Jason Josselyn

MET CS 575 - Graduate Operating Systems

Project Technical Specifications

1 Introduction

Jason Josselyn’s CS575 Project OS is a custom operating system built using the COSMOS (C# Open Source Managed Operating System) construction kit. It was written exclusively by Jason Josselyn in C#. The COSMOS construction kit uses the IL2CPU compiler to convert C# into native instructions.

**Requirements to run OS:**

<https://raw.githubusercontent.com/pro-tester/JasonJosselynsOS/master/JasonJosselynCS575Project/bin/Release/JasonJosselynCS575ProjectBoot.iso>

VMWare (Other Linux 2.6.x)

Windows

**Requirements to develop OS:**

GitHub (checkout <https://github.com/pro-tester/JasonJosselynsOS.git>)

Visual Studio 2015

.NET Framework 4.5

COSMOS User Kit

VMWare Player

1.1 Virtual Machine

The operating system uses the standard COSMOS bootloader which is the Syslinux associated library. The generated .iso file can be booted in VMWare on a Windows machine by selecting operating system type “Linux” and sub type “Other Linux 2.6.x kernel”. While it will open on a macbook, keyboard input from a Macintosh is not currently supported. Nested VM support was also abandoned as in testing it did not resolve the keyboard input problem. Debugging is also supported for development.

1.2 IDE

Visual Studio 2015 was the IDE used for development because it is required for COSMOS development. In fact, the COSMOS code base itself is essentially a custom Virtual Studio template project type that installs into Visual Studio as an extension. To make any edits to the source you must open with Visual Studio as an extension.

2 Features

The features of the operating system are comprised of an introduction, password protection, and console commands that allow the user to complete the simple tasks: about os, clear screen, echo, help, settings, and reboot.

2.1 Welcome Message

Upon startup the operating system will display the welcome message:



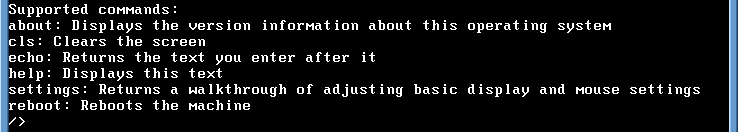
2.2 Password Protection

The operating system has a password prompt that is checked against a hardcoded string before allowing commands to be entered:



2.3 Help

After the correct password is entered the “help” command is automatically called so the user knows all the command options:



2.4 About

The “about” command displays version information about this operating system:



2.5 Clear Screen

The “cls” command clears the screen leaving nothing but a prompt:



2.6 Echo

The “echo” command echoes back any string after it:

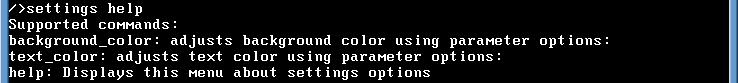


2.7 Settings

The “settings” command takes other commands as parameters such as help, background\_color, and text\_color.

2.7.1 Settings Help

The “settings” command when given any non-existant subcommand (including an empty string) or “help” provides the list of settings subcommands:



2.7.4 Settings Color Help

If the “settings” subcommand “background\_color” or “text\_color” with no string it displays a list of the possible colors:



2.7.2 Settings Background Color

The “settings” subcommand “background\_color” changes the background color of the text:



2.7.3 Settings Text Color

The “settings” subcommand “text\_color” changes the background color of the text:



2.8 Reboot

When the “reboot” command is given the machine restarts:



3 Obstacles

I encountered many obstacles that will help explain some of the source code decisions made, as well as save time for anyone attempting to further develop this operating system in the future.

3.1 No Nested VMs

While initially trying to develop on a Macintosh using the Visual Studio IDE, I was forced to use a virtual windows machine. It was quickly discovered that by default VMWare does not allow virtual machines to be nested. Making it difficult to debug the operating system inside another virtual machine. The simple way around that is to download the COSMOS Development Kit and include allowNested=true in the default .vmx file. Then compile COSMOS and install User Kit.

3.2 No Input on Macintosh

I successfully got the released .iso file to run on Macintosh OS X in both VMWare and Parallels but was never able to get keyboard input to work.

3.3 No Dictionaries, Array, String, or Enum Interfaces

The COSMOS construction kit converts C# into assembly, but it’s compiler still has a lot of work to be done. It does not handle interfaces well which eliminates the ability to use Dictionaries, Array helper methods like .IndexOf, String methods such as .Split and .Substring, and Enum method .Parse.

3.4 No Bool

A known bug that was fixed in November 2015 but is not yet in a COSMOS User Kit release is bools fail to compile. This led to using ints instead of bools in the input helper method.

4 Improvements

Dealing with the major obstacles listed in the previous section while learning a new language I did not get nearly as much done as I would have liked so I have listed possible development improvements for future releases in this section.

4.1 Password Wild Cards

The password is currently being entered in plain text but really it should read key by key and replace each stroke with an asterisk. It should also clear screen upon login.

4.2 Password Hashing

The password is also being checked against a hard coded string with no encryption. It should really be stored in at least a md5 if not stronger encryption.

4.3 Blinking Cursor

Currently there is an unimplemented method in the COSMOS Console API to enable the blinking cursor pointed where the focus is. That would be helpful to implement.

4.4 Mouse

It would be awesome to implement a mouse cursor in order to add more sophisticated user interface applications or even a graphical user interface.

5 Kernel Source Code

Kernel.cs:

using System;  
using System.Collections.Generic;  
using System.Text;  
using Sys = Cosmos.System;  
  
namespace JasonJosselynCS575Project  
  
{  
 public class Kernel : Sys.Kernel  
 {  
 protected override void BeforeRun()  
 {  
 Console.Clear();  
 Console.WriteLine("Welcome to Jason Josselyn's CS575 Project OS!");  
 Console.WriteLine("---------------------------------------------");  
 }  
  
 protected override void Run()  
 {  
 Console.Write("Password: ");  
 string input = Console.ReadLine();  
 while (input != "Passw0rd")  
 {  
 Console.WriteLine("Incorrect password. Please try again.");  
 Console.Write("Password: ");  
 input = Console.ReadLine();  
 }  
 TopLevelCommandsFactory.PrintCommands();  
 while (true)  
 {  
 Console.Write("/>");  
 InputHelper inputHelper = new InputHelper(Console.ReadLine());  
 string command = inputHelper.Command;  
 string parameters = inputHelper.Parameters;  
 TopLevelCommandsFactory.CreateCommandType(command).CreateCommand(parameters).Execute();  
 }  
 }  
 }  
  
 class InputHelper  
 {  
 public string Command { get; }  
 public string Parameters { get; }  
 public InputHelper(string input)  
 {  
 Command = "";  
 Parameters = "";  
 int partOfCommand = 1;  
 foreach (Char c in input)  
 {  
 if (c != ' ' && partOfCommand == 1)  
 {  
 Command += c;  
 }  
 else if (c == ' ' && partOfCommand == 1)  
 {  
 partOfCommand = 0;  
 }  
 else  
 {  
 Parameters += c;  
 }  
 }  
 }  
 }  
  
 class CommandsFactory  
 {  
 private Creator[] creators;  
  
 public CommandsFactory(Creator[] creators)  
 {  
 this.creators = creators;  
 }  
  
 public Creator CreateCommandType(string command)  
 {  
 foreach (Creator creator in creators)  
 {  
 if (creator.Command == command)  
 {  
 return creator;  
 }  
 }  
 return new ConcreteCreatorHelp();  
 }  
  
 public void PrintCommands()  
 {  
 Console.WriteLine("Supported commands:");  
 foreach (Creator creator in creators)  
 {  
 Console.WriteLine(creator.Command + ": " + creator.Description);  
 }  
 }  
 }  
  
 static class TopLevelCommandsFactory  
 {  
 private static CommandsFactory commandFactory = new CommandsFactory(new Creator[] { new ConcreteCreatorAbout(), new ConcreteCreatorClearScreen(), new ConcreteCreatorEcho(), new ConcreteCreatorHelp(), new ConcreteCreatorSettings(), new ConcreteCreatorReboot() });  
  
 public static Creator CreateCommandType(string command)  
 {  
 return commandFactory.CreateCommandType(command);  
 }  
  
 public static void PrintCommands()  
 {  
 commandFactory.PrintCommands();  
 }  
 }  
  
 static class SettingsLevelCommandsFactory  
 {  
 private static CommandsFactory commandFactory = new CommandsFactory(new Creator[] { new ConcreteSettingsCreatorBackgroundColor(), new ConcreteSettingsCreatorTextColor(), new ConcreteSettingsCreatorHelp() });  
  
 public static Creator CreateCommandType(string command)  
 {  
 return commandFactory.CreateCommandType(command);  
 }  
  
 public static void PrintCommands()  
 {  
 commandFactory.PrintCommands();  
 }  
 }  
  
 abstract class Command  
 {  
 protected string parameters;  
 protected Command()  
 {  
 this.parameters = "";  
 }  
 protected Command(string parameters)  
 {  
 this.parameters = parameters;  
 }  
 public virtual void Execute()  
 {  
 Console.WriteLine("This command has not been implmented.");  
 }  
 }  
  
 class ConcreteCommandAbout : Command  
 {  
 public override void Execute()  
 {  
 Console.WriteLine("OS: Jason Josselyn's CS575 Project");  
 Console.WriteLine("Version: 1.0.0.0");  
 }  
 }  
  
 class ConcreteCommandClearScreen : Command  
 {  
 public override void Execute()  
 {  
 Console.Clear();  
 }  
 }  
  
 class ConcreteCommandEcho : Command  
 {  
 public ConcreteCommandEcho(string parameters) : base(parameters) { }  
 public override void Execute()  
 {  
 Console.WriteLine(parameters);  
 }  
 }  
  
 class ConcreteCommandHelp : Command  
 {  
 public override void Execute()  
 {  
 TopLevelCommandsFactory.PrintCommands();  
 }  
 }  
  
 class ConcreteCommandSettings : Command  
 {  
 public ConcreteCommandSettings(string parameters) : base(parameters) { }  
 public override void Execute()  
 {  
 if (parameters == "")  
 {  
 SettingsLevelCommandsFactory.PrintCommands();  
 }  
 else  
 {  
 InputHelper inputHelper = new InputHelper(parameters);  
 string settingsCommand = inputHelper.Command;  
 string settingsParameters = inputHelper.Parameters;  
 SettingsLevelCommandsFactory.CreateCommandType(settingsCommand).CreateCommand(settingsParameters).Execute();  
 }  
 }  
 }  
  
 class ConcreteSettingsCommandBackgroundColor : Command  
 {  
 public ConcreteSettingsCommandBackgroundColor(string parameters) : base(parameters) { }  
 public override void Execute()  
 {  
 ConsoleColor consoleColor = ConsoleColorFactory.GetColor(parameters);  
 if (consoleColor == ConsoleColor.DarkCyan)  
 {  
 Console.WriteLine(ConsoleColorFactory.GetColorsArrayString());  
 }  
 else  
 {  
 Console.BackgroundColor = consoleColor;  
 }  
 }  
 }  
  
 class ConcreteSettingsCommandTextColor : Command  
 {  
 public ConcreteSettingsCommandTextColor(string parameters) : base(parameters) { }  
 public override void Execute()  
 {  
 ConsoleColor consoleColor = ConsoleColorFactory.GetColor(parameters);  
 if (consoleColor == ConsoleColor.DarkCyan)  
 {  
 Console.WriteLine(ConsoleColorFactory.GetColorsArrayString());  
 }  
 else  
 {  
 Console.ForegroundColor = consoleColor;  
 }  
 }  
 }  
  
 class ConcreteSettingsCommandHelp : Command  
 {  
 public override void Execute()  
 {  
 SettingsLevelCommandsFactory.PrintCommands();  
 }  
 }  
  
 class ConcreteCommandReboot : Command  
 {  
 public override void Execute()  
 {  
 Sys.Power.Reboot();  
 }  
 }  
  
 abstract class Creator  
 {  
 public abstract string Command { get; }  
 public abstract string Description { get; }  
 public abstract Command CreateCommand(string parameters);  
 }  
  
 class ConcreteCreatorAbout : Creator  
 {  
 public override string Command  
 {  
 get { return "about"; }  
 }  
 public override string Description  
 {  
 get { return "Displays the version information about this operating system"; }  
 }  
 public override Command CreateCommand(string parameters)  
 {  
 return new ConcreteCommandAbout();  
 }  
 }  
  
 class ConcreteCreatorClearScreen : Creator  
 {  
 public override string Command  
 {  
 get { return "cls"; }  
 }  
 public override string Description  
 {  
 get { return "Clears the screen"; }  
 }  
 public override Command CreateCommand(string parameters)  
 {  
 return new ConcreteCommandClearScreen();  
 }  
 }  
  
 class ConcreteCreatorEcho : Creator  
 {  
 public override string Command  
 {  
 get { return "echo"; }  
 }  
 public override string Description  
 {  
 get { return "Returns the text you enter after it"; }  
 }  
 public override Command CreateCommand(string parameters)  
 {  
 return new ConcreteCommandEcho(parameters);  
 }  
 }  
  
 class ConcreteCreatorHelp : Creator  
 {  
 public override string Command  
 {  
 get { return "help"; }  
 }  
 public override string Description  
 {  
 get { return "Displays this text"; }  
 }  
 public override Command CreateCommand(string parameters)  
 {  
 return new ConcreteCommandHelp();  
 }  
 }  
  
 class ConcreteCreatorSettings : Creator  
 {  
 public override string Command  
 {  
 get { return "settings"; }  
 }  
 public override string Description  
 {  
 get { return "Returns a walkthrough of adjusting basic display and mouse settings"; }  
 }  
 public override Command CreateCommand(string parameters)  
 {  
 return new ConcreteCommandSettings(parameters);  
 }  
 }  
  
 class ConcreteCreatorReboot : Creator  
 {  
 public override string Command  
 {  
 get { return "reboot"; }  
 }  
 public override string Description  
 {  
 get { return "Reboots the machine"; }  
 }  
 public override Command CreateCommand(string parameters)  
 {  
 return new ConcreteCommandReboot();  
 }  
 }  
  
 class ConcreteSettingsCreatorBackgroundColor : Creator  
 {  
 public override string Command  
 {  
 get { return "background\_color"; }  
 }  
 public override string Description  
 {  
 get { return "adjusts background color using parameter string of color"; }  
 }  
 public override Command CreateCommand(string parameters)  
 {  
 return new ConcreteSettingsCommandBackgroundColor(parameters);  
 }  
 }  
  
 class ConcreteSettingsCreatorTextColor : Creator  
 {  
 public override string Command  
 {  
 get { return "text\_color"; }  
 }  
 public override string Description  
 {  
 get { return "adjusts text color using parameter string of color"; }  
 }  
 public override Command CreateCommand(string parameters)  
 {  
 return new ConcreteSettingsCommandTextColor(parameters);  
 }  
 }  
  
 class ConcreteSettingsCreatorHelp : Creator  
 {  
 public override string Command  
 {  
 get { return "help"; }  
 }  
 public override string Description  
 {  
 get { return "Displays this menu about settings options"; }  
 }  
 public override Command CreateCommand(string parameters)  
 {  
 return new ConcreteSettingsCommandHelp();  
 }  
 }  
  
 static class ConsoleColorFactory  
 {  
 private static string[] consoleColorsStrings = new string[] { "Black", "Blue", "Cyan", "DarkBlue", "DarkGray", "DarkGreen", "DarkMagenta", "DarkRed", "DarkYellow", "Gray", "Green", "Magenta", "Red", "White", "Yellow" };  
 private static ConsoleColor[] consoleColors = new ConsoleColor[] { ConsoleColor.Black, ConsoleColor.Blue, ConsoleColor.Cyan, ConsoleColor.DarkBlue, ConsoleColor.DarkGray, ConsoleColor.DarkGreen, ConsoleColor.DarkMagenta, ConsoleColor.DarkRed, ConsoleColor.DarkYellow, ConsoleColor.Gray, ConsoleColor.Green, ConsoleColor.Magenta, ConsoleColor.Red, ConsoleColor.White, ConsoleColor.Yellow };  
  
 public static ConsoleColor GetColor(string input)  
 {  
 for (int index = 0; index < consoleColorsStrings.Length; index++)  
 {  
 string consoleColorString = consoleColorsStrings[index];  
 if (input == consoleColorString)  
 {  
 return consoleColors[index];  
 }  
 }  
 return ConsoleColor.DarkCyan;  
 }  
 public static string GetColorsArrayString()  
 {  
 string extraColorString = "";  
 foreach (string consoleColorString in consoleColorsStrings)  
 {  
 extraColorString += consoleColorString;  
 extraColorString += ", ";  
 }  
 string colorString = "";  
 for (int index=0; index<extraColorString.Length-2; index++)  
 {  
 colorString += extraColorString[index];  
 }  
 return colorString;  
 }  
 }  
}