Multithreaded Web Crawler - Description

Andrei-David Nan,
Operating Systems 2, Final Project,
andrei.nan03@e-uvt.ro

April 7, 2024

1 Brief overview of the project

My project consists of a multithreaded web crawler built in C++, which will be able to traverse the web, retrieve webpage data, and extract relevant information for analysis, optionally outputing crawled websites statistics. The project aims to be configurable, allowing users to customize parameters such as crawl delay, maximum number of threads, depth limit through, etc. either through a configuration file or as command-line arguments.

Upon discovering a website, the crawler should establish a connection using sockets, send HTTP requests to retrieve the webpage data, and process the received responses. It will then extract URLs from the webpage content, filtering out irrelevant content (ex CSS, JavaScript files), and will schedule additional crawling tasks for linked sites.

- Tools used: C++, CMake, Git

2 Key project features

- Multithreading: will utilise multithreading for concurrent crawling of multiple websites, enhancing efficiency.
- Configurability: custom configuration settings can be provided, allowing users to customize parameters like crawl delay, maximum threads, depth limit, etc.
- URL Parsing: will utilise functions to extract the hostname and host path from a given URL.

- HTTP Response Handling: will parse HTTP responses to extract URLs and filter out irrelevant content such as CSS, JavaScript, PDF files, etc.
- Socket Communication: sockets will be utilised for HTTP communication with web servers.
- Error Handling: will implement error handling mechanisms for scenarios like DNS lookup failures, socket creation/connectivity issues, or failed requests.
- (Optional) Statistics Generation: will be able to compute and output statistics for each discovered website, such as response times, discovered pages, failed pages, and linked sites.

3 OS concepts employed

- Concurrency: threads will be used for concurrent execution of crawling tasks. Mutexes and condition variables will be used to synchronize access to shared resources (ex. pending site queue, discovered sites map) among threads, ensuring thread safety.
- File I/O: will be used to read configuration settings and outping statistics.
- Sockets: sockets will be used to establish connections with web servers, perform HTTP requests, and receive responses.
- Resource Management: system resources such as threads, sockets, and memory should be managed as effectively as possible.
- Thread Scheduling: scheduling mechanism will be used to distribute the crawling tasks among threads efficiently.
- Time Management: time-related OS functions will be used, as well as concepts for controlling the crawling process, including enforcing crawl delays between requests and measuring response times for statistics generation.