Homework 1

The Benefits of Concurrency

Tanya Sharma
NetId: tds104
28 March 2023

Collaboration and References

Links	Uses
https://man7.org/linux/man-pages/man7/epoll.7.html	Linux Manual page for information on the implementation and use of epoll
https://copyconstruct.medium.com/the-method-to-epolls-madness-d9d2d6378642	For a deeper dive into event polling and edge-triggered epoll
https://man7.org/linux/man-pages/man2/poll.2.html	Linux Manual page for poll, to understand the mechanism and the basis of epoll
https://stackoverflow.com/questions/9162712/what-is-the-purpose-of-epolls-edge-triggered-option	Understanding edge-triggered vs level-triggered epoll
https://www.baeldung.com/cs/whats-the-p99-latency	Understanding the significance of the 99th percentile latency in measuring the performance of a web server and networks
https://developer.mozilla.org/en-US/docs/Web/Performance/Understanding_latency	Information on latency
https://mvolo.com/why-average-latency-is-a-terrible-way-to-track-website-performance-and-how-to-fix-it/	While trying to understand why one of my test cases is not producing the expected outcome, I came across this post that I found interesting while trying how to interpret average latency

Configuration

Q3. What is the configuration of the (virtual) machine where you ran your experiments? Are you using a VM? How many cores and how much memory does your system have? Which operating system and version does it run? If you're using a VM, report the VM's parameters, not those of your physical machine.

Machine used @ cpp.cs.rutgers.edu

Architecture:	x86_64
CPU op-mode(s):	32-bit, 64-bit
Byte Order:	Little Endian
Address sizes:	39 bits physical, 48 bits virtual
CPU(s):	20
On-line CPU(s) list:	0-19
Thread(s) per core:	2
Core(s) per socket:	10
Socket(s):	1
NUMA node(s):	1
Vendor ID:	GenuineIntel
CPU family:	6
Model:	165
Model name:	Intel(R) Core(TM) i9-10900 CPU @ 2.80GHz
Stepping:	5
CPU MHz:	2800.000
CPU max MHz:	5200.0000
CPU min MHz:	800.0000
BogoMIPS:	5599.85
Virtualization:	VT-x
L1d cache:	320 KiB
L1i cache:	320 KiB
L2 cache:	2.5 MiB
L3 cache:	20 MiB
NUMA node0 CPU(s):	0-19

Performance

Q4. With the command provided under step #4, what are the measured average latencies for the provided web server and your epoll one? What about the 99th percentile? What is the average number of requests/second?

- **webserver.c**

```
Running 30s test @ http://localhost:8080
  2 threads and 100 connections
  Thread calibration: mean lat.: 20.588ms, rate sampling interval: 84ms
  Thread calibration: mean lat.: 24.392ms, rate sampling interval: 87ms
```

Thread Stats	Avg	Stdev	Max +/-	Stdev
Latency	22.07ms	15.32ms	49.18ms	57.71%
Req/Sec	1.01k	229.83	1.18k	73.48%

Latency Distribution (HdrHistogram - Recorded Latency)

50.000%	21.87ms
75.000%	36.48ms
90.000%	42.78ms
99.000%	47.84ms
99.900%	48.64ms
99.990%	48.83ms
99.999%	49.22ms
100.000%	49.22ms

```
59721 requests in 30.00s, 3.36MB read
Requests/sec:   1990.61
Transfer/sec:    114.69KB
```

- **<u>epoll.c</u>**

```
Running 30s test @ http://localhost:8080
  2 threads and 100 connections
  Thread calibration: mean lat.: 0.807ms, rate sampling interval: 10ms
  Thread calibration: mean lat.: 0.807ms, rate sampling interval: 10ms
```

Thread Stats	**Avg**	**Stdev**	**Max**	**+/-**	**Stdev**
Latency	798.28us	398.23us	3.81m	s	65.92%
Req/Sec	1.08k	116.83	1.67k		84.93%

Latency Distribution (HdrHistogram - Recorded Latency)

50.000%	792.00us
75.000%	1.07ms
90.000%	1.29ms
99.000%	1.80ms
99.900%	2.12ms
99.990%	3.21ms
99.999%	3.81ms
100.000%	3.81ms

```
59804 requests in 30.00s, 3.36MB read
Requests/sec:   1993.41
Transfer/sec:    114.85KB
```

Inference

Q5. Do you see a difference in measured requests/sec or average and 99th pc latency between the two web servers? Why or why not? Can you explain the difference in terms of their underlying designs?

There is a marked difference in the measured requests/sec, average latency and latency at the 99th percentile between the two web servers while the number of requests for both servers are about the same.

The average requests/second of the webserver is slightly lower than that of the epoll server which means that the epoll server can handle slightly more requests in the given period of time.

The average latency of the epoll server is significantly lower than that of the webserver which implies that the epoll server can handle requests more swiftly than the webserver overall and on average.

The 99th percentile latency of the epoll server is also much lower than the webserver which indicates that 99% of the requests in the epoll server are handled in the given time which is much lower than the webserver one. This also means that there are only a few requests that take a longer time to be processed than the given time.

Overall, upon observing these performance metrics we can infer that the performance of the epoll server in handling a higher number of connection requests is much better than that of the webserver because of the former’s low latency and high throughput.

The underlying design of epoll contributes significantly to this observed performance superiority. The epoll server employs the event-driven I/O notification model that can handle several concurrent connections very efficiently. The server can effectively check the I/O readiness of multiple sockets using epoll and can subsequently handle numerous connections at once without utilizing a large number of system resources. The server will wait for an event to occur on any of its sockets and then proceeds to handle these requests, it is thus using non-blocking I/O mechanism where it does not block the processes or threads while waiting for an event to occur and can instantly handle events without waiting for other to be completed. Therefore, we can handle many requests and events with a small number of threads or processes. Even the list of file descriptors are stored efficiently which can make lookups swift and efficient. The event-driven model means that the server will only know that data has arrived at the socket and does not care about when the data is ready to be read.

Our webserver, on the other hand deals with incoming connection requests one at a time. When a connection comes in, the server blocks or stops responding until the request has been completely processed. When the server is managing a lot of requests, this might result in higher latency and lower throughput because the threads and processes have to block while they wait for a request to arrive.

Testing

Q6. Explore different parameters to run the benchmark: e.g., targeting higher or lower request- s/sec, number of threads, and number of connections. Explain your findings with reasoning around what you observe. You may instrument your server by recording timestamps, remove print statements, etc. to make your evaluations and reasoning more rigorous.

According to the README associated with wrk2, the minimum recommended runtime for the benchmarking tool is 30s as it takes at least 10 seconds to calibrate and running the test for less than 10-20s would not produce useful results. For this reason, we will run all our tests for a period of 30s.

We select number of threads, number of connections and request per second as are parameters and vary any one of them to observe the trend of latency.
We choose the following default values for our parameters:

Number of connections: 100
Number of requests=2000
Number of threads=2

<u>Test # 1:</u> Varying number of threads

For this test we will vary the number of threads we will be using. For more accurate observations and inferences we will keep the number of requests and the number of connections constant.
Number of requests per second: 2000
Number of open HTTP connections: 100

The number of threads will vary as demonstrated in the following table and we will note the average latency and 99.00%th latency for each scenario:

Number Of Threads	**Webserver.c**		**Epoll.c**	
	Average Latency	**99.00%th latency**	**Average Latency**	**99.00%th latency**
2	23.71ms	47.17ms	800.89us	2.01ms
4	25.66ms	50.69ms	734.99us	2.04ms
8	24.31ms	49.34ms	815.21us	1.66ms
16	34.40ms	49.02ms	813.74us	1.51ms
32	40.40ms	49.41ms	805.58us	1.95ms
64	29.08ms	33.12ms	809.07us	1.71ms

Observations:

- For webserver.c, the average latency decreases with an increase in number of threads to a point and then seems to start increasing again after 8 threads.
- For epoll.c, the average latency initially decreases with an increase in number of threads and then seems to start increasing again after 16 threads
- For both webserver.c and epoll.c, increasing the number of threads can lead to a slight increase in the average latency. This could mean that increasing the number of threads with the given number of connection, requests and our machine specifications does not necessarily increase performance but it may be possible that there is an additional overhead that has been introduced due to thread management and because the threads could be competing for the CPU.
- For webserver.c, increasing the number of threads keeps the 99th latency quite similar or a slight decrease in latency overall which could indicate better performance as the server is able to handle requests much faster. Since the webserver handles requests one at a time, increasing number of threads could thus increase performance albeit not very significantly.
- For epoll.c, increasing the number of threads consistently decreases the 99th percentile latency which indicates that the server becomes more consistent and is able to handle incoming requests more predictably.
- Epoll.c has better performance in terms of lower latency than webserver.c in terms of both average and 99th percentile latencies

To conclude, despite increasing the number of threads, epoll server outperforms the other one when dealing with requests and also has slightly better performance upon increasing number of threads. It is also useful to consider that the iLab machine being used has only one socket with 10 cores. There may be a correlation between the peak average latency observed when using 8 threads and the subsequent decline in latency due to overheads that are present with thread management. Nonetheless, an increase in number of threads could lead to lower latency values but there is a limit to such an increase after which the overheads involved with thread management begin contributing negatively to the performance and latency values for our webserver and epoll server.

<u>Test # 2:</u> Varying number of connection

For this test we will vary the number of connection we will be using. For more accurate observations and inferences we will keep the number of requests and the number of threads constant.
Number of requests per second: 2000
Number of threads: 2

The number of connections will vary as demonstrated in the following table and we will note the average latency and 99.00%th latency for each scenario:

Number Of Connections	Webserver.c		Epoll.c	
	Average Latency	99.00%th latency	Average Latency	99.00%th latency
10	1.08ms	2.04ms	748.84us	1.45ms
50	10.73ms	22.17ms	747.66us	1.75ms
100	24.15ms	47.13ms	802.61us	1.76ms
500	148.41ms	249.73ms	0.86ms	1.90ms
1000	387.03ms	1.52s	1.27ms	4.99ms
2000	3.45s	16.50s	1.40ms	5.14ms

Observations:

- For both the webserver.c and epoll.c, the average latency and the 99th percentile latency increase with the number of connections. Since more connections mean more requests for the server than it could potentially handle, the increase in latency in expected.
- For webserver.c, the latency is increasing at a faster rate than epoll.c which could indicate that epoll is more efficiently able to handle a large number of connections with less changes to its latency potentially due to its I/O event driven mechanism.
- For a small number of connections, there is not a very significant difference in the latency of both the servers and both have low latency but as the number of connections increases, epoll starts performing better than the webserver in both average and 99th percentile latency.
- For a large number of connections, both servers have quite high latency but webserver has much higher latency than the epoll server which again could indicate that epoll's architecture allows it handle a large number of connections more efficiently and with lesser delays. The high latency could also indicate performance issues or a bottleneck in our system.

To conclude, we can infer that by increasing the number of connections, the server will have to handle a higher number of simultaneous requests which could increase the load on the server and cause connection of resources which all could attribute to an increase in latency. Furthermore, the consistent performance of the epoll server as opposed to the webserver is a testament to its design that can handle a higher number of requests much more seamlessly and with little change to the latency as a measure of performance.

<u>Test # 3:</u> Varying number of requests per second

For this test we will vary the number of requests per second we will be using. For more accurate observations and inferences we will keep the number of threads and the number of connections constant.
Number of open HTTP connections: 100
Number of threads: 2

The number of requests will vary as demonstrated in the following table and we will note the average latency and 99.00%th latency for each scenario:

Number Of Requests/s	Webserver.c		Epoll.c	
	Average Latency	99.00%th latency	Average Latency	99.00%th latency
500	110.93ms	196.74ms	758.04us	1.80ms
1000	53.23ms	53.23ms	786.43us	1.76ms
2000	22.70ms	47.13ms	811.45us	1.81ms
3000	17.20ms	30.59ms	827.57us	1.79ms
4000	7.05ms	23.39ms	830.35us	1.80ms
5000	9.24ms	18.05ms	0.93ms	1.94ms

Observations:

- The average and 99th percentile latency for the epoll server is consistently lower than the webserver for all our test cases
- The 99th percentile latency for both the epoll server and the webserver are relatively low even at higher requests per second
- For epoll.c, latency is generally more consistent across all test cases and remains low, which we can attribute to its design for handling a large number of requests concurrently since epoll uses the epoll I/O event mechanism that can potentially scale to monitor a large number of file descriptors.
- The latency of the webserver.c seems to fluctuate significantly especially when compared to epoll.c and decreases with an increase in number of requests. This is not expected behaviour since the webserver handles requests one at a time and one would expect the average latency and 99th percentile latency of the webserver to increase with time given its inability to handle large volumes of connection requests. We can only theorise that our system is allocating more resources to webserver.c with an increasing number of requests given that we are using an iLab machine with scalable system resources.

To conclude, because of a difference in the behaviour of our webserver and epoll sever, it is difficult to draw a conclusion about the relationship between the number of requests and latency in both our servers. The epoll server has more desirable values for latency for any number of requests. Without further investigation into other factors that could be affecting the performance of our web servers, we cannot present conclusive results about the relationship between the number of requests and latency of web servers. Although, theoretically an increase in number of requests should lead to higher values for latency due to the fact that the webserver handles requests one at a time and a higher number of requests would present substantial delays in the time the server takes to response such requests.

While changing one factor that could contribute to the change of average latency in our server, we are not considering the effect of the other two factors that we can test on, having an effect on our final results. For example, testing with two threads while changing number of connections and requests per second, we could potentially cause a bottleneck in our performance. For this reason, we check the changes in latency by varying our parameters again for a different set of parameters to make sure that our results are consistent.

Number of connections: 100
Number of requests=5000
Number of threads=10

<u>Test # 1:</u> Varying number of threads

For this test we will vary the number of threads we will be using. For more accurate observations and inferences we will keep the number of requests and the number of connections constant.
Number of requests per second: 5000
Number of open HTTP connections: 100

The number of threads will vary as demonstrated in the following table and we will note the average latency and 99.00%th latency for each scenario:

Number Of Threads	**Webserver.c**		**Epoll.c**	
	Average Latency	**99.00%th latency**	**Average Latency**	**99.00%th latency**
2	10.33ms	17.44ms	0.89ms	1.86ms
4	8.33ms	17.26m	0.86ms	1.87ms
8	8.34ms	19.87ms	843.76us	1.76ms
16	6.38ms	20.50ms	761.53us	1.90ms
32	7.78ms	20.13ms	0.86ms	1.66ms
64	4.54ms	13.99ms	708.70us	1.26ms

<u>Observations:</u>

- For both the websever and the poll server, latency generally seems to decrease with an increase in number of threads but does not do so linearly as with an increase in number of threads we could have overhead associated with handling multiple threads. The decrease in latency could be because with more threads, we are spreading the load across more processors and thus processing requests in parallel which could lead to lower latency.
- For websever.c, the 99th percentile latency goes down with an increase in number of threads. Thus, the worst case latency for many requests is improving as we add more threads because there are more avenues for processing of requests which reduces latency.
- For epoll.c, the 99th percentile latency remains consistent across all scenarios which could be due to the fact that epoll is more scalable in general.

- Epoll.c performs better than webserver.c in terms of both average latency and 99th percentile latency which could be because epoll has a very efficient I/O multiplexing model as opposed to the webserver's sequential or one at a time approach.

To conclude, when comparing both scenarios, we can clearly infer that performance of both servers does get better with an increase in number of threads. In our previous observations, the number of requests were 2000 instead of 5000 and the average latency of the servers would reach a record low before starting to increase again because at that point there were more threads than were needed to handle the number of requests coming in and thread contention amongst unnecessary threads caused a marked increase in latency. In this scenario, a higher number of requests causes the record for the low point of latency to shift to a higher number of threads because an increase in threads now could be helpful due to the large number of requests. Threads can improve concurrency but not offer much of parallelism.

<u>Test # 2:</u> Varying number of connection

For this test we will vary the number of connection we will be using. For more accurate observations and inferences we will keep the number of requests and the number of threads constant.
Number of requests per second: 5000
Number of threads: 10

The number of connections will vary as demonstrated in the following table and we will note the average latency and 99.00%th latency for each scenario:

Number Of Connections	**Webserver.c**		**Epoll.c**	
	Average Latency	**99.00%th latency**	**Average Latency**	**99.00%th latency**
10	1.00ms	3.16ms	668.26us	1.18ms
50	4.83ms	11.88ms	772.82us	1.44ms
100	8.01ms	18.40ms	729.00us	1.86ms
500	52.91ms	100.99ms	737.35us	1.69ms
1000	154.80ms	202.37ms	0.87ms	2.75ms
2000	340.22ms	400.89ms	2.10ms	4.71ms

<u>Observations:</u>

- For both webserver.c and epoll.c we see an increase in the average and 99th percentile with an increase in number of connections. Since a large number of requests implies that the server will have more request to handle at the same time, the increase in latency is as expected.
- For both webserver.c and epoll.c, we notice that for a large number of connections, the latency increase is more noticeable because after one point

the server can become contained for resources which could result in a higher latency

- Epoll.c performs better than webserver.c in terms of both the average latency and 99th percentile latency. This is because epoll.c uses an event driven architecture that allows it to handle a lot of connections with not much overhead while webserver.c uses a one-thread-per-connection mechanism that can cause high overhead as the number of connections increases.

To conclude, despite other parameters, increasing the number of connections causes an increase in latency because the server will now have to handle requests at the same time which could use up more resources.

<u>Test # 3:</u> Varying number of requests per second

For this test we will vary the number of connections per second we will be using. For more accurate observations and inferences we will keep the number of threads and the number of connections constant.
Number of open HTTP connections: 100
Number of threads: 10

Number Of Requests/s	**Webserver.c**		**Epoll.c**	
	Average Latency	**99.00%th latency**	**Average Latency**	**99.00%th latency**
500	137.56ms	201.09ms	701.28us	1.78ms
1000	76.49ms	101.31ms	806.06us	1.63ms
2000	25.02ms	51.10ms	750.32us	1.69ms
3000	18.03ms	34.05ms	736.27us	1.88ms
4000	14.95ms	26.50ms	721.23us	1.65ms
5000	7.62ms	20.08ms	707.56us	1.81ms

<u>**Observations:**</u>

- For epoll.c, the average and 99th percentile latency seems to remain fairly consistent across an increasing number of requests.
- For webserver.c, the average and 99th percentile latency seems to decrease with an increase in number of threads. This is not expected behaviour as we would assume that the average latency of the sever increases with an increase in number of threads due to the fact that it services requests one at a time and would have to service a higher number of requests.
- For both epoll.c and webserver.c, 99th percentile latency is quite high which could mean that there are some requests that are experiencing high latency which could be caused due to the load on the server.

To conclude, epoll.c has consistent values for average and 99th percentile latency across a varying number of requests and webserver.c shows unexpected behaviour by having decreasing values for latency with an increase in number of requests. Since the behaviour of the webserver is consistent across two different values for threads and number of connections, we can say with certainty that this is not an anomaly and further investigation into the role of our system and network parameters must be done in order to offer more conclusive results.