Dokuz Eylul University
Dept. of Computer Engineering
www.cs.deu.edu.tr

Multi-Threaded Downloaded Manager (MTDM)

Technical Documentation
License: GPL

Ferad Zyulkyarov
Samet Basaran
Ozgur Deveci

Table of Contents

ABSTRACT

This document describes the design and implementation of multi threaded download manager. Implementation utilizes POSIX threads and is written in C/C++ programming language.

INTRODUCTION

A multi threaded download manager establishes multiple HTTP connections with the remote HTTP server. Each connection downloads a separate segment from the specified URL file. In this way the multiple threads perform a parallel download of different segments of the remote URL file. The achieved result of such technique is fast and accelerated download process compared to a single thread performed downloads.

APPROACH

Because of our restriction to use C or C++ programming languages to implement the download manager we will compare the advantages and disadvantages of both and decide on which language to use.

C

The main advantage of using C as a main programming language is that POSIX threads are implemented for C language. There will not be any difficulty on lunching threads and their management. Also C is respectively low level compared to C++, therefore some functions may appear to be easier for implementation than in C++. But the main an serious disadvantage of C is that it is not object oriented programming language (OOPL) as C++ is. What this means is that the program would not be easily extended if needed. For example, if we decide to add an additional functionality to download files from an FTP server we will need to alter great amount of the source code. This is rather unwanted condition, because code written in C is not often reusable.

C++

C++ is OOPL and once a code is written it can be reused and the program may be extended easily. C++ supports all functionality of C because C++ derived from C. The main disadvantage is that C++ functions are not compatible with POSIX threads. But this limitation can be overcome if C++ object is wrapped in a C function.

DESIGN

Upon discussions made in the previous section we conclude that C++ is superior than C and decide to implement the download manager in C++ programming language.

Main operation that has to be handled by the download manager are:

- Socket operations
  - Creating Socket
  - Binding Socket
  - Open Socket
  - Close Socket
  - Send Data Through Socket
  - Receive Data From Socket

- HTTP connection
  - Creating a HTTP connection
  - Open a HTTP connection
  - Close a HTTP connection
  - Sending a HTTP request
  - Receiving a HTTP response

- Storing remote file to local drive
  - Create file
  - Open file
  - Close file
  - Manage simultaneous write requests of multiple threads

- Download manager
  - Manages the parallel downloads performed by multiple threads
  - Keeps track for each thread involved in a download process

All of the processes listed above are slightly described in the forthcoming subsection and detaily described in Implementation section.

Architecture

Below is given a figure that depicts the conceptual model of the download manager with the components handling classified operations described at the beginning of this section.

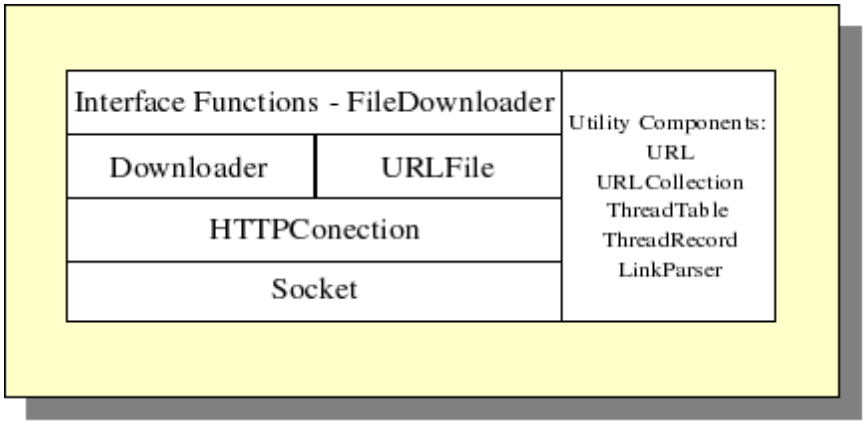

Figure 1 *– Download Manager Architecture*

Socket

Socket component, as earlier discussed, manages all socket operations. It creates, binds, opens, closes socket connections. This component also handles sending data though socket and receiving data from socket. This component is utilized by the one upper level component HTTPConnection.

HTTPConnection

HTTPConnection component manages HTTP connections established with the HTTP server. This component uses Socket component to connect to a HTT server, send request to HTTP server and receive responses from HTTP server. HTTPConnection component is used by Downloader and URLFile components.

Downloader

Downloader component manages download process that is performed by multiple threads in parallel. This component collaborates with the URLFile object to store downloaded data in a physical file.

Downloader object creates as many HTTP connections as requested by the Interface functions. After with the utilization of URLFile object it stores downloaded data in a consistent manner solving the race condition that occurs whenever two or more threads wants to write in the file.

Interface Functions - FileDownloader

This component plays the role of interface for the download manager. It lunches download threads as requested by the user and creates the objects that are involved in the download process.

This component is implemented in C programming language to provide an entry point function for POSIX threads that are being created. The function that is an entry point for each thread wraps Downloader object. The wrapping technique is shown in the code below.

```
class Downloader    // Downloader component
{
       public:
              bool downloadThreadId(int threadId);
       ...
       ...
}

//In FileDownloader component

int download_file(string url, int threadNum)
{
Downloader downloader(url, threadNum);  // Create a Downloader object
int status;
int i;

pthread_t threads[threadNum];

cout << "STATUS: Download started..." << endl;

for (i = 0; i<threadNum; i++)
{
       // Prepare threads argument
       // Wrap Downloader object and pass it as argument to the thread that will
       // be created
       DownloadFunctionAttribute dfa(&downloader, i);

       if ((pthread_create(&threads[i], NULL, download_hand, (void*)&dfa )) !=0 )
       {
              cerr << "ERROR: Download failed..." << endl;
              return ERROR;
       }

       sleep(1);
}

// Join threads
for (i = 0; i<threadNum; i++)
{
       pthread_join(threads[i], NULL);
}

downloader.saveFile();

return SUCCESS;
// Resources will be freed automaticaly by destrucotrs
}
```

```
// FileDownloader - Thread Entry Point
void *download_hand(void *arg)
{
DownloadFunctionAttribute *dfa = (DownloadFunctionAttribute*)arg;

Downloader *downloader = dfa->downloader; // Unwrap the Downloader object

int id = dfa->threadId;

downloader->downloadThreadId(id); // Start download for thread with ID=id

pthread_exit(0);
}
```

Utility Components

Utility components are objects that are utilized by all other components. They are not involved directly in the download process but provide services for the better operation of other components.

- URL – abstracts an URL address
- URLCollections – abstracts a list of URL addresses
- ThreadRecord – abstract the information for a thread necessary for its management
- ThreadTable – groups ThreadRecord objects. This object is used to synchronize multiple threads that are involved in a download.
- LinkParser – this object parser the content of a HTML file for hyperlinks.

USER MANUAL

This section is describes how to compile, run and test the program.

Compilation

Along with the sources we provide a Makefile for automatic compilation. Whatever is necessary to do is just to type `make` in the terminal under the the directory containing `Makefile`. As an output two binary executable files will be created – `downloadmanager.exe` and `comparefiles.exe`.

Running

Name of the executable file that downloads remote files is downloadmanager.exe. Its usage is

./downloadmanager.exe -d | -n [positive number] -u <URL>

Parameters that the program takes are:

- *-d* – downloads all hyperlinks in depth 1 found in <URL>, using this parameter MTDM acts as a web robot.
- *-n* – specifies the number of threads to use during the download process. By default *n* is equal to 5. The use of this parameter together with -d is not supported. For now you can use it with -u only.
- *-u* – specifies the target URL file that has to be downloaded.
- Parameters -d and -u are mutually exclusive, meaning that you can't use them at the same time.

Example Usage -u:

```
$./downloadmanager.exe -n 12 -u http://localhost/myfile.zip
```

The above command will download file named *myfile.zip* from host *localhost* using *12* threads binding to the default port *80*.

```
$./downloadmanager.exe -u http://www.sc.deu.edu.tr:8080/test.exe
```

The above command will download file named *test.exe* from host named *www.cs.deu.edu.tr* using threads *5* binding to port number *8080*.

Example Usage -d:

```
$./downloadmanager.exe -d http://www.linux.org:1010/index.html
```

The above command downloads all links found in file *index.html* hosted at server named *www.linux.org*.

Testing

You can test the download manager using the provided executable *comparefiles.exe*. Its usage is:

```
$./comparefiles.exe <filename1> <filename2>
```

- `filename1` could be the original file hosted in the server.
- `filename2` could be the file downloaded from the server.

comparefiles.exe performs binary comparison on both files and prints a report – *equal* or *not equal*.

IMPLEMENTATION

LINKPARSER

Public class **LinkParser**

LinkParser class implements url parsing accesses on a specified web file. This class takes source file name and host name and starts finding links in source file. Basic aim of this class finding links and parsing those with some operations so provide useful and non-cracked links for other classes needs.

LinkParser class at creating point uses another class 'URLCollection' for keeping information in. That is presenting some useful functions for accessing items well. In constructor point, an object that is at URLColection type, specified special capacity. It is optional for users defining capacity. This variable is used for keeping Urls that are finding in source file.

This class provides a public function 'getLinks' that is used for taking links in a variable that is a kind of data structure that is URLCollection object.

Field Detail

_sourcefile

```
private string _sourcefile
```

The string variable that is specified source file name ,is getting from user at creation object point. Used for specified source file name.

_hostname

```
private string _hostname
```

The string variable that is specified host name,is getting from user at creation object point. Used for some links which are without current hostname ,at that time used current hostname.

_urls

```
private URLCollection _urls
```

This is an object of URLCollection class. This is declared and at creation session is initialized according to special capacity info. This is provide to user to free selection on buffer choice.

Constructor Detail

LinkParser

Public

```
LinkParser(string hostname,  string filename)
```

Constructor is empty with the specified constant capacity and essential parameters.

Parameters

`hostname` – the current host name.

`filename` – the source file name that is used.

Method Detail

getLinks

```
public URLCollection& getLinks()
```

Get the links in source file with using the private 'getTAGlinks' function. This calls that function according to speciefied tags (i.e. HREF, SRC..). After that, return an object that is holding urls.

Returns

An object that is holding urls

---

getTAGLinks

```
private void getTAGLinks(string tag)
```

Foundation links eliminating operations is occured this function. According to getting tag , file is looking over for finding links, to purify from white spaces and unnecessary characters. And useful links is added buffer object that is URLCollection. Return

Parameters

`tag` – a string that is specifieded for tag that is looked over.

URLCOLLECTION

Public class **URLCollection**

URLCollection class is specified for holding urls and some useful function for moving and accessing on. This can be called basic data structer for urls.

This class specifies object on three creation. Default constructor doesn't get any parameter , capacity is defined constant 20. Copy constructor is used for cloning existing object, gets a address of existing object. Another one is take a parameter for initial capacity, that is an integer and specifies a capacity.

URLCollection has severel useful functions. Those are put, get, size which are for adding new url, getting existing one according to posetion, and size value.

Field Detail

_size

```
private int _size
```

Collection size by default 20.

_cap

```
private int _cap
```

Capacity of url buffer. Default value is 20.

_rec

```
private char **_rec
```

Array of URL that are to be downloaded.

Constructor Detail

URLCollection

Public

```
URLCollection()
```

This is a default constructor. _size and _cap value is initialized. Essential memory

is reserved for container _rec according to _cap value.

Puclic

```
URLCollection(URLCollection &url)
```

This is a copy constructor. The info of existing url object is used for initializing new one and reserving necessary memory.

Parameters

`url` – existing url object

Public

```
URLCollection(int cap)
```

This a constructor which is getting parameter capacity for specifeing obvious capacity value. Like this, first initial is implemented.

Parameters

`cap` – desired initial capacity value.

Method Detail

put

```
public void put(string url)
```

Inserts url into the collection. Each url in a collection is unique to avoid repeated downloads.

Parameters

`url` – the url to be inserted.

---

get

```
public char* get(int pos)
```

Retrieves the url provided by its unique index.

Parameters

`pos` – posetion of requested url in container.

Returns

If url already exists in container, that is returned, else null is returned.

---

getSize

```
public int getSize()
```

Specified number of urls in container.

Returns

Return size of container that is holding urls.

---

isurlExists

```
private bool isUrlExists(string url)
```

Check same url already exists or not

Parameters

`url` – url that is compare with existing ones.

Returns

If same url is existing returned True boolean value , else False.

URL

Class URL is used for parsing url subsets : protocol, hostname, port, identifier. If URL is cracked, it is an error. That time it sends error message to user about this event.

This class has three constructor. One is default is specified according to some common rules. This standart is for every part , protocol is HTTP , hostname is LOCALHOST, port number is 80, and last identifier is index.html. Copy constructor creates a new object like existing one. End last one creates an URL object specified by the url string.

This class provides to user useful functions that are using for getting part of url such as getport. And foundation operations as checking , parsing and so is doing private two functions that are parseUrl and parseHandUrl.

This is essential class for parsing url and checking cracked status.

Field Detail

_protocol

```
private string _protocol
```

it holds protocol of url. Default value for it is Http

_hostname

```
private string _hostname
```

it holds hostname of url. Default value for it is localhost

_port

```
private int _port
```

it holds number port of url. Default value for it is 80

_identifier

```
private string _identifier
```

it holds file name or directory of file with name. Default value is index.html

Constructor Details

URL

Public

```
URL():_protocol("http"), _hostname("localhost"),
_port(80), _identifier("index.html") {}
```

It is default constructor all part of url is firstly initialized.

Public

```
URL(string url)
```

Creates an URL object specified by the url string

Parameters

`url` – a string parameter that is url.

Public

```
URL(const URL &url)
```

Copy constructor that creates an object from URL class , from existing one.

Parameters

`url` – an object of URL class.

Method Detail

parseUrl

```
private int parseUrl(string url)
```

parses the specified url into _protocol, _hostname, _port and _identifier

Parameters

`url` – a string that specifies url

Returns

0 if successful or 1 in failure

---

parseUrlHand

```
private int parseUrlHand(const char *url, char
*hostname, int *port, char *identifier)
```

Low leverl URL parser method. This method is called by int parseUrl(string)

Parameters

`url` – a const char pointer specifies url

`hostname` – a char pointer (string) specifies hostname

`port` – an integer specifies port number

`identifier` – a char pointer specifies only file name or directory with file name

Returns

Return ERROR or SUCCESS status if an error is occured return ERROR, else SUCCESS

---

getProtocol

```
public string getProtocol()
```

get current protocol

Returns

Return protocol

---

setProtocol

```
public void setProtocol(string protocol)
```

set current protocol with parameter

Parameters

`protocol` – a string specifies protocol

---

getHostname

```
public string getHostname()
```

get current hostname

Returns

Return hostname

---

setHostname

```
public void setHostname(string hostname)
```

set current hostname with parameter

Parameters

`hostname` – a string specifies hostname

---

getPort

```
public int getPort()
```

get current port

Returns

Return port

---

setPort

```
public void setPort(int port)
```

set current port with parameter

Parameters

`port` – an integer specifies port

---

getIdentifier

```
public string getIdentifier()
```

get current identifier

Returns

Return identifier

---

setIdentifier

```
public void setIdentifier(string Identifier)
```

set current identifier with parameter

Parameters

`identifier` – a string specifies identifier

---

toString

```
public string toString()
```

print part of url suitable form

Returns

`res` – a string specifies url

DOWNLOADLINKS

This file is not a class just include two functions for download operation. This

functions uses above classes functions and properities. Foundation download operation is occured these functions. These functions download links in web file on syncronization.

Method Detail

download_links

```
public int download_links(string url)
```

Manages download process on several links downloading syncronization

Parameters

`url` – a string specifies url that is downloaded

Returns

Return an integer value that specifies download status. Whether an error is occurred returned ERROR(0) value else SUCCESS(1)

---

download_links_hand

```
private void *download_links_hand(void* arg)
```

download handle. Entry point for threads

Parameters

`arg` – pointer to void specified arguman of thread function.

FILEDOWNLOADER

This file defines the function that are involved in a download process. This functions uses above classes functions and properties. Foundation download operation is occurred these functions. Thread creation and so is used here.

Method Detail

download_files

```
public int download_file(string url, int threadNum)
```

Manages download process.

Parameters

`url` – a string specifies url that is downloaded

`threadNum` – an integer specifies thread number

Returns

Return an integer value that specifies download status. Wheather an error is occured returned ERROR(0) value else SUCCESS(1)

---

download_file_hand

```
private void *download_file_hand(void* arg)
```

download handle. Entry point for threads

Parameters

`arg` – pointer to void specified arguman of thread function.

URLFILE

Public class **URLFile**

This file defines a URLFile class. URLFile class abstracts a remotely hoted file that has to be downloaded. Instantiated objects apears as an interface between the remote URL file and the local file in a computer's drive.

Field Detail

_name

```
private string _name
```

The string variable that is specified source file name with directory.

_url

```
private URL _url
```

An object of URL class is specified url,and used for some URL operations such as getHostname.

_cursorPos

```
private unsigned long _cursorPos
```

A unsigned long integer is specified cursor posation on file.

_size

```
private unsigned long _size
```

A unsigned long integer is specified size of file base byte.

_isOpen

```
private bool _isOpen
```

A boolean variable is specified file open status.

_diskFile

```
private ofstream _diskFile
```

It is a stream of file.

Constructor Detail

URLFile

Public

```
URLFile() : _name(), _url(), _cursorPos(0), _size(0),
_isOpen(false), _diskFile() {}
```

It is default constructor some variables are initializedwith default value.

Public

```
URLFile(URL url, string fileName) : _url(url),
_cursorPos(0), _name(fileName), _isOpen(false), _diskFile()
```

This constructor is getting url and file name from user. Size value is initialized with function that is finding file size.

Parameters

`url` – an object of URL class is specified url.

`fileName` – a string is specified file name.

Public

```
URLFile(URL url) : _url(url), _cursorPos(0),
_isOpen(false), _diskFile(){}
```

This constructor is getting an object of URL class from user at creation point. And size variable is initialized with function.

Method Detail

open

```
public bool open ()
```

This method creates the a file in the HDD and initializes its size to the size of the target URL file. Before calling write method file open method MUST be called.

Returns

true if file is successfully created and opened for writing.

---

write

```
public long write(ThreadRecord &rec)
```

This method writes the buffer of passed argument to te proper place within the file.

Parameters

`rec` – this argument contains cruical information about writing the information

Returns

Number of bytes written to the file

---

close

```
public bool close()
```

closes the file

Returns

true if file is closed successfully and false on failure

---

getSize

```
public unsigned long getSize()
```

Determines the size of the file to be downloaded, sending HEAD request through HTTPConnection.

Returns

The size of the file to be downloaded in bytes

---

isOpen

```
public bool isOpen()
```

check file open status

Returns

true if file is opened and false if file is not opened.

---

findFileSize

```
private unsigned long findFileSize()
```

This mehtod creates a HTTPConnection object and sends HEAD request to get the file size to be downloaded. If error occurs returned value is 0

Returns

The size of the file to be downloaded (in bytes). 0 if error occurs.

THREAD RECORD

Public class **ThreadRecord**

ThreadRecord includes information about specified thread. Every thread has own record to hold self data. In record, there are information about thread, these information are using in downloading operation.

This class provides user to reach all fields of record. User can set and get each field by using methods of this class.

Field Detail

_id

```
private int _id
```

Identification number of thread

_startByte

```
private long _startByte
```

The byte where thread start to download from specified URL.

_endByte

```
private long _endByte
```

The end byte where download will stop.

_downloadedBytes

```
private long _downloadedBytes
```

Amount of downloaded bytes

_fileCursorPos

```
private long _fileCursorPos
```

Cursor position in file where writing operation will start.

_status

```
private int _status
```

Download status information. If status is 0 thread is completed download, otherwise thread is downloading.

---

buffer

```
public char buffer[BUFFER_SIZE]
```

Thread read the data from specified URL and store the data here. Buffer is temporary place before writing to the file.

Constructor Detail

ThreadRecord

```
public ThreadRecord()
```

Default constructor

ThreadRecord

```
public ThreadRecord(int id, long startByte, long
_endByte)
```

Parameters:

`id` – thread id number.

`startByte` – the byte where thread start to download.

`endByte` – the end byte where thread stop downloading.

ThreadRecord

```
public ThreadRecord(const ThreadRecord &rec)
```

Copy constructor

Parameters:

`rec` – Address of the thread record which is copied.

Method Detail

getId

```
public int getId()
```

Returns id number of thread.

Returns:

Id number of thread

---

setId

```
public void setId(int id)
```

Change id number of thread with taken parameter.

Parameters:

`id` – new id number

---

getStartByte

```
public long getStartByte()
```

Returns start byte where thread start downloading.

Returns:

Start byte of downloading process.

---

setStartByte

```
public void setStartByte(long startByte)
```

Change start byte of the thread.

Parameters:

`startByte` – new start byte number.

---

getEndByte

```
public long getEndByte()
```

Returns end byte where thread stop downloading.

Returns:

End byte for downloading process.

---

setEndByte

```
public void setEndByte(long endByte)
```

Change end byte of the thread with endByte.

Parameters:

`startByte` – new end byte number.

---

getDownloadedBytes

`public long getDownloadedBytes()`

Returns amount of downloaded bytes.

Returns:

Amount of downloaded bytes.

---

setDownloadedBytes

`public void setDownloadedBytes(long downloadedBytes)`

Change amount of downloaded bytes with new value.

Parameters:

`downloadedBytes` – New value of downloaded bytes.

---

getFileCursorPos

`public long getFileCursorPos()`

Returns cursor position where downloaded data will be written.

Returns:

Cursor position in file.

---

setFileCursorPos

`public void setFileCursorPos(long fileCursorPos)`

Change cursor position in file.

Parameters:

`fileCursorPos` – New cursor position.

---

getStatus

```
public int getStatus()
```

Returns status of the downloading process. If it is 0 downloading process is finished, else downloading process is going on.

Returns:

Status of downloading process.

---

setSatus

```
public void setStatus(int status)
```

Change status of the downloading process.

Parameters:

`status` – New value of status.

THREAD TABLE

Public class **ThreadTable**

Thread table maintains information that is crucial to manage multiple threads during download process. Thread Table consist of rows of type threadRecord. In one downloading operation, all created threadRecords are adding to this table.

In this class user can insert new threadRecord object to table and can obtain pointer of any threadRecord.

Field Detail

_thRecord

```
private ThreadRecord *_thRecords
```

The array of thread records.

_capacity

```
private int _capacity
```

Thread table capacity.

_size

```
private int _size
```

Number of thread records in table.

Constructor Detail

ThreadTable

```
public ThreadTable():_thRecords(NULL), _capacity(0),
_size(0)
```

Default constructor.

ThreadTable

```
public ThreadTable(int capacity)
```

Paramaters:

`capacity` – the capacity of the thread table.

Method Detail

insert

```
public int insert(ThreadRecord rec)
```

This method inserts a new thread record in a table. It is assumed that caller does not insert a duplicate record. Function may return 1 as failure if there is not enough space to insert the record.

Parameters:

`rec` – A new thread record, will be inserted to the table.

Returns:

0 – if it inserts new item successful

1 – if fail has occurred.

getRecordHandle

```
public ThreadRecord *getRecordHandle(int threadId)
```

This method returns a pointer to a ThreadRecord object that match with the argument threadId. If such object does not exists returns NULL. Caller must be careful manipulating the returned result because all changes will be implicated in the object.

Parameters:

`threadId` – specifies id numbers of the thread records.

Returns:

It returns pointer of the threadRecord object whose id number is equal to threadId.

SOCKET

Public class **Socket**

This class creates Socket object which used to connect to remote hosts in a network.

This class create and establish socket connection with the specified hostname on the specified port number. On this socket connection caller can send requests and receive data.

Field Detail

sd

```
private int sd
```

This is socket descriptor.

---

servAddr

```
private sockaddr_in servAddr
```

It holds server address where downloading data will be taken.

---

localAddr

```
private sockaddr_in localAddr
```

It holds local address.

Constructor Detail

Socket

```
public Socket()
```

Default constructor

Method Detail

create

```
public bool create()
```

This function creates new socket object.

Returns:

`true` – if it is created successfully
`false` – if fail has occurred

---

bind

```
public bool bind(const int port)
```

This function binds the socket to the specified port number.

Parameters:

`port` – port number

Returns:

`true` – if binding has been performed successfully

`false` – if fail has occurred

---

close

```
public bool close()
```

This function closes the socket object.

Returns:

`true` – if socket has been closed successfully

`false` – if fail has occurred

---

connect

```
public bool connect(const string host, const int port)
```

This function establish socket connection with the specified hostname on the specified port number.

Parameters:

`host` – hostname

`port` – port number

Returns:

`true` – if connected successfully

`false` – if fail has occurred

---

send

```
public bool send(const string request) const
```

This function sends a request to another socket (server).

Parameters:

`request` – the request which is to be sent to the server.

Returns:

`true` – if message has been send successfully

`false` – if fail has occurred

---

receive

```
public int receive(char *buffer) const
```

This function is used to read from a socket to receive a response generated by the remote host. Received message is returned through buffer parameter passed as argument. This method returns the number of read characters. -1 if error occurs during data transmission, 0 if nothing has been received. Both -1 and 0 as returned values should be considered as end of stream and the socket should be closed if no other communication is pending.

Parameters:

`buffer` – it is temporary place to store received data.

Returns:

This function returns number of received data if it is successful.

`-1` – if fail has occurred

`0` – if nothing is accepted

---

isValid

```
public bool isValid() const
```

Tests the socket, if it is valid returns true, else it returns false.

Returns:

`true` – if socket is valid

`false` – if socket is invalid

HTTPCONNECTION

Public class **HTTPConnection**

HTTPConnection class provides establishing HTTP connection with a HTTP server. This class utilizes Socket class to connect to the HTTP server.

Through HTTP connection, this class sends head or get requests. There are three different request types, head request to obtain header data, sendGet request to obtain any data, and senGetRange request to obtain range of data. After send get requests, caller has to use receiveGet to obtain data.

Field Detail

_url

```
public URL _url
```

URL object, it uses for parsing url and reach its fields easily.

_id

```
public int _id
```

Connection id number

_socket

```
public Socket _socket
```

Socket object, it uses for receiving data from the server.

---

mutex

```
public pthread_mutex_t mutex
```

Mutual exclusion, it prevents race condition between threads while they are trying to access file.

Constructor Detail

HTTPConnection

```
public HTTPConnection():_url(), _id(), _socket()
```

Default constructor

HTTPConnection

```
public HTTPConnection(int id, URL url):_id(id),
_url(url),_socket()
```

Parameters:

`id` – HTTPConnection id number

`url` – URL object

Method Detail

sendGet

```
public bool sendGet()
```

This method sends a get request to download the target file from specified URL.

Returns:

`true` – if request is send successfully

`false` – if fail has occurred

---

sendGetRange

```
public bool sendGetRange(int startByte, int endByte)
```

This method sends a get range request. startByte specifies the beginning of download and end byte specifies the end byte of the range.

Returns:

`true` – if request is send successfully

`false` – if fail has occurred

---

head

```
public string head()
```

This method performs a head request and returns the header as a result. After calling this method caller should close the connection if (s)he does not need it any more.

Returns:

Header information related to the target URL.

---

receiveGet

```
public int receiveGet(ThreadRecord &threadRecord)
```

This method reads the response from the HTTP server. It should be called after calling sendGet or sendGetRange methods. Passed argument is a reference to the threadRecord associated with the thread calling this method. This method alters fields _downloadedBytes, buffer and sets status to 1 when buffer is full.

Parameters:

`threadRecord` – address of the threadRecord. This method alters _downloadedBytes, buffer, status fields in record.

Returns:

This method returns number of downloaded bytes.

---

close

```
public bool close()
```

This method close HTTPConnection.

Returns:

`true` – if closed successfully

`false` – if fail has occurred

---

createGetRequest

```
private string createGetRequest()
```

This method creates request for sendGet function.

Returns:

This method returns created get request string.

---

createGetRangeRequest

```
private string createGetRangeRequest(int startByte, int endByte)
```

This method creates request for sendGetRange function.

Parameters:

`strartByte` – start byte of the range where download will start

`endByte` – end byte of the range where download will be finished

Returns:

This method returns created get range request string.

---

createHeadRequest

```
private string createHeadRequest()
```

This method creates request for head function.

Returns:

This method returns created head request string.

DOWNLOADER

Public class **Downloader**

Downloader class handles multi threaded download of a remote file through a http protocol. Download class uses HTTPConnection class to connect to the the HTTP server and receive the file. URLFile class is used to handle the process of writing obtained data from HTTPConnection object to a physical file in a local drive.

An instantiated Downloader object may be used to download one url target at a time.

Field Detail

_threadNum

```
private int _threadNum
```

Number of threads

_threadTable

```
private ThreadTable _threadTable
```

Thread table object

_url

```
private URL _url
```

Url object, it holds specified URL.

_file

```
private URLFile _file
```

The URLFile that object handles file operations such as writing.

---

fileLocker

```
private pthread_mutex_t fileLocker
```

Mutex used to avoid moultiple access to the shared file

_fileName

```
private string _fileName
```

File name, where downloaded will be written.

Constructor Detail

Downloader

```
public Downloader()
```

Default constructor

Downloader

```
public Downloader(string url, int threadNum)
```

Parameters:

`url` – spacified url
`threadNum` – number of threads

Downloader

```
public Downloader(string url, int threadNum, string
fileName)
```

Parameters:

`url` – spacified url

`threadNum` – number of threads

`fileName` – file name where downloaded data will be written.

Method Detail

downloadThreadId

```
public bool downloadThreadId(int threadId)
```

This method downloads data according to threadId number from specified URL.

Returns:

`true` – if data downloaded successfully

`false` – if fail has occurred

---

saveFile

```
public bool saveFile()
```

This method close file with using URLFile object's close function.

Retuns:

`true` – if file is closed successfully

`false` – if fail has occurred

DOWNLOADFUNCTIONATTRIBUTE

Public struct **DowloaderFunctionAttribute**

DownloadFunctionAttribute is a struct which consist of thread id and pointer of downloader object . An object of this class is created before creating a download thread and passed to the trhead's start function as argument.

Field Detail

threadId

```
public int threadId
```

Id number of the thread.

---

downloader

```
public Downloader *downloader
```

A pointer of the downloader object.

Constructor Detail

DownloadFunctionAttribute

```
public DownloadFunctionAttribute()
```

Default constructor

DownloadFunctionAttribute

```
public DownloadFunctionAttribute(Downloader
*downloaderPth, int thId)
```

Parameters:

`downloaderPth` – A pointer of Downloader object
`thId` – id number of the thread record