Pre-forked Web Server

IS F462: Network Programming

Assignment 2

Bir Anmol Singh
Guntaas Singh
K Vignesh

1 Objective

In the given problem, the goal is to implement a concurrent, connection-oriented web server handling HTTP requests based on pre-forking model. The server process creates and regulates a pool of child processes which wait for incoming connections on the same port. Incoming connection requests are scheduled among the processes in the pool by the kernel, which are then handled iteratively by each process.

The size of the process pool is of critical importance. If too small, clients experience large delays. If too large, system performance degrades on account of large memory and context switching overhead. Hence, the process pool size is adjusted dynamically on the basis of incoming traffic and user specified parameters.

2 Design

The parent process regulates the process pool on the basis of the number of idle processes in the pool. However, incoming connections are directly accepted by child processes. Hence, child processes need to communicate with the parent process and notify their status, whenever it changes. This can be accomplished through various available techniques for **inter-process communication**. As per the problem specification, we use UNIX domain sockets for this purpose.

The parent process stores information about each process in the pool and updates it on receiving messages from child processes. It needs to monitor file descriptors associated with sockets connected to all child processes. Similarly, each child process needs to monitor both - the listening socket (for incoming connections) and UNIX domain socket (for messages from the parent process). Thus, **I/O multiplexing** is required in both cases.

In order to limit the amount of memory that may be accidentally leaked, the number of connections that are (iteratively) serviced by a process is limited by a user-defined parameter. On reaching this limit, the process is terminated and a new process is added to the pool to take its place.

3 Implementation

Messages Since the parent process keeps track of all the processes in the pool, the messages exchanged between them only serve to indicate the occurrence of certain events, and so the protocol required is quite rudimentary.

1. "C" - Child to Parent - New Connection Accepted
2. "F" - Child to Parent - Connection Serviced, now Idle
3. "K" - Parent to Child - Kill Command (child should terminate)

UNIX Domain Sockets ***socketpair()*** is used to create a pair of sockets forming a bidirectional stream pipe for communication between the parent and child process, ahead of forking. Since no path names are bound to the sockets, the channel is not visible to other processes.

I/O Multiplexing ***epoll()*** is the most scalable option for monitoring a large number of file descriptors since it maintains the interest list in kernel space. Hence, we use the same for both - the parent and child processes.

Thundering Herd Problem For *epoll()*, when a wakeup event occurs and multiple epoll file descriptors are attached to the same target file, all of them receive an event. This means that on the arrival of a connection request on the listening socket, all the processes in the pool are woken up. One of them accepts the connection, and the rest go back to sleep. However, briefly, all of them contend for CPU time, which may lead to performance degradation over multiple occurrences.

This problem can be avoided using the **EPOLLEXCLUSIVE** event in conjunction with EPOLLIN, so that the wakeup event is only delivered to one of the processes. (It should be noted that this flag is only available since Linux 4.5.)

HTTP Capabilities On account of the focus on pre-forking, the web server implemented does not actually service HTTP requests, but replies with a dummy response. Support for persistent connections or pipelining of requests is also absent.