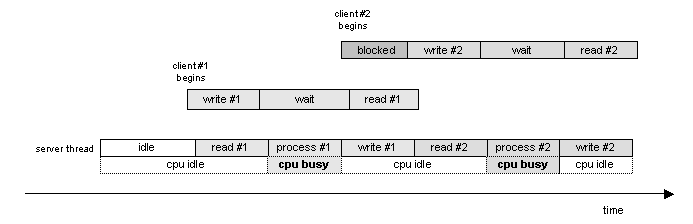
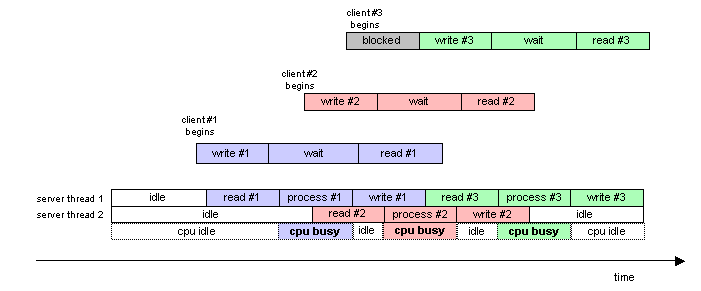
527326-13-12E AID: 258164| 27/06/2020

As I'm sure you know, the Internet has exploded in popularity over the past decade. Driving much of this explosion is the proliferation of networked servers, which are programs that run "in the guts" of the Internet. Client programs (such as web browsers) communicate with these servers over the network, sending requests and reading back the servers' responses. For example, a web browser will send a web server a request for a particular web page; the server will process this request, and then respond by sending back the page's HTML, which the web browser reads, parses, and displays.One of the challenges in building Internet services is dealing with concurrency. In particular, many independent clients can simultaneously send requests to a given server. A server therefore must be designed to deal with concurrent requests. There are many different strategies for doing this; we will explore two in this project.

A very simple (too simple, in fact) design strategy is to build the server as a single-threaded program. The server's single thread waits until a new request arrives, reads the request, processes the request, and then writes a response back to the originating client. In the meantime, if another request arrives while the server is processing the first, that other request is ignored until the first has been dealt with fully:



A more sophisticated strategy would be to build the server as a multithreaded program. When a task arrives at the server, a thread is dispatched to handle it. If multiple tasks arrive concurrently, then multiple server threads will execute concurrently. Each dispatched thread does exactly what the single-threaded server did previously; it reads the request, processes it, then writes a response:



As the picture above shows, even though request #2 arrives while request #1 is being processed by the first server thread, the second server thread is available to deal with request #2 concurrently. As a result, while the first server thread is processing request #1 (hence using the CPU), the second server thread can read request #2 over the network. Thus, network I/O and computation are overlapped. This has many beneficial effects: the CPU is more effectively utilized, the throughput of the server is increased, and the response time of the server (as seen by the clients) is decreased.

In this assignment, you will be given the source code to a very, very simple single-threaded networked server. Your goal is to convert this single-threaded server into a multithreaded server, by building a "thread pool" and integrating it into the server. A thread pool is an object that contains a fixed number of threads and supports two operations: dispatch, which causes one thread from the pool to wake up and enter a specified function, and destroy, which kills off all of the threads in the pool and cleans up any memory associated with the pool.