Creating a CPU Monitor app.

1. **Purpose**

In this project, we are creating a a CPU Monitor which will display the current CPU and RAM Usage. If the CPU usage is too high it will issue a warning with the help of voice assistance. After issuing a warning, it will ask the user to sign-out of the system.

2. **Collaboration**

This project is implemented by a group of three students currently studying in the third year. The workload for this project was equally divided amongst them.

3. **What to Use**

Here, the programming is done in C# and executed using Visual Studio. Visual Studio is a software that is used to build solutions for Console Application Projects. After building the solution, an executable file is generated that is added to the Windows Task Scheduler and run. If you want to run the program only once, then you can directly open the
.exe file and run the application.

4. **Interface**

This entire project has been coded in C#. The program files can be saved in any directory on your computer. These programs are executed with the help of visual Studio. In visual studio the Program has to be loaded/opened as a project. After the program is loaded it has to be built. Visual Studio provides an option of build. Once you build your Program it creates a .sln and .exe file

The CPU monitor is built using visual studio and the .exe file is scheduled in the task manager. Now, the CPU monitor tells you the amount of RAM and CPU being used. If the CPU utilization reaches 100% at any moment a warning in the form of a voice message will be issued to the user. The message will tell the user to save all his current work as the device isgoing to shut-down in the next few minutes.

5. Implementation.

The “using” keyword is used to reference namespaces. Namespace is basically a container that consists of classes, interfaces, etc.
When creating a C# project in Visual Studio, the above namespacesare generated by default.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using System.IO;
using System.Diagnostics;
```

You can also define your own namespace.

```
namespace ConsoleApp1
```

Implementation of the CPU Monitor App :-

```
using System.Diagnostics;
```

The Diagnostics class was used to get real-time hardware information such as CPU usage and RAM usage.

```
static PerformanceCounter cpuCount = new
PerformanceCounter("Processor Information", "% Processor
Time","_Total");

static PerformanceCounter memCount = new
PerformanceCounter("Memory", "% Committed Bytes in Use");
```

We send the necessary parameters into the Performance Counter constructor according to the necessary hardware information required.

```
using System.Speech.Synthesis;
```

The Speech.Synthesis class was used to enable voice assistance in the application.

```
static SpeechSynthesizer synth = new SpeechSynthesizer();

public static void aSpeak(String mesg)
{
   synth.Speak(mesg);
}

mesg = "CPU Monitor is monitoring your CPU and RAM usage now!!!!!";

aSpeak(mesg);
```

We created an instance of SpeechSynthesizer class called 'synth'

We then created a user defined method called aSpeak (meaning assistant speak)

A string is passed as a parameter to this method

The user defined method consists of the Speak method from the SpeechSynthesizer class to which we pass a string that was passed to the aSpeak method.

This string is then converted to speech by its Text to Speech Engine.

```
using System.Threading;
```

The System.Threading class was used to using multithreading and to place a delay of a specified interval.

```
Thread.Sleep(100);
```

This method i.e Sleep() is from the System.Threading class
As the CPU usage varies with time user must be updated of this change but as this process is too quick it is not possible to show infinitesimal changes to the user, i.e if the values change every milliseconds user cannot understand the values displayed as its refresh rate is too high
So here we introduce a delay of 500ms
In this case the user is shown the details after every 500ms

```
using System.Windows.Forms;
```

As we have created a GUI for the application the Windows.Forms class was used.

```
t = new System.Threading.Thread(monitor_system);
t.Start();
```

As this is a GUI application the fetching of data and displaying it on the Interface cannot be done simultaneously.
Even if it is done simultaneously it would cause an error of accuracy,

because there is a possibility that in the interval of these two processes, the details fetched might change.

To solve this issue we are running these two processes on different threads so that both processes run parallel.

```
cval = (int)cpuCount.NextValue();
cpu_text.Invoke(new MethodInvoker(setcpu));
```

As we are performing operations out of the scope of standard GUI operations like click event, keypress event we need to manually invoke the GUI elements to display data on it.

For this we use the Invoke method with respect to the element data to be modified.

```

Process cmd = new Process();
cmd.StartInfo.FileName = "cmd.exe";
cmd.StartInfo.RedirectStandardInput = true;
cmd.StartInfo.RedirectStandardOutput = true;
cmd.StartInfo.CreateNoWindow = true;
cmd.StartInfo.UseShellExecute = false;
cmd.Start();

cmd.StandardInput.WriteLine("shutdown /l");

cmd.StandardInput.Flush();
cmd.StandardInput.Close();
cmd.WaitForExit();

Console.WriteLine(cmd.StandardOutput.ReadToEnd());
```

In case of extremely high usage in order to keep the components safe we are killing all running processes and shutting down the computer.

To do this we are sending a command to a hidden command prompt window using the above piece of code.

The judgment to be taken when to send the shutdown signal is done by the code below :-

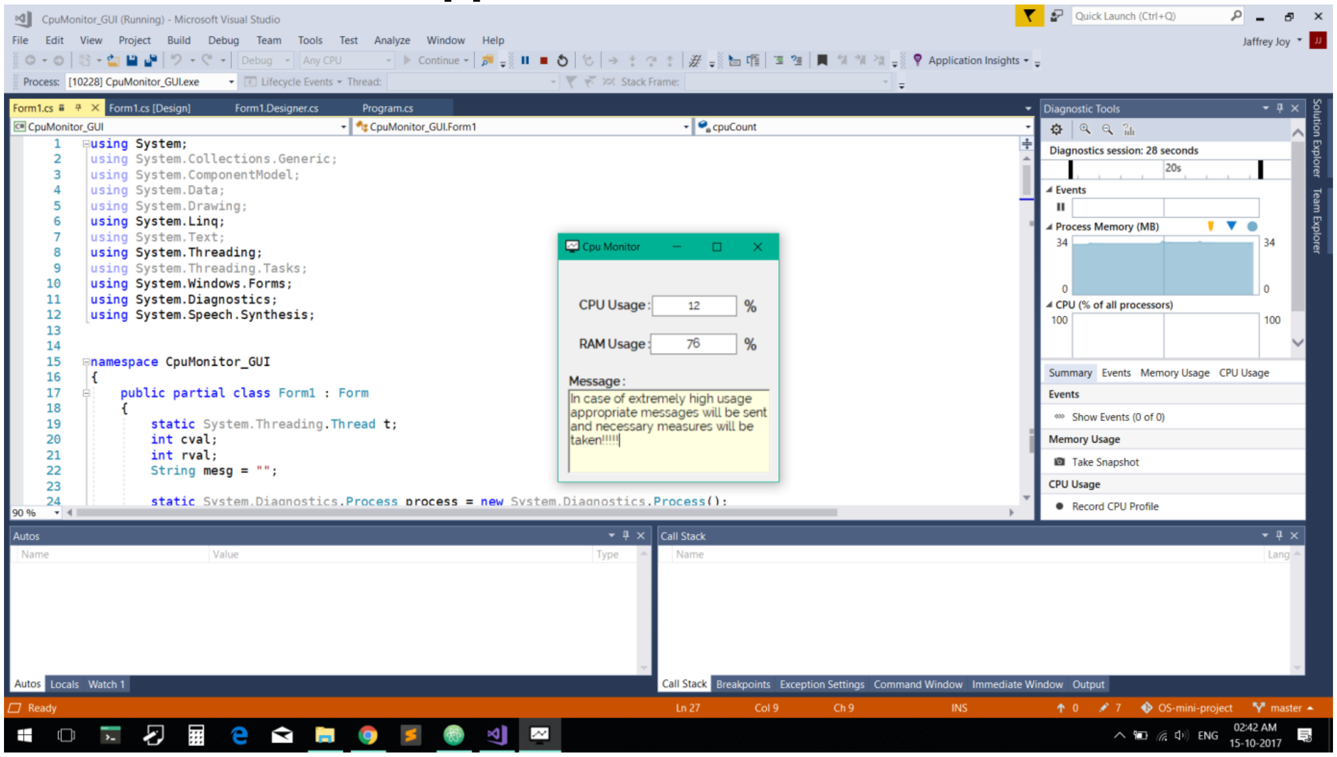
```
while (true)
{
    cval = (int)cpuCount.NextValue();
    cpu_text.Invoke(new MethodInvoker(setcpu));


    rval = (int)memCount.NextValue();
    ram_text.Invoke(new MethodInvoker(setram));


    if (cval > 10)
    {
        //mesg = "cval greater than 10";
        //aSpeak(mesg);
        cflag[i++] = 1;
        if (i != 0)
        {
            //mesg = "i not 0";
            //aSpeak(mesg);
            if (cflag[i - 1] == 0)
            {
                //mesg = "previous cflag 0";
                //aSpeak(mesg);
                stopWatch.Stop();
                ts = stopWatch.Elapsed;
                stopWatch.Reset();
                stopWatch.Start();
            }
        }
        else if (ix != 0)
        {
            if (cflag[9] == 0)
            {
                //mesg = "cflag 9 is 0";
                //aSpeak(mesg);
                stopWatch.Stop();
                ts = stopWatch.Elapsed;
                stopWatch.Reset();
                stopWatch.Start();
            }
        }
        if (i == 10)
            i = 0;
    }
    else
    {
        //mesg = "cval less than 10";
        //aSpeak(mesg);
        cflag[i++] = 0;
        stopWatch.Stop();
        ts = stopWatch.Elapsed;
        stopWatch.Reset();
        stopWatch.Start();
        if (i == 10)
            i = 0;
    }
    if (cflag.Sum() == 10)
    {
        ts = stopWatch.Elapsed;
        //mesg = "cval sum = 10, ts = "+ts.Seconds.ToString();
        //aSpeak(mesg);
        if (ts.Seconds > 10)
        {
            mesg = "ALERT!! Your CPU Usage is extremely High";
            mesg_text.Invoke(new MethodInvoker(setmesg));
            aSpeak(mesg);

            mesg = "System will start shutdown process in 10 seconds";
            mesg_text.Invoke(new MethodInvoker(setmesg));
```

```
            aSpeak(mesg);

            mesg = "Please save your work and Brace for Impact";
            mesg_text.Invoke(new MethodInvoker(setmesg));
            aSpeak(mesg);

            //startInfo.WindowStyle =
System.Diagnostics.ProcessWindowStyle.Hidden;
            //startInfo.FileName = "cmd.exe";
            //startInfo.Arguments = "shutdown /l /t 10";
            //process.StartInfo = startInfo;
            //process.Start();

            Process cmd = new Process();
            cmd.StartInfo.FileName = "cmd.exe";
            cmd.StartInfo.RedirectStandardInput = true;
            cmd.StartInfo.RedirectStandardOutput = true;
            cmd.StartInfo.CreateNoWindow = true;
            cmd.StartInfo.UseShellExecute = false;
            cmd.Start();

            cmd.StandardInput.WriteLine("shutdown /l");

            cmd.StandardInput.Flush();
            cmd.StandardInput.Close();
            cmd.WaitForExit();
            Console.WriteLine(cmd.StandardOutput.ReadToEnd());


            //stopWatch.Reset();
            //stopWatch.Start();
        }
    }
    ix++;
    Thread.Sleep(100);
}
```

For entire source-code :-

https://github.com/Jaffrey98/CpuMonitor

Screenshot of the application


CpuMonitor_GUI (Running) - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Process: [10228] CpuMonitor_GUI.exe
Form1.cs
Form1.cs [Design]
Form1.Designer.cs
Program.cs
CpuMonitor_GUI
CpuMonitor_GUI.Form1
cpuCount
Diagnostic Tools
Diagnostics session: 28 seconds
Events
Process Memory (MB)
CPU (% of all processors)
Summary Events Memory Usage CPU Usage
Show Events (0 of 0)
Take Snapshot
Record CPU Profile
Autos
Call Stack
Ready

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Diagnostics;
using System.Speech.Synthesis;


namespace CpuMonitor_GUI
{
    public partial class Form1 : Form
    {
        static System.Threading.Thread t;
        int cval;
        int rval;
        String mesg = "";

        static System.Diagnostics.Process process = new System.Diagnostics.Process();
```

Cpu Monitor

CPU Usage : 12 %

RAM Usage : 76 %

Message :

In case of extremely high usage appropriate messages will be sent and necessary measures will be taken!!!!!

6. Statement of Work

You can implement various processes in C# and run them using the task manager. C# is used with the .NET framework in windows. The above programs achieve the following:-

- The CPU monitor is implemented to give the user an accurate account of the usage of RAM and CPU. This shuts/closes (for ease of this project we’ve chosen to sign-out instead of shut down) unwanted processes and it may save your device from excess exhaustion.