based interface on the console, i.e., a graphical user interface is not necessary.

The server repeatedly receives requests from the client, performs the service and replies to the client. The received requests and the produced replies should be printed on the screen. **The files available for remote access must be stored on the local disk of the server computer.**

Based on the above system description, **you are required to design the structures for request and reply messages, and describe your design in the report**. Note that different operations may need different types of arguments and return different types of results. All messages transmitted between the server and the clients must be in the form of a sequence of bytes. Therefore, **you must marshal the integer values, strings etc. before transmission and unmarshal them upon receipt**. Note that the strings (e.g., the file pathnames, the content to insert into files) need to be marshaled because their lengths are not fixed and may vary from request to request. You may consider adding the length information of the string in marshaled messages. The following functions in C are useful in marshalling and unmarshalling:

```
#include <netinet/in.h>
uint32_t htonl(uint32_t hostlong);
uint32_t ntohl(uint32_t netlong);
```

The htonl() function converts the unsigned integer hostlong from host byte order to network byte order. The ntohl() function converts the unsigned integer netlong from network byte order to host byte order. The network byte order, as used in the Internet, is Most Significant Byte first, whereas the host byte order depends on the specific architecture of the host.

To implement client-side caching, you will need to maintain some meta-information of cached content (e.g., the file pathname of the cached content, the time when the cached content was last validated). In cache consistency maintenance, the following functions in C may be useful for checking the last modification times of the files:

```
#include <sys/stat.h>
int stat(char *path, struct stat *buf);
int fstat(int fd, struct stat *buf);
```

You are required to implement the system with two different invocation semantics: at-least-once and at-most-once. To do so, you will need to implement techniques like timeouts, filtering duplicate request messages, and maintaining histories. You can consult the textbook and the lecture notes for the details of invocation semantics and associated techniques. Which semantics to use may be specified as an argument in the command that starts the client/server. You can assign a request identifier to each request for detecting duplicate requests. **As part of this project, you should simulate the loss of request and reply messages in your program, and design experiments to compare the two invocation semantics**. Show that at-least-once invocation semantics can lead to wrong results for non-idempotent operations, while at-most-once invocation semantics work correctly for all operations. **Describe your experiments and discuss the results in the report.**

The basic requirement of this project is to write both the client and server programs in either C/C++, Java, or any other language supporting inter-process communication. An additional but optional requirement is to implement the client and the server in different programming languages – for example, one in C/C++ and one in Java. In this case, you may need to take care of different data representations in different programming languages.

You may work on your own laptops and use your laptops for demonstration at the end of the project.

For the remaining design and implementation details, make your own reasonable assumptions and decisions.

4. REPORT AND DEMONSTRATION

You are required to submit a lab report and well-commented source code at the end of the project. The submission deadline is October 23, 2023 (Monday). Late submission will not be accepted.

The source code should be submitted electronically through the assignment module on the course website in NTULearn. Please take note of the following important notices: (1) the assignment module accepts one file only, so you will need to pack all your source programs; (2) the assignment module does not allow resubmission, so please submit the source code only when the final version is ready; (3) for each group, only one representative of the group members needs to submit the source code (please do NOT submit a copy of the source code from each member in the group); (4) in case there is any unexpected problem in the NTULearn system, please try submission again after a while. **The source code is to be submitted before the midnight of the deadline date. Please avoid submission at the last minute to prevent overloading the NTULearn server.**

A lab report is required from each group. On the cover of your report, please indicate the names of all group members (as on your matriculation cards) and the percentage of work each member did in the project. In the report, clearly explain your design and implementation strategies. **The report should also be submitted electronically through the assignment module on the course website in NTULearn.**

You will be required to demonstrate working programs where the clients and server run on different computers. To demonstrate the approximate one-copy update semantics and the callback, besides the server, you will need to run two clients on two different computers, among which one client updates a file and the other client reads/monitors the content of the file. **The demonstration will be conducted during the class on October 26, 2023 (Thursday).** The schedule will be arranged later.

Important Notice: Please be reminded that PLAGIARISM (copying part of or complete programs) is strictly prohibited. You will get zero mark for the course project if you copy the programs from others or give your programs to others for copy.