Report 1: Rudy

Chen Sihan

September 8, 2020

1 Introduction

This report is to show how to use Erlang to build a simple server and the communication flow between the server and clients.

The main topic related to distributed systems is the management of the requests from clients in parallel.

2 Main problems and solutions

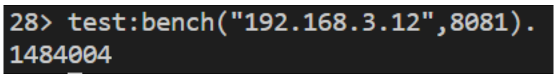
Set up the server on one machine and access it from another machine

Like the instruction in the homework pdf, three files are built: the http parser file (http.erl), the server file which is registered under the name ”rudy” (reply.erl) and a client file (test.erl) which is used to invoke requests to the server.

The assignment simply need a testing of the functions from another machine. Therefore, I let my partner from another machine execute the client request functions. As my WLAN IPv4 address is 192.168.3.12 and I open the port 8081 to start the rudy server:

```
c(reply).
reply:start(8081).
```

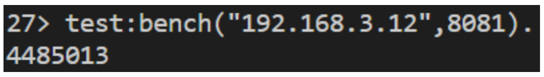
Then my partner connect the host and port and make 100 requests sequentially. In this case no sleeping time in a process is defined. The outcomes are shown as follows:



```
28> test:bench("192.168.3.12",8081).
1484004
```

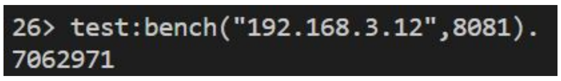
Figure 1: outcome when 100 requests to rudy without sleeping time

The return value is 1484004 micro seconds, namely the duration 100 requests to rudy with no sleeping time in server process. Besides, I also test the cases with 20 and 40 milliseconds, which are shown as follows:



```
27> test:bench("192.168.3.12",8081).
4485013
```

Figure 2: outcome when 100 requests to rudy each with 20ms sleeping time



```
26> test:bench("192.168.3.12",8081).
7062971
```

Figure 3: outcome when 100 requests to rudy each with 40ms sleeping time

Since the product of the number of requests and the number of 20ms sleeping time in each request is 100 * 20ms = 2000ms, which is nearly equal to difference between the time of different testing cases in 1, 2, 3. The functions are shown to be correctly implemented.

3 Bonus task

The multi-threaded extension is implemented in the experiment. The idea is to create a handler pool to manage the requests of the clients. Erlang allows different processes to listen to a socket, which means we can create multiple processes and let them all listen to the socket. Whenever a request comes, one vacant process starts to serve the request. Whenever a process finishes serving one request, it is released to the handler pool and serves another request. The above functionality is implemented in **rudy4**
.erl. Currently, we must simulate the requests from multiple clients so in **test2**
.erl, we create 4 client processes to run 40 requests (each with 10 requests). The testing results are shown as follows:

Handler(server) processes	Time
1	4282ms
2	1922ms
4	953ms
8	922ms

Table 1: Caption

The results demonstrate that with the number of handler processes increasing, the time it takes to handle the same number of requests decreases. However, as there are only 4 clients run requests simultaneously, which means that only 4 processes can be dealt with concurrently, when the handler processes exceed 4, it still spends the same time as 4 processes does.