Linux Server Performance

Session 2005-2009

Project Advisor

Mr. Muhammad Ahmad Adnan

Submitted By

Muaz Ahmed Mian 050620-030

Sharjeel Aslam 050620-078

Department of Computer Science

University of Management and Technology

A report submitted to the

Department of Computer Science

in partial fulfillment of the requirement for the degree

Bachelor of Science

in

Computer Science

By

Muaz Ahmed Mian

Sharjeel Aslam

University of Management and Technology

August 22, 2009

Acknowledgements

First of all we would like to thank Almighty Allah without Whom we are nothing. We acknowledge the cooperation and help given by Mr. Muhammad A. Adnan of the Department of Computer Science, University of Management and Technology. He has been a constant source of guidance throughout the course of this project. We are also thankful to our friends and families whose silent support helped us to complete our project.

(Signed)

Muaz Ahmed Mian 050620-030

Sharjeel Aslam 050620-078

Date

August 22, 2009

List of Figures

Table of Contents

Chapter 1

Scheduling Strategies

1.1 Round Robin Scheduling Strategy

This algorithm schedules all processes/threads/requests by assigning time slices to each request in equal portions and in circular order, handling all requests without any priority. Round robin job scheduling may not be desirable if the numbers of the jobs or tasks are strongly varying. A process that produces large jobs would be favored over the other processes because that specific process is the reason new jobs of tasks are being created. This problem may be solved by time-sharing, i.e. by giving each job a time slot or quantum (its allowance of CPU time), and interrupt the job if it is not completed by then. The job is resumed next time, and a time slot is assigned to that process.

The distribution tends to fit in with the general interactive multiprogramming philosophy in which each of N threads/processes receives approximately 1/N time units of processing for every real time unit. For example if a processor has 10 seconds and 5 requesting processes, then according to round robin, each process, will have processor for 2 seconds, and then processor will move to next process, even if the first process is not completed, and will come to this process later. Using this scheduling policy, our web server will share processor time for requests from clients. Every request will be handled on equal number of bytes read.

Implementing round robin was a critical part of this project which was implemented on a multi-threading server. The thread-worker-pool strategy was used here. A thread-worker-pool strategy creates a fixed number of workers at the beginning of execution instead of creating a new thread each time a connection is made. Thread-worker-pool strategy is explained below.

1.1.1 Round Robin with Thread-Worker-Pool Strategy

A thread-worker-pool strategy creates a fixed number of workers at beginning of execution instead of creating a new thread each time a connection is made. In this project the implementation of round robin is with thread pool, i.e. we switch the processor between threads in a round robin fashion.

1.1.2 Advantages of Using Thread-Worker-Pool Strategy

Thread-worker-pool implementation has several advantages over thread-per-request implementation.

1. The cost of creating worker threads is incurred only at startup and does not grow with the number of requests serviced.

2. Thread-per-request implementations do not limit the number of simultaneous active requests, and the server could run out of file descriptors if requests come in rapid succession. Thread-worker-pool implementations limit the number of open file descriptors based on the number of workers.

3. Thread-worker-pool implementations impose natural limits on the number of simultaneous active requests; they are less likely to overload the server when a large number of requests come in.

1.2 First-Come-First-Served (FCFS)

It is one of the simplest algorithms used in computer scheduling policies and in daily life as well. This strategy handles requests as the server receives them, the name is self-explanatory that the request that comes first is served first and vice versa. In FCFS, the priority assigned to processes is in the order in which they request the server. Server has a limit on its child processes. If the number of requests exceeds that limit, the remaining requests are put into a queue, with priority assigned in the order of arrival. All requests are processed one by one. In this strategy, the processor moves to next process when the first one is completed so there is no sharing of processor.

1.2.1 First-Come-First-Served Server with Multi-Threading

A multi-threaded server using FCFS works simply. It creates a new thread to handle every new request. If there are multiple requests, it puts them into a queue and takes them out one by one.

1.2.2 First-Come-First-Served Server with Multi-Processing

In a Multi processing server, the main process creates a child process when it receives a new request and the child process handles the request. Similarly if multiple requests arrive it puts them in a queue.

Chapter 2

Web Server Design

2.1 Overview

A web server is a server that handles HTTP requests. This project requires building web servers based on different scheduling strategies; these strategies have already been discussed in chapter 1. Based on these strategies four web servers were build for this project:

1. Basic Web Server
2. FCFS Multiprocessing Web Server
3. FCFS Multithreading Web Server
4. Round Robin Web Server with Thread Pool Strategy

The designs of these web servers are discussed later in this chapter.

2.2 Handling HTTP Requests

The HTTP request that a browser sends is given in Figure 1,

GET /index.html HTTP/1.1 *Request line*

Date: Thu 20 May 2009 21:12:55 GMT

Connection: Close ……. *General Headers*

Figure 2.1 A sample HTTP request message

We only require the request line which gives us the file name, it ends with a line feed character '\n'. So when we accept a request we only accept till the first line feed which helps us avoid the unnecessary data.

2.2.1 Tokenizing the Request

To extract the proper filename we save the request line in an array and then apply the C function ***strtok()*** twice on the request line with space " " as token identifier. This process helps saving the proper filename requested by the user, concatenate the filename with our path of data server using ***strcat()*** C function and use the complete path to access and open the file and send it back.

2.3 Basic Web Server

The basic web server is a single process; it creates a listening socket on port 80 and then accepts a HTTP request via this port and saves the communication socket. It then accepts the request and tokenizes it; as described above, opens the file and sends the complete file and comes back to wait for another request. This server is executing FCFS scheduling algorithm. This web server is different from the rest of the web servers because it is not a multi-threaded or multi-processing web server. Figure 2.2 gives a flow chart of Basic Web Server

2.4 FCFS Multiprocessing Web Server

FCFS multiprocessing web server uses multiprocessing. It works similar to the basic web server, creates and opens a socket on port 80 and waits for a request. The only difference in it is when it accepts a new request. This web server will create a new process to handle the incoming HTTP requests and waits for the child process to exit before it can again wait for a new request. This is FCFS; it handles one request at a time. Figure 2.3 gives a flowchart of FCFS Multiprocessing web server.

2.5 FCFS Multithreading Web Server

FCFS multithreading web server uses multithreading. Its working is not much different from FCFS multiprocessing web server. The difference lies in handling a new request. For every new request it creates a new thread to handle the new HTTP request rather than creating a new process. Threads are light weight and work faster. The base thread after creating a child thread waits for it to finish and then carry on with its own execution. Figure 2.4 gives a flowchart of FCFS Multithreading web server.

2.6 Round Robin Web Server with Thread Pool Strategy

A pool of worker threads is created at the start of the process; each thread is assigned a task. In this project, round robin is used with the thread-worker-pool strategy. A pool of 128 threads is created, every thread will create and open a socket on port 80 and wait for an HTTP request when the thread is given its processor time it will check for a request in the queue, if no request is there it will pass the processor to the next thread. If a thread gets a request it will handle it. Round robin is being implemented between the threads; a thread which will have a request will send 2048 bytes and then block itself, passing the processor to the next thread where as a thread which does not have a request will simply check the queue for a request if no request is there it will simply block itself and pass the