TCSS 422

PROJECT 1: WEB CRAWLER

[Gian Lendl V. Bata](https://www.facebook.com/glvbata)

Thang Tran

Levon Kechichian

Khanh Nguyen

***Web Crawler – Single Thread Implementation***

***(Please add the Jsoup jar file to the library)***

When launching the program, the web crawler GUI will show up. Users should type or copy-paste in a link to the first text box (“Enter the URL”), for example: [http://www.cplusplus.com](http://www.cplusplus.com/) . In the second text box, “Enter the Keywords”, users can enter a string of keywords (a string can be up to 10 words). To stop the crawling process, user must type in a number of maximum page they want the program to run through. This has to be a positive integer, otherwise the program will crash since we don't have any check for these cases. When everything is ready, click Go. If users is done with the program, click Exit or just close the program.

The web crawler program then will be executed, and display the number of pages it has retrieved, average word per page, average URLs per page, as well as the average parse time per page and total running time of the program. Below all these information is the panel that display all the keywords from the user input string, average words per page and total word from all the page web crawler program has retrieved. The user input string is delimited into separate keywords.

Once the user type in all the information, the retriever is initialized which has a parser, then analyzer is initialized. After the retriever retrieves the first page for the parser, while in the while loop, the parser gets a new page from the retriever with the URL from top of the stack, then collects all the URLs from that page to populate the stack. The parser collects the text from each page and sends the list of text to the analyzer which then analyzes information and sends them to the GUI for displaying.

***Web Crawler –Multi Thread Implementation***

After launching the program, the inputs are entered through the console with number of pages, url, and keywords enter in that order. The program then should display the output after its process. The same information is displayed as the Single Threaded output.

This Multithreaded program was designed by creating bridging classes to solve the many consumer producer problems present in the relationships between the classes. The most complex relationship is the Parser and Retriever who are both a Producer and a Consumer to each other. This is done by Boolean flags as well as queues to store the URL’s as well as the pages grabbed. The “Bridge” class for these two consumer/producer classes is a sort of a double Bridge class which serves a go between for the two classes (implemented by having each class maintain a reference to that same instance of that Bridge class). These bridge classes are also present for the Parser to Analyzer and Analyzer to GUI although these Bridge Classes are simpler because it’s a one-way producer consumer relationship.

The program starts with the retriever retrieving the first initial page and placing that page into the bridge class where the parser can be alerted then told to grab that page for parsing. The parsing gets the url from that page and places a url into the bridge class for the retriever to grab a new page with the url and repeat this process until the url’s stop coming or if the the max number of urls are reached.

As the parser grabs pages it stores those page’s text into the bridge class between the parser and the analyzer who then grabs the text and stores it into a data structure for analysis. The analyzer, when told by the Parser that the Parser is done, sends the output to the GUI where it can display.

***Web Crawler –Comparison***

Figure 1: Time per Page versus Pages

Figure 2: Total Time versus Pages

As indicated by Figure 2. As the number of pages search increases Single Threaded implementation takes up more time in total then Multithreaded. It would appear that Multithreaded has a linear growth while single may have polynomial growth. In terms of processers, single threaded has no bearing as the quad core took 3222 milliseconds to do 9 pages while the dual core takes 2999 milliseconds. No correlation could be found between Single Threaded dual and quad core programs. For multithreaded programs, quad core programs found a slight increase in efficiency with 1275 milliseconds for 9 pages with a quad core and 1475 for dual cores.

Furthermore, the implementation for the Multithreaded was much for simpler and easier to design because of the producer consumer problem provides for an easy abstraction to rely upon whereas the single threaded path required thoughts about designs on who has what and how information was transmitted to between objects. For example, the Parser required a complicated algorithm involving a stack to recursively have the Parser send url’s to the Retrievers and have the Retrievers retriever pages for the Parser to repopulate the stack with the new page from the Retriever. This is solved simply with a bridge class that can be the go between the classes as they do their own processes.

In conclusion, not only is Multithreaded more efficient then Single Threaded whenever it’s appropriate Multithreaded may also be the easier solution to design and code.