Concurrent HTTP Server: User Manual

Inês Batista, 124877 & Maria Quinteiro, 124996 (P3G4)

1. Introduction

The Concurrent HTTP Server is a web server developed to demonstrate how a system can handle multiple client connections at the same time while serving static website content. It is intended to run on a Linux environment and is able to deliver files such as HTML documents, CSS stylesheets, images and small media assets to any browser or HTTP client that connects to it. This manual explains how to install, configure, run, monitor and troubleshoot the server in a manner accessible even to readers without experience in systems programming or networking.

1.1. Overview

At its core, the server listens on a specific network port of the machine and waits for incoming connections from web browsers or command-line clients. When a user attempts to access a web page by entering the server's address in the browser, the request is received, interpreted and matched to a file stored in a directory known as the web root. The server then sends the file back to the client and records details of the interaction in a log file. One of the most important characteristics of the server is that it can process many such requests at the same time, allowing several users to load web pages without causing delays for one another. This concurrency is managed internally through a pool of worker threads that divide the workload while the main thread continues to accept new connections. Although the internal mechanisms rely on typical systems programming constructs, users do not need to understand them in order to operate the server effectively.

1.2. System Requirements

To run the server, you need a reasonably modern Linux distribution equipped with a C compiler such as gcc, the make build tool and permission to run executables. The hardware requirements are minimal, and the server will run comfortably on most machines with at least a single processor core and a few hundred megabytes of memory. No additional external software, databases or libraries are required beyond what is included in the project folder and standard system tools.

2. Installation

2.1. Prerequisites

Before building the server, you must ensure that your Linux system has the necessary compilation tools. If you are unsure whether gcc or make is installed, running a simple command such as "gcc --version" or "make --version" in the terminal usually reveals their presence. If they are missing, the system's package manager can be used to install them quickly. It is also helpful to know how to navigate directories through the terminal since all installation and execution steps take place there.

```
sudo apt-get update
sudo apt-get install build-essential gcc make
```

2.2. Compilation

After extracting the project folder, open a terminal and change into that directory. The compilation process consists only of typing the command "make". The Makefile included in the project coordinates the entire build and automatically compiles all necessary source files. When the process completes successfully, a server executable is produced. This file is the program you will run.

Compilation Steps:

1. Open a terminal in the project root directory.
2. Clean previous builds (optional but recommended):

```
make clean
```

3. Compile the server (This will generate the server executable in the current directory):

```
make
```

Advanced Build Targets:

- make debug: Compiles with debug symbols and sanitizers (for development).
- make release: Compiles with optimizations (-O3) for maximum performance.

3. Configuration

3.1. Configuration Parameters

The server can be customized using a configuration file named `server.conf`. This file contains key-value pairs that control various aspects of the server's behavior. Below is a complete reference of all configurable parameters:

Parameter	Description	Default Value
PORT	TCP port to listen on	8080
NUM_WORKERS	Number of parallel worker processes	4
THREADS_PER_WORKER	Number of threads per worker	10
DOCUMENT_ROOT	Directory containing website files	./www
LOG_FILE	Path to the access log	./logs/access.log

CACHE_SIZE_MB	RAM limit for file cache (per worker)	10
TIMEOUT_SECONDS	Connection idle timeout	30
DEFAULT_FILE	Default file to serve when a directory is requested	index.html
MAX_QUEUE_SIZE	Maximum number of pending connections in the queue	100
CACHE_ENABLED	Enable cache (1 = Yes, 0 = No)	1
LOG_LEVEL	Log detail level (INFO or DEBUG)	INFO
LOG_ROTATE_SIZE_MB	Maximum log size before rotation (in MB)	10
SOCKET_BACKLOG	Pending connections backlog in the operating system	128
SOCKET_REUSEADDR	Allow quick server restart on the same port (1 = Yes)	1

3.2. Environment Variables

The server also accepts configuration through environment variables, which take precedence over values set in the configuration file. This allows you to override settings without modifying configuration files:

Environment Variable	Overrides Parameter
HTTP_PORT	PORT
HTTP_WORKERS	NUM_WORKERS
HTTP_THREADS	THREADS_PER_WORKER
HTTP_ROOT	DOCUMENT_ROOT
HTTP_TIMEOUT	TIMEOUT_SECONDS
HTTP_LOG_FILE	LOG_FILE
HTTP_CACHE_SIZE	CACHE_SIZE_MB

3.3. Command line options

For quick adjustments, the server supports command-line options that provide immediate configuration overrides:

Option	Description	Example
-c <file>	Load a custom configuration file	./server -c my.conf
-p <port>	Override the listening port	./server -p 9090
-d	Run in Daemon Mode (background process)	./server -d
-v	Enable Verbose Logging (debug output)	./server -v
-h	Show the help message	./server -h

4. Usage

4.1. Starting the Server

To start the server, run the executable from the terminal, either with default settings or with the additional parameter indicating the configuration file. When the server launches, it prints a small confirmation message indicating the port it is listening on and the directory from which it will serve files. At this point the server is fully operational, and any browser on the same network can access it by entering the machine's address followed by the port number.

4.2. Monitoring

The server includes a real-time web dashboard accessible at:
http://localhost:8080/dashboard.html (adjust the port if configured differently).

Dashboard Features:

- Active Connections: See how many users are currently connected
- Request Stats: Total requests, bytes transferred, and error rates (404/500)
- Cache Health: Monitor cache hit rates to verify performance
- Worker Status: View the status of each worker process
- System Resources: Monitor server resource usage

Log Monitoring:
While the server runs, it writes detailed information into its log file. These logs form the primary source of insight into the server's behavior. They show which clients connected, which files they requested, whether the server encountered any issues retrieving or sending those files and how long certain operations took. Additionally, the server periodically writes small statistical summaries to the log, indicating how many requests have been processed so far, how many connections are active at that moment and general indicators of performance such as average latency. Watching these logs in real time can be extremely helpful during testing. A common way to do this is by opening a second terminal window and using a command that continuously displays the latest log entries as they are written.

Live log monitoring example:

```
tail -f ./logs/access.log
```

4.3. Stopping the Server

When you want to stop the server, you can return to the terminal where it is running and interrupt it by pressing Ctrl+C. The server is designed to stop gracefully, which means that it stops accepting new connections while still allowing any active connections to finish. It then releases all used resources, closes the log file and shuts down cleanly. If the server was started in the background or you need to stop it from a different terminal, you can do so by sending a termination signal to the process using its process identifier:

```
# Find the server process ID
ps aux | grep server

# Send termination signal
kill <process_id>

# Force termination if needed
kill -9 <process_id>
```

5. Troubleshooting

Using the server is generally straightforward, but certain situations may arise. One common issue is that the server cannot start because the chosen port is already in use. This simply means another process is listening on that port, and you can either stop that process or choose a different port for your server. Another frequent problem occurs when files return as unavailable because the server does not have appropriate permission to read them. Correcting the file or directory permissions usually resolves the situation instantly. If a file is genuinely missing, you should check the web root directory to confirm that the filename matches exactly what the browser is requesting, taking into account that Linux distinguishes between uppercase and lowercase letters. If the server appears slow under heavy usage, increasing the number of worker threads or ensuring that the web root resides on faster storage often leads to visible improvements. Should the server ever be terminated abnormally and leave behind temporary resources, restarting it cleanly after checking for leftover processes is usually enough to restore normal operation.

Common Issues and Solutions:

1. Port already in use: Change the PORT value or use -p option with a different port
2. Permission denied: Ensure the server has read permissions for the document root and its files
3. Connection refused: Verify the server is running and the port is correct
4. High memory usage: Reduce CACHE_SIZE_MB or NUM_WORKERS values
5. Slow performance: Increase NUM_WORKERS or THREADS_PER_WORKER values
6. Log files not created: Check write permissions in the log directory