

Daniil Galakhov

**CREATION OF AN EDUCATIONAL
OPERATING SYSTEM**
Using the C# and .NET Framework

Bachelor's thesis
Information Technology

2017



South-Eastern Finland
University of Applied Sciences

Author (authors)	Degree	Time
Daniil Galakhov	Bachelor of Engineering	May 2017
Title Creation of an Educational Operating System Using the C# and .NET Framework		63 pages 1 pages of appendices
Commissioned by South-Eastern Finland University of Applied Sciences		
Supervisor Timo Mynttinen		
Abstract <p>The purpose of this study was to conduct a research on Operating System fundamentals and apply the knowledge gained to a real environment by creating Educational Operating System. The outcome of this project may also serve as a starting point of the Information Technology studies for beginner students. As a secondary objective, this project intended to demonstrate the power of .NET Framework and C# Programming Language in a limited execution environment.</p> <p>Programming and Operating Systems are two fundamental disciplines of the Information Technology. Therefore, techniques demonstrated in this project can be applied practically in any environment and any area of IT. Results achieved here can later become part of a bigger educational project as well.</p> <p>This work is targeted towards students who want to learn the basics of Operating Systems and software developers willing to explore different applications of the .NET Framework.</p> <p>Theoretical part of the project includes:</p> <ul style="list-style-type: none"> ◆ Prehistory – explanation of how Operating Systems came alive and what originally led to their creation, interesting historical facts about different Operating Systems, development of different kernels. ◆ Basic concepts – reflection of internal operation principle of the Operating Systems, their main components and functions, different types of kernels, memory management, types of file systems, bootloaders, etc. ◆ Definitions and environment – planning and defining the Operating System itself, project hosting and version control environment, debugging, compiling and virtualization tools. <p>Practical part of the project consists of:</p> <ul style="list-style-type: none"> ◆ Designing and building the core – creating the kernel and file system, building basic CLI, implementing and connecting necessary plugs, structuring and optimizing the code, etc. ◆ Developing the shell – constructing the shell scripting language, virtual text screens and command processor. ◆ Testing – running the system on real hardware, software debugging. 		
Keywords Operating System, C#, .NET Framework, IL2CPU, Open-source, Education, Shell, DimOS.		

CONTENTS

1	INTRODUCTION	1
2	MOTIVATION.....	2
2.1	Project Naming.....	2
2.2	Educational Operating System	3
2.3	Questionnaire Survey – Implementation	4
2.4	Questionnaire Survey – Results.....	7
3	PREPARATION.....	13
3.1	Project Hosting and Source Control.....	13
3.2	Development and Testing Environment.....	15
3.3	Project Definitions.....	16
4	BRANDING	17
4.1	Mascot Character.....	17
4.2	Advertising.....	20
4.3	Copyright and Licensing	23
5	THEORETICAL BACKGROUND.....	24
5.1	Evolution of Operating Systems – Prehistory	24
5.2	Evolution of Operating Systems – Nowadays.....	25
5.3	Reflection of Operating System Concepts	26
5.4	Quick Introduction to Cosmos	27
5.5	Overview of the Booting Process	28
5.6	Basics of Command Shell	29
5.7	File Systems Explained	31
5.8	Artificial Intelligence – Virtual Personal Assistant.....	33
5.9	Artificial Intelligence – Implementation Methods	34
6	INITIAL PLANNING.....	35
6.1	Educational Process – Learning Flow	35
6.2	Educational Process – Study Tools	36
6.3	Command Line Interface.....	37
6.4	DimFS – Memory Based Pseudo File System.....	38
6.5	DimSH – Intelligent Command Shell	39
6.6	DimSH – Command Evaluation.....	40
6.7	Shell Scripting and Artificial Intelligence Tutor	41

7	ENVIRONMENT SETUP.....	42
7.1	Hosting the Project on CodePlex	42
7.2	Installing IDE and Supporting Tools.....	43
7.3	Creating the Visual Studio Solution.....	44
7.4	Configuring the Source Control	45
8	IMPLEMENTATION.....	46
8.1	DimFS – The Concept of Nodes	46
8.2	DimFS – Permissions	47
8.3	DimFS – Attributes	48
8.4	DimFS – Strict Paths	49
8.5	DimSH – Interactive Environment.....	50
8.6	DimSH – Command Journaling	51
8.7	DimSH – Input Processing	52
8.8	DimSH – Development Suite.....	53
8.9	Applications and Shell Commands	56
9	PLUGS.....	57
9.1	Mathematical Functions	57
9.2	Collections.....	58
9.3	Char and DateTime	59
9.4	Strings.....	60
10	TESTING.....	61
11	CONCLUSION.....	62
	BIBLIOGRAPHY.....	63
	APPENDIX	64

CODE REFERENCES

H Scripting Language's Engine:

<https://dimos.codeplex.com/SourceControl/latest#DimOS/H/Engine.cs>

Main Shell Class:

<https://dimos.codeplex.com/SourceControl/latest#DimOS/DimSH/Shell.cs>

Implementation of Virtual Text Screens:

<https://dimos.codeplex.com/SourceControl/latest#DimOS/VTS/Screen.cs>

Brainfuck Interpreter:

<https://dimos.codeplex.com/SourceControl/latest#DimOS/Brainfuck/Interpreter.cs>

1 INTRODUCTION

An **Operating System (OS)** is a computer program, which is, after being initially loaded by a boot program, capable of managing hardware and software resources of the target computer system, providing them a way to communicate (Figure 1) through API and ABI interfaces and system calls. Operating System is an essential part of computing, because without it the computer would be simply useless and unable to operate. Nowadays people tend to forget its importance, having no clue of what happens behind the fancy buttons they press to complete one action or another, which leads to the serious decrease of computer literacy among students.

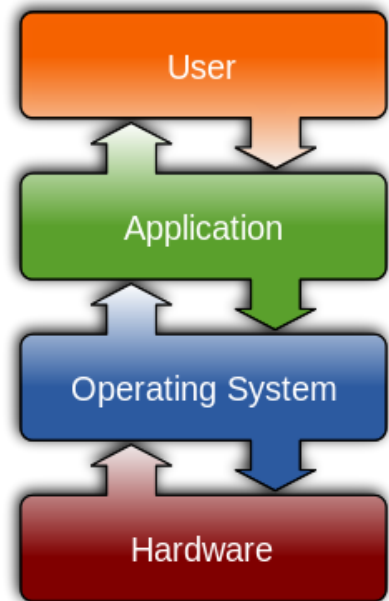


Figure 1. Interaction Flow.

For Information Technology students Operating Systems are taught as the separate course that extensively covers all the topics described in this project work. However, students tend not to have enough motivation and will to study it seriously, skipping the technical details provided. During my studies, I have encountered Information Technology students having no idea what the Operating Systems are needed for and how they actually function, even after successfully completing the dedicated university course. Moreover, they simply did not want to know this.

This gave me an inspiration to further investigate this issue and develop a solution that might help students to learn Operating Systems in a way that will not require them to cram material. Then I realized that the most interesting and exciting way to learn Operating Systems would be to learn them from an Operating System itself. However, Windows is too simplified to be used as a learning platform, meaning that most of the actions are tied to specific buttons and cannot be traced in terms of their internal operation. Although Linux pays more attention to user-OS interaction process, enforcing its users to think carefully before committing changes or producing some action, it happens to be too advanced for beginners. During the laboratory work, which involved working with the Unix Shell (Bash) I got feedbacks such as “I have no idea what I am doing” from some students. Understanding that there is clearly a need for the Operating System which is tailored for the educational needs and is, at the same time, capable of providing a decent study material and assessing students’ knowledge, I started this research.

2 MOTIVATION

The most important thing needed for every project is motivation. Before starting to work on the actual topic an assurance is needed that all the work done and efforts put into it will not end up being worthless and unusable. This section describes the methods used to obtain such an assurance, as well as the project's main aims and goals, together with the proof of concept.

2.1 Project Naming

Being one of the most important parts of branding, name of the project defines its overall look and feel, its direction and, in some cases, even its final outcome. In my case it will also reflect my personal ambitions and thoughts about an IT industry and Operating Systems in particular. With rough understanding of the concept and clear ideas in mind I have decided to name the Operating System "DimOS". This name happened to be perfectly fitting my initial intentions, as it combines multiple meanings that are also based on my own experience and expectations:

1. "Dimos" is exactly how I was calling my Informatics teacher during the classes in the middle and high school back then, so this name has some special moments of my life and positive memories associated with it, as my Informatics teacher was the very first person who introduced me to the world of programming and Information Technology.
2. The first programming language I learned was Visual Basic where the keyword "dim" is used to declare variables, which originates from BASIC where it was brought from FORTRAN and stands for "dimension" (because the keyword was used for defining the dimensions of an array back then), but, more importantly, in the old programming guides there was also an alternative interpretation of the "dim" keyword – "Declare In Memory", which perfectly fits my Operating System designed to work in the memory.
3. As an English word "dim" can mean something faint, unclear, obscure, something that is not clearly recalled, which consummately describes the current situation of beginner students, who are willing to learn Operating Systems, meaning that the subject may be unclear to them first, but this may change significantly if they use DimOS to study the OS fundamentals, just like any dim light has potential to become brighter and stronger.
4. The word "DIM" as an abbreviation can mean Diligence, Inspiration and Motivation – three main fundamental concepts of a successful project, work or study, provided that the diligence symbolizes persistence of students, inspiration – their desire or intention, and motivation – their main driving force or willingness towards the studying process.

2.2 Educational Operating System

Educational Operating Systems are usually created with the strong academic purpose in mind, mainly targeting Information Technology and Computer Literacy education in high schools or colleges and universities. Depending on their scope, different Educational Operating Systems can either contain a specific simple learning environment which is intended to provide various knowledge assessment methods together with the study material, or consist of small parts and modules with logically structured and documented open-source code which demonstrates the basic working principle of Operating Systems. Two most noticeable examples of Educational Operating Systems are MINIX (Figure 2) and Edubuntu (Figure 3). Ubuntu Education Edition (Edubuntu) applies the first approach, providing special learning environment for classrooms. MINIX, on the contrary, utilizes the second approach which turns the Operating System itself into a complex learning instrument originally designed for educational and research purposes. DimOS will employ both of those methods in order to guarantee a better learning experience.

```

CPU 0 freq 3200 MHz
Wed Aug 25 11:43:42 GMT 2010
Setting /usr on cd is /dev/c0d2p2
/dev/c0d2p2 is read-only mounted on /usr
Starting services: random printer ipc.
Starting daemons: update syslogd.
Local packages (start): done.

Welcome to MINIX 3.1.8.

The system is now running and many commands work normally. To use MINIX
in a serious way, you need to install it to your hard disk, which you
can do by typing 'Setup' while logged in as root. Then just follow the
on-screen directions.

After setup is complete, type 'shutdown' and when the boot monitor
starts, boot your new system by following the instructions at the end of
setup. Keep the CD-ROM in the drive, login as root and type 'packman'
to begin installing the many software packages available. After you
have installed the packages, type 'xdm' to start X Windows if you have
installed it.

Minix Release 3 Version 1.8 (console)
minix login: _

```

Figure 2. Screenshot of MINIX's CLI.

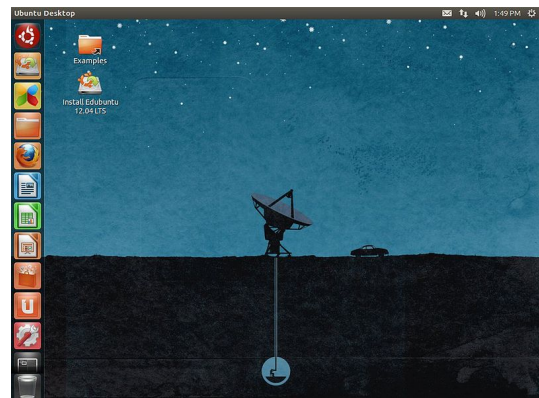


Figure 3. Screenshot of Edubuntu's GUI.

Furthermore, DimOS will combine the methods described above with a special functionality for teachers, allowing them to customize the feature set of the Operating System and change the study material according to their own needs. Such flexibility will lead to a better quality and effectiveness of education if utilized properly. DimOS will also include a supporting set of tools (bootable media or Virtual Machine creator, etc.) to provide the ease of deployment.

Being a student myself I feel very excited about this project. DimOS will pose an Educational Operating System created by a student for other students. This aim puts some restrictions and requirements on my coding style and my usual workflow. Nevertheless, it gives me the strong motivation to move forward. This project possesses a great opportunity for me to improve my project management and programming skills, as well as my knowledge of Operating Systems.

2.3 Questionnaire Survey – Implementation

To support my ambitions and expectations with the proof of concept and develop my ideas further I have decided to conduct a questionnaire survey. In order to be able to address the wider audience I made the survey available on three different languages – *Russian, English* and *Japanese*. Technical part of the survey was made using common web development tools and technologies (HTML, CSS, JavaScript, PHP) and is presented as a website hosted on the Virtual Private Server. All answers for the questionnaire are sent to a special E-mail address (hosted on the same server) in pre-defined format (Figure 4), which makes it easier to collect results of the survey and present them in a well-structured human-readable form. The survey itself is divided into three logically separated parts based on the type of the questions asked – General, Conceptual and Targeted. The flow control between these parts is handled by PHP script (Figure 5) in such a way that the respondent will be unable to change the answers after moving to the next part. In addition, third part can be accessed only with the positive answer to a question presented in the second part, which guarantees the validity of the survey results. The whole questionnaire consists of 10 questions (Figure 6), but those respondents who gave a negative answer to the fifth one will be presented with only a half of them, as stated above.

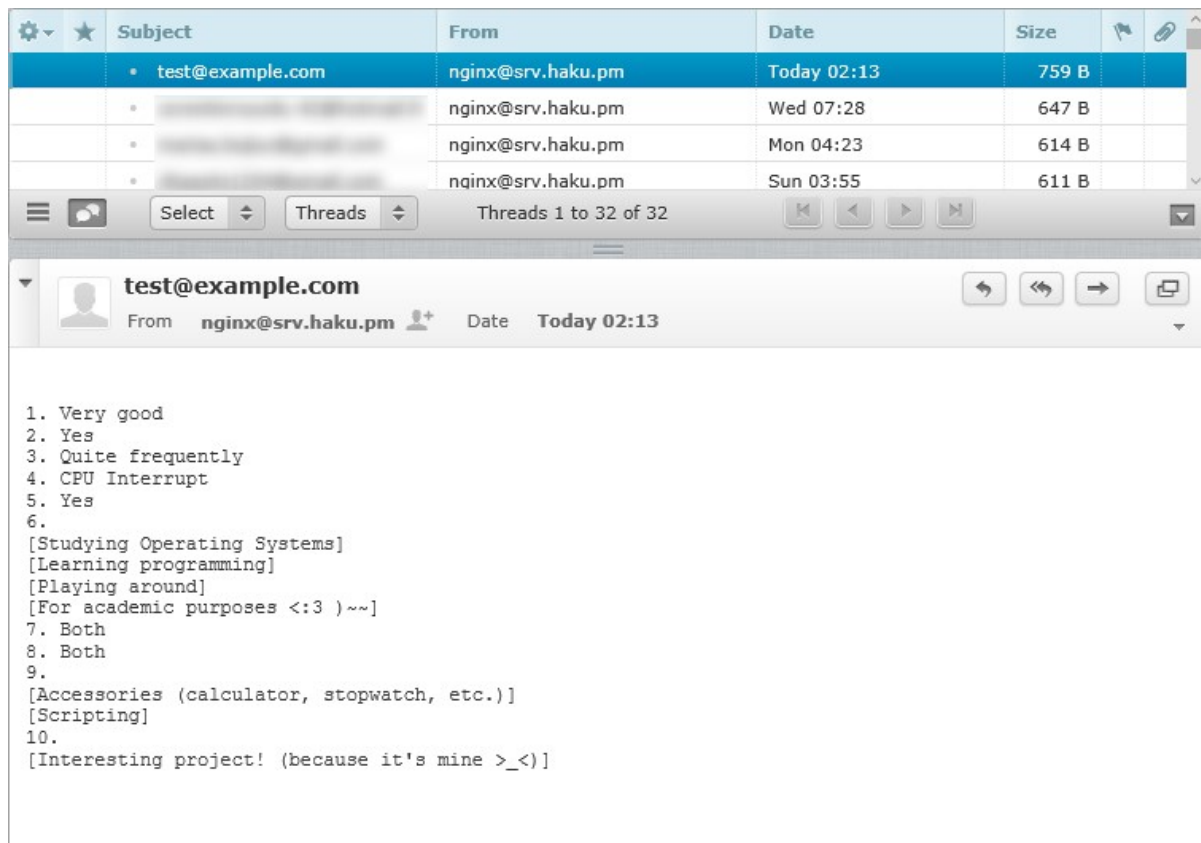


Figure 4. Format of Automatically Generated E-mail Message.


```

1 <?php
2     session_start(); //Initialize a session to transfer data between different parts.
3
4     if (!$_POST) part_1(); //If the form has not been submitted yet, output the 1st part.
5
6     else if ($_POST["5"] == null) part_2(); //If the form was submitted once, output 2nd part.
7
8     else if (($_POST["5"] == "Yes" //If the answer to the 5th question is "Yes"
9             || $_POST["5"] == "Maybe") //or "Maybe"
10            && $_POST["7"] == null) part_3(); //and the form was submitted twice, output the 3rd part.
11
12     else send(); //Otherwise generate and send the resulting message.
13
14     function part_1(){...} //Function that outputs the 1st part.
15     function part_2(){...} //Function that outputs the 2nd part.
16     function part_3(){...} //Function that outputs the 3rd part.
17     function send(){...} //Function that send a message and outputs the gratitude.
18 ?>

```

Figure 5. Flow Control of the Survey (PHP Script).

1. How would you describe your knowledge of Operating Systems?

Great
 Very good
 Good
 Average
 Bad

2. Would you like to improve your knowledge?

Yes
 No

3. How frequently do you use Command Shell (Command Prompt, Unix Shell, etc.)?

Quite frequently
 Sometimes
 Only when there is no other way
 I don't use it at all

4. What is INT 21H or 0x21?

Kernel
 Bootloader
 CPU Interrupt
 BIOS
 I don't know

5. Would you use such a system (if you were an IT student)?

Yes
 No
 Maybe

6. What would you use DimOS for?

Studying Operating Systems
 Learning programming
 Playing around
 Other (please specify)

7. Where would you use DimOS?

At home
 In the classroom
 Both

8. In which execution environment would you use DimOS?

In Virtual Machine
 On the real hardware
 Both

9. What kind of additional features would you want DimOS to have?

Games
 Accessories (calculator, stopwatch, etc.)
 Scripting
 Other (please specify)

10. Please write any ideas or feedback (optional):

Figure 6. Questions of the Survey (Presented on Website).

General part (questions 1 to 4) is used to gather general information about the respondent and includes two control questions (1 and 4), the main purpose of which is to identify whether the respondent is answering honestly, taking the whole survey seriously. These control questions can also show the level of respondent's self-confidence. Conceptual part consists of the small one-page description of DimOS supported by a short showcase video (hosted and embedded from YouTube) and one question (5), which checks if the respondent is interested in DimOS. Targeted part can only be viewed after answering "Yes" or "Maybe" for the question 5, as it contains questions (6 to 10) about the concept and its future development concretely, making this part the most valuable and important in the survey. When all the questions are answered the PHP script generates an E-mail message with survey results and displays a gratitude page.

The completion flow of the survey (Figure 7) is quite straightforward and linear, which makes it user-friendly and allows respondents of all ages with different level of computer literacy to take part in the questionnaire. Each part of it contains “Reset” and “Next” or “Send” buttons.

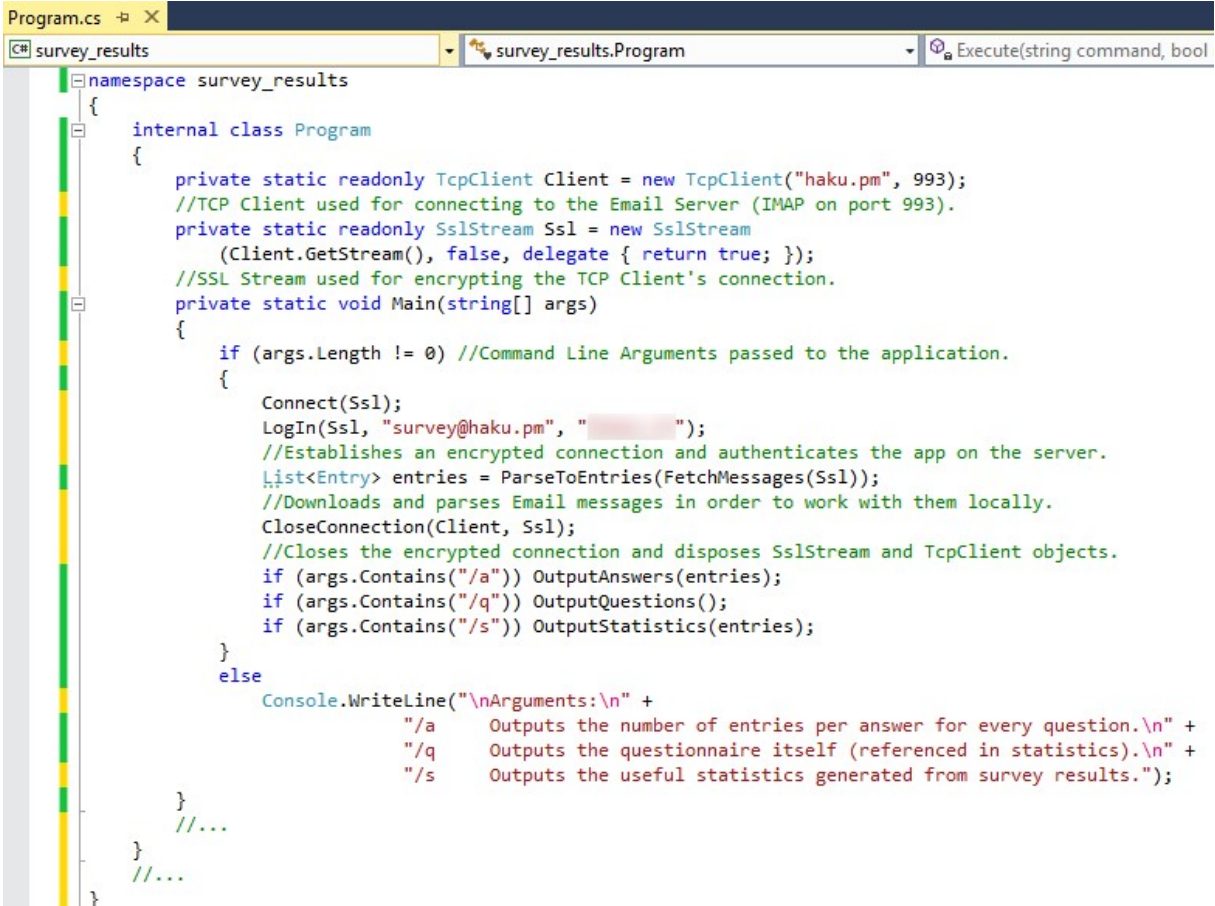


Figure 7. Completion Flow of the Survey.

2.4 Questionnaire Survey – Results

This questionnaire was targeting university students of various degree programs, as well as the high school and college students. People of different nationalities from multiple international universities took part in the survey during the one month period. In total, 50 unique entries of respondents' answers were collected, which allowed me to statistically analyze the concept of this project and make significant improvements to its main definitions and objectives. Results of the survey can also serve as the proof of concept, clearly showing its relevance by the level of students' interest in the project topic and their desire to study the Operating Systems better.

A small C# Console Application (Figure 8) was used to generate and show the statistical data out of the survey results. This application uses TCP sockets to send IMAP commands directly to the E-mail server. All messages in Inbox are downloaded from the E-mail server and then parsed to Entries (Figure 9), which can later be easily queried against using LINQ. Data can be represented either as the number of entries per answer for every question or in a form of useful statistic statements, where percentages are calculated based on specific conditions (Figure 10).



```

Program.cs  x
C# survey_results  survey_results.Program  Execute(string command, bool
namespace survey_results
{
    internal class Program
    {
        private static readonly TcpClient Client = new TcpClient("haku.pm", 993);
        //TCP Client used for connecting to the Email Server (IMAP on port 993).
        private static readonly SslStream Ssl = new SslStream
            (Client.GetStream(), false, delegate { return true; });
        //SSL Stream used for encrypting the TCP Client's connection.
        private static void Main(string[] args)
        {
            if (args.Length != 0) //Command Line Arguments passed to the application.
            {
                Connect(Ssl);
                LogIn(Ssl, "survey@haku.pm", " ");
                //Establishes an encrypted connection and authenticates the app on the server.
                List<Entry> entries = ParseToEntries(FetchMessages(Ssl));
                //Downloads and parses Email messages in order to work with them locally.
                CloseConnection(Client, Ssl);
                //Closes the encrypted connection and disposes SslStream and TcpClient objects.
                if (args.Contains("/a")) OutputAnswers(entries);
                if (args.Contains("/q")) OutputQuestions();
                if (args.Contains("/s")) OutputStatistics(entries);
            }
            else
                Console.WriteLine("\nArguments:\n" +
                    "/a  Outputs the number of entries per answer for every question.\n" +
                    "/q  Outputs the questionnaire itself (referenced in statistics).\n" +
                    "/s  Outputs the useful statistics generated from survey results.");
            //...
        }
    }
    //...
}

```

Figure 8. C# Console Application – Generating Statistical Data out of Survey Results.

```

Program.cs  X
C# survey_results  survey_results.StringExt  Truncate(string value, int maxLength)

class Message
{
    public string Topic { get; set; } //Stores the message's topic (respondent's Email).
    public string Body { get; set; } //Stores the message's body (questionnaire answers).

    public Message(string topic, string body) //Basic parametrized constructor for Message class.
    {
        Topic = topic;
        Body = body;
    }
}
//Class for storing the Email messages parsed directly from the SSL Stream in a simple textual form.

class Entry
{
    public string Email { get; set; } //Stores the respondent's Email (no validation).
    public string Q1 { get; set; } //Stores the respondent's answer for Question 1.
    public string Q2 { get; set; } //Stores the respondent's answer for Question 2.
    public string Q3 { get; set; } //Stores the respondent's answer for Question 3.
    public string Q4 { get; set; } //Stores the respondent's answer for Question 4.
    public string Q5 { get; set; } //Stores the respondent's answer for Question 5.
    public List<String> Q6 = new List<string>(); //Stores the respondent's answer for Question 6.
    public string Q7 { get; set; } //Stores the respondent's answer for Question 7.
    public string Q8 { get; set; } //Stores the respondent's answer for Question 8.
    public List<String> Q9 = new List<string>(); //Stores the respondent's answer for Question 9.
    public string Q10 { get; set; } //Stores the respondent's answer for Question 10.
}
//Class for storing the survey Entries (to be queried against using LINQ) parsed from the Messages.

```

Figure 9. Classes for Storing and Processing Survey Results as Structured Data.

```

Administrator: Command Prompt
x:\VSPROJECTS\survey_results\survey_results\bin\Debug>survey_results
Arguments:
/a      Outputs the number of entries per answer for every question.
/q      Outputs the questionnaire itself (referenced in statistics).
/s      Outputs the useful statistics generated from survey results.

x:\VSPROJECTS\survey_results\survey_results\bin\Debug>survey_results /q /s
1. How would you describe your knowledge of Operating Systems?
2. Would you like to improve your knowledge?
3. How frequently do you use Command Shell (Command Prompt, Unix Shell, etc)?
4. What is INT 21H or 0x21?
5. Would you use such a system (if you were an IT student)?
6. What would you use DimOS for?
7. Where would you use DimOS?
8. In which execution environment would you use DimOS?
9. What kind of additional features would you want DimOS to have?
10. Please write any ideas or feedback (optional).

Answered "Great" or "Very good" to Q1 and "I don't know" to Q4: 6% | 30%
Answered "Great", "Very good" or "Good" to Q1, but wrong to Q4: 8% | 18.1%
Answered correctly to Q4 with answers "Average" or "Bad" to Q1: 2% | 3.57%
Answered "Average" or "Bad" to Q1 and then answered "No" to Q2: 12% | 21.4%
Answered "Great" or "Very good" to Q1 and "Don't use it" to Q3: 2% | 10%
Answered "Great" or "Very good" to Q1 & "Yes" or "Maybe" to Q5: 16% | 80%
Answered "Average" or "Bad" to Q1 with "Yes" or "Maybe" for Q5: 50% | 89.2%
Answered "No" to Q2, but then answered "Yes" or "Maybe" for Q5: 10% | 62.5%
Answered "Study Operating Systems" / "Learn programming" to Q6: 60% | 66.6%
Answered "Study Operating Systems" & "Learn programming" to Q6: 10% | 11.1%

```

Figure 10. Useful Statistic Statements Generated by Application.

Upon closer inspection of generated statistics the following justified facts can be discovered:

1. Most of the students are willing to improve their knowledge of the Operating Systems.
2. Most of the students want or would probably use DimOS to study Operating Systems.
3. Students would equally want to use DimOS at home and in the classroom (Figure 11).
4. Many of the students would use DimOS in Virtual Machine and on the real hardware.
5. Students would equally want to exploit Games, Accessories and Scripting (Figure 12).
6. Students would use DimOS mostly for studying Operating Systems or playing around.

```
7. Where would you use DimOS?
   At home: 15 | In the classroom: 15 | Both: 15
```

Figure 11. Number of Survey Entries per Answer for Question 7.

```
9. What kind of additional features would you want DimOS to have?
   Games: 20 | Accessories: 22 | Scripting: 20
```

Figure 12. Number of Survey Entries per Answer for Question 9.

The distribution of answers for question 5 presented as a pie chart (Figure 13) clearly defines the validity of my concept. There is no strong interest in it yet, but people want to try DimOS.

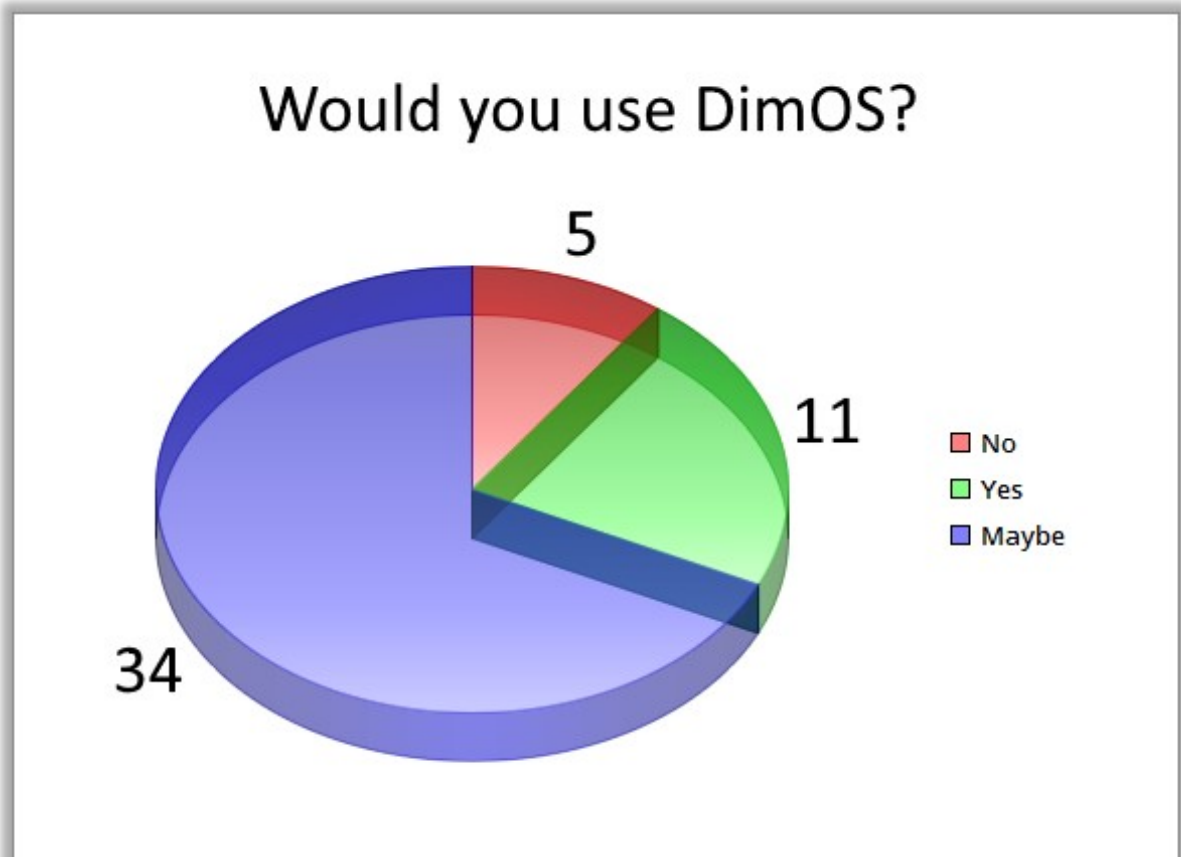


Figure 13. Distribution of Answers for Question 5.

Answers collected for the question 6, in turn, show that many people are equally interested in both studying the Operating Systems and playing around. Moreover, only 16% of respondents (or 17.7% of those who answered “Yes” or “Maybe” to the question 5) marked both of these answers, which means that there are people who want or would like to use DimOS without a particular purpose. Such people might be coming from the areas not related to IT, so DimOS should not only take an education approach, but also has to have features that can make an OS interesting and enjoyable. Some people would like to use DimOS for learning programming as well, therefore my code has to fulfill the fundamental coding tone requirements and follow the best coding practices and guidelines (Figure 15). The bar diagram of answers to the question 6 (Figure 14) illustrates the level of students’ concernment about different purposes of DimOS.

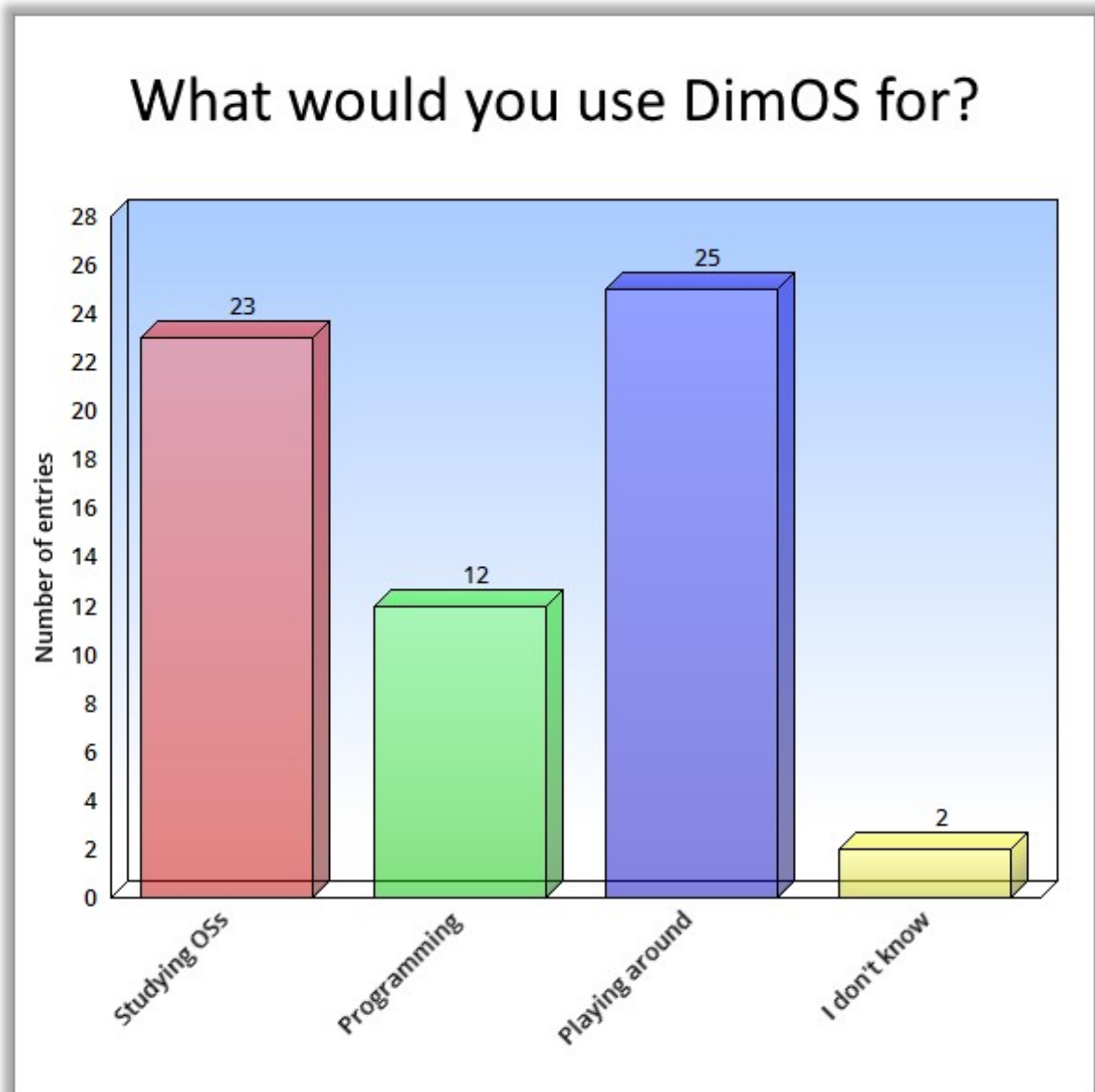


Figure 14. Distribution of Answers for Question 6.

Note: there was no “I don’t know” option, so it was inferred from the answers to question 10.

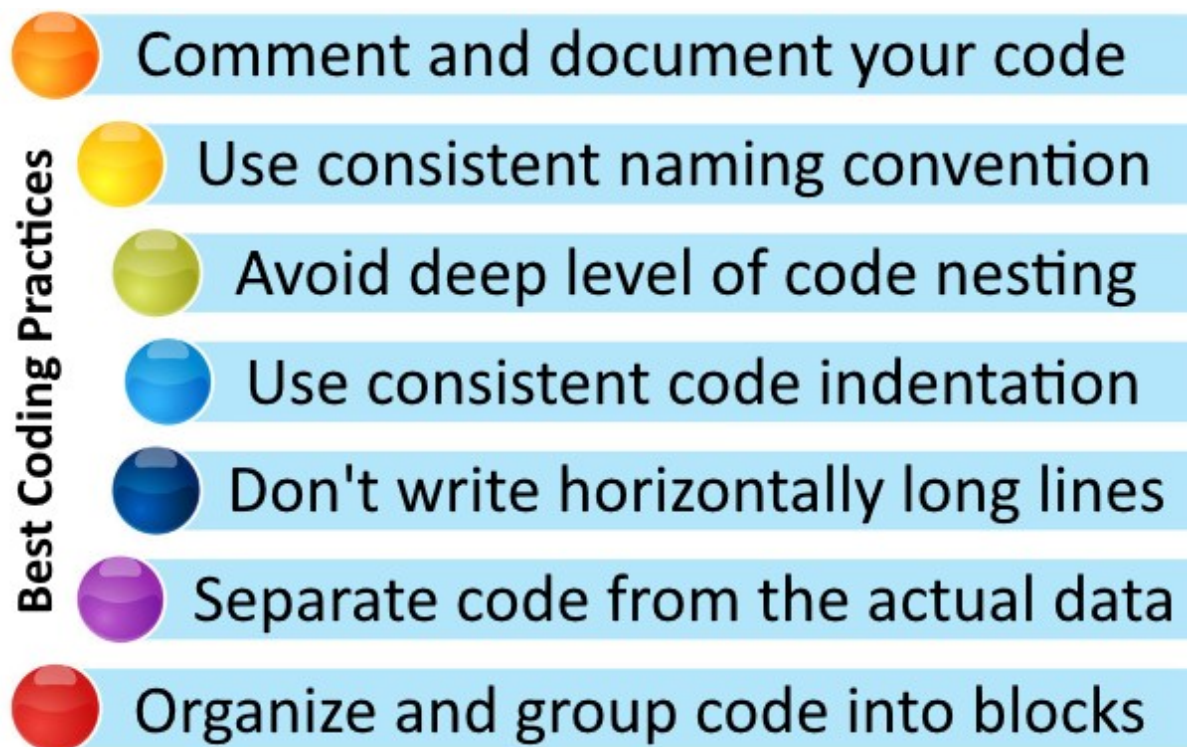


Figure 15. Best Coding Practices and Guidelines.

On the contrary, answers to the question 8 (Figure 16) show that the overwhelming number of students would use DimOS in the virtualized environment only, which means that main effort should be put on its stable operation in the Virtual Machines, rather than on the real hardware.

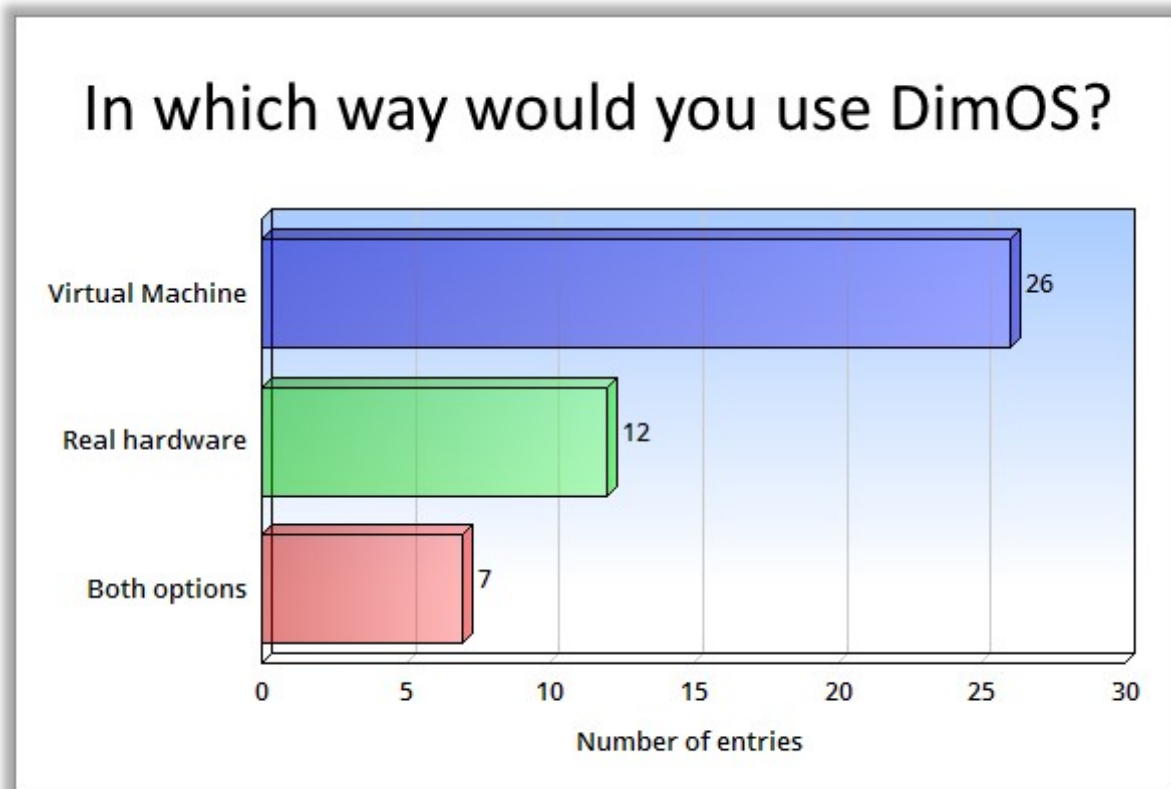


Figure 16. Distribution of Answers for Question 8.

Considering the huge diversity in the level of students' knowledge and experience with the Operating Systems (Figure 17), simple and user-friendly interface should be implemented in DimOS, making it possible for every student to understand how it works and how to use it. Additionally, an assistant system should be added to help users get used to the said interface.

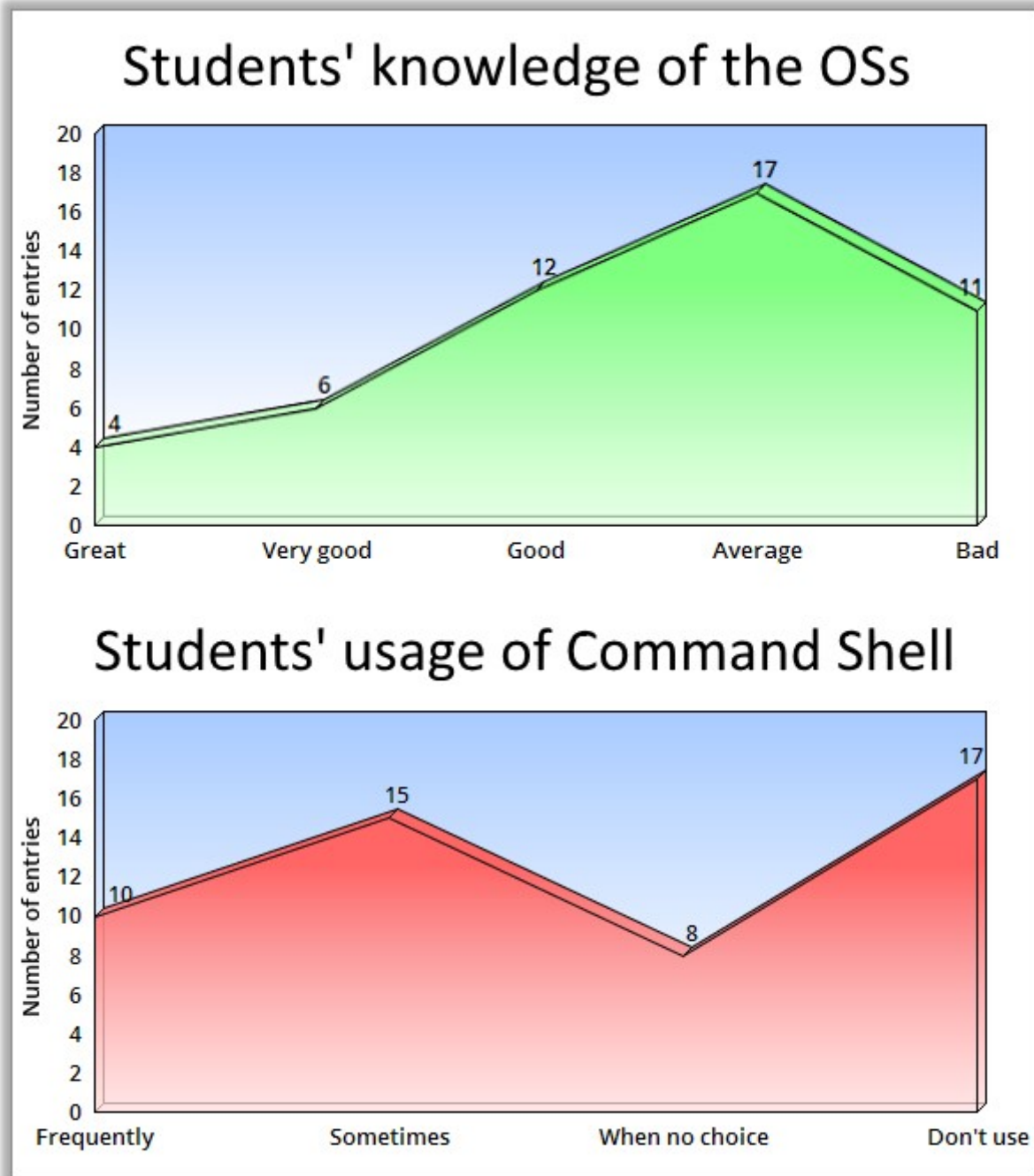


Figure 17. Distribution of Answers for Questions 1 and 3.

Overall results of this questionnaire survey gave me a clear understanding of what should be done in this project and how it should be done. All the conclusions made here will be defined more precisely in the next section. During this moderate research the concept of my Operating System was proved and polished, allowing me to move forward to the first step of this project.

3 PREPARATION

The first step of every project is, without any doubt, preparation. Fundamental considerations, such as the project's vector, project hosting and source control systems, a working and testing environment, skeleton (structure) of the project and outcome expectations will take place here.

3.1 Project Hosting and Source Control

Nowadays, when programming becomes more and more simplified through the use of various complex development environments and frameworks, the need for source control and project hosting solutions becomes vital. Such solutions usually tend to combine the hosting capability with SCM (Software Configuration Management) functionality. SCM provides establishment of baselines and revision control which ensures that all changes to the source are tracked and recorded properly. Hosting site is usually unlimited in terms of storage space given, but there may be certain limitations, such as public repositories only, shortened permission control and collaboration functionality or blocked access to some services/parts of the system, which can be partially or fully removed depending on the pricing plan. There are entirely free-of-charge project hosting solutions as well, but they may lack some of the features that paid ones utilize.

Source control systems can differ in internal structure, the way how they treat changes in data and the storage model applied to that data. Some systems treat data as the set of separate items (files/documents, folders/directories, etc.), which is valuable for tracking and structuring small changes, but turns out to be complicated with complex operations, such as *renaming*, *splitting* or *merging* entities. Other systems employ generic approach, meaning that all data is treated as a whole, which makes arrangement of simple changes less intuitive, but greatly simplifies this process with the complex operations. There are two storage models typically used by common source control systems: centralized and distributed (Figure 18). Centralized systems have only one repository (the data store), which is used by multiple developers to commit (check-in) and create local working copy (check-out). On the contrary, distributed systems have two or more repositories, which can either be completely independent, or connected to the main data store.



Figure 18. Source Control Storage Models.

There are a lot of project hosting solutions available on the market, each of them having its own set of features, advantages and disadvantages. In order to decide which one is better for this project, quick comparison (Figure 19) of the main project hosting solutions was created.

Name	Started	Prices	Ads	Features	Alexa
Assembla	2005	\$24/M (12U)	No	CR, BT, WH, W, TS	12547
Bitbucket	2008	Free (private 5U)	No	CR, BT, WH, W	834
CloudForge	2012	\$10/M (5U)	No	BT, WH, W	136654
CodePlex	2006	Free (only public)	No	BT, W, ML, F	2689
GitHub	2008	Free (public)	No	CR, BT, WH, W	60
GitLab	2011	Free	No	CR, BT, WH, W	4465
Launchpad	2004	Free	No	CR, BT, TS, ML	4985
SourceForge	1999	Free	Yes	CR, BT, WH, W, ML, F	320

M = Month, U = User, CR = Code Review, BT = Bug Tracking, WH = Web Hosting, W = Wiki, TS = Translation System, ML = Mailing List, F = Forum, Alexa = Alexa Rank. Alexa Rank shows the popularity of web resource, lower rank means higher traffic.

Figure 19. Comparison of Project Hosting Solutions.

As I am the only developer, this project does not require large feature set, meaning that in this particular case code review, web hosting, translation system and mailing list will be of no use.

On the other hand, features like bug tracking and wiki are the most important for me, making it much easier to track/fix bugs and write the technical documentation. Forum or a discussion board might be an interesting addition, because it simplifies the process of getting a feedback.

Taking into account the required future set, pricing and the overall look-and-feel I decided to use CodePlex. CodePlex is completely free to use with no subscription or registration charges. Together with the project title it is possible to define a sub-domain name, which will be given to the project, creating a unique project URL. The source control systems supported are: Team Foundation Server, Git and Mercurial. However, Git is preferred over Mercurial, because the interaction with the latter one appears to be slower and less reliable. CodePlex enforces users to publish the project (make the source code publically available) within one month from the date when it was created. Otherwise it will be deleted as seen on the control panel (Figure 20).



Figure 20. Project's Control Panel on CodePlex.

3.2 Development and Testing Environment

When the main course of the project is well-defined, one of the most important things to think about is the actual development environment. In most cases, development/programming tools, various frameworks and libraries, debugging/testing platform determine the overall success of the project. In this project the main emphasis will be put on the coding suite and testing tools.

The whole project is oriented exclusively towards Microsoft technologies, so the work will be held in Windows 10, where these technologies are well-integrated, thus the main development environment chosen is Visual Studio Ultimate 2013, which is a time-proven solution with the full support of .NET Framework 4.5 and useful extensions, such as JetBrains ReSharper (R#).

Cosmos User-kit (108477) is an Operating System development kit, which is fully integrated with the Visual Studio in the form of Project Templates, providing seamless compilation and debugging interface through IL2CPU and VMware VIX API. Therefore, it was chosen to be the base of my Operating System (together with the Syslinux bootloader) and part of its core.

VMware Player will serve as a primary testing environment during the development process. However, the final product will be widely evaluated against all major virtualization software (including the Microsoft Hyper-V, Oracle VM VirtualBox and Linux KVM) under different platforms in order to ensure its stable and continuous operation in any required environment.

In contrast to an object-oriented approach, which will be utilized in the development process, a low-level analysis of the generated CPU opcodes (Assembly instructions) will be applied to guarantee smooth, efficient and steady execution flow. For the very same reason, all processor registers and the state of the stack will be monitored accordingly during the debugging phase.

The Operating System will be tested on the real hardware as well. All major AMD and Intel CPU models will be tried in order to fix possible compatibility issues and guarantee that the generated instruction set is fully compliant with the coding standard of the **x86** architecture.

Rufus 2.9 Portable (an advanced USB media formatting utility) will be used to assist testing on the real hardware by providing fast and reliable bootable USB drive creation capabilities. Serial cable can be used if the hardware debugging is needed, but this is not likely to happen.

3.3 Project Definitions

Now, when all fundamental considerations are made, the information provided earlier should be summarized. For that reason, all significant definitions of the project will be listed below.

Educational Operating System

Project's Name: *DimOS*

Kernel Type: *Monolithic*

User Interface: *CLI/TUI*

Target Architecture: *x86*

OS Bootloader: *Syslinux*

Model of Source: *Open*

License of Source: *BSD*

Language of Source: *C#*

Targeted Users: *Students*



Figure 21. Official Logo of DimOS.

Note: the official logo (Figure 21) was designed exclusively for the DimOS by Zhang Yancan.

Hardware and Software Used

Development: *Visual Studio Ultimate 2013, Cosmos User-kit (108477)*

Debugging: *VMware Player, VMware VIX API, serial cable (possibly)*

VM testing: *Microsoft Hyper-V, Oracle VM VirtualBox, Linux KVM*

Hardware testing: *major AMD and Intel CPU models, x86 architecture*

Hardware testing assistance: *Rufus (ver. 2.9) Portable, the USB drive*

Main Objectives

1. Create a fully operational Educational Operating System for the Intel **x86** architecture.
2. Design and build a simple and user-friendly interactive command-line interface (CLI).
3. Design and build several User Level applications introducing basic concepts of an OS.
4. Demonstrate the use of C# with .NET Framework in a limited execution environment.
5. Develop the shell scripting engine and scripting language for command-line interface.

Outcome Expectations

The outcome of this project should be applicable both theoretically and practically in various areas of Information Technology, serving for different educational purposes and needs as well.

4 BRANDING

Based on results of a questionnaire survey I can conclude that people's interest in this project is a bit low. Even though it can be mostly explained by non-IT affiliation of respondents, this is a nice opportunity for me to think about project's branding. In this section all methods used to build up the image of my Operating System and promote it will be described and explained.

4.1 Mascot Character

We live in the century where interesting and appealing characters are used in different spheres of business and various public activities to represent and promote a certain party, entity or the product. Sometimes an already existing character can become the mascot, but most of the time original characters are created to serve that purpose. Such characters usually have personality.

In Operating Systems world those mascot characters are called OS-tans (-tan is an intentional mispronunciation of a Japanese honorific diminutive suffix -chan, which is generally used to address cute and attractive women, children, animals, etc.). Originating from Japanese imageboards as an Internet phenomenon, personifications of Microsoft Windows Operating Systems got quickly recognized by the Microsoft itself. Although anthropomorphized versions of OSs do exist for a long time already (various characters for DOS family and BSD distributions started to appear in 1990s and early 2000s), Microsoft began to utilize the concept officially only with Windows 7. Madobe (Figure 22) and Aizawa (Figure 23) families were created for advertising Microsoft's OSs and other products on Singaporean, Taiwanese and mainly Japanese markets.



Saseko (unofficial)
Windows XP



Figure 22. Madobe Sisters – Microsoft's Operating Systems.



Figure 23. Aizawa Sisters – Microsoft's Software Products.

With the very same idea in mind, I wanted to have something unique that could represent, and increase the interest in my Operating System. Thereafter I found an artist capable of doing that kind of commission and wrote a technical task (Figure 24) for original anime character design.

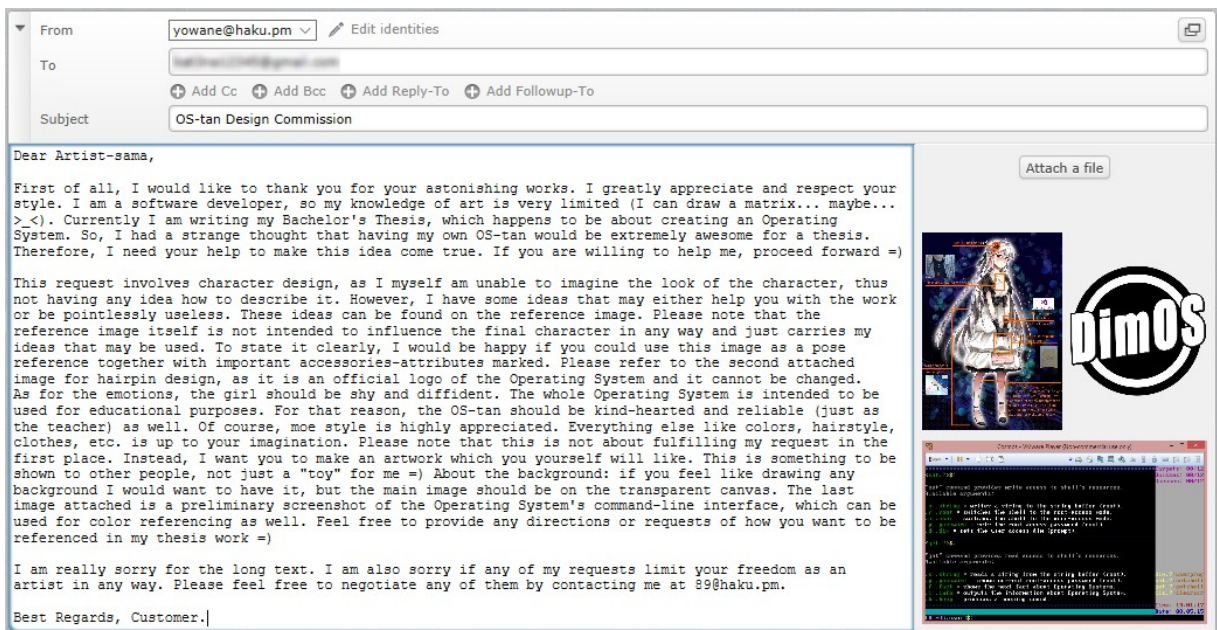
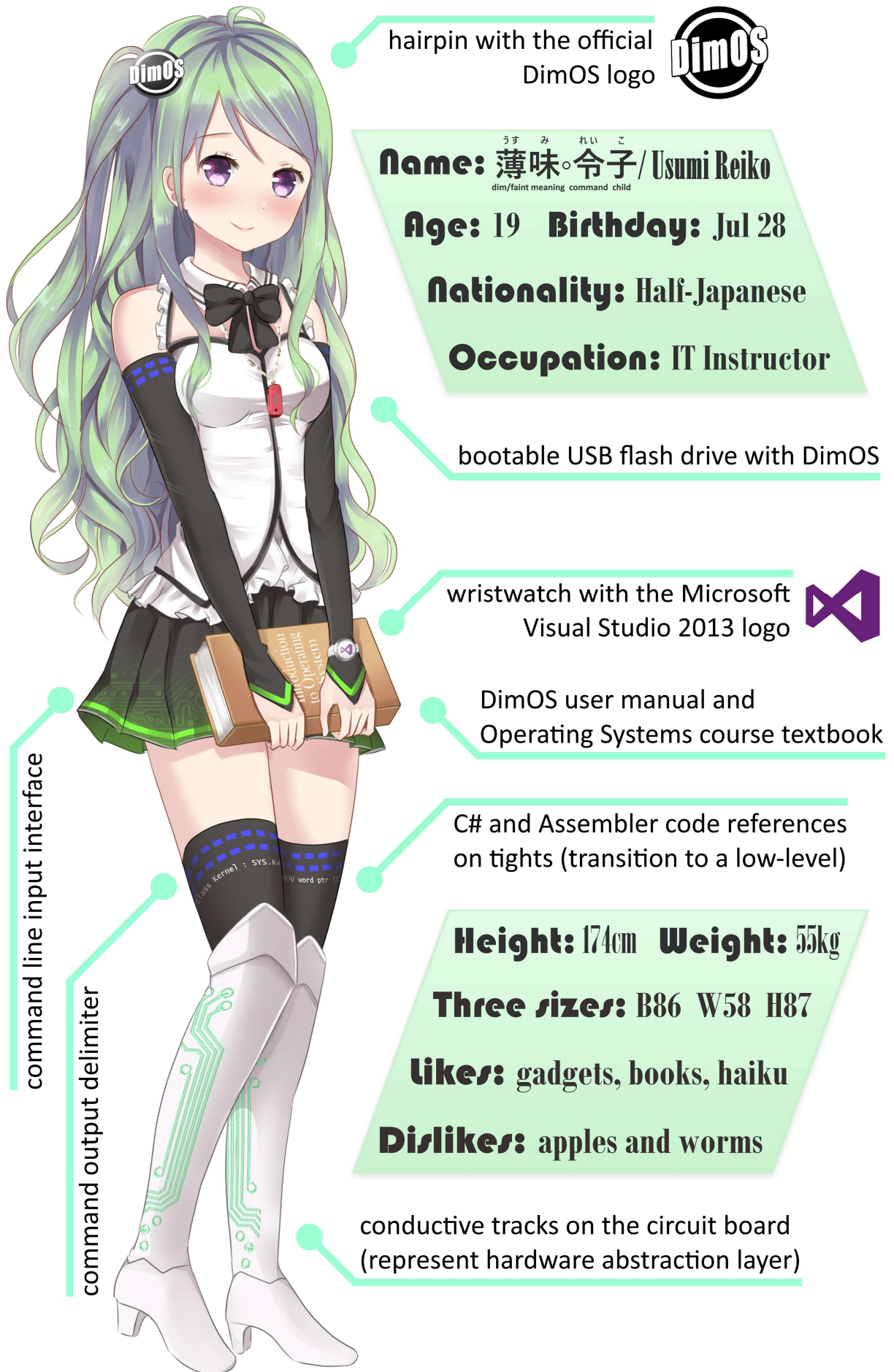


Figure 24. Technical Task for OS-tan Design Commission.

The whole process took one week and involved close collaboration with the artist. Every step was thoroughly discussed and negotiated in order to make right decisions. Furthermore, I was able to easily track the progress through WIPs (Work In Progress reports) constantly received from the artist. The work was handled in a very professional way, allowing me to concentrate on the actual personality of the OS-tan, entrusting her appearance to the artist. As a result, all the conditions were perfectly met and all my requests were fulfilled, which led to the creation of a truly astonishing piece of art. By this delicate work my own OS-tan (Figure 25) was born.



hairpin with the official DimOS logo



Name: ^{うす み れい こ} 薄味。令子 / Usumi Reiko
dim/faint meaning command child

Age: 19 **Birthday:** Jul 28

Nationality: Half-Japanese

Occupation: IT Instructor

bootable USB flash drive with DimOS

wristwatch with the Microsoft Visual Studio 2013 logo



DimOS user manual and Operating Systems course textbook

command line input interface

C# and Assembler code references on tights (transition to a low-level)

command output delimiter

Height: 174cm **Weight:** 55kg

Three sizes: B86 W58 H87

Likes: gadgets, books, haiku

Dislikes: apples and worms

conductive tracks on the circuit board (represent hardware abstraction layer)

Figure 25. DimOS-tan Character Reference.

4.2 Advertising

A mascot character on its own will not advertise my Operating System. For that reason, I need a clear idea of the character's proper usage. OS-tans are usually used together with the various forms of merchandise on different public events and advertising campaigns. However, as I am the only creator and developer of the DimOS, I took a simple and inexpensive approach to this issue and decided to create a multi-purpose multi-functional business card. The back side of a business card would contain a short description of the Operating System, its official web page at CodePlex and my contact E-mail address, whereas the front side can have an OS-tan image and the bootable USB flash drive (with DimOS itself) on it. Such card will serve as a product and its promotional merchandise at the same time. The overall design (Figure 26) of the actual card was made by me, with the chibi (special Japanese caricature style which assumes that the characters are drawn in a very exaggerated way, usually with the oversized heads and/or other deformed parts, making those characters look cute or resemble small children) version of the OS-tan created by the same artist, who is also referenced on the front side of the business card.

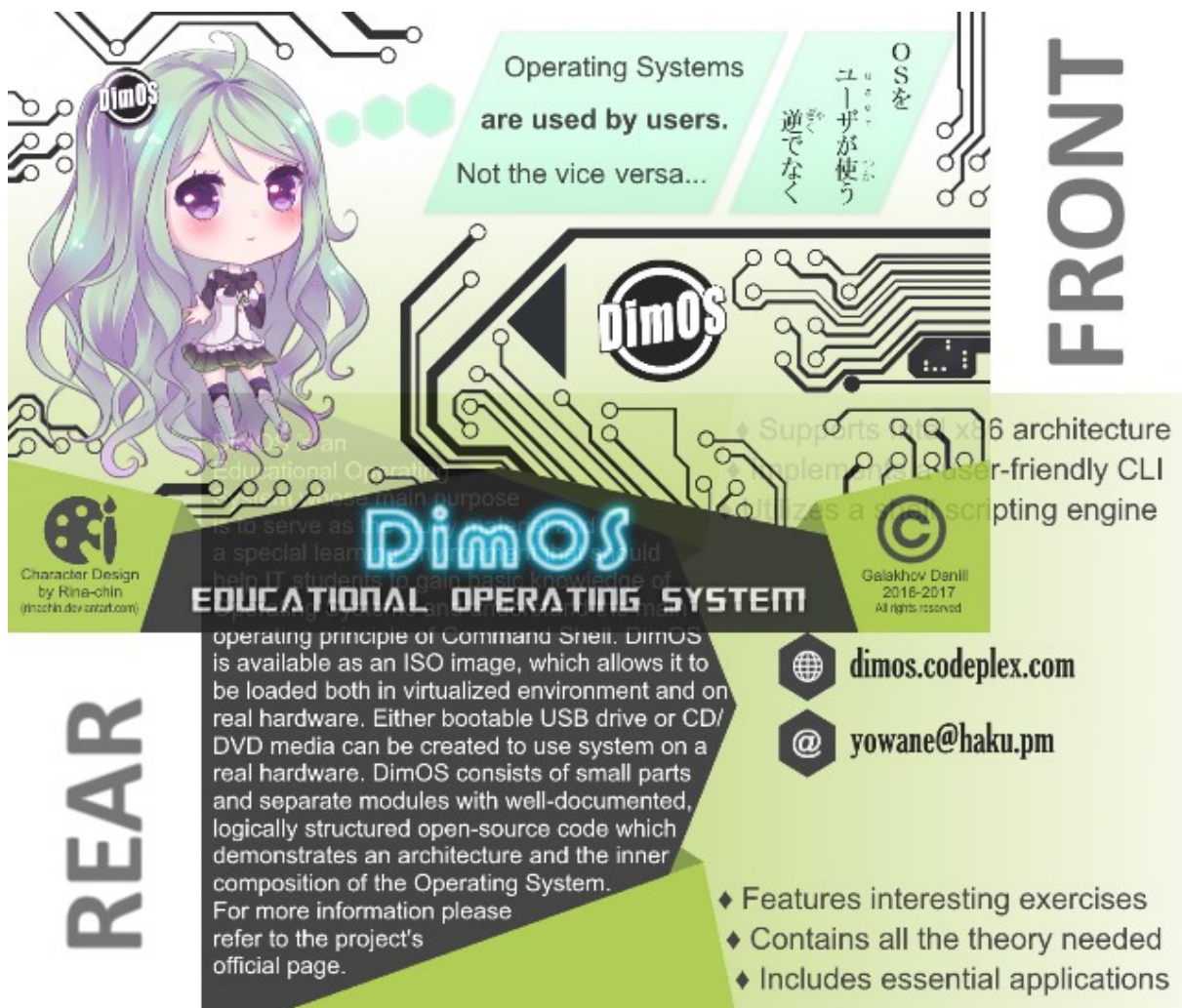


Figure 26. Business Card Design.

In addition to USB flash drive, an alternative removable media for the Operating System must always be available, as some of the old but potentially still usable computers do not have USB interfaces. Floppy disks are too old and have a very low capacity, which made them unusable for most of the current technological needs and purposes. On the contrary, CD and DVD disks have been widely used since 1996 and are still supported in modern laptops and workstations, which consequently makes them the best choice in my case. With the use of background made by the same artist (referenced here too), I designed a custom CD/DVD cover print (Figure 27).



Figure 27. CD/DVD Disk Design.

In order to demonstrate how both of the designs will look on the real physical products, I have looked through different printing services and ordered two low-cost custom prints: on a DVD (Figure 28), and a business card with the flash drive embedded in it (Figure 29). The products created are fully functional and are ready to be used for educational and promotional purposes.



Figure 28. Custom Printed DimOS DVD Disk.



Figure 29. Custom Printed DimOS USB Business Card.

4.3 Copyright and Licensing

Copyright is a legal “instrument” which protects the ownership of an intellectual property and grants an owner the exclusive right to use and distribute that property. Not to be confused with license, in software development copyright is used to uniquely specify the entity which has the ownership or in some cases authorship of a certain code, piece of software, product or a brand.

All original designs, artworks, mascot character and code created for this project are subject to copyright, thus contain an appropriate copyright notice, which includes the “Copyright” word, an ubiquitously recognized © symbol, years during which the product was released or updated and the copyright holder’s name with “All rights reserved” statement (not necessarily present).

On the other hand, license is usually a permissive document that gives others the rights to use, change/modify or redistribute the product when certain conditions are met. Software licenses are usually perpetual and require users to follow a set of rules. Common open-source licenses:

1. *GNU GPLv3 (General Public License Version 3)* – one of the most popular free/open licenses, which allows others to modify and redistribute the code, if the GPL’s terms and conditions are not violated, but the code should be opened under the same license.
2. *Apache 2.0* – one of the most recent permissive licenses designed with patent relations and “human readability” in mind but requires certain references to be attached to work.
3. *New BSD* – the most plain (along with MIT), consistent and permissive of all licenses.

I decided on the New BSD license for its simplicity and clear definition of the responsibilities. Consequently, LICENSE file with a textual notice (Figure 30) should be added to the project.

```
Copyright (c) 2016-2017, Galakhov Daniil
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided
that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and
the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and
the following disclaimer in the documentation and/or other materials provided with the distribution.

* Neither the name of the DimOS Project, nor the names of its contributors may be used to endorse or
promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR
ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

Figure 30. Contents of LICENSE.TXT.

5 THEORETICAL BACKGROUND

The whole process of creating an Operating System involves a lot of theoretical considerations to be made and many potential issues or limitations to be taken into account. This section will accordingly introduce the reader to the actual theoretical background needed to understand the rest of the report, as well as an educational material that will be used in the Operating System.

5.1 Evolution of Operating Systems – Prehistory

Back in the 1950s computer programs were written on punch cards (medium for storing the instructions of a program in the form of 0s and 1s represented by holes or their absence on a special paper card) and every program could have a full control over the system's resources. Because each program or even each version of one program were completely different (they had to be written from scratch), there was no support for software, so it made portability and compatibility impossible. This issue was later solved by creating code libraries but the actual programs still had to work with hardware directly, thus requiring all programs to be rewritten completely for different hardware. To solve this problem, specialists came up with idea of an interface between hardware and software. With such an abstraction layer there is no need for programs to take a full control over hardware, because they can communicate with it through a single unified, consolidated interface. This interface is now known as the Operating System.

The first Operating System known to exist was developed in 1956 for the IBM 704 computer system by General Motors and North American Aviation, and was designed for general access to the input/output and batch processing, hence the name "GM-NAA I/O". It also became the first OS capable of executing programs written on Assembly language. In 1959 the improved version of it was created by SHARE Inc. under the name of SHARE Operating System (SOS).

In 1970 the first UNIX came alive. Developed by Bell Laboratories (AT&T) and written in C programming language (created at Bell Labs as well) the system offered multitasking support and allowed multi-user. Possessing an interactive command shell, modular kernel and unified filesystem, UNIX has quickly spread across a variety of machine systems and became popular.

After the initial release of Microsoft Disk Operating System (MS-DOS) in August 1981, many GUIs for it started to appear, causing Microsoft to come up with a Windows 1.0 shell in 1985.

5.2 Evolution of Operating Systems – Nowadays

30 years of development turned Windows 1.0 into Windows 10 – the most popular Operating System for desktop computers with a steadily growing market share (as the users switch from Windows 7, which still has the highest market share among systems in the Windows family). Windows 10 features an interactive, responsive and visually attractive GUI (Figure 31) which appears to be one of the main reasons of its indisputable popularity on the desktop computers.



Figure 31. Screenshot of Windows 10's GUI.

Unlike the straight-forward and direct development path of Windows, original UNIX became a predecessor of numerous completely different (besides the modular architecture or so-called “UNIX-philosophy” that they all follow) Operating System distributions and kernels, the most noticeable of which are Linux, FreeBSD, OpenBSD, NetBSD, Solaris, HP-UX and MacOS/X.

Linux, in turn, became a base for hundreds of distributions, which are typically derived either from Debian, Slackware or Red Hat. Although not being popular on desktop systems, Linux has the greatest influence on mobile market (Android is based on Linux), server environment and the world of supercomputing. Linux kernel is also widely used in the embedded systems.

Although taking fundamentally different development and design approaches, both Windows and UNIX had a tremendous impact on computing and IT, turning it into what it is nowadays.

5.3 Reflection of Operating System Concepts

Kernel is the core of any Operating System; it provides programs with an access to hardware resources, implements memory management and program execution, works with file systems.

Memory Management is an essential process in the Operating System and is responsible for:

1. Dynamically allocating the memory for high priority processes and freeing it for reuse.
2. Providing paging (uses a secondary data storage) and virtual memory (implements the address translation, which maps all virtual addresses to the physical ones) capabilities, allowing programs to use more memory than it is physically available in the computer.
3. Controlling the Kernel's Read/Write operations and handling Memory Fragmentation.

Program Management, being closely related to Memory Management, is similarly used for:

1. Abstraction of programs by implementing different levels of access to OS's resources.
2. Handling requests of the programs for OS's resources, such as memory allocation, etc.
3. Preventing the programs from damaging the OS's data or overwriting other programs.

Multitasking is the ability of an OS to execute multiple processes concurrently, as it includes:

1. Capability of providing certain execution timeframes to many simultaneous processes.
2. Task State Management (special segment of memory holding information about tasks).

Multituser capability allows multiple-user concurrent access to OS and programs and ensures:

1. Separation and protection of user accounts and user-specific data, such as user settings.
2. Availability of specific roles or groups and clear management of system's permissions.

File System is a set of rules, regulations and components that describe how binary data should be represented in the Operating System and provide mechanisms for interacting with that data.

Bootloader is a small program, which gets loaded by the BIOS and executes either the Kernel of the Operating System or a Second-Stage Bootloader (has a simple UI and an actual loader).

Interrupt is a certain signal generated by an event and sent to the CPU in order to temporarily switch its execution flow to that high-priority event using an interrupt handler (function that is called by hardware interrupt request or software exception) which then resumes the execution.

5.4 Quick Introduction to Cosmos

Cosmos (C# Open-Source Managed Operating System) is an OS Construction Kit created by a former Microsoft Developer Chad Hower a.k.a. Kudzu. A small team of C# programmers and OS engineers, together with the Open-Source community, is responsible for development and improvement of User-kit and Dev-kit which, alongside with the logo (Figure 32), make up the Cosmos Project. Source code was released on CodePlex (now on GitHub) under BSD license.



Figure 32. Official Logo of Cosmos.

Cosmos has the microkernel structure (implements basic inter-process communication, virtual memory and scheduling), which acts as OS's basis. The kernel is split into four levels (Rings):

1. *Core (Ring 0)* – provides direct hardware access and is intended only for the code that requires special permissions and can only exist in the Ring 0, such as Assembly code.
2. *HAL (Ring 1)* – provides wrapping and encapsulation of functions that perform tasks, which involve direct interactions with hardware, and acts as a sub-level for the Ring 0, having no special permissions, as it is mostly intended for implementation of various hardware-specific drivers and any other code that needs to access hardware via Ring 0.
3. *System (Ring 2)* – provides higher level of functionality and exists specifically for the code that implements system functionality, such as the file systems or a network stack.
4. *User (Ring 4)* – provides an extensive functionality of all user and system applications via the .NET Framework's methods/functions wrapped into plugs, and OOP structures.

The whole Cosmos Project is built around the IL2CPU. IL2CPU is an AOT (Ahead-Of-Time) compiler, which takes the IL (Intermediate Language) byte code produced from the C# (or any other .NET compatible language) code by Visual Studio, and translates it to the CPU opcodes (Assembler). The output .ASM file is then assembled by NASM into an ELF (Executable and Linkable Format) object. Finally, tools like *ld.exe* and *objdump* are used for retrieval of object information, and *mkisofs.exe* is used to produce the final ISO image, which contains Syslinux bootloader as well. With the help of Visual Studio the compilation process becomes seamless.

5.5 Overview of the Booting Process

For an ordinary user the whole process happening between the press of a power button and the actual appearance of an Operating System's logo on the screen is nearly seamless. In practice, the booting process is a bit more complex. When the user presses the power button, two wires get connected, resulting in a current flowing through the motherboard to a PSU (Power Supply Unit), which, in turn, recognizes it as a signal to start supplying the power to the motherboard and connected devices. When a sufficient amount of power is provided to all the components, meaning that PSU has passed its self-tests and entered its normal operation mode, it sends the Power Good signal back to the motherboard, where it then reaches BIOS (Basic Input/Output System), which is stored either in ROM (Read-Only Memory) or in the flash chip (EEPROM).

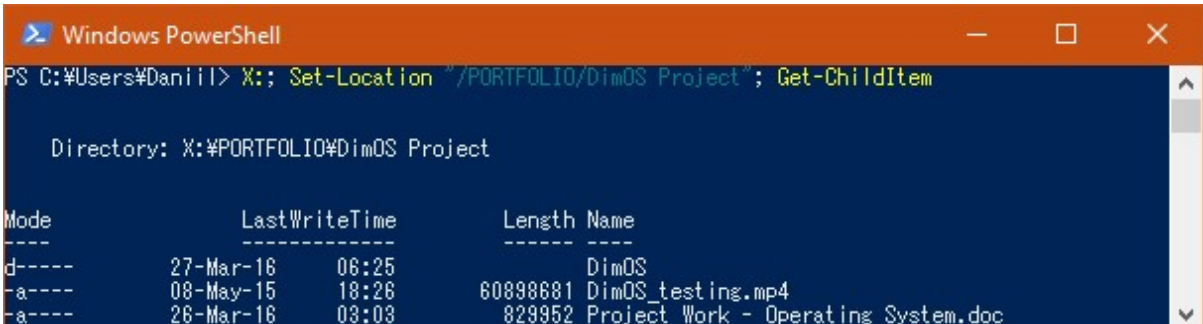
After receiving the Power Good signal, BIOS starts to initialize a POST (Power-On Self-Test) process, which ensures that enough power is being supplied to all devices, itemizes hardware and checks if the critical components (CPU and RAM) are operating properly. Depending on the BIOS, on some computer systems POST sequence shows its output on the monitor, while on the other systems errors are reported through the motherboard speaker/buzzer using special beep codes or via an I/O port 80 which requires special POST card connected to motherboard.

When the POST procedure successfully finishes, it gives control to the main pre-boot routines of the BIOS. Those routines include initialization of an Interrupt Vector Table (list of the most common interrupts stored in the first 1024 bytes of memory) and usage of default/pre-defined boot order to determine which mass-storage device should be assumed as bootable in the first place. BIOS then calls an INT 19h (0x19) interrupt to read the first sector of this device, trying to find the bootloader. In case of a failure BIOS tries the next available entry in the boot order.

When the bootloader is found, BIOS loads it into memory and passes the execution to the boot code, which initializes the memory and loads the Operating System's kernel or a second stage bootloader (GRUB, NTLDR/BOOTMGR, Syslinux, LILO, etc.). Second stage bootloaders are usually much more advanced as they tend to provide multi-boot functionality (presenting a list of different OSs that might be located on different bootable devices), different loading options (such as safe-mode or CLI-mode) and even various programs (like memtest86+ – the memory testing utility). Some second stage bootloaders include a GUI and contain games like snake or tetris. Once the actual Operating System is loaded, bootloaders are removed from the memory.

5.6 Basics of Command Shell

Today we tend to use Graphical User Interface for all daily tasks, as well as in the professional working environment. Even such complex activities as programming and web design can now be easily handled without the use of Command Line Interface, because many IDEs (Integrated Development Environments) are so powerful that there is a GUI available for almost any task. However, there are areas where command shell is not only an essential user interface, but also the most efficient, consistent and fast way of a human-machine interaction. Such areas include server technologies, embedded systems, administration and maintenance of computer systems, automation, networking and so on. Command line is also widely used in all UNIX-based OSs. Being a “blast from the past”, command shell remains one the most stable and common means of communication between applications and different parts of the OS as well. For example, in Windows many Installation Wizards and device driver packages rely on the Command Prompt for configuration of various system components, installation of services, cleanup of temporary files and directories, etc. For server environment and remote administration Command Prompt was quickly superseded by PowerShell (Figure 33), which is an extensible shell together with the corresponding scripting language and full access to COM (Component Object Model) and WMI (Windows Management Instrumentation). PowerShell is based on .NET Framework and is, therefore, cross-platform and open-source (released as “PowerShell Core”, by analogy with the .NET Core). Linux, FreeBSD and other POSIX-compliant OSs generally use Bourne again shell or shortly Bash (Figure 34), which is a powerful command processor and an easy-to-use scripting language that provides virtually full access to OS. Bash is available on Windows 10.



```

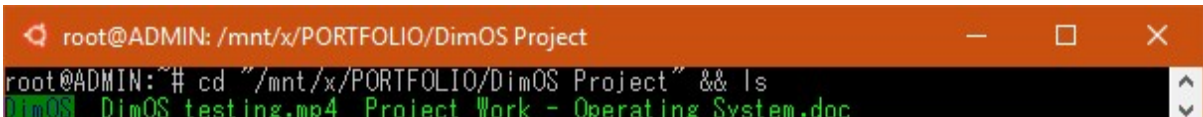
Windows PowerShell
PS C:\Users\Danil> X:: Set-Location "/PORTFOLIO/DimOS Project"; Get-ChildItem

Directory: X:\PORTFOLIO\DimOS Project

Mode                LastWriteTime         Length Name
----                -
d-----           27-Mar-16    06:25         DimOS
-a-----           08-May-15    18:26    60898681 DimOS_testing.mp4
-a-----           26-Mar-16    03:03    829952   Project Work - Operating System.doc

```

Figure 33. Setting Working Location and Listing Directory Contents – PowerShell.



```

root@ADMIN: /mnt/x/PORTFOLIO/DimOS Project
root@ADMIN:~# cd "/mnt/x/PORTFOLIO/DimOS Project" && ls
DimOS DimOS_testing.mp4 Project Work - Operating System.doc

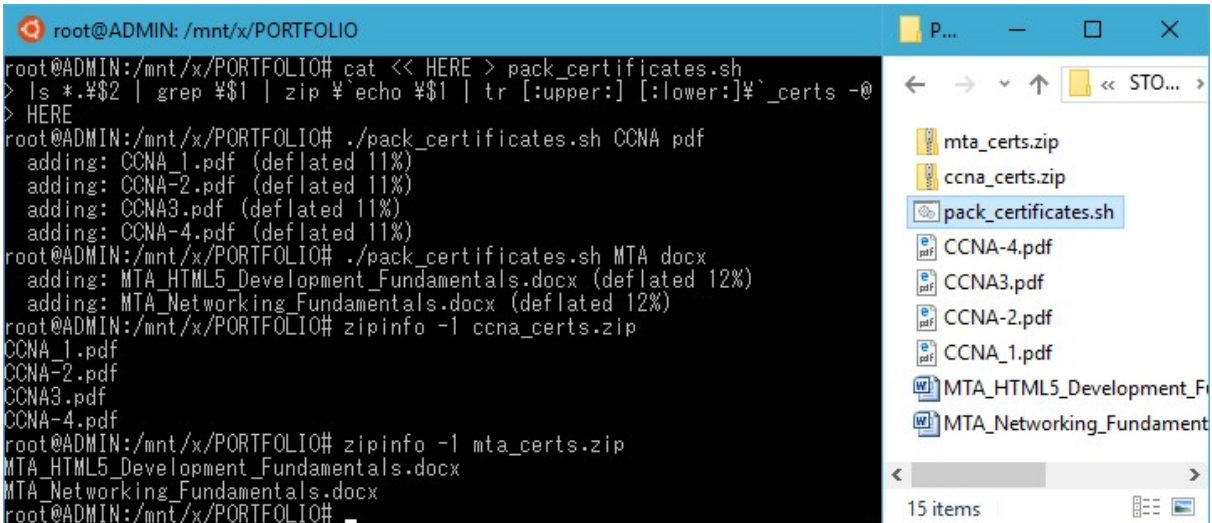
```

Figure 34. Setting Working Location and Listing Directory Contents – Bash.

Even though the major command shells are significantly different in scripting capabilities and interface interactivity, they all have a set of basic features in common. Those features include:

1. *Filename Globbing* – a process of dynamically generating directory and file names (or extensions) based on the wildcard matching, which implies a set of special symbols to be used instead of the usual characters (allowed in the filename) in the bulk operations.
2. *Command Piping* – usage of a special delimiter/separator character to type a sequence of commands represented as one command line entry where an output of the command serves as an input of the next command in the sequence, creating the software pipeline.
3. *Here Document/String* – an input redirection method, which uses a certain pre-defined keyword(s) passed to the shell at both the beginning and the end of a user input, stating that everything entered in between should be treated as plain text, instead of command.
4. *Command Substitution* – technique similar to the command piping, that provides more flexibility and control over the piping process, as the output of a command (or multiple commands) can be fed to the input of another command or serve as its option/flag, etc.
5. *Scripting Engine* – a built-in interpreter capable of executing the shell commands from the text file, recognizing and interpreting all keywords and syntax of the shell scripting language with support for variables and control structures (condition-testing, iteration).
6. *Command History* – a set of previously entered commands displayed to the user, or the substitution of current command line entry by the last command submitted to the shell, triggered with the press of a pre-defined shortcut (typically F7 or an UP ARROW key).

Demonstration of described functionality (Figure 35) shows the usage of here document input for creation of the shell script, which accepts 2 arguments and uses file globbing together with the pipelines and command substitution to archive the files with a given “type” and extension.



```

root@ADMIN: /mnt/x/PORTFOLIO
root@ADMIN:/mnt/x/PORTFOLIO# cat << HERE > pack_certificates.sh
> ls *.*$2 | grep $!$1 | zip $ echo $!$1 | tr [:upper:] [:lower:];#*_certs -@
> HERE
root@ADMIN:/mnt/x/PORTFOLIO# ./pack_certificates.sh CCNA pdf
adding: CCNA_1.pdf (deflated 11%)
adding: CCNA-2.pdf (deflated 11%)
adding: CCNA3.pdf (deflated 11%)
adding: CCNA-4.pdf (deflated 11%)
root@ADMIN:/mnt/x/PORTFOLIO# ./pack_certificates.sh MTA docx
adding: MTA_HTML5_Development_Fundamentals.docx (deflated 12%)
adding: MTA_Networking_Fundamentals.docx (deflated 12%)
root@ADMIN:/mnt/x/PORTFOLIO# zipinfo -l ccna_certs.zip
CCNA_1.pdf
CCNA-2.pdf
CCNA3.pdf
CCNA-4.pdf
root@ADMIN:/mnt/x/PORTFOLIO# zipinfo -l mta_certs.zip
MTA_HTML5_Development_Fundamentals.docx
MTA_Networking_Fundamentals.docx
root@ADMIN:/mnt/x/PORTFOLIO#

```

Figure 35. Demonstration of Basic Shell Functionality – Bash.

5.7 File Systems Explained

File system is a software facility that defines and controls the means, by which the data should be stored, structured and ordered on a certain readable and/or writable media. File systems are used to logically separate and/or isolate the information, making the data properly identifiable and accessible for both the user and an Operating System. Depending on actual storage media and the functionality required, file systems can greatly vary in flexibility, speed of data access, security features, capacity limits, overall complexity, internal structure, design limitations, etc.

However, regardless of their application, all file systems provide following base functionality:

1. Effectively tracking and reporting the available storage space left on the storage media.
2. Safely (with no occasional overwrites of the data) creating new directories and/or files.
3. Reading the data from existing files into memory (in non-memory-based file systems).
4. Updating and modifying data in the existing files, if those files are not write-protected.
5. Deleting directories and/or files (if these are not write-protected) from a storage media.

Additionally, besides CRUD (Create, Read, Update and Delete) operations listed above (2 to 5), most of the general-purpose file systems have the following features implemented as well:

1. Consistent file and directory naming/addressing scheme with a hierarchical structure.
2. Human-readable typization of files (such as the file extensions and related metadata).
3. Access permissions and advanced file attributes (executable, read-only, hidden, etc.).
4. File caching and read/write bufferization, that greatly reduces performance overhead.
5. Built-in support for advanced file operations, such as renaming, copying and moving.
6. Automatic encryption mechanisms (NTFS) and transaction logging/journaling (ext3).

For simplicity, file systems' internal structure can be logically divided into layers (Figure 36).

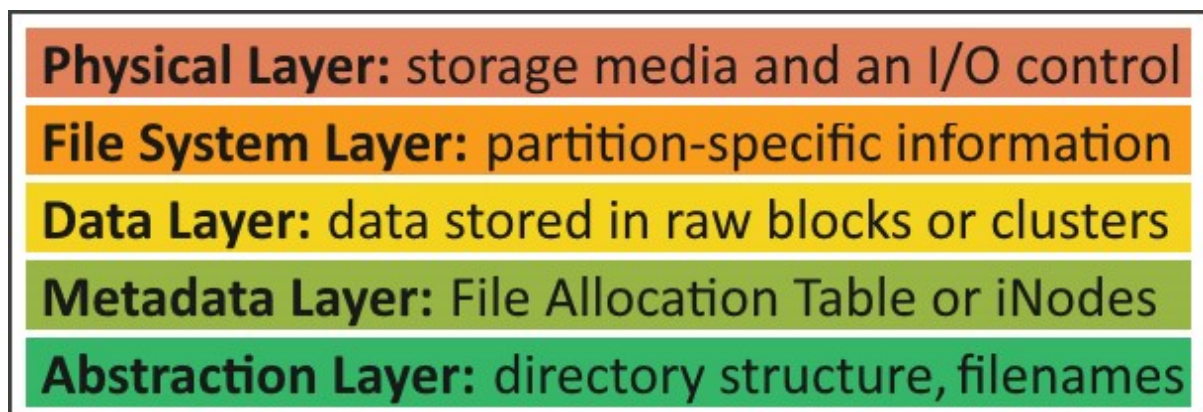


Figure 36. Common File System Layers.

Different computer systems are used for different tasks. Depending on the particular operating environment, target architecture and the actual purpose file systems are generally divided into:

1. *Disk FS* – the largest group of file systems designed to operate on a disk storage media and allow multiple processes to access data seamlessly (regardless its actual placement on the media); most notable disk file systems: NTFS, FAT32, FAT16, ext3, ext4, ZFS.
2. *Optical Disk FS* – basically the same as disk file systems, but designed specifically for the use with CD, DVD and Blu-ray disks; known file systems: ISO 9660, Joliet, UDF.
3. *Flash FS* – file systems designed specifically for the use on flash memory-based media to avoid the contradiction of utilizing a block-based data access on non-block devices and related drawbacks (such as write amplification); flash file systems: JFFS2, LogFS. Note: USB sticks, MMC and SD cards, SSD disks, etc. have built-in Flash Translation Layer (TrueFFS or ExtremeFFS), and therefore are represented as usual block devices.
4. *Tape FS* – file systems intended to be used on magnetic tapes, allowing the files to be stored in a self-describing form (meaning that it also contains index metadata), as well as reducing performance overhead resulting from sequential nature of a magnetic tape. Many implementations (IBM, Oracle, Quantum, HP) of Linear Tape File System exist.
5. *Database FS* – file systems with built-in support for rich metadata and hierarchical file management features based on the database concepts, which allows files to be indexed and queried fast by their attributes; popular file systems: Apache Hadoop, GFS, BeFS.
6. *Network FS* – an abstraction layer (used on top of general-purpose file systems), which provides seamless access to remote files via network protocols: NFS, SMB, AFS, FTP.
7. *Virtual FS* – a high level abstraction layer that provides an Operating System with the uniform access (path disambiguation, abstracted file management operations, etc.) to the underlying general-purpose file systems (may be located on different devices or in memory), transparently integrating them into file tree through the use of mount points. VFS also allows non-file entities (I/O devices, processes, etc.) to be presented as files.
8. *Shared Disk FS* – centralized way of providing multiple machines or machine clusters with a direct concurrent access to the same shared block storage device, preventing the write collisions through the use of a lock manager; variations: GFS2, StorNext, CXFS.
9. *Flat FS* – type of file systems that do not support sub-directories (resulting in the non-hierarchical structure and absence of file paths); common examples: MFS, VTOC, S3.

In addition to the types described above, for special cases (like with DimOS) a memory-based pseudo-filesystem can be used. Such a file system works entirely in RAM, and thus is volatile.

5.8 Artificial Intelligence – Virtual Personal Assistant

As the age of computing constantly brings us new technological flavors all big players (Apple, Google, Microsoft) on the OS market start to take advantage of the AI (Artificial Intelligence), integrating their IPAs (Intelligent Personal Assistant) – Siri, Google Now and Cortana into the iOS, Android and Windows respectively, reaching out to numerous users all around the world.

IPA is a software agent that tries to mimic the human behavior and perform respective actions, such as holding a conversation, making various decisions on user's behalf, storing information obtained from the user in order to customize and personalize the user experience. IPA is often capable of machine learning and reasoning, which allows it to perform autonomous activities.

Such advanced and powerful assistants as Cortana and Siri can control almost all functionality of their Operating Systems by using the API provided by applications and system components. For example, they can automatically manage user's daily schedule, send E-mail messages, add calendar events, look for the information, start/stop the programs, control media playback, etc.

IPAs are usually controlled by voice commands or textual input (just as if you would talk with a real person), and therefore have little or no GUI. Most of the implementations possess only a text field, voice input button and an answer output area as the interactive interface (Figure 37).

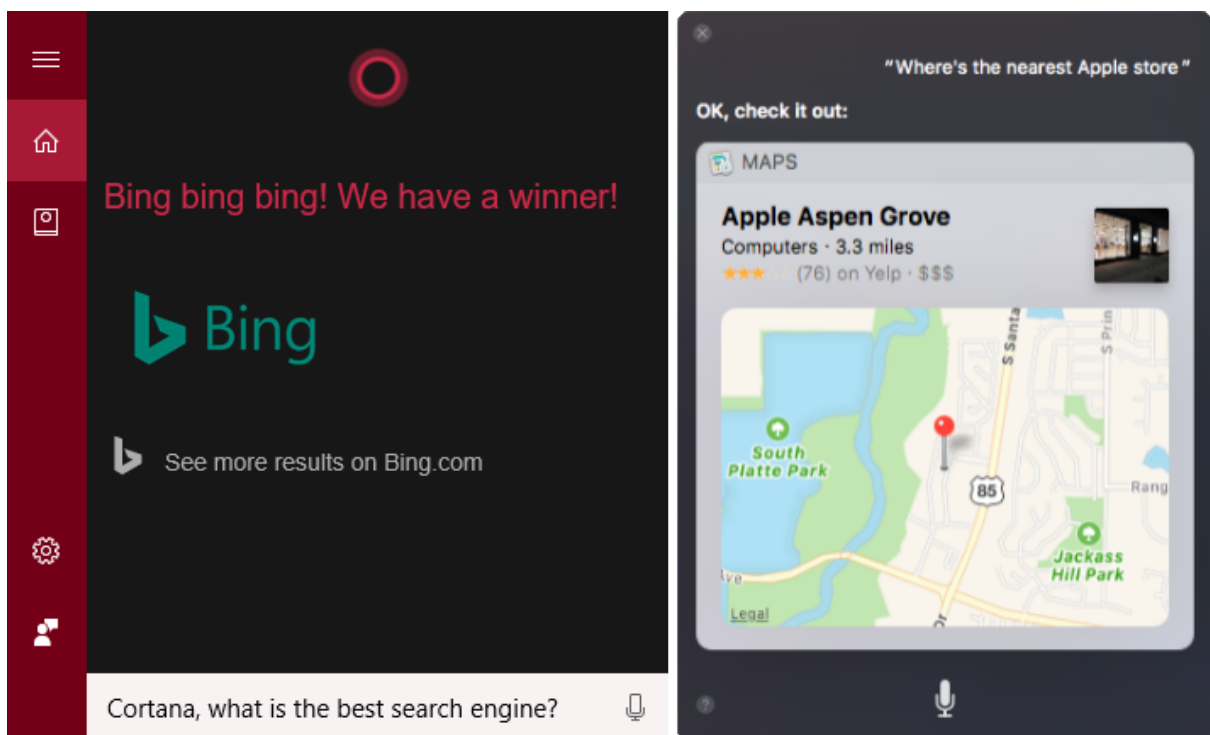


Figure 37. Interactive User Interface of Microsoft's Cortana and Apple's Siri.

5.9 Artificial Intelligence – Implementation Methods

People have been trying to create program that will be able to pass the Turing test for decades. There are many different mathematical, statistical, logical, linguistic and symbolic approaches to AI, but none of them proved to be exclusively right or the most appropriate, as intelligence itself is something yet to be defined. Therefore, I chose two approaches that would fit an IPA.

AIML (Artificial Intelligence Markup Language) is an XML-based open standard used in the natural language processing software to describe the behavioral patterns and question-answer relations that together make up an AI. AIML is one of the most simple and dull approaches to implement. Nevertheless, it was able to achieve quite good results with an acceptable quality of responses and, even though not being “clever” enough to pass the Turing test, AIML gave birth to numerous chatterbots and won the annual Loebner Prize AI Competition three times.

AIML specification includes the mandatory `<aiml>`, `<category>`, `<pattern>`, `<template>` tags, which define the start and the end of an AIML document, unit of knowledge (particular topic or question to be processed, etc.), a certain pattern matching the user input and the actual set of slightly different responses (or just one specific response) to that pattern respectively. The common example of an AIML file (Figure 38) would also contain some of the additional tags.

```

1  <?xml version = "1.0" encoding = "UTF-8"?>
2  <aiml>
3    <category>
4      <pattern>My name is */</pattern>
5      <template>That's a nice name!<think><set name = "username"><star/></set></think></template>
6    </category>
7    <category>
8      <pattern>Hello</pattern>
9      <template>Hi <get name = "username"/>, nice to meet you!</template>
10   </category>
11  </aiml>

```

Figure 38. AIML File Example – Remembering User’s Name.

AIML interpreters are capable of processing only the pre-defined patterns, which makes them predictable and not truly intelligent. However, their functionality can be extended through the use of an artificial neural network (machine learning model that operates similarly to a human brain). In RNN (Recurrent Neural Network) input layer of nodes (neural units) adds an output of the previous iteration, which becomes a context layer, to the processing model, making the algorithm literally learn the information retrieved from the user input. Combined with AIML, RNN can be used to populate the knowledge base of an AI with the new words and sentences provided by user. This technique can help the AI to resemble a real person in the conversation.

6 INITIAL PLANNING

Being the starting point of an actual development process, this section represents the transition from theoretical part of the project to a practical one. All substantial components of the system (User Interface, Command Shell, File System and Applications) will be briefly explained here. Furthermore, practical aspects of the concept, such as the educational process, will be defined.

6.1 Educational Process – Learning Flow

The whole studying process in DimOS revolves around the Operating System itself. Different components together should create a comfortable learning environment, where user can study the basics naturally, just by using the built-in knowledge base and educational tools provided. Such tools will utilize various self-assessment methods and practical tasks that should ideally familiarize user with the Command Shell and common design concepts of Operating Systems.

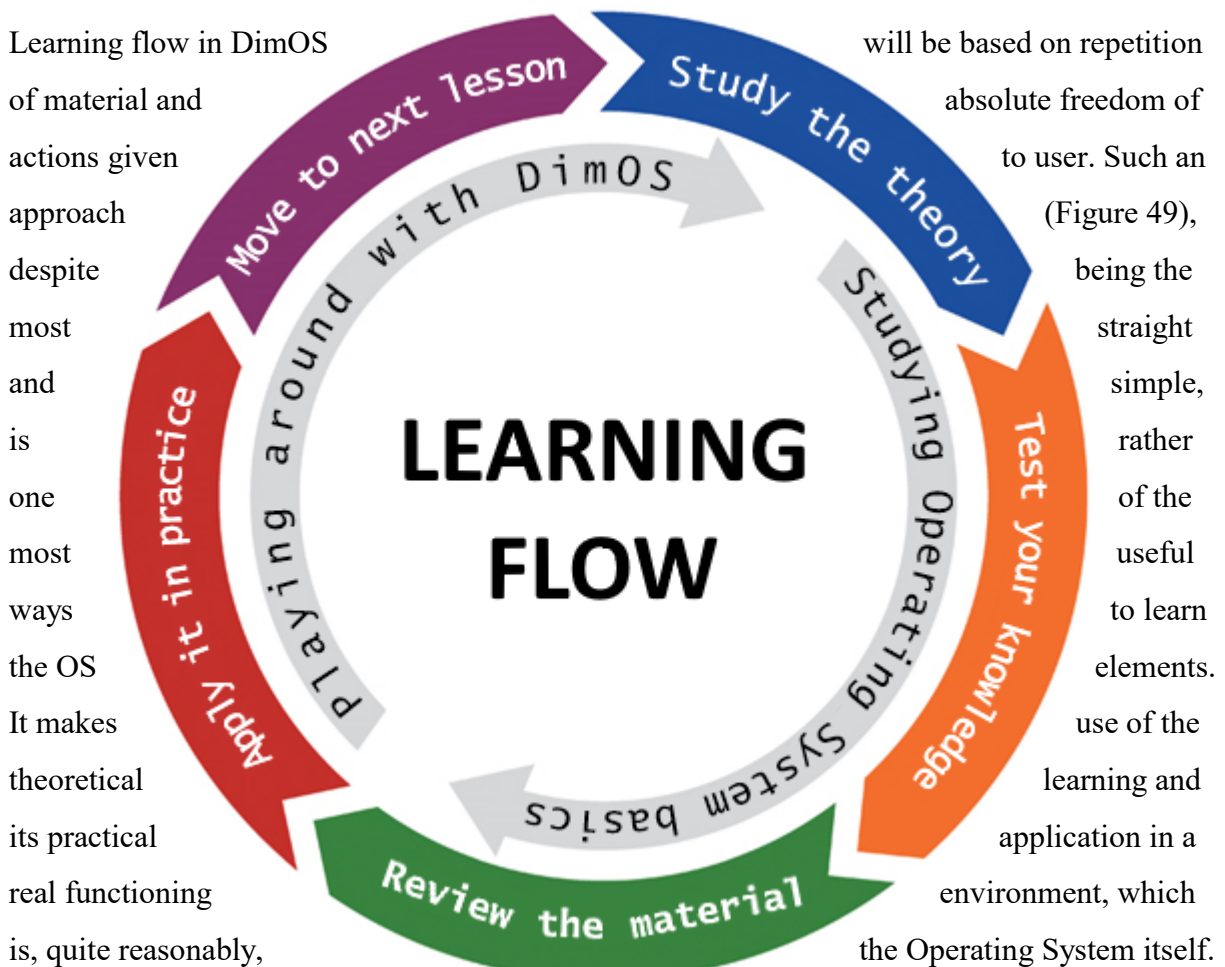


Figure 39. Learning Flow in DimOS.

6.2 Educational Process – Study Tools

DimOS will have the special set of tools, which are supposed to track user’s learning progress and provide different assessment methods, including traditional multiple-choice, short-answer, matching and true/false questions. Each tool will have its own dedicated question bank, which should contain questions of the corresponding type, logically divided into topics or categories.

“Quizzes” tool (Figure 40) is responsible for handling multiple-choice questions that have the number of answer options, only one of which is correct. User should select the correct answer.

```
Microkernel is a small operating system core that provides the foundation for modular
1. Process
2. Programs
3. Application
4. Extensions
Your answer [4] is correct!
```

Figure 40. Conceptual Screenshot of Quizzes Tool.

“Guesses” tool (Figure 41) is used to provide short-answer questions, where an answer can be “guessed”, as there are no hints or options provided. User should type answer into the prompt.

```
There is a term for loading an operating system into memory. What is it?
Your answer [booting] is correct!
```

Figure 41. Conceptual Screenshot of Guesses Tool.

“Targets” tool (Figure 42) manages a set of specific actions or small tasks to be accomplished by the user. The “target” will be marked as complete when user performs the required actions.

```
Switch to the root-access mode [DONE]
Set the password for root-access mode [....]
Set a string in the string buffer [DONE]
Switch back to the user-access mode [....]
```

Figure 42. Conceptual Screenshot of Targets Tool.

Other tools, such as “Scripts”, “Matches” and “Options”, where user will need to write certain scripts, match the terms with their corresponding definitions and select all answer options that apply respectively, will be implemented as well, providing a rich set of generic study features and possibilities that should together guarantee the smooth and enjoyable learning experience.

6.3 Command Line Interface

DimOS will feature a simple yet powerful Command Line Interface consisting of four sections (Figure 43), into which the text screen will be logically divided based on desired functionality.

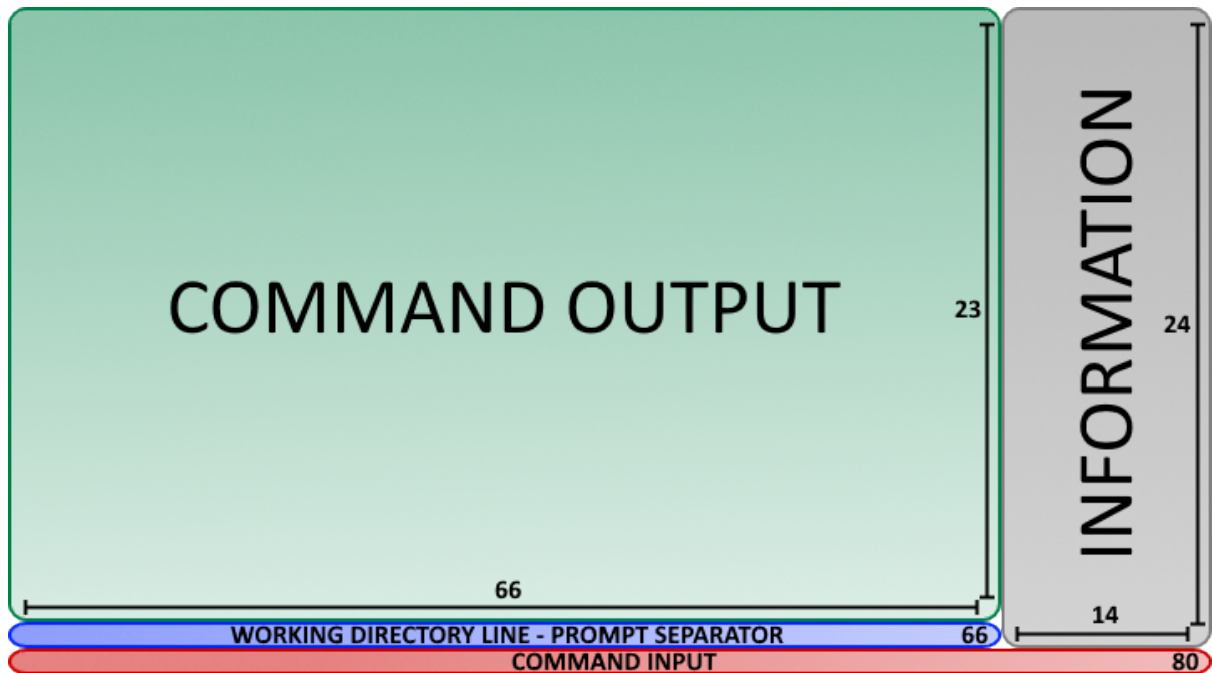


Figure 43. Logical Text Screen Separation.

All sections should be visually separated from one another and each of them should be serving its own dedicated purpose. Apart from self-descriptive command input and output sections, an “Information” section and the special prompt separator line will form the vivid UI (Figure 44).

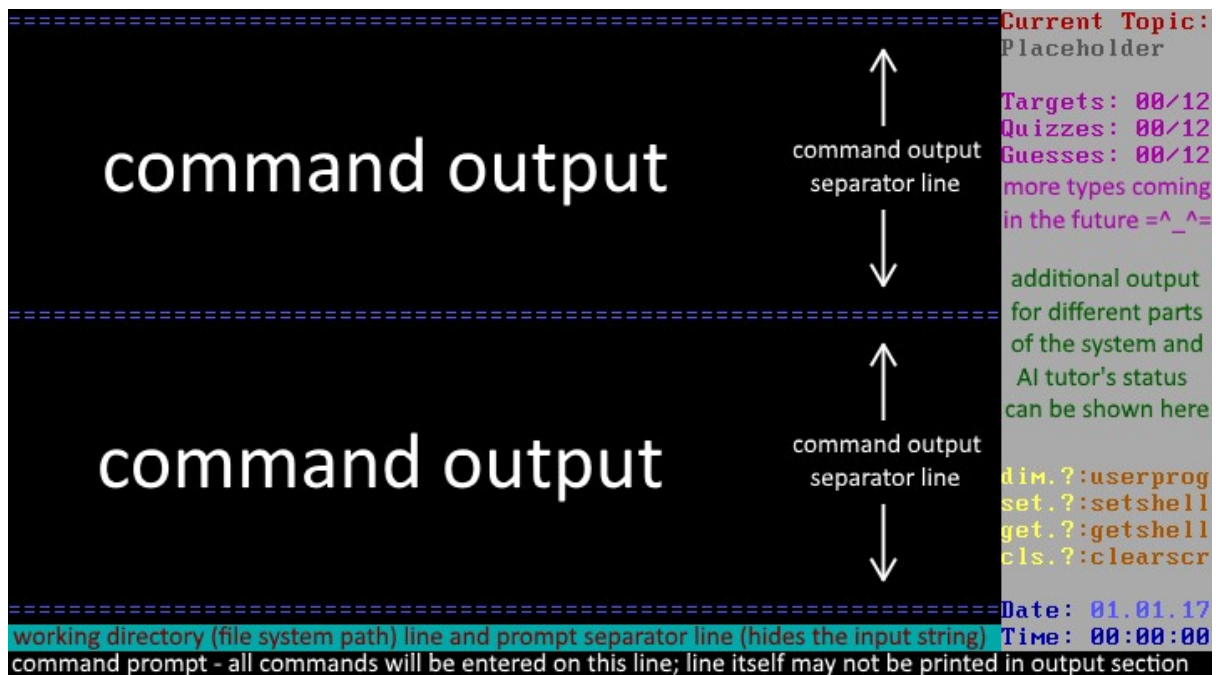


Figure 44. Conceptual Screenshot of DimOS User Interface.

6.4 DimFS – Memory Based Pseudo File System

DimFS (not to be confused with the DFS) is the controversial part of an Operating System that aims to provide the virtual directory and file management interface, which allows user to store data in the tree-like folder structure where pieces of that data are logically represented as files. The controversy lies in the file system’s final purpose – showing the basic operating principles of file systems (common operations with files and directories, difference between the absolute and relative paths, etc.), instead of concentrating on the actual implementation (stability, speed of file access, efficiency of file grouping and enumeration, access permissions, etc.). DimFS is called a pseudo file system because it only mimics the operation of a real disk FS, whereas the inner structure may resemble that of a VFS, and thus contain no physical storage management.

Unlike complex and cumbersome directory structure of modern Operating Systems, DimFS is fairly simple and easy to understand, as it has logically organized hierarchy (Figure 45), which consists of the four main top-level directories residing inside the top-most root directory. Both “temp” and “user” directories are writable for the “user” account, while “data” and “conf” are writable only by “root”, and therefore, along with the contents, appear as read-only for “user”.

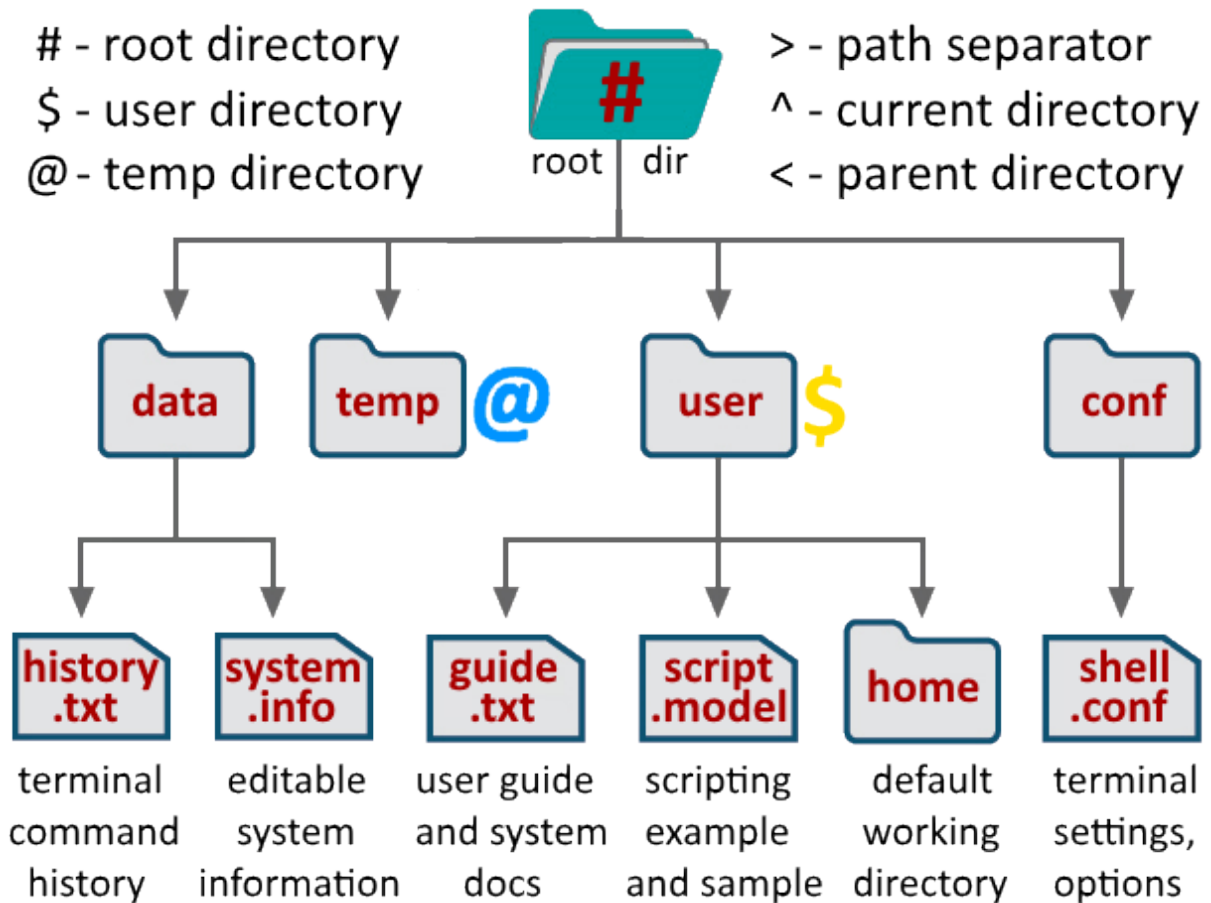


Figure 45. Default Directory Structure in DimFS.

6.5 DimSH – Intelligent Command Shell

DimSH is the only interaction interface between the user and OS. Being an irreplaceable and vital component of the system, it must possess a high degree of both stability and consistency. Comparing to modern command shells, which are designed solely for computer professionals and IT specialists, DimSH is absolutely unique, because it targets intermediate students, total beginners and experienced computer users at the same time. The shell aims to provide smooth and continuous operation through the use of advanced input processing and modular structure.

The prompt line (Figure 46), which constantly accepts and processes user input, consists of an OS identifier (“DimOS”) and an access-mode identifier (“user” or “root”) separated by the dot and enclosed in square brackets followed by the home directory identifier (“\$” for user-access mode or “#” for root-access mode) and a colon. Submitted input is not projected to the output.

```
=====Date: 01.01.17
#>user>home>Time: 00:00:00
[DimOS.user]$:
```

Figure 46. Conceptual Screenshot of DimSH Prompt.

The command output (Figure 47) is completely isolated from the prompt input and consists of submitted command with its parameter/arguments enclosed in square brackets followed by the home directory identifier and a colon, after which an actual output from the shell is presented.

```
=====
[dim.expression 50 + 25 / (4 * 10 - 2) * 4^2]$:
60.52631
=====
```

Figure 47. Conceptual Screenshot of DimSH Command Output.

Input Processing Modules (Figure 48) are responsible for executing and showing the output of an appropriate system instruction selected on the basis of user input. The IPMs are extendable.



Figure 48. Main Input Processing Modules of DimSH.

6.6 DimSH – Command Evaluation

Command evaluation in DimSH is based on advanced string processing mechanism, which is responsible for a proper recognition and execution of all shell commands (Figure 49) and user aliases. The corresponding IPM will be tightly integrated with the main shell scripting engine, enhancing its functionality. Commands are organized hierarchically based on the target usage.

A	set. get.	alias	creates or shows an alias for command(s)
B	dim.	beep	makes a beeping sound of specified length
C	dim.	copy	copies file/directory to a specified location
D	set. get.	directory	sets or displays current working directory
E	dim.	explain	get or provide an explanation for a term
F	set. get.	function	defines or shows a parametrized function
G	dim.	gui	starts the graphical user interface (future)
H	set. get.	help	sets or shows manual for given command
I	dim.	internal	executes internal OS functions (3rd level)
J	set. get.	journal	shows or creates a new command history
K	set. get.	keymap	maps or displays actions for the given key
L	dim.	list	lists the contents of directory or collection
M	dim.	move	moves file/directory to a specified location
N	dim.	new	creates new file/directory of a given name
O	dim.	open	opens a given file in default viewer/editor
P	set. get.	permission	sets or shows file/directory permissions
Q	set. get.	query	creates or displays the search/filter query
R	dim.	remove	removes file/directory or a system object
S	dim.	search	searches for object(s) based on the query
T	set. get.	topic	sets or displays the list of study/talk topics
U	set. get.	user	sets or displays the current access-mode
V	set. get.	variable	creates/modifies or shows system variable
W	dim.	workbook	opens a workbook with various exercises
X	dim.	xor	performs a xor operation on file or object
Y	set. get.	yawn	sets or shows a probability of random talk
Z	set. get.	zone	sets or displays current system time zone

Figure 49. DimSH Built-in Shell Commands.

6.7 Shell Scripting and Artificial Intelligence Tutor

Scripting is one of the most valuable capabilities of every shell, as it provides all facilities and instruments needed for efficient task automation and quick problem solving. DimSH will take the modular approach to scripting, which allows different programming/scripting languages to be used in DimOS. Based on the language declaration statement (found in the beginning of the script file) “dim” command should pass its source code to a right language interpreter. Each of the interpreters will be implemented as a separate module. Prospectively DimSH will support:

1. *H* (Figure 50) – the main scripting language of DimOS, which features simple yet very powerful syntax, fully dynamic and modular structure, high-level data abstractions and full access to all functions and capabilities of the shell and the Operating System itself.
2. *Brainfuck* (Figure 51) – the most famous of all esoteric languages, which contains only 8 commands that make it easy to implement and mind-blowing to use, hence the name.
3. *Cat* (Figure 52) – functional stack-based interpreted programming language created by Christopher Diggins and released to public domain in the form of C#-based interpreter.

<pre> i = 0; while (i++ < 10) { if (i % 2 == 0) print (i, " - even"); else print (i, " - odd"); } </pre>	<pre> +++++++ [>+++ ++++>+++++++ >+++<<<-]>+.>+ .+++++. .++++.> ++.<<+++++++ +++++.> .+++ .--- ---.-----.>+. </pre>	<pre> >> 33 3 * main stack: 99 >> 12 + 5 * main stack: 99 15 >> > main stack: true >> pop main stack: _empty_ </pre>
---	--	---

Figure 50. H Syntax (concept). Figure 51. Brainfuck Syntax. Figure 52. Cat Syntax.

The most interesting feature of DimOS is an Artificial Intelligence Tutor (AIT), which will be implemented as the “last resort” Input Processing Module, meaning that the input not accepted by any of the IPMs will be directed to it. AIT will also stay active while the user interacts with other parts of the Operating System, showing the output in a small container (Figure 53), which consists of a plain border and the name of DimOS-tan, who will be acting as a tutor as well. Text screen of the container is 12 characters wide and 4 characters high, as it is intended for short comments, explanations or expressions that should not be displayed in the “Command Output” section, where the AIT will provide learning material or attempt to evaluate the user input properly.



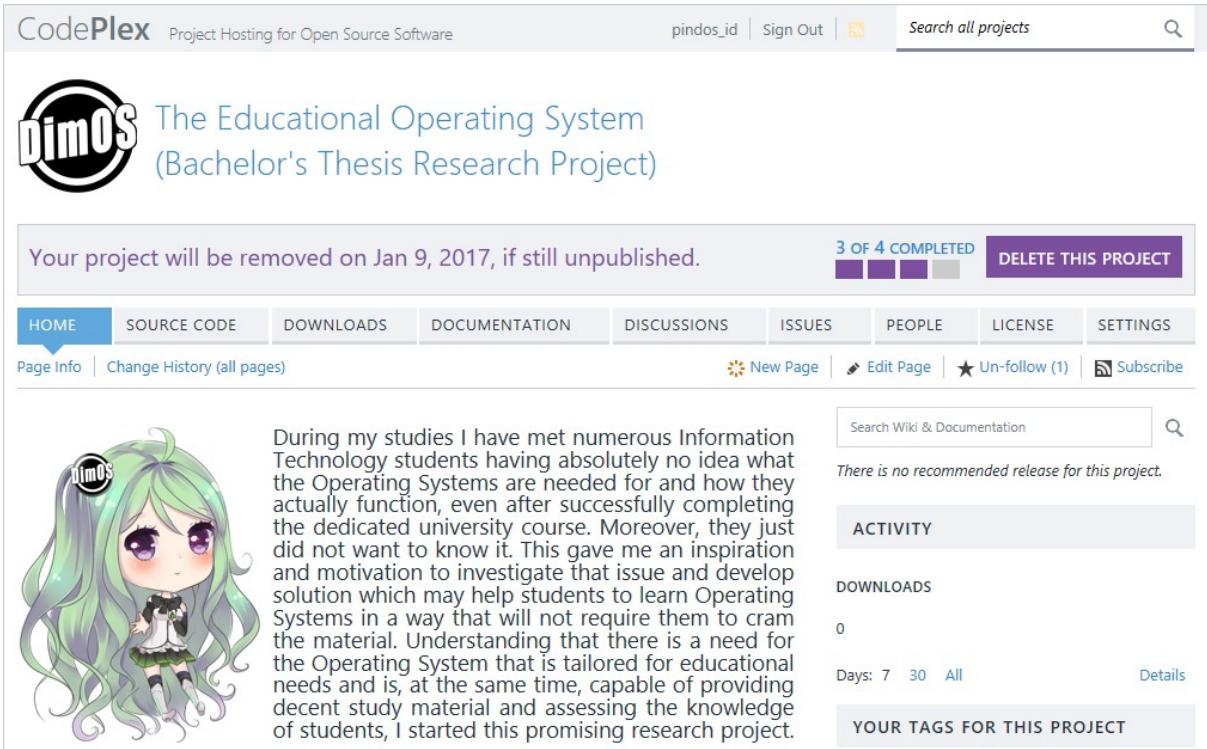
Figure 53. AI Tutor.

7 ENVIRONMENT SETUP

Practical implementation of every project starts with an environment setup – the section where all necessary preparations and configurations (hosting the project, installing the software, etc.) are done. These processes might significantly change over time, making the material outdated. Note: proper operation of a source code is guaranteed only with software versions shown here.

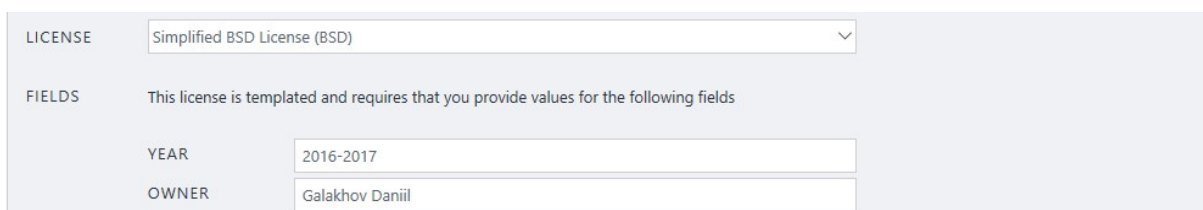
7.1 Hosting the Project on CodePlex

Hosting a project on CodePlex is as easy as clicking the “Create Project” button and following the simple steps provided by the system. After the project is created the user is given an ability to edit its home page (Figure 54), modify various project settings, add the source code, release a particular version of the project, write its documentation, specify the license (Figure 55), etc.



The screenshot shows the CodePlex project page for "The Educational Operating System (Bachelor's Thesis Research Project)". The page features a navigation menu with options like HOME, SOURCE CODE, DOWNLOADS, DOCUMENTATION, DISCUSSIONS, ISSUES, PEOPLE, LICENSE, and SETTINGS. A warning banner indicates the project will be removed on Jan 9, 2017, if still unpublished. The main content area includes a search bar for Wiki & Documentation, a description of the project, and sections for ACTIVITY, DOWNLOADS, and YOUR TAGS FOR THIS PROJECT. The description text reads: "During my studies I have met numerous Information Technology students having absolutely no idea what the Operating Systems are needed for and how they actually function, even after successfully completing the dedicated university course. Moreover, they just did not want to know it. This gave me an inspiration and motivation to investigate that issue and develop solution which may help students to learn Operating Systems in a way that will not require them to cram the material. Understanding that there is a need for the Operating System that is tailored for educational needs and is, at the same time, capable of providing decent study material and assessing the knowledge of students, I started this promising research project."

Figure 54. Writing the Project’s Home Page.



The screenshot shows the license selection form. The LICENSE dropdown is set to "Simplified BSD License (BSD)". Below it, the FIELDS section indicates that this license is templated and requires values for the following fields:

YEAR	2016-2017
OWNER	Galakhov Daniil

Figure 55. Selecting the Project’s License.

7.2 Installing IDE and Supporting Tools

Practical implementation of the project will be held in Visual Studio Ultimate 2013. However, Ultimate Edition requires a paid subscription, which is offered for free of charge by Microsoft DreamSpark (now known as Microsoft Imagine) program. Therefore, Community Edition can be used as well, as it contains all features needed for regular development and is provided free of charge, with the only restriction being the limited number (five) of developers, who can use the IDE concurrently for commercial purposes. It has different installation options (Figure 56).

The screenshot shows two main installation paths for Visual Studio 2013. On the left, 'Visual Studio Community 2013 with Update 4' is highlighted, with a description: 'Download Visual Studio Community for a free, full-featured IDE with powerful coding productivity features, cross-platform mobile development tools for Windows, iOS and Android, and access to thousands of extensions. This edition of Visual Studio is available at no cost for non-enterprise application development.' Below this, there are two options: 'Microsoft Visual Studio Community 2013 with Update 4 - English' with an 'Install now' button, and 'Microsoft Visual Studio Community 2013 with Update 4 - English' with a 'DVD5 ISO image' link. On the right, 'Visual Studio 2013 Language Pack' is shown, described as a 'free add-on for Visual Studio 2013 Language Pack to switch the user interface language'. It includes a 'ダウンロード言語' (Download Language) dropdown menu set to '日本語' (Japanese), and an 'インストールのオプション' (Installation Options) section with a 'Microsoft Visual Studio 2013 Language Pack - 日本語' icon and a '今すぐインストール' (Install Now) button.

Figure 56. Visual Studio Community 2013 Installation Options.

In order to debug and test the Operating System efficiently, VMware Workstation 12 Player or any other compatible version can be used. Additionally, VMware VIX 1.15 API (scripting and task automation engine for VMware products with the bindings for C, Perl, VB, VBS and C #) can be used to integrate it with Visual Studio. Both products are available for free (Figure 57).

The screenshot shows the VMware Download Center interface. At the top, there are tabs for 'Product Downloads', 'Drivers & Tools', and 'Open Source'. The main content area features 'VMware Workstation 12.5.2 Player for Windows 64-bit Operating Systems.' with a prominent blue 'Download' button. To the right, there is an 'About This Product' section with a 'DESCRIPTION' table. Below this, there is a section for 'VIX API 1.15', which includes release information: 'Released 24 AUG 2015 | Download SDK 1.15' and 'VIX 1.15 is the release for VMware Fusion 8 and Workstation 12.' To the right of this section is another table with product details.

DESCRIPTION	FILE NAME	FILE NAME
VMware Workstation 12.5.2 Player	VMware-player-12.5.2-4638234.exe	VMware-VIX-1.15.6-4638234.exe
	BUILD NUMBER 4638234	BUILD NUMBER 4638234
	RELEASE DATE 2016-11-13	RELEASE DATE 2016-11-13

Figure 57. VMware Download Center.

The most important component to be installed is Cosmos User-kit version 108477, which can be downloaded from the project's page on CodePlex. An installation is quite straight-forward.

7.3 Creating the Visual Studio Solution

When all required software packages are installed and configured, the working environment is ready for use. First of all, a new Solution (Figure 58) should be created in Visual Studio. Note that an “Add to source control” option was selected. This will automatically configure the Git.

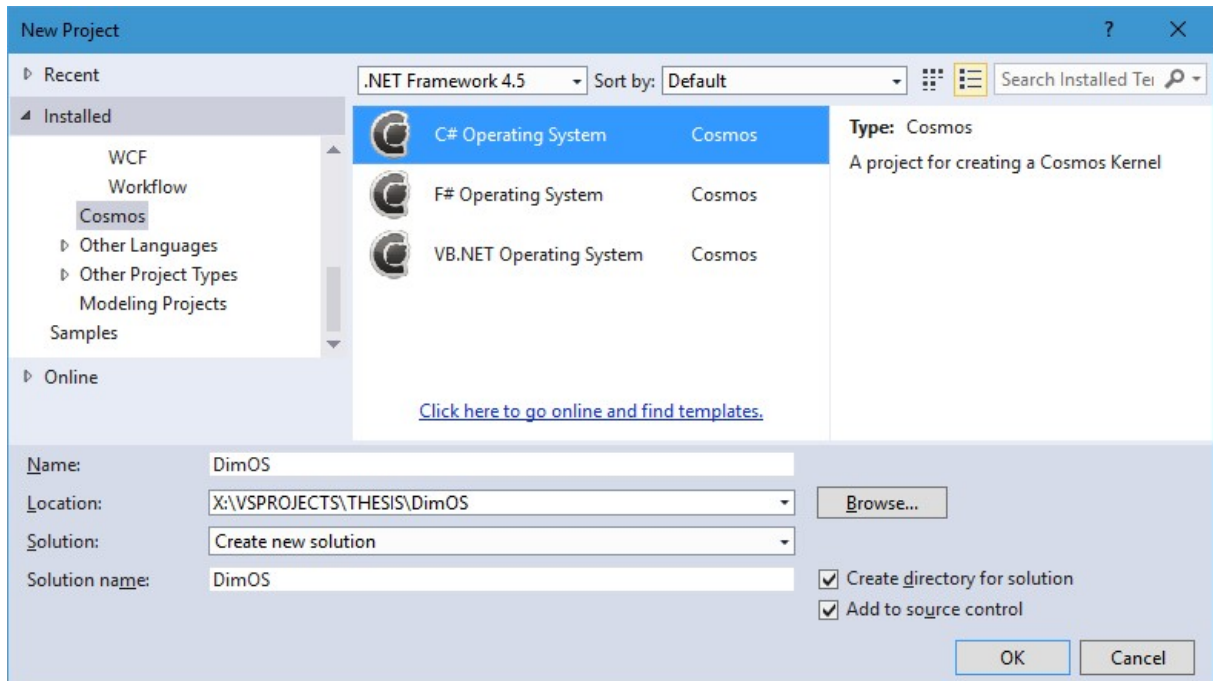


Figure 58. Visual Studio Solution Creation Process.

When the Solution is generated, developer is presented with the neat VS interface (Figure 59).

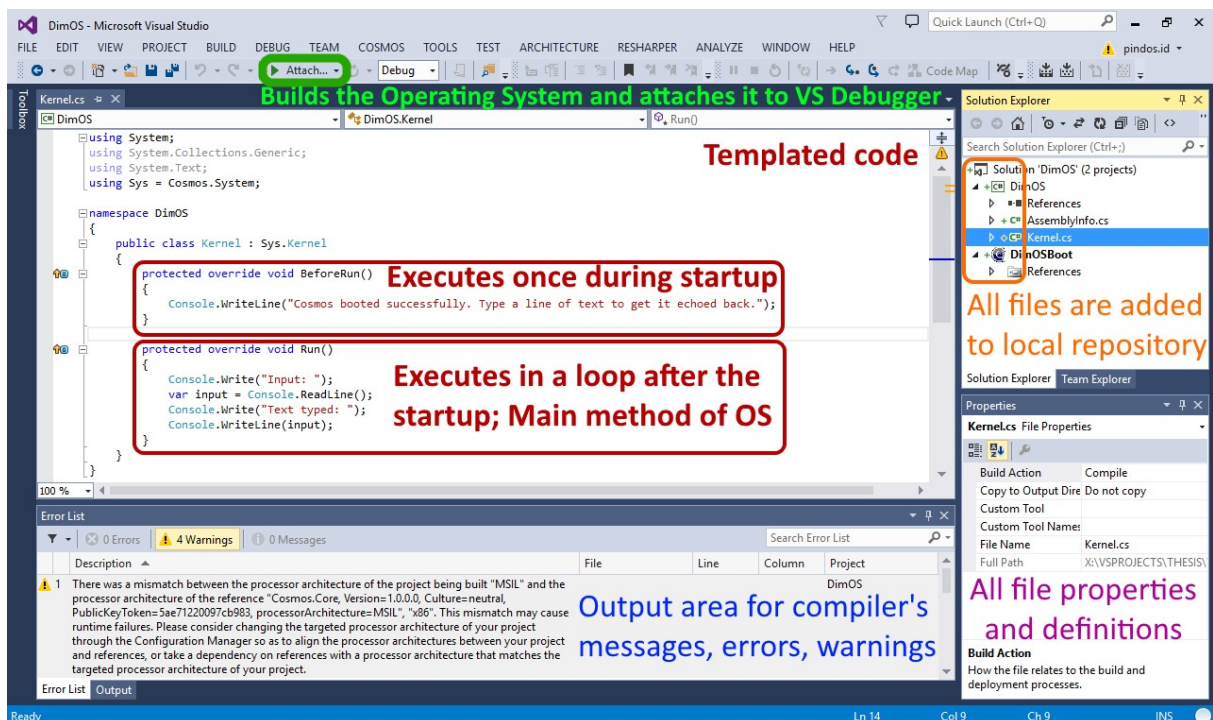


Figure 59. User Interface of Visual Studio.

7.4 Configuring the Source Control

Finally, the project can be published on CodePlex and synchronized with its remote repository that is displayed (Figure 60) under the “Clone” button, which is located on “Source Code” tab.

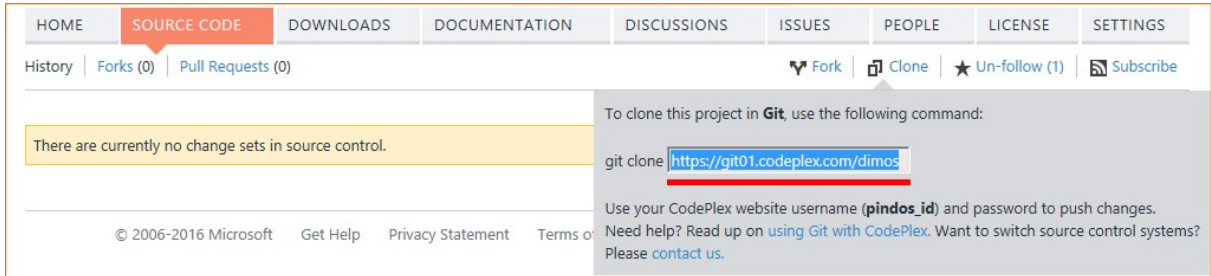


Figure 60. Project’s Git Repository on CodePlex.

CodePlex suggests installing 3rd-party Git tools (either Git BASH or Git GUI). This might be a good choice for those who are already familiar with Git and/or want to have granular control over synchronization process and the repository. However, as everything is now configured to work seamlessly through the Visual Studio, a better approach would be to use tools integrated into the IDE and available straight out of the box. VS provides the Team Explorer (Figure 61).

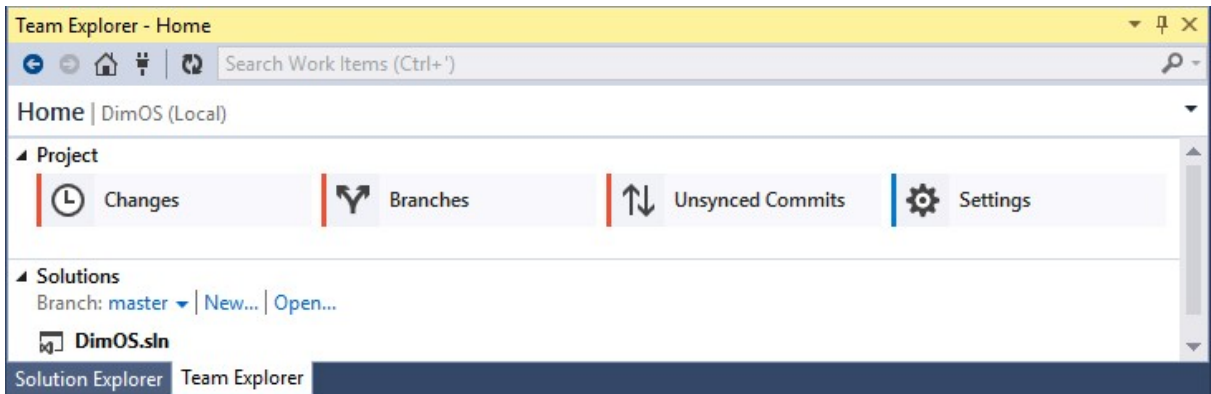


Figure 61. Visual Studio Team Explorer.

“Settings” => “Git Settings” menu allows the developer to configure a User Name and E-mail Address. When this is done developer can start committing. After the first local commit Team Explorer will suggest publishing it to the remote repository. This can be easily done by setting the link to it and entering log-in credentials. CodePlex will now show the commit (Figure 62).

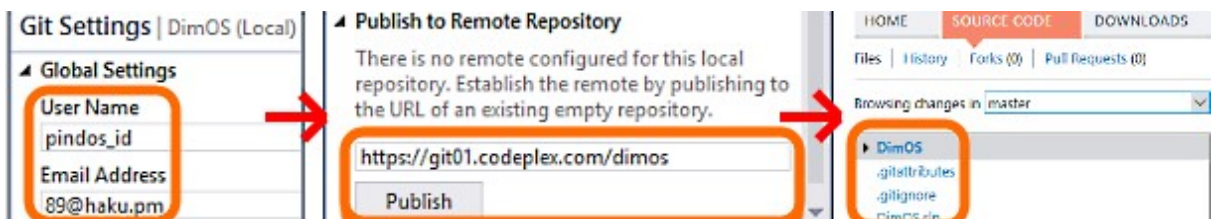


Figure 62. Publishing to Remote Repository.

8 IMPLEMENTATION

This part contains an actual implementation of an Operating System and its main components. With the emphasis put on conceptual logic of various solutions and considerations, the source code will be shown and described in short. Please refer to project's page for a full code listing.

8.1 DimFS – The Concept of Nodes

Instead of files and/or directories, DimFS will store the information in nodes. A single node is self-consistent, as it contains all properties needed by the Operating System, including the link to its parent, a set of permissions, attributes and timestamps. Node in DimFS is different from an inode, which is used to store information about (and link to) the file in all UNIX-based file systems. Any given node has exactly the same implementation and is stored in the OS's RAM exactly the same way as all other nodes, which allows the system to perform complex file and directory operations easily. Functions of a particular node will be defined solely by its Content property, which stores the actual data (list of the child nodes or contents of the file) that can be accessed directly. Due to the memory-based nature of nodes and target architecture (x86) of an Operating System, files larger than 2GB (although such files can be created) can easily cause a heap overflow, which might eventually lead to a page fault, finally resulting in a Kernel Crash. Size property of the node class (Figure 63) will be controlled by DimFS, thus having no setter.

```
//Represents a single node (file/folder) in File System
public class Node
{
    public string Name { get; set; } //Name of the node (no path separator or "hidden")
    public Node Parent { get; set; } //Link to the parent node (null for the root node)
    public Permissions Permissions { get; set; } //Basic permissions for the node (and descendants)
    public Attributes Attributes { get; set; } //Simple attributes for the node (and descendants)

    private Content _content;
    public Content Content //Content of the node (either child nodes or data)
    {
        get { return _content; }
        set { _content = value; Content.Accessed =
            Content.Modified = DateTime.Now; }
    }

    public int Size { get { return Content.Size(); } } //Size of the node's Content (calculated on "get")
    public DateTime Created { get; private set; } //DateTime value showing when the node was created
    public DateTime Modified //DateTime value showing when Content was modified
    { get { return Content.Modified; } }
    public DateTime Accessed //DateTime value showing when Content was accessed
    { get { return Content.Accessed; } }
}
```

Figure 63. Node Implementation.

8.2 DimFS – Permissions

Permissions in DimFS (Figure 64) are applied to the “user” account only. The Owner property is determined during the node creation based on the current access-mode (account) in use. The “user” account by default has full access to the nodes created by it. However, all nodes created under the “root” account can only be read by “user”, unless needed permissions are configured explicitly. All permissions are enforced on a file system level, meaning that DimOS itself does not need to be aware of them. Default directory layout will be initialized statically (Figure 65).

```
public class Permissions
{
    public Owner Owner { get; set; } //Owner of the particular node (either User or Root)

    public bool Read { get; set; } //Permission that allows reading the node's contents

    public bool Write { get; set; } //Permission which allows writing to node's contents

    public bool Change { get; set; } //Permission which allows changing node's properties

    public bool Delete { get; set; } //Permission that allows deleting or moving the node

    public Permissions(Owner owner, bool read, bool write, bool change, bool delete)
    {
        Owner = owner;
        Read = read;
        Write = write;
        Change = change;
        Delete = delete;
    }

    public static Permissions Default(Owner owner)
    {
        if (owner == Owner.Root) { return new Permissions(owner, true, false, false, false); }
        if (owner == Owner.User) { return new Permissions(owner, true, true, true, true); }
        return null;
    }
}
```

Figure 64. Permissions Implementation.

```
public static List<Node> List = new List<Node>
{
    new Node("#", null, Permissions.Default(Owner.Root), new Attributes(/*todo*/), new Content(/*todo*/)),
    new Node("data", List[0], Permissions.Default(Owner.Root), new Attributes(/*todo*/), new Content(/*todo*/)),
    new Node("temp", List[0], new Permissions(Owner.Root, true, true, false, false),
        new Attributes(/*todo*/), new Content(/*todo*/)),
    new Node("user", List[0], new Permissions(Owner.Root, true, true, false, false),
        new Attributes(/*todo*/), new Content(/*todo*/)),
    new Node("conf", List[0], Permissions.Default(Owner.Root), new Attributes(/*todo*/), new Content(/*todo*/)),
    new Node("home", List[3], Permissions.Default(Owner.User), new Attributes(/*todo*/), new Content(/*todo*/)),
};
```

Figure 65. Permissions of Default Directory Layout.

8.3 DimFS – Attributes

Attributes are special properties assigned to the nodes and used by the system for specific data operations, such as the execution of search queries, creation of temporary files, detection of all changes in system settings and so on. Implemented as Boolean fields (Figure 66), all attributes can be either enabled or disabled. They are assigned to the default nodes statically (Figure 67).

```
public class Attributes
{
    public bool System { get; set; } //Indicates that node is used by system
    public bool TempDir { get; set; } //Marks the node as temporary directory
    public bool UserDir { get; set; } //Marks node as a user-access directory
    public bool Settings { get; set; } //Indicates that node contains settings
    public bool Journal { get; set; } //Marks the node as the command history
    public bool HelpFile { get; set; } //Marks the node as command's help file
    public bool Query { get; set; } //Marks the node as system search query

    public Attributes(bool system, bool tempDir, bool userDir, bool settings,
        bool journal, bool helpFile, bool query)
    {
        System = system;
        TempDir = tempDir;
        UserDir = userDir;
        Settings = settings;
        Journal = journal;
        HelpFile = helpFile;
        Query = query;
    }

    public static Attributes Default()
    { return new Attributes(false, false, false, false, false, false, false); }
}
```

Figure 66. Attributes Implementation.

```
public static List<Node> List = new List<Node>
{
    new Node("#", null, Permissions.Default(Owner.Root),
        new Attributes(true, false, false, false, false, false, false), new Content(/*todo*/)),
    new Node("data", List[0], Permissions.Default(Owner.Root),
        new Attributes(true, false, false, false, false, false, false), new Content(/*todo*/)),
    new Node("temp", List[0], new Permissions(Owner.Root, true, true, false, false),
        new Attributes(true, true, false, false, false, false, false), new Content(/*todo*/)),
    new Node("user", List[0], new Permissions(Owner.Root, true, true, false, false),
        new Attributes(true, false, true, false, false, false, false), new Content(/*todo*/)),
    new Node("conf", List[0], Permissions.Default(Owner.Root),
        new Attributes(true, false, false, false, false, false, false), new Content(/*todo*/)),
    new Node("home", List[3], Permissions.Default(Owner.User), Attributes.Default(), new Content(/*todo*/)),
    new Node("history.txt", List[1], Permissions.Default(Owner.Root),
        new Attributes(true, false, false, false, true, false, false), new Content(/*todo*/)),
    new Node("system.info", List[1], Permissions.Default(Owner.Root), Attributes.Default(), new Content(/*todo*/)),
    new Node("guide.txt", List[3], Permissions.Default(Owner.User), Attributes.Default(), new Content(/*todo*/)),
    new Node("script.model", List[3], Permissions.Default(Owner.User), Attributes.Default(), new Content(/*todo*/)),
    new Node("shell.conf", List[4], Permissions.Default(Owner.Root),
        new Attributes(true, false, false, true, false, false, false), new Content(/*todo*/)),
};
```

Figure 67. Attributes of Default Nodes.

8.4 DimFS – Strict Paths

Paths in DimFS are thoroughly designed to utilize the full potential of nodes and their mutual similarity. Unlike other Operating Systems, DimOS attempts to evaluate and resolve any path, as long as it starts with one of the special symbols – path delimiters (#, \$, @, >, ^ and <). The path can contain wildcard symbols (* or ?) that will get matched either against any number of characters or just one single character. Such path, after expansion, can point to multiple nodes, allowing any operation to succeed if at least one of resolved nodes possesses all attributes and permissions needed for that operation. Paths starting with # (root/top of the directory tree) are considered absolute, whereas all other paths should be identified and treated as relative. Node starting with a dot (.) is considered hidden, which means that it will not be resolved during the path expansion or listing operations. Hidden nodes should be addressed explicitly via the path.

In order to facilitate an educational process and boost capabilities of the file system Strict Path Enforcement (Figure 68) was applied. This feature ensures that the path can solely identify the purpose (whether it should act as a file or a folder) of all nodes it resolves by enforcing the use of path delimiter (>) after traversable nodes (directories) and disallowing its use after the non-traversable ones (files), which provides a logical separation between two behaviours, allowing both the user and the shell to take advantage (Figure 69) of the concept of nodes. By default, a particular node must be utilized either as a file or a directory. However, with “Advanced Path” setting switched on, any node can be used naturally as both file and directory at the same time.



Figure 68. Strict Path Enforcement Rules.

```

dim.list #>home> lists the contents of "home" directory (ls)
dim.list #>passwd lists the contents of file "passwd" (cat)
dim.new >temp> creates empty directory "temp" (mkdir)
dim.new @>test creates empty file named "test" (touch)

```

Figure 69. Command Convergence of File Operations.

8.5 DimSH – Interactive Environment

High degree of interactivity and a natural ease of use are two distinctive properties of DimSH. Text-based user interface of the shell incorporates a variety of useful indicators (Figure 70) to supplement the command prompt. “Command Output” section can be scrolled vertically using PAGE UP and PAGE DOWN keys, which is immediately reflected on the Scrolling indicator in the form of an arrow showing the current position of on-screen data in the buffer. Similarly, “Command Input” section can be scrolled horizontally (LEFT ARROW and RIGHT ARROW keys) with the scrolling status displayed on Input-fit indicator. Input validity indicator is rather special, since it can show in real time (as the user types) whether the command entered will be executed with (red) or without (green) errors. It also aims to warn (yellow) user about possible syntax mistakes, such as unmatched parenthesis, unclosed brackets, wrong/invalid statements.

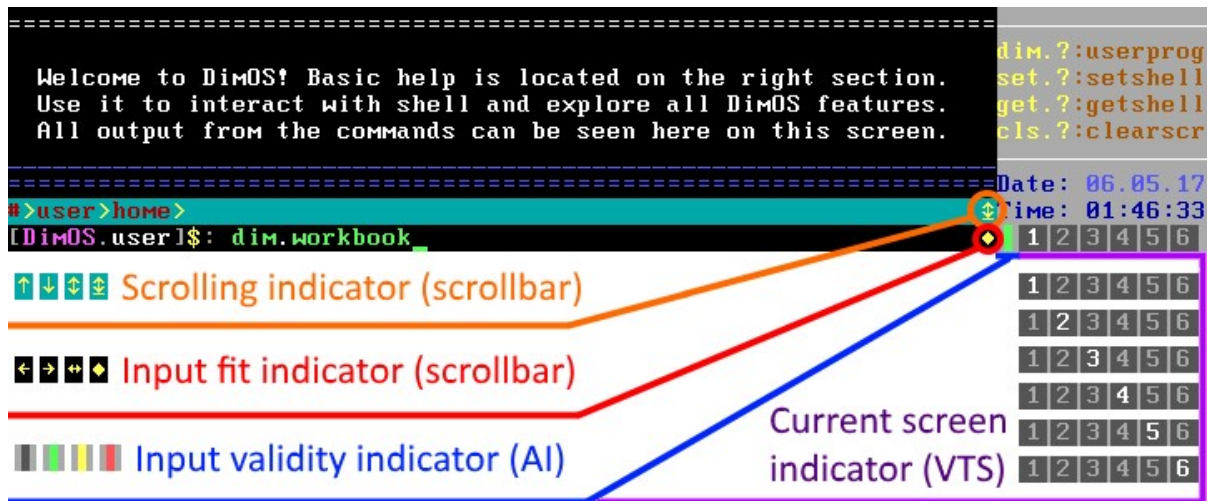


Figure 70. Prompt Indicators.

Current screen indicator is paired with 6 virtual text screens, which the user can easily switch between by pressing F1-F6 functional keys, and shows the screen that is currently active. The actual data to be projected to the real screen is stored in the linear text buffers (one per virtual screen) located in VGA Display Memory. When a certain virtual screen is activated, contents of its buffer are copied over the real screen’s memory block, which is used by VGA controller to render ASCII characters on 80x25 grid (VGA Mode 3) using 16-colour palette (Figure 71).

```
var IS = new CORE.IOPort(0x3DA); //Input Status #1 register
var AC = new CORE.IOPort(0x3C0); //Attribute Control Write register

//Reading from IS triggers AC to await address/index pointing to specific register
var temp = IS.Byte; AC.Byte = 0x10 | 0x20; //Point AC to 10h, don't clear the screen
var value = new CORE.IOPort(0x3C1).Byte; //Read value from Attribute Control Read register
AC.Byte = (byte)(value & 0xF7); //Write the value to AC with 4th bit (blinking) set to 0
```

Figure 71. Enabling 16-Colour Palette (Bright-bit Mode).

8.6 DimSH – Command Journaling

Analogously to other shells, DimSH records all submitted commands in a Journal, which can be seamlessly scrolled back/forth using UP ARROW and DOWN ARROW keys respectively. Each Journal Entry (Figure 72) consists of the command itself (stored as a colored string), its timestamp, comment from the AI Tutor and a link to the parent entry (with a multi-line input).

```
public class JournalEntry
{
    public DateTime Timestamp { get; set; }           //Date and time when the command was submitted
    public List<CharColor> Command { get; set; }     //Command string (includes syntax highlighting)
    public string Commentary { get; set; }          //Possible comment/remark/advice from AI Tutor
    public JournalEntry Parent { get; set; }        //Reference to the parent entry for here-input

    public JournalEntry(List<CharColor> command, string commentary = "", JournalEntry parent = null)
    {
        Timestamp = DateTime.Now;
        Command = command;
        Commentary = commentary;
        Parent = parent;
    }

    public JournalEntry(DateTime timestamp, List<CharColor> command,
        string commentary = "", JournalEntry parent = null)
    {
        Timestamp = timestamp;
        Command = command;
        Commentary = commentary;
        Parent = parent;
    }

    public JournalEntry()
    { Timestamp = null; Command = null; Commentary = ""; Parent = null; }
}
```

Figure 72. JournalEntry Implementation.

Journal can be accessed (Figure 73) by the user through the Journal Node (#>data>history.txt), which acts as a normal text file and gets updated at command submissions, reflecting changes.

```
[ "[10.05.17, 01:42:31]
test command

[10.05.17, 01:44:48]
#(i :: 10) @(i, i % 2 != 0 :: " is odd" :: " is even", &);

[10.05.17, 01:48:57]
@(50 + 25 / (4 * 10 - 2) * 4^2);

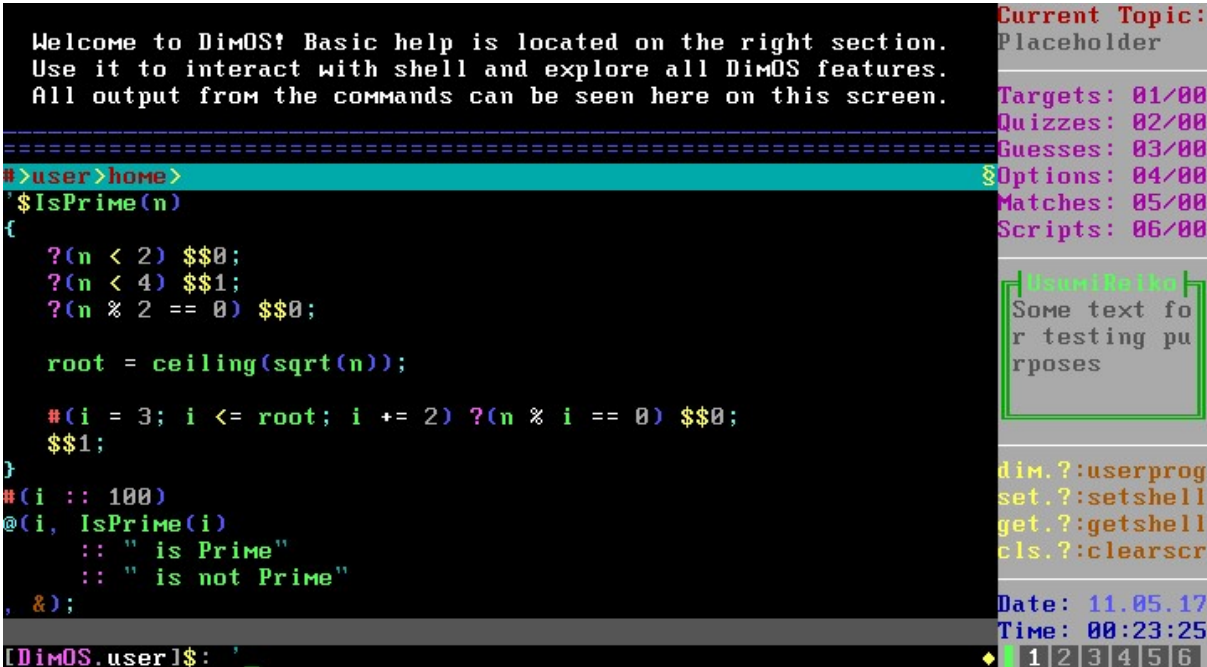
[10.05.17, 01:49:47]
@(x=<"#>data>history.txt");

"]
#>user>home>
```

Figure 73. Contents of Journal Node.

8.7 DimSH – Input Processing

As it was mentioned earlier, DimSH supports here-input and character- or token- based syntax highlighting. Both of these features (Figure 74) require some input processing to be done. This task will be handled by the InputProcessor, which performs lexical analysis of a raw input and determines how exactly it should be treated. Priority-based lexer/tokenizer (Figure 75) divides the string into tokens that can later be parsed by the analyzer. By default, all input is treated as a shell script, which should be passed to an H interpreter for execution and then, together with the outcome produced by scripting engine, forwarded to an AI Tutor for evaluation. However, special cases (advanced token-based syntax highlighting, natural language input, study flow or interaction with the tutor) might provoke an entirely different behaviour involving other IPMs.



```

Welcome to DimOS! Basic help is located on the right section.
Use it to interact with shell and explore all DimOS features.
All output from the commands can be seen here on this screen.

=====
#>user>home>
'$IsPrime(n)
{
  ?(n < 2) $$$0;
  ?(n < 4) $$$1;
  ?(n % 2 == 0) $$$0;

  root = ceiling(sqrt(n));

  #(i = 3; i <= root; i += 2) ?(n % i == 0) $$$0;
  $$$1;
}
#(i :: 100)
@(i, IsPrime(i)
  :: " is Prime"
  :: " is not Prime"
, &);

```

Current Topic: Placeholder

Targets: 01/00
 Quizzes: 02/00
 Guesses: 03/00
 Options: 04/00
 Matches: 05/00
 Scripts: 06/00

Usami@etika
 Some text for testing purposes

dim.?:userprog
 set.?:setshell
 get.?:getshell
 cls.?:clearscr

Date: 11.05.17
 Time: 00:23:25

[DimOS.user]\$:

Figure 74. Multi-line Input and Character-based Syntax Highlighting.

```

public static List<Token> Tokenize(string input)
{
  var list = List.GroupByStartPosition(Token.Find(input));
  var result = new List<Token>(); Token temp = null;

  for (int i = 0; i < list.Count; i++)
  {
    var group = list[i]; //Select the best match in group
    List.SortByPriority(ref group, 0, group.Count - 1);
    if (temp == null || group[0].StartPosition >= temp
      .EndPosition) { temp = group[0]; result.Add(temp); }
  }
  return result;
}

```

Figure 75. Priority-based Tokenizer.

8.8 DimSH – Development Suite

H is a general programming/scripting language designed specifically for DimSH. H interpreter is built directly into the shell, which allows it to perform system-specific actions (changing the prompt, manipulating the file system, accessing VTS memory, etc.) directly without the use of API calls. There is no standard language specification, as H is genuinely dynamic (to an extent that the way source code is interpreted, including the behaviour of operators, functions and the actual data types, can be changed freely during the runtime of a program by that very program) and highly modular (additional functionality in the form of reusable code libraries, source files or scripts written on another language can be plugged in anywhere inside the program), but the main language constructs (Figure 76) will be documented and described in general user guide.

SYM.	NATURAL	EXAMPLE USAGE	DESCRIPTION
	+ - * / ^ %	2 + 2.5 - 0.5; 3^2;	Common mathematical operations (^ is not xor)
	:	"test" : "s"; "3" 0;	IndexOf and Convert ops. (can be used with =)
=	math.op. =	a = 4; a += 2; a -= 2;	Normal- and combined- assignment operations
	&& etc.	1 && 1; "A" ~~ "a";	Logical operations (can be combined together)
	>> << ++ --	"test string">>; a--;	Unary operations (sort, reverse, incr., decr.)
	::	var == "str" :: 1 :: 0;	Ternary op. (similar to if, but returns any type)
	=> = =<	"text" => "@>file";	Input/Output ops. (= - overwrite, => - append)
@	io	@("text"); a = io();	Terminal Read/Write op. (no parameters - read)
!@	substring	a = !@("text",2,2);	Substring function (source, start ind., end ind.)
@@	replace	@@"inn","i","n");	Replace (all occur.) function (source, from, to)
#	loop	#{i = 0; i <= 10; i++}	Loop expression (combines for, while, foreach)
!#	break	#{1} if (a == b) !#;	Break statement (terminates the nearest loop)
##	continue	#{i :: 10}{?(i) ##; ...}	Continue statement (iterates the nearest loop)
\$	define	\$ newFunc(i) { ... }	Defines/Includes new functions (path - include)
!\$	throw	!\$"Error occurred";	Throw statement (throws a custom exception)
\$\$	return	if (a == b) \$\$1; \$\$0;	Return statement (returns a value of any type)
? ..??	if..elif..else	if (i>0) { ... } ?? { ... }	Conditional statement (if ... else if ... else ...)
!?	try	!?(ex) { ... } { @ex; }	Try..Catch statement (parameter - ex. message)
!! &	size wait ↵	a = &(b); !!(5); @&;	Size (var./collection); Wait (seconds); New line;
	math. func.	sqrt round min etc.	Various mathematical functions (pi is a variable)

Figure 76. H Language Constructs.

H source code may contain both symbolic (special characters that are present on the keyboard act as language keywords) and natural (English words are used as language constructs) syntax forms. Syntax comparison (Figure 77) shows the difference between DimSH and major shells.

PowerShell	BASH	DimSH
Write a function which checks whether the string is a palindrome		
<pre>Function Palindrome([String] \$Text) { \$Copy = \$Text.ToCharArray() [Array]::Reverse(\$Copy) \$Text.ToCharArray() -eq \$Copy } </pre>	<pre>function Palindrome { ["\$(echo -n \$1 tac -rs .)" = "\$1"]; } </pre>	<pre>\$Palindrome(text) { \$\$ text == <<text; } </pre>
Write a function which counts all non-overlapping occurrences of substring in a string		
<pre>Function Count([String] \$Source, [String] \$Target) { (\$Source Select-String \$Target -AllMatches).Matches .Count } </pre>	<pre>function Count(){ source=\$1 cnt=0 until ["\${source/\$2/}" == "\$source"]; do source=\${source/\$2/} let cnt+=1 done echo \$cnt } </pre>	<pre>\$Count(source, target) { \$\$ &(source/target) - 1; } </pre>
Write a program which accepts the list of directory paths and returns a common path		
<pre>function CommonPath(\$paths) { for (\$i = 0; \$i -lt \$paths.Count; \$i++) { \$paths[\$i] = (\$paths[\$i] .TrimStart('/').Split('/')) } \$c = -1; \$found = \$false; do { \$t = \$paths[0][++\$c]; for (\$r = 1; \$r -lt \$paths.Count; \$r++) { if(\$t -ne \$paths[\$r][\$c]) { \$found=\$true; break } } until (\$found) for (\$i = 0; \$i -lt \$c; \$i++) { \$s += "/" + \$paths[0][\$i] } return \$s } "Result: " + (CommonPath ("/home/tmp/coverage/test", "/home/tmp/covert/operator/test", ">home/tmp/govern/members/null>")) </pre>	<pre>function CommonPath() { declare -a names declare -a parts declare i = 0 names=("\$0") name="\$1" while x=\$(dirname "\$name"); ["\$x" != "/"] do parts[\$i] = "\$x" i=\$((i + 1)) name="\$x" done for r in "\${parts[@]}" / do for n in "\${names[@]}" do if ["\${n#\$r/}" = "\$n"] then continue 2 fi done echo \$r break done } set -- "/home/tmp/coverage/test" "/home/tmp/covert/operator/test" ">home/tmp/govern/members/null>" echo "Result: " echo CommonPath "\$@" </pre>	<pre>test = [">home>tmp>coverage>test", ">home>tmp>covert>operator>test", ">home>tmp>govern>members>>null>"]; \$CommonPath(list) { r = ""; ?((&list) == 0) \$\$r; tmp = 0; t = list[0]; #(i::(&t) - 1) { #(j::list) ?(i !< (&j) t[i] !~ j[i]) \$\$!@(r,0,tmp) + ">"; ? (t[i] == ">") tmp = i; r += l[i]; } \$\$r; /*Can be improved later*/ } @("Result: ", CommonPath(test)); </pre>
Write a function which checks whether the string/sentence is a pangram		
<pre>Function Pangram([String] \$text) { \$text = \$text.ToLower() \$alphabet = "abcdefghijklmnopqrstuvwxyz" + "iklmnopqrstuvwxyz" \$pangram = \$alphabet .ToCharArray().Where { \$text.Contains(\$_) } .Count -eq \$alphabet .Length return \$pangram } </pre>	<pre>function Pangram { local alphabet =abcdefghijklmnopqrstuvwxyx local string="\$*" string="\${string,,}" while [[-n "\$string" && -n "\$alphabet"]]; do local ch="\${string%%\${string#?}}" string="\${string#?}" alphabet="\${alphabet/\$ch}" done [[-z "\$alphabet"]] } </pre>	<pre>\$Pangram(text) { text ^= -1; #(a = "a"; a >= "a" && a <= "z"; a++) ?(text <> a) \$\$0; \$\$1; } </pre>

Figure 77. PowerShell, BASH and DimSH Syntax Comparison Table.

Due to the modular structure of the kernel, it is extremely easy to implement other interpreters and seamlessly plug them to the shell. For academic purposes H is accompanied by Brainfuck. Being probably the most popular of esoteric programming languages, Brainfuck still has some educational value, because its working principle is similar to that of a Turing machine that can be used as a mechanism to describe the general principles of computation and programming as a whole, including the Turing completeness, halting problem and sequential access memory. It is clear that Brainfuck, as a language, is inapplicable for any practical task, but a vast adoption of its theoretical use and the technical simplicity of its implementation make it an excellent, or at least relevant (in the context of its ethically questionable name) tool for programming study.

Brainfuck interpreter in DimOS is slightly more interactive than other non-GUI interpreters in that it utilizes unique features of DimSH to visualize (Figure 78) the memory (buffer) contents at runtime. Implementation preserves the original instruction set of Urban Müller's Brainfuck, but two additional instructions were added to allow the use of breakpoints (#) and inline input (!) in the source code, which enables the programmer to debug the program by feeding it with specific input or pausing its execution at certain places to observe the current state of memory.

The screenshot shows a terminal window with a Brainfuck interpreter. The left pane displays a list of memory addresses and their corresponding values, such as 1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880, 3628800, 39916800, 479001600, 6227020800, 87178291200, 1307674368000, 20922789888000, 355687428096000, 6402373705728000, 121645100408832000, 2432902008176640000, 51090942171709440000, and 1124000727777607680000. The right pane shows the 'Current Topic: Placeholder' and progress indicators for 'Targets: 01/00', 'Quizzes: 02/00', 'Guesses: 03/00', 'Options: 04/00', 'Matches: 05/00', and 'Scripts: 06/00'. A green box highlights a text area with the text 'Some text for testing purposes'. The bottom status bar shows the date '18.05.17', time '03:52:58', and a message: '[Brainfuck]: A breakpoint was reached. Press Enter to continue...'. A navigation bar at the bottom right contains buttons 1 through 6.

Figure 78. Visualization of Factorial Program (Brainfuck).

Input/output facilities are provided by the shell, but only one-line input is allowed (Figure 79).

The screenshot shows a terminal window with a Brainfuck interpreter. The left pane displays the text 'Here is the first line' and 'And here is the second one'. The right pane shows the date '18.05.17' and time '14:00:59'. The bottom status bar shows the message: '[Brainfuck]: Here is the first line¶And here is the second one_'. A navigation bar at the bottom right contains buttons 1 through 6.

Figure 79. Single-line Input in Brainfuck (¶ as NewLine).

8.9 Applications and Shell Commands

Command (Figure 80) in DimSH is a special container type for self-consistent function, which accepts parameters separated by space and produces a fairly descriptive and verbose output for both the user (by writing it directly to the “Command Output” section) and interpreter that was used to call/invoke it. Every command must be able to work in a silent mode and report errors.

```
public class Command
{
    public string Key { get; set; } //Command's unique identifier
    public string[] Aliases { get; set; } //Set of alias names for Key
    public string Parent { get; set; } //Hashed(.) parenting command
    public delegate void Entry(List<string> parameters, string //Delegate that describes the
    parent, bool silent, out string result, out Error error); //entries in CommandEvaluator
    public Entry Action { get; set; } //Unique instance of delegate
    public string Description { get; set; } //Command's short description
    public Help Help { get; set; } //Comman's help(manual) entry

    public Command(string key, string[] aliases, Entry action, string description, Help help)
    {
        Key = key;
        Aliases = aliases;
        Action = action;
        Description = description;
        Help = help;
    }
}
```

Figure 80. Command Implementation.

Applications are similar to shell commands, but they cannot be considered a part of the system command tree. Binary execution is not yet implemented in DimOS, so applications are textual files containing a simple header (tells the shell which interpreter to use) and the source code to be interpreted. The header must start with the “#” symbol followed by a string literal (between “ and ”) containing either a path to another application or one of the built-in interpreters’ name (“H”, “Brainfuck” or “Cat”) and end with the semi-colon. Files without headers will be treated as H scripts. Applications can be started with the stand-alone “dim” command. For example, a typical stopwatch/timer application (Figure 81) written on H scripting language and located at #>user>home>stopwatch.h would be started with the command: dim \$>home>stopwatch.h 10

```
1 # "h"; //This is the header for H program / script
2 s = 0; ?(size($) > 0) s = $[0] | 0; //Read param.
3 #(i = 1; i <= s; i++) { !!1; @("Tick: ", i, &); }
```

Figure 81. Timer Example Application.

9 PLUGS

.NET Framework was originally designed to be a monolithic set of the platform-specific code, which sometimes opts for excessive use of the native calls (WinAPI) or constructs that in turn require or partially depend on system libraries. DimOS runs on bare hardware, where all these dependencies simply do not exist, so the code containing them will not compile. This part will demonstrate how to overcome such obstacles and ensure that the whole system will build fine.

9.1 Mathematical Functions

Most of the methods and functions in System.Math namespace use a double-precision (64-bit) floating-point data type for the calculations. However, DimOS is built for the x86 architecture, and therefore is unable to utilize 64-bit types. Such functions/methods must be reimplemented (Figure 82) using the “float” data type. They can be used as System.Single extension methods.

```
public static class Math
{
    public static float Round(this float value)
    { if (value == 0) return value; if (value > 0) { var checker =
      Convert.ToInt32(value) + 1; if (Convert.ToInt32(value + 0.5f)
      == checker) return checker; return checker - 1; } else { var
      checker = Convert.ToInt32(value) - 1; if (Convert.ToInt32
      (value - 0.5f) == checker) return checker; return checker + 1; } }

    public static float Ceiling(this float value)
    { if (value == 0 || Convert.ToInt32(value) == value) return value; if (value
      > 0) return Convert.ToInt32(value) + 1; return Convert.ToInt32(value); }

    public static float Floor(this float value)
    { if (value == 0 || Convert.ToInt32(value) == value) return value; if (value
      < 0) return Convert.ToInt32(value) - 1; return Convert.ToInt32(value); }

    public static float Remainder(float source, float target)
    { return (source - (target * Convert.ToInt32(source / target))); }
}
```

Figure 82. Plug for System.Math.

There is no special Boolean data type in H, so boolean values are stored in floating-point data type variables, which requires generic conversion functions (Figure 83) to be designed as well.

```
public static float ToFloat(bool expression) { return expression ? 1 : 0; }
public static bool ToBoolean(float expression) { return expression == 1; }
```

Figure 83. Float-Boolean Conversion Functions.

9.2 Collections

Neither `IEnumerable<T>`, nor `IEnumerator<T>` interfaces are supported, as both of them need classes from `System.Threading` namespace to be present. DimOS is a single-process Operating System, so it completely lacks the implementation of any of those classes, making it infeasible to plug the interfaces mentioned above. Instead, higher-level abstractions, such as `List<T>` and `Array<T>` (uses non-generic version of an `IEnumerable`) should be plugged. Most of extension methods (Figure 84) and type-specific functions (Figure 85) do not require any complex logic.

```
public static class Array
{
    public static bool Equals(this int[] source, int[] target)
    {
        if (source.Length != target.Length) return false; for
        (int i = 0; i < source.Length; i++) if (source[i] !=
        target[i]) return false; return true;
    }

    public static int[] Clone(this int[] source)
    {
        var result = new int[source.Length]; for (int i = 0; i <
        source.Length; i++) result[i] = source[i]; return result;
    }

    public static string[] Concatenate(this string[] source, string[] target)
    {
        var result = new string[source.Length + target.Length]; int i;
        for (i = 0; i < source.Length; i++) result[i] = source[i]; for
        (int j = 0; j < target.Length; j++) result[i++] = target[j];
        return result;
    }
    //...
```

Figure 84. Plug for Array's Extension Methods.

```
public static class List
{
    public static bool Contains(List<string> source, string item)
    {
        if (source.Count == 0) return false; for (int i = 0; i < source
        .Count; i++) if (source[i] == item) return true; return false;
    }

    public static void Reverse(List<Value> list)
    {
        for (int i = 0; i < list.Count / 2; i++) { var temp = list[i]; list
        [i] = list[list.Count - i - 1]; list[list.Count - i - 1] = temp; }
    }

    public static void Insert(List<Value> list, int index, Value item)
    {
        var temp = new List<Value>(); for (int i = 0; i < list.Count; i++)
        { if (i == index) temp.Add(item); temp.Add(list[i]); } list = temp;
        if (list.Count == 0) list.Add(item);
    }
    //...
```

Figure 85. Plug for List's Functions.

9.3 Char and DateTime

Most of the functions in `System.Char` make use of `System.Globalization.CultureInfo` for some language-specific operations, which are valid only for the Unicode-based character sets. These functions can be safely implemented (Figure 86) without the `CultureInfo` calls, because ASCII is the only encoding supported in DimOS. However, Strings are stored in memory in Unicode.

```
public static class Char
{
    public static char ToLower(char character)
    { if ('A' <= character && character <= 'Z') character =
      (System.Char)(character | 0x20); return character; }

    public static char ToUpper(char character)
    { if ('a' <= character && character <= 'z') character =
      (System.Char)(character & ~0x20); return character; }

    public static bool IsDigit(char character)
    { return (character >= '0' && character <= '9'); }

    public static bool IsLetter(char character)
    { character |= (char)0x20; return (character >= 'a' && character <= 'z'); }

    public static bool IsLetterOrDigit(char character)
    { return IsLetter(character) || IsDigit(character); }

    public static bool IsWhiteSpace(char character)
    { if (character == Tokens.Space || (character >= '\x0009' && character <= '\x000d')
      || character == '\x00a0' || character == '\x0085') return true; return false; }
}
```

Figure 86. Plug for Char's Functions.

`System.DateTime` utilizes `CultureInfo` as well, but it also stores date and time in 64-bit integer data type as the number of 100 nanosecond intervals passed since the 12 AM January 1st, year 1 A.D. in Gregorian Calendar. For that reason, a new `DateTime` should be created (Figure 87).

```
public class DateTime
{
    public int Year { get; set; }

    public byte Month { get; set; }

    public byte Day { get; set; }

    public byte Hour { get; set; }

    public byte Minute { get; set; }

    public byte Second { get; set; }

    public DateTime(int year, byte month, byte day, byte hour, byte minute, byte second)
    { Year = year; Month = month; Day = day; Hour = hour; Minute = minute; Second = second; }

    public static DateTime Now
    { get { return new DateTime(Convert.ToInt32(HAL.RTC.Century + "" + HAL.RTC.Year), HAL
      .RTC.Month, HAL.RTC.DayOfTheMonth, HAL.RTC.Hour, HAL.RTC.Minute, HAL.RTC.Second); } }
}
```

Figure 87. Plug for System.DateTime.

9.4 Strings

String operations require CultureInfo class too and also have to be reimplemented (Figure 88).

```

public static class String
{
    public static byte[] Encode(this string input)
    { var output = new byte[input.Length]; for (int i = 0; i < input
      .Length; i++) output[i] = (byte)input[i]; return output; }

    public static string Decode(this byte[] input)
    { var output = ""; for (int i = 0; i < input.Length; i++)
      output += (char)input[i]; return output; }

    public static bool Matches(this string text, string pattern)
    {
        if (text.Length + pattern.Length == 0) return true; //Nothing left to evaluate

        //Ensure that the pattern's part after WildcardAny(*) exists in the text string
        if (pattern.Length > 1 && pattern[0] == Tokens.WildcardAny && text.Length == 0)
            return false;

        //Check if text's character matches the character or WildcardOne(?) in the pattern
        if ((pattern.Length > 1 && pattern[0] == Tokens.WildcardOne) || (pattern.Length != 0 &&
          text.Length != 0 && (pattern[0] == text[0] || pattern[0] == Tokens.WildcardOne)))
            return Matches(text.Substring(1), pattern.Substring(1));

        //Either evaluate or skip the text's character if WildcardAny(*) was met
        if (pattern.Length != 0 && pattern[0] == Tokens.WildcardAny) return
          Matches(text, pattern.Substring(1)) || Matches(text.Substring(1), pattern);

        return false;
    }

    public static bool StartsWith(string source, string target)
    { return IndexOf(source, target) == 0; }

    public static bool EndsWith(string source, string target)
    { if (target.Length > source.Length) return false; return source
      .Substring(source.Length - target.Length) == target; }

    public static string Replace(string text, string from, string to, bool skipMirrored = false)
    {
        var result = ""; do { var index = IndexOf(text, from); if (index == -1) break; result
          += (index != 0 ? text.Substring(0, index + (skipMirrored && (text[index - 1] == Tokens
            .MirrorLeft || text[index - 1] == Tokens.MirrorRight) ? from.Length : 0)) : "") + to;
          text = text.Substring(index + from.Length); } while (text.Length != 0); return result
          + text;
    }

    public static int IndexOf(string text, string substring)
    {
        for (int i = 0; i < text.Length; i++) { bool found = true; for (int j = 0; j < substring
          .Length; j++) { if (text[i + j] != substring[j]) { found = false; break; } } if (found)
          return i; } return -1;
    }

    public static int IndexOfAny(string text, char[] characters, int start = 0)
    {
        var min = -1; for (int i = 0; i < characters.Length; i++)
        { var index = text.IndexOf(characters[i], start); if (index
          != -1 && (index < min || min == -1)) min = index; } return min;
    }
    //...
}

```

Figure 88. Plug for String's Functions and Extension Methods.

10 TESTING

Every commit to the Git repository is thoroughly and carefully tested to ensure stability of the system. Uncovered bugs are then inspected further during the debugging phase (Figure 89) by using software breakpoints (step-by-step debugging) and observing an Assembly listing along with the state of CPU registers and the stack. In addition to manual testing, unit tests are used.

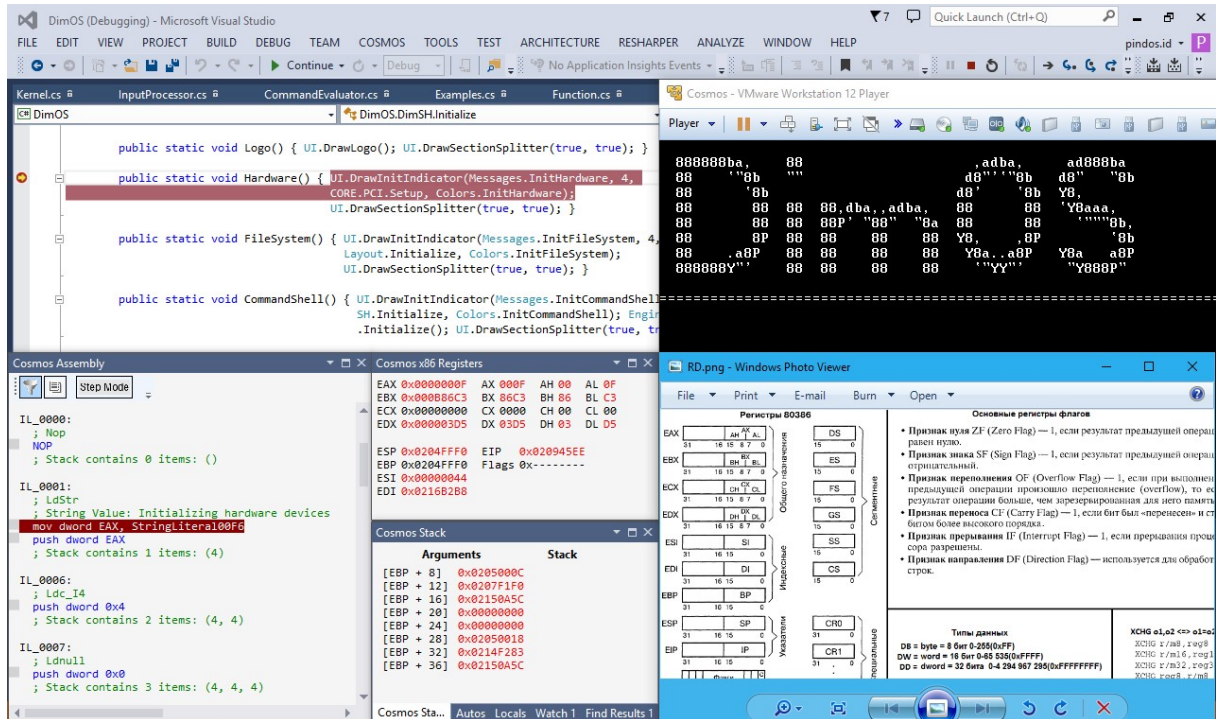


Figure 89. Testing/Debugging Environment.

Occasionally DimOS is also tested on real hardware (Figure 90) to ensure stable and swift run.

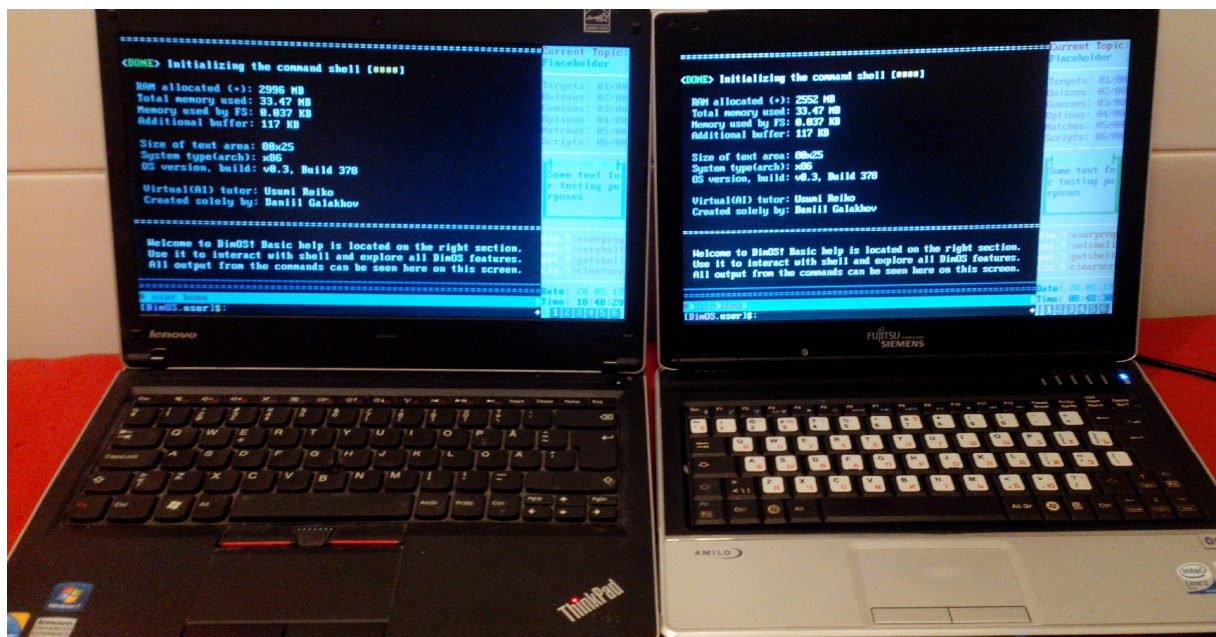


Figure 90. DimOS Running on Real Hardware.

11 CONCLUSION

Due to the time constraints, the project, at the time of this writing, still remains unfinished. An actual state of the system can be classified as the early alpha. However, despite the fact that all key functionality has not yet been implemented, DimOS was proved to be a fairly stable piece of software with rich set of features and high degree of usability. During this project, 10 major commits (Figure 91) with important changes were pushed to a remote repository on CodePlex.











Today		
	Small fixes. Possible feature freeze until the first release. pindos_id - d8e30b6f	master 21:59
Last Week		
	Implemented and plugged in H scripting language. Introduced better handling for possible page fault. pindos_id - 8fc8486a	Thu 11-May-17
Older		
	Finished the interactive Brainfuck interpreter. pindos_id - 5f8c430e	03-Mar-17
	Implemented brainfuck interpreter. Various small fixes. Some progress with shell tokenizer. pindos_id - 89b7d432	26-Feb-17
	Partly implemented the prompt. Stable build. pindos_id - e2502395	01-Feb-17
	Implemented scrollable Virtual Text Screens pindos_id - 30246925	27-Jan-17
	Implemented the main function for resolving paths. pindos_id - 21b287c6	08-Jan-17
	Implemented signature classes for future FS pindos_id - fe7b0b90	03-Jan-17
	Implemented the basic logic for command processor. pindos_id - 87ca8593	23-Dec-16
	Initial conceptual build pindos_id - 50263d0a	13-Dec-16

Figure 91. Git Repository Commit History.

Considering complexity of the project topic, there is a huge room for future improvements and modifications in both hardware- and software- related aspects. The most significant challenges to tackle in the near (provided that there is enough time and motivation) future would include:

1. Adding support for real file systems (FAT, Ext2) to store data in non-volatile memory.
2. Implementing binary execution to run pre-compiled instructions without an interpreter.
3. Designing Artificial Intelligence engine to provide comprehensive support for the user.
4. Creating a Grapical User Interface with frame buffer rendering and support for mouse.

During this research project I was able to accomplish all main objectives and gain tremendous amount of knowledge about Operating Systems, computing and software development with C#.

BIBLIOGRAPHY

BrokenThorn Entertainment Co., 2008. *Operating System Development Series*. WWW page.
<http://www.brokenthorn.com/Resources/OSDevIndex.html>

Fountain Computer. *BIOS Key Codes*. WWW page.
http://www.fountainware.com/EXPL/bios_key_codes.htm

Kevin Matz, 1997. *Atrevida Game Programming Tutorial #6: Hardware Ports*. WWW page.
<http://atrevida.comprenica.com/atrtut06.html>

K. G. Finogenov, 12.02.1990. *Working with MS-DOS*. Booklet. ISBN: 5-03-002346-1.

J. D. Neal, 1998. *Hardware Level VGA and SVGA Video Programming Information Page. Attribute Control Registers*. WWW page.
<http://www.osdever.net/FreeVGA/vga/attrreg.htm#10>

McGraw-Hill Education, Herbert Schildt, 18.05.2010. *C# 4.0 The Complete Reference*. Book.
<https://www.amazon.com/4-0-Complete-Reference-Herbert-Schildt/dp/007174116X>

Microsoft Corporation. *.NET Framework 4.5 Development Guide*. WWW page.
[https://msdn.microsoft.com/en-us/library/hh156542\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/hh156542(v=vs.110).aspx)

Paul Krzyzanowski, 26.01.2014. *Operating System Concepts*. WWW page.
<https://www.cs.rutgers.edu/~pxk/416/notes/03-concepts.html>

RSSING.COM, 25.12.12. *Archived Cosmos RSS Channel*. WWW page.
http://cosmos343.rssing.com/chan-7149039/all_p1.html

SPDX Workgroup. *BSD 3-clause "New" or "Revised" License*. WWW page.
<https://spdx.org/licenses/BSD-3-Clause.html>

Wikipedia Foundation Inc., 2017. *Artificial Intelligence Markup Language*. WWW page.
<https://en.wikipedia.org/wiki/AIML>

APPENDIX

Microsoft Corporation, Microsoft Windows 10. Operating System.

<https://www.microsoft.com/en-us/windows/get-windows-10>

Microsoft Corporation, Microsoft Visual Studio Ultimate 2013. Software Package.

<https://www.microsoft.com/en-US/download/details.aspx?id=44915>

Jetbrains, S.R.O, JetBrains ReSharper. Software Package.

<https://www.jetbrains.com/resharper/>

The Cosmos Project, Cosmos User-kit Release 108477. Software Package.

<http://cosmos.codeplex.com/releases/view/123476>

VMware Inc., VMware Workstation Player 12.5.2. Software Package.

https://my.vmware.com/en/web/vmware/free#desktop_end_user_computing/vmware_workstation_player/12_0|PLAYER-1252|product_downloads

VMware Inc., VMware VIX API 1.15.6. Software Package.

https://my.vmware.com/en/web/vmware/free#desktop_end_user_computing/vmware_workstation_player/12_0|PLAYER-1252|drivers_tools

Microsoft Corporation, Client Hyper-V. Software Component.

<https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/index>

Oracle Corporation, Oracle VM VirtualBox. Software Package.

<https://www.virtualbox.org/wiki/Downloads>

Pete Batard, Rufus 2.9 Portable. Portable Executable.

<http://rufus.akeo.ie/downloads/rufus-2.9p.exe>

Microsoft Corporation, CodePlex Project Hosting Service: <http://www.codeplex.com/>

Daniil Galakhov, DimOS Project Link: <https://dimos.codeplex.com/>