COMP3310 Assignment 2

Alvin Tang (u7447218)

Due 26 April 2024

This PDF version is provided as a backup for the Markdown file (`README.md`).

1 Gopher Indexing Client

This Internet Gopher client written in the C programming language indexes directories and files. It is implemented in accordance with RFC 1436. The overall design is to ensure that outbound requests strictly adhere to the protocol standard whilst handling inbound responses as gracefully as possible.

1.1 Initialisation

The program is intended for Linux machines with GCC installed. The standard C library is sufficient for compilation. Use the command `make` to compile the program with the source code. Simply execute the command `./client <hostname> <port>` to run the program. For example, if the Gopher server is hosted locally at port 70, `./client localhost 70` connects to it. An IP address can also be directly used. As an example, the command `./client 127.0.0.1 70` is also valid.

For local testing, a local Gopher server can be started using Motsognir with the command `sudo motsognir`. The listening port and the process can be listed using the command `sudo lsof -i :70`.

1.2 Program Design

The C programming language is used because of its extensive use in native Linux programs and its efficiency relative to alternatives (*e.g.*, Python, Java). A standard Linux installation should be able to execute the program; no additional packages are required.

The overall design relies on performing a breadth-first search (BFS) of the filesystem hosted on the Gopher server. The indexed files and directories are recorded in a linked list implemented using the `entry` struct. Each `entry` contains the type of the record (`item_type`), the record itself (`record`). The pointer `next` directs to the next entry in the linked list.

There are six types of records.

Entry type (`item_type`)	Record (`record`)
`DIRECTORY`	Pathname to the directory
`TEXT`	Pathname to the text file
`BINARY`	Pathname to the binary file
`ERROR`	Invalid request line
`EXTERNAL`	Request line referencing an external server
`TIMEOUT`	Pathname to the file taking excessive time to download
`TOO_LARGE`	Pathname to the file exceeding the size limit

Since the protocol is stateless and the connection is terminated once the server has responded, the function `gopher_connect()` establishes and closes a connection every time a new request is made. The function `gopher_connect()` appends “`\r\n`” to the request which is passed as the parameter `request`. It sends the request line to the server. The function `func()`, passed as another argument to `gopher_connect()`, is responsible for receiving the response from the server.

Instantiations of `func()` include `indexing()`, `evaluate_file_size()` and `print_response()`. With reference to the protocol guidelines, the client listens for a response until the server terminates the connection. Therefore, these three functions use `recv()` provided by the Socket API in a while loop. The built-in `timeval` struct and the `errno` library in C are utilised for handling timeouts and failed responses. Notice that a failure to receive a response is handled differently from an empty string response from the server.

1.2.1 Recording Indexed Files and Directories

As long as the server adheres to the protocol's format, its response to a pathname is always the directory index with the first character of a row indicating the nature of the file. For instance, `0` indicates that the file is a (non-binary) text file whereas `1` refers to a subdirectory or an external server. The function `indexing()` uses this information to determine the type of the file/directory. A new entry is added to the linked list, including the item's pathname and type.

If the request is invalid, the corresponding response begins with `3`. In this case, a new entry is added to the linked list if and only if it does not already exist. This enables us to count the number of invalid references at a later stage.

With reference to RFC 1436, the canonical type `9` refers to binary files. Actual server implementations often have more specific types and non-canonical types that gained popularity after RFC 1436 was published. In this client program, files that are not in plain text are considered binary.

Classification	File types
Directory	`1`
Reference to external server	`1`
Text file	`0`
Error	`3`
Binary file	`4` (BinHex-encoded file), `5` (DOS file), `6` (uuencoded file) `9` (Binary file), `g` (GIF), `I` (Image), `:` (Bitmap) `;` (Movie), `<` (Audio), `d` (Document), `h` (HTML) `p` (PNG), `r` (RTF), `s` (Sound), `P` (PDF), `X` (XML)

Any row in the response starting with the character `i` is a human-readable informational message, thus ignored by the indexation process. Other references such as Telnet (`8` and `T`), CCSO nameserver (`2`) and mirror (`+`) do not fall within any of the categories above. These types are therefore disregarded by the indexation process.

1.2.2 Recursively Index Subdirectories

Following the indexation of the root directory using the request "`\r\n`", `main()` goes through the linked list and calls `gopher_connect()` for every subdirectory. This effectively models a breadth-first search of the filesystem hosted on the Gopher server.

1.2.3 Evaluation and Loading of File Content

Upon indexation of all directories and files in the filesystem, `evaluate()` is called to find the following information:

1. Number of directories, text files and binary files
2. Content of the smallest text file (if multiple exist, the first one indexed)
3. Sizes of the smallest and the largest text files
4. Sizes of the smallest and the largest binary files
5. Number of invalid references (each unique invalid request is counted once)

6. List of external server references (hostname and port) and their connectivity
7. List of references causing issues/errors

Note that if the server ends its response with "`.\r\n`" as per the protocol standard, the three characters are included in the size counts. This helps with determining whether the server is replying with proper responses.

The helper function `test_external_servers()` checks the connectivity to external servers by attempting to establish a connection. A server is considered "up" if it accepts the connection in a few seconds. Otherwise, it is marked as "down".

1.2.4 Terminal Output

As the client program runs, logs are printed to the terminal (`stdout` and `stderr`). The terminal outputs to `stdout` are in the following formats.

- `Request sent at <timestamp>: <request>`
- `Indexed <item type>: <pathname of newly indexed item>`
- `Number of <directories/text files/binary files/invalid references>: <number>`
- `Size of the <smallest/largest text/binary file>: <number>`
- Other information produced by `evaluate()`

Issues and errors are printed to the terminal through `stderr`:

- `Usage: <program name> <hostname> <port>`
- `Error: <error message>`
- `File too large: <pathname>`
- `Transmission timeout: <pathname>`

1.2.5 Minimising Errors and Maximising Security

Whilst this client program follows the Gopher protocol standard in RFC 1436, measures are taken to reduce the impact of malformed responses from servers. For instance, the response is written to a string variable and is always explicitly terminated with the null terminator ('`\0`'). This avoids problems such as out-of-bound reading/writing.

Safer built-in functions in C (such as `strncpy()` rather than `strcpy()`) are used to avoid unexpected behaviour that might possibly be the result of long requests or responses. Indexed records in the linked list, with memory allocated using `malloc()`, are cleaned up by `cleanup()` before `main()` terminates.

At the beginning of `gopher_connect()`, the program is terminated if the connection cannot be established. An error message is printed to `stderr` specifying the issue.

1.2.6 Handling Empty Responses and Timeouts

A correctly implemented server should respond with some content under all circumstances. "Empty response from the server" is printed to the terminal if the server terminates the connection without any response. This clarifies to the user whether the connection is terminated or the client is still waiting for a response.

Servers may be busy or unresponsive at times. There are two types of timeouts.

1. The server accepts the connection but does not respond to a request.
2. The server accepts the connection and responds but it takes too much time.
3. The server does not accept the connection within the time limit.

Timeouts prevent the program from getting stuck indefinitely. The first situation is managed using the function `setsockopt()`, a built-in feature in the Socket API, and the `timeval` struct. The configuration is set by `gopher_connect()`. The limit is currently set as 10 seconds.

The second situation is handled using `select()` and `fcntl()`. The configuration is set by the function responsible for receiving the response. The limit is currently defined as 5 seconds.

The third situation applies in `test_external_servers()` with a 5-second limit.

1.2.7 Handling Edge Cases: Malformed or Non-Standard Responses

A well-implemented server's directory index response should include four pieces of information, separated by tab characters. In case a line therein is malformed, the program handles it gracefully to avoid indexing invalid things. It is also designed to handle long responses from the server.

Motsognir, for example, ensures that every line including errors and informational messages includes three tab characters as separators.

```
3error fake fake 0
iInformational fake fake 0
```

Other servers format these responses differently. The class server, for instance, uses `invalid` rather than `fake`.

```
3error invalid 0
iInformation invalid 0
```

The client program ensures correct handling of responses when interacting with different servers.

A server's response should end with a line containing "`.\r\n`" followed by the null terminator. However, if the server does not follow the standard protocol to the full extent and terminates transmission without "`.\r\n`", the program still handles the response gracefully. The client program perceives a server's termination of the connection as the end of data transmission. An example is `/misc/malformed2` on the class server.

Unicode was not published at the time the Gopher protocol was publicised. We assume that pathnames contain only ASCII characters, or else converted into ASCII representations on the server side (as it is in Motsognir). For example, a file named "ɡoʊfər" is detected as "%C9%A1o%CA%8Af%C9%99r".

Some malformed responses omit the hostname and the port in an entry of the directory index. Examples include `/misc/malformed1`. This client still attempts to index the file, assuming it exists on the server. If the file does not exist in the server, the pathname is regarded as an invalid reference.

1.2.8 Handling Edge Cases: Exceptionally Large Files

As part of the `evaluate()` function's execution, the client has to download the files for analysis of their properties, such as their size. The files may sometimes be unexpectedly large. The global constant `FILE_LIMIT` specifies the maximum file size that the client may accept. This ensures that the client will not be flooded with infinite/unacceptable network traffic, of which malicious parties may take advantage. Files exceeding the size limit are not considered in the statistics (*e.g.*, sizes of largest files) evaluated by `evaluate()`.

Currently, `FILE_LIMIT` is set as 65536 bytes, considering that most files on `comp3310.ddns.net` do not exceed this limit. In realistic situations, users of this client should know the size scale of files that the target server hosts. This global constant can be adjusted accordingly. Files exceeding the file limit are logged and printed before the client program execution ends.

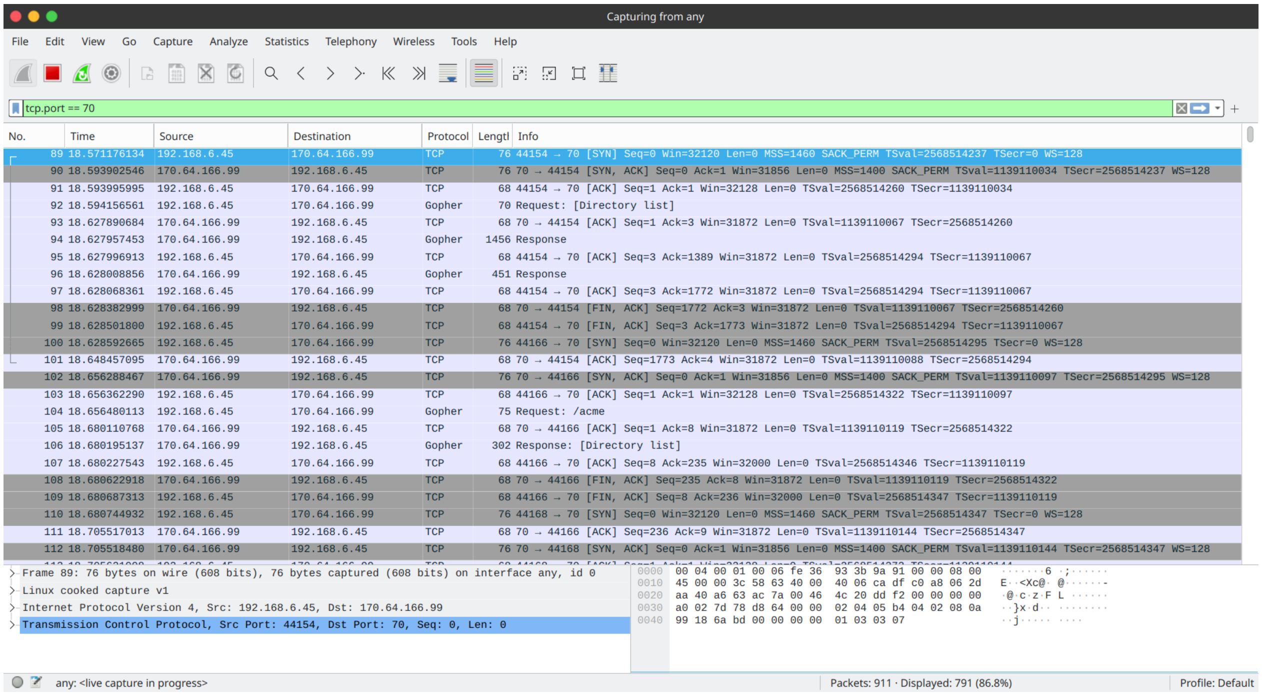
1.3 Testing

The client program is tested with the class server and a local Motsognir server. The terminal outputs are included for reference:

- `assets/output1.txt` — Class server, 65536-byte file size limit.

- `assets/output2.txt` — Local Motsognir server, no file size limit.

Wireshark is used for monitoring the network traffic. The initial request is detected as “Request: [Directory List]”. There are occasional retransmissions of TCP packets and spurious acknowledgement (ACK) signals.



```
tcp.port == 70

No.  Time         Source         Destination    Protocol Lengtl Info
  89 18.571176134 192.168.6.45   170.64.166.99  TCP        76 44154 → 70 [SYN] Seq=0 Win=32120 Len=0 MSS=1460 SACK_PERM TSval=2568514237 TSecr=0 WS=128
  90 18.593902546 170.64.166.99  192.168.6.45   TCP        76 70 → 44154 [SYN, ACK] Seq=0 Ack=1 Win=31856 Len=0 MSS=1400 SACK_PERM TSval=1139110034 TSecr=2568514237 WS=128
  91 18.593995995 192.168.6.45   170.64.166.99  TCP        68 44154 → 70 [ACK] Seq=1 Ack=1 Win=32128 Len=0 TSval=2568514260 TSecr=1139110034
  92 18.594156561 192.168.6.45   170.64.166.99  Gopher     70 Request: [Directory list]
  93 18.627890684 170.64.166.99  192.168.6.45   TCP        68 70 → 44154 [ACK] Seq=1 Ack=3 Win=31872 Len=0 TSval=1139110067 TSecr=2568514260
  94 18.627957453 170.64.166.99  192.168.6.45   Gopher   1456 Response
  95 18.627996913 192.168.6.45   170.64.166.99  TCP        68 44154 → 70 [ACK] Seq=3 Ack=1389 Win=31872 Len=0 TSval=2568514294 TSecr=1139110067
  96 18.628008856 170.64.166.99  192.168.6.45   Gopher    451 Response
  97 18.628068361 192.168.6.45   170.64.166.99  TCP        68 44154 → 70 [ACK] Seq=3 Ack=1772 Win=31872 Len=0 TSval=2568514294 TSecr=1139110067
  98 18.628382999 170.64.166.99  192.168.6.45   TCP        68 70 → 44154 [FIN, ACK] Seq=1772 Ack=3 Win=31872 Len=0 TSval=1139110067 TSecr=2568514260
  99 18.628501800 192.168.6.45   170.64.166.99  TCP        68 44154 → 70 [FIN, ACK] Seq=3 Ack=1773 Win=31872 Len=0 TSval=2568514294 TSecr=1139110067
 100 18.628592665 192.168.6.45   170.64.166.99  TCP        76 44166 → 70 [SYN] Seq=0 Win=32120 Len=0 MSS=1460 SACK_PERM TSval=2568514295 TSecr=0 WS=128
 101 18.648457095 170.64.166.99  192.168.6.45   TCP        68 70 → 44154 [ACK] Seq=1773 Ack=4 Win=31872 Len=0 TSval=1139110088 TSecr=2568514294
 102 18.656288467 170.64.166.99  192.168.6.45   TCP        76 70 → 44166 [SYN, ACK] Seq=0 Ack=1 Win=31856 Len=0 MSS=1400 SACK_PERM TSval=1139110097 TSecr=2568514295 WS=128
 103 18.656362290 192.168.6.45   170.64.166.99  TCP        68 44166 → 70 [ACK] Seq=1 Ack=1 Win=32128 Len=0 TSval=2568514322 TSecr=1139110097
 104 18.656480113 192.168.6.45   170.64.166.99  Gopher     75 Request: /acme
 105 18.680110768 170.64.166.99  192.168.6.45   TCP        68 70 → 44166 [ACK] Seq=1 Ack=8 Win=31872 Len=0 TSval=1139110119 TSecr=2568514322
 106 18.680195137 170.64.166.99  192.168.6.45   Gopher    302 Response: [Directory list]
 107 18.680227543 192.168.6.45   170.64.166.99  TCP        68 44166 → 70 [ACK] Seq=8 Ack=235 Win=32000 Len=0 TSval=2568514346 TSecr=1139110119
 108 18.680622918 170.64.166.99  192.168.6.45   TCP        68 70 → 44166 [FIN, ACK] Seq=235 Ack=8 Win=31872 Len=0 TSval=1139110119 TSecr=2568514322
 109 18.680687313 192.168.6.45   170.64.166.99  TCP        68 44166 → 70 [FIN, ACK] Seq=8 Ack=236 Win=32000 Len=0 TSval=2568514347 TSecr=1139110119
 110 18.680744932 192.168.6.45   170.64.166.99  TCP        76 44168 → 70 [SYN] Seq=0 Win=32120 Len=0 MSS=1460 SACK_PERM TSval=2568514347 TSecr=0 WS=128
 111 18.705517013 170.64.166.99  192.168.6.45   TCP        68 70 → 44166 [ACK] Seq=236 Ack=9 Win=31872 Len=0 TSval=1139110144 TSecr=2568514347
 112 18.705518480 170.64.166.99  192.168.6.45   TCP        76 70 → 44168 [SYN, ACK] Seq=0 Ack=1 Win=31856 Len=0 MSS=1400 SACK_PERM TSval=1139110144 TSecr=2568514347 WS=128

> Frame 89: 76 bytes on wire (608 bits), 76 bytes captured (608 bits) on interface any, id 0
> Linux cooked capture v1
> Internet Protocol Version 4, Src: 192.168.6.45, Dst: 170.64.166.99
> Transmission Control Protocol, Src Port: 44154, Dst Port: 70, Seq: 0, Len: 0

0000  00 04 00 01 00 06 fe 36  93 3b 9a 91 00 00 08 00   ·······6 ·;······
0010  45 00 00 3c 58 63 40 00  40 06 ca df c0 a8 06 2d   E··<Xc@· @······-
0020  aa 40 a6 63 ac 7a 00 46  4c 20 dd f2 00 00 00 00   ·@·c·z·F L ······
0030  a0 02 7d 78 d8 64 00 00  02 04 05 b4 04 02 08 0a   ··}x·d·· ········
0040  99 18 6a bd 00 00 00 00  01 03 03 07               ··j····· ····

any: <live capture in progress>        Packets: 911 · Displayed: 791 (86.8%)        Profile: Default
```

References

[1] F. Anklesaria, M. McCahill, P. Lindner, D. Johnson, D. Torrey, and B. Alberti. RFC 1436: The Internet Gopher Protocol, March 1993. URL `https://www.rfc-editor.org/rfc/rfc1436`.

[2] caf. C: socket connection timeout. Stack Overflow, April 2010. URL `https://stackoverflow.com/a/2597774`.

[3] dgookin. The `gettimeofday()` Function. C For Dummies Blog, July 2020. URL `https://c-for-dummies.com/blog/?p=4236`.

[4] Foobar. Client in C++, use `gethostbyname` or `getaddrinfo`. Stack Overflow, October 2018. URL `https://stackoverflow.com/q/52727565`.

[5] Frédéric Hamidi. What does `EAGAIN` mean? Stack Overflow, October 2010. URL `https://stackoverflow.com/a/4058377`.

[6] IEEE and The Open Group. `setsockopt`, 2003. URL `https://linux.die.net/man/3/setsockopt`.

[7] Arkaitz Jimenez. Time stamp in the C programming language. Stack Overflow, September 2009. URL `https://stackoverflow.com/a/1444440`.

[8] Milan. How to convert string to IP address and vice versa. Stack Overflow, March 2011. URL `https://stackoverflow.com/a/5328184`.

[9] Basile Starynkevitch. How to use `select()` on sockets properly? Stack Overflow, September 2015. URL `https://stackoverflow.com/a/32711642`.

[10] Tomek Szpakowicz. When to use static keyword before global variables? Stack Overflow, December 2009. URL `https://stackoverflow.com/a/1856642`.

[11] Wikipedia contributors. Gopher (protocol). Wikipedia. URL `https://en.wikipedia.org/wiki/Gopher_(protocol)`.