**1. Code architecture and Lessons learned**

**Architecture:** The web crawler designed for this project is a multi-threaded application capable of processing multiple URLs concurrently to efficiently crawl web content. Here's a breakdown of its core components and functionality.

- **URL Management:** URLs are managed through a queue system where each thread can pull tasks independently, ensuring efficient distribution of work.
- **HTTP Communications:** The crawler constructs HTTP requests to fetch pages and handles responses, parsing the headers and content.
- **DNS Resolution:** Incorporates DNS lookup to resolve hostnames into IP addresses, which is crucial for network requests.
- **Concurrency Control:** Utilizes mutexes and atomic operations to manage concurrent access to shared resources like URL lists and counters, preventing race conditions and ensuring data integrity.
- **Error Handling:** Robust error handling mechanisms to manage network errors, invalid responses, and other runtime exceptions.
- **Statistics Tracking:** Real-time tracking of various metrics such as number of pages crawled, number of links found, HTTP status codes encountered and others.

**Key Components:**

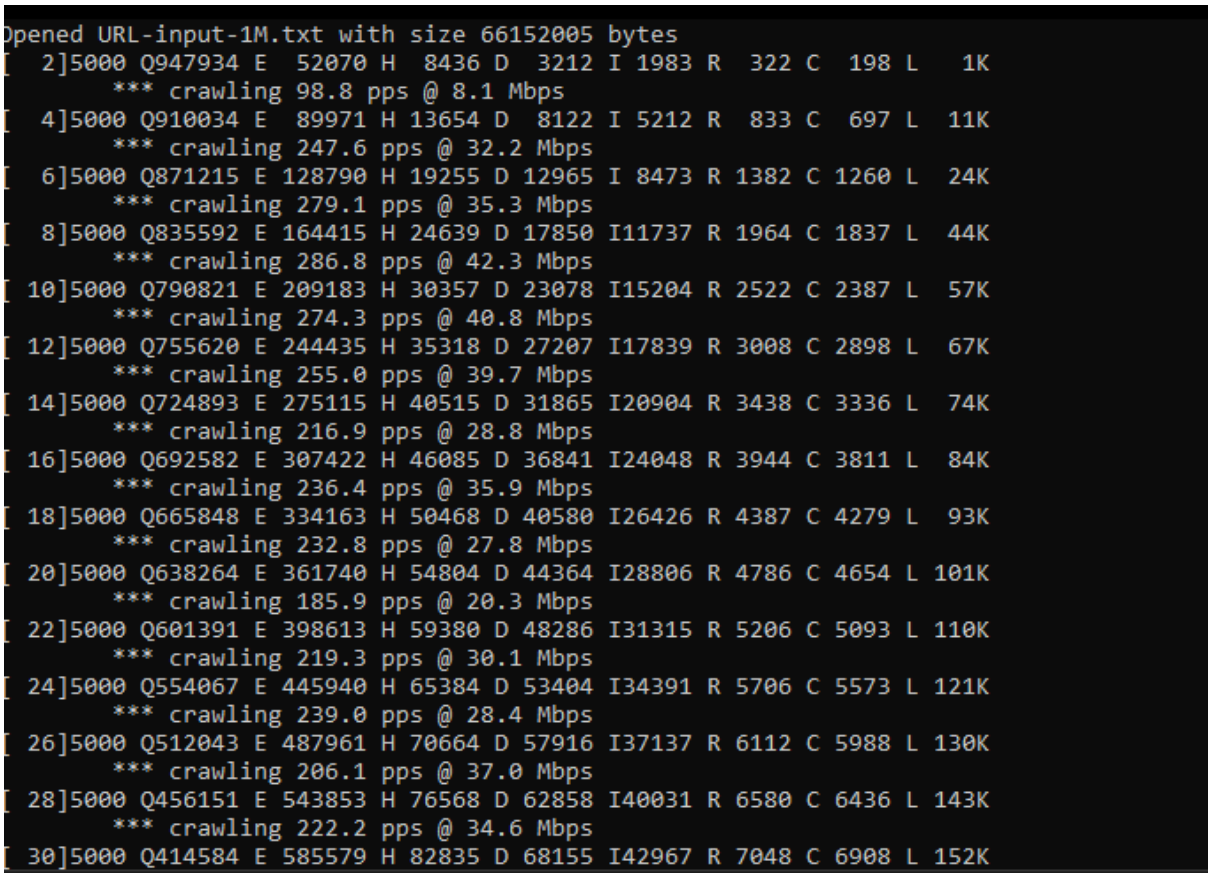
- **Robots.txt Compliance**: Checks each domain for a robots.txt file to ensure the crawler respects site-specific scraping rules.
- **Link Extraction**: Uses an HTML parsing library to extract links from each webpage, which are then processed and added to the queue if they have not been visited.
- **Resource Management**: Handles network connections and system resources efficiently, ensuring that sockets are properly managed and closed after transactions to prevent resource leaks.

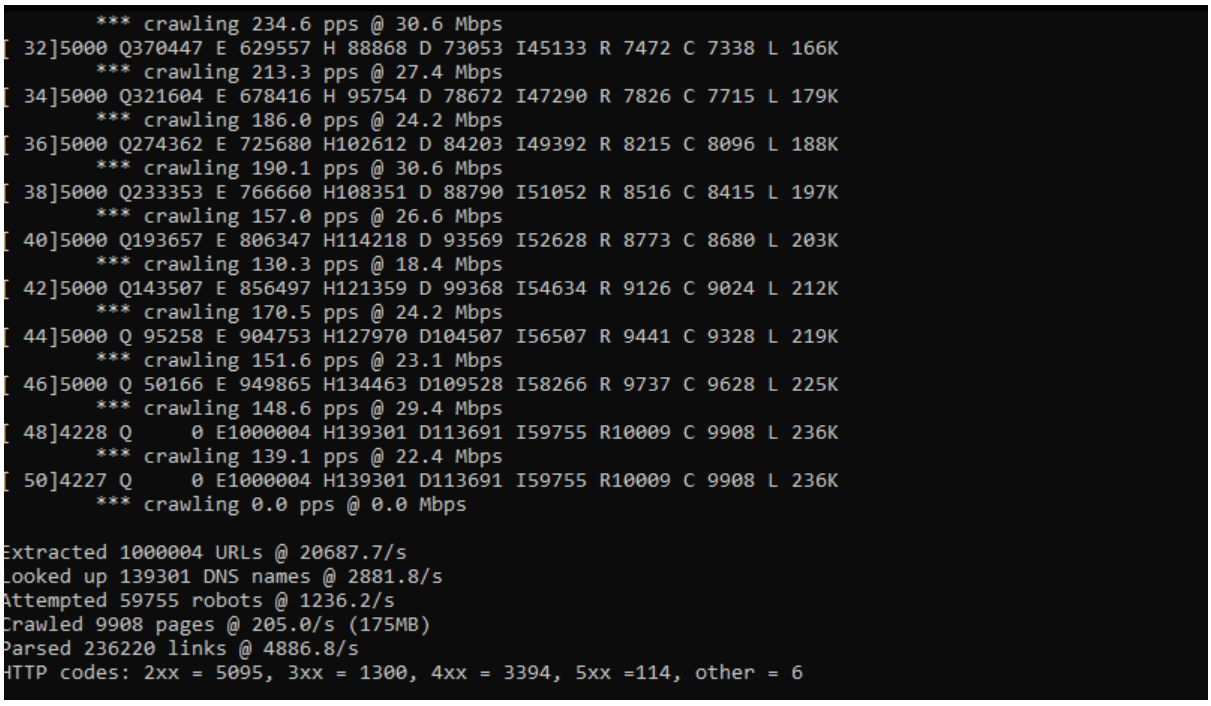
**Lessons Learned:**

- **Multi-threading Efficiency:** Learned the importance of balancing the number of threads to optimize the use of CPU and network resources without overwhelming the web server or causing contention issues.
- **Error Handling:** Reinforced the need for comprehensive error handling in network applications to manage unpredictable network behavior and server responses.
- **Data Integrity:** The critical role of synchronization mechanisms in preventing data corruption and ensuring the accuracy of shared data among threads.
- **Resource Management:** Gained insights into the management of system resources, particularly in network programming where proper handling of sockets and responses is crucial for maintaining crawler stability and performance.

- **Compliance and Ethics:** Understood the importance of adhering to web standards and ethical scraping guidelines, such as respecting robots.txt and managing request rates to avoid negatively impacting the performance of target websites.

The following figures show a complete trace of 1M URLs input.




**2. Calculate the average number of HTML links per 2xx page**

Average numofLinks per 2xx page= L/2xx pages =  236020/5095 = 46.32 L/Page

Edges per Page = Average Links per Page

Total Edges = Total Pages×Edges per Page

Storage per Edge = 64 bits per link=8 bytes(since1byte=8bits)

Total Storage = Total Edges × 8 bytes

- **Average Number of HTML Links per Page with a 2xx Status Code:** The average number of links per page for pages that successfully returned a 2xx HTTP status code is approximately **46.32 links per page**.
- **Estimate of the Size of Google's Webgraph:**
    - **Total Edges in the Webgraph:** Given the average of 46.32 links per page and assuming Google crawls 1 trillion pages, the total number of edges is approximately 46.32×1 trillion≈46.32 trillion.
    - **Total Storage Required for the Webgraph:** If each edge requires 8 bytes (since each URL/link is represented by a 64-bit hash), the total storage needed is approximately **370.59 petabytes**.

**3. Determine the average page size and estimate bandwidth needs**

- Crawled 9908 pages
- Total data downloaded: 175 MB.
- Average Page Size: Total data downloaded/ Crawled pages = 0.018 MB/Page
- To estimate the bandwidth needed to crawl 10 billion pages in one day:
    - Bandwidth in Gbps = Total Data in Gigabits/Seconds in a day; and convert Mbps to Gbps
    - Bandwidth in Gbps = 0.018MB per Page * 10 * 10^10*8/(86400 *1024) = 16.28 Gbps

**4. Probability analyses**

**Probability of a link containing a unique host**

- Total number of links in the input file = 1000004
- Total number of unique hosts = 139301
    - probability that a link contains a unique host = 139301/1000004= 0.139

**Probability that a unique host has a valid DNS record**

- Total number of unique hosts = 139301
- Number of unique hosts that successfully resolved a DNS query = 113691
    - probability that a unique host has a valid DNS record = 113691/139301 = 0.82

**Percentage of contacted sites that had a 4xx robots file**

- Total number of sites where robots.txt was requested (passed unique IP check) = 59755
- Number of those requests that returned a 4xx status = 10009

- percentage of contacted sites had a 4xx robots file = 10009/59755 = 0.17

**5. Hyperlink analysis to TAMU domain (1M URLs)**

- **Hyperlinks to TAMU**
  - The number of 2xx pages that contain at least one hyperlink to 'tamu.edu': 3.
  - The number of pages hosted outside the TAMU network: 3.

The following Figure show the output after adding the (Hyperlink Analysis of TAMU domain)

```
Extracted 1000004 URLs @ 15534.0/s
Looked up 139301 DNS names @ 2163.9/s
Attempted 59758 robots @ 928.3/s
Crawled 9907 pages @ 153.9/s (172MB)
Parsed 214786 links @ 3336.5/s
HTTP codes: 2xx = 5039, 3xx = 1313, 4xx = 3436, 5xx =112, other = 7
Total Pages with valid TAMU links found: 3
Total Pages with valid TAMU links originating from outside TAMU: 3
```

- **Regex Matching**: Introduced a regular expression to identify links that point to the Texas A&M University (TAMU) domain (tamu.edu). The regex checks for URLs that strictly belong to the TAMU domain, ensuring it captures the right format and excludes subdomains or similar patterns not exactly matching tamu.edu.
- **Checking Origin**: To determine whether a hyperlink originates from outside the TAMU network, a condition was created where the crawler checks the host against the TAMU domain using the same regex. If the host of the page being crawled does not match the TAMU regex, it is counted as an external link.

The following figure illustrates the changes in the code.

```
regex tamuRegex(R"(^http:\/\/(?:.*\.)?tamu\.edu\b(?:\/.*)?)");
for (int i = 0; i < nLinksPage; i++) {
    string link(linkBufferPage);
    if (regex_match(link, tamuRegex)) {
        stats.incrementPagesWithValidTAMULinks();
        string origin = "http://" + parts.host;
        if (!regex_match(origin, tamuRegex)) {
            stats.incrementExternalOriginTAMULinks();
        }
        break;
    }
    linkBufferPage += strlen(linkBufferPage) + 1;
}
```