Scalable Server

Version 1.0

Eric Tsang, A00841554, 6D

|  |
| --- |
| Table of Contents [Table of Contents 1](#_Toc443006354)  [1 Multi-Threaded Server 2](#_Toc443006355)  [1.1 Finite State Machine: Main Thread 2](#_Toc443006356)  [1.2 Finite State Machine: Worker Thread 3](#_Toc443006357)  [1.3 Pseudocode: Main Thread 3](#_Toc443006358)  [1.4 Pseudocode: Worker Thread 3](#_Toc443006359)  [2 E-Poll & Select Server 4](#_Toc443006360)  [2.1 Finite State Machine: Parent Process 4](#_Toc443006361)  [2.2 Finite State Machine: Child Process 5](#_Toc443006362)  [2.3 Pseudocode: Parent Process 5](#_Toc443006363)  [2.4 Pseudocode: Child Process 5](#_Toc443006364)  [3 E-Poll Client 6](#_Toc443006365)  [3.1 Finite State Machine: Parent Process 6](#_Toc443006366)  [3.2 Finite State Machine: Child Process 7](#_Toc443006367)  [3.3 Pseudocode: Parent Process 7](#_Toc443006368)  [3.4 Pseudocode: Child Process 7](#_Toc443006369) |

# Multi-Threaded Server

The following are the finite state machines and pseudocode for the multi-threaded scalable server.

There is a main thread that continuously spawns worker threads to handle connections even before any connection requests are received. The goal is to lower the overhead of creating threads on demand.

## Finite State Machine: Main Thread



|  |  |  |
| --- | --- | --- |
| State Name | Description | Events |
| Configure | Main thread is validating and parsing acquired user input. | Success: user input passes validation, and is successfully parsed.  Invalid parameters: user input fails validation, or is not successfully parsed. |
| Usage | Main thread prints out usage message, specifying how to invoke the program. | Done: usage message is printed to the screen. |
| Setup | Main thread acquires resources to be shared by worker threads such as IPC mechanisms, the server socket and more. | Done: all shared system resources are acquired successfully. |
| Spawn | Main thread spawns new worker threads to accept and serve a pending, or future connections from the server socket. | Done: application has hit the cap of allowed threads pending to accept a new connection. |
| Wait | Main thread is suspended. | SIGINT: application receives the SIGINT signal.  New connection: a new connection is received and accepted. |
| Terminated | Main thread is terminated, and no longer executing. |  |

## Finite State Machine: Worker Thread



|  |  |  |
| --- | --- | --- |
| State Name | Description | Events |
| Setup | Worker thread is allocating memory, and parsing thread parameters. | Done: memory is allocated, and thread parameters are parsed. |
| Accept | Worker thread is waiting for a new connection request to accept. | Accepted: a new connection request is received on the server socket, and was accepted by the worker thread. |
| Serving | Worker thread is communicating with the peer, and echoing received data back. | Connection ends: the connection is terminated after termination is initialized by peer. |
| Terminated | Worker thread is terminated, and no longer executing. |  |

## Pseudocode: Main Thread

Main program point of entry; routine executed by the main thread of the application.

|  |
| --- |
| 1. Validate and parse user input |
| 1. If input is invalid, print usage and terminate application |
| 1. Create shared system resources such as synchronization mechanisms, and the server socket |
| 1. Continuously spawn new worker threads to handle new and existing connections, making sure there is at least some threads waiting for a new connection to arrive |

## Pseudocode: Worker Thread

Routine executed by spawned worker threads.

|  |
| --- |
| 1. Allocate all memory upfront, and perform any other heavy tasks before anything else happens to minimize future delays. |
| 1. Wait for a new connection request on the shared server socket, and accept the connection request. |
| 1. Read everything from the connection, and send it back immediately. If the connection is closed by remote, or an error occurs, end the thread. |

# E-Poll & Select Server

The following are finite state machines and pseudocode for the servers implemented using select and epoll. The interfaces of select and epoll are conceptually very similar, so they can use the same design.

The parent process gathers and parses user input, then spawns child processes, and passes the relevant arguments to each child process.

Child processes will accept new and handle existing connections that they have accepted from the shared server socket.

## Finite State Machine: Parent Process



|  |  |  |
| --- | --- | --- |
| State Name | Description | Events |
| Configure | Main process validates and parses acquired user input. | Done: input is validated and parsed successfully. |
| Usage | Main process prints out usage message, specifying how to invoke the program. | Done: usage message is printed to the screen. |
| Setup | Main process acquires resources to be shared by child processes such as IPC mechanisms, the server socket and more. | Done: all shared system resources are acquired successfully. |
| Fork | Main process forks off various child processes used to service remote clients. | Done: all child processes forked successfully. |
| Wait | Main process waits for SIGINT to be received. | SIGINT: SIGINT is received. |
| Terminated | Main process is terminated, and no longer executing. |  |

## Finite State Machine: Child Process



|  |  |  |
| --- | --- | --- |
| State Name | Description | Events |
| Setup | Child process acquires system resources, and sets signal handlers. | Done: system resources have been acquired and signal handlers are set. |
| Serving | Child process is going through event loops, accepting new connections and servicing existing ones. | SIGINT: the SIGINT signal is received. |
| Terminated | Child process is terminated, and no longer executing. |  |

## Pseudocode: Parent Process

Main program point of entry; routine executed by the main thread of the application.

|  |
| --- |
| 1. Validate and parse user input |
| 1. If input is invalid, print usage and terminate application |
| 1. Create shared system resources such as synchronization mechanisms, and the server socket |
| 1. Spawn user specified amount of processes to handle connections |
| 1. Wait for all child processes to terminate, or SIGINT |
| 1. Release acquired system resources then terminate |

## Pseudocode: Child Process

Routine executed by spawned child processes.

|  |
| --- |
| 1. Acquire appropriate system resources. |
| 1. Using either epoll or select, manage the server socket and all existing connections, echoing back received data from clients. |
| 1. When SIGINT is received, release system resources and terminate. |

# E-Poll Client

The following are finite state machines and pseudocode for the epoll client application.

The parent process gathers user input, then spawns child threads, and passes the relevant parameters to each child process.

Child processes will connect to the user specified server machine, and send user specified data to the server a user specified number of times, before disconnecting, and reconnecting.

## Finite State Machine: Parent Process



|  |  |  |
| --- | --- | --- |
| State Name | Description | Events |
| Configure | Main process validates and parses acquired user input. | Done: input is validated and parsed successfully. |
| Usage | Main process prints out usage message, specifying how to invoke the program. | Done: usage message is printed to the screen. |
| Setup | Main process sets up IPC mechanisms to communicate with child processes. | Done: all shared system resources are acquired successfully. |
| Fork | Main process forks off various child processes to connect to remote servers. | Done: all child processes forked successfully. |
| Wait | Main process waits for timeout to be met, or SIGINT to be received. | SIGINT: SIGINT is received, or timeout exceeded. |
| Teardown | Main process releases acquired system resources back to the system. | Done: system resources are successfully released. |
| Terminated | Main process is terminated, and no longer executing. |  |

## Finite State Machine: Child Process



|  |  |  |
| --- | --- | --- |
| State Name | Description | Events |
| Setup | Child process acquires system resources, and sets signal handlers. | Done: system resources have been acquired and signal handlers are set. |
| Connect | Child process is going through event loops, making new connections and maintaining existing ones. | SIGINT: the SIGINT signal is received. |
| Print Stats | Child process prints statistics to stdout. | Done: statistics are printed to the screen. |
| Terminated | Child process is terminated, and no longer executing. |  |

## Pseudocode: Parent Process

Main program point of entry; routine executed by the main thread of the application.

|  |
| --- |
| 1. Validate and parse user input |
| 1. If input is invalid, print usage and terminate application |
| 1. Create shared system resources such as synchronization mechanisms |
| 1. Spawn user specified amount of processes to initiate connections |
| 1. Wait for all child processes to terminate, or SIGINT |
| 1. Release acquired system resources then terminate |

## Pseudocode: Child Process

Routine executed by spawned child processes.

|  |
| --- |
| 1. Acquire appropriate system resources. |
| 1. Using epoll, manage all existing connections, sending user specified data, and received data from the server. Collect statistics for each connection session.­­­­­ |
| 1. When SIGINT is received, print out collected statistics, release system resources and terminate. |