Introduction
Lecture
Lab
Assignments
Mini Libc
Memory Allocator
Parallel Graph
Mini Shell
Asynchronous Web Server
Hackathons
Rules and Grading
Resources

Asynchronous Web Server

Objectives
Statement
Details and recommendations for the implementation
Support Code
HTTP Parser
API and Implementation Tasks
Testing and Grading
Behind the Scenes
Debugging
Resources

Objectives

- Deepening the concepts related to working with sockets.
- Developing skills in implementing and designing applications that use asynchronous operations and other advanced I/O operations.
- Deepening the use of the API for advanced I/O operations in the Linux operating system.

Statement

Implement a web server that uses the following advanced I/O operations:

- Asynchronous operations on files
- Non-blocking operations on sockets
- Zero-copying
- Multiplexing I/O operations

The server implements a limited functionality of the HTTP protocol: passing files to clients.

The web server will use the multiplexing API to wait for connections from clients - epoll. On the established connections, requests from clients will be received and then responses will be distributed to them.

The server will serve files from the `AWS_DOCUMENT_ROOT` directory, defined within the assignments' header. Files are only found in subdirectories `AWS_DOCUMENT_ROOT/static/` and `AWS_DOCUMENT_ROOT/dynamic/`. The corresponding request paths will be, for example, `AWS_DOCUMENT_ROOT/static/test.dat` and `AWS_DOCUMENT_ROOT/dynamic/test.dat`. The file processing will be:

- The files in the `AWS_DOCUMENT_ROOT/static/` directory are static files that will be transmitted to clients using the zero-copying API - sendfile)
- Files in the `AWS_DOCUMENT_ROOT/dynamic/` directory are files that are supposed to require a server-side post-processing phase. These files will be read from disk using the asynchronous API and then pushed to the clients. Streaming will use non-blocking sockets (Linux)
- An HTTP 404 message will be sent for invalid request paths

After transmitting a file, according to the HTTP protocol, the connection is closed.

Details and recommendations for the implementation

- Implementing the assignment requires having a state machine for each connection, which you periodically query and update as the transfer proceeds. Check the `connection_state` data structure defined in the assignment header.
- Find the `connection` data structure defined in the assignment header. This can be used to keep track of an open connection.
- Definitions of other useful macros and data structures can be found in the assignment header.
- HTTP responses will have the code `200` for existing files and `404` for not existing files.
  - A valid response consists of the HTTP header, containing the related directives, two newlines (`\r\n\r\n`), followed by the actual content (the file).
  - Sample answers can be found in the parser test file or in the provided sample.
  - You can use predefined request directives such as `Date`, `Last-Modified`, etc.
    - The `Content-Length` directive **must** specify the size of the HTTP content (actual data) in bytes.
    - The `Connection` directive **must** be initialized to `close`.
- The port on which the web server listens for connections is defined within the assignment header: the `AWS_LISTEN_PORT` macro.
- The root directory relative to which the resources/files are searched is defined within the assignment header as the `AWS_DOCUMENT_ROOT` macro.

Support Code

HTTP Parser

The clients and server will communicate using the HTTP protocol. For parsing HTTP requests from clients we recommend using this HTTP parser, also available in the assignments' http-parser. You will need to use a callback to get the path to the local resource requested by the client. Find a simplified example of using the parser in the samples directory.

API and Implementation Tasks

The skel/aws.c file contains the code skeletton with several functions that have to be implemented. Follow the `TODO` areas in the file to start your implementation.

> It can be reorganized as desired, as long as all the requirements of the assignment are implemented.

Testing and Grading

The testing is automated. Tests are located in the `tests/` directory.

To test your implementation, do the following steps:

- Run the `make` command inside the `skel/` directory and make sure it compiles with no errors and that the `aws` executable is generated.
- Run the `make check` command in the `tests/` directory.

There are 35 tests for this assignment, of which 13 are doubled by a memory leak check test. A successful run looks as the following:

```
student@so:~/operating-systems/content/assignments/async-web-server/tests$ make
make -C _test
make[1]: Entering directory '/home/student/operating-systems/content/assignments/
make[1]: Nothing to be done for 'all'.
make[1]: Leaving directory '/home/student/operating-systems/content/assignments/

                        = Testing - Asynchronous Web Server =

01) Test executable exists.....................................................passed  [
02) Test executable runs.......................................................passed  [
03) Test listening.............................................................passed  [
04) Test listening on port.....................................................passed  [
05) Test accepts connections...................................................passed  [
06) Test accepts multiple connections..........................................passed  [
07) Test epoll usage...........................................................passed  [
08) Test disconnect............................................................passed  [
09) Test multiple disconnect...................................................passed  [
10) Test connect disconnect connect............................................passed  [
11) Test multiple connect disconnect connect...................................passed  [
12) Test unordered connect disconnect connect..................................passed  [
13) Test replies http request..................................................passed  [
13) Test replies http request - memcheck.......................................passed  [
14) Test second replies http request...........................................passed  [
15) Test sendfile usage........................................................passed  [
16) Test small static file wget................................................passed  [
17) Test small static file wget cmp............................................passed  [
17) Test small static file wget cmp - memcheck.................................passed  [
18) Test large static file wget................................................passed  [
19) Test large static file wget cmp............................................passed  [
19) Test large static file wget cmp - memcheck.................................passed  [
20) Test bad static file 404...................................................passed  [
21) Test bad path 404..........................................................passed  [
22) Test get one static file then another......................................passed  [
22) Test get one static file then another - memcheck...........................passed  [
23) Test get two simultaneous static files.....................................passed  [
23) Test get two simultaneous static files - memcheck..........................passed  [
24) Test get multiple simultaneous static files................................passed  [
24) Test get multiple simultaneous static files - memcheck.....................passed  [
25) Test io_submit usex........................................................passed  [
26) Test small dynamic file wget...............................................passed  [
27) Test small dynamic file wget cmp...........................................passed  [
27) Test small dynamic file wget cmp - memcheck................................passed  [
28) Test large dynamic file wget...............................................passed  [
29) Test large dynamic file wget cmp...........................................passed  [
29) Test large dynamic file wget cmp - memcheck................................passed  [
30) Test bad dynamic file 404..................................................passed  [
31) Test get one dynamic file then another.....................................passed  [
31) Test get one dynamic file then another - memcheck..........................passed  [
32) Test get two simultaneous dynamic files....................................passed  [
32) Test get two simultaneous dynamic files - memcheck.........................passed  [
33) Test get multiple simultaneous dynamic files...............................passed  [
33) Test get multiple simultaneous dynamic files - memcheck....................passed  [
34) Test get two simultaneous static and dynamic files.........................passed  [
34) Test get two simultaneous static and dynamic files - memcheck..............passed  [
35) Test get multiple simultaneous static and dynamic files....................passed  [
35) Test get multiple simultaneous static and dynamic files - memcheck.passed  [

                                                                        Total:  [
```

Individual tests can be run using the `./run_test.sh` bash script as the following:

```
student@so:~/operating-systems/content/assignments/async-web-server/tests$ ./_te
03) Test listening.............................................................passed  [
```

Where `3` is the test you want to run.

Some tests are doubled by a memory check test. This will only run if the regular test passed. For example, test 31 will output the following in case of success:

```
student@so:~/operating-systems/content/assignments/async-web-server/tests$ ./_te
31) Test get one dynamic file then another.....................passed  [
31) Test get one dynamic file then another - memcheck..........passed  [
```

and one of the following in case of error:

```
# if the regular tests failed, the memory check tests is not performed
student@so:~/operating-systems/content/assignments/async-web-server/tests$ ./_te
31) Test get one dynamic file then another.....................failed  [
31) Test get one dynamic file then another - memcheck..........passed  [
```

> Note: The memcheck test for failed regular tests will not be taken into consideration for the final score. This output will be fixed in the next commit.

Tests use the `static/` and `dynamic/` folders. These folders are created and removed using the `init` and `cleanup` arguments to `_test/run_test.sh`.

Behind the Scenes

Tests are basically unit tests.

Each test function follows the unit test patter: initialization, action, evaluation.

Each test starts the server, creates a given context, checks for validity and then terminates the server process.

Debugging

Logs are collected in `test.log` and `wget.log` files.

Resources

- sendfile
- io_setup & friends
- epoll