COMPUTER NETWORKS

PROGRAMMING ASSIGNMENT 2

1 Problem description

Write a program (name it `client-udp`) that sends UDP packets, accordingly to protocol described below, to predefined server in order to download a file.

The program has to work from Linux command line and accept three arguments: *port-number*, *local-file-name* and *file-length* in bytes. The goal of your application is to download first *file-length* bytes of a file served by provided server listening on host `aisd.ii.uni.wroc.pl` and port *port-number*. Downloaded bytes have to be written out to *local-file-name* file.

In order to download a fragment of the file you have to send a message to the server consisting of following string:

```
GET start length\n
```

Value of *start* has to be an integer from range $[0, 1000000]$ and *length* from range $[0, 1000]$. String `\n` denotes new line character as in Unix-like systems. Words in the message are separated by single space character. Every packet sent to the server has to adhere to this specification, otherwise it will be ignored.

As a reply the server will send a message that looks as follows:

```
DATA start length\n
```

Values of *start* and *length* are same as in the request. The header is then followed by *length* bytes, that represent contents of the file starting from position *start* to the position at $start + length - 1$.

Server awaits for UDP packets at port range from 40001 to 40010. Please use mentioned ports to test your code. The program has to handle invalid data and show diagnostic message if something fails. The program is disallowed to print out unnecessary diagnostic messages, unless they show a progress in downloading consecutive parts of the file.

1.1 Examples

Your application is called as follows:

```
$> ./client-udp 40001 output 1000
```

It will communicate with `aisd.ii.uni.wroc.pl` on port 40001 and may choose to send two datagrams:

```
GET 0 600\n
```

and:

```
GET 600 400\n
```

After the data is sent back to the client, it will write the bytes to file named *output*.

1.2 Unreliable communication

Remember that `UDP` packets on their way to destination can be lost, duplicated or reordered. The server running at `aisd.ii.uni.wroc.pl` will help you exercise all aforementioned cases by employing following rules:

1. Reply for given request will be sent back with probability of about $1/2$.
2. Each reply sent back will be duplicated with probability of about $1/5$. Note that a duplicate is treated as a message and can be duplicated too! So it's possible to receive a duplicate of duplicate with probability of about $1/25$, and so on...
3. Each datagram is sent back with random delay of from 0 to 2 seconds.
4. The server maintains a queue of at most 100 datagrams ready to be sent back.

Additionally you should consider extra factors like unreliability of the Internet and server's queue overflows. However, effects observed as a result of these events are negligible.

1.3 Extra remarks

It's necessary to verify that source address and port are valid – so that we know the packet originates from the server and not some other address. You can assume, that packet's contents is not corrupted and adheres to the specification (you don't have to enforce it[1]), as long as the sender of the reply matches with the recipient of the request.

Remember, your program will be tested for correctness and performance.

In the simplest variant your program would:

1. send a request for file's fragment (maximum length of 1000 bytes),
2. wait for a reply no longer that 2 seconds,
3. repeat everything from step 1 if the packet hasn't been received,
4. write out the packet's contents to disk and move to next fragment.

Use `select()` or `poll()` system calls to wait for datagram on given socket.

Behavior described above matches what `client-udp-slower` does (statically linked Linux binary for 32-bit and 64-bit systems can be downloaded from Marcin Bieńkowski's web page). For correct implementation of such method, a student may obtain maximum of 6 points.

Aforementioned approach is very time inefficient. To improve it you can maintain fixed sized set of requests that had already been sent, but replies for them haven't arrived yet. Timeout is then bound to each request separately, and if it expires, the program sends the request again. In the event of

[1] In real-life applications such approach would be extremely naive and plainly stupid.

reply reception, removal of matching request from the set is triggered. Eventually new request is put into the set, to ensure it's kept full at all time if possible. This approach was implemented in client-udp-faster-32 and client-udp-faster-64. For such implementation maximum number of points will be granted.

You may choose to implement one of the variants described above, but of course you're free to try to implement your own approach. Your program will be compared with `client-udp-faster`.

2 Implementation

Files The program should be sent to your teacher as a single compressed archive (preferably in *.tar.gz* format). It must contain a single directory and within it:

- Source code in C or C++, i.e. `*.c` and `*.h` files or `*.cpp` and `*.h` files. Each of `*.c` and `*.cpp` source files must contain a comment (at the beginning of file) with following information: first and last name, an "indeks" number.
- `Makefile` file, that enables the program to be compiled through running `make` command.
- Optionally, `README` file containing any remarks from a student.

The directory **must not** any other files that mentioned above, and especially compiled program (in binary form) and linkable objects `*.o` or any other files not required to build the program.

Compilation The program will be compiled and run in Linux 64-bit environment with fairly recent `GCC` compiler.

In case of C language compilation, your program is allowed to use ISO C99 standard, possibly with GNU extensions (compiler option `-std=c99` or `-std=gnu99`).

Compilation will be performed with `-Wall` and `-W` options. The compiler must not report any warning with regards to your code.

Marcin Bieńkowski
Translation: *Krystian Bacławski*