Design and Implementation of an Image Server

Part I: Simple Image Server

In this part of the assignment, you are required to design and implement an image server. The image server holds large number of images. Clients connect to the server and request image files by issuing the names of the desired files in a request string. You need to design and implement both the server and client in C or C++ and they should run a Linux machine (e.g., the lab machines).

The client is a simple program that launches requests serially from a randomly generated sequence or a sequence provided by the user at startup. The server needs to be concurrent because it needs to serve multiple clients. The server should use multiple threads such that each client request (once accepted is processed by a thread). The server would listen on a port for incoming client requests and it will handover the accepted connections to a worker thread. A worker thread can be chosen from a pool of pre-created worker threads (a bit complicated but gives better performance) or created on demand as the request arrives. It is necessary to set a maximum number of active threads so that the server can provide acceptable throughput for accepted connections. The server can reject client connections when the limit is reached so they can retry later for a connection.

To complete the first part, you need to implement the client and server and demonstrate that the server is able to serve up to 100 clients. You can use a batch script to launch many clients using the same program.

Part 2: Adaptive Image Server

In this part of the assignment, you are required to enhance the server to be adaptive. With modern touch-based mobile devices (e.g., tablets and smartphones), user activity (e.g., panning, tapping, and zooming) launches different forms of image download requests to the backend servers. The adaptive server you develop here will approximate a particular behavior in tablets (i.e., panning).

Consider the image downloading process in an online photo album software. New images are downloaded as the user pans through previously downloaded images. When the user is panning fast, the user is likely not interested in looking at all images or just trying to get an idea of what kind of images are available in the album.

For this assignment, the client is modified so that it reports the panning speed to the server. For simplicity, it is assumed to report the panning speed periodically. The image download requests are also sent from the client to the server. The server is expected to prioritize the requests because not all requests can be served by the server such that they are “useful” for the client. For instance, an image that arrives after the user has panned over the image location is not very useful. Because it is very hard to

know the exact location of the image and the usefulness of an image for rendering purposes, we can use a simplifying heuristic such as the following: when panning speed is high, give the requests lower priority. With priority based scheduling, we can have starvation. You need to design the scheduler such that it avoids starvation -- you could use a variation of the weighted fair scheduler we will study in Chapter 7.

Unlike the simple image server, the adaptive server connects each client using two sockets. One is for normal requests for downloading images. The other one is for the client to notify the server of the changes in panning speed -- this is priority information. Here is a suggested design for the adaptive image server. You are free to modify it.

The low priority socket is serviced by a thread dedicated to the client. The high priority sockets receive the panning speed information from the clients. These sockets are watched by a central scheduler. You could use epoll to watch all the sockets. Each active client is served by it own dedicated server thread. You need to design the scheduler such that it can still control the amount of throughput (number of images) served by the server for each client based on the priority computed from the panning speed. A high priority client (slow panning) can have all of it requests serviced by the server. A low priority client might have 1 out of N requests served by the server. The exact service ratio will depend on the overall load and fairness considerations.

In this part, you will create a design for the server architecture and then implement it.

Marking Scheme:

First part: 50 points
Second part: 50 points
Bonus: 25 points (for elegant scheduler design)