Contents

Una descrizione rigorosa di ogni file, funzione e struttura, generata con Doxygen a partire dalla documentazione sul codice sorgente, è disponibile qui.

1 Utilizzo del software

E' possibile compilare il progetto tramite make. La compilazione genera l'eseguibile linuxserver o, nel caso in cui si esegua la build su Windows, il file winserver.exe e due binari aggiuntivi. Viene effettuato il linking alla libreria pthread su GNU/Linux e alla libreria Winsock (Ws2_32) su Windows.
Il programma riconosce le seguenti opzioni da riga di comando:

```
-d DIR
   Utilizza la directory DIR come root del server.
-f FILE
   Leggi le configurazioni da FILE.
   Per informazioni sulla struttura del file, vedere la sezione
   1.1 Lettura della configurazione.
-h
   Mostra la sintassi.
-l FILE
   Logga i trasferimenti sul file FILE.
   Default: logfile.
-m
   Utilizza un processo per ogni client.
   Default: no.
-p PORT
   Metti il server in ascolto sulla porta PORT.
   Default: 7070.
-v
   Imposta il livello di log informativi del server
   (syslog su Linux, console su Windows). I livelli sono:
   0 - Non loggare nulla.
   1 - Logga gli errori.
   2 - Logga gli errori e i warning.
   3 - Logga gli errori, i warning e i messaggi di info.
   4 - Logga errori, warning, messaggi di info e messaggi di debug.
   Default: 2.
```

Le opzioni da riga di comando hanno precedenza su quelle scritte nel file di configurazione.
Se non viene specificata alcuna root, verrà usata la directory contenente l'eseguibile.

1.1 Lettura della configurazione

Se viene utilizzato un file di configurazione con l'opzione -f, le impostazioni non specificate da riga di comando vengono lette dal file dato in input. Questo deve essere una sequenza di righe del tipo CHIAVE = VALORE, dove sono accettate le seguenti coppie:

- port = <numero di porta>

- multiprocess = yes/no
- verbosity = 1..4
- logfile = <path>
- root = <path>

La lettura avviene all'avvio dell'applicazione e ogniqualvolta venga ricevuto il segnale SIGHUP (caso Linux) o CTRL_BREAK (caso Windows) durante il main loop del server; tuttavia, nel caso in cui si ricarichino le configurazioni mentre il server è in esecuzione, non verranno aggiornati né il path di logging né la root directory.

2 Struttura del progetto

Il progetto è strutturato con un approccio modulare gerarchico. Ogni modulo esegue uno specifico compito all'interno dell'applicazione, servendosi delle funzionalità dei livelli inferiori ed esponendo funzionalità ai livelli superiori. A livello implementativo, a ogni modulo corrisponde uno o più header e uno o più file sorgente. I quattro moduli sono:

- Server - Riceve le richieste di connessione dei client e crea nuovi thread o processi per inviare le risposte. Si occupa inoltre della lettura dei file di configurazione e della gestione dei segnali per l'interazione con l'utente.
- Protocollo - Ha il compito di leggere, validare, interpretare e soddisfare la richiesta di un singolo client, costruendo la risposta secondo lo standard del protocollo Gopher. In questo modulo avviene la lettura del file system e l'eventuale mapping in memoria dei file.
- Logger - Gestisce il processo di logging, che si occupa di registrare le operazioni di invio di file.
- Compatibilità - API minimale che offre diverse funzioni di supporto portabili. Espone interfacce a varie chiamate di sistema e altre operazioni comuni non portabili, nonché wrapper di tipi di dato.

Per quanto il progetto sia stato impostato con struttura modulare, le sue componenti sono comunque interconnesse e progettate per coesistere in un'unica applicazione. Per questo motivo, la maggior parte delle configurazioni pertinenti a tutto il programma ha scope globale.
Ogni file espone, tramite il proprio header, funzioni utili a espletare i compiti previsti dal modulo logico a cui il file fa riferimento. Queste funzioni utilizzano come valore di ritorno delle costanti per indicare il successo o il fallimento dell'operazione (a volte messe in OR con altre costanti per fornire ulteriori informazioni sull'esito).

Le dichiarazioni utilizzano i tipi definiti in datatypes.h, per astrarre dai tipi specifici utilizzati dalla libreria Win32 o dalle funzioni POSIX. L'implementazione interna delle funzioni invece utilizza quando conveniente i tipi propri della piattaforma, e consiste, laddove impossibile scrivere codice portabile o utilizzare solo funzioni offerte dal modulo di Compatibilità, di un'implementazione ad hoc per Windows e una per GNU/Linux, demandando la scelta dell'implementazione da utilizzare al preprocessore.

3 Scelte implementative

3.1 Server

Il modulo Server è implementato dall'header server.h e dal file server.c.
L'header dichiara la struttura server_t, utilizzata come interfaccia:

```
/** A struct representing an instance of a gopher server */
typedef struct {
    /** The socket of the server */
    socket_t sock;
    /** The socket address */
    struct sockaddr_in sockAddr;
    /** The port the server is listening on */
    unsigned short port;
    /** A flag stating whether the server should spawn a process or a thread per request */
    bool multiProcess;
} server_t;
```

La maggior parte delle funzioni dichiarate dall'header richiede in input un puntatore a una struttura server_t, dalla quale leggere o nella quale scrivere informazioni.

Per supportare l'interazione utente, i gestori dei segnali utilizzano le variabili booleane globali

```
    static sig_atomic volatile updateConfig;
    static sig_atomic volatile requestShutdown;
```

che vengono lette periodicamente fintanto che il server è in ascolto. L'accesso a queste variabili è protetto da una CRITICAL_SECTION nel caso Windows e dall'uso del tipo sig_atomic_t (wrappato in sig_atomic per comodità) nel caso Linux, che assicura l'accesso atomico alle variabili in presenza di interrupt asincroni. Nel caso in cui la lettura delle configurazioni dovesse fallire vengono utilizzate le impostazioni di default, salvate in globals.h.

3.1.1 Avvio del server

Il server viene avviato con la funzione runServer, che prende in input un puntatore a una struttura server_t opportunamente inizializzata. La funzione attende con una select una richiesta di connessione, controllando periodicamente la ricezione di segnali. Questa scelta è motivata dal fatto che gli eventi da console di Windows non interrompono la select. Quando il server riceve una richiesta di connessione, genera un nuovo thread o processo (a seconda della modalità utilizzata), il quale passa il controllo al protocollo tramite la funzione gopher Quando il server riceve una richiesta di terminazione, la funzione runServer ritorna con successo. Il modulo Server non si occupa di liberare le proprie risorse.

3.2 Protocollo

Il modulo Protocollo è implementato dall'header protocol.h, dal file protocol.c e dal file winGopherProcess.c nel caso in cui si compili su Windows.
L'header espone all'esterno, oltre a diverse costanti, una singola funzione gopher. La funzione scansiona l'input dal socket ricevuto fin quando non rileva CRLF. L'input letto viene validato (poiché il selettore stesso viene utilizzato per recuperare la risorsa richiesta, non può contenere costrutti per la navigazione del file system, quali ./ ../ etc) e normalizzato, per consentire una maggiore flessibilità circa le richieste accettate. Se la connessione viene chiusa o si verifica un errore, la funzione fallisce; il protocollo si incarica di chiudere il socket del client sia in caso di fallimento che in caso di errore.

3.2.1 Costruzione della risposta

La lista di file disponibili, se richiesta una directory, viene costruita utilizzando la funzione iterateDir, la quale funge da interfaccia per le primitive specifiche delle piattaforme Linux e Windows, operando in modo analogo.
Il carattere prefisso a ogni riga della risposta, che rappresenta il tipo del file, viene calcolato cercando determinate parole chiave nell'output del comando "file" nel caso Linux; nel caso Windows viene letta l'estensione del file: le estensioni riconosciute sono memorizzate nell'array extensions, e l'appartenenza di un'estensione a un tipo di file (testo, immagine...) viene calcolata attraverso delle maschere di bit. Ogni maschera rappresenta un gruppo di estensioni e ha settati a 1 i bit che corrispondono agli indici degli elementi dell'array extensions che fanno parte di quel gruppo.
Se viene richiesto un file, questo viene mappato in memoria utilizzando la funzione di compatibilità getFileMap, che wrappa le rispettive funzioni mmap di Linux e CreateFileMapping/MapViewOfFile di Windows (queste ultime racchiuse in un'unica funzione poiché non è previsto il mapping in memoria di porzioni diverse dello stesso file). Viene poi creato un nuovo thread, che si occupa di inviare il file mappato in memoria al client, di chiudere la connessione e di costruire la stringa da passare al logger tramite la funzione logTransfer.

3.3 Logger

Il modulo Logger è implementato dall'header logger.h, dal file logger.c e in aggiunta dal file winLogger.c, utilizzato nel caso in cui si compili su Windows per generare l'eseguibile con il quale avviare il processo.
L'header dichiara la struttura logger_t, utilizzata come interfaccia:

```
 /**
 * A struct representing an instance of a transfer logger
*/
typedef struct {
    /** Pipe to use for read/write */
    pipe_t logPipe;
    /** Pointer to the mutex guarding the log pipe */
    mutex_t* pLogMutex;
    /** [Linux only] Pointer to the condition variable to notify the logger for incoming data */
    cond_t* pLogCond;
    /** [Windows only] Event to notify the logger for incoming data */
    event_t logEvent;
    /** The pid of the log process */
    proc_id_t pid;
} logger_t;
```

Come nel caso del server, le funzioni comunicano con i rispettivi chiamanti tramite puntatori a strutture logger_t. L'header espone tre funzioni: per avviare il logger, per interromperlo e per loggare una stringa. A causa delle differenze tra Windows e Linux circa l'IPC, le due implementazioni sono completamente distinte.

3.3.1 Linux

Nel caso Linux, la funzione di avvio startTransferLog inizializza una pipe, un mutex e una condition variable in uno spazio di memoria condiviso e ne scrive i riferimenti nella struttura logger_t ricevuta in input. Dopodiché viene effettuata una fork per avviare il processo che esegua il loop principale. Il loop principale apre il file di logging e attende sulla condition variable del logger ricevuto in input. Quando riceve una richiesta di logging (notificata sulla condition variable) acquisisce un lock esclusivo sul file di logging tramite fcntl e scrive la stringa sul file. La funzione per interrompere il logger si occupa di liberare tutte le risorse e di inviare SIGINT al processo.

3.3.2 Windows

Il logger di Windows funziona in modo analogo al caso Linux, fatta eccezione per alcune differenze, (principalmente dovute all'assenza della fork, che consentiva al processo di ereditare i puntatori alle aree di memoria condivise). Il mutex e l'evento (che sostituisce la condition variable di Linux) vengono creati con nome, per potervi poi accedere con facilità dal processo di logging. L'estremo

in lettura della pipe viene impostato come input del processo creato. Il loop principale è demandato a un eseguibile generato a partire dal file winLogger.c, che opera in modo analogo alla funzione di Linux.
Anche in questo caso la funzione di interruzione libera le risorse e termina il processo di logging mediante la funzione TerminateProcess.

3.4 Compatibilità

Il modulo di Compatibilità è implementato dagli header platform.h, datatypes.h, log.h e wingetopt.h, e dai sorgenti platform.c e wingetopt.c. Offre numerose funzionalità implementate in modo portabile per essere riutilizzate dai moduli che lo richiedano. Le funzioni si articolano in:

- utilità generiche (operazioni sulla working directory, logging...)
- operazioni sui socket
- operazioni su thread e processi
- operazioni sul file system
- wrapper per tipi di dato
- messaggi di log e debug uniformi
- parsing di opzioni da riga di comando.

Su queste funzioni non c'è molto da dire, poiché spesso sono in corrispondenza diretta con le corrispettive funzioni specifiche della piattaforma, a meno di qualche aggiunta o piccola modifica.

3.5 Main

Le componenti sopra descritte vengono combinate in maniera intuitiva nel main. Vengono creati i due oggetti server_t e logger_t per gestire rispettivamente server e logger. Dopo aver letto e settato opportunamente le configurazioni, viene eseguita la procedura per rendere il processo demone (nel caso Linux). Dopodiché vengono avviati, tramite le apposite funzioni, il processo di logging e il server.