uget: A Multi-Protocol File Retriever for Niche and Classic Web Protocols

Svetlana Tadevosyan
Supervisor: Norayr Chillingaryan
May, 2025

Made by Oberon

Supported Protocols	Port	Rationale
gemini://	1965	Lightweight alternative to the web. Popular among minimalists, privacy-conscious users, and alternative content platforms.
spartan://	300	Even more minimal than Gemini. Useful for simple plain text documents; allows access to ultra-minimal servers.
nex://	1900	Custom/local experimental protocol. Demonstrates extensibility of uget to support future or niche protocols.
http://	80	Standard websites and file servers. Most widely used protocol for transferring web data and files.
https://	443	Secure version of HTTP with encryption. Allows fetching from secure sites and APIs.

Protocol properties

	Encryption	Header (Response)
gemini://	TLS	Response header: status code and MIME type or meta info (e.g. 20 text/gemini)
spartan://	None	Response header: MIME type only, followed by a blank line and the body (e.g. text/plain)
nex://	None	
http://	None	status line (HTTP(s)/1.1 200 OK), followed by key-value headers (Content-Type, Content-Length, etc.)
https://	TLS	

TLS (Transport Layer Security) and TLS Lib

TLS is a protocol that encrypts data to ensure secure communication over the network.

In uget, TLS is used to support secure protocols like HTTPS and Gemini, which require encrypted connections.

The library **mbedtls** was chosen because it's lightweight and made it easy to write simple wrappers for TLS sockets in Oberon.

Why Oberon?

Modular Design Philosophy!

Oberon is a language built around simplicity and modularity. This matches well with the architecture of uget, which is composed of cleanly separated modules for each protocol (HTTP, HTTPS, Gemini, Nex, Spartan).

Future Compatibility with Oberon OS!

In theory, the project can later be ported or directly used under the Oberon operating system, which is minimal, secure, and elegant. This aligns with the long-term goal of making uget lightweight and system-friendly.

Ease of Extending!

Thanks to Oberon’s module system, new protocol handlers can be plugged in later without affecting the existing codebase. This makes uget scalable and flexible for future development.

Educational Value!

Writing in Oberon forces clarity of thought and results in compact, readable, and correct code. It helped maintain focus on the core logic of each protocol without distraction from excessive language features.

Related Work

curl (1997) – Versatile command-line tool supporting many protocols like HTTP, FTP, and more.

wget (1996) – Recursive downloader for HTTP, HTTPS, and FTP, popular for scripting.

netcat / nc (1995) – Low-level networking tool for reading/writing TCP/UDP connections.

Lagrange (2020) – Graphical Gemini browser focused on user-friendly browsing.

Amfora (2020) – Terminal-based Gemini client written in Rust, focused on minimalism.

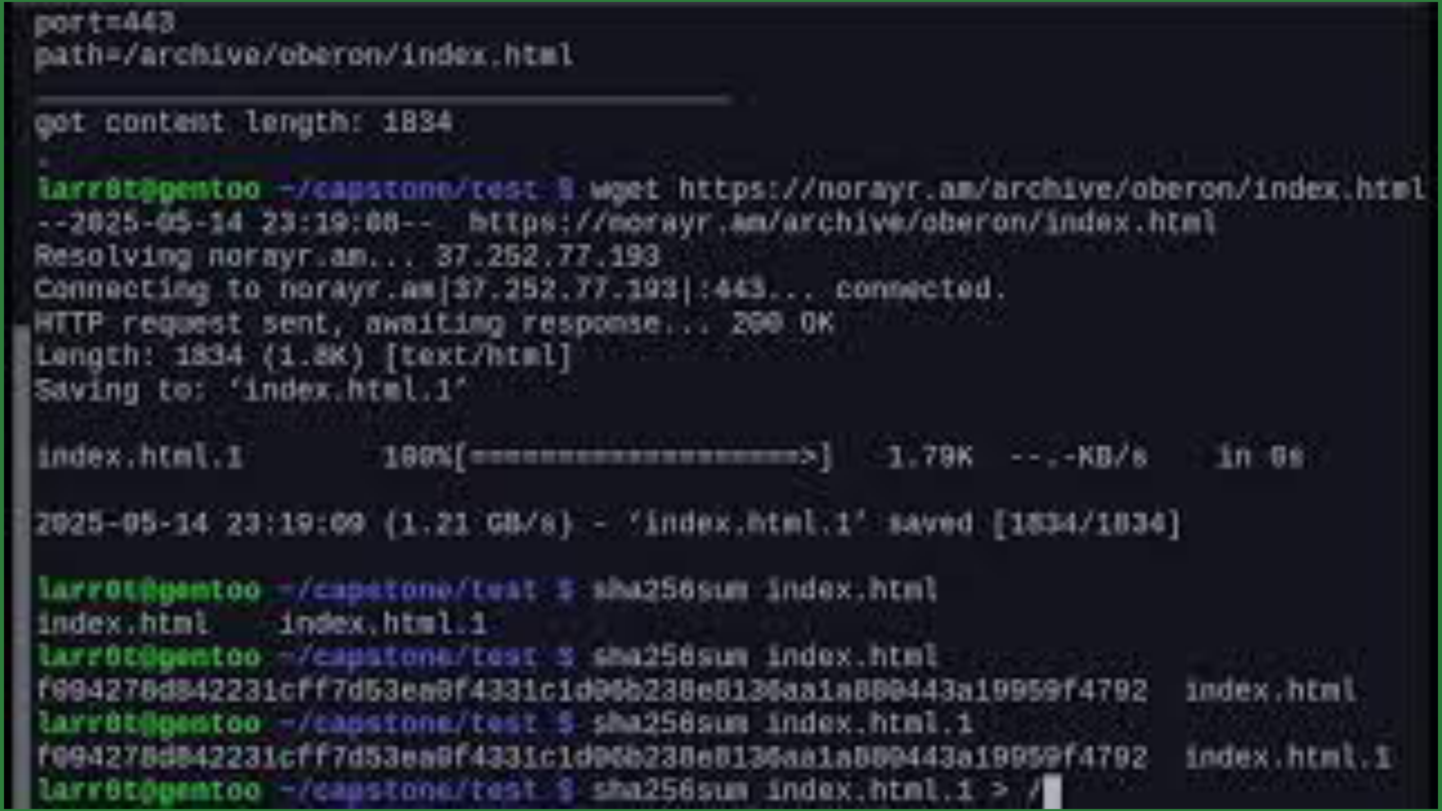
libgemini (2020) – Lightweight Gemini protocol implementation used in simple clients.

fetch (BSD tool, 1998) – Simple file download tool mainly for HTTP/FTP on BSD systems.

Plume (2021) – Gemini protocol blog engine, showcases how Gemini is used beyond just browsing.

Verifying file Integrity with Sha256 crypto checksum

After downloading files using uget, a SHA-256 cryptographic hash was computed. The hash was compared against the expected checksum provided by the source.



```
port=443
path=/archive/oberon/index.html
────────────────────────────────────────
got content length: 1834

larr0t@gentoo ~/capstone/test $ wget https://norayr.am/archive/oberon/index.html
--2025-05-14 23:19:08--  https://norayr.am/archive/oberon/index.html
Resolving norayr.am... 37.252.77.193
Connecting to norayr.am|37.252.77.193|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1834 (1.8K) [text/html]
Saving to: ‘index.html.1’

index.html.1        100%[====================>]   1.79K  --.-KB/s    in 0s

2025-05-14 23:19:09 (1.21 GB/s) - ‘index.html.1’ saved [1834/1834]

larr0t@gentoo ~/capstone/test $ sha256sum index.html
index.html    index.html.1
larr0t@gentoo ~/capstone/test $ sha256sum index.html
f094278d842231cff7d53ea0f4331c1d06b238e8136aa1a800443a19959f4792  index.html
larr0t@gentoo ~/capstone/test $ sha256sum index.html.1
f094278d842231cff7d53ea0f4331c1d06b238e8136aa1a800443a19959f4792  index.html.1
larr0t@gentoo ~/capstone/test $ sha256sum index.html.1 > /
```

How did we deal with unknown content length?

>Some protocols don't provide content length upfront, we used a dynamic array structure that automatically resizes when needed.

>The array's capacity doubles each time it grows, ensuring we can append more data without worrying about overflow.

>This approach allows us to handle variable-length content from multiple sources reliably and efficiently.

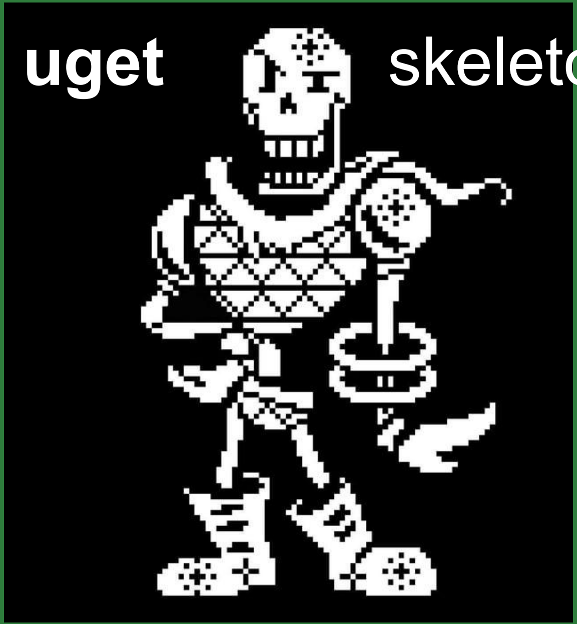
```
DEFINITION dynamicarray;

 TYPE
       DynamicarrayDesc = RECORD (dynarray)
       size-: INT32;
       capacity-: INT32;
       content-: POINTER TO ARRAY OF CHAR;
       appender-: PROCEDURE (a: dynamicarray; str: ARRAY OF CHAR);
       END;
       dynamicarray = POINTER TO DynamicarrayDesc;
       dynarray = RECORD [100H]
       END;

 PROCEDURE Append(arr: dynamicarray; str: ARRAY OF CHAR);
 PROCEDURE Create(): dynamicarray;
 PROCEDURE Init(arr: dynamicarray; size: INT32);
 PROCEDURE Resize(arr: dynamicarray): dynamicarray;
 PROCEDURE writetofile(arr: dynamicarray; name: ARRAY OF CHAR): dynamicarray;

END dynamicarray.
```

uget skeleton



```
TYPE
 ARG = RECORD
        protocol: ARRAY 64 OF
CHAR;
        host: ARRAY 64 OF CHAR;
        path: ARRAY 246 OF CHAR;
        port: ARRAY 5 OF CHAR;
        prothost: strTypes.pstring;
END;


PROCEDURE ConnectSpartan;
PROCEDURE ConnectNex;
PROCEDURE ConnectGemini;
PROCEDURE ConnectHttp;
PROCEDURE ConnectHttps;
PROCEDURE parseCommand(cmd: ARRAY OF CHAR);
PROCEDURE getCommand(i: INTEGER);
PROCEDURE Main > authentication
```

Future Development

>> Browser with support for more protocols

>> Support for Gopher and Guppy protocols

>> Functionality Refinement

>> Development with Oberon!

Thank you!