

# Program 3: Distributed Media Server (mserv3)

## CS 4210 Advanced Operating Systems \* Fall 99 Hutto

### Groups of 3!

#### DUE:

Group formation (send email to [pwh@cc.gatech.edu](mailto:pwh@cc.gatech.edu)): 7pm Thursday, November 18<sup>th</sup>

Design presentation: Tuesday 23 November 99

**Final turn-in: 3pm (classtime) Thursday, December 9<sup>th</sup>, 1999**

#### HELP:

RPC Help session: 7pm Thursday, November 18<sup>th</sup> in CCB 102

*Questions? Office hours, email, newsgroup, ...*

The final programming assignment of the semester will implement a distributed media server system, with multiple clients and suppliers, each implemented as a separate processes running on more than one machine. It is highly recommended that you use your project 2 code as a starting point for this program.

You may work in teams of 3 for this project. You have until 3 pm on December 9<sup>th</sup> to complete the program. There will be a detailed design check on Tuesday 23 November during class. Each team should hand in a detailed design document (about 3 pages) and do a brief (5 to 10 minutes) class presentation.

There are three components to this project: a name server, a media server, and a client/supplier. Each are detailed below:

## The Name Server

This project will require the use of a **single-threaded** (serial) name server that you will write. The name server will maintain a registry of currently available media servers and the range of titles that they will serve. The server and client/supplier processes should know how to contact the name server. When a media server starts up, it will contact the name server and register itself. The procedures for receiving or transmitting a media title by the server in project 2 will serve as the entry point procedures for clients and suppliers to access the store of media files (See **man rpc** for more information and all the related calls to register entry points with the RPC package).

The name server will need to keep its state saved in a file. When the name server first boots up, it should check for this file, and if it exists, restore its state to the saved one. This is a simple crash-recovery mechanism in the case that the name server crashes for a few minutes.

The name server will support the following calls:

- **Register** - Registers a media server with the name server. When a media server registers, it must tell the name server what range of media titles it will serve (ex. titles starting with A through J). The registering media server will also need to tell the name server the host that it's running on and its instance (as you should be able to have more than one instance of a media server running on one host).
- **Unregister** - Unregisters a media server with the name server. Media servers should call this before they go to shutdown.
- **Lookup** - The client/supplier processes will need to call the name server's lookup function to find a media server that is serving the title they're requesting. If more than one media server exists for the requested title, the name server should return a list of all the applicable servers.
- **Shutdown** - Shuts down the name server and terminates the process.

## The Media Server

The media server will still serve the same three media types (AUDIO, GRAPHICS, and VIDEO). The server process is **multithreaded** (using the multithreaded option of genrpc), and will interact with the client/supplier and name server through remote procedure calls (RPCs). The media server will run continuously until manually told to stop.

Each media server will be responsible for a different portion of the store of media titles (ex. Server 1 handles titles A-J, server 2 handles K-P, and server 3 handles Q-Z) and will be told what portion based upon its command file. It is possible to have media servers with overlapping or the same name space. Each media server will have its own commands file (ex. server1.cmds, server2.cmds, etc,...). The command-line argument used when starting the server will specify the command file to be read. Requests made to a media server for a media title that it isn't responsible for should be handled in a graceful way and generate appropriate error messages. The media server should store the title, type, size, and number of copies for each title in its collection. Media titles shouldn't be listed or made available to clients until they have been fully received from a supplier.

The media server will support the following calls:

- **List** – Returns a list of all the media titles that the media server has in its store. It should also return the size and type of each title.
- **Checkout** – Checks-out a media file and serves it to the requesting client. The media file should be returned in its entirety inside the return structure from the call. The media server will need to keep track of the number of available copies of a title and not allow more than the number it has to be checked out.
- **Checkin** – Effectively releases the hold that a client has on a media file, allowing another the media server to give the copy to someone else.
- **Add** – Adds a new media file to the server's collection. The entire media file should be supplied in the parameter structure to the call.
- **Shutdown** – Tells the media server to wrap up its actions, unregister itself with the name server and terminate.

## Server Sample Command File (server1.cmds)

```
start: a
end: j
nameserver: [machine name]
```

## The Client/Supplier

In this project, the client and supplier are integrated into one process. Client/supplier will be manually started on a machine, with a command-line argument for the command file they should read their actions from. The command file will be specific to the client process you manually start so that all clients aren't executing exactly the same actions. Client/suppliers will be **single threaded** in this project.

The command file will specify if the client/supplier is to act as a client or as a supplier. It will then be followed by the actions that it should execute. Requests will be made to the server by first calling the name server's lookup procedure to find the server instances that might contain the needed media title. If there's more than one media server for the desired name space, the client should try the servers one-by-one until it finds a media server with the desired media title. Once the proper server has been determined, a call will be made to that server's "checkout" procedure to perform the file transfer. If no server exists to serve the desired media title, an error should be logged and the request ignored. When the client is done using the media file, the "checkin" procedure of the media server who supplied the file should be called. If no media

server exists, or the title is not on any of the media servers returned by the name server, the client should handle it gracefully and produce proper error messages. Clients should return all checked-out media titles before terminating through the proper check-in procedure.

A similar method should be used for supplying media titles in that it should contact the name server, randomly pick one of the returned media server instances, and call that server's "add" procedure. Client actions should be logged to a client specific log file (ex. client1.log, client2.log, etc,...) The commands file should be named in a similar manner (ex. client1.cmds, client2.cmds, etc,...).

## Client Command File Sample #1 (clients1.cmds)

```
clientsupplier: client
nameserver: [machine name]
list audio
checkout graphics "hutto.jpg"
checkout audio "lecture10.au"
pause 100
checkin graphics "hutto.jpg"
checkout graphics "ccb101.jpg"
checkin audio "lecture10.au"
pause 200
checkin graphics "ccb110.jpg"
shutdown
```

## Client/Supplier Command File Sample #2 (clients2.cmds)

```
supplier
add graphics "hutto.jpg" 3
add audio "lecture10.au" 2
pause 200
add graphics "ccb110.jpg" 5
add [type] [title] [#copies]
shutdown
```

## Problems to Think About

- The absence of a name server should be handled gracefully by the client/suppliers and media servers.
- For "proper operation", there is an implied starting order to the different processes. A good start-up order would be: name server, media servers, suppliers, clients.

## References

- Stevens "Unix Network Programming: Vol 2 Second Edition" Chapter 16
- man rpc
- <http://docs.sun.com/ab2/coll.45.10/ONCDG/@Ab2TocView?Ab2Lang=C&Ab2Enc=iso-8859-1> (or go to docs.sun.com, then browse by collection, then Solaris 7 Software Development, then ONC+)

## Extra Credit (choose any of the following)

- Make the media server provide and receive media files in chunks rather than one big glob at a time. This way a client can tell the server to stop sending a file, or the server can abort receiving a file from a supplier.

- Make the client/supplier process multi-threaded, with a main thread reading the command file, spawning new threads to handle each request.
- You can implement a client-side cache for extra credit. The client processes will keep a cache of recently access media files on the local machine, and check to see if a new request is already available in the cache before requesting it from the server. The details of the cache design are up to you, but you should have a limited cache (don't just save everything you get and re-use it).