COMP30023 Project 1 – Web crawler

Worth 15% of the final mark

Due: 23:59:59 Friday 3 April, 2020.

1 Background

How do web search engines know what is on the web? They have to look for it the slow way, by following every link on the web. They use sophisticated algorithms to search efficiently. For example, they don't follow each link equally often; content that changes often is followed more often.

In this project you will write simple code to crawl a web site. This will teach you about socket programming, which is fundamental to writing all internet applications, and also about the HTTP application layer protocol. The crawler must be written in C and cannot use any existing HTTP libraries.

2 Crawling

The program will be given a URL on the command line, pointing to an HTML page. The program will be called `crawler` and it might, for example, be invoked as:

```
crawler http://web1.comp30023
```

The program will first fetch the indicated HTML page. The program will then recursively follow all hypertext links (`href="..."` attributes in `<a ...>` anchor tags) and fetch the indicated pages *if* all but the first components of the host match the host of the URL on the command line. or the complete hostname of that URL

For example, if the original host is `web1.comp30023` then a link to `web2.comp30023` should be followed, but a link to `comp30023` or `128.250.106.72` or `unimelb.edu.au` should not.

For the purpose of this project, it can be assumed that pages which need to be crawled will always have the MIME-Type of `text/html`. When the MIME-Type header is missing, the crawler is not required to guess the media type of the entity-body and does not have to parse the page.

Note that filename extensions are not the same as file types. Some files may end in `.html` or `.htm` but the header indicates that the MIME-type is text/plain. On the other hand, a file may have an extension `.txt` — or no extension — but have a MIME-type of text/html. The MIME-type defines the true type of the file.

3 Project specification

The program must perform several tasks, and can expect certain things of the testing setup. These are described as follows, grouped by task.

3.1 Crawling

The code should crawl as described in the previous section.

No page should be fetched more than once. Two pages are considered to be "the same page" if the URLs indicate that they are the same. That is different from "having the same URL", because multiple URLs can point to the same page. For example, in a document at `http://www.comp30023.example/a.html`, the URLs `http://www.comp30023.example/b.html` and `/b.html` refer to the same page. Relative URLs also exist, like `b.html`. However, pages `one/b.html` and `two/b.html` are different pages, despite ending in the same file name. The equivalence of pages is governed by the specification of URLs, given at the end of this document.

The program should print a log of the URLs it attempts to fetch — whether successfully or unsuccessfully. The log should be sent to `stdout`. Print one URL per line, with no other characters.
The program should send no other output to `stdout`; other output may be sent to `stderr`, which will be ignored for assessment.

The order of fetching pages does not matter.

You do not need to fetch more than 100 distinct pages (but there may be more than 100 URLs). (Hint: How can you use this knowledge to simplify your code?)

3.2 HTTP

The servers hosting the URLs you are to crawl will conform to HTTP 1.1 and all content will be served over port 80.

You must provide a User-Agent header consisting of your username (only), such as

```
User-Agent: joeb
```

This will be used to observe which pages you fetch. Without this, you risk getting no marks.

Note that all requests must also contain the Host header[1].

The server may fail to serve a page that you request, and either indicate that with an error code or simply send a file shorter than the length in the header. You should keep reading data until the connection is closed by the server or `Content-Length` bytes have been read. For *transient* (non-permanent) errors, you can re-request such pages without penalty, but not that the transient error condition may last for quite a while.

No server response will be longer than 100,000 bytes.

3.3 Parsing HTML

The HTML files will be valid HTML (unlike nearly all real web pages; web browsers silently fix many mistakes).

The file will not contain the characters "<" or ">" except at the start and end of tags. No anchor tags will be split across lines.

There will be no `<base>` tags or canonical link elements.

Both inline and external javascript should be ignored.

Submissions may (optionally) use the `pcre` library for pattern manipulation, or the standard functions declared in `regex.h`. Please read the submission instructions carefully if you plan to use `pcre`.

3.4 Parsing URLs

There will be no URLs using protocols other than `http`. In particular, there will be no `https`.

[1] `https://tools.ietf.org/html/rfc2616#section-14.23`

Note that URLs are partially case sensitive; see the references at the end of this document. HTML tags and field names are case-insensitive.

You can ignore URLs containing "`.`" and "`..`" path segments[2] (nominally "same directory" and "parent directory"), character codes of the form `%XY` or the characters `#` and `?`. URLs of these forms will exist in the data, and you may parse them if you wish.

You do not need to parse URLs longer than 1000 bytes.

Here are some examples of URLs which will not appear:

- `http://web1.comp30023:8080` (not port 80)
- `https://web1.comp30023` (protocol not http)
- `mailto:no-reply@unimelb.edu.au` (protocol not http)

Here are some examples of URLs which may be ignored:

- `./a1` and `bar/../`
- `http://web1.comp30023/assignments/./`
- `http://web1.comp30023/assignments/a2/../a1`
- `http://web1.comp30023/search?q=comp30023` (contains ?)
- `http://web1.comp30023/assignments#a1` (contains #)
- `http://web1.comp30023/%20` (contains URL encoded character)

3.5 Extensions

There are several versions of this project, each more challenging than the previous and each with a higher possible mark.

The more challenging extensions are intended as options. If you are finding the project too big, stick to a simple version.

3.5.1 Minimum requirement

Only considers

- anchor tags that are the first tag on the line;
- anchor tags of the form `<a href="...">`;
- whether the status code is 200 (success) or not 200 (treated as a permanent failure);
- pages whose length header is correct (no truncated pages).

3.5.2 Basic Extension

Correctly omits (or re-fetches) truncated pages.

Only parses pages that have MIME-type text/html (irrespective of the extension).

Considers all anchor tags on a line. Considers anchor tags with other fields between the `<a` and the `href="..."`. (No URLs in the page of the first URL will be of that form. You can get the marks for B and C below without handling these anchor tags.)

[2] `https://tools.ietf.org/html/rfc3986#section-5.2.4`

3.5.3 Medium Extension

It is expected that most students will not attempt this option.

Responds correctly to responses with status codes 200, 404, 410, 414, 503, 504. Clearly document in your code why you think your code's response is a "correct" response in each case.

(Note that you can group these into three cases, and treat all status codes within a group the same. One group would be "permanent failures".)

Note that responses with other status codes may be thrown by the server (e.g., when a malformed request is made) but need not be parsed.

All text/html pages should be parsed and crawled, including error report pages. The crawler does not have to (but can) parse the page when other status codes are received.

The crawler will attempt to revisit a page when the status code of a response indicates a temporary failure. Such URLs may be revisited without penalty. There will be no redirect loops or pages which always return a temporary failure.

3.5.4 Advanced Extension

It is expected that only one or two students will attempt this option.

Respond correctly to responses with status codes 301 and 401.

For 401, authentication should be handled using the `Authorization` header and the Basic Authentication Scheme[3]. Only supply the `Authorization` header when the server asks for authentication (indicated by response with status code 401).

Use your own username as `userid`, and the string "password" (without quotes) for `password`. You may hard code the base64 encoded user-pass string.

4 Marking Criteria

The marks are broken down as follows:

	Marks	Task
A	1	User-Agent header is valid
B	1	First request is valid
C	1	Valid requests for all URLs linked from the first URL
D	1	Produces log of URLs you *tried* to fetch
E	1	Handles HTTP status codes correctly
F	1	Basic extension works correctly
G	1	Medium extension works correctly
H	1	Advanced extension works correctly
I	1	Code quality
J	1	Build quality
K	5	Fetches pages once each

4.1 Code quality

Factors considered in code quality include: Choice of variable names, comments, indentation, remaining debugging code (it is OK to have a small amount if, for example, it is disabled by an #ifdef DEBUG), useful output indicating progress.

[3] `https://tools.ietf.org/html/rfc2617#section-2`

If the program crashes for any reason, the code quality mark will be 0. If there is any input that you don't know how to handle (e.g., a very long URL), ignore it rather than trying to handle it and crashing.

4.2 Build quality

Running `make clean && make && ./crawler http://XYZ` should execute the submission. If this fails for any reason, you will be told the reason, and allowed to resubmit (with the usual late penalty) but lose the build quality marks.

Compiling using "`-Wall`" should yield no warnings.

4.3 Fetching each pages once each

A score of +0.1 is awarded for each page from the evaluation site fetched exactly once. The score is reduced by -0.1 for each page fetched more than once (excluding times the server fails to send a valid page).

Note that multiple URLs can refer to the same page.

This score is clipped to be between 0 and 5.

5 Submission

You must push your submission to the repository named `comp30023-2020-project-1` in the subgroup with your username of the group `comp30023-2020-projects` on `https://gitlab.eng.unimelb.edu.au`.

We will harvest this repository on the due date (and on each subsequent day for late submissions). If you do not use your git repository for the project you will not have a submission and will be awarded zero marks.

If you have enrolled in the subject late and cannot see your repository, please comment on the discussion thread named "Project 1: Gitlab issues" on the LMS.

The repository must contain a Makefile that produces an executable named "`crawler`", along with all source files (and libraries) required to compile the executable. Place the Makefile at the root of your repository, and ensure that running `make` places the executable there too. Make sure that your Makefile, source code and dependencies are committed and pushed. Do not add/commit object files or executables.

Your Makefile must set up, compile and link any libraries which are required to compile and run your program, and your repository must include the source of the libraries. External libraries (including `pcre`) *will not* be preinstalled in the testing environment.

Place a git tag named "submission" on the snapshot you wish to be marked. If you do not know how to place a git tag, then ask on the forum many days before the deadline.

For your own protection, it is advisable to commit your code to git at least once per day. Be sure to `push` after you `commit`. You can push debugging branches if your code at the end of a day is in an unfinished state, but make sure that your final submission is in the `master` branch.

The git history will *not* be marked, but may be considered for matters such as special consideration and potential plagiarism.

5.1 Late penalty

There will be a late penalty of 2 marks per day.

6 Testing

A server will be established on the cloud that you can practice crawling. This will be similar to the site you will be evaluated on, but will not be the same. Details will be provided soon.

You should also push regularly to benefit from automated testing that we have set up on Gitlab Pipelines. Copy the sample `.gitlab-ci.yml` from the `comp30023-2020-projects/project-1` repository and push to have your submission tested against a basic, non-exhaustive set of pre-deadline tests.

7 Collaboration

You can discuss this project abstractly with your classmates, but should not share code. If possible, do not even look at anyone else's code. If you really want to help someone debug, don't look at the screen and pretend you're doing it over the phone. (The project says that every year, but this year you probably don't need to pretend!)

7.1 "Real programmers use Stack-Exchange"

Code will be run through software to detect copied code. You are welcome to use code fragments from Stack Exchange, or any other source of information on the web, but be sure to quote the URL whose code you used, so that it doesn't appear that you copied a classmate. Also, be warned that much of the code on question and answer sites is buggy. If you do reuse code, check it carefully to make sure it does what it claims to.

8 References

- Authoritative definition of HTTP: `https://tools.ietf.org/html/rfc2616` (actually, this was made obsolete by five RFCs)
- How to process 308 status code: `https://tools.ietf.org/html/rfc7538#section-3`
- Understandable guide to HTTP: `https://jmarshall.com/easy/http`
- Authoritative definition of HTML: `https://html.spec.whatwg.org/multipage`
- Understandable guide to HTML: `https://www.w3schools.com/html/html_basic.asp`
- Understandable guide to anchor tags: `https://www.w3schools.com/tags/tag_a.asp`
- Authoritative definition of URLs: `https://tools.ietf.org/html/rfc3986`
- guide to URLs: `https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_URL`