COMP3310 - Assignment 2: Indexing a Gopher.

Background:

- This assignment is worth 15% of the final mark.
- **It is due by 23:55 Thursday 17 April AEST (end of Week 7)**
- Late submissions will not be accepted, except in special circumstances.
  - Extensions must be requested as early as possible before the due date, with suitable evidence or justification, via the extension app on the wattle site.
- ***If you would like feedback on particular aspects of your submission, please note that in the report with your submission.***

This is a coding assignment, to enhance and check your network programming skills. The main focus is on native socket programming, and your ability to understand and implement the key elements of an application protocol from its RFC specification. Your submission will be a single zip package of your program file(s) together with a text report, as outlined below.

Please note that this is an ongoing experiment for the course, trialling gopher for this assignment. We may discover some additional challenges as we go, that requires some adjustments to the assignment activities, or a swap of server. Any adjustments will be noted via a forum Announcement.

Assignment 2 outline

An **Internet Gopher** server was one of the precursors to the web, combining a simple query/response protocol with a reasonably flexible content server, and a basic model for referencing and describing resources on different machines. The name comes from the (Americanised) idea to "*go-for*" some content… and also the complexity of their interconnected burrows[1].

For this assignment, you need to write your own gopher client in C, Java or Python[2,3], **without** the use of any external gopher-related libraries. The client will need to 'spider' or 'crawl' or 'index' a specified server, do some simple analysis and reporting of what resources are there, as well as detect, report and deal with any issues with the server or its content.

Your code MUST open sockets in the standard socket() API way, as per the tutorial exercises. Your code MUST make appropriate and correctly-formed gopher requests on its own, and capture/interpret the results on its own. You will be handcrafting gopher protocol packets, so you'll need to understand the structures of requests/responses as per the gopher RFC 1436.

We will provide a class gopher server to ultimately run against, with a mix of content – text and binary files, across some directory structure, along with various other pointers to resources. And possibly exhibiting some poor behaviours.

In the meantime, you SHOULD install a gopher server on your computer for local access, debugging and wiresharking to get familiar with the packet formats. There are a number available, with pygopherd perhaps

[1] https://en.wikipedia.org/wiki/Gopher

[2] As most high-performance networking servers, and kernel networking modules, are written in C with other languages a distant second, it is worth learning C, one day. But, time is short, and everyone has a different background.

[3] If you want to use another language (outside of C/Java/Python), discuss with your tutor – it has to have native socket access, and somebody on the tutoring team has to be able to mark it.

the more recently updated but more complex, and Motsognir, which is a bit older but simpler. If you find another good one, please share on the forum.

Wireshark will be very helpful for debugging purposes. Remember to be conservative in what you send and reasonably liberal in what you accept.

What your successful and highly-rated indexing client will need to do:

1. Connect to the class gopher server, and get the initial response.
    a. Wireshark (just) this **initial-response** conversation in both directions, from the starting TCP connection (SYN) to its closing, and include a screenshot of that wireshark summary in your report (don't expand to the lowest level, one row per packet/message is plenty).
    b. *The class gopher site is not yet fully operational, an announcement will be made when it's ready.*

2. Starting with the initial response, automatically scan through the directories on the server, following links to any other directories on the same server, and download any text and binary (non-text) files you find. The downloading allows you to measure the file characteristics. You don't need to keep the downloads, apart from those for the summary below. Keep scanning till you run out of unique references to visit. Note that there will be items linked more than once, so beware of getting stuck in a loop.

3. While running, prints to STDOUT:
    a. The timestamp (time of day, local time) of each request, with
    b. The client-request you are sending. This is good for debugging and checking if something gets stuck somewhere, especially when dealing with a remote server.

4. Count, possibly store, and (at the end of the run) print out:

    a. The number of Gopher directories on the server.
    b. The number, and a list of all simple text files (full path)
    c. The number, and a list of all binary (i.e. non-text) files (full path)
    d. The contents of the smallest text file.
    e. The size of the largest text file.
    f. The size of the smallest and the largest binary files.
    g. The number of unique invalid references (those with an "error" type)
    h. A list of external servers (those on a different host and/or port) that were referenced, and whether or not they were "up" (i.e. whether they accepted a gopher connection on the specified port).
        i. You should only connect to each external server (unique host+port combination) once. Don't crawl their contents! We only need to know if they're "up" or not.
    i. Any references that have "issues/errors", that your code needs to explicitly deal with.

Requests that return errors, or that had to abort (e.g. due to a timeout, or for any other reason) do not count towards the number of (smallest/largest)(text/binary) files. Put a copy of the results in your report.

You will need to keep an eye on your client while it runs, as some items might be a little challenging if you're not careful… Not every server provides perfectly formed replies, nor in a timely fashion, nor properly terminates, for example. Identify any such situations you find on the gopher server in your report or code comments, and how you dealt with each of them – being reasonably liberal in what you accept and can interpret, or flagging what you cannot accept. Good Internet practice.

We will test your code against the specified gopher, and check its outputs. If you have any uncertainties about how to count some things, you can ask your tutor or in the forum. In general, if you explain in your report how you decide to count things and handle edge-cases, that will be fine.

You can make your crawler's output pretty or add additional information if you'd like, but don't go overboard. We need to be able to easily see everything that's listed here.

Submission and Assessment

There are a number of existing gopher clients, servers and libraries out there, many of them with source. While perhaps educational for you, the assessors know they exist and they will be checking your code against them, and against other submissions from this class.

You need to submit your source code, and the report. Any instructions to run the code, and any additional comments and insights, please provide those in the report. Your submission must be a zip file, packaging everything as needed, and submitted through the appropriate link on wattle.

Your code will be assessed on [with marks available, out of a total of 30]

1. Output correctness [14 marks]
    - Does the gopher server correctly respond to all of your queries?
    - Does your code report the right numbers? (within your interpretation, perhaps)
    - Does your code cope well with issues it encounters?
    - Does your code provide the running log of requests as above?
2. Performance [3 marks]
    - A great indexer should run as fast as the server allows, and not consume vast amounts of memory, nor take a very long time. There won't be too many resources on the server.
3. Code "correctness, clarity, and style" [13 marks]
    - Use of native sockets, writing own gopher requests correctly.
    - Documentation, i.e. comments in the code and the report - how easily can somebody else pick this code up and, say, modify it.
    - How easy the code is to run, using a standard desktop environment and the latest versions of everything.
    - How does it neatly handle edge-cases, where the server may not be responding perfectly.

During marking your tutor may ask you to explain some particular coding decisions.

Reminder: Wireshark is very helpful to check behaviours of your code by comparing against existing gopher clients (some are preinstalled in Linux distributions, or are easily added). There are a number of youtube videos on gopher as well that e.g. show how the clients work. Your tutor can help you with advice (direct or via the forum) as can fellow students. It's fine to work in groups, but your submission has to be entirely your own work.