11 February 2022

Operating System Projet

Concurrency webserver

ELFATIHI Maxence
OUASFI Amine

Outline

Web server architecture

The initial webserver implementation relies on:

- A server executable running a loop handling upcoming requests
- A client executable generating requests to the server

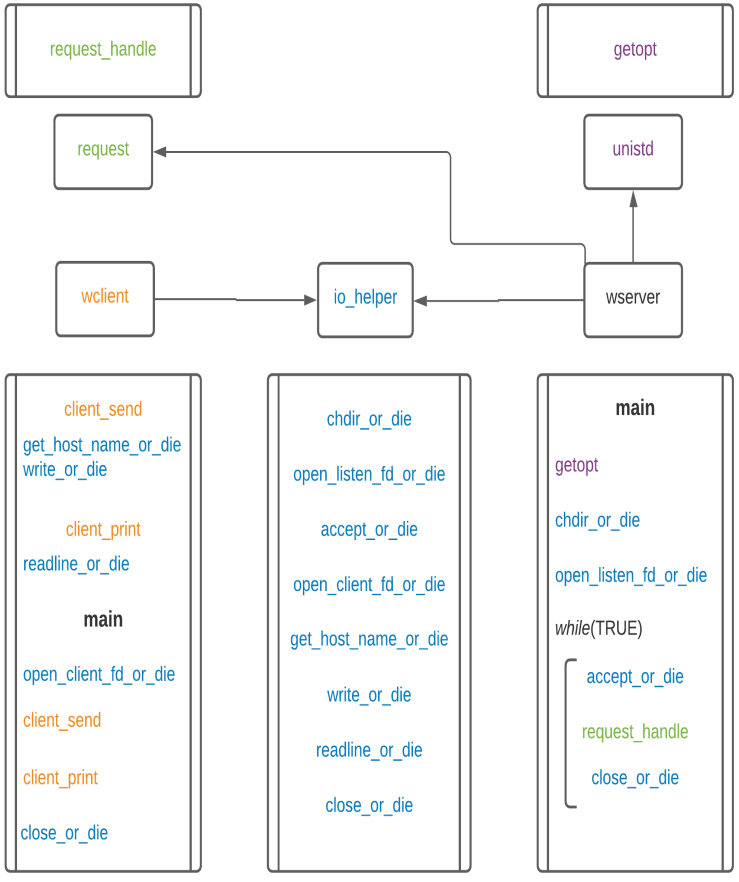

Figure 1: Dependency diagram of the server.

Limitations

The monothreaded webserver can handle one request at a time which is not suitable for:

- Multiple clients simultaneously requesting files
- Requesting dynamic content
- No security consideration for accessing restricted files

Multi-threaded server

The main idea provide a buffer where client requests are stored and a pool of threads that handle these requests concurrently

- Consumer producer model
- Producer: receives requests and add them to the buffer
- Consumer: handles requests and remove them from the buffer

Protected blocks: Read and write operations on the buffer

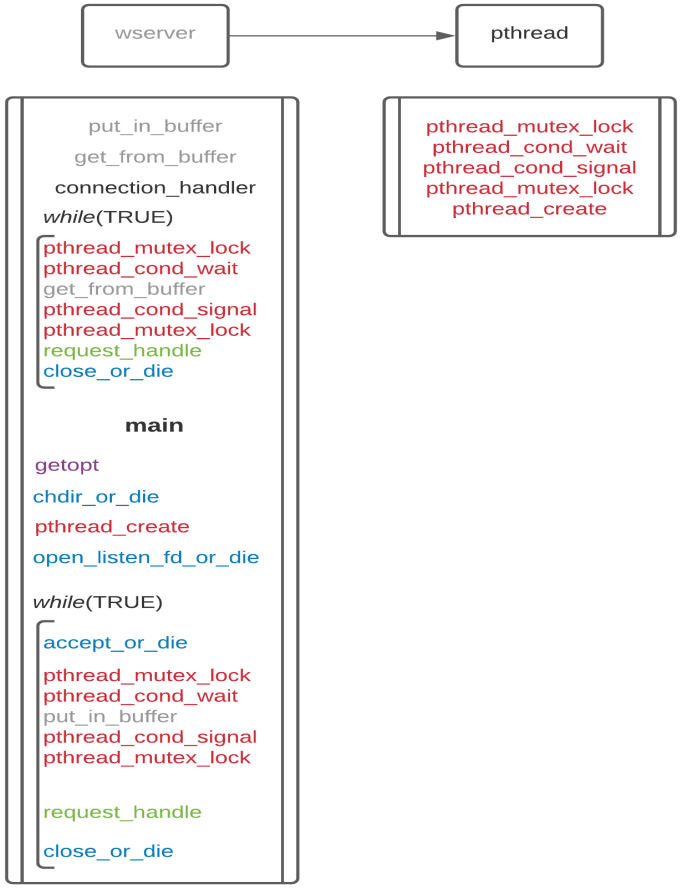

Figure 2: Multi-threaded server dependencies and structure.

Producer

Algorithm 1 Main function and producer

```
initialize server arguments: buffersize, root_dir, port, n_threads, count
initialize mutex lock and condition variables empty and fill
update server arguments based on argv
initialize threads
for thread in threads do
    pthread start thread
end for
start listening to port port
//here starts the producer
while True do
    Wait for an incoming request and store it's file_descriptor when accepted.
    Enable mutex lock
    while buffer is full do
        Release lock and block the current thread on the condition variable empty;
    end while
    put request in buffer and increment count
    unblock at least one of the threads that are blocked on the condition variable fill
    disable mutex lock
end while
```

Consumer

Algorithm 2 Consumer

```
while True do
    Enable mutex lock
    while buffer is empty do
        Release lock and block the current thread on the condition variable fill;
    end while
    get request from buffer and decrement count
    unblock at least one of the threads that are blocked on the condition variable empty
    disable mutex lock
    handle request
    close or die
end while
```

Design choices

The multithreaded webserver was built around the following design choices:

- We store and access the requests using a queue structure
- The main function of the webserver implements the master thread
- We use locks (and not semaphores) to insure threads can safely access the shared memory
- We make sure that the only accessible files are within the working directory

Test settings

Question: How does the execution time evolve w.r.t to the buffer size, the number of clients or the number of threads?

- Requests on `spin.cgi` file for **5 seconds**.
- Single client request
- Multiple client requests
- We tested each parameter with a low and a high value.

Results

Table 1: **Multi-threaded web-server execution time.**

Test	buffer size	n_clients	n_threads	execution time
A	8	9	10	5,446s
B	8	9	100	5,126s
C	8	90	10	45,464s
D	8	90	100	6,641s
E	80	9	10	5,114s
F	80	9	100	5,111s
G	80	90	10	45,218s
H	80	90	100	5,975s