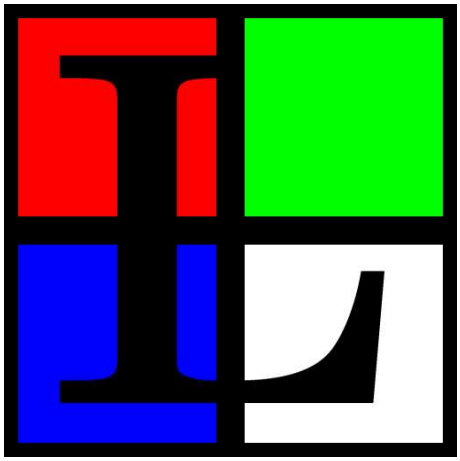
Charles University in Prague
Faculty of Mathematics and Physics
2002

SOFTWARE PROJECT

Links

The Web Browser



Project documentation

Authors: Mikuláš Patočka
Martin Pergel
Petr Kulhavý
Karel Kulhavý

Supervisor: Mgr. David Bednárek

Contents

1. Introduction

This is a project documentation for the program **Links**. You can read here about development course, history, participants and the result of the project. The printed version of documentation doesn't contain project mailing correspondence. The full documentation can be found in a PDF format on a CD with the program.

Links is a web browser for operating systems Unix and OS/2. The browser can be run both in text and graphics mode. In the graphics mode the browser can run under systems X-Windows, SVGAlib, AtheOS and Pmshell. The browser supports javascript (version 1.1 by Netscape Corporation), protocols HTTP 1.0 and 1.1, FTP, Finger, and SSL, formats HTML version 4.0 without CSS and image formats GIF, XBM, TIFF, JPEG, and PNG.

2. The order

The objective of the project is a web browser written in C language under operating system Linux, The basic engine (FTP, HTTP requeste, HTML formatter etc.) will be written by the project participants. The code written by other people may be used in case of libraries for processing of other formats (jpeg, png, mpeg, giff, ogg etc.). No libraries for work with HTTP and FTP will be used.

The broser should be stable, it's output high-quality if possible (easily readable). The comfort for the user should be so high that it could be used for everyday, fast and simple browsing on the Internet. The greates accent will be put on stability, because a crash of the browser means great nuissance for the user.

Middle-sized accent will be put on output quality (from the image quality point of view and from the document formatting point of view), because browsers are nowadays used for relatively large percentage of time spent sitting by computer and eye comfort is a requirement for successful and comfortable work with a computer. Aside from the output quality a speed limited only by typical transmission infrastructure will stand. The speed should not be limited by speed of CPU and graphics subsystem. If a conflict between quality and speed occurs, quality will be preferred. However, should this bring unjustified slowing down, then speed will be preferred.

The last position will occupy user comfort, which is going to be minimal, however elaborate and especially such that it wouldn't slow down the work with the browser and wouldn't require lengthy study of user interface documentation. We count also portability amongsth the comfort. Portability such that porting of the code doesn't bring significant problems, because work transparency in multiplatform environments brings the user high comfort without impinging upon other functions of the browser.

Expected browser features:

- Portability: OS/2, Unix
- Tables
- Frames
- Pull-down menus
- Translation of the menus to several languages
- Bookmarks
- Background file download
- Keepalive connections
- Asynchronous DNS lookup

- Possiblity of running external programs on unknown data types
- Possibility of switching between a text browser running on console and graphical browser running under SVGAlib, OS/2, W-Window, AtheOS
- Image display when running under graphics mode
- HTML version 4.0 without CSS
- No crash in case of invalid HTML
- Javascript
- Without warranty: cookies (switchable off)
- Caching (expriation, refresh) without warranty

3. Why did we select this project

There already are many web page browsers, however none of them was according to our expectations regarding stability, security, output quality and also speed. Moreover majority of them have over-complicated user control, which makes the work of the user unpleasant.

Concurrency browsers aim more at using so called progressive technologies and supporting the latest features than at stability, portability and speed. Regarding Javascript interpretation, concurrency browsers again aim at supporting latest features of the languages (best possibly such that don't work with other browsers) and are very unstable. It could be said that javascript rules the user, instead of user ruling the javascript. For every concurrency browser (Netscape, Mozilla, Opera, Konqueror, Internet Explorer) a javascript code few lines long exists that makes the browser unmanageable and it's necessary to restart it, or even worse, they pwerform a DoS-type attack on the user's behalf (either by huge memory requirements, or using up all CPU time).

Thereore we decided to write out own web browser. The goal of the project was writing quality, fast, secure and stable, however still simple, web page browser. Stress was put especially on stability, output quality and speed. It should crash in no way, prevent the user from control over itself, or making unjustified demands on the memory or CPU time. The browser should be portable to majority of Unices and compilable with minimal requirements on the system.

4. Project participants

The developper team, under supervision of Mgr. David Bednárek, consisted of the following members:

- Mikuláš Patočka
- Martin Pergel
- Petr Kulhavý
- Karel Kulhavý

Originally Jakub Drnec should have been in the team, but he left the faculty, so that only four people were left. The first year was under supervision of Petr Merta, but he left after one year and the project has been handed over to Mgr. Bednárek.

Also other people participated on the project, especially those who translated texts into various foreign languages. These people are listed at the end of the Credits chapter.

5. Program Structure

In this chapter you can read about rough division of the project into parts and who wrote which part. More detailed description of structure of the project is in the development documentation.

The core element of the browser is **scheduler** a.k.a. **select loop**. All actions that are due to happen are being planned here and individual parts of the browser are being called from here. The select loop is a cooperative scheduler. All parts of the browser are written to ensure that control doesn't stay for too long in them, because other parts could not work for a long time. This would manifest for example by not reacting to user's input.

Another very important part is **session** that ensures download management, document formatting and display, user control and movement around the document, and javascript execution. Session itself doesn't do anything, only calls other parts which do the actions themselves. Session calls object requester, javascript, HTML parse and the part providing display and interaction with the user.

Object requester manages document download from the network. Object requester calls **request scheduler**. File cache caching all files downloaded from network is connected to request scheduler. Interfaces of individual protocols (HTTP, HTTPS, FTP, finger and local disk download) are connected to request scheduler too.

HTML parser processes HTML documents and translates them into internal structures of the browser. **View** a.k.a. **displayer** manages document displaying, line breaking and handling of events generated by user. Aside from movement around the document the user can use menus. Part handling menus is called **menu**. The displayer calles functions of individual display drivers: terminal, X, SVGAlib, ... It is necessary to typeset in graphics mode and this is handled by part called **fonty**, which contains font cache and graphics routines for type processing and printing.

Javascript consists of lexical analyser, syntax analyser, interpreter and builtin functions. There is an interface between javascript and session called **javascript interface**, which negotiates access of javascript to internal data structures of the browser (document, pictures, links, forms, ...). Javascript interface contains functions that start and end interpretation of javascript code and similar functions. **event handlers** relate to javascript. Event handlers are pieces of code that are being called by various events, especially caused by user. For example by clicking on a button, moving a mouse, loading a document, pressing a key, changing a text field etc. Event handlers are built directly into the displayer.

Partitioning of work:

- **Mikuláš Patočka:** session, scheduler, menu, object requester (including implementation of individual protocols), HTML parser, text typesetter (both graphics and textmode), menu, OS/2 port, AtheOS
- **Martin Pergel:** parser and javascript interpreter
- **Petr Kulhavý:** X-Window system graphics driver, javascript interface (upcalls, even handlers), bookmarks, Czech translation, xbm and tiff image decoders, font making, documentation, testing
- **Karel Kulhavý:** SVGAlib graphics driver, images (displayer, cache, jpg, gif and png decoders, alpha channel handling), graphics processing (dithering, resampling, resizing and gamma correction), fonts (font selection, helper programs for font preparation, font making), Czech translation, logo

6. History

- **Winter 1998/1999:** Mikuláš started to write a HTTP requester. začal se psát HTTP requester. **Important decision:** to write an own scheduler insted of thread or multiple processes
- **Jaro 1999:** HTTP requester works
- **Start of summer 1999:** A simple page formatting and couple of HTML tags work. It is not possible to test the browser functionality on the Internet, because Mikulas's network card broke .
- **Léto 1999:** Mikulas doesn't have an Internet access over holidays, he tests the browser using a content cached in personal HTTP cache. HTTP requester debugged, tables written.
- **Autumn 1999:** Table algorithm changed to respect „width" entry. This algorithm is used in Links with couple of bugfixes in Links until today in both graphics and text version. Many things acceded like download, text search, associations and more.
- **24.11.1999:** Version 0.80 released.
- **25.11.1999:** Verze 0.80 announced on www.freshmeat.net, many people are finding bugs and therefore quickly a next version follows (see Changelog).
- **27.11.1999:** Dividing into two versions: **stable**, čnumbered 0.8x, in which only bugfixed are being done, and **experimental**, marked as „links-current", nto which more features are being added and which contains relatively many bugs.
- **11.03.2000:** Version 0.84 released - the last from the 0.8x series
- **15.06.2000:** Links-current is tested-through enough and bugs are removed from it, preparation for release. It becomes 0.90pre1.
- **28.06.2000: Links** 0.90 released.
- **28.06.2000: Links** 0.91 released, because 0.90 contained a bug in frames made in the last while (when a link has been activated, then always a file over the whole screen and not in a frame was displayed).
- **23.07.2000:** Division into two versions: **0.9x**, where again only bugfixes are being made, and a **links-current** with graphics and experimental functions.

Development of „links-current" follows:

- **Summer 2000:** OS/2 graphics driver, portig of user interface into graphics.
- **October 2000:** Announcing a project under leadership of Petr Merta.
- **Autumn 2000:** Graphics driver interface designed, Czech translation written.
- **Autumn, winter 2000/2001:** Getting the JS grammar, lexical, syntax, and semantic analysis, intercode generator. Page displayer in graphics, writing graphics driver analýza, generátor mezikódu. Zobrazovač stránek v grafice, writing X graphics driver.
- **March 2001:** Official project meeting, demonstrating what has been produced to the leader, assessment of our work by the leader
- **Spring and summer 2001:** Writing Javascript interpreter without upcalls, writing general lists and bookmarks.
- **Beginning of summer 2001:** Designed javascript interface.
- **October 2001:** Change of the leadership from Mgr. Merta to Mgr. Bednarek. Desing of implementation of images.

- **October 5, 2001:** Moved on from system of mutual sending of patches to CVS.
- **October 18, 2001:** Publishing a yearly report.
- **November 2002:** Started implementing upcalls.
- **Autumn 2001, winter 2001/2002:** Writing upcalls, tons of tiny things for better usability.
- **January 11, 2002:** Official meeting with the project leader, where state, up-to-date results was assessed. It was figured out who will program what, and project submission deadline was agreed on.
- **Winter–spring 2002:** Massive debugging and testing of javascript, customizing javascript to reality (badly written pages). Tables in graphics.
- **Spring 2002:** Writing documentation, lots of tiny things.
- **Leden 2002:** Font switching
- **Únor 2002:** Dithering algorithm rewritten, image manipulation rewritten, PNG image support written, started writing event handler in javascript.
- **Březen 2002:** GIF and JPG decoder adder. Completely finished writing upcalls.
- **Duben 2002:** XBM and TIFF decoder written. Started writing a driver for framebuffer.
- **March 24, 2002:** Meeting with the leader and showing project before coming plea. We decided to stop the development and only test and remove bugs found.
- **May 10, 2002:** Version 0.97 released.
- **May 16, 2002:** First official release of pre-version of the browser.
- **May 20, 2002:** Version 2.0pre1 released, Slovak translation updated and couple of bugs repaired
- **May 23, 2002:** Version 2.0pre2 released, a bug with referer fixed, `frame.top` bug in javascript fixed, increased portability (libpng 1.2.2).
- **May 23, 2002:** Version 2.0pre3 released with several fixed bugs.
- **May 26, 2002:** Version 2.0pre4 released, couple of segfaults fixed, bug in TIFF and 16-bit PNG images on big endians fixed, incompatibilities with MIPSpro Compilers fxed and couple of other small things.
- **May 28, 2002:** Version 2.0pre5 released with a fixed segfault on 16-bit PNG images
- **May 30, 2002:** Version 2.0pre6 released with updated Hungarian translation.

Project meeting were occuring very often, usually in queue in school dining room, during lunch, in computer laboratory, during 0verkill sessions and every time we saw each other we were discussing project problems, new things, what it is necessary to write and more. We also discussed many things using talk or e-mail. The project had three official meetings with the leader. First by writing out the project, second after half a ear and third again after another about half year. We wrote the project by ourselves, only here and there we needed to consult with the leader. We regularly wrote reports about project status.

7. Problems and decisions

In this chapter we are going to discuss problems we hit during the development of the project and important decisions that we were facing, including solutions.

7.1 Scheduler

We decided to write our own scheduler instead of making multithreaded or multiprocess web browser. Mikuláš was once trying to write a multithreaded variant for about two day, but he didn't like it so he deleted it and wrote his own scheduler. Having an own scheduler is more effective and as it is cooperative, then the individual parts can more easily switch between themselves and don't have to do complicated waiting.

7.2 Graphics interfaces

During writing a driver for X-Window graphics system we had a choice from an amount of already existing toolkits, however, we didn't exploit this possibility and wrote our own driver using directly Xlib library. The reason was that toolkits are many and every user is using a different one. Therefore the user would be potentially forced to install more libraries on his system, which would make the installation more complicated. Moreover, when there are so many similar toolkits then there isn't a reason for using some particular one. Using a toolkit would benefit in just a similarity to other applications and making the programming easier. But we strived for maximum similarity to text mode usage, which was not achievable with any toolkit.

We designed the graphics interface on a low level to be portable, simple and that a new driver could be easily added. The interface contains elementary functions for printing a bitmap, painting an area with colour, drawing a line, scroll and so on. These functions can be written very easily on any graphics interface.

Thanks to this design of graphics interface, the browser is using only one window. This has an advantage of easy predictability where which window will be placed, so it can't get lost on neighbouring screen neither it can be overlaid with different window. This makes the user comfort higher.

7.2.1 SVGAlib

Linksmay not work on older versions of SVGAlib, because they contain various bugs like a bad code that draws nonsenses over the screen during performing graphical operations. Using **Links**on older SVGAlibs can pose a security risk, because for example SVGAlib 1.2.10 was incorrectly giving up root privileges.

This is not a mistake of **Links**, but of the SVGAlib designers. I (Karel Kulhavý) have an opinion that SVGAlib is broken by design and should be rewritten, which is however not within the scope of **Links**project.

Even the new SVGAlib has bugs in the function `vga_setlinearaddressing` (which manifest for example on S3 Virge 3d). Due to these bugs this function had to be commented out in **Links**(the same is done for example in the program Quake). Due to this reason **Links**is slower in some circumstances, which is not a mistake of **Links**, but of SVGAlib.

Some graphics functions were so buggy in SVGAlib that they didn't work at all and had to be emulated using more primitive functions, for example filling a rectangular area using `putpixel` (2-color modes). Because even this interface was drawing nonsenses, and regarding the fact that 2-color modes are not capable of displaying of any colours, they were removed from the **Links**at last.

Linksis using acceleration primitives of SVGAlib in a case when SVGAlib supports them. But the problem is that SVGAlib doesn't support them for all cards, at least not for the one the **Links**team had an access to. Due to overall conception of SVGAlib it can be expected that they will be massively buggy and that **Links**will on some cards heavily crash the system. This is not a problem of **Links**, but of SVGAlib.

7.2.2 Framebuffer

Writing the framebuffer driver on Linux was very uneasy due to very bad or completely missing documentation. Therefore a 100 percent reliability is not guaranteed. Also it is not guaranteed that the driver will work with all cards on all computers. We were often forced to replace documentation by reverse engineering the source codes of the linux kernel, reading graphics drivers for the indivudual graphics cards and reading very sparse comments in header files of the framebuffer interface. These conditions made our work very hard and unnecessarily extended the development time. Often we have to determine things directly by the authors of the linux kernel or empirically try the behaviour and judge according that how the interface should be used. We consider this way very unclean, however due to absence of quality documentation for the interface we didn't have another option.

We had to be determining mouse position during writing framebuffer. We used the `gpm` library for this, using which the state of mouse is being assessed on a terminal (in text mode). Framebuffer is a de-facto terminal with a possibility of graphics output. Framebuffer doesn't offer any own interface for the mouse, only allows the graphics output to the screen. There isn't any other option of reading the mouse on a terminal, because not on all computers an ordinary user has an access to the mouse device (for example `/dev/mouse`). The `gpm` library reads the mouse device and allows user applications to determine the mouse position and state of button depression using a socket. Fortunately majority of Linux computers have `gpm` installed and therefore it can be used.

But the problem is that `gpm` is strictly textual and therefore the cursor moves by characters and not by pixels. We tried to read the mouse by characters and by a dirty trick fake larger size of the screen, but it didn't work with older versions of `gpm`. Therefore we made a reading of relative mouse movements, which works with all versions. But the problem is that the mouse is moving very slowly. Therefore we had to finally anyway multiply the relative movement by a constant and therefore it is not possible to move the mouse smoothly and a real danger exists that there will be a spot on the screen where the user will not be able to click however necessary it will be. This problem can be partially solved by enlarging the text and pictures, but it unfortunatelly can't be solved completely. We tried to alleviate this situation by providing a possibility to make a fine movement of the mouse using F5 – F8 keys.

The client application unfortunately can't set the mouse sensitivity, because it is set by the administrator hard during staring up the `gpm` daemon. Therefore we chose a compromise between sensitivity and accuracy. Therefore the cursor moves around the screen acceptably.

Theoretically it would be possible to write graphics support into `gpm`, because `gpm` is being spread under GPL licence. Writing graphics support should not be a big problem, there is even a certain chance that if the modified library would be sent to the authors, the authors would publish it in the next version. But the problem is that not everyone would have this library installed on his computer, therefore again the user without administrator privileges couldn't use a mouse on framebuffer again. This would be rather a long-term solution, because it could be counted on that the graphics-capable version of `gpm` would become common.

For the users that have administrator privileges we made a patch by application of which the mouse starts moving smoothly over the screen. The patch is distributed together with the browser and is in the file `PATCH-gpm-1.20.0-smooth-cursor`. The patch was made for version 1.20.0 of`gpm`, but it will probably work with different version as well, possibly after a small change.

We managed only to implement framebuffer on Intel platform. We had an acess also to computers of Sparc architecture, on which Linux with framebuffer was running, but these computers don't have linear mapping of video memory, which complicates

the implementation. As we didn't have a documentation, we didn't make support for nonlinear memory mapping. nelineární mapování paměti jsme nepodpořili.

7.3 Javascript

We decided to write the interpreter from scratch, because we came to an opinion that a javascript interpreter in **Links**may be useful. Moreover Mgr. Bednarek said on compilers lecture that writing a compiler or interpreter from scratch happens to few people and we consider design and making of an interpreter an abnormally interesting business, though a bit demanding.

7.3.1 Grammar

The first problem we encountered during writing javascript was getting a norm according to which we could write. We tried hard for about a month to find it on the Internet, but we couldn't find anything. We managed to get ECMA 262 norm descibing ECMA script grammar, but at that time we didn't have a slightest notion that Javascript 1.3 norm exists. We also didn't have a slightest notion that is is governed by ECMA script grammar. Then someone managed to find Javascript 1.1 norm by Netscape (not on Netscape webpages), according to which we proceeded. We got known about Javascript norm about year after finding the 1.1 norma and after starting it's implementation. Javascript 1.1 had grammar simpler than ECMA 262 grammar – it didn't contain two-reducction conflicts and almost no shift-reduce conflicts. In the ECMA grammar there were tens to hundreds of two-reduction conflicts and shift-reduction conflicts.

We too document-object model from Netscape 2.0 according to a found documentation. We added some enhancements.

7.3.2 Parser

During designing the javascript parser we had a possibility to write a lexical and syntax analyzer either our own, or use already existing tools Bison and Flex for manufacture of syntax and lexical analyzers.

In the case of syntax analyzer a machine manufacture won without doubts, because the grammar is too complicated for manual writing. The grammar contains 131 rules and the automat from Bison has 212 states. The resulting syntax analyzer is therefore LALR(1). There are about 40 shift-reduce conflicts in the grammar which are resolved by shift operation precedence. We did a machine rewrite into bison-acceptable form of the grammar after download. We then manually wrote reduction actions.

We used a tool called Flex for lexical analysis, because manual writing of an automaton is lenghty and usually the resulting automaton contains lots of bugs. For using Flex program it was necessary to redefine functions `getc` and `ungetc`, because we didn't manae to determine, how to force Flex read the input from memory and not from a stream.

A problem is related to decision to use Flex and Bison: what if the user will not have these tools installed on his computer? These tools are not as common as for example C compiler on Unix system. Therefore we decided to supply machine-generated outputs from these tools into source distribution. The original Flex and Bison sources are of course also provided in the source distribution, therefore it's always possible to regenerate the automats at any time.

7.3.3 Intercode

According to an advice from Mgr. Bednarek, the javascript interpreter was divided up into parser with intercode generator and intercode interpreter, because without the division, multitasking implementation would be difficult in the automata. Therefore it would be necessary to wait for until the script ends and only then continue.

We chose a tree form of the intercode, now a real DAG, without doing any optimizations. Tree intercode was chosen because a tree is directly outputted by the syntax analysis. If we had to generate quadruple or triple intercode, it's generation would take some time, which would delay interpretation of short scripts, which we wanted to be able to interpret quickly. After consultation with Jan Hubicka (of GCC fame), who said that tree intercode is better to be interpreted by a stack, we decided to write a stack-based interpreter.

The intercode tree contains in every node an information. This information contains line number where the particular piece of code occurs, operator and space for 6 arguments. The arguments could be placed into for example linked list, in an array of variable size or in a different data structure. As operator can never have a variable number of arguments, we chose an array of fixed size which has advantage of constant access time.

Names are translated into keys, list of which is hashed. Identifier have keys within one context unique. This solution has been chosen for ensuring higher interpreter speed. The interface is designed for change of „address spaces“ size to be possible, currently it is size of 128 addresses which is a compromise between size and number of collisions. According to theroretical computations, with quadratic size of mean number of collisions is constant. I. e. first 11 records should collide with a probability of $P \leq 50\%$. On the other hand a collision in every slot of hash in expected case occurs only after $128 \log 128$ records, so that only after defining 896 variables the table „overflows“. Moreover in linked lists leading from each hash table slot MFR is applied, which shows results at most $2\times$ worse than optimal self-maintaining list (see RNDr. Václav Koubek: Datové struktury).

Identification by keys means remarkable speed-up, but sometimes it is harmful — for example when an object defined in document is to be searched for. Identification by keys is in many cases advantageous , because interpretation consists mainly of identifier searches and with setting according to the norm at least first search is performed always internally in javascript, second search by at least second access too (for example Tdocument.object.. . .).

7.3.4 Security

Javascript interface is designed so that it doesn't allow javascript to access foreign objects. Foreighn objects are all objects in documents from different servers, than the server with the document with the accessing javascript is. All upcalls, which work which work with some objects strictly test these access rights, therefore javascript doesn't see „foreign“ objects at all and has a feeling that they don't exist at all.

We were warned that many things can overflow in an interpreter. After that we decided to write a „memory accounting“. The user has a possibility to set tp maximum memory amount that javascript is allowed to allocated. After exceeding this limit at the end of an elementary javascript step is the script (context) killed and a „purge“ performed on it. This purge is done for the remaining contexts to be able to continue. The accounting can be performed either for each context separately, or for all of them together. If we accounted each context separately, there would be a danger that the script would open too many contexts by which the interpreter would exhaust the memory. Therefore we chose memory accounting for all contexts.

Overflow would be possible to be detected directly in the allocation function, but a problem would arise: how to end up the script from the allocation function. Therefore memory is allocated in allocation function always and overflow check is performed only at the end of function `zvykni` (zvykni is Czech word for chew). As this function performs only elementary interpreation steps, there is not a danger of allocation of too big a chunk of memory in one step. Therefore this method is safe. According to our compuations the memory should not grow in elemantary step more than in a linear way.

With memory complexity of javascript is also connected a potential danger of stack

overflow by running for example an infinite recursion. To prevent this danger we introduced clipping of maximum recursion depth – the user has again a possibility to set this up in a menu.

Due to possibility of interpreter starvation by running an infinite loop it was decided that the interpreter **must** return control after finite time and reschedule itself. Scheduling is therefore performed after intercode generation and then after hundred times calling of function `zvykni` (100 is an empirical constant that shows relatively sufficient ratio of price to complexity), moreover the interpretation is being interrupted also by certain upcalls. Only one thread may run within one context. Running in more contexts at a time is however not limited and even occurs very often. Number of steps until „preemption“ occurs can be changed by a compile-time constant. It would be possible to leave te user to change this constant during runtime, but we were fearing that some users would not comprehend the meaning of this setting and would only try what numbers can be entered. The interpreter would then interpret slowly, or would have too big a latency after a keypress.

After experience with other browsers we came to a conclusion that these security measures do not suffice. The attack can be also accomplished for example by printing warning in an infinite loop, permanent changing of the URL and so on. It is this type of attack which no existing browser can defeat. For example the following oneliner code is enough:

```
while(1)alert("You are a BFU!");
```

This is not a typical DoS attack which would exhaust all memory or 100 % CPU time, therefore the authors of other browsers probably didn't think about that. In a case of attack like this, the user is forced to infinitely click on javascript windows due to which he never gets to other control elements of the browser. Therefore this attack is not mounted against computer, but against user. Some browsers (for example Netscape Navigator) even don't wait for user's reaction and are making still more and more windows, which understandably prevents the user from even using the other applications.

Therefore we decided for a solution that is simple, but efficient: to allow the user to kill the script in every window opened by javascript. In case of URL change, opening and closing browser window the browser will ask the user if he allows this action. The user has a possibility to permit, refuse or end the script. This prevents javascript from pestering the user by unsolicited URL changes, closing browser window or flooding with more and more new windows.

7.3.5 Errors and warnings

The javascript standard says what is correct and what is incorrect very ambiguosly. Therefore if it's possible, we ignore errors during interpretation, moreover we gave a freedom to the user to set up level of tolerance to errors. We ignore some errors straight away and only notice the user about them by a warning. We allow **all** type conversions automatically, as well as turning off warning messages, turning off the whole javascript or immediate stopping of **all** currently running interpretations, not mentioning the possibility to kill javascript during **every** window message (if it still makes sense).

Error means definitive end of interpretation in the given context, i. e. all following scripts in given contexts are already ignored, therefore on pages based on javascript and written badly the user had no other option to peek into page source, understand what the script was intended to do, interpret it himself and perform the possible required actions by hand (for example URL change). Killing all scripts causes error in all existing contexxt, therefore on pages containing only links through javascript it is absolutely unsuitable to ask for killing out all scripts.

The decision not to continue in interpretation after occcurence of error is supported

by the fact that after a syntax error it would be necessary to make some functions invalid and remove part of the tree. Following interpretations typically reference the previous results, so that it would ultimately lead to an error anyway. To continue after a semantic error it would be necessary to remove problems with stacks of parents and arguments and to do magic with address spaces. It is enough that similar magic is necessary for operations `break`, `return`, and `continue`. Errors „chained“ or better „introduced“ by previous would happen. Not continuing after an error is usual also in other browsers.

Because javascript authors often do not mention object document when accessing it's members, we decided to allow name resolution in the main address space, but only if the user asks for this. The user has a possibility to tick up global name resolution in a menu. This setting is on by default. We gave this choice to the user because global name resolution is slower than local one and also because some web pages require it.

7.4 Images

In the first implementation of images the displayer of images (decoder and dithering engine) was unrestartable, which showed up as a large insufficiency, because display of images was slow and was blocking other actions of the browser. Therefore we had to rewrite the displayer into a restartable version so that it is possible to reschedule in middle of decoding and dithering.

7.4.1 TIFF

We decided to support TIFF graphics format. This format is not common on the web, but sometimes there are important documents that are not available in a different format. We used `libtiff` library for decoding, because due to many variants and diversity of the format hand writing the decoder would be too time consuming. The TIFF format specifies that decoding may start only after loading the whole file. Therefore TIFFs won'y display during downloading from the network. This is however not a bug of **Links**, but a feature of the TIFF format.

Other browsers do not support TIFF format, they need a plugin to display it. This is however only a pity, because TIFF is not an unused format.

7.4.2 JPEG 2000

When we were writing decoders for various graphics formats, we also wanted to write a support for the new stadard JPEG 2000, which was interesting for us for its high compression. We had an opinion that this format will surely be used in the future, therefore it's support would be a good investment into the future. Unfortunately we didn't find any open source library supporting this format. Hand writing of wavelet decoder would be time consuming job, therefore we dismissed support of this format. Nevertheless as soon as a platform-independent library supporting JPEG 2000 is available, it will not be a problem to add this format into the browser thanks to flexible design of the image decoding interface.

7.5 Others

We decided to place all data structures and executable code into one binary file for portability reason. By this the user is freed from problems where to place the browser data and a problem with finding these data is solved also (different users typically want to place their data into different directories, therefore after copying the binary file **Links** could not be run. This way just copying the binary file onto a different computer with the same platform is sufficient and the browser will run there.

C was chosen as programming language because C is a universal programming language characterized by efficient expressions, modern flow control, stucture of the data

and wealth of operators. Generality of the C language makes it more suitable and more efficient for many tasks than other „more powerful“ languages. We were respecting the ANSI C standard to reach portability to as many systems as possible.

7.5.1 Browsing History

In the history the whole formatted page including cursor position and contents of all forms is stored. Therefore if the user goes back in history, he will get to exactly the same place from which he went to the new page. The form content will not get lost even not by page reload, as it is with other browsers.

7.5.2 Portability

For the project to be good we had to ensure code portability. Therefore we wrote in ANSI C code. Nevertheless at least some parts had to be platform dependent. In spite of this fact we had to ensure portability. For this reason we used tools `Autoconf` a `Automake`, with which it's possible to make a makefile suited to a particular computer. Without using these tools it would be very labour-intensive to make a portable Makefile and ensure compilation on various systems.

We provide a generated script `configure`, which serves the purpose of adaptiong configuration for a particular computer. Aside from it we also supply input files for programs `Autoconf` and `Automake`, so that the script can be regenerated at any time.

7.5.3 Translations

Links has it's menu translated into many languages. We counted with this feature from the very beginning of the project. Therefore we were facing decision how to make these translations at the beginning of the project. For string translation there is a tool called `gettext`, which we were using during the development. Gettext however showed to be limiting, because it is not portable, doesn't know other code pages than ISO 8859-2 and in libc different from glibc 2 contains many bugs.

For this reason we dismissed gettext and wrote our own system of language recoding. Languages are being added during compilation time. If a change is made, the developer has to run a script that regenerates source files in C language. The character set translation system works in a similar fashion.

7.5.4 Control

We chose browser control to be by a system of simple interactive offers. In the beginning of the browser when it was still working only in text mode we got inspired by textmode browser Lynx, that is being controlled using hot keys. This control is not much smart, because the user has to remember amount of keys for all possible functions. We kept backward compatibility of control with Lynx (hot keys are therefore the same), because it was widespread at that time and users were accustomed to its control. Moreover we added simple interactive menus, from which the user can invoke all the functions. The who do not remember amount of hot keys have therefore a way of easy control. In the text mode we also made possible to use mouse (using libgpm library). This makes control even easier. In the graphics mode we somewhat changed the control, more emphasis is given on the mouse, for example movement over links using keyboard doesn't work and mouse must be used (remark of the translator: now it works, it has been added as a fature).

We tried for the control to consume minimum screen space (which has a great value especially in the text mode where the resolution is small), therefore the menu is not visible on the first sight and is invoked by escape key. We don't like the overcombined control in other browsers in which we not only badly orient, but also it consumes significant part of the screen area. In **Links** the screen is consumed only by status line (below) and page

title (above). We categorically dismissed a bar for entering URL and a bar with icons, because they consume space and these functions can more easily be invoked by pressing a key than trying to hit a button with mouse.

7.5.5 Fonts

During writing the graphics part we were facing the problem what fonts to use for typesetting. There was a requirement of easy legibility and scalability. We had basically a possibility to use already existing fonts – for example from X Window or Ghostscript – or distribute our own fonts. We chose our own fonts for portability reason and independence on other programs and setting of user's computer.

Then we had a choice whether to use vector fonts or bitmap ones. We chose the bitmap variant especially for the reason that to be able to antialias vector fonts in a sufficient way, it would be necessary to generate them in much larger resolution than the resolution of our bitmap fonts is. It would consume very much time, moreover such generated bitmap would have to be resampled, which would also consume lot of time. To add or change font the user would need special typographic tools for work with vector fonts, this way only any graphics program or scanner suffices. If someone wanted to add a font from a book, he would need a specialized software for vectorization of raster format, this way any graphics program (for example Gimp) suffices.

Bitmap files are part of the binary file which again improves portability, because the fonts don't need to be searched for in various directories and also installation is made easier. The fonts are stored in a big resolution in PNG format (which substantially reduces their size) and during display they are being antialiased for their legibility to be improved also at small sizes (practically tested: at 8 pixel size the antialiased font is still legible and X font is already illegible). For better legibility we used font Computer Moderm from Ghostscript. The user can also smoothly set type size, which will definitely appreciate people with sight impairment.

8. Project Result

The result of the project is a single binary file which can be executed and work in text as well in graphics mode. The code is written portably, so that it is possible to compile it without problems on all Unices and also on OS/2. In graphics mode graphics system X-Window is especially used, which is on said operating systems most used and most portable. This also allows using the browser across the network on a remote computer. Further graphics systems SVGAlib (on Linux) and Pmshell (on OS/2) are supported.

All code is written in ANSI C language in a way to be portable across platforms. We recomment GCC compiler and GNU Make program for compilation, nevertheless other compilers should work too. For making JavaScript interpreter tools Bison and Flex were used. Further for easier compilation and portability programs Autoconf and Automake were used.

For text output no terminal libraries were used because they are not portable. Therefore the output is being displayed using standard ANSI terminal escape sequences with possibility of switching on various nonstandard extensions in menu, for example colours, various types or frames and cursor shape.

8.1 Javascript

8.1.1 Upcall and internal function implementation notesThis

chapter contains deviations of the implemantation from javascript norm. Especially you find here a list of things that we implemented in a different way and then also those

which we implemented in addition to Javascript 1.1 norm by Netscape Corporation.

In grammar we had to allow not ending the statement with a semicolon in addition, therefore making the source texts ambiguous, allowing mixing of array operator and member operators (`a[ ].b nebo a[ ][ ]`) and allowing opening a comment in source text (`<script> <!--`). The first two nasties even occur in an amount larger than small in examples in the norm, what javascript should properly look like and how should it work.

Against specification we added event handlers `onkeypress, onkeyup, onkeydown` by fields for entering text.

We call `Onchange` handler at text field in case that the field loses focus and the user has changed something in the field. We didn't find in the specification how this handler should behave, therefore we were directing according to implementation in other browsers and using the handler on web pages. An interpretation that the `onchange` handler should be called every time when something is changed, but this would then correspond to the handler `onkeypress`.

Setting text in status line in `onclick` and `onmouseover` handler should according to the specification be performed only in case that the handlers returns value true. We set the text in status line everytime the scripts asks, because the implementation would be unnecessary complicated with regard to the outcome. Moreover browsers Netscape 4.51 set the value of statue value without respect to the returned value.

A rule of **delayed URL change** works in te browser. It is for ensuring functionality of event handlers (for example `onclick` at references, `onsubmit` at forms and similarly). In these cases it is necessary to wait until the handler finishes and only after that it can be proceeded to a different page (send a form, follow a link). Therefore it is not changed to the different URL until all javascripts finish in the given context. This has also it's disadvantages, for example when javascript is running on some page in an infinite loop and the user wants to click some link. In this case the links „will not work". This is not a bug, but a feature. „Manual" change of the URL will work (by entering the URL into a dialog).

With regard to often occurence of global address space on web pages we implemented allowing of resolution of names in main address space. The user an option to tick up this possibility in javascript setup. After switching this on, correct working can be observed for example with object Tdocument, when `document` is not mentioned, which ocurrs nontrivially often on pages. This solution is suggested also by the fact that some pages on the other hand require this resolution to be turned off. Neither the norm nor the document object model say anything about behaviour in such a situation, therefore we did a solution that occured to us as reasonable and usable.

The user has moreover an option to allow or forbid all type conversions (they are on by default). This switch allows conversion to be allowed which are in conflict with the norm. Namely they are conversions of type undefined to another type. When all type conversions are allowed, it is allowed to derefernce a nonexistent arrat, but it is forbidden to write into it. We built in this feature after we realized that on many pages javascript didn't work because of using undocumented variables, which are understandably not supported from our side and according to the norm have (typically for assignment of undefined value or it's conversion) to end with an error. They are for example variables that are defined only in javascript „norm" by Microsoft company.

We added a series of variables and functions that some browsers support, should it be assignment to the object `location`, or it's duplication into object `window`. Namely: `Math.md5()` is an undocumented function, which computes MD5 of a string passed as an argument. It's occurence is for example on the server `http://www.post.cz`. Next undocumented function of object `Math`, which we did implement, is `Math.floor()`. As the name suggests, returns integral part of a float The

whole `location` object can be accessed not only via `document.location`, via `window.location`, but also via `location`. It is possible to assign also a value into the object `location`. The assignment behaves the same way as an assignment into `location.href`, so it causes a transition to another page. Object `location` moreover has a method called `replace`, which accepts one argument — a string with URL. This method does the same as an assignment into `location.href` does. Function `Array` can be called also under the name `Object`. Intuition says that `Array` should be „more capable“, but constructor `Object` is not described anywhere and when we were searching for it in the terrain, we came to a conclusion that from all it resembles most the function `Array`. Therefore we made these two identical. At objects `Date` a `Image` some variables may be fake, it means implemented, but doing nothing.

We do not support changes of image size, should it be by writing into variables `image.width` a `image.height`, or by change of the image source. If we allowed javascript to change sizes of images, then after every change the whole page would have to be reparsed, which would on one hand look awful and on the other would be very slow. Write into attributes Twidth a `height` is forbidden and after a change the new image will be displayed in a cutout of the size of the original one. If the new images has different dimensions it is either accompanied with background or clipped to given size. Typically the new image has either the same size as the original one (usually javascript changes for example button from depressed state to undepressed and similarly), therefore we chose this strategy. Stretching the new image into old dimensions would be unnecessarily slow and superfluous, especially in cases, where not much able page creator doesn't manage to create the image with the same size and the image is differing in it's size for example by 1 pixel.

Javascript doesn't have an access to background image. This image is not inserted using HTML element `img`, but is an attribute of the whole document. Moreover if we allowed changing the background image, the whole page would have to be reparsed, fonts and transparent images recalculated, which would be very demanding. Therefore we didn't allow this and javascript hasn't slightest notion about background image.

We ignore the image attribute `lowsrc`, because we don't images with low resolution. As well in javascript this attribute is not supported.

We do not support methods `open` a `close` of `dokument` object, because they use MIME-streams, which are not implemented in the browser. Similarly creeping options in the functions `window.open` and `frame.open` are not supported by us, because again they serve a purpose for the MIME-streams.

`onselect` handler at a field for insertion of text is not supported, because a text can not be selected in the field, the handler is ignored. For the same reason function `select()` at text field does nothing (function `select()` won't do anything and returns value `undefined`).

`OnUnload` handler is not supported, because during leaving the page the javascript is killed and context deleted, therefore this handler would lack sense.

Historii v javascriptu podporujeme pouze směrem dozadu, neboť historie dopředu v prohlížeči implementována není.

Functions working with select object return only select, where only one entry can be chosen. Other selects (where more entried may be selected) look like checkboxes. This is due to internal representation of select object inside the browser.

We didn't implement the function eval in the browser because it would be necessary to rewrite the javascript interpreter. The problems is that in the function `eval` it is necessary to take an argument, run again lexical and syntax analysis on it and then interpret it. The problem with lexical and syntax analysis would not be as significant

as with the interpretation. In the interpreter we suppose that in one context only one script runs. This way in one context two scripts would run, as `eval` needs to access the original tree (into the same „address space“), and our interpreter is not prepared for this. Therefore an implamentation of the function `eval` would require a large change. Reconvalescence from errors during interpretation would be very unpleasant. Although it looks like this function doesn't have a significant importance, as most usages can be written in a different way, it occurs quite often in praxis. Lazy and inept authors of javascript use it even for such trivial operation like adding two numbers (a calculator written using `eval`). And as such pages are usually based on `eval`, javascript doesn't work on them with **Links**.

We already managed to design a way how to implement the `eval`, unfortunately, we didn't have enough time to perform these changes and mainly followingly test the code sufficiently, therefore we didn't implement eval before finishing the project. However we plan to implement this construction in the nearest future.

We implemented eval for the time being because of often usage for string concatentaion that it only concatenates strings that it gets. This made many pages functional where `eval` didn't work with javascript.

For the same reason the function `sort` is not implemented. It's implementation is similar to the `eval` function. `sort` gets one argument which is a function to compare elementes. Calling this function would be done in a similar way as processing arguments by `eval`. SImilarly it's with error recovery. For the same reason also implicit call of function `toString` or `valueOf` doesn't work.

Another often construct which we met on web pages is function call in a member expression (contruction of type `document.function().element`. This construct is in collision with norm grammar. We didn't support it at the beginning, but because it was occuring too often and we found a way how to implement it nonviolently, we finally supported it.

Associative array is also not in norm (in the 1.1 norm array can be indexed only by nonnegative number, associative array is in norms newer than 1.1), but regarding the fact that it is largely exploited in praxis, we decided to support this construction. First we were thinking that it will be too complicated to implement it, but finally we found a way how to implement an associative array.

We do not support `prototype` attribute that is by all objects. This attribute is allowed by the norm and is adding new methods and attributes to the object (possibly with given value). We didn't implement this attribute because it would be a problem to implement it and our opinion is that it is not useful. This attribute is being had also for example by objects `number` and `string`, which we implement directly and no object nor attributes is counted upon with these objects.

Colours in objects `document` (for example `fgColor`, `bgCOlor` ...) are fake for the reason that they are not stored anywhere in the browser and they are being forgotten shortly after formatting the document. They are not being stored into any internal structures because at the time when the page formatter was being written, javascript was not considered. Therefore javascript cannot acces them. Writing that javascript could change colours on the page would require a design change of the whole document formatter. Not mentioning the fact that setting colours by javascript is a cosmetic issue which doesn't harm page display nor browser functionality in any way.

`this` object is implemented exactly according to the Javascript 1.1 norm, so it points to current object, which is either the main address space, or local address space (in case that we are in a function). A correct usage therefore looks like this (manufacture of local variables):

```
function f(){this.a='a';}
```

However we have already met several pages where object `this` is used for example for identification of an object inside an event handler. Therefore similarly as in for example in C++. Here is an example of bad usage, which is in a conflict with the Javascript 1.1 norm:

```
<img src="a.jpg"
onmouseover="this.src='b.jpg';">
```

Which should create a picture „a.jpg“, which then changes to picture „b.jpg“ when a mouse is placed over it.

8.2 Formats

Links knows displaying images GIF, JPEG, PNG, TIFF, and XBM. PNG, TIFF and JPEG images are being decoded by libraries `libpng`, `libtiff`, `libjpeg`, respectively. For images GIF and XBM we wrote our own decoders. Thanks to used libraries the browser supports all variants and peculiarities of JPEG, PNG and TIFF formats.

8.2.1 Animated GIFs

The browser intentionally displays only the first frame from animated gifs. This approach brings a significant relief to the user from unpleasant advertisements. Most advertisements are animated GIFs. Flash animation is not supported by the browser at all, for the same reason. In flash animations and animated gifs there is in vast majority of cases no information necessary for serious work with web content, therefore this feature doesn't bring any handicap for professional usage of WWW space.

8.2.2 HTML

The browser support HTML format version 4.0. The browser hasn't CSS (cascading style sheets) implemented. Our implementation differs from the norm in the following tags:

- **OBJECT** tag is not supported, because we wouldn't be able to always display it's content. Therefore instead of it a link is being displayed (the user is presented with a button [OBJ]), after whose clicking the content is displayed. If the content is of some particular MIME type, then the user has an option to set up an external viewer. Therefore for example, when the content of OBJECT is a flash animation and the user sets up a flash animation viewer, after clicking on the link a flash animation browser is automatically invoked. If the content of `OBJECT` tag is an image, the image is directly displayed (and therefore [OBJ] is not displayed anymore).
- **LINK**, or a document header link, is displayed as a link at the beginning of the page. The user sees for example: `Link: Next page`, where there is a link behind „Link:“, which leads to the target of the `LINK` element.
- **IFRAME**, or inserted frame, is also displayed as a link (in the place where tag `IFRAME` occurs). We didn't implemented inserted frame because is froom 98 % used for advertisement purpose. Similarly to previous cases, the user is presented with a link for example `IFrame: main.html`, on which he can click and after clicking upon which the desired frame is displayed.

Moreover we implemented HTML tag `embed` which behaves like the tag `img` if the source is image, otherwise only a link is displayed saying that it is an `embed` tag. This tag is an extension of HTML 4.0 used by both browsers Netscape and Explorer. Every manufacturer at his extension implementation accepts different attributes. Fortunatelly attributes `src`, `width` and `height` are supported by all vendors and have also the same meaning. We implemented the tag only with three attributes.

We decided to implement `embed` tag after realizing that some web pages display images using this tag and that these images are not visible without `embed` tag and the page looks different.

Image maps (tags `ISMAP` and `USEMAP`) display one button, after pressing which a menu is displayed, where the user can select individual links from the imagemap.

A redirect in HTML code is as contrary to HTTP redirect not performed automatically. If a redirect occurs on page (`<META HTTP-EQUIV=„Refresh" ...>`), `Refresh:` is displayed in the browser, after which a links to the respective page follows. Writing the redirection automatically would not be a problem, but we chose a manual variant, because it's very annoying to get back for a user in case of automatic redirect. When he goes back, refresh automaticall redirects him forward and so on in a loop. Therefore the user must invoke the „history“ function, where he in a complicated way selects an entry before the redirection, and only after that he gets before the redirection. , where he in a complicated way selects an entry before the redirection, and only after that he gets before the redirection.

8.3 Others

8.3.1 Cookies

Links fully supports cookies, only with slight security limitations. The browser doesn't accept cookies from nonstandard second-order domains, the domain must be at least third-order one. An example is a cookie from „domain.com“, which the browser doesn't accept, because the domain is a second-order one. This security measure is

because of existence of domains like „.co.uk“ or „.co.jp“, which are too general. The browser accepts cookies from second-order domains for so-called standard domains, which are for example „.net, .org, .com“ and similar. For security reason **Links**' doesn't save cookies to disk, as opposed to other web brosers, because cookies very often contain sensitive informations, which someone unwanted could read from the disk.

8.4 Cache

Links uses 3 file caches:

- **file cache**, where all downloaded files are being stored
- **document cache** for already formatted documents
- **image cache** for storing partially and completely decoded images (running images decoders).

Document and image caches use LRU strategy for removing documents. File cache also uses LRU strategy, but first removes large files (gives advantage to small files). Cache is being walked through in three passes after addition of a file and during determination which files to remove. In the first pass files are being walked through from oldest to youngest and as long as the size exceeds ccertain limit, the files are being marked for removal. In the second pass the whole cache is being passed through from the youngest files and if a marked file is found, which could still be kept in the cache without exceeding the cache size, the file is unmarked. Finally in the third pass marked files are removed from the cache. During download into a file („download“ function) files are not being placed into the cache if their size exceeds 1/4 of maximum cache size.

The size of file and image cache can be set by the user in a menu to customize the browser as much as possible to parameters of his own computer. Already formatted documents are being stored into cache which shortens latency during cache usage. As well setting number of formatted documents is left to the user. Both constants are by default preset to low values (1MB size for image and file cache and 5 documents maximu) so that the browser can be used also on systems with little physical memory.

We intentionally didn't implement disk cache because we think that it doesn't have a meaning and only takes up space on the disk. In a typical case the user runs a browser, uses it for a long time and then closes it. After some nontrivial time he runs it again. As the new run occurs typically after long time (hours to days), most documents in the cache expires and therefore the cache loses it's meaning. Moreover a disk cache is dangerous because someone unwanted may look which pages the user browsed (or is browsing at the moment).

The memory cache is aggressive, i. e. stores all documents regardless of the „nocache“ parameter and ignores expiry time. We implemented the cache in this way after we realized that both the „nocache“ parameter and expiry time are being often abused by page authors for example for advertisement downloads. Agressive cache very reduces network load during display of documents that have been displayed before. If the user doesn't want to use the cache, he has to explicitly request document reload by pressing CTRL-R.

8.5 Outstanding features of the browser

- **Portability** on all systems Unix, OS/2 and Cygwin (under Windows)
- **Table, frame and image display**
- **Pull-down menus**
- **Bookmarks** — the user has a possibility to store bookmarks pointing to interesting pages, the bookmarks are organized in convenient directories. Format of the

bookmark file on disk is compatible witn Netscape Navigator.

- **Translations into many languages**
- **Downloading files in background**
- **Keepalive connections** — the connection with the server is being kept for some time after downloading a document, because downloading more documents is expected. Then the connection need not be re-established every time and the user waits for a shorter time to download a page.
- **Asynchronous DNS lookup** — during waiting for a DNS reply (server for translation of domain names into IP addresse) the browser won't „freeze up“ like other unnamed browsers. The user won't basically ever notice that it is being waited upon a server reply.
- **Possibility of running external programs** on unknown data types — the user has an option to prescribe programs that should be run on files that **Links** is not able to display: for example music, video, and various documents (PDF, PostScript, Word) and similar.
- **Possibility to switch between a textmode browser running on a console or a graphical browser running under SVGAlib, OS/2, X-Window, AtheOS**
- **Displaying a page already during loading** — as soon as a page start being downloaded from the Internet, then almost immediately the user can start reading.
- **HTML version 4.0 without CSS**
- **Cookies** — without saving to disk, for security reasons a bit stricter accepting.
- **Caching** — storing documents into memory, during return to a page (for example by the back button) the page is being loaded from memory (cache) and waiting for network need not be performed. The cache doesn't honour „nocache“ and „expire“ parameters. The cache uses LRU strategy.
- **Javascript** derived from specification 1.1 of Netscape Corporation, document object model taken from Netscape 2.0 according to the found documentation, some enhancements added.
- **Cache for formatted documents** — **Links** is the only browser that has a formatted document cache. Thanks to this cache a document is displayed immediately upon return to the page and the user doesn't have to wait long for formatting the page.
- In a case of bad HTML code the browser will not crash.
- After reparsing or reloading a page the user-filled content of forms doesn't get lost.

8.6 Supported protocols and formats

Links supports these protocols and formats:

- **FTP** — according to RFC 959
- **Finger** — according to RFC 1288
- **HTTP** — version 1.0 (according to RFC 1945) and 1.1 (according to RFC 2068), version is being automatically switched. Some servers know only 1.0 or contain bugs, so that only 1.0 can be used with them.
- **HTTPS** — according to used SSL library
- **Javascript** — the basic concept is designed according to javascript 1.1 norm by Netscape Corporation. Implementation is not exact, doesn't know for example eval function or sort method at array. Most changes are in the direction of gen-

eralization and allows interpretation of scripts that are against the norm. These generalizations have been partially motivated by the fact that the examples in the norm were in conflict with the norm itself. In javascript many „dirty practices“ have been allowed, because javascript on web pages has (mildly said) very long way to the norm. Deviations from the norm were usually implemented according to other web browser, especially Netscape, version approximatel 4.51.

- **HTML 4.0 without CSS** (without Cascade Style Sheets)
- **JPEG** — decoded by libjpeg library, we support progressive download, we support JPEGs with 8 bits per channel and do not support JPEGs with 12 bits per channel because libjpeg always knows only one varian. We didn't hit a 12-bit JPEG on the Web and usual JPEGs are 8-bit.
- **PNG** — decoded by libpng library, we support gamma correction, alpha channel, progressive loading.
- **GIF** — own decoder, supports transparency and interlacing. Only the first frame is being displayed from animated GIFs.
- **XBM** — own decoder, support X10 format.
- **TIFF** — decoded using libtiff library

Images are being calculated in colour depth 16 bits per channel with minimum possible degradation for a monitor. We do not support white point calibration, because it is not much visible when omitted, the user typically doesn't know the data about the monitor and it would be too complicated to calculate the values.

8.7 Jazyky

The browser is localized into about 30 foreign languages. As we do not know all these languages, we had to rely on translations that excited users sent to us. Therefore all languages except English and Czech do not originate by anyone of the author team. These translations originated in the times of purely textual version of the browser and with adding more function and new texts it was not within our capabilities to update all languages, therefore some texts in some languages are not translated and display in English. Therefore it depends solely on generous users if people that do an update will be found.

Links is translated into these languages:

- English
- Brazilian Portuguese
- Bulgarian
- Catalan
- Czech
- Danish
- Dutch
- Estonian
- Finnish
- French
- Galician
- German
- Greek

- Hungarian
- Icelandish
- Indonesian
- Italian
- Lithuanian
- Norwegian
- Polish
- Romanian
- Russian
- Slovak
- Spanish
- Swedish
- Turkish
- Ukrainian

8.8 Platforms

Links is running on these platforms under text mode and under X-Window system. With some operating systems there are additional graphics platforms in parentheses under which **Links** runs under these systems:

- AIX
- AtheOS (AtheOS graphics environment)
- BeOS
- Cygwin under Windows
- FreeBSD
- FreeMint
- HPUX
- Irix
- Linux (SVGAlib, framebuffer)
- MacOS X
- NetBSD
- OpenBSD
- OS/2 (Pmshell)
- Solaris
- SunOS
- Tru64

The source package contains also a historic Win32 port, but this is unstable and wasn't written by anyone from the team. We left it in the project if somebody wanted to use it in the future.

8.9 Bugs and problems with the program

8.9.1 Wannabe bugs

Here is a list of the most common problems that are not caused by the **Links** browser althought they may look like that and users complain about them as if they were bugs of **Links**. There is an explanation added to every problem.

- Some old versions of libpng do not know how to process certain kinds (the more exotic kinds) of PNG images, because functions implementing interface to features these PNG's are having are missing in them. Therefore **Links** in case that it's compiled with such a bad libpng this automatically detects thanks to autoconf and in case that such a PNG is encountered, it will terminate with a message that the user should get a better libpng. Therefore it's necessary to use reasonably new libpng.
- Some libtiff and libjpeg libraries can be compiled without support for certain images (for example TIFF with LZW compression, TIFF with JPEG compression and similar), therefore related pictures look like a small broken frame (broken picture) with such libraries. The users then complain because they think that this is a bug of **Links**.
- Some bad pages lack second quotation mark after clickable button „Send“, so that this doesn't display and the users complain on a bad address.
- Some badly writte pages don't contain a semicolon after constructions like `&`, `&quot;`, `>`, ` `..., which results in that the according `&amp` (or others, respectively) is printed out. Users incorrectly consider this to be a bug of **Links**.
- Some pages from servers by the Microsoft company contain numeric representation of an unicode character, but the number is not an unicode code. The users than incorrectly blame the browser for displaying a blotch instead of their beloved character. oblíbeného znaku zobrazí kaňku.

  The code for primitive operations in 16-colour modes and 2-colour mode is written very wrong, therefore these two modes work very slowly — this is to be blamed on SVGAlib and not on **Links**.
- SVGAlib before 1.9.4 must be suid root and **Links**also isn't responsible for this. The new SVGAlib already has a module in the kernel and the user program is then harmless.
- libpng about before 1.2.0 contained a bug in MMX optimized code, which caused that every time libpng was compiled with MMX optimization and displayed a certain image from the PNG test suite, **Links** crashed on segfault, because the whole address space has been overwritten by a constant repeating value.

8.9.2 Problem with SVGAlib

It can happen during switching of graphics console in SVGAlib in Linux, that an application crash occurs or in worse case the whole system crashes. This is caused by broken-by-design API of VT_ACTIVAYE in the Linux kernel, which causes unreliability of graphics applications during console switches. This is not only our opinion, but this idea is supported also by the following mail from the linux-kernel mailing list.

```
From Peter Benie <peterb@chiark.greenend.org.uk>
Delivered-To: clock@atrey.karlin.mff.cuni.cz
To: jsimmons@transvirtual.com,
        Karel Kulhavy <clock@atrey.karlin.mff.cuni.cz>
Cc: linux-kernel@vger.kernel.org
Subject: Re: Two rapid VT_ACTIVATE's
In-Reply-To: <Pine.LNX.4.10.10204050924080.21397-100000@www.transvirtual.com>
Organization: Linux Unlimited
Cc:
From: Peter Benie <peterb@chiark.greenend.org.uk>
```

```
Date: Sun, 07 Apr 2002 23:42:42 +0100

In article <Pine.LNX.4.10.10204050924080.21397-100000@www.transvirtual.com>
you write:
>
>> I wonder what happens when somebody calls VT-ACTIVATE in close
>> succession.  So that the second one overwrites the first one. Let's
>> assume the first one is generated by program #1 and the second one
>> by program #2. And some of the programs may be using graphics and
>> some of them text. What will happen? Will screen corruption and/or
>> kernel crash induced by botched videocard registers result?
>
>In theory the vt_dont_switch lock should protect you from this.

vt_dont_switch has nothing to do with it, and is fundamentally broken
for other reasons. (You need a call to atomically activate and lock a
particular VT, not a call to lock whichever VT happens to be selected
at the moment.)

With the VT_ACTIVATE race, what will typically happen is:

#1 calls VT_ACTIVATE to request the VT change
#2 calls VT_ACTIVATE does likewise

#1 calls VT_WAITACTIVE to wait for the change to complete
#2 calls VT_WAITACTIVE does likewise (and overwrites want_console)

#2 returns from VT_WAITACTIVE (there is now no VT change pending)
#1 does not return until the user explicitly changes to the right
console, or the ioctl is interrupted by a signal

You can see this in practice if you start two X servers from xdm.
After a while, xdm sends a SIGTERM to the X server that hasn't started
yet, interrupting the ioctl, but the X server doesn't check the return
value from the ioctl, assumes the VT change completed, and fiddles
with the video card parameters. Sometimes you get a peculiar mixture of
two X servers, sometimes the machine just crashes.

Clearly, the crash is due to a bug in the X server (ignoring the error
from the ioctl), but the VT API is also faulty since the X server
cannot find out that its VT_ACTIVATE was 'lost' by the kernel.

Another API problem in this area is with VT_OPENQRY. An application
calls this to get the next free VT number, then VT_ACTIVATE to swtich
to it, allocating it if necessary.  If two applications do this, they
will both get the same VT number from VT_OPENQRY, and both will then
call VT_ACTIVATE and VT_SETMODE. Both applications believe they
have a VT which has been allocated and locked, despite only having one
VT between them!

Peter
```

8.9.3 Problem with framebuffer and gpm

During writing a driver for linux framebuffer we encountered many problems. First we were coping with very weak or nonexistant documentation of the framebuffer API.

When we managed to write the driver by reverse-engineering of framebuffer drivers for individual cards and of linux kernel sources, we hit another problem — the mouse.

The only possibility how to write mouse reading on framebuffer is through the gpm library. gpm is a daemon that reads mouse and allows copy-paste of text using the mouse. It allows the clients – programs – to read coordinates of the mouse from a socket. This way reading the mouse is implemented in text mode. It is not possible to read the mouse in a different way on framebuffer. It would be theoretically possible to read directly the mouse device, but it is not guaranteed that on every system the user will have an access to the device. For example in unix laboratory at Lesser Quarter, Prague, Czech Republic the users are not allowed to read from the mouse device. Moreover, for this solution it would be necessary to ask the user for mouse type and write drivers for all mouse protocols.

Another possibility how to write a mouse driver for framebuffer is changing already existing gpm and either use it, or send the changes to the authors for them to release them officially with a new version of gpm. The first solution hits the already mentioned access rights problem, the second one would be better, but not everyone would have this new library installed on his computer and therefore again without administrator rights this solution would be useless.

Therefore we hadn't other option than writing the mouse on framebuffer with using already existing gpm library. But problem is that gpm is strictly textual and the authors didn't count with framebuffer existence during the development. Therefore the mouse resolution is limited on the text screen size. Coordinates of the mouse (both relative and absolute) are then given in character, not in pixels. As reading the mouse by characters is too rough, we solved the problem by counting relative movement of the mouse. It must be however still multiplied by a constant, because the cursor moved otherwise too slowly. The mouse sensitivity is unfortunately set up „hard“ by the administrator during running gpm, here the user application can't set it itself. Therefore we chose a compromise between accuracy and mouse sensitivity. This however brings a problem that an object on the screen can exists that is not reachable by clicking. This danger is unfortunately real, however nothing can be against it. It can be partially solved by for example enlarging the letter on the screen or by enlarging images.

To solve this problem by available means, we allowed the user to move the mouse finely by pixely with keys F5 – F8. For users that have administrator rights we made a patch for the gpm library, which adjusts gpm so that the mouse pointer in **Links** moves smoothly. The patch is included within the distribution and is meant for library gpm 1.20.0, however with different version it should work too (or possibly after a small change).

gpm also doesn't allow to read mouse wheel, therefore mouse wheel doesn't work neither in text mode nor in framebuffer.

There are the mentioned problem with mouse on framebuffer, however when the mouse can't be read in a different way, nothing can be done with it, This solution is however, as is said, „better than a sharp stick in the eye“ We implemented framebuffer even despite these limitations for reason that some users may be forced to use it and if they don't have a possibility to use another graphical interface, they will be able to run the browser at least with these limitations.

8.9.4 Language translations.

Links is translated into about 25 languages. These translations except the Czech one have been performed by external authors exclusively for text version of the browser. As we have added new texts into languages and we don't rule with all the languages, the languages may contain English texts. This is in case that given text hasn't yet been translated into that particular language.

8.10 Comparison with other browsers

Other browsers have compared to **Links** a series of interesting features, like random freezing, slow start, improper display of images, unmanageability of javascript execution. For example Netscape arbitrarily and absolutely randomly stops displaying the document (will display an empty page) and then must be restarte for it to be usable. Javascript in Netscape causes severe errors and crashes of the browser, which must be restarted (by switching javascript off these problems disappear).

Regarding image display, the browsers have usually problems with transparency and with displaying PNG image format. For example Netscape Navigator 4.51 and lower display the darkest grade of gray as white, which renders black/white images unintelligible. MSIE 5.50 incorrectly displays slpha channel, instead of the page background there is always white or yellow in the transparent image (tested on PNG test suite `http://www.libpng.org/pub/png/pngsuite.html`), PNG images only with transparency and background chunk are displayed as opaque with gray background, which is against PNG standard which says that background chunk should be ignored and the image displayed as transparent. No other web browser known to us is able to apply gamma correction on images nor dither them. Browsers have ugly and badly legible typeface, which can be enlarged only in limited scale. The typecase is dependent on installed fonts in X-Window system.

The browsers have no or badly implemented file caching, which „appreciate“ especially users of modem connection (or some other slow connection) or users paying for transferred data. For example Netscape browser is still downloading files from the network even if the user goes just back in history, when the user would expect immediate page display. If the browsers have a cache then only for files, the documents are being reformatted again and again. During page reload the browsers don't keep contents of forms.

The browsers have over-complicated control, it is not uncommon when a quarter to third of window area is consumed by overcomplicated user interface. Even experienced users lose a considerable while to orient in user control of the browsers, when they see one for the first time. Even looking for the URL entry bar itself is very demanding. The user is permanently offered something he doesn't want. The browsers are changing URL at their will (for example when Netscape is run for the first time and the user wants to enter URL). The Opera browser even forces the user to look at adverts even before the first page is displayed. Moreover, these adverts can not be turned off.

Netscape is not able to display a table when the terminating `</table>` element is missing. It also contains amount of further subtle bugs, for example in case of an empty table cell (`<td></td>` it doesn't display the frame around the cell, or when a black background and white text is set, buttons, radio buttons and checkboxes are displayed over white background and only when forms are inserted into a table, the elements of the forms are displayed correctly. Moreover, it doesn't display tables during download of the document - the user is forced to wait until the whole table is downloaded and only after this the table is displayed, which is unpleasant with big tables (for example overviews, prices lists and similar). **Links** can format also partially loaded tables. Document (including tables) is being reformatted during download.

The greates problem is experienced by browsers prbably with javascript and security. Usually a very short and very trivial script suffices to ruin the browser. We tested browsers Mozilla 6.0, IE 5.50, Netscape Communicator 4.77, Opera 6.01 (for Linux as well as for Windows), Konqueror.

```
<html><body>
<script language=javascript>
while(1)alert('Click as you want!');
```

```
</script>
</body></html>
```

This script hasn't been survived by any web browser we had access to. All web browser happened to be unmanageable. Some of them ferociously created more and more windows, some of them waited for user's click. But all browsers had to be restarted for a regained usability after running this script.

Links doesn't suffer the lightest problem with these scripts, because the user has the possibility to stop interpretation of all scripts with every dialog created by javascript. Therefore an alert will appear and after pressing „Kill script“ the user can continue in his work with the browser.

```
<html><body>
<script language=javascript>
a='a';
while(1)a+=a;
</script>
</body></html>
```

This script will mount an attack against memory by continuous allocation. It doesn't attack only the browser, but the whole user's computer. With Netscape 4.77 browser on Linux it will cause an unpleasant quarter hour with disk lighted up and **very** slow reactions of the computer. Remaining browsers also crashed or stopped responding and started to overload the system massively, however with Netscape 4.77 the results were most fatal.

V **Links** will this script end up soon, because it will exhaust memory limit assigned to javascript. The user has a possibility to set up this memory limit in menu.

```
<html><body>
<script language=javascript>
while(1)document.location="again-and-again.html";
</script>
</body></html>
```

The above mentioned script expects that it will be saved in the file `again-and-again.html`. The only tested web browser that neither crashed after interpretation of this script, nore became unmanageable, was Opera browser on Linux operating system, which was furhter usable after light difficulties.

Links this script again doesn't make difficulties, because at every URL change the user is asked whether he wishes to approve the change or dismiss it. The user has again the possibility to immediately stop interpretation of all scripts in a dialog. Moreover if the URL is the same as URL of current page, nothing is performed. In this case nothing will then happen. The attackes can however simply bypass this arrangement simply by making two scripts, which will point each to the other. But in this case the user will be asked by every URL change, therefore he will have an option to stop the „attack“.

```
<html><body>
<script language=javascript>
while(1)window.open("window-still.html");
</script>
</body></html>
```

This script is a light variation on the previous one, we suppose that the script is stored in the file `window-still.html`. The difference is only in that the script opens new and new windows, which way it clogs up the system, or ar least makes the browser unusable. Even, not a signle tested browser passed this test. In case of IE the operating system Windows 2000 has sever problems with so many windows which left permanent consequences on a task manager having been run, which could not be closed, because it's windows started to be displeyed without the top quarter.

The script again didn't make problems to our user, because before every opening or closing window the user is asked for a confirmation. Therefore a small window was displayed whether the user wished to open new window and the user has again the choice to stop interpretation of the scripts.

```
<html><body>
<script language=javascript>
function f(){f();}
f();
</script>
</body></html>
```

Infinite recusion is already taken care of in newer browsers. Browsers Opera and Mozilla did nothing, other browsers (Netscape, IE, Konqueror) almost immediately crashed.

If this script is run in **Links**, a dialog will pop up soon that javascript exceeded limit of maximum function recursion, and that the script is violently terminated. The limit can be again set up in a menu by the user.

We consider the mentioned problems of browser being **very severe** security problems. We hold the opinion that this should not happen to a correct browser in any case. We counted with this philosophy from the very beginning of designing javascript and thus we armoured the browser agains all attacks from the javascript side. Javascript is strictly written with the goal that the user has full control over the script under all circumstances.

8.11 Used libraries

During writing the project we tried to use as few foreign libraries as possible for the code to be as portable as possible and for the user not to have to first install megabytes of various libraries, when he wants to use **Links**We got convinced from our own experience that different versions of libraries are often mutually incompatible (even backwards). Libraries also contain various bugs and security holes, which would be brought into **Links** browser this way.

Therefore we employed libraries only for image decoding (jpeg and png) and for work with SSL protocol. These libraries are very common and relatively stable and with respect to complexity level of problems they solve it wouldn't pay off to write an own code. Own code could contain more mistakes than time- and and praxis- proven library code. Moreover writing these libraries would require much time for study of documentation describing these standard, not talking about time necessary for testing correctness of implementation.

In text mode there aren't necessary any special libraries and only the standard `libc` library suffices. For compilation in graphics mode the following libraries are necessary:

- `libpng, libjpeg, libtiff, zlib` — for work with images
- `libvga` — for work with Linux SVGAlib

- `libX11` — for work with X-Window system

The following libraries are not mandatory, but recommended. Without them the browser functions mentioned by the library won't work.

- `libssl, libcrypto` — podpora protokolu SSL
- `libgpm` — podpora klikání myší v textovém režimu na systému Linux

8.12 Development and testing

8.12.1 Development environment

Links has been most time developed on Linux operating system. Part (some code of Mikulas Patocka) has been developed on OS/2. For compilation we used mainyl the GCC compiler (mostly versions 2.95.x and 2.96.x) and GNU Make tool (version 3.79). All the code has been written in VIM editor. Autoconf (version 2.13) and Automake (version 1.4) are among support programs we used. Javascript parser has been written using the tool Flex (version 2.5.2) and Bison (version 1.24). X-Window interface has been developed mainly on Linux XFree86 version 3.3.6 and 4.0.2. SVGAlib interface has been tested with SVGAlib 1.9.x and 1.4.x. The fonts have been processed by programs GIMP and ImageMagick.

8.12.2 Portability testing

We performed the portability tests on Unices and machines available in computer labs and on our own (and for us available) computer. We performed test for Alpha through an account on www.testdriver.compaq.com.

- Linux on x86
- Linux on Sparc64
- FreeBSD on x86
- OpenBSD on x86
- NetBSD on x86
- OS/2 on x86
- Cygwin under Windows
- BeOS on x86
- Atheos on x86
- Solaris on Sparc64
- Irix on SGI
- Linux on Alphě,
- Linux on PowerPC
- MacOS-X on PowerPC
- Digital Unix on Alphě
- Tru64 on Alphě
- FreeBSD on Alphě
- NetBSD on Alphě
- OpenBSD on Alphě
- Linux on IA64

Other declared supported operating systems are unfortunately only based on verbal information, because we hadn't a chance to testit. We got known about them from users that wrote us excited that **Links** runs on even such systems.

We tried to compile with compilers:

- GCC 2.7.2.x
- EGCC 1.0.3
- GCC 2.95.x
- GCC 3.0.4
- GCC 2.96.x
- GCC 2.95 — compilation couldn't be performed because the compiler contains too much bugs (for the program to compile all optimizations had to be turned off).
- WorkShop Compilers 4.2
- MIPSpro Compilers 7.2.1
- Apple Computer, Inc. version GCC-932.1 (based on GCC 2.95.2)
- Sun WorkShop 6 update 1 C 5.2 2000/09/11
- Compaq C V6.3-129 (dtk) on Digital UNIX V4.0G (Rev. 1530) Compiler Driver V6.3-126 (dtk) cc Driver
- Compaq C V6.4-014 on Compaq Tru64 UNIX V5.1A (Rev. 1885) Compiler Driver V6.4-215 (sys) cc Driver

8.12.3 Testing the program, spotting bugs and optimization

During the development we used method of regression code testing. This method is for example successfully used during the development of GCC compiler. In our case it consists of testing the browser and with every problem (crash, memory leak, infinite loop, stack overwrite or any other error) determining the direct cause and storing the conditions during which the problem occured. Therefore downloading and storing HTML page on disk and storing keys that have been pressed. After this it is always possible to summon all situation at which some problem or error happened. This allows a control, that the given bug has been really removed and that by later code changes, the bug hasn't been introduced again. It's just necessary to run an automated script which tests the browser on problematic inputs.

We introduced this procedure after we discovered that some bugs repeat. We did very intensive testing of the browser as we didn't practically used any other browser for web browsing except **Links**.

For removing memory leaks and shooting into memory we used our own technologies. We wrote wrappers around malloc, free, realloc, ... functions. These functions create a red zone around allocated memory and save informations on which line of code the memory has been allocated and how many bytes has beebn allocated. The uninitialized memory is filled with a pattern to earlier detect usage of allocated, but uninitialized memory (the pattern usually causes an error or crash of the program). At the time of freeing the memory, consistency of the red zone is being tested, the freed memory is being filled with pre-arranged pattern and at the end of the program it is being examined whether all memory blocks have been freed.

Employing this mechanism we achieved very efficient detection and removal of memory leaks, writes off allocated space (for example array overflow) and detection of uninitialized or already deallocated memorry usage. For this purpose a tool alread exists – the `electric fence` library, however this library is very inefficient and unusable slows down the program and also introduces bugs into the program, which would not exist without

usage of thiis library. Therefore we basically were not using this library.

To achieve maximum throughput and optimal code (especially in graphics routines) we were using code profiling technology. We compiled the program with profiling information support, after that we were testing the browser on graphically very demanding inputs. Using the program `gprof` we then assessed, how much time is being spent in which function and how long doesn which function last. This way we obtained very precious informations for following intensive manual code optimizing based on compiler and machnine code knowledge. This way we managed to optimize the code very efficiently.

8.13 Tools recommended for compilation

During writing the javascript interpreter we used tools `Flex` and `Bison` for manufacture of automata for lexical and syntax analysis. These tools are not necessary during compilation, because in the distribution we already ship outputs from these tools – source codes in C language.

For compilation, program `Make` and any ANSI C language compiler (GCC recommended) is necessary.

For determination of portability we used tools `autoconf` and `automake`. These tools serve the purpose of generating the file `Makefile`. Programs `Autoconf` and `Automake` create a platform-independent script `configure`, which is run during installation and which generates the file `Makefile`. Neither tools `Autoconf` nor `Automake` are necessary for installation, as we ship already generated script `configure`.

More we created couple of script for generation of fonts, translations and character set tables. These scripts should be portable on Unix systems, but again, they are not necessary for installation, because in the distributiion package there already are C language source files generated this way.

8.14 Code taken over from other authors

No character set translations with exception of English and Czech were written by anyone from the author team. The translations are taken over from enthusiasistsm which sent them us.

9. Conclusion

We think that the project has fulfulled its purpose, because not only we managed to write a quality web browser and learned and tried lot of interesting things, but mainly we learned group communication, interface specification and documentation writing. Which we think that was the main purpose of this project.

The project brought us improvement in C language programming, knowledge how programming should be performed and how not. We learned that the resulting code should run as fast as possible, should not contain any bugs, should fullfill norms and standard, the code should be portable, program function should be limited only to necessary minimum enough for effective work of the user. The program should be monolithic and heterogenous. It should be kept in mind during programming that any bug found in the program can be considered a total failure of the program.

The programmer should not yield to creeping featurism during programming (creeping featurism means wrapping more functions over the existing program like snow on a snowball). Programming should not be performed with bugs and in a way that first the program is written faulty and after that, based on bugreports, iit is being repaired. Programming should not be conveyed in contradiction to RFC's and standards. Method-

ics and structure should not be employed in cases where it harms speed and effectivity. Code readability should not be enforced against speed, bugs in other libraries should not be worked around in cases where it collides with simplicity, reliability or speed. The program should not perform unnecessary operations which do nothing. Data should not be filtered througgh layers of bureaucratic interfaces, not even for reason of methodicity and/or program or design structure or code legibility. Bugs in program should not be thinked about as permissible because builtin self-testing mechanism should be able to spot or remove them. Programming should not be done in a way not related to functioning of real computer. New functions should not be added into the program in case it is not flawless. The programmer should not stop programming at the moment when the code works, but should check it after himsel in maximum possible manner. The programmer must not admit existence of testers that will test the program and must not take not seriously results of possible bugs in the program. The program should not be hurried up during programming and the programmer should not be forced into shortening the time to market.

9.1 Problems during development

We realized during the developmet of the program that honesty roughly doesn't pay off. Namely it was about adhering to standards. The browser worked exactly according to RFC and protocol specification, however as 90 % web pages is written wrong, much servers don't honour protocols, then it often happens that on such pages the browser displayes less than other browsers, which are unstable and don't adhere to standards. Nevertheless we decided to honour standards anyway, because this is the only way how to principally reach reasonable interoperability and a solution consisting of emulation of bugs of other programs showed as unviable, because bugs in other programs soon after introducing such scheme start to contradict to themselves.

Most this prolem showed probably during development of javascript interpreter. Because javascript interpretations from various vendors differ significantly and the authors of the web pages honour standards wery little.

10. Plans for the future

We would like to develop **Links** futher because we think that there is always something to improve. We are convinced about this also by reaction of part of our users in mailing-list (`links-list@linuxfromscratch.org`). For example we could support CSS in HTML, floating objects, possibly enhance javascript with new constructions to display more pages that are written wrong.

We plan to implement the `eval` construction, unallocating unused parts of the tree during runtime, possibly compression of source code kept in memory (for example holding code of the same wording only once). Aside from this we want to expand the existing document object model with for example `document.all`, `document.scripts` functions and more.

11. Used materials

11.1 Javascript

1) lecture notes from Data Structures
2) lecture notes from Compilers
3) lecture notes from Probabilistic Algorithms

4) Proceedings of the SIGPLAN 82 Symposium on Compiler Constructions, Vol. 17, Num. 6
5) info flex, info bison
6) Javascript 1.1 norm bby Netscape Corporation, Javascript 1.3
7) ECMA–262 norm
8) consulations with Mgr. Mareš, Doc. Sgall, Mgr. Bednárek and colleague Hubička
9) A. Motwani, P. Raghawan: Randomized algorithms
10) Aho, Sethi, Ullmann: Compilers: Principles, Techniques and Tools
11) K. Mehlhorn: Data structures and algorithms
12) David Gries: Digital Computer Compilers

11.2 Graphics

1) consulations with RNDr. Pelikán
2) Josef Pelikán: Pokročilá 2D počítačová grafika (Advanced 2D Computer Graphics) (study texts for the lecture on MFF UK)
3) Josef Pelikán: Počítačová grafika 1 (Computer Graphics 1) (study texts for the lecture on MFF UK)
4) Charles Poynton: Gamma FAQ (*http://www.informap.net/∼poynton/GammaFAQ.html*)
5) Charles Poynton: Color FAQ (*http://www.informap.net/∼poynton/ColorFAQ.html*)
6) norms and standards: JFIF 1.02, ITU T.81, CCIR Recommendation 601, RFC 2083, ISO DIS 10918-1, IEC 61966-2-1, ISO 9241

11.3 Graphic drivers

1) Adrian Nye: Xlib Programming Manual
2) consultation wit authors of framebuffer and SVGAlib
3) source codes of Linux kernel
4) consulation with Mgr. Martinem Beran

11.4 Credits

Here is a list of people that somehow participated on development of the browser. Namely it's regarding translations into foreign languages. Also lot of other people were participating into the project their own way, they were the people that were noticing us about vrious bugs. They are not mentioned here, because wouldn't fit these pages. This however changes nothing on their honour of testing the browser.

We would then like to thank this way to all that however participated on development of the browser, should it be by testing, reporting or fixing bugs, translations into foreign languages, their own code, or just sending ideas what to improve.

Unai Uribarri	History
Uwe Hermann	Manual page, command line switch „-version“, opening a link in a new xterm
Alexander Mai	Support for xterm under OS/2, fixing includes for AIX, updating manual pages

Dakshinamurthy Karra	porting on Win NT, storing goto hhistory
Oleg Deribas	Window title and clipboard support in OS/2
Arkadiusz Sochala	Polish translation
Dmitrij M. Klimov	Frames in KOI8-R, Russian translation
Jurij Raškovskij	Updating russian translation
beckers	German translation
Armon Red	Icelandic translation
Wojtek Bojdøl	Updating the polish translation
Serge Winitzki	Updating the Russian translation
Aurimas Mikalauskas	Lithuanian translation
Martin Norback	Swedish translation
Jimenez Martinez, Angel Luis, David Mediavilla, Ezquibela	Spanish translation
Suveg Gabor	Hungarian translation
Gianluca Montecchi	Italian translation
Sergej Boruševskij	No-proxy-for, Ctrl-W filling-in, SSL
Fabrice Haberer-Proust	French translation
Cristiano Guadagnino	Updated italian translation
Fabio Junior Beneditto	Translation into Brazilial Portuguese
Kaloian Doganov	Bulgarian translation
Baris Metin	Turkish translation
Dmitrij Pinčukov	Ukrainian translation
Taniel Kirikal	Estonian translation
zas@norz.org	Updated French translation
Alberto García	Galician translation
Radovan Staš	Slovenian translation
Marco Bodrato	Twinterm supportTwintermu
Kaloian Doganov	Updating Bulgarian translation
Olexander Kunytsa	Updating Ukrainian translation
Mediavilla David	Updating Spanish translation
Simos Xenitellis, Alejandros Diamandidis	Greek codepages and translation
Stefan de Groot	Dutch translation
Carles	Catalan translation
Ionel Mugurel Ciobc	Romanian translation
Petr Baudiš	Using „imgtitle“ when „alt“ is not present, adding „LISTING“ tag, updating manual page
Muhamad Faizal	Indonesian translation
Peter Naulls	Support for RiscOS
Jonas Fonseca	Danish translation
Miroslav Rudišin	Updating Slovak translation