

Dokumentacja HackTTP

1) Ogólne założenia projektu

HackTTP to prosty serwer HTTP zrealizowany w oparciu o mechanizmy systemowe.

Zaimplementowane funkcje obejmują:

- 1) Realizowanie podstawowych metod HTTP (POST, GET)
- 2) Obsługę kilku formatów plików (ascii text, jpg, png)
- 3) Obsługę skryptów CGI
- 4) Obsługę cookies

Dodatkowe funkcje to m.in:

- 1) Logowanie aktywności serwera
- 2) Obsługa wybranych sygnałów (SIGINT, SIGUSR1)
- 3) Wczytywanie danych z pliku konfiguracyjnego

2) Zastosowane mechanizmy systemowe:

- **Sockety**

- Dwukierunkowe punkty końcowe połączeń stanowiące podstawowe narzędzie komunikacji serwera z aplikacjami klienckimi
- Umożliwiają strumieniową (SOCK_STREAM) wymianę danych między procesami przy użyciu protokołu TCP
- Wywołanie kolejno funkcji socket(), bind() i listen() powoduje uzyskanie deskryptora pliku odpowiadającego server socketowi, przypisanie go do konkretnego portu oraz rozpoczęcie oczekiwania na nadchodzące połączenia - jest to zasadniczy element działania serwera
- Wywołanie accept() po nadejściu połączenia powoduje uzyskanie kolejnego deskryptora pliku, odpowiadającego socketowi realizującemu konkretne połączenie - umożliwia to m.in. wielowątkową obsługę requestów
- Odbieranie requestów i wysyłanie odpowiedzi zachodzi przy użyciu poleceń recv() i send()
- Server socket po nawiązaniu połączenia nadal pozostaje powiązany z portem i oczekuje na kolejne połączenia
- Za utworzenie server socketa oraz akceptowanie kolejnych połączeń w HackkTTP odpowiada klasa Router. Za zarządzanie pulą wątków (jeden wątek na połączenie) odpowiada klasa Manager. Za obsługę poszczególnych połączeń odpowiada klasa Worker, przy udziale BasicHTTP (parsowanie requestów i składanie nagłówków odpowiedzi) i DataHandler (realizowanie zapytań, tzn. ładowanie plików, uruchamianie skryptów CGI).

- **Wątki/procesy**

- Narzędzie umożliwiające jednoczesną obsługę kilku nadchodzących do serwera requestów
- Każdy Worker działa w osobnym wątku, ilość dostępnych Workerów (a więc ilość jednocześnie obsługiwanych połączeń) jest definiowana w configu.
- Dodatkowo w celu odczytywania odpowiedzi ze skryptów CGI tworzymy nowe podprocesy przy użyciu poleceń fork/exec w DataHandler::Exec

- **Potoki (pipes)**

- Umożliwiają prostą komunikację między wątkami
- Wykorzystywane podczas obsługi skryptów CGI (DataHandler::Exec) - deskryptor pliku, do którego pisze moduł odpalający skrypt, powiązany jest ze standardowym wejściem podprocesu, w którym działa skrypt, podobnie deskryptor pliku z którego czyta moduł odpalający skrypt, powiązany jest ze standardowym wyjściem podprocesu

- **Sygnały (signals)**

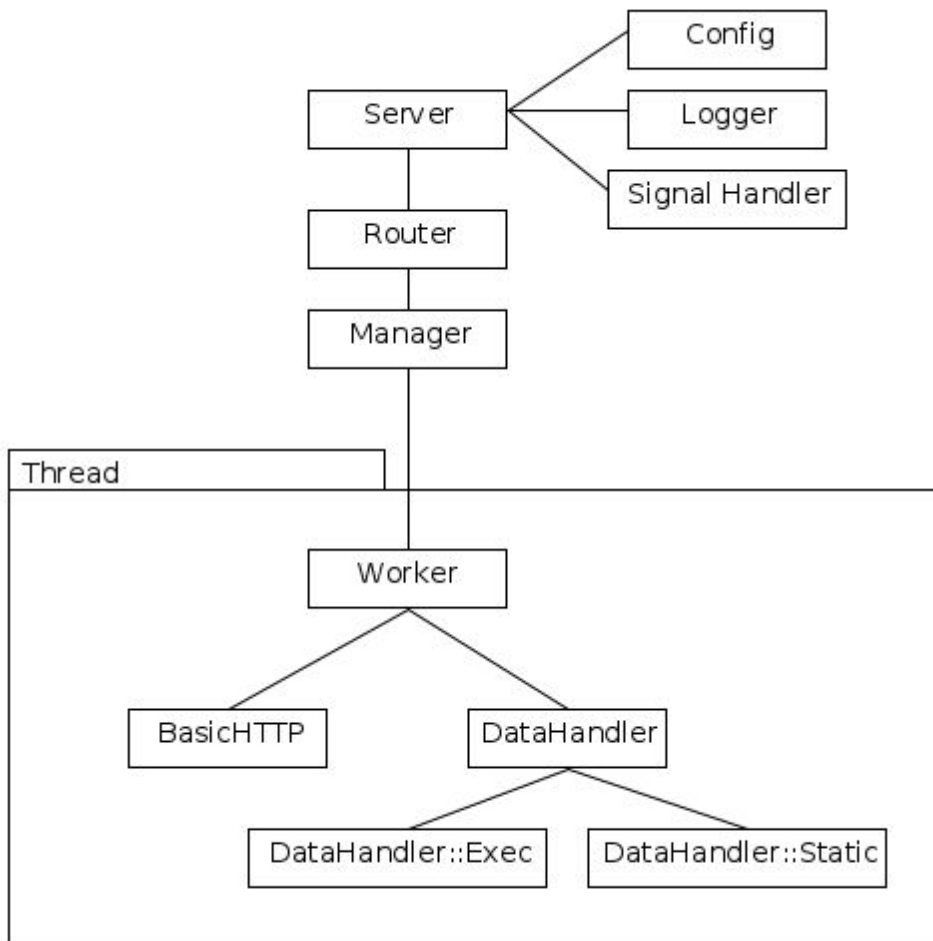
- Narzędzie sterowania serwerem i asynchronicznego przekazywania poleceń, niezależnie od stanu wykonania programu
- Serwer obsługuje 2 sygnały:
 - SIGINT - zakończenie pracy serwera
 - SIGUSR1 - ponowne wczytanie danych z pliku konfiguracyjnego

- **Mechanizmy obsługi plików**

- open(), read(), write() oraz close()

- Wczytywanie informacji z pliku konfiguracyjnego oraz zapisywanie danych do loga
- Komunikacja między wątkami (pisanie do potoków)

3) Podstawowe moduły programu:



Server - main(), inicjalizuje komponenty potrzebne do działania serwera

Router - Utworzenie i obsługa server socket, przekazuje deskryptory plików zwrócone przez accept() do Managera. Monitoruje wartość zmiennej isSigintRecieved, jej zmiana na true powoduje zakończenie nasłuchiwanie na server socket i zakończenie pracy serwera. Funkcje:

- int init_socket() - utworzenie socketa, przypisanie go do portu zdefiniowanego w configu. Zwraca deskryptor server socket
- void watch() - rozpoczyna oczekiwanie na nadchodzące połączenia na server socket, akceptuje je i przekazuje do Managera.

Manager - Zarządzanie Workerami: znajdowanie wolnego Workera, przekazywanie Workerom deskryptorów plików zwróconych przez accept() Funkcje:

- void handle_request() - tworzy nowy wątek dla nowego Workera, obsługuje wyjątki związane z tworzeniem wątków
- int get_free_worker_index() - znajduje indeks wolnego Workera w puli Workerów, obsługuje wyjątek związany z brakiem wolnych Workerów

- void *worker_runner() - przekazywana jako argument do pthread_create, odpowiada za stworzenie nowego Workera oraz wywołanie na nim funkcji handle_request()

Worker - Zasadnicza klasa obsługująca request HTTP. Funkcje:

- void handle_request() - przy użyciu klas pomocniczych (BasicHTTP, DataHandler) sprawdza poprawność requestu (HTTP_BAD_REQUEST), parsuje request oraz przygotowuje odpowiedź. Obsługuje wyjątki (HTTP_UNSUP_MEDIA_TYPE, HTTP_FORBIDDEN, HTTP_NOT_FOUND).
- void send_msg() - wysyła przygotowaną w handle_request odpowiedź

BasicHTTP - Podstawowa obsługa HTTP: sprawdzanie poprawności requestu, parsowanie, przygotowywanie nagłówków do odpowiedzi. Funkcje:

- bool is_valid() - Proste sprawdzenie poprawności requestu (kryteria: wersja 1.0 lub 1.1, metoda: POST lub GET)
- request parse_request() - Przygotowuje strukturę request o polach: string method, string uri, string cookies, DataHandler::resource data, bool valid, string http_version. W zależności od wartości pola method wywołuje handle_post_method() lub handle_get_method()
- response render_headers() - przygotowuje nagłówki dla odpowiedzi
- string fetch_cookies() - zwraca cookies, jeżeli są dostępne
- void handle_get_method(), void handle_post_method() - obsługa metod GET i POST, tj. poprawne (adekwatne do metody) odczytanie danych wejściowych z zapytania, które później mogą zostać przekazane do skryptu CGI

DataHandler - w oparciu o zidentyfikowany typ zasobu (używając polecenia systemowego "file") na który wskazuje zapytanie, uruchamia:

- DataHandler::Static - służący do ładowania danych z plików statycznych takich jak:
 - pliki tekstowe (w tym HTML/CSS)
 - obrazki (JPG, PNG oraz ikony)
- DataHandler::Exec - służący do uruchamiania skryptów CGI lub dowolnych innych plików wykonywalnych, następnie przekazania danych wejściowych z zapytania i odczytania odpowiedzi ze standardowego wyjścia

Config - Wczytuje plik konfiguracyjny, umożliwia ponowne wczytanie pliku po otrzymaniu SIGUSR1. Funkcje:

- void loadConfigFileToMap() - otwiera plik, wczytuje dane, zapisuje je do mapy
- int get_int_setting(), string get_str_setting() - zwracają wartość odpowiadającą podanemu kluczowi. Jeżeli program otrzymał SIGUSR1 to ponownie wczytują zawartość pliku konfiguracyjnego

Logger - Wyświetla loga na konsoli oraz zapisuje go do pliku, którego ścieżka jest określona w configu. W celu pisanie do pliku korzysta z małego LoggingSingletona korzystającego z mutexów. Funkcje:

- void log()

SignalHandler - Odpowiada za obsługę sygnałów. Nadchodzący sygnał ustawia globalną flagę (isSigintRecieved, isSigusrRecieved). Flagi są okresowo sprawdzane przez Router oraz Config. Funkcje:

- void sigintHandler(), void sigusrhandler() - ustawiają odpowiednie flagi